# pygamelib Documentation

Release 1.3.0

**Arnaud Dupuis** 

# **CONTENTS (API REFERENCE):**

1	Forewords	3
2	Introduction	5
3	Tutorials 3.1 actuators	<b>7</b> 7
	3.2 assets	
	3.3 base	240 261
	3.5 constants	
	3.6 engine	
	3.8 Credits	
	3.9 Release notes	751
4	Indices and tables	757
Py	thon Module Index	759
In	dex	761



### **CHAPTER**

# **ONE**

## **FOREWORDS**

Historically, this library was (and still is) used as a base to teach coding to kids from 6 to 15. It aims at giving an environment to new and learning developers (including kids) that let them focus on the algorithm instead of the lousy display or precise management.

It started as a very simple library with very little capabilities, but over time it became something more. To the point that it is now possible to make very decent terminal games with it.

So this is **obviously** still extremely simple compared to other game framework and it still does not have the pretention of being anything serious for real game developers. However, it can now be used by aspiring game developers for an introduction to 2D games development.

### **CHAPTER**

# **TWO**

# **INTRODUCTION**

First of all, his module is exclusively compatible with python 3.6+.

The core concept is that writting a game mostly involve the *Game* object, the *Board* object and the derivatives of *board\_items*.

More advanced game will use the ui module to create terminal user interfaces (or TUI) and the GFX core module to improve the graphics with Sprite and Color.

Here is an example of what the current version allow to build:

And a quick peak at the new features in the most recent version:

**CHAPTER** 

**THREE** 

## **TUTORIALS**

Most tutorials to teach you how to use the library to build games are (or will be) on the wiki.

Tutorials that teach you how to expand the library are (or will be) centralized here.

The complete API documentation is referenced bellow.

### 3.1 actuators

This module contains the base classes for simple and advanced actuators. These classes are the base contract for actuators. If you wish to create your own one, you need to inherit from one of these base class.

### 3.1.1 Actuator

class pygamelib.actuators.Actuator(parent: BoardItem | None)

Bases: PglBaseObject

Actuator is the base class for all Actuators. It is mainly a contract class with some utility methods.

By default, all actuators are considered movement actuators. So the base class only require next\_move() to be implemented.

#### **Parameters**

parent - the item parent.

\_\_init\_\_(parent: BoardItem | None)

The constructor take only one (positional) parameter: the parent object.

**Important:** The default state of ALL actuators is RUNNING. If you want your actuator to be in a different state (PAUSED for example), you have to do it yourself.

### **Methods**

init(parent)	The constructor take only one (positional) parameter: the parent object.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load serialized data, create and returns a new actuator out of these data.
next_move()	That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the actuator state to PAUSED.
serialize()	Serializes the actuator and returns it as a dict.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

### **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

### Return type

bool

### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

#### **Return type**

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

**load**(data: dict)  $\rightarrow Actuator$ 

Load serialized data, create and returns a new actuator out of these data.

That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.

#### Raises

NotImplementedError

```
next_move() \rightarrow Vector2D \mid int
```

That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.

#### Raises

NotImplementedError

#### **Returns**

```
Vector2D | int
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

### property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

value (int) – the screen column

### Return type

in

### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (*int*) – the screen row

### Return type

int

### $\textbf{serialize()} \rightarrow dict$

Serializes the actuator and returns it as a dict.

That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.

#### Raises

NotImplementedError

### start()

Set the actuator state to RUNNING.

If the actuator state is not RUNNING, actuators' next\_move() function (and all derivatives) should not return anything.

Example:

```
mygame.start()
```

### stop()

Set the actuator state to STOPPED.

Example:

```
mygame.stop()
```

### $store\_screen\_position(\mathit{row: int, column: int}) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row(int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

### 3.1.2 Behavioral

class pygamelib.actuators.Behavioral(parent: BoardItem | None)

Bases: Actuator

The behavioral actuator is inheriting from Actuator and is adding a next\_action() method. The actual actions are left to the actuator that implements Behavioral.

#### **Parameters**

parent - the item parent.

\_\_init\_\_(parent: BoardItem | None)

The constructor simply construct an Actuator. It takes on positional parameter: the parent object.

#### **Methods**

init(parent)	The constructor simply construct an Actuator.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load serialized data, create and returns a new actua-
	tor out of these data.
<pre>next_action()</pre>	That method needs to be implemented by all behav-
	ioral actuators or a NotImplementedError exception
	will be raised.
next_move()	That method needs to be implemented by all actuators
	or a NotImplementedError exception will be raised.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the actuator state to PAUSED.
serialize()	Serializes the actuator and returns it as a dict.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.
store_screen_position(row, column)	Store the screen position of the object.

### **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

### **Return type**

bool

#### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

#### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

### **load**(data: dict) $\rightarrow Actuator$

Load serialized data, create and returns a new actuator out of these data.

That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.

#### **Raises**

NotImplementedError

### next\_action()

That method needs to be implemented by all behavioral actuators or a NotImplementedError exception will be raised.

#### Raises

NotImplementedError

```
next_move() \rightarrow Vector2D \mid int
```

That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.

#### Raises

NotImplementedError

#### Returns

```
Vector2D | int
```

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*PglBaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

### property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

 $\textbf{value}\ (int)-the\ screen\ column$ 

### Return type

int

### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

#### **Return type**

int

```
serialize() \rightarrow dict
```

Serializes the actuator and returns it as a dict.

That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.

#### Raises

NotImplementedError

#### start()

Set the actuator state to RUNNING.

If the actuator state is not RUNNING, actuators' next\_move() function (and all derivatives) should not return anything.

Example:

```
mygame.start()
```

#### stop()

Set the actuator state to STOPPED.

Example:

```
mygame.stop()
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

### 3.1.3 PathActuator

 $\textbf{class} \ \ \textbf{pygamelib.actuators.} \ \textbf{PathActuator}(\textit{path: List[int]} \mid \textit{None} = \textit{None}, \textit{parent: BoardItem} \mid \textit{None} = \textit{None})$ 

Bases: Actuator

The path actuator is a subclass of *Actuator*. The move inside the function next\_move depends on path and index. If the state is not running it returns None otherwise it increments the index & then, further compares the index with length of the path. If they both are same then, index is set to value zero and the move is returned back.

#### **Parameters**

- **path** (*list*) A list of paths.
- parent (pygamelib.board\_items.BoardItem) The parent object to actuate.

\_\_init\_\_(path: List[int] | None = None, parent: BoardItem | None = None)

The constructor take only one (positional) parameter: the parent object.

**Important:** The default state of ALL actuators is RUNNING. If you want your actuator to be in a different state (PAUSED for example), you have to do it yourself.

#### **Methods**

init([path, parent])	The constructor take only one (positional) parameter: the parent object.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new PathActuator out of it.
next_move()	Return the movement based on current index
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the actuator state to PAUSED.
serialize()	Return a dictionary with all the attributes of this object.
<pre>set_path(path)</pre>	Defines a new path
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

### **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

### Return type

bool

### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

### classmethod load(data: dict) $\rightarrow PathActuator$

Load data and create a new PathActuator out of it.

#### **Parameters**

**data** (dict) – Data to create a new actuator (usually generated by serialize())

### Returns

A new actuator.

### Return type

PathActuator

#### Example:

```
path_actuator = PathActuator.load(actuator_data)
```

### $next\_move() \rightarrow int$

Return the movement based on current index

The movement is selected from path if state is RUNNING, otherwise it returns NO\_DIR from the *constants* module. When state is RUNNING, the movement is selected before incrementing the index by 1. When the index equal the length of path, the index should return back to 0.

#### Returns

The next movement

### Return type

```
int | pygamelib.constants.Direction.NO_DIR
```

Example:

```
path_actuator.next_move()
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

#### pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

### property screen\_column: int

A property to get/set the screen column.

### **Parameters**

**value** (int) – the screen column

### Return type

int

#### property screen\_row: int

A property to get/set the screen row.

### **Parameters**

**value** (int) – the screen row

#### Return type

int

### **serialize()** $\rightarrow$ dict

Return a dictionary with all the attributes of this object.

### Returns

A dictionary with all the attributes of this object.

### Return type

dict

```
set_path(path: List[int])
```

Defines a new path

This will also reset the index back to 0.

#### **Parameters**

path (list) – A list of movements.

Example:

#### start()

Set the actuator state to RUNNING.

If the actuator state is not RUNNING, actuators' next\_move() function (and all derivatives) should not return anything.

Example:

```
mygame.start()
```

### stop()

Set the actuator state to STOPPED.

Example:

mygame.stop()

 $\textbf{store\_screen\_position}(\textit{row: int, column: int}) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- $\mathbf{row}(int)$  The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

### 3.1.4 PatrolActuator

Bases: PathActuator

The patrol actuator is a subclass of *PathActuator*. The move inside the function next\_move depends on path and index and the mode. Once it reaches the end of the move list it will start cycling back to the beginning of the list. Once it reaches the beginning it will start moving forwards If the state is not running it returns None otherwise it increments the index & then, further compares the index with length of the path. If they both are same then, index is set to value zero and the move is returned back.

### **Parameters**

**path** (*list*) – A list of directions.

```
__init__(path: List[int] | None = None, parent: BoardItem | None = None)
```

The constructor take only one (positional) parameter: the parent object.

**Important:** The default state of ALL actuators is RUNNING. If you want your actuator to be in a different state (PAUSED for example), you have to do it yourself.

### **Methods**

init([path, parent])	The constructor take only one (positional) parameter: the parent object.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new PatrolActuator out of it.
next_move()	Return the movement based on current index
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the actuator state to PAUSED.
serialize()	Return a dictionary with all the attributes of this ob-
	ject.
set_path(path)	Defines a new path
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

### **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

### Return type

bool

### Example:

(continued from previous page)

```
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### **Returns**

True or False depending on the success of the operation.

### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

### classmethod load(data: dict) $\rightarrow PatrolActuator$

Load data and create a new PatrolActuator out of it.

#### **Parameters**

**data** (dict) – Data to create a new actuator (usually generated by serialize())

#### **Returns**

A new actuator.

#### **Return type**

PatrolActuator

### Example:

```
patrol_actuator = PatrolActuator.load(actuator_data)
```

#### $next\_move() \rightarrow int$

Return the movement based on current index

The movement is selected from path if state is RUNNING, otherwise it returns NO\_DIR from the *constants* module. When state is RUNNING, the movement is selected before incrementing the index by

1. When the index equals the length of path, the index should return back to 0 and the path list should be reversed before the next call.

#### Returns

The next movement

### **Return type**

```
int | pygamelib.constants.Direction.NO_DIR
```

Example:

```
patrol_actuator.next_move()
```

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

### property screen\_column: int

A property to get/set the screen column.

```
Parameters
```

**value** (int) – the screen column

# Return type

int

### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

### Return type

int

### $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

#### Returns

A dictionary with all the attributes of this object.

### **Return type**

dict

### set\_path(path: List[int])

Defines a new path

This will also reset the index back to 0.

#### **Parameters**

path (list) – A list of movements.

Example:

#### start()

Set the actuator state to RUNNING.

If the actuator state is not RUNNING, actuators' next\_move() function (and all derivatives) should not return anything.

Example:

```
mygame.start()
```

#### stop()

Set the actuator state to STOPPED.

Example:

mygame.stop()

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- $\mathbf{row}(int)$  The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

### 3.1.5 PathFinder

Bases: Behavioral

**Important:** This module assume a one step movement. If you need more than one step, you will need to sub-class this module and re-implement next\_waypoint().

This actuator is a bit different than the simple actuators (SimpleActuators) as it requires the knowledge of both the game object and the actuated object.

The constructor takes the following parameters:

#### **Parameters**

- game (pygamelib.engine.Game) A reference to the instantiated game engine.
- actuated\_object (pygamelib.board\_items.BoardItem) The object to actuate. Deprecated in favor of parent. Only kept for backward compatibility.
- parent (pygamelib.board\_items.BoardItem) The parent object to actuate.
- **circle\_waypoints** (*bool*) If True the next\_waypoint() method is going to circle between the waypoints (when the last is visited, go back to the first)
- algorithm (constant) ALGO\_BFS BFS, ALGO\_ASTAR AStar

\_\_init\_\_(game: engine.Game | None = None, actuated\_object: board\_items.BoardItem | None = None, circle\_waypoints=True, parent: board\_items.BoardItem | None = None, algorithm=Algorithm.BFS)

The constructor simply construct an Actuator. It takes on positional parameter: the parent object.

### **Methods**

init([game, actuated_object,])	The constructor simply construct an Actuator.
<pre>add_waypoint(row, column)</pre>	Add a waypoint to the list of waypoints.
attach(observer)	Attach an observer to this instance.
<pre>clear_waypoints()</pre>	Empty the waypoints stack.
<pre>current_path()</pre>	This method simply return a copy of the current path of the actuator.
<pre>current_waypoint()</pre>	Return the currently active waypoint.
<pre>detach(observer)</pre>	Detach an observer from this instance.
<pre>find_path()</pre>	Find a path to the destination.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new PathFinder out of it.
<pre>next_action()</pre>	That method needs to be implemented by all behavioral actuators or a NotImplementedError exception will be raised.
<pre>next_move()</pre>	This method return the next move calculated by this actuator.
<pre>next_waypoint()</pre>	Return the next active waypoint.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the actuator state to PAUSED.
<pre>remove_waypoint(row, column)</pre>	Remove a waypoint from the stack.
serialize()	Return a dictionary with all the attributes of this object.
<pre>set_destination([row, column])</pre>	Set the targeted destination.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

### **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

### add\_waypoint(row: int, column: int)

Add a waypoint to the list of waypoints.

Waypoints are used one after the other on a FIFO basis (First In, First Out).

If not destination (i.e destination == (None, None)) have been set yet, that method sets it.

### **Parameters**

- row (int) The "row" part of the waypoint's coordinate.
- **column** The "column" part of the waypoint's coordinate.

### Raises

**PglInvalidTypeException** – If any of the parameters is not an int.

Example:

```
pf = PathFinder(game=mygame, actuated_object=npc1)
pf.add_waypoint(3,5)
pf.add_waypoint(12,15)
```

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

#### Returns

True or False depending on the success of the operation.

### Return type

bool

### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### clear\_waypoints()

Empty the waypoints stack.

Example:

```
pf.clear_waypoints()
```

### **current\_path**() → List[Tuple[int, int]]

This method simply return a copy of the current path of the actuator.

The current path is to be understood as: the list of positions still remaining. All positions that have already been gone through are removed from the stack.

**Important:** A copy of the path is returned for every call to that function so be wary of the performances impact.

Example:

```
current_waypoint() → Tuple[int | None, int | None]
```

Return the currently active waypoint.

If no waypoint have been added, this function return None.

#### Returns

Either a None tuple or the current waypoint.

### Return type

A None tuple or a tuple of integer.

Example:

```
(row,column) = pf.current_waypoint()
pf.set_destination(row,column)
```

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

### Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

```
find_path() \rightarrow List[Tuple[int, int]]
```

Find a path to the destination.

Destination (PathFinder.destination) has to be set beforehand.

Example:

**Warning:** PathFinder.destination is a tuple! Please use PathFinder.set\_destination(x,y) to avoid problems.

Path Finding Algorithm Description:

Breadth First Search: This method implements a Breadth First Search algorithm (Wikipedia: BFS) to find the shortest path to destination.

A\* Search: This method implements a A\* Search algorithm (Wikipedia: A\*) to find the shortest path to destination.

#### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

### classmethod load(data: dict) $\rightarrow PathFinder$

Load data and create a new PathFinder out of it.

#### **Parameters**

**data** (dict) – Data to create a new actuator (usually generated by serialize())

#### Returns

A new actuator.

#### **Return type**

**PathFinder** 

### Example:

```
path_finder = PathFinder.load(actuator_data)
```

### next\_action()

That method needs to be implemented by all behavioral actuators or a NotImplementedError exception will be raised.

#### Raises

NotImplementedError

### $next\_move() \rightarrow int$

This method return the next move calculated by this actuator.

In the case of this PathFinder actuator, next move does the following:

- If the destination is not set return NO\_DIR (see *constants*) If the destination is set, but the path is empty and actuated object's position is different from destination: call *find\_path()*
- Look at the current waypoint, if the actuated object is not at that position return a direction from the *constants* module. The direction is calculated from the difference between actuated object's position and waypoint's position.
- If the actuated object is at the waypoint position, then call next\_waypoint(), set the destination and return a direction. In this case, also call <code>find\_path()</code>.
- In any case, if there is no more waypoints in the path this method returns NO\_DIR (see *constants*)

Example:

#### **next\_waypoint()** → Tuple[int | None, int | None]

Return the next active waypoint.

If no waypoint have been added, this function return None. If there is no more waypoint in the stack:

- if PathFinder.circle\_waypoints is True this function reset the waypoints stack and return the first one.
- else, return None.

#### Returns

Either a None tuple or the next waypoint.

#### Return type

A None tuple or a tuple of integer.

#### Example:

```
pf.circle_waypoints = True
(row,column) = pf.next_waypoint()
pf.set_destination(row,column)
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

### remove\_waypoint(row: int, column: int)

Remove a waypoint from the stack.

This method removes the first occurrence of a waypoint in the stack.

If the waypoint cannot be found, it raises a ValueError exception. If the row and column parameters are not int, an PglInvalidTypeException is raised.

#### **Parameters**

- row (int) The "row" part of the waypoint's coordinate.
- **column** The "column" part of the waypoint's coordinate.

### Raises

- PglInvalidTypeException If any of the parameters is not an int.
- **ValueError** If the waypoint is not found in the stack.

### Example:

```
path_finder.remove_waypoint(2,5)
```

### property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

value (int) – the screen column

### Return type

int

### property screen\_row: int

A property to get/set the screen row.

### **Parameters**

**value** (int) – the screen row

### Return type

int

### $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

### Returns

A dictionary with all the attributes of this object.

### **Return type**

dict

### $set_destination(row: int = 0, column: int = 0)$

Set the targeted destination.

#### **Parameters**

- row (int) "row" coordinate on the board grid
- column (int) "column" coordinate on the board grid

#### Raises

**PglInvalidTypeException** – if row or column are not int.

Example:

```
mykillernpc.actuator.set_destination(
    mygame.player.pos[0], mygame.player.pos[1]
)
```

#### start()

Set the actuator state to RUNNING.

If the actuator state is not RUNNING, actuators' next\_move() function (and all derivatives) should not return anything.

Example:

```
mygame.start()
```

#### stop()

Set the actuator state to STOPPED.

Example:

```
mygame.stop()
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

### 3.1.6 RandomActuator

class pygamelib.actuators.RandomActuator( $moveset: List[Vector2D \mid int] \mid None = None, parent: BoardItem \mid None = None)$ 

Bases: Actuator

A class that implements a random choice of movement.

The random actuator is a subclass of *Actuator*. It is simply implementing a random choice in a predefined move set.

#### **Parameters**

30

- moveset (list) A list of movements.
- parent (pygamelib.board\_items.BoardItem) The parent object to actuate.

```
__init__(moveset: List[Vector2D | int] | None = None, parent: BoardItem | None = None)
```

The constructor take only one (positional) parameter: the parent object.

**Important:** The default state of ALL actuators is RUNNING. If you want your actuator to be in a different state (PAUSED for example), you have to do it yourself.

### **Methods**

init([moveset, parent])	The constructor take only one (positional) parameter: the parent object.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new RandomActuator out of it.
next_move()	Return a randomly selected movement
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the actuator state to PAUSED.
serialize()	Return a dictionary with all the attributes of this object.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

### **Attributes**

moveset	Return the moveset.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

 ${\bf observer} \ (PglBase Object) - {\bf An \ observer \ to \ attach \ to \ this \ object}.$ 

### Returns

True or False depending on the success of the operation.

### Return type

bool

Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

#### classmethod load(data: dict) $\rightarrow RandomActuator$

Load data and create a new RandomActuator out of it.

#### Parameters

data (dict) - Data to create a new actuator (usually generated by serialize())

#### Returns

A new actuator.

### Return type

RandomActuator

### Example:

```
npc2.actuator = actuators.RandomActuator.load( npc1.actuator.serialize() )
```

```
property moveset: List[Vector2D | int]
```

Return the moveset.

#### Returns

The moveset.

## Return type

list

```
next_move() \rightarrow Vector2D \mid int
```

Return a randomly selected movement

The movement is randomly selected from moveset if state is RUNNING, otherwise it returns NO\_DIR from the *constants* module.

#### Returns

The next movement *Vector2D* | int

Example:

```
random_actuator.next_move()
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

## property screen\_column: int

A property to get/set the screen column.

### **Parameters**

value (int) - the screen column

# Return type

int

### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

value (int) – the screen row

3.1. actuators 33

### **Return type**

int

#### **serialize()** $\rightarrow$ dict

Return a dictionary with all the attributes of this object.

#### Returns

A dictionary with all the attributes of this object.

### Return type

dict

#### start()

Set the actuator state to RUNNING.

If the actuator state is not RUNNING, actuators' next\_move() function (and all derivatives) should not return anything.

Example:

```
mygame.start()
```

#### stop()

Set the actuator state to STOPPED.

Example:

```
mygame.stop()
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

## **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

### 3.1.7 Unidirectional Actuator

class pygamelib.actuators. Unidirectional Actuator (direction: int = Direction.RIGHT, parent: Board Item | None = None)

Bases: Actuator

A class that implements a single movement.

The unidirectional actuator is a subclass of *Actuator*. It is simply implementing a mono directional movement. It is primarily target at projectiles.

### **Parameters**

- **direction** (*int*) A single direction from the Constants module.
- parent (pygamelib.board\_items.BoardItem) The parent object to actuate.

```
__init__(direction: int = Direction.RIGHT, parent: BoardItem | None = None)
```

The constructor take only one (positional) parameter: the parent object.

Important: The default state of ALL actuators is RUNNING. If you want your actuator to be in a different state (PAUSED for example), you have to do it yourself.

### **Methods**

init([direction, parent])	The constructor take only one (positional) parameter: the parent object.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new Unidirectional Actuator out of it.
next_move()	Return the direction.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the actuator state to PAUSED.
serialize()	Return a dictionary with all the attributes of this object.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

## **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
```

3.1. actuators 35

(continues on next page)

(continued from previous page)

```
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### **Returns**

True or False depending on the success of the operation.

## Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## classmethod load(data: dict) $\rightarrow UnidirectionalActuator$

Load data and create a new Unidirectional Actuator out of it.

#### **Parameters**

**data** (dict) – Data to create a new actuator (usually generated by serialize())

### **Returns**

A new actuator.

### **Return type**

*UnidirectionalActuator* 

### Example:

```
unidir_actuator = UnidirectionalActuator.load(actuator_data)
```

#### $next\_move() \rightarrow int$

Return the direction.

The movement is always direction if state is RUNNING, otherwise it returns NO\_DIR from the *constants* module.

#### Returns

The next movement

## Return type

int | pygamelib.Direction.NO\_DIR

Example:

```
unidirectional_actuator.next_move()
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

## property screen\_column: int

A property to get/set the screen column.

### **Parameters**

**value** (int) – the screen column

#### Return type

int

## property screen\_row: int

A property to get/set the screen row.

### **Parameters**

**value** (int) – the screen row

### **Return type**

int

## $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

#### Returns

A dictionary with all the attributes of this object.

3.1. actuators 37

## Return type

dict

### start()

Set the actuator state to RUNNING.

If the actuator state is not RUNNING, actuators' next\_move() function (and all derivatives) should not return anything.

Example:

```
mygame.start()
```

#### stop()

Set the actuator state to STOPPED.

Example:

```
mygame.stop()
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## 3.2 assets

The assets sub-module holds all the classes that are adding features without being core features. The graphics module is a good example of that: it is cool to have and provides a nice default set of assets to build games. But the library can work without it.

# 3.2.1 graphics

**Important:** The Graphics module was introduced in version 1.1.0.

The Graphics module hold many variables that aims at simplifying the use of unicode characters in the game development process.

This module also import colorama. All styling features are accessible through:

- Graphics.Fore for Foreground colors.
- Graphics.Back for Background colors.
- Graphics.Style for styling options.

For convenience, the different entities are scattered in grouping classes:

- All emojis are in the Models class.
- The UI/box drawings are grouped into the BoxDrawings class.
- The block glyphs are in the Blocks class.
- The geometric shapes are in the GeometricShapes class.

This modules defines a couple of colored squares and rectangles that should displays correctly in all terminals.

These are kept for legacy purpose (I personally have a lot of kids that are still using it), but for anyone starting fresh, it is better to use the <color>\_rect() and <color>\_square() static methods of the <code>Sprixel</code> class. Particularly if you are going to use them as background for your Board.

### Colored rectangles:

- WHITE\_RECT
- BLUE\_RECT
- RED RECT
- MAGENTA\_RECT
- GREEN\_RECT
- YELLOW\_RECT
- BLACK RECT
- CYAN RECT

## Then colored squares:

- WHITE\_SQUARE
- MAGENTA\_SQUARE
- GREEN\_SQUARE
- RED\_SQUARE
- BLUE\_SQUARE
- YELLOW\_SQUARE
- BLACK\_SQUARE
- CYAN\_SQUARE

And finally an example of composition of rectangles to make different colored squares:

- RED\_BLUE\_SQUARE = RED\_RECT+BLUE\_RECT
- YELLOW\_CYAN\_SQUARE = YELLOW\_RECT+CYAN\_RECT

The Graphics module contains the following classes:

### **Blocks**

### class pygamelib.assets.graphics.Blocks

Bases: object

Block elements (unicode)

Here is the list of supported glyphs:

- UPPER\_HALF\_BLOCK =
- LOWER\_ONE\_EIGHTH\_BLOCK =
- LOWER\_ONE\_QUARTER\_BLOCK =
- LOWER\_THREE\_EIGHTHS\_BLOCK =
- LOWER\_HALF\_BLOCK =
- LOWER\_FIVE\_EIGHTHS\_BLOCK =
- LOWER\_THREE\_QUARTERS\_BLOCK =
- LOWER\_SEVEN\_EIGHTHS\_BLOCK =
- FULL\_BLOCK =
- LEFT\_SEVEN\_EIGHTHS\_BLOCK =
- LEFT\_THREE\_QUARTERS\_BLOCK =
- LEFT\_FIVE\_EIGHTHS\_BLOCK =
- LEFT\_HALF\_BLOCK =
- LEFT\_THREE\_EIGHTHS\_BLOCK =
- LEFT\_ONE\_QUARTER\_BLOCK =
- LEFT\_ONE\_EIGHTH\_BLOCK =
- RIGHT\_HALF\_BLOCK =
- LIGHT\_SHADE =
- MEDIUM\_SHADE =
- DARK\_SHADE =
- UPPER\_ONE\_EIGHTH\_BLOCK =
- RIGHT\_ONE\_EIGHTH\_BLOCK =
- QUADRANT\_LOWER\_LEFT =
- QUADRANT\_LOWER\_RIGHT =
- QUADRANT\_UPPER\_LEFT =
- QUADRANT\_UPPER\_LEFT\_AND\_LOWER\_LEFT\_AND\_LOWER\_RIGHT =
- QUADRANT\_UPPER\_LEFT\_AND\_LOWER\_RIGHT =
- QUADRANT\_UPPER\_LEFT\_AND\_UPPER\_RIGHT\_AND\_LOWER\_LEFT =
- QUADRANT\_UPPER\_LEFT\_AND\_UPPER\_RIGHT\_AND\_LOWER\_RIGHT =
- QUADRANT\_UPPER\_RIGHT =
- QUADRANT\_UPPER\_RIGHT\_AND\_LOWER\_LEFT =

• QUADRANT\_UPPER\_RIGHT\_AND\_LOWER\_LEFT\_AND\_LOWER\_RIGHT = \_\_init\_\_()

### **Methods**

\_\_init\_\_()

#### **Attributes**

DARK\_SHADE FULL\_BLOCK LEFT\_FIVE\_EIGHTHS\_BLOCK LEFT\_HALF\_BLOCK LEFT\_ONE\_EIGHTH\_BLOCK LEFT\_ONE\_QUARTER\_BLOCK LEFT\_SEVEN\_EIGHTHS\_BLOCK LEFT\_THREE\_EIGHTHS\_BLOCK LEFT\_THREE\_QUARTERS\_BLOCK LIGHT\_SHADE LOWER\_FIVE\_EIGHTHS\_BLOCK LOWER\_HALF\_BLOCK LOWER\_ONE\_EIGHTH\_BLOCK LOWER\_ONE\_QUARTER\_BLOCK LOWER\_SEVEN\_EIGHTHS\_BLOCK LOWER\_THREE\_EIGHTHS\_BLOCK LOWER\_THREE\_QUARTERS\_BLOCK MEDIUM\_SHADE QUADRANT\_LOWER\_LEFT

continues on next page

Table 1 – continued from previous page

```
QUADRANT_LOWER_RIGHT
QUADRANT_UPPER_LEFT
QUADRANT_UPPER_LEFT_AND_LOWER_LEFT_AND_LO
QUADRANT_UPPER_LEFT_AND_LOWER_RIGHT
QUADRANT_UPPER_LEFT_AND_UPPER_RIGHT_AND_L
QUADRANT_UPPER_LEFT_AND_UPPER_RIGHT_AND_L
QUADRANT_UPPER_RIGHT
QUADRANT_UPPER_RIGHT_AND_LOWER_LEFT
QUADRANT_UPPER_RIGHT_AND_LOWER_LEFT_AND_L
RIGHT_HALF_BLOCK
RIGHT_ONE_EIGHTH_BLOCK
UPPER_HALF_BLOCK
UPPER_ONE_EIGHTH_BLOCK
```

```
DARK_SHADE = ''
FULL_BLOCK = ''
LEFT_FIVE_EIGHTHS_BLOCK = ''
LEFT_HALF_BLOCK = ''
LEFT_ONE_EIGHTH_BLOCK = ''
LEFT_ONE_QUARTER_BLOCK = ''
LEFT_SEVEN_EIGHTHS_BLOCK = ''
LEFT_THREE_EIGHTHS_BLOCK = ''
LEFT_THREE_QUARTERS_BLOCK = ''
LIGHT_SHADE = ''
LOWER_FIVE_EIGHTHS_BLOCK = ''
LOWER_HALF_BLOCK = ''
LOWER_ONE_EIGHTH_BLOCK = ''
LOWER_ONE_QUARTER_BLOCK = ''
```

```
LOWER_SEVEN_EIGHTHS_BLOCK = ''
    LOWER_THREE_EIGHTHS_BLOCK = ''
    LOWER_THREE_QUARTERS_BLOCK = ''
    MEDIUM_SHADE = ''
    QUADRANT_LOWER_LEFT = ''
    QUADRANT_LOWER_RIGHT = ''
    QUADRANT_UPPER_LEFT = ''
    QUADRANT_UPPER_LEFT_AND_LOWER_LEFT_AND_LOWER_RIGHT = ''
    QUADRANT_UPPER_LEFT_AND_LOWER_RIGHT = ''
    QUADRANT_UPPER_LEFT_AND_UPPER_RIGHT_AND_LOWER_LEFT = ''
    QUADRANT_UPPER_LEFT_AND_UPPER_RIGHT_AND_LOWER_RIGHT = ''
    QUADRANT_UPPER_RIGHT = ''
    QUADRANT_UPPER_RIGHT_AND_LOWER_LEFT = ''
    QUADRANT_UPPER_RIGHT_AND_LOWER_LEFT_AND_LOWER_RIGHT = ''
    RIGHT_HALF_BLOCK = ''
    RIGHT_ONE_EIGHTH_BLOCK = ''
    UPPER_HALF_BLOCK = ''
    UPPER_ONE_EIGHTH_BLOCK = ''
BoxDrawings
class pygamelib.assets.graphics.BoxDrawings
    Bases: object
    Box drawing elements (unicode)
    Here is the list of supported glyphs:
       • LIGHT HORIZONTAL = -
       • HEAVY HORIZONTAL =
       • LIGHT_VERTICAL =
       • HEAVY_VERTICAL =
```

LIGHT\_TRIPLE\_DASH\_HORIZONTAL =
 HEAVY\_TRIPLE\_DASH\_HORIZONTAL =
 LIGHT\_TRIPLE\_DASH\_VERTICAL =
 HEAVY\_TRIPLE\_DASH\_VERTICAL =

• LIGHT\_QUADRUPLE\_DASH\_HORIZONTAL =

- HEAVY\_QUADRUPLE\_DASH\_HORIZONTAL =
- LIGHT\_QUADRUPLE\_DASH\_VERTICAL =
- HEAVY\_QUADRUPLE\_DASH\_VERTICAL =
- LIGHT\_DOWN\_AND\_RIGHT =
- DOWN LIGHT AND RIGHT HEAVY =
- DOWN HEAVY AND RIGHT LIGHT =
- HEAVY\_DOWN\_AND\_RIGHT =
- LIGHT\_DOWN\_AND\_LEFT =
- DOWN\_LIGHT\_AND\_LEFT\_HEAVY =
- DOWN\_HEAVY\_AND\_LEFT\_LIGHT =
- HEAVY\_DOWN\_AND\_LEFT =
- LIGHT\_UP\_AND\_RIGHT = L
- UP\_LIGHT\_AND\_RIGHT\_HEAVY =
- UP\_HEAVY\_AND\_RIGHT\_LIGHT =
- HEAVY\_UP\_AND\_RIGHT =
- LIGHT\_UP\_AND\_LEFT =
- UP\_LIGHT\_AND\_LEFT\_HEAVY =
- UP\_HEAVY\_AND\_LEFT\_LIGHT =
- HEAVY\_UP\_AND\_LEFT =
- LIGHT\_VERTICAL\_AND\_RIGHT = |-
- VERTICAL\_LIGHT\_AND\_RIGHT\_HEAVY =
- UP\_HEAVY\_AND\_RIGHT\_DOWN\_LIGHT =
- DOWN\_HEAVY\_AND\_RIGHT\_UP\_LIGHT =
- VERTICAL\_HEAVY\_AND\_RIGHT\_LIGHT =
- DOWN\_LIGHT\_AND\_RIGHT\_UP\_HEAVY =
- UP\_LIGHT\_AND\_RIGHT\_DOWN\_HEAVY =
- HEAVY\_VERTICAL\_AND\_RIGHT =
- LIGHT\_VERTICAL\_AND\_LEFT =
- VERTICAL\_LIGHT\_AND\_LEFT\_HEAVY =
- UP\_HEAVY\_AND\_LEFT\_DOWN\_LIGHT =
- DOWN\_HEAVY\_AND\_LEFT\_UP\_LIGHT =
- VERTICAL\_HEAVY\_AND\_LEFT\_LIGHT =
- DOWN\_LIGHT\_AND\_LEFT\_UP\_HEAVY =
- UP\_LIGHT\_AND\_LEFT\_DOWN\_HEAVY =
- HEAVY\_VERTICAL\_AND\_LEFT =
- LIGHT\_DOWN\_AND\_HORIZONTAL =

- LEFT\_HEAVY\_AND\_RIGHT\_DOWN\_LIGHT =
- RIGHT\_HEAVY\_AND\_LEFT\_DOWN\_LIGHT =
- DOWN\_LIGHT\_AND\_HORIZONTAL\_HEAVY =
- DOWN\_HEAVY\_AND\_HORIZONTAL\_LIGHT =
- RIGHT LIGHT AND LEFT DOWN HEAVY =
- LEFT LIGHT AND RIGHT DOWN HEAVY =
- HEAVY\_DOWN\_AND\_HORIZONTAL =
- LIGHT\_UP\_AND\_HORIZONTAL =
- LEFT\_HEAVY\_AND\_RIGHT\_UP\_LIGHT =
- RIGHT\_HEAVY\_AND\_LEFT\_UP\_LIGHT =
- UP\_LIGHT\_AND\_HORIZONTAL\_HEAVY =
- UP\_HEAVY\_AND\_HORIZONTAL\_LIGHT =
- RIGHT\_LIGHT\_AND\_LEFT\_UP\_HEAVY =
- LEFT\_LIGHT\_AND\_RIGHT\_UP\_HEAVY =
- HEAVY\_UP\_AND\_HORIZONTAL =
- LIGHT VERTICAL AND HORIZONTAL =
- LEFT HEAVY AND RIGHT VERTICAL LIGHT =
- RIGHT\_HEAVY\_AND\_LEFT\_VERTICAL\_LIGHT =
- VERTICAL\_LIGHT\_AND\_HORIZONTAL\_HEAVY =
- UP\_HEAVY\_AND\_DOWN\_HORIZONTAL\_LIGHT =
- DOWN\_HEAVY\_AND\_UP\_HORIZONTAL\_LIGHT =
- VERTICAL\_HEAVY\_AND\_HORIZONTAL\_LIGHT =
- LEFT\_UP\_HEAVY\_AND\_RIGHT\_DOWN\_LIGHT =
- RIGHT\_UP\_HEAVY\_AND\_LEFT\_DOWN\_LIGHT =
- LEFT\_DOWN\_HEAVY\_AND\_RIGHT\_UP\_LIGHT =
- RIGHT\_DOWN\_HEAVY\_AND\_LEFT\_UP\_LIGHT =
- DOWN\_LIGHT\_AND\_UP\_HORIZONTAL\_HEAVY =
- UP LIGHT AND DOWN HORIZONTAL HEAVY =
- RIGHT\_LIGHT\_AND\_LEFT\_VERTICAL\_HEAVY =
- LEFT\_LIGHT\_AND\_RIGHT\_VERTICAL\_HEAVY =
- HEAVY\_VERTICAL\_AND\_HORIZONTAL =
- LIGHT\_DOUBLE\_DASH\_HORIZONTAL =
- HEAVY\_DOUBLE\_DASH\_HORIZONTAL =
- LIGHT\_DOUBLE\_DASH\_VERTICAL =
- HEAVY\_DOUBLE\_DASH\_VERTICAL =
- DOUBLE HORIZONTAL =

- DOUBLE VERTICAL =
- DOWN\_SINGLE\_AND\_RIGHT\_DOUBLE =
- DOWN\_DOUBLE\_AND\_RIGHT\_SINGLE =
- DOUBLE\_DOWN\_AND\_RIGHT =
- DOWN SINGLE AND LEFT DOUBLE =
- DOWN DOUBLE AND LEFT SINGLE =
- DOUBLE\_DOWN\_AND\_LEFT =
- UP\_SINGLE\_AND\_RIGHT\_DOUBLE =
- UP\_DOUBLE\_AND\_RIGHT\_SINGLE =
- DOUBLE\_UP\_AND\_RIGHT =
- UP\_SINGLE\_AND\_LEFT\_DOUBLE =
- UP\_DOUBLE\_AND\_LEFT\_SINGLE =
- DOUBLE\_UP\_AND\_LEFT =
- VERTICAL\_SINGLE\_AND\_RIGHT\_DOUBLE =
- VERTICAL\_DOUBLE\_AND\_RIGHT\_SINGLE =
- DOUBLE VERTICAL AND RIGHT =
- VERTICAL SINGLE AND LEFT DOUBLE =
- VERTICAL\_DOUBLE\_AND\_LEFT\_SINGLE =
- DOUBLE\_VERTICAL\_AND\_LEFT =
- DOWN\_SINGLE\_AND\_HORIZONTAL\_DOUBLE =
- DOWN\_DOUBLE\_AND\_HORIZONTAL\_SINGLE =
- DOUBLE\_DOWN\_AND\_HORIZONTAL =
- UP\_SINGLE\_AND\_HORIZONTAL\_DOUBLE =
- UP DOUBLE AND HORIZONTAL SINGLE =
- DOUBLE\_UP\_AND\_HORIZONTAL =
- VERTICAL\_SINGLE\_AND\_HORIZONTAL\_DOUBLE =
- VERTICAL\_DOUBLE\_AND\_HORIZONTAL\_SINGLE =
- DOUBLE\_VERTICAL\_AND\_HORIZONTAL =
- LIGHT\_ARC\_DOWN\_AND\_RIGHT =
- LIGHT\_ARC\_DOWN\_AND\_LEFT =
- LIGHT\_ARC\_UP\_AND\_LEFT =
- LIGHT\_ARC\_UP\_AND\_RIGHT =
- LIGHT\_DIAGONAL\_UPPER\_RIGHT\_TO\_LOWER\_LEFT =
- LIGHT\_DIAGONAL\_UPPER\_LEFT\_TO\_LOWER\_RIGHT = \
- LIGHT DIAGONAL CROSS =
- LIGHT\_LEFT =

- LIGHT\_UP =
- LIGHT\_RIGHT =
- LIGHT\_DOWN =
- HEAVY\_LEFT =
- HEAVY\_UP =
- HEAVY\_RIGHT =
- HEAVY\_DOWN =
- LIGHT\_LEFT\_AND\_HEAVY\_RIGHT =
- LIGHT\_UP\_AND\_HEAVY\_DOWN =
- HEAVY\_LEFT\_AND\_LIGHT\_RIGHT =
- HEAVY\_UP\_AND\_LIGHT\_DOWN =

\_\_init\_\_()

### **Methods**

\_\_init\_\_()

### **Attributes**

DOUBLE\_DOWN\_AND\_HORIZONTAL

 $DOUBLE\_DOWN\_AND\_LEFT$ 

DOUBLE\_DOWN\_AND\_RIGHT

DOUBLE\_HORIZONTAL

DOUBLE\_UP\_AND\_HORIZONTAL

DOUBLE\_UP\_AND\_LEFT

DOUBLE\_UP\_AND\_RIGHT

DOUBLE\_VERTICAL

DOUBLE\_VERTICAL\_AND\_HORIZONTAL

DOUBLE\_VERTICAL\_AND\_LEFT

DOUBLE\_VERTICAL\_AND\_RIGHT

continues on next page

Table 2 – continued from previous page

DOWN\_DOUBLE\_AND\_HORIZONTAL\_SINGLE

DOWN\_DOUBLE\_AND\_LEFT\_SINGLE

DOWN\_DOUBLE\_AND\_RIGHT\_SINGLE

DOWN\_HEAVY\_AND\_HORIZONTAL\_LIGHT

DOWN\_HEAVY\_AND\_LEFT\_LIGHT

DOWN\_HEAVY\_AND\_LEFT\_UP\_LIGHT

DOWN\_HEAVY\_AND\_RIGHT\_LIGHT

DOWN\_HEAVY\_AND\_RIGHT\_UP\_LIGHT

DOWN\_HEAVY\_AND\_UP\_HORIZONTAL\_LIGHT

DOWN\_LIGHT\_AND\_HORIZONTAL\_HEAVY

DOWN\_LIGHT\_AND\_LEFT\_HEAVY

DOWN\_LIGHT\_AND\_LEFT\_UP\_HEAVY

DOWN\_LIGHT\_AND\_RIGHT\_HEAVY

DOWN\_LIGHT\_AND\_RIGHT\_UP\_HEAVY

DOWN\_LIGHT\_AND\_UP\_HORIZONTAL\_HEAVY

DOWN\_SINGLE\_AND\_HORIZONTAL\_DOUBLE

DOWN\_SINGLE\_AND\_LEFT\_DOUBLE

DOWN\_SINGLE\_AND\_RIGHT\_DOUBLE

HEAVY\_DOUBLE\_DASH\_HORIZONTAL

HEAVY\_DOUBLE\_DASH\_VERTICAL

HEAVY\_DOWN

 ${\it HEAVY\_DOWN\_AND\_HORIZONTAL}$ 

 $HEAVY\_DOWN\_AND\_LEFT$ 

HEAVY\_DOWN\_AND\_RIGHT

HEAVY\_HORIZONTAL

HEAVY\_LEFT

continues on next page

Table 2 – continued from previous page

HEAVY\_LEFT\_AND\_LIGHT\_RIGHT

HEAVY\_QUADRUPLE\_DASH\_HORIZONTAL

HEAVY\_QUADRUPLE\_DASH\_VERTICAL

HEAVY\_RIGHT

HEAVY\_TRIPLE\_DASH\_HORIZONTAL

HEAVY\_TRIPLE\_DASH\_VERTICAL

HEAVY\_UP

HEAVY\_UP\_AND\_HORIZONTAL

HEAVY\_UP\_AND\_LEFT

HEAVY\_UP\_AND\_LIGHT\_DOWN

HEAVY\_UP\_AND\_RIGHT

HEAVY\_VERTICAL

HEAVY\_VERTICAL\_AND\_HORIZONTAL

 $HEAVY\_VERTICAL\_AND\_LEFT$ 

HEAVY\_VERTICAL\_AND\_RIGHT

LEFT\_DOWN\_HEAVY\_AND\_RIGHT\_UP\_LIGHT

LEFT\_HEAVY\_AND\_RIGHT\_DOWN\_LIGHT

LEFT\_HEAVY\_AND\_RIGHT\_UP\_LIGHT

LEFT\_HEAVY\_AND\_RIGHT\_VERTICAL\_LIGHT

LEFT\_LIGHT\_AND\_RIGHT\_DOWN\_HEAVY

 $LEFT\_LIGHT\_AND\_RIGHT\_UP\_HEAVY$ 

 $LEFT\_LIGHT\_AND\_RIGHT\_VERTICAL\_HEAVY$ 

LEFT\_UP\_HEAVY\_AND\_RIGHT\_DOWN\_LIGHT

LIGHT\_ARC\_DOWN\_AND\_LEFT

 $LIGHT\_ARC\_DOWN\_AND\_RIGHT$ 

LIGHT\_ARC\_UP\_AND\_LEFT

continues on next page

Table 2 – continued from previous page

LIGHT\_ARC\_UP\_AND\_RIGHT

LIGHT\_DIAGONAL\_CROSS

 $LIGHT\_DIAGONAL\_UPPER\_LEFT\_TO\_LOWER\_RIGHT$ 

LIGHT\_DIAGONAL\_UPPER\_RIGHT\_TO\_LOWER\_LEFT

LIGHT\_DOUBLE\_DASH\_HORIZONTAL

LIGHT\_DOUBLE\_DASH\_VERTICAL

LIGHT\_DOWN

LIGHT\_DOWN\_AND\_HORIZONTAL

 $LIGHT\_DOWN\_AND\_LEFT$ 

LIGHT\_DOWN\_AND\_RIGHT

LIGHT\_HORIZONTAL

LIGHT\_LEFT

 $LIGHT\_LEFT\_AND\_HEAVY\_RIGHT$ 

LIGHT\_QUADRUPLE\_DASH\_HORIZONTAL

LIGHT\_QUADRUPLE\_DASH\_VERTICAL

LIGHT\_RIGHT

LIGHT\_TRIPLE\_DASH\_HORIZONTAL

LIGHT\_TRIPLE\_DASH\_VERTICAL

LIGHT\_UP

LIGHT\_UP\_AND\_HEAVY\_DOWN

 $LIGHT\_UP\_AND\_HORIZONTAL$ 

 $LIGHT\_UP\_AND\_LEFT$ 

LIGHT\_UP\_AND\_RIGHT

LIGHT\_VERTICAL

 $LIGHT\_VERTICAL\_AND\_HORIZONTAL$ 

LIGHT\_VERTICAL\_AND\_LEFT

continues on next page

Table 2 – continued from previous page

LIGHT\_VERTICAL\_AND\_RIGHT

RIGHT\_DOWN\_HEAVY\_AND\_LEFT\_UP\_LIGHT

RIGHT\_HEAVY\_AND\_LEFT\_DOWN\_LIGHT

RIGHT\_HEAVY\_AND\_LEFT\_UP\_LIGHT

RIGHT\_HEAVY\_AND\_LEFT\_VERTICAL\_LIGHT

RIGHT\_LIGHT\_AND\_LEFT\_DOWN\_HEAVY

RIGHT\_LIGHT\_AND\_LEFT\_UP\_HEAVY

RIGHT\_LIGHT\_AND\_LEFT\_VERTICAL\_HEAVY

RIGHT\_UP\_HEAVY\_AND\_LEFT\_DOWN\_LIGHT

UP\_DOUBLE\_AND\_HORIZONTAL\_SINGLE

UP\_DOUBLE\_AND\_LEFT\_SINGLE

UP\_DOUBLE\_AND\_RIGHT\_SINGLE

UP\_HEAVY\_AND\_DOWN\_HORIZONTAL\_LIGHT

UP\_HEAVY\_AND\_HORIZONTAL\_LIGHT

UP\_HEAVY\_AND\_LEFT\_DOWN\_LIGHT

UP\_HEAVY\_AND\_LEFT\_LIGHT

UP\_HEAVY\_AND\_RIGHT\_DOWN\_LIGHT

UP\_HEAVY\_AND\_RIGHT\_LIGHT

UP\_LIGHT\_AND\_DOWN\_HORIZONTAL\_HEAVY

UP\_LIGHT\_AND\_HORIZONTAL\_HEAVY

 $UP\_LIGHT\_AND\_LEFT\_DOWN\_HEAVY$ 

UP\_LIGHT\_AND\_LEFT\_HEAVY

UP\_LIGHT\_AND\_RIGHT\_DOWN\_HEAVY

UP\_LIGHT\_AND\_RIGHT\_HEAVY

 $UP\_SINGLE\_AND\_HORIZONTAL\_DOUBLE$ 

UP\_SINGLE\_AND\_LEFT\_DOUBLE

continues on next page

Table 2 – continued from previous page

```
UP_SINGLE_AND_RIGHT_DOUBLE
VERTICAL_DOUBLE_AND_HORIZONTAL_SINGLE
VERTICAL_DOUBLE_AND_LEFT_SINGLE
VERTICAL_DOUBLE_AND_RIGHT_SINGLE
VERTICAL_HEAVY_AND_HORIZONTAL_LIGHT
VERTICAL_HEAVY_AND_LEFT_LIGHT
VERTICAL_HEAVY_AND_RIGHT_LIGHT
VERTICAL_LIGHT_AND_HORIZONTAL_HEAVY
VERTICAL_LIGHT_AND_LEFT_HEAVY
VERTICAL_LIGHT_AND_RIGHT_HEAVY
VERTICAL_SINGLE_AND_HORIZONTAL_DOUBLE
VERTICAL_SINGLE_AND_LEFT_DOUBLE
VERTICAL_SINGLE_AND_RIGHT_DOUBLE
```

```
DOUBLE_DOWN_AND_HORIZONTAL = ''
DOUBLE_DOWN_AND_LEFT = ''
DOUBLE_DOWN_AND_RIGHT = ''
DOUBLE_HORIZONTAL = ''
DOUBLE_UP_AND_HORIZONTAL = ''
DOUBLE_UP_AND_LEFT = ''
DOUBLE_UP_AND_RIGHT = ''
DOUBLE_VERTICAL = ''
DOUBLE_VERTICAL_AND_HORIZONTAL = ''
DOUBLE_VERTICAL_AND_LEFT = ''
DOUBLE_VERTICAL_AND_RIGHT = ''
DOWN_DOUBLE_AND_HORIZONTAL_SINGLE = ''
DOWN_DOUBLE_AND_LEFT_SINGLE = ''
DOWN_DOUBLE_AND_RIGHT_SINGLE = ''
```

```
DOWN_HEAVY_AND_HORIZONTAL_LIGHT = ''
DOWN_HEAVY_AND_LEFT_LIGHT = ''
DOWN_HEAVY_AND_LEFT_UP_LIGHT = ''
DOWN_HEAVY_AND_RIGHT_LIGHT = ''
DOWN_HEAVY_AND_RIGHT_UP_LIGHT = ''
DOWN_HEAVY_AND_UP_HORIZONTAL_LIGHT = ''
DOWN_LIGHT_AND_HORIZONTAL_HEAVY = ''
DOWN_LIGHT_AND_LEFT_HEAVY = ''
DOWN_LIGHT_AND_LEFT_UP_HEAVY = ''
DOWN_LIGHT_AND_RIGHT_HEAVY = ''
DOWN_LIGHT_AND_RIGHT_UP_HEAVY = ''
DOWN_LIGHT_AND_UP_HORIZONTAL_HEAVY = ''
DOWN_SINGLE_AND_HORIZONTAL_DOUBLE = ''
DOWN_SINGLE_AND_LEFT_DOUBLE = ''
DOWN_SINGLE_AND_RIGHT_DOUBLE = ''
HEAVY_DOUBLE_DASH_HORIZONTAL = ''
HEAVY_DOUBLE_DASH_VERTICAL = ''
HEAVY_DOWN = ''
HEAVY_DOWN_AND_HORIZONTAL = ''
HEAVY_DOWN_AND_LEFT = ''
HEAVY_DOWN_AND_RIGHT = ''
HEAVY_HORIZONTAL = ''
HEAVY_LEFT = ''
HEAVY_LEFT_AND_LIGHT_RIGHT = ''
HEAVY_QUADRUPLE_DASH_HORIZONTAL = ''
HEAVY_QUADRUPLE_DASH_VERTICAL = ''
HEAVY_RIGHT = ''
HEAVY_TRIPLE_DASH_HORIZONTAL = ''
HEAVY_TRIPLE_DASH_VERTICAL = ''
HEAVY_UP = ''
HEAVY_UP_AND_HORIZONTAL = ''
```

```
HEAVY_UP_AND_LEFT = ''
HEAVY_UP_AND_LIGHT_DOWN = ''
HEAVY_UP_AND_RIGHT = ''
HEAVY_VERTICAL = ''
HEAVY_VERTICAL_AND_HORIZONTAL = ''
HEAVY_VERTICAL_AND_LEFT = ''
HEAVY_VERTICAL_AND_RIGHT = ''
LEFT_DOWN_HEAVY_AND_RIGHT_UP_LIGHT = ''
LEFT_HEAVY_AND_RIGHT_DOWN_LIGHT = ''
LEFT_HEAVY_AND_RIGHT_UP_LIGHT = ''
LEFT_HEAVY_AND_RIGHT_VERTICAL_LIGHT = ''
LEFT_LIGHT_AND_RIGHT_DOWN_HEAVY = ''
LEFT_LIGHT_AND_RIGHT_UP_HEAVY = ''
LEFT_LIGHT_AND_RIGHT_VERTICAL_HEAVY = ''
LEFT_UP_HEAVY_AND_RIGHT_DOWN_LIGHT = ''
LIGHT_ARC_DOWN_AND_LEFT = ''
LIGHT_ARC_DOWN_AND_RIGHT = ''
LIGHT_ARC_UP_AND_LEFT = ''
LIGHT_ARC_UP_AND_RIGHT = ''
LIGHT_DIAGONAL_CROSS = ''
LIGHT_DIAGONAL_UPPER_LEFT_TO_LOWER_RIGHT = '\'
LIGHT_DIAGONAL_UPPER_RIGHT_TO_LOWER_LEFT = ''
LIGHT_DOUBLE_DASH_HORIZONTAL = ''
LIGHT_DOUBLE_DASH_VERTICAL = ''
LIGHT_DOWN = ''
LIGHT_DOWN_AND_HORIZONTAL = ''
LIGHT_DOWN_AND_LEFT = ''
LIGHT_DOWN_AND_RIGHT = ''
LIGHT_HORIZONTAL = '-'
LIGHT_LEFT = ''
LIGHT_LEFT_AND_HEAVY_RIGHT = ''
```

```
LIGHT_QUADRUPLE_DASH_HORIZONTAL = ''
LIGHT_QUADRUPLE_DASH_VERTICAL = ''
LIGHT_RIGHT = ''
LIGHT_TRIPLE_DASH_HORIZONTAL = ''
LIGHT_TRIPLE_DASH_VERTICAL = ''
LIGHT_UP = ''
LIGHT_UP_AND_HEAVY_DOWN = ''
LIGHT_UP_AND_HORIZONTAL = ''
LIGHT_UP_AND_LEFT = ''
LIGHT_UP_AND_RIGHT = 'L'
LIGHT_VERTICAL = '|'
LIGHT_VERTICAL_AND_HORIZONTAL = ''
LIGHT_VERTICAL_AND_LEFT = ''
LIGHT_VERTICAL_AND_RIGHT = '-
RIGHT_DOWN_HEAVY_AND_LEFT_UP_LIGHT = ''
RIGHT_HEAVY_AND_LEFT_DOWN_LIGHT = ''
RIGHT_HEAVY_AND_LEFT_UP_LIGHT = ''
RIGHT_HEAVY_AND_LEFT_VERTICAL_LIGHT = ''
RIGHT_LIGHT_AND_LEFT_DOWN_HEAVY = ''
RIGHT_LIGHT_AND_LEFT_UP_HEAVY = ''
RIGHT_LIGHT_AND_LEFT_VERTICAL_HEAVY = ''
RIGHT_UP_HEAVY_AND_LEFT_DOWN_LIGHT = ''
UP_DOUBLE_AND_HORIZONTAL_SINGLE = ''
UP_DOUBLE_AND_LEFT_SINGLE = ''
UP_DOUBLE_AND_RIGHT_SINGLE = ''
UP_HEAVY_AND_DOWN_HORIZONTAL_LIGHT = ''
UP_HEAVY_AND_HORIZONTAL_LIGHT = ''
UP_HEAVY_AND_LEFT_DOWN_LIGHT = ''
UP_HEAVY_AND_LEFT_LIGHT = ''
UP_HEAVY_AND_RIGHT_DOWN_LIGHT = ''
UP_HEAVY_AND_RIGHT_LIGHT = ''
```

```
UP_LIGHT_AND_DOWN_HORIZONTAL_HEAVY = ''
UP_LIGHT_AND_HORIZONTAL_HEAVY = ''
UP_LIGHT_AND_LEFT_DOWN_HEAVY = ''
UP_LIGHT_AND_LEFT_HEAVY = ''
UP_LIGHT_AND_RIGHT_DOWN_HEAVY = ''
UP_LIGHT_AND_RIGHT_HEAVY = ''
UP_SINGLE_AND_HORIZONTAL_DOUBLE = ''
UP_SINGLE_AND_LEFT_DOUBLE = ''
UP_SINGLE_AND_RIGHT_DOUBLE = ''
VERTICAL_DOUBLE_AND_HORIZONTAL_SINGLE = ''
VERTICAL_DOUBLE_AND_LEFT_SINGLE = ''
VERTICAL_DOUBLE_AND_RIGHT_SINGLE = ''
VERTICAL_HEAVY_AND_HORIZONTAL_LIGHT = ''
VERTICAL_HEAVY_AND_LEFT_LIGHT = ''
VERTICAL_HEAVY_AND_RIGHT_LIGHT = ''
VERTICAL_LIGHT_AND_HORIZONTAL_HEAVY = ''
VERTICAL_LIGHT_AND_LEFT_HEAVY = ''
VERTICAL_LIGHT_AND_RIGHT_HEAVY = ''
VERTICAL_SINGLE_AND_HORIZONTAL_DOUBLE = ''
VERTICAL_SINGLE_AND_LEFT_DOUBLE = ''
VERTICAL_SINGLE_AND_RIGHT_DOUBLE = ''
```

## GeometricShapes

class pygamelib.assets.graphics.GeometricShapes

Bases: object

Geometric shapes elements (unicode)

Here is the list of supported glyphs:

- BLACK SQUARE =
- BLACK\_LARGE\_SQUARE =
- WHITE\_SQUARE =
- WHITE\_SQUARE\_WITH\_ROUNDED\_CORNERS =
- WHITE\_SQUARE\_CONTAINING\_BLACK\_SMALL\_SQUARE =
- SQUARE\_WITH\_HORIZONTAL\_FILL =

- SQUARE\_WITH\_VERTICAL\_FILL =
- SQUARE\_WITH\_ORTHOGONAL\_CROSSHATCH\_FILL =
- SQUARE\_WITH\_UPPER\_LEFT\_TO\_LOWER\_RIGHT\_FILL =
- SQUARE\_WITH\_UPPER\_RIGHT\_TO\_LOWER\_LEFT\_FILL =
- SQUARE WITH DIAGONAL CROSSHATCH FILL =
- BLACK SMALL SQUARE =
- WHITE\_SMALL\_SQUARE =
- BLACK\_RECTANGLE =
- WHITE\_RECTANGLE =
- BLACK\_VERTICAL\_RECTANGLE =
- WHITE\_VERTICAL\_RECTANGLE =
- BLACK\_PARALLELOGRAM =
- WHITE\_PARALLELOGRAM =
- BLACK\_UP\_POINTING\_TRIANGLE =
- WHITE\_UP\_POINTING\_TRIANGLE =
- BLACK UP POINTING SMALL TRIANGLE =
- WHITE\_UP\_POINTING\_SMALL\_TRIANGLE =
- BLACK\_RIGHT\_POINTING\_TRIANGLE =
- WHITE\_RIGHT\_POINTING\_TRIANGLE =
- BLACK\_RIGHT\_POINTING\_SMALL\_TRIANGLE =
- WHITE\_RIGHT\_POINTING\_SMALL\_TRIANGLE =
- BLACK\_RIGHT\_POINTING\_POINTER =
- WHITE\_RIGHT\_POINTING\_POINTER =
- BLACK\_DOWN\_POINTING\_TRIANGLE =
- WHITE DOWN POINTING TRIANGLE =
- BLACK\_DOWN\_POINTING\_SMALL\_TRIANGLE =
- WHITE\_DOWN\_POINTING\_SMALL\_TRIANGLE =
- BLACK\_LEFT\_POINTING\_TRIANGLE =
- WHITE\_LEFT\_POINTING\_TRIANGLE =
- BLACK\_LEFT\_POINTING\_SMALL\_TRIANGLE =
- WHITE\_LEFT\_POINTING\_SMALL\_TRIANGLE =
- BLACK\_LEFT\_POINTING\_POINTER =
- WHITE\_LEFT\_POINTING\_POINTER =
- BLACK\_DIAMOND =
- WHITE\_DIAMOND =
- WHITE\_DIAMOND\_CONTAINING\_BLACK\_SMALL\_DIAMOND =

- FISHEYE =
- LOZENGE =
- WHITE\_CIRCLE =
- DOTTED\_CIRCLE =
- CIRCLE\_WITH\_VERTICAL\_FILL =
- BULLSEYE =
- BLACK\_CIRCLE =
- CIRCLE\_WITH\_LEFT\_HALF\_BLACK =
- CIRCLE\_WITH\_RIGHT\_HALF\_BLACK =
- CIRCLE\_WITH\_LOWER\_HALF\_BLACK =
- CIRCLE\_WITH\_UPPER\_HALF\_BLACK =
- CIRCLE\_WITH\_UPPER\_RIGHT\_QUADRANT\_BLACK =
- CIRCLE\_WITH\_ALL\_BUT\_UPPER\_LEFT\_QUADRANT\_BLACK =
- LEFT\_HALF\_BLACK\_CIRCLE =
- RIGHT\_HALF\_BLACK\_CIRCLE =
- INVERSE BULLET =
- INVERSE\_WHITE\_CIRCLE =
- UPPER\_HALF\_INVERSE\_WHITE\_CIRCLE =
- LOWER\_HALF\_INVERSE\_WHITE\_CIRCLE =
- UPPER\_LEFT\_QUADRANT\_CIRCULAR\_ARC =
- UPPER\_RIGHT\_QUADRANT\_CIRCULAR\_ARC =
- LOWER\_RIGHT\_QUADRANT\_CIRCULAR\_ARC =
- LOWER\_LEFT\_QUADRANT\_CIRCULAR\_ARC =
- UPPER\_HALF\_CIRCLE =
- LOWER\_HALF\_CIRCLE =
- BLACK\_LOWER\_RIGHT\_TRIANGLE =
- BLACK\_LOWER\_LEFT\_TRIANGLE =
- BLACK\_UPPER\_LEFT\_TRIANGLE =
- BLACK\_UPPER\_RIGHT\_TRIANGLE =
- WHITE\_BULLET = •
- BULLET = •
- RING\_OPERATOR =
- SQUARE\_WITH\_LEFT\_HALF\_BLACK =
- SQUARE\_WITH\_RIGHT\_HALF\_BLACK =
- SQUARE\_WITH\_UPPER\_LEFT\_DIAGONAL\_HALF\_BLACK =
- SQUARE\_WITH\_LOWER\_RIGHT\_DIAGONAL\_HALF\_BLACK =

- WHITE\_SQUARE\_WITH\_VERTICAL\_BISECTING\_LINE =
- WHITE\_UP\_POINTING\_TRIANGLE\_WITH\_DOT =
- UP\_POINTING\_TRIANGLE\_WITH\_LEFT\_HALF\_BLACK =
- UP\_POINTING\_TRIANGLE\_WITH\_RIGHT\_HALF\_BLACK =
- LARGE CIRCLE = O
- WHITE\_SQUARE\_WITH\_UPPER\_LEFT\_QUADRANT =
- WHITE\_SQUARE\_WITH\_LOWER\_LEFT\_QUADRANT =
- WHITE\_SQUARE\_WITH\_LOWER\_RIGHT\_QUADRANT =
- WHITE\_SQUARE\_WITH\_UPPER\_RIGHT\_QUADRANT =
- WHITE\_CIRCLE\_WITH\_UPPER\_LEFT\_QUADRANT =
- WHITE\_CIRCLE\_WITH\_LOWER\_LEFT\_QUADRANT =
- WHITE\_CIRCLE\_WITH\_LOWER\_RIGHT\_QUADRANT =
- WHITE\_CIRCLE\_WITH\_UPPER\_RIGHT\_QUADRANT =
- UPPER\_LEFT\_TRIANGLE =
- UPPER\_RIGHT\_TRIANGLE =
- LOWER LEFT TRIANGLE =
- WHITE MEDIUM SQUARE =
- BLACK\_MEDIUM\_SQUARE =
- WHITE\_MEDIUM\_SMALL\_SQUARE =
- BLACK\_MEDIUM\_SMALL\_SQUARE =
- LOWER\_RIGHT\_TRIANGLE =

\_\_init\_\_()

### **Methods**

\_\_init\_\_()

#### **Attributes**

BLACK\_CIRCLE

BLACK\_DIAMOND

BLACK\_DOWN\_POINTING\_SMALL\_TRIANGLE

BLACK\_DOWN\_POINTING\_TRIANGLE

continues on next page

Table 3 – continued from previous page

BLACK\_LARGE\_SQUARE

BLACK\_LEFT\_POINTING\_POINTER

BLACK\_LEFT\_POINTING\_SMALL\_TRIANGLE

 $BLACK\_LEFT\_POINTING\_TRIANGLE$ 

BLACK\_LOWER\_LEFT\_TRIANGLE

BLACK\_LOWER\_RIGHT\_TRIANGLE

BLACK\_MEDIUM\_SMALL\_SQUARE

BLACK\_MEDIUM\_SQUARE

BLACK\_PARALLELOGRAM

BLACK\_RECTANGLE

BLACK\_RIGHT\_POINTING\_POINTER

BLACK\_RIGHT\_POINTING\_SMALL\_TRIANGLE

BLACK\_RIGHT\_POINTING\_TRIANGLE

BLACK\_SMALL\_SQUARE

BLACK\_SQUARE

BLACK\_UPPER\_LEFT\_TRIANGLE

BLACK\_UPPER\_RIGHT\_TRIANGLE

BLACK\_UP\_POINTING\_SMALL\_TRIANGLE

BLACK\_UP\_POINTING\_TRIANGLE

BLACK\_VERTICAL\_RECTANGLE

BULLET

BULLSEYE

CIRCLE\_WITH\_ALL\_BUT\_UPPER\_LEFT\_QUADRANT\_B.

CIRCLE\_WITH\_LEFT\_HALF\_BLACK

 $CIRCLE\_WITH\_LOWER\_HALF\_BLACK$ 

CIRCLE\_WITH\_RIGHT\_HALF\_BLACK

continues on next page

Table 3 – continued from previous page

CIRCLE\_WITH\_UPPER\_HALF\_BLACK

CIRCLE\_WITH\_UPPER\_RIGHT\_QUADRANT\_BLACK

CIRCLE\_WITH\_VERTICAL\_FILL

DOTTED\_CIRCLE

FISHEYE

INVERSE\_BULLET

INVERSE\_WHITE\_CIRCLE

LARGE\_CIRCLE

LEFT\_HALF\_BLACK\_CIRCLE

LOWER\_HALF\_CIRCLE

LOWER\_HALF\_INVERSE\_WHITE\_CIRCLE

LOWER\_LEFT\_QUADRANT\_CIRCULAR\_ARC

LOWER\_LEFT\_TRIANGLE

 $LOWER\_RIGHT\_QUADRANT\_CIRCULAR\_ARC$ 

LOWER\_RIGHT\_TRIANGLE

**LOZENGE** 

RIGHT\_HALF\_BLACK\_CIRCLE

RING\_OPERATOR

SQUARE\_WITH\_DIAGONAL\_CROSSHATCH\_FILL

SQUARE\_WITH\_HORIZONTAL\_FILL

SQUARE\_WITH\_LEFT\_HALF\_BLACK

 $SQUARE\_WITH\_LOWER\_RIGHT\_DIAGONAL\_HALF\_BLA$ 

SQUARE\_WITH\_ORTHOGONAL\_CROSSHATCH\_FILL

SQUARE\_WITH\_RIGHT\_HALF\_BLACK

SQUARE\_WITH\_UPPER\_LEFT\_DIAGONAL\_HALF\_BLAC

SQUARE\_WITH\_UPPER\_LEFT\_TO\_LOWER\_RIGHT\_FIL.

continues on next page

Table 3 – continued from previous page

SQUARE\_WITH\_UPPER\_RIGHT\_TO\_LOWER\_LEFT\_FIL.

SQUARE\_WITH\_VERTICAL\_FILL

UPPER\_HALF\_CIRCLE

UPPER\_HALF\_INVERSE\_WHITE\_CIRCLE

UPPER\_LEFT\_QUADRANT\_CIRCULAR\_ARC

UPPER\_LEFT\_TRIANGLE

UPPER\_RIGHT\_QUADRANT\_CIRCULAR\_ARC

UPPER\_RIGHT\_TRIANGLE

UP\_POINTING\_TRIANGLE\_WITH\_LEFT\_HALF\_BLACK

UP\_POINTING\_TRIANGLE\_WITH\_RIGHT\_HALF\_BLAC

WHITE\_BULLET

WHITE\_CIRCLE

WHITE\_CIRCLE\_WITH\_LOWER\_LEFT\_QUADRANT

WHITE\_CIRCLE\_WITH\_LOWER\_RIGHT\_QUADRANT

WHITE\_CIRCLE\_WITH\_UPPER\_LEFT\_QUADRANT

WHITE\_CIRCLE\_WITH\_UPPER\_RIGHT\_QUADRANT

WHITE\_DIAMOND

WHITE\_DIAMOND\_CONTAINING\_BLACK\_SMALL\_DIAM

WHITE\_DOWN\_POINTING\_SMALL\_TRIANGLE

WHITE\_DOWN\_POINTING\_TRIANGLE

WHITE\_LEFT\_POINTING\_POINTER

 ${\it WHITE\_LEFT\_POINTING\_SMALL\_TRIANGLE}$ 

WHITE\_LEFT\_POINTING\_TRIANGLE

WHITE\_MEDIUM\_SMALL\_SQUARE

 ${\it WHITE\_MEDIUM\_SQUARE}$ 

WHITE\_PARALLELOGRAM

continues on next page

Table 3 – continued from previous page

```
WHITE_RECTANGLE
WHITE_RIGHT_POINTING_POINTER
WHITE_RIGHT_POINTING_SMALL_TRIANGLE
WHITE_RIGHT_POINTING_TRIANGLE
WHITE_SMALL_SQUARE
WHITE_SQUARE
WHITE_SQUARE_CONTAINING_BLACK_SMALL_SQUAR.
WHITE_SQUARE_WITH_LOWER_LEFT_QUADRANT
WHITE_SQUARE_WITH_LOWER_RIGHT_QUADRANT
WHITE_SQUARE_WITH_ROUNDED_CORNERS
WHITE_SQUARE_WITH_UPPER_LEFT_QUADRANT
WHITE_SQUARE_WITH_UPPER_RIGHT_QUADRANT
WHITE_SQUARE_WITH_VERTICAL_BISECTING_LINE
WHITE_UP_POINTING_SMALL_TRIANGLE
WHITE_UP_POINTING_TRIANGLE
WHITE_UP_POINTING_TRIANGLE_WITH_DOT
WHITE VERTICAL RECTANGLE
```

```
BLACK_CIRCLE = ''

BLACK_DIAMOND = ''

BLACK_DOWN_POINTING_SMALL_TRIANGLE = ''

BLACK_DOWN_POINTING_TRIANGLE = ''

BLACK_LARGE_SQUARE = ''

BLACK_LEFT_POINTING_POINTER = ''

BLACK_LEFT_POINTING_SMALL_TRIANGLE = ''

BLACK_LEFT_POINTING_TRIANGLE = ''

BLACK_LOWER_LEFT_TRIANGLE = ''

BLACK_LOWER_RIGHT_TRIANGLE = ''
```

```
BLACK_MEDIUM_SMALL_SQUARE = ''
BLACK_MEDIUM_SQUARE = ''
BLACK_PARALLELOGRAM = ''
BLACK_RECTANGLE = ''
BLACK_RIGHT_POINTING_POINTER = ''
BLACK_RIGHT_POINTING_SMALL_TRIANGLE = ''
BLACK_RIGHT_POINTING_TRIANGLE = ''
BLACK_SMALL_SQUARE = ''
BLACK_SQUARE = ''
BLACK_UPPER_LEFT_TRIANGLE = ''
BLACK_UPPER_RIGHT_TRIANGLE = ''
BLACK_UP_POINTING_SMALL_TRIANGLE = ''
BLACK_UP_POINTING_TRIANGLE = ''
BLACK_VERTICAL_RECTANGLE = ''
BULLET = '•'
BULLSEYE = ''
CIRCLE_WITH_ALL_BUT_UPPER_LEFT_QUADRANT_BLACK = ''
CIRCLE_WITH_LEFT_HALF_BLACK = ''
CIRCLE_WITH_LOWER_HALF_BLACK = ''
CIRCLE_WITH_RIGHT_HALF_BLACK = ''
CIRCLE_WITH_UPPER_HALF_BLACK = ''
CIRCLE_WITH_UPPER_RIGHT_QUADRANT_BLACK = ''
CIRCLE_WITH_VERTICAL_FILL = ''
DOTTED_CIRCLE = ''
FISHEYE = ''
INVERSE_BULLET = ''
INVERSE_WHITE_CIRCLE = ''
LARGE_CIRCLE = '0'
LEFT_HALF_BLACK_CIRCLE = ''
LOWER_HALF_CIRCLE = ''
LOWER_HALF_INVERSE_WHITE_CIRCLE = ''
```

```
LOWER_LEFT_QUADRANT_CIRCULAR_ARC = ''
LOWER_LEFT_TRIANGLE = ''
LOWER_RIGHT_QUADRANT_CIRCULAR_ARC = ''
LOWER_RIGHT_TRIANGLE = ''
LOZENGE = ''
RIGHT_HALF_BLACK_CIRCLE = ''
RING_OPERATOR = ''
SQUARE_WITH_DIAGONAL_CROSSHATCH_FILL = ''
SQUARE_WITH_HORIZONTAL_FILL = ''
SQUARE_WITH_LEFT_HALF_BLACK = ''
SQUARE_WITH_LOWER_RIGHT_DIAGONAL_HALF_BLACK = ''
SQUARE_WITH_ORTHOGONAL_CROSSHATCH_FILL = ''
SQUARE_WITH_RIGHT_HALF_BLACK = ''
SQUARE_WITH_UPPER_LEFT_DIAGONAL_HALF_BLACK = ''
SQUARE_WITH_UPPER_LEFT_TO_LOWER_RIGHT_FILL = ''
SQUARE_WITH_UPPER_RIGHT_TO_LOWER_LEFT_FILL = ''
SQUARE_WITH_VERTICAL_FILL = ''
UPPER_HALF_CIRCLE = ''
UPPER_HALF_INVERSE_WHITE_CIRCLE = ''
UPPER_LEFT_QUADRANT_CIRCULAR_ARC = ''
UPPER_LEFT_TRIANGLE = ''
UPPER_RIGHT_QUADRANT_CIRCULAR_ARC = ''
UPPER_RIGHT_TRIANGLE = ''
UP_POINTING_TRIANGLE_WITH_LEFT_HALF_BLACK = ''
UP_POINTING_TRIANGLE_WITH_RIGHT_HALF_BLACK = ''
WHITE_BULLET = 'o'
WHITE_CIRCLE = ''
WHITE_CIRCLE_WITH_LOWER_LEFT_QUADRANT = ''
WHITE_CIRCLE_WITH_LOWER_RIGHT_QUADRANT = ''
WHITE_CIRCLE_WITH_UPPER_LEFT_QUADRANT = ''
WHITE_CIRCLE_WITH_UPPER_RIGHT_QUADRANT = ''
```

```
WHITE_DIAMOND = ''
WHITE_DIAMOND_CONTAINING_BLACK_SMALL_DIAMOND = ''
WHITE_DOWN_POINTING_SMALL_TRIANGLE = ''
WHITE_DOWN_POINTING_TRIANGLE = ''
WHITE_LEFT_POINTING_POINTER = ''
WHITE_LEFT_POINTING_SMALL_TRIANGLE = ''
WHITE_LEFT_POINTING_TRIANGLE = ''
WHITE_MEDIUM_SMALL_SQUARE = ''
WHITE_MEDIUM_SQUARE = ''
WHITE_PARALLELOGRAM = ''
WHITE_RECTANGLE = ''
WHITE_RIGHT_POINTING_POINTER = ''
WHITE_RIGHT_POINTING_SMALL_TRIANGLE = ''
WHITE_RIGHT_POINTING_TRIANGLE = ''
WHITE_SMALL_SQUARE = ''
WHITE_SQUARE = ''
WHITE_SQUARE_CONTAINING_BLACK_SMALL_SQUARE = ''
WHITE_SQUARE_WITH_LOWER_LEFT_QUADRANT = ''
WHITE_SQUARE_WITH_LOWER_RIGHT_QUADRANT = ''
WHITE_SQUARE_WITH_ROUNDED_CORNERS = ''
WHITE_SQUARE_WITH_UPPER_LEFT_QUADRANT = ''
WHITE_SQUARE_WITH_UPPER_RIGHT_QUADRANT = ''
WHITE_SQUARE_WITH_VERTICAL_BISECTING_LINE = ''
WHITE_UP_POINTING_SMALL_TRIANGLE = ''
WHITE_UP_POINTING_TRIANGLE = ''
WHITE_UP_POINTING_TRIANGLE_WITH_DOT = ''
WHITE_VERTICAL_RECTANGLE = ''
```

## **MiscTechnicals**

## class pygamelib.assets.graphics.MiscTechnicals

Bases: object

Miscellanous Technical block (unicode)

## Here is the list of supported glyphs:

- DIAMETER\_SIGN = ""
- ELECTRIC\_ARROW = ""
- HOUSE = ""
- UP\_ARROWHEAD = ""
- DOWN ARROWHEAD = ""
- PROJECTIVE = ""
- PERSPECTIVE = ""
- WAVY\_LINE = ""
- LEFT\_CEILING = ""
- RIGHT\_CEILING = ""
- LEFT\_FLOOR = ""
- RIGHT\_FLOOR = ""
- BOTTOM\_RIGHT\_CROP = ""
- BOTTOM\_LEFT\_CROP = ""
- TOP\_RIGHT\_CROP = ""
- TOP\_LEFT\_CROP = ""
- REVERSED NOT SIGN = ""
- SQUARE\_LOZENGE = ""
- ARC = ""
- SEGMENT = ""
- SECTOR = ""
- TELEPHONE\_RECORDER = ""
- POSITION\_INDICATOR = ""
- VIEWDATA\_SQUARE = ""
- PLACE\_OF\_INTEREST\_SIGN = ""
- TURNED\_NOT\_SIGN = ""
- WATCH = ""
- HOURGLASS = ""
- TOP\_LEFT\_CORNER = ""
- TOP\_RIGHT\_CORNER = ""
- BOTTOM\_LEFT\_CORNER = ""

- BOTTOM RIGHT CORNER = ""
- TOP HALF INTEGRAL = ""
- BOTTOM\_HALF\_INTEGRAL = ""
- FROWN = ""
- SMILE = ""
- UP ARROWHEAD BETWEEN TWO HORIZONTAL BARS = ""
- OPTION\_KEY = ""
- ERASE\_TO\_THE\_RIGHT = ""
- X\_IN\_A\_RECTANGLE\_BOX = ""
- KEYBOARD = ""
- LEFT\_POINTING\_ANGLE\_BRACKET = "\"
- RIGHT\_POINTING\_ANGLE\_BRACKET = ">"
- ERASE\_TO\_THE\_LEFT = ""
- BENZENE RING = ""
- CYLINDRICITY = ""
- ALL\_AROUND\_PROFILE = ""
- **SYMMETRY** = ""
- TOTAL RUNOUT = ""
- DIMENSION\_ORIGIN = ""
- CONICAL TAPER = ""
- SLOPE = ""
- COUNTERBORE = ""
- COUNTERSINK = ""
- APL\_FUNCTIONAL\_SYMBOL\_I\_BEAM = ""
- APL\_FUNCTIONAL\_SYMBOL\_SQUISH\_QUAD = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_EQUAL = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DIVIDE = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DIAMOND = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_JOT = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_CIRCLE = ""
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_STILE = ""
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_JOT = ""
- APL\_FUNCTIONAL\_SYMBOL\_SLASH\_BAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_BACKSLASH\_BAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_SLASH = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_BACKSLASH = ""

- APL FUNCTIONAL SYMBOL QUAD LESS THAN = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_GREATER\_THAN = ""
- APL\_FUNCTIONAL\_SYMBOL\_LEFTWARDS\_VANE = ""
- APL\_FUNCTIONAL\_SYMBOL\_RIGHTWARDS\_VANE = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_LEFTWARDS\_ARROW = ""
- APL FUNCTIONAL SYMBOL QUAD RIGHTWARDS ARROW = ""
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_BACKSLASH = ""
- APL\_FUNCTIONAL\_SYMBOL\_DOWN\_TACK\_UNDERBAR = ""
- APL FUNCTIONAL SYMBOL DELTA STILE = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DOWN\_CARET = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DELTA = ""
- APL\_FUNCTIONAL\_SYMBOL\_DOWN\_TACK\_JOT = ""
- APL\_FUNCTIONAL\_SYMBOL\_UPWARDS\_VANE = ""
- APL FUNCTIONAL SYMBOL QUAD UPWARDS ARROW = ""
- APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_OVERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_DEL\_STILE = ""
- APL FUNCTIONAL SYMBOL QUAD UP CARET = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DEL = ""
- APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_JOT = ""
- APL\_FUNCTIONAL\_SYMBOL\_DOWNWARDS\_VANE = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DOWNWARDS\_ARROW = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUOTE\_UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_DELTA\_UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_DIAMOND\_UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_JOT\_UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_UP\_SHOE\_JOT = ""
- APL FUNCTIONAL SYMBOL QUOTE QUAD = ""
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_STAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_COLON = ""
- APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_DIAERESIS = ""
- APL\_FUNCTIONAL\_SYMBOL\_DEL\_DIAERESIS = ""
- APL\_FUNCTIONAL\_SYMBOL\_STAR\_DIAERESIS = ""
- APL\_FUNCTIONAL\_SYMBOL\_JOT\_DIAERESIS = ""
- APL FUNCTIONAL SYMBOL CIRCLE DIAERESIS = ""
- APL FUNCTIONAL SYMBOL DOWN SHOE STILE = ""

- APL FUNCTIONAL SYMBOL LEFT SHOE STILE = ""
- APL\_FUNCTIONAL\_SYMBOL\_TILDE\_DIAERESIS = ""
- APL\_FUNCTIONAL\_SYMBOL\_GREATER\_THAN\_DIAERESIS = ""
- APL\_FUNCTIONAL\_SYMBOL\_COMMA\_BAR = ""
- APL FUNCTIONAL SYMBOL DEL TILDE = ""
- APL FUNCTIONAL SYMBOL ZILDE = ""
- APL\_FUNCTIONAL\_SYMBOL\_STILE\_TILDE = ""
- APL\_FUNCTIONAL\_SYMBOL\_SEMICOLON\_UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_NOT\_EQUAL = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_QUESTION = ""
- APL\_FUNCTIONAL\_SYMBOL\_DOWN\_CARET\_TILDE = ""
- APL\_FUNCTIONAL\_SYMBOL\_UP\_CARET\_TILDE = ""
- APL\_FUNCTIONAL\_SYMBOL\_IOTA = ""
- APL\_FUNCTIONAL\_SYMBOL\_RHO = ""
- APL\_FUNCTIONAL\_SYMBOL\_OMEGA = ""
- APL\_FUNCTIONAL\_SYMBOL\_ALPHA\_UNDERBAR = ""
- APL FUNCTIONAL SYMBOL EPSILON UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_IOTA\_UNDERBAR = ""
- APL\_FUNCTIONAL\_SYMBOL\_OMEGA\_UNDERBAR = ""
- APL FUNCTIONAL SYMBOL ALPHA = ""
- NOT\_CHECK\_MARK = ""
- RIGHT\_ANGLE\_WITH\_DOWNWARDS\_ZIGZAG\_ARROW = ""
- SHOULDERED\_OPEN\_BOX = ""
- BELL\_SYMBOL = ""
- VERTICAL LINE WITH MIDDLE DOT = ""
- INSERTION\_SYMBOL = ""
- CONTINUOUS\_UNDERLINE\_SYMBOL = ""
- DISCONTINUOUS UNDERLINE SYMBOL = ""
- EMPHASIS\_SYMBOL = ""
- COMPOSITION\_SYMBOL = ""
- WHITE\_SQUARE\_WITH\_CENTRE\_VERTICAL\_LINE = ""
- ENTER\_SYMBOL = ""
- ALTERNATIVE\_KEY\_SYMBOL = ""
- HELM\_SYMBOL = ""
- CIRCLED\_HORIZONTAL\_BAR\_WITH\_NOTCH = ""
- CIRCLED\_TRIANGLE\_DOWN = ""

- BROKEN\_CIRCLE\_WITH\_NORTHWEST\_ARROW = ""
- UNDO\_SYMBOL = ""
- MONOSTABLE\_SYMBOL = ""
- HYSTERESIS\_SYMBOL = ""
- OPEN CIRCUIT OUTPUT H TYPE SYMBOL = ""
- OPEN CIRCUIT OUTPUT L TYPE SYMBOL = ""
- PASSIVE\_PULL\_DOWN\_OUTPUT\_SYMBOL = ""
- PASSIVE\_PULL\_UP\_OUTPUT\_SYMBOL = ""
- DIRECT\_CURRENT\_SYMBOL\_FORM\_TWO = ""
- SOFTWARE\_FUNCTION\_SYMBOL = ""
- APL\_FUNCTIONAL\_SYMBOL\_QUAD = ""
- DECIMAL\_SEPARATOR\_KEY\_SYMBOL = ""
- PREVIOUS\_PAGE = ""
- NEXT PAGE = ""
- PRINT\_SCREEN\_SYMBOL = ""
- CLEAR SCREEN SYMBOL = ""
- LEFT PARENTHESIS UPPER HOOK = ""
- LEFT\_PARENTHESIS\_EXTENSION = ""
- LEFT\_PARENTHESIS\_LOWER\_HOOK = ""
- RIGHT PARENTHESIS UPPER HOOK = ""
- RIGHT\_PARENTHESIS\_EXTENSION = ""
- RIGHT\_PARENTHESIS\_LOWER\_HOOK = ""
- LEFT\_SQUARE\_BRACKET\_UPPER\_CORNER = ""
- LEFT\_SQUARE\_BRACKET\_EXTENSION = ""
- LEFT SQUARE BRACKET LOWER CORNER = ""
- RIGHT\_SQUARE\_BRACKET\_UPPER\_CORNER = ""
- RIGHT\_SQUARE\_BRACKET\_EXTENSION = ""
- RIGHT SQUARE BRACKET LOWER CORNER = ""
- LEFT\_CURLY\_BRACKET\_UPPER\_HOOK = ""
- LEFT\_CURLY\_BRACKET\_MIDDLE\_PIECE = ""
- LEFT\_CURLY\_BRACKET\_LOWER\_HOOK = ""
- CURLY\_BRACKET\_EXTENSION = ""
- RIGHT\_CURLY\_BRACKET\_UPPER\_HOOK = ""
- RIGHT\_CURLY\_BRACKET\_MIDDLE\_PIECE = ""
- RIGHT\_CURLY\_BRACKET\_LOWER\_HOOK = ""
- INTEGRAL EXTENSION = ""

- HORIZONTAL LINE EXTENSION = ""
- UPPER\_LEFT\_OR\_LOWER\_RIGHT\_CURLY\_BRACKET\_SECTION = ""
- UPPER\_RIGHT\_OR\_LOWER\_LEFT\_CURLY\_BRACKET\_SECTION = ""
- SUMMATION\_TOP = ""
- SUMMATION\_BOTTOM = ""
- TOP SQUARE BRACKET = ""
- BOTTOM\_SQUARE\_BRACKET = ""
- BOTTOM\_SQUARE\_BRACKET\_OVER\_TOP\_SQUARE\_BRACKET = ""
- RADICAL SYMBOL BOTTOM = ""
- LEFT\_VERTICAL\_BOX\_LINE = ""
- RIGHT\_VERTICAL\_BOX\_LINE = ""
- HORIZONTAL\_SCAN\_LINE\_1 = ""
- HORIZONTAL\_SCAN\_LINE\_3 = ""
- HORIZONTAL SCAN LINE 7 = ""
- HORIZONTAL SCAN LINE 9 = " "
- DENTISTRY SYMBOL LIGHT VERTICAL AND TOP RIGHT = ""
- DENTISTRY SYMBOL LIGHT VERTICAL AND BOTTOM RIGHT = ""
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_WITH\_CIRCLE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTAL\_WITH\_CIRCLE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_WITH\_CIRCLE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_WITH\_TRIANGLE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTAL\_WITH\_TRIANGLE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_WITH\_TRIANGLE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_WAVE = ""
- DENTISTRY SYMBOL LIGHT DOWN AND HORIZONTAL WITH WAVE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_WITH\_WAVE = ""
- DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTAL = ""
- DENTISTRY SYMBOL LIGHT UP AND HORIZONTAL = ""
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_TOP\_LEFT = ""
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_BOTTOM\_LEFT = ""
- SQUARE\_FOOT = ""
- RETURN\_SYMBOL = ""
- EJECT\_SYMBOL = ""
- VERTICAL\_LINE\_EXTENSION = ""
- METRICAL BREVE = ""
- METRICAL LONG OVER SHORT = ""

- METRICAL SHORT OVER LONG = ""
- METRICAL\_LONG\_OVER\_TWO\_SHORTS = ""
- METRICAL\_TWO\_SHORTS\_OVER\_LONG = ""
- METRICAL\_TWO\_SHORTS\_JOINED = ""
- METRICAL TRISEME = ""
- METRICAL TETRASEME = ""
- METRICAL\_PENTASEME = ""
- EARTH\_GROUND = ""
- FUSE = ""
- TOP\_PARENTHESIS = ""
- BOTTOM\_PARENTHESIS = ""
- TOP\_CURLY\_BRACKET = ""
- BOTTOM\_CURLY\_BRACKET = ""
- TOP TORTOISE SHELL BRACKET = ""
- BOTTOM\_TORTOISE\_SHELL\_BRACKET = ""
- WHITE TRAPEZIUM = ""
- BENZENE RING WITH CIRCLE = ""
- STRAIGHTNESS = ""
- FLATNESS = ""
- AC CURRENT = ""
- ELECTRICAL\_INTERSECTION = ""
- DECIMAL\_EXPONENT\_SYMBOL = ""
- BLACK\_RIGHT\_POINTING\_DOUBLE\_TRIANGLE = ""
- BLACK\_LEFT\_POINTING\_DOUBLE\_TRIANGLE = ""
- BLACK\_UP\_POINTING\_DOUBLE\_TRIANGLE = ""
- BLACK\_DOWN\_POINTING\_DOUBLE\_TRIANGLE = ""
- BLACK\_RIGHT\_POINTING\_DOUBLE\_TRIANGLE\_WITH\_VERTICAL\_BAR = ""
- BLACK LEFT POINTING DOUBLE TRIANGLE WITH VERTICAL BAR = ""
- BLACK\_RIGHT\_POINTING\_TRIANGLE\_WITH\_DOUBLE\_VERTICAL\_BAR = ""
- ALARM\_CLOCK = ""
- STOPWATCH = ""
- TIMER CLOCK = ""
- HOURGLASS\_WITH\_FLOWING\_SAND = ""
- BLACK\_MEDIUM\_LEFT\_POINTING\_TRIANGLE = ""
- BLACK MEDIUM RIGHT POINTING TRIANGLE = ""
- BLACK\_MEDIUM\_UP\_POINTING\_TRIANGLE = ""

- BLACK\_MEDIUM\_DOWN\_POINTING\_TRIANGLE = ""
- DOUBLE\_VERTICAL\_BAR = ""
- BLACK\_SQUARE\_FOR\_STOP = ""
- BLACK\_CIRCLE\_FOR\_RECORD = ""
- POWER\_SYMBOL = ""
- POWER\_ON\_OFF\_SYMBOL = ""
- POWER\_ON\_SYMBOL = ""
- POWER\_SLEEP\_SYMBOL = ""
- OBSERVER\_EYE\_SYMBOL = ""

## **Attributes**

DIAMETER_SIGN	
ELECTRIC_ARROW	
HOUSE	
UP_ARROWHEAD	
DOWN_ARROWHEAD	
PROJECTIVE	
PERSPECTIVE	
WAVY_LINE	
LEFT_CEILING	
RIGHT_CEILING	
LEFT_FLOOR	
RIGHT_FLOOR	
BOTTOM_RIGHT_CROP	
BOTTOM_LEFT_CROP	
TOP_RIGHT_CROP	
TOP_LEFT_CROP	
REVERSED_NOT_SIGN	
	continues on poyt page

continues on next page

		_		_	
Tahla	4 _	continue	d fr∩m	nrevious	nage

rable 4 – continued from previous page	
SQUARE_LOZENGE	
ARC	
SEGMENT	
SECTOR	
TELEPHONE_RECORDER	
POSITION_INDICATOR	
VIEWDATA_SQUARE	
PLACE_OF_INTEREST_SIGN	
TURNED_NOT_SIGN	
WATCH	
HOURGLASS	
TOP_LEFT_CORNER	
TOP_RIGHT_CORNER	
BOTTOM_LEFT_CORNER	
BOTTOM_RIGHT_CORNER	
TOP_HALF_INTEGRAL	
BOTTOM_HALF_INTEGRAL	
FROWN	
SMILE	
UP_ARROWHEAD_BETWEEN_TWO_HORIZONTAL_BARS	
OPTION_KEY	
ERASE_TO_THE_RIGHT	
X_IN_A_RECTANGLE_BOX	
KEYBOARD	
LEFT_POINTING_ANGLE_BRACKET	
RIGHT_POINTING_ANGLE_BRACKET	
	continues on next page

Table 4 – continued from previous page

ERASE\_TO\_THE\_LEFT BENZENE\_RING CYLINDRICITY ALL\_AROUND\_PROFILE **SYMMETRY** TOTAL\_RUNOUT DIMENSION\_ORIGIN CONICAL\_TAPER SLOPE **COUNTERBORE COUNTERSINK** APL\_FUNCTIONAL\_SYMBOL\_I\_BEAM  $APL\_FUNCTIONAL\_SYMBOL\_SQUISH\_QUAD$ APL\_FUNCTIONAL\_SYMBOL\_QUAD\_EQUAL  $APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DIVIDE$  $APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DIAMOND$ APL\_FUNCTIONAL\_SYMBOL\_QUAD\_JOT APL\_FUNCTIONAL\_SYMBOL\_QUAD\_CIRCLE  $APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_STILE$ APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_JOT  $APL\_FUNCTIONAL\_SYMBOL\_SLASH\_BAR$ APL\_FUNCTIONAL\_SYMBOL\_BACKSLASH\_BAR APL\_FUNCTIONAL\_SYMBOL\_QUAD\_SLASH  $APL\_FUNCTIONAL\_SYMBOL\_QUAD\_BACKSLASH$ APL\_FUNCTIONAL\_SYMBOL\_QUAD\_LESS\_THAN APL\_FUNCTIONAL\_SYMBOL\_QUAD\_GREATER\_THAN

continues on next page

Table 4 – continued from previous page

 $APL\_FUNCTIONAL\_SYMBOL\_LEFTWARDS\_VANE$ 

APL\_FUNCTIONAL\_SYMBOL\_RIGHTWARDS\_VANE

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_LEFTWARDS\_ARRO

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_RIGHTWARDS\_ARR

APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_BACKSLASH

APL\_FUNCTIONAL\_SYMBOL\_DOWN\_TACK\_UNDERBAR

APL\_FUNCTIONAL\_SYMBOL\_DELTA\_STILE

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DOWN\_CARET

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DELTA

APL\_FUNCTIONAL\_SYMBOL\_DOWN\_TACK\_JOT

APL\_FUNCTIONAL\_SYMBOL\_UPWARDS\_VANE

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_UPWARDS\_ARROW

APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_OVERBAR

APL\_FUNCTIONAL\_SYMBOL\_DEL\_STILE

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_UP\_CARET

 $APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DEL$ 

APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_JOT

APL\_FUNCTIONAL\_SYMBOL\_DOWNWARDS\_VANE

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DOWNWARDS\_ARRO

APL\_FUNCTIONAL\_SYMBOL\_QUOTE\_UNDERBAR

 $APL\_FUNCTIONAL\_SYMBOL\_DELTA\_UNDERBAR$ 

APL\_FUNCTIONAL\_SYMBOL\_DIAMOND\_UNDERBAR

APL\_FUNCTIONAL\_SYMBOL\_JOT\_UNDERBAR

 $APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_UNDERBAR$ 

 $APL\_FUNCTIONAL\_SYMBOL\_UP\_SHOE\_JOT$ 

APL\_FUNCTIONAL\_SYMBOL\_QUOTE\_QUAD

continues on next page

Table 4 – continued from previous page

 $APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_STAR$ 

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_COLON

APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_DIAERESIS

APL\_FUNCTIONAL\_SYMBOL\_DEL\_DIAERESIS

APL\_FUNCTIONAL\_SYMBOL\_STAR\_DIAERESIS

APL\_FUNCTIONAL\_SYMBOL\_JOT\_DIAERESIS

APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_DIAERESIS

APL\_FUNCTIONAL\_SYMBOL\_DOWN\_SHOE\_STILE

APL\_FUNCTIONAL\_SYMBOL\_LEFT\_SHOE\_STILE

APL\_FUNCTIONAL\_SYMBOL\_TILDE\_DIAERESIS

APL\_FUNCTIONAL\_SYMBOL\_GREATER\_THAN\_DIAERE.

APL\_FUNCTIONAL\_SYMBOL\_COMMA\_BAR

APL\_FUNCTIONAL\_SYMBOL\_DEL\_TILDE

APL\_FUNCTIONAL\_SYMBOL\_ZILDE

APL\_FUNCTIONAL\_SYMBOL\_STILE\_TILDE

APL\_FUNCTIONAL\_SYMBOL\_SEMICOLON\_UNDERBAR

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_NOT\_EQUAL

APL\_FUNCTIONAL\_SYMBOL\_QUAD\_QUESTION

APL\_FUNCTIONAL\_SYMBOL\_DOWN\_CARET\_TILDE

 $APL\_FUNCTIONAL\_SYMBOL\_UP\_CARET\_TILDE$ 

APL\_FUNCTIONAL\_SYMBOL\_IOTA

 $APL\_FUNCTIONAL\_SYMBOL\_RHO$ 

APL\_FUNCTIONAL\_SYMBOL\_OMEGA

 $APL\_FUNCTIONAL\_SYMBOL\_ALPHA\_UNDERBAR$ 

APL\_FUNCTIONAL\_SYMBOL\_EPSILON\_UNDERBAR

APL\_FUNCTIONAL\_SYMBOL\_IOTA\_UNDERBAR

continues on next page

Table 4 – continued from previous page

APL\_FUNCTIONAL\_SYMBOL\_OMEGA\_UNDERBAR

APL\_FUNCTIONAL\_SYMBOL\_ALPHA

NOT\_CHECK\_MARK

RIGHT\_ANGLE\_WITH\_DOWNWARDS\_ZIGZAG\_ARROW

SHOULDERED\_OPEN\_BOX

BELL\_SYMBOL

VERTICAL\_LINE\_WITH\_MIDDLE\_DOT

INSERTION\_SYMBOL

CONTINUOUS\_UNDERLINE\_SYMBOL

DISCONTINUOUS\_UNDERLINE\_SYMBOL

EMPHASIS\_SYMBOL

COMPOSITION\_SYMBOL

WHITE\_SQUARE\_WITH\_CENTRE\_VERTICAL\_LINE

ENTER\_SYMBOL

ALTERNATIVE\_KEY\_SYMBOL

HELM\_SYMBOL

CIRCLED\_HORIZONTAL\_BAR\_WITH\_NOTCH

CIRCLED\_TRIANGLE\_DOWN

BROKEN\_CIRCLE\_WITH\_NORTHWEST\_ARROW

UNDO\_SYMBOL

 ${\tt MONOSTABLE\_SYMBOL}$ 

 ${\it HYSTERESIS\_SYMBOL}$ 

OPEN\_CIRCUIT\_OUTPUT\_H\_TYPE\_SYMBOL

OPEN\_CIRCUIT\_OUTPUT\_L\_TYPE\_SYMBOL

 ${\it PASSIVE\_PULL\_DOWN\_OUTPUT\_SYMBOL}$ 

 ${\it PASSIVE\_PULL\_UP\_OUTPUT\_SYMBOL}$ 

continues on next page

Table 4 – continued from previous page

DIRECT\_CURRENT\_SYMBOL\_FORM\_TWO

SOFTWARE\_FUNCTION\_SYMBOL

 $APL\_FUNCTIONAL\_SYMBOL\_QUAD$ 

DECIMAL\_SEPARATOR\_KEY\_SYMBOL

PREVIOUS\_PAGE

NEXT\_PAGE

PRINT\_SCREEN\_SYMBOL

CLEAR\_SCREEN\_SYMBOL

LEFT\_PARENTHESIS\_UPPER\_HOOK

LEFT\_PARENTHESIS\_EXTENSION

LEFT\_PARENTHESIS\_LOWER\_HOOK

RIGHT\_PARENTHESIS\_UPPER\_HOOK

RIGHT\_PARENTHESIS\_EXTENSION

RIGHT\_PARENTHESIS\_LOWER\_HOOK

LEFT\_SQUARE\_BRACKET\_UPPER\_CORNER

LEFT\_SQUARE\_BRACKET\_EXTENSION

LEFT\_SQUARE\_BRACKET\_LOWER\_CORNER

RIGHT\_SQUARE\_BRACKET\_UPPER\_CORNER

RIGHT\_SQUARE\_BRACKET\_EXTENSION

RIGHT\_SQUARE\_BRACKET\_LOWER\_CORNER

 $LEFT\_CURLY\_BRACKET\_UPPER\_HOOK$ 

LEFT\_CURLY\_BRACKET\_MIDDLE\_PIECE

LEFT\_CURLY\_BRACKET\_LOWER\_HOOK

CURLY\_BRACKET\_EXTENSION

 $RIGHT\_CURLY\_BRACKET\_UPPER\_HOOK$ 

RIGHT\_CURLY\_BRACKET\_MIDDLE\_PIECE

continues on next page

Table 4 – continued from previous page

RIGHT\_CURLY\_BRACKET\_LOWER\_HOOK INTEGRAL\_EXTENSION HORIZONTAL\_LINE\_EXTENSION UPPER\_LEFT\_OR\_LOWER\_RIGHT\_CURLY\_BRACKET\_S. UPPER\_RIGHT\_OR\_LOWER\_LEFT\_CURLY\_BRACKET\_S. SUMMATION\_TOP SUMMATION\_BOTTOM TOP\_SQUARE\_BRACKET BOTTOM\_SQUARE\_BRACKET BOTTOM\_SQUARE\_BRACKET\_OVER\_TOP\_SQUARE\_BRA RADICAL\_SYMBOL\_BOTTOM LEFT\_VERTICAL\_BOX\_LINE RIGHT\_VERTICAL\_BOX\_LINE HORIZONTAL\_SCAN\_LINE\_1 HORIZONTAL\_SCAN\_LINE\_3 HORIZONTAL\_SCAN\_LINE\_7 HORIZONTAL SCAN LINE 9 DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_TOP\_R DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_BOTTO. DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_WITH\_CIRC. DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTA. DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_ DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_WITH\_TRIA. DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTA. DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_ DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_WAVE

continues on next page

Table 4 – continued from previous page

DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTA.

DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_

DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTA.

DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL

DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_TOP\_L.

DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_BOTTO.

SQUARE\_FOOT

RETURN\_SYMBOL

EJECT\_SYMBOL

VERTICAL\_LINE\_EXTENSION

METRICAL\_BREVE

METRICAL\_LONG\_OVER\_SHORT

METRICAL\_SHORT\_OVER\_LONG

METRICAL\_LONG\_OVER\_TWO\_SHORTS

METRICAL\_TWO\_SHORTS\_OVER\_LONG

METRICAL\_TWO\_SHORTS\_JOINED

METRICAL TRISEME

METRICAL\_TETRASEME

METRICAL\_PENTASEME

EARTH\_GROUND

FUSE

 $TOP\_PARENTHESIS$ 

BOTTOM\_PARENTHESIS

TOP\_CURLY\_BRACKET

 ${\it BOTTOM\_CURLY\_BRACKET}$ 

TOP\_TORTOISE\_SHELL\_BRACKET

continues on next page

Table 4 – continued from previous page

BOTTOM\_TORTOISE\_SHELL\_BRACKET

WHITE\_TRAPEZIUM

BENZENE\_RING\_WITH\_CIRCLE

STRAIGHTNESS

**FLATNESS** 

AC\_CURRENT

ELECTRICAL\_INTERSECTION

DECIMAL\_EXPONENT\_SYMBOL

BLACK\_RIGHT\_POINTING\_DOUBLE\_TRIANGLE

BLACK\_LEFT\_POINTING\_DOUBLE\_TRIANGLE

BLACK\_UP\_POINTING\_DOUBLE\_TRIANGLE

BLACK\_DOWN\_POINTING\_DOUBLE\_TRIANGLE

BLACK\_RIGHT\_POINTING\_DOUBLE\_TRIANGLE\_WITH

BLACK\_LEFT\_POINTING\_DOUBLE\_TRIANGLE\_WITH\_

BLACK\_RIGHT\_POINTING\_TRIANGLE\_WITH\_DOUBLE.

ALARM\_CLOCK

STOPWATCH

TIMER CLOCK

HOURGLASS\_WITH\_FLOWING\_SAND

 $BLACK\_MEDIUM\_LEFT\_POINTING\_TRIANGLE$ 

BLACK\_MEDIUM\_RIGHT\_POINTING\_TRIANGLE

 ${\it BLACK\_MEDIUM\_UP\_POINTING\_TRIANGLE}$ 

BLACK\_MEDIUM\_DOWN\_POINTING\_TRIANGLE

DOUBLE\_VERTICAL\_BAR

 $BLACK\_SQUARE\_FOR\_STOP$ 

 $BLACK\_CIRCLE\_FOR\_RECORD$ 

continues on next page

Table 4 – continued from previous page

```
POWER_SYMBOL

POWER_ON_OFF_SYMBOL

POWER_ON_SYMBOL

POWER_SLEEP_SYMBOL

OBSERVER_EYE_SYMBOL
```

```
AC_CURRENT = ''
ALARM_CLOCK = ''
ALL_AROUND_PROFILE = ''
ALTERNATIVE_KEY_SYMBOL = ''
APL_FUNCTIONAL_SYMBOL_ALPHA = ''
APL_FUNCTIONAL_SYMBOL_ALPHA_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_BACKSLASH_BAR = ''
APL_FUNCTIONAL_SYMBOL_CIRCLE_BACKSLASH = ''
APL_FUNCTIONAL_SYMBOL_CIRCLE_DIAERESIS = ''
APL_FUNCTIONAL_SYMBOL_CIRCLE_JOT = ''
APL_FUNCTIONAL_SYMBOL_CIRCLE_STAR = ''
APL_FUNCTIONAL_SYMBOL_CIRCLE_STILE = ''
APL_FUNCTIONAL_SYMBOL_CIRCLE_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_COMMA_BAR = ''
APL_FUNCTIONAL_SYMBOL_DELTA_STILE = ''
APL_FUNCTIONAL_SYMBOL_DELTA_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_DEL_DIAERESIS = ''
APL_FUNCTIONAL_SYMBOL_DEL_STILE = ''
APL_FUNCTIONAL_SYMBOL_DEL_TILDE = ''
APL_FUNCTIONAL_SYMBOL_DIAMOND_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_DOWNWARDS_VANE = ''
APL_FUNCTIONAL_SYMBOL_DOWN_CARET_TILDE = ''
APL_FUNCTIONAL_SYMBOL_DOWN_SHOE_STILE = ''
```

```
APL_FUNCTIONAL_SYMBOL_DOWN_TACK_JOT = ''
APL_FUNCTIONAL_SYMBOL_DOWN_TACK_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_EPSILON_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_GREATER_THAN_DIAERESIS = ''
APL_FUNCTIONAL_SYMBOL_IOTA = ''
APL_FUNCTIONAL_SYMBOL_IOTA_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_I_BEAM = ''
APL_FUNCTIONAL_SYMBOL_JOT_DIAERESIS = ''
APL_FUNCTIONAL_SYMBOL_JOT_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_LEFTWARDS_VANE = ''
APL_FUNCTIONAL_SYMBOL_LEFT_SHOE_STILE = ''
APL_FUNCTIONAL_SYMBOL_OMEGA = ''
APL_FUNCTIONAL_SYMBOL_OMEGA_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_QUAD = ''
APL_FUNCTIONAL_SYMBOL_QUAD_BACKSLASH = ''
APL_FUNCTIONAL_SYMBOL_QUAD_CIRCLE = ''
APL_FUNCTIONAL_SYMBOL_QUAD_COLON = ''
APL_FUNCTIONAL_SYMBOL_QUAD_DEL = ''
APL_FUNCTIONAL_SYMBOL_QUAD_DELTA = ''
APL_FUNCTIONAL_SYMBOL_QUAD_DIAMOND = ''
APL_FUNCTIONAL_SYMBOL_QUAD_DIVIDE = ''
APL_FUNCTIONAL_SYMBOL_QUAD_DOWNWARDS_ARROW = ''
APL_FUNCTIONAL_SYMBOL_QUAD_DOWN_CARET = ''
APL_FUNCTIONAL_SYMBOL_QUAD_EQUAL = ''
APL_FUNCTIONAL_SYMBOL_QUAD_GREATER_THAN = ''
APL_FUNCTIONAL_SYMBOL_QUAD_JOT = ''
APL_FUNCTIONAL_SYMBOL_QUAD_LEFTWARDS_ARROW = ''
APL_FUNCTIONAL_SYMBOL_QUAD_LESS_THAN = ''
APL_FUNCTIONAL_SYMBOL_QUAD_NOT_EQUAL = ''
APL_FUNCTIONAL_SYMBOL_QUAD_QUESTION = ''
APL_FUNCTIONAL_SYMBOL_QUAD_RIGHTWARDS_ARROW = ''
```

```
APL_FUNCTIONAL_SYMBOL_QUAD_SLASH = ''
APL_FUNCTIONAL_SYMBOL_QUAD_UPWARDS_ARROW = ''
APL_FUNCTIONAL_SYMBOL_QUAD_UP_CARET = ''
APL_FUNCTIONAL_SYMBOL_QUOTE_QUAD = ''
APL_FUNCTIONAL_SYMBOL_QUOTE_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_RHO = ''
APL_FUNCTIONAL_SYMBOL_RIGHTWARDS_VANE = ''
APL_FUNCTIONAL_SYMBOL_SEMICOLON_UNDERBAR = ''
APL_FUNCTIONAL_SYMBOL_SLASH_BAR = ''
APL_FUNCTIONAL_SYMBOL_SQUISH_QUAD = ''
APL_FUNCTIONAL_SYMBOL_STAR_DIAERESIS = ''
APL_FUNCTIONAL_SYMBOL_STILE_TILDE = ''
APL_FUNCTIONAL_SYMBOL_TILDE_DIAERESIS = ''
APL_FUNCTIONAL_SYMBOL_UPWARDS_VANE = ''
APL_FUNCTIONAL_SYMBOL_UP_CARET_TILDE = ''
APL_FUNCTIONAL_SYMBOL_UP_SHOE_JOT = ''
APL_FUNCTIONAL_SYMBOL_UP_TACK_DIAERESIS = ''
APL_FUNCTIONAL_SYMBOL_UP_TACK_JOT = ''
APL_FUNCTIONAL_SYMBOL_UP_TACK_OVERBAR = ''
APL_FUNCTIONAL_SYMBOL_ZILDE = ''
ARC = ''
BELL_SYMBOL = ''
BENZENE_RING = ''
BENZENE_RING_WITH_CIRCLE = ''
BLACK_CIRCLE_FOR_RECORD = ''
BLACK_DOWN_POINTING_DOUBLE_TRIANGLE = ''
BLACK_LEFT_POINTING_DOUBLE_TRIANGLE = ''
BLACK_LEFT_POINTING_DOUBLE_TRIANGLE_WITH_VERTICAL_BAR = ''
BLACK_MEDIUM_DOWN_POINTING_TRIANGLE = ''
BLACK_MEDIUM_LEFT_POINTING_TRIANGLE = ''
BLACK_MEDIUM_RIGHT_POINTING_TRIANGLE = ''
```

```
BLACK_MEDIUM_UP_POINTING_TRIANGLE = ''
BLACK_RIGHT_POINTING_DOUBLE_TRIANGLE = ''
BLACK_RIGHT_POINTING_DOUBLE_TRIANGLE_WITH_VERTICAL_BAR = ''
BLACK_RIGHT_POINTING_TRIANGLE_WITH_DOUBLE_VERTICAL_BAR = ''
BLACK_SQUARE_FOR_STOP = ''
BLACK_UP_POINTING_DOUBLE_TRIANGLE = ''
BOTTOM_CURLY_BRACKET = ''
BOTTOM_HALF_INTEGRAL = ''
BOTTOM_LEFT_CORNER = ''
BOTTOM_LEFT_CROP = ''
BOTTOM_PARENTHESIS = ''
BOTTOM_RIGHT_CORNER = ''
BOTTOM_RIGHT_CROP = ''
BOTTOM_SQUARE_BRACKET = ''
BOTTOM_SQUARE_BRACKET_OVER_TOP_SQUARE_BRACKET = ''
BOTTOM_TORTOISE_SHELL_BRACKET = ''
BROKEN_CIRCLE_WITH_NORTHWEST_ARROW = ''
CIRCLED_HORIZONTAL_BAR_WITH_NOTCH = ''
CIRCLED_TRIANGLE_DOWN = ''
CLEAR_SCREEN_SYMBOL = ''
COMPOSITION_SYMBOL = ''
CONICAL_TAPER = ''
CONTINUOUS_UNDERLINE_SYMBOL = ''
COUNTERBORE = ''
COUNTERSINK = ''
CURLY_BRACKET_EXTENSION = ''
CYLINDRICITY = ''
DECIMAL_EXPONENT_SYMBOL = ''
DECIMAL_SEPARATOR_KEY_SYMBOL = ''
DENTISTRY_SYMBOL_LIGHT_DOWN_AND_HORIZONTAL = ''
DENTISTRY_SYMBOL_LIGHT_DOWN_AND_HORIZONTAL_WITH_CIRCLE = ''
```

```
DENTISTRY_SYMBOL_LIGHT_DOWN_AND_HORIZONTAL_WITH_TRIANGLE = ''
DENTISTRY_SYMBOL_LIGHT_DOWN_AND_HORIZONTAL_WITH_WAVE = ''
DENTISTRY_SYMBOL_LIGHT_UP_AND_HORIZONTAL = ''
DENTISTRY_SYMBOL_LIGHT_UP_AND_HORIZONTAL_WITH_CIRCLE = ''
DENTISTRY_SYMBOL_LIGHT_UP_AND_HORIZONTAL_WITH_TRIANGLE = ''
DENTISTRY_SYMBOL_LIGHT_UP_AND_HORIZONTAL_WITH_WAVE = ''
DENTISTRY_SYMBOL_LIGHT_VERTICAL_AND_BOTTOM_LEFT = ''
DENTISTRY_SYMBOL_LIGHT_VERTICAL_AND_BOTTOM_RIGHT = ''
DENTISTRY_SYMBOL_LIGHT_VERTICAL_AND_TOP_LEFT = ''
DENTISTRY_SYMBOL_LIGHT_VERTICAL_AND_TOP_RIGHT = ''
DENTISTRY_SYMBOL_LIGHT_VERTICAL_AND_WAVE = ''
DENTISTRY_SYMBOL_LIGHT_VERTICAL_WITH_CIRCLE = ''
DENTISTRY_SYMBOL_LIGHT_VERTICAL_WITH_TRIANGLE = ''
DIAMETER_SIGN = ''
DIMENSION_ORIGIN = ''
DIRECT_CURRENT_SYMBOL_FORM_TWO = ''
DISCONTINUOUS_UNDERLINE_SYMBOL = ''
DOUBLE_VERTICAL_BAR = ''
DOWN_ARROWHEAD = ''
EARTH_GROUND = ''
EJECT_SYMBOL = ''
ELECTRICAL_INTERSECTION = ''
ELECTRIC_ARROW = ''
EMPHASIS_SYMBOL = ''
ENTER_SYMBOL = ''
ERASE_TO_THE_LEFT = ''
ERASE_TO_THE_RIGHT = ''
FLATNESS = ''
FROWN = ''
FUSE = ''
HELM_SYMBOL = ''
```

```
HORIZONTAL_LINE_EXTENSION = ''
HORIZONTAL_SCAN_LINE_1 = ''
HORIZONTAL_SCAN_LINE_3 = ''
HORIZONTAL_SCAN_LINE_7 = ''
HORIZONTAL_SCAN_LINE_9 = '_'
HOURGLASS = ''
HOURGLASS_WITH_FLOWING_SAND = ''
HOUSE = ''
HYSTERESIS_SYMBOL = ''
INSERTION_SYMBOL = ''
INTEGRAL_EXTENSION = ''
KEYBOARD = ''
LEFT_CEILING = ''
LEFT_CURLY_BRACKET_LOWER_HOOK = ''
LEFT_CURLY_BRACKET_MIDDLE_PIECE = ''
LEFT_CURLY_BRACKET_UPPER_HOOK = ''
LEFT_FLOOR = ''
LEFT_PARENTHESIS_EXTENSION = ''
LEFT_PARENTHESIS_LOWER_HOOK = ''
LEFT_PARENTHESIS_UPPER_HOOK = ''
LEFT_POINTING_ANGLE_BRACKET = '('
LEFT_SQUARE_BRACKET_EXTENSION = ''
LEFT_SQUARE_BRACKET_LOWER_CORNER = ''
LEFT_SQUARE_BRACKET_UPPER_CORNER = ''
LEFT_VERTICAL_BOX_LINE = ''
METRICAL_BREVE = ''
METRICAL_LONG_OVER_SHORT = ''
METRICAL_LONG_OVER_TWO_SHORTS = ''
METRICAL_PENTASEME = ''
METRICAL_SHORT_OVER_LONG = ''
METRICAL_TETRASEME = ''
```

```
METRICAL_TRISEME = ''
METRICAL_TWO_SHORTS_JOINED = ''
METRICAL_TWO_SHORTS_OVER_LONG = ''
MONOSTABLE_SYMBOL = ''
NEXT_PAGE = ''
NOT_CHECK_MARK = ''
OBSERVER_EYE_SYMBOL = ''
OPEN_CIRCUIT_OUTPUT_H_TYPE_SYMBOL = ''
OPEN_CIRCUIT_OUTPUT_L_TYPE_SYMBOL = ''
OPTION_KEY = ''
PASSIVE_PULL_DOWN_OUTPUT_SYMBOL = ''
PASSIVE_PULL_UP_OUTPUT_SYMBOL = ''
PERSPECTIVE = ''
PLACE_OF_INTEREST_SIGN = ''
POSITION_INDICATOR = ''
POWER_ON_OFF_SYMBOL = ''
POWER_ON_SYMBOL = ''
POWER_SLEEP_SYMBOL = ''
POWER_SYMBOL = ''
PREVIOUS_PAGE = ''
PRINT_SCREEN_SYMBOL = ''
PROJECTIVE = ''
RADICAL_SYMBOL_BOTTOM = ''
RETURN_SYMBOL = ''
REVERSED_NOT_SIGN = ''
RIGHT_ANGLE_WITH_DOWNWARDS_ZIGZAG_ARROW = ''
RIGHT_CEILING = ''
RIGHT_CURLY_BRACKET_LOWER_HOOK = ''
RIGHT_CURLY_BRACKET_MIDDLE_PIECE = ''
RIGHT_CURLY_BRACKET_UPPER_HOOK = ''
RIGHT_FLOOR = ''
```

```
RIGHT_PARENTHESIS_EXTENSION = ''
RIGHT_PARENTHESIS_LOWER_HOOK = ''
RIGHT_PARENTHESIS_UPPER_HOOK = ''
RIGHT_POINTING_ANGLE_BRACKET = '>'
RIGHT_SQUARE_BRACKET_EXTENSION = ''
RIGHT_SQUARE_BRACKET_LOWER_CORNER = ''
RIGHT_SQUARE_BRACKET_UPPER_CORNER = ''
RIGHT_VERTICAL_BOX_LINE = ''
SECTOR = ''
SEGMENT = ''
SHOULDERED_OPEN_BOX = ''
SLOPE = ''
SMILE = ''
SOFTWARE_FUNCTION_SYMBOL = ''
SQUARE_FOOT = ''
SQUARE_LOZENGE = ''
STOPWATCH = ''
STRAIGHTNESS = ''
SUMMATION_BOTTOM = ''
SUMMATION_TOP = ''
SYMMETRY = ''
TELEPHONE_RECORDER = ''
TIMER_CLOCK = ''
TOP_CURLY_BRACKET = ''
TOP_HALF_INTEGRAL = ''
TOP_LEFT_CORNER = ''
TOP_LEFT_CROP = ''
TOP_PARENTHESIS = ''
TOP_RIGHT_CORNER = ''
TOP_RIGHT_CROP = ''
TOP_SQUARE_BRACKET = ''
```

```
TOP_TORTOISE_SHELL_BRACKET = ''
TOTAL_RUNOUT = ''
TURNED_NOT_SIGN = ''
UNDO_SYMBOL = ''
UPPER_LEFT_OR_LOWER_RIGHT_CURLY_BRACKET_SECTION = ''
UPPER_RIGHT_OR_LOWER_LEFT_CURLY_BRACKET_SECTION = ''
UP_ARROWHEAD = ''
UP_ARROWHEAD_BETWEEN_TWO_HORIZONTAL_BARS = ''
VERTICAL_LINE_EXTENSION = ''
VERTICAL_LINE_WITH_MIDDLE_DOT = ''
VIEWDATA SOUARE = ''
WATCH = ''
WAVY_LINE = ''
WHITE_SQUARE_WITH_CENTRE_VERTICAL_LINE = ''
WHITE_TRAPEZIUM = ''
X_IN_A_RECTANGLE_BOX = ''
```

## **Models**

class pygamelib.assets.graphics.Models

Bases: object

List of models (emojis by unicode denomination)

Models are filtered emojis. This class does not map the entire specification.

Models replaces the previous Sprites class. Renaming that class is necessary with the introduction of a real Sprite class in the GFX module.

This class contains 1328 emojis (this is not the full list). All emoji codes come from: https://unicode.org/emoji/charts/full\_emoji\_list.html Additional emojis can be added by codes.

The complete list of aliased emojis is:

- GRINNING\_FACE =
- GRINNING\_FACE\_WITH\_BIG\_EYES =
- GRINNING FACE WITH SMILING EYES =
- BEAMING\_FACE\_WITH\_SMILING\_EYES =
- GRINNING\_SQUINTING\_FACE =
- GRINNING\_FACE\_WITH\_SWEAT =
- ROLLING\_ON\_THE\_FLOOR\_LAUGHING =

- FACE\_WITH\_TEARS\_OF\_JOY =
- SLIGHTLY\_SMILING\_FACE =
- UPSIDE\_DOWN\_FACE =
- WINKING\_FACE =
- SMILING\_FACE\_WITH\_SMILING\_EYES =
- SMILING\_FACE\_WITH\_HALO =
- SMILING\_FACE\_WITH\_HEARTS =
- SMILING\_FACE\_WITH\_HEART\_EYES =
- STAR\_STRUCK =
- FACE\_BLOWING\_A\_KISS =
- KISSING\_FACE =
- SMILING\_FACE =
- KISSING\_FACE\_WITH\_CLOSED\_EYES =
- KISSING\_FACE\_WITH\_SMILING\_EYES =
- SMILING\_FACE\_WITH\_TEAR =
- FACE\_SAVORING\_FOOD =
- FACE\_WITH\_TONGUE =
- WINKING\_FACE\_WITH\_TONGUE =
- ZANY\_FACE =
- SQUINTING\_FACE\_WITH\_TONGUE =
- MONEY\_MOUTH\_FACE =
- HUGGING\_FACE =
- FACE\_WITH\_HAND\_OVER\_MOUTH =
- SHUSHING\_FACE =
- THINKING\_FACE =
- ZIPPER\_MOUTH\_FACE =
- FACE\_WITH\_RAISED\_EYEBROW =
- NEUTRAL\_FACE =
- EXPRESSIONLESS\_FACE =
- FACE\_WITHOUT\_MOUTH =
- SMIRKING\_FACE =
- UNAMUSED\_FACE =
- FACE\_WITH\_ROLLING\_EYES =
- GRIMACING\_FACE =
- LYING\_FACE =
- RELIEVED\_FACE =

- PENSIVE\_FACE =
- SLEEPY\_FACE =
- DROOLING\_FACE =
- SLEEPING\_FACE =
- FACE\_WITH\_MEDICAL\_MASK =
- FACE\_WITH\_THERMOMETER =
- FACE\_WITH\_HEAD\_BANDAGE =
- NAUSEATED\_FACE =
- FACE\_VOMITING =
- SNEEZING\_FACE =
- HOT\_FACE =
- COLD\_FACE =
- WOOZY\_FACE =
- DIZZY\_FACE =
- EXPLODING\_HEAD =
- COWBOY\_HAT\_FACE =
- PARTYING\_FACE =
- DISGUISED\_FACE =
- SMILING\_FACE\_WITH\_SUNGLASSES =
- NERD\_FACE =
- FACE\_WITH\_MONOCLE =
- CONFUSED\_FACE =
- WORRIED\_FACE =
- SLIGHTLY\_FROWNING\_FACE =
- FROWNING\_FACE =
- FACE\_WITH\_OPEN\_MOUTH =
- HUSHED\_FACE =
- ASTONISHED\_FACE =
- FLUSHED\_FACE =
- PLEADING\_FACE =
- FROWNING\_FACE\_WITH\_OPEN\_MOUTH =
- ANGUISHED\_FACE =
- FEARFUL\_FACE =
- ANXIOUS\_FACE\_WITH\_SWEAT =
- SAD\_BUT\_RELIEVED\_FACE =
- CRYING\_FACE =

- LOUDLY\_CRYING\_FACE =
- FACE\_SCREAMING\_IN\_FEAR =
- CONFOUNDED\_FACE =
- PERSEVERING\_FACE =
- DISAPPOINTED\_FACE =
- DOWNCAST\_FACE\_WITH\_SWEAT =
- WEARY\_FACE =
- TIRED\_FACE =
- YAWNING\_FACE =
- FACE\_WITH\_STEAM\_FROM\_NOSE =
- POUTING\_FACE =
- ANGRY\_FACE =
- FACE\_WITH\_SYMBOLS\_ON\_MOUTH =
- SMILING\_FACE\_WITH\_HORNS =
- ANGRY\_FACE\_WITH\_HORNS =
- SKULL =
- SKULL\_AND\_CROSSBONES =
- PILE\_OF\_POO =
- CLOWN\_FACE =
- OGRE =
- GOBLIN =
- GHOST =
- ALIEN =
- ALIEN\_MONSTER =
- ROBOT =
- GRINNING\_CAT =
- GRINNING\_CAT\_WITH\_SMILING\_EYES =
- CAT\_WITH\_TEARS\_OF\_JOY =
- SMILING\_CAT\_WITH\_HEART\_EYES =
- CAT\_WITH\_WRY\_SMILE =
- KISSING\_CAT =
- WEARY\_CAT =
- CRYING\_CAT =
- POUTING\_CAT =
- SEE\_NO\_EVIL\_MONKEY =
- HEAR\_NO\_EVIL\_MONKEY =

- SPEAK\_NO\_EVIL\_MONKEY =
- KISS\_MARK =
- LOVE\_LETTER =
- HEART\_WITH\_ARROW =
- HEART\_WITH\_RIBBON =
- SPARKLING\_HEART =
- GROWING\_HEART =
- BEATING\_HEART =
- REVOLVING\_HEARTS =
- TWO\_HEARTS =
- HEART\_DECORATION =
- HEART\_EXCLAMATION =
- BROKEN\_HEART =
- RED\_HEART =
- ORANGE\_HEART =
- YELLOW\_HEART =
- GREEN\_HEART =
- BLUE\_HEART =
- PURPLE\_HEART =
- BROWN\_HEART =
- BLACK\_HEART =
- WHITE\_HEART =
- HUNDRED\_POINTS =
- ANGER\_SYMBOL =
- COLLISION =
- DIZZY =
- SWEAT\_DROPLETS =
- DASHING\_AWAY =
- HOLE =
- BOMB =
- SPEECH\_BALLOON =
- LEFT\_SPEECH\_BUBBLE =
- RIGHT\_ANGER\_BUBBLE =
- THOUGHT\_BALLOON =
- ZZZ =
- WAVING\_HAND =

- RAISED\_BACK\_OF\_HAND =
- HAND\_WITH\_FINGERS\_SPLAYED =
- RAISED\_HAND =
- VULCAN\_SALUTE =
- OK\_HAND =
- PINCHED\_FINGERS =
- PINCHING\_HAND =
- VICTORY\_HAND =
- CROSSED\_FINGERS =
- LOVE\_YOU\_GESTURE =
- SIGN\_OF\_THE\_HORNS =
- CALL\_ME\_HAND =
- BACKHAND\_INDEX\_POINTING\_LEFT =
- BACKHAND\_INDEX\_POINTING\_RIGHT =
- BACKHAND\_INDEX\_POINTING\_UP =
- MIDDLE\_FINGER =
- BACKHAND\_INDEX\_POINTING\_DOWN =
- INDEX\_POINTING\_UP =
- THUMBS\_UP =
- THUMBS\_DOWN =
- RAISED\_FIST =
- ONCOMING\_FIST =
- LEFT\_FACING\_FIST =
- RIGHT\_FACING\_FIST =
- CLAPPING\_HANDS =
- RAISING\_HANDS =
- OPEN\_HANDS =
- PALMS\_UP\_TOGETHER =
- HANDSHAKE =
- FOLDED\_HANDS =
- WRITING\_HAND =
- NAIL\_POLISH =
- SELFIE =
- FLEXED\_BICEPS =
- MECHANICAL ARM =
- MECHANICAL\_LEG =

- LEG =
- FOOT =
- EAR =
- EAR\_WITH\_HEARING\_AID =
- NOSE =
- BRAIN =
- ANATOMICAL\_HEART =
- LUNGS =
- TOOTH =
- BONE =
- EYES =
- EYE =
- TONGUE =
- MOUTH =
- BABY =
- CHILD =
- BOY =
- GIRL =
- PERSON =
- PERSON\_BLOND\_HAIR =
- MAN =
- MAN\_BEARD =
- WOMAN =
- OLDER\_PERSON =
- $OLD_MAN =$
- OLD\_WOMAN =
- PERSON\_FROWNING =
- PERSON\_POUTING =
- PERSON\_GESTURING\_NO =
- PERSON\_GESTURING\_OK =
- PERSON\_TIPPING\_HAND =
- PERSON\_RAISING\_HAND =
- DEAF\_PERSON =
- PERSON\_BOWING =
- PERSON\_FACEPALMING =
- PERSON\_SHRUGGING =

- POLICE\_OFFICER =
- DETECTIVE =
- GUARD =
- NINJA =
- CONSTRUCTION\_WORKER =
- PRINCE =
- PRINCESS =
- PERSON\_WEARING\_TURBAN =
- PERSON\_WITH\_SKULLCAP =
- WOMAN\_WITH\_HEADSCARF =
- PERSON\_IN\_TUXEDO =
- PERSON\_WITH\_VEIL =
- PREGNANT\_WOMAN =
- BREAST\_FEEDING =
- BABY\_ANGEL =
- SANTA\_CLAUS =
- MRS\_CLAUS =
- SUPERHERO =
- SUPERVILLAIN =
- MAGE =
- FAIRY =
- VAMPIRE =
- MERPERSON =
- ELF =
- GENIE =
- ZOMBIE =
- PERSON\_GETTING\_MASSAGE =
- PERSON\_GETTING\_HAIRCUT =
- PERSON\_WALKING =
- PERSON\_STANDING =
- PERSON\_KNEELING =
- PERSON\_RUNNING =
- WOMAN\_DANCING =
- MAN\_DANCING =
- PERSON\_IN\_SUIT\_LEVITATING =
- PEOPLE\_WITH\_BUNNY\_EARS =

- PERSON\_IN\_STEAMY\_ROOM =
- PERSON\_CLIMBING =
- PERSON\_FENCING =
- HORSE\_RACING =
- SKIER =
- SNOWBOARDER =
- PERSON\_GOLFING =
- PERSON\_SURFING =
- PERSON\_ROWING\_BOAT =
- PERSON\_SWIMMING =
- PERSON\_BOUNCING\_BALL =
- PERSON\_LIFTING\_WEIGHTS =
- PERSON\_BIKING =
- PERSON\_MOUNTAIN\_BIKING =
- PERSON\_CARTWHEELING =
- PEOPLE\_WRESTLING =
- PERSON\_PLAYING\_WATER\_POLO =
- PERSON\_PLAYING\_HANDBALL =
- PERSON\_JUGGLING =
- PERSON\_IN\_LOTUS\_POSITION =
- PERSON\_TAKING\_BATH =
- PERSON\_IN\_BED =
- WOMEN\_HOLDING\_HANDS =
- WOMAN\_AND\_MAN\_HOLDING\_HANDS =
- MEN\_HOLDING\_HANDS =
- KISS =
- COUPLE\_WITH\_HEART =
- FAMILY =
- SPEAKING\_HEAD =
- BUST\_IN\_SILHOUETTE =
- BUSTS\_IN\_SILHOUETTE =
- PEOPLE\_HUGGING =
- FOOTPRINTS =
- LIGHT\_SKIN\_TONE =
- MEDIUM\_LIGHT\_SKIN\_TONE =
- MEDIUM\_SKIN\_TONE =

- MEDIUM\_DARK\_SKIN\_TONE =
- DARK\_SKIN\_TONE =
- RED\_HAIR =
- CURLY\_HAIR =
- WHITE\_HAIR =
- BALD =
- MONKEY\_FACE =
- MONKEY =
- GORILLA =
- ORANGUTAN =
- DOG\_FACE =
- DOG =
- GUIDE\_DOG =
- POODLE =
- WOLF =
- FOX =
- RACCOON =
- CAT\_FACE =
- CAT =
- LION =
- TIGER\_FACE =
- TIGER =
- LEOPARD =
- HORSE\_FACE =
- HORSE =
- UNICORN =
- ZEBRA =
- DEER =
- BISON =
- COW\_FACE =
- OX =
- WATER\_BUFFALO =
- COW =
- PIG\_FACE =
- PIG =
- BOAR =

- PIG\_NOSE =
- RAM =
- EWE =
- GOAT =
- CAMEL =
- TWO\_HUMP\_CAMEL =
- LLAMA =
- GIRAFFE =
- ELEPHANT =
- MAMMOTH =
- RHINOCEROS =
- HIPPOPOTAMUS =
- MOUSE\_FACE =
- MOUSE =
- RAT =
- HAMSTER =
- RABBIT\_FACE =
- RABBIT =
- CHIPMUNK =
- BEAVER =
- HEDGEHOG =
- BAT =
- BEAR =
- KOALA =
- PANDA =
- SLOTH =
- OTTER =
- SKUNK =
- KANGAROO =
- BADGER =
- PAW\_PRINTS =
- TURKEY =
- CHICKEN =
- ROOSTER =
- HATCHING\_CHICK =
- BABY\_CHICK =

- FRONT\_FACING\_BABY\_CHICK =
- BIRD =
- PENGUIN =
- DOVE =
- EAGLE =
- DUCK =
- SWAN =
- OWL =
- DODO =
- FEATHER =
- FLAMINGO =
- PEACOCK =
- PARROT =
- FROG =
- CROCODILE =
- TURTLE =
- LIZARD =
- SNAKE =
- DRAGON\_FACE =
- DRAGON =
- SAUROPOD =
- T\_REX =
- SPOUTING\_WHALE =
- WHALE =
- DOLPHIN =
- SEAL =
- FISH =
- TROPICAL\_FISH =
- BLOWFISH =
- SHARK =
- OCTOPUS =
- SPIRAL\_SHELL =
- SNAIL =
- BUTTERFLY =
- BUG =
- ANT =

- HONEYBEE =
- BEETLE =
- LADY\_BEETLE =
- CRICKET =
- COCKROACH =
- SPIDER =
- SPIDER\_WEB =
- SCORPION =
- MOSQUITO =
- FLY =
- WORM =
- MICROBE =
- BOUQUET =
- CHERRY\_BLOSSOM =
- WHITE\_FLOWER =
- ROSETTE =
- ROSE =
- WILTED\_FLOWER =
- HIBISCUS =
- SUNFLOWER =
- BLOSSOM =
- TULIP =
- SEEDLING =
- POTTED\_PLANT =
- EVERGREEN\_TREE =
- DECIDUOUS\_TREE =
- PALM\_TREE =
- CACTUS =
- SHEAF\_OF\_RICE =
- HERB =
- SHAMROCK =
- FOUR\_LEAF\_CLOVER =
- MAPLE\_LEAF =
- FALLEN\_LEAF =
- LEAF\_FLUTTERING\_IN\_WIND =
- GRAPES =

- MELON =
- WATERMELON =
- TANGERINE =
- LEMON =
- BANANA =
- PINEAPPLE =
- MANGO =
- RED\_APPLE =
- GREEN\_APPLE =
- PEAR =
- PEACH =
- CHERRIES =
- STRAWBERRY =
- BLUEBERRIES =
- KIWI\_FRUIT =
- TOMATO =
- OLIVE =
- COCONUT =
- AVOCADO =
- EGGPLANT =
- POTATO =
- CARROT =
- EAR\_OF\_CORN =
- HOT\_PEPPER =
- BELL\_PEPPER =
- CUCUMBER =
- LEAFY\_GREEN =
- BROCCOLI =
- GARLIC =
- ONION =
- MUSHROOM =
- PEANUTS =
- CHESTNUT =
- BREAD =
- CROISSANT =
- BAGUETTE\_BREAD =

- FLATBREAD =
- PRETZEL =
- BAGEL =
- PANCAKES =
- WAFFLE =
- CHEESE\_WEDGE =
- MEAT\_ON\_BONE =
- POULTRY\_LEG =
- CUT\_OF\_MEAT =
- BACON =
- HAMBURGER =
- FRENCH\_FRIES =
- PIZZA =
- HOT\_DOG =
- SANDWICH =
- TACO =
- BURRITO =
- TAMALE =
- STUFFED\_FLATBREAD =
- FALAFEL =
- EGG =
- COOKING =
- SHALLOW\_PAN\_OF\_FOOD =
- POT\_OF\_FOOD =
- FONDUE =
- BOWL\_WITH\_SPOON =
- GREEN\_SALAD =
- POPCORN =
- BUTTER =
- SALT =
- CANNED\_FOOD =
- BENTO\_BOX =
- RICE\_CRACKER =
- RICE\_BALL =
- COOKED\_RICE =
- CURRY\_RICE =

- STEAMING\_BOWL =
- SPAGHETTI =
- ROASTED\_SWEET\_POTATO =
- ODEN =
- SUSHI =
- FRIED\_SHRIMP =
- FISH\_CAKE\_WITH\_SWIRL =
- MOON\_CAKE =
- DANGO =
- DUMPLING =
- FORTUNE\_COOKIE =
- TAKEOUT\_BOX =
- CRAB =
- LOBSTER =
- SHRIMP =
- SQUID =
- OYSTER =
- SOFT\_ICE\_CREAM =
- SHAVED\_ICE =
- ICE\_CREAM =
- DOUGHNUT =
- COOKIE =
- BIRTHDAY\_CAKE =
- SHORTCAKE =
- CUPCAKE =
- PIE =
- CHOCOLATE\_BAR =
- CANDY =
- LOLLIPOP =
- CUSTARD =
- HONEY\_POT =
- BABY\_BOTTLE =
- GLASS\_OF\_MILK =
- HOT\_BEVERAGE =TEAPOT =
- TEACUP\_WITHOUT\_HANDLE =

- SAKE =
- BOTTLE\_WITH\_POPPING\_CORK =
- WINE\_GLASS =
- COCKTAIL\_GLASS =
- TROPICAL\_DRINK =
- BEER\_MUG =
- CLINKING\_BEER\_MUGS =
- CLINKING\_GLASSES =
- TUMBLER\_GLASS =
- CUP\_WITH\_STRAW =
- BUBBLE\_TEA =
- BEVERAGE\_BOX =
- MATE =
- ICE =
- CHOPSTICKS =
- FORK\_AND\_KNIFE\_WITH\_PLATE =
- FORK\_AND\_KNIFE =
- SPOON =
- KITCHEN\_KNIFE =
- AMPHORA =
- GLOBE\_SHOWING\_EUROPE\_AFRICA =
- GLOBE\_SHOWING\_AMERICAS =
- GLOBE\_SHOWING\_ASIA\_AUSTRALIA =
- GLOBE\_WITH\_MERIDIANS =
- WORLD\_MAP =
- MAP\_OF\_JAPAN =
- COMPASS =
- SNOW\_CAPPED\_MOUNTAIN =
- MOUNTAIN =
- VOLCANO =
- MOUNT\_FUJI =
- CAMPING =
- BEACH\_WITH\_UMBRELLA =
- DESERT =
- DESERT\_ISLAND =
- NATIONAL\_PARK =

- STADIUM =
- CLASSICAL\_BUILDING =
- BUILDING\_CONSTRUCTION =
- BRICK =
- ROCK =
- WOOD =
- HUT =
- HOUSES =
- DERELICT\_HOUSE =
- HOUSE =
- HOUSE\_WITH\_GARDEN =
- OFFICE\_BUILDING =
- JAPANESE\_POST\_OFFICE =
- POST\_OFFICE =
- HOSPITAL =
- BANK =
- HOTEL =
- LOVE\_HOTEL =
- CONVENIENCE\_STORE =
- SCHOOL =
- DEPARTMENT\_STORE =
- FACTORY =
- JAPANESE\_CASTLE =
- CASTLE =
- WEDDING =
- TOKYO\_TOWER =
- STATUE\_OF\_LIBERTY =
- CHURCH =
- MOSQUE =
- HINDU\_TEMPLE =
- SYNAGOGUE =
- SHINTO\_SHRINE =
- KAABA =
- FOUNTAIN =
- TENT =
- FOGGY =

- NIGHT\_WITH\_STARS =
- CITYSCAPE =
- SUNRISE\_OVER\_MOUNTAINS =
- SUNRISE =
- CITYSCAPE\_AT\_DUSK =
- SUNSET =
- BRIDGE\_AT\_NIGHT =
- HOT\_SPRINGS =
- CAROUSEL\_HORSE =
- FERRIS\_WHEEL =
- ROLLER\_COASTER =
- BARBER\_POLE =
- CIRCUS\_TENT =
- LOCOMOTIVE =
- RAILWAY\_CAR =
- HIGH\_SPEED\_TRAIN =
- BULLET\_TRAIN =
- TRAIN =
- METRO =
- LIGHT\_RAIL =
- STATION =
- TRAM =
- MONORAIL =
- MOUNTAIN\_RAILWAY =
- $TRAM\_CAR =$
- BUS =
- ONCOMING\_BUS =
- TROLLEYBUS =
- MINIBUS =
- AMBULANCE =
- FIRE\_ENGINE =
- POLICE\_CAR =
- ONCOMING\_POLICE\_CAR =
- TAXI =
- ONCOMING\_TAXI =
- AUTOMOBILE =

- ONCOMING\_AUTOMOBILE =
- SPORT\_UTILITY\_VEHICLE =
- PICKUP\_TRUCK =
- DELIVERY\_TRUCK =
- ARTICULATED\_LORRY =
- TRACTOR =
- RACING\_CAR =
- MOTORCYCLE =
- MOTOR\_SCOOTER =
- MANUAL\_WHEELCHAIR =
- MOTORIZED\_WHEELCHAIR =
- AUTO\_RICKSHAW =
- BICYCLE =
- KICK\_SCOOTER =
- SKATEBOARD =
- ROLLER SKATE =
- BUS\_STOP =
- MOTORWAY =
- RAILWAY\_TRACK =
- OIL\_DRUM =
- FUEL\_PUMP =
- POLICE\_CAR\_LIGHT =
- HORIZONTAL\_TRAFFIC\_LIGHT =
- VERTICAL\_TRAFFIC\_LIGHT =
- STOP\_SIGN =
- CONSTRUCTION =
- ANCHOR =
- SAILBOAT =
- CANOE =
- SPEEDBOAT =
- PASSENGER\_SHIP =
- FERRY =
- MOTOR\_BOAT =
- SHIP =
- AIRPLANE =
- SMALL\_AIRPLANE =

- AIRPLANE\_DEPARTURE =
- AIRPLANE\_ARRIVAL =
- PARACHUTE =
- SEAT =
- HELICOPTER =
- SUSPENSION\_RAILWAY =
- MOUNTAIN\_CABLEWAY =
- AERIAL\_TRAMWAY =
- SATELLITE =
- ROCKET =
- FLYING\_SAUCER =
- BELLHOP\_BELL =
- LUGGAGE =
- HOURGLASS\_DONE =
- HOURGLASS\_NOT\_DONE =
- WATCH =
- ALARM\_CLOCK =
- STOPWATCH =
- TIMER\_CLOCK =
- MANTELPIECE\_CLOCK =
- TWELVE\_OCLOCK =
- TWELVE\_THIRTY =
- ONE\_OCLOCK =
- ONE\_THIRTY =
- TWO\_OCLOCK =
- TWO\_THIRTY =
- THREE\_OCLOCK =
- THREE\_THIRTY =
- FOUR\_OCLOCK =
- FOUR\_THIRTY =
- FIVE\_OCLOCK =
- FIVE\_THIRTY =
- SIX\_OCLOCK =
- SIX\_THIRTY =
- SEVEN\_OCLOCK =
- SEVEN\_THIRTY =

- EIGHT\_OCLOCK =
- EIGHT\_THIRTY =
- NINE\_OCLOCK =
- NINE\_THIRTY =
- TEN\_OCLOCK =
- TEN\_THIRTY =
- ELEVEN\_OCLOCK =
- ELEVEN\_THIRTY =
- NEW\_MOON =
- WAXING\_CRESCENT\_MOON =
- FIRST\_QUARTER\_MOON =
- WAXING\_GIBBOUS\_MOON =
- FULL\_MOON =
- WANING\_GIBBOUS\_MOON =
- LAST\_QUARTER\_MOON =
- WANING\_CRESCENT\_MOON =
- CRESCENT\_MOON =
- NEW\_MOON\_FACE =
- FIRST\_QUARTER\_MOON\_FACE =
- LAST\_QUARTER\_MOON\_FACE =
- THERMOMETER =
- SUN =
- FULL\_MOON\_FACE =
- SUN\_WITH\_FACE =
- RINGED\_PLANET =
- STAR =
- GLOWING\_STAR =
- SHOOTING\_STAR =
- MILKY\_WAY =
- CLOUD =
- SUN\_BEHIND\_CLOUD =
- CLOUD\_WITH\_LIGHTNING\_AND\_RAIN =
- SUN\_BEHIND\_SMALL\_CLOUD =
- SUN\_BEHIND\_LARGE\_CLOUD =
- SUN\_BEHIND\_RAIN\_CLOUD =
- CLOUD\_WITH\_RAIN =

- CLOUD\_WITH\_SNOW =
- CLOUD\_WITH\_LIGHTNING =
- TORNADO =
- FOG =
- WIND\_FACE =
- CYCLONE =
- RAINBOW =
- CLOSED\_UMBRELLA =
- UMBRELLA =
- UMBRELLA\_WITH\_RAIN\_DROPS =
- UMBRELLA\_ON\_GROUND =
- HIGH\_VOLTAGE =
- SNOWFLAKE =
- SNOWMAN =
- SNOWMAN\_WITHOUT\_SNOW =
- COMET =
- FIRE =
- DROPLET =
- WATER\_WAVE =
- JACK\_O\_LANTERN =
- CHRISTMAS\_TREE =
- FIREWORKS =
- SPARKLER =
- FIRECRACKER =
- SPARKLES =
- BALLOON =
- PARTY\_POPPER =
- CONFETTI\_BALL =
- TANABATA\_TREE =
- PINE\_DECORATION =
- JAPANESE\_DOLLS =
- CARP\_STREAMER =
- WIND\_CHIME =
- MOON\_VIEWING\_CEREMONY =
- RED ENVELOPE =
- RIBBON =

- WRAPPED\_GIFT =
- REMINDER\_RIBBON =
- ADMISSION\_TICKETS =
- TICKET =
- MILITARY\_MEDAL =
- TROPHY =
- SPORTS\_MEDAL =
- FIRST\_PLACE\_MEDAL =
- SECOND\_PLACE\_MEDAL =
- THIRD\_PLACE\_MEDAL =
- SOCCER\_BALL =
- BASEBALL =
- SOFTBALL =
- BASKETBALL =
- VOLLEYBALL =
- AMERICAN FOOTBALL =
- RUGBY\_FOOTBALL =
- TENNIS =
- FLYING\_DISC =
- BOWLING =
- CRICKET\_GAME =
- FIELD\_HOCKEY =
- ICE\_HOCKEY =
- LACROSSE =
- PING\_PONG =
- BADMINTON =
- BOXING\_GLOVE =
- MARTIAL\_ARTS\_UNIFORM =
- GOAL\_NET =
- FLAG\_IN\_HOLE =
- ICE\_SKATE =
- FISHING\_POLE =
- DIVING\_MASK =
- RUNNING\_SHIRT =
- SKIS =
- SLED =

- CURLING\_STONE =
- DIRECT\_HIT =
- YO\_YO =
- KITE =
- BALL =
- CRYSTAL\_BALL =
- MAGIC\_WAND =
- NAZAR\_AMULET =
- VIDEO\_GAME =
- JOYSTICK =
- SLOT\_MACHINE =
- GAME\_DIE =
- PUZZLE\_PIECE =
- TEDDY\_BEAR =
- $PI\tilde{n}ATA =$
- NESTING\_DOLLS =
- SPADE\_SUIT =
- HEART\_SUIT =
- DIAMOND\_SUIT =
- CLUB\_SUIT =
- CHESS\_PAWN =
- JOKER =
- MAHJONG\_RED\_DRAGON =
- FLOWER\_PLAYING\_CARDS =
- PERFORMING\_ARTS =
- FRAMED\_PICTURE =
- ARTIST\_PALETTE =
- THREAD =
- SEWING\_NEEDLE =
- YARN =
- KNOT =
- GLASSES =
- SUNGLASSES =
- GOGGLES =
- LAB\_COAT =
- SAFETY\_VEST =

- NECKTIE =
- T\_SHIRT =
- JEANS =
- SCARF =
- GLOVES =
- COAT =
- SOCKS =
- DRESS =
- KIMONO =
- SARI =
- ONE\_PIECE\_SWIMSUIT =
- BRIEFS =
- SHORTS =
- BIKINI =
- WOMANS\_CLOTHES =
- PURSE =
- HANDBAG =
- CLUTCH\_BAG =
- SHOPPING\_BAGS =
- BACKPACK =
- THONG\_SANDAL =
- MANS\_SHOE =
- RUNNING\_SHOE =
- HIKING\_BOOT =
- FLAT\_SHOE =
- HIGH\_HEELED\_SHOE =
- WOMANS\_SANDAL =
- BALLET\_SHOES =
- WOMANS\_BOOT =
- CROWN =
- WOMANS\_HAT =
- $TOP\_HAT =$
- GRADUATION\_CAP =
- BILLED\_CAP =
- MILITARY\_HELMET =
- RESCUE\_WORKERS\_HELMET =

- PRAYER\_BEADS =
- LIPSTICK =
- RING =
- GEM\_STONE =
- MUTED\_SPEAKER =
- SPEAKER\_LOW\_VOLUME =
- SPEAKER\_MEDIUM\_VOLUME =
- SPEAKER\_HIGH\_VOLUME =
- LOUDSPEAKER =
- MEGAPHONE =
- POSTAL\_HORN =
- BELL =
- BELL\_WITH\_SLASH =
- MUSICAL\_SCORE =
- MUSICAL\_NOTE =
- MUSICAL\_NOTES =
- STUDIO\_MICROPHONE =
- LEVEL\_SLIDER =
- CONTROL\_KNOBS =
- MICROPHONE =
- HEADPHONE =
- RADIO =
- SAXOPHONE =
- ACCORDION =
- GUITAR =
- MUSICAL\_KEYBOARD =
- TRUMPET =
- VIOLIN =
- BANJO =
- DRUM =
- LONG\_DRUM =
- MOBILE\_PHONE =
- MOBILE\_PHONE\_WITH\_ARROW =
- TELEPHONE =
- TELEPHONE\_RECEIVER =
- PAGER =

- FAX\_MACHINE =
- BATTERY =
- ELECTRIC\_PLUG =
- LAPTOP =
- DESKTOP\_COMPUTER =
- PRINTER =
- KEYBOARD =
- COMPUTER\_MOUSE =
- TRACKBALL =
- COMPUTER\_DISK =
- FLOPPY\_DISK =
- OPTICAL\_DISK =
- DVD =
- ABACUS =
- MOVIE\_CAMERA =
- FILM\_FRAMES =
- FILM\_PROJECTOR =
- CLAPPER\_BOARD =
- TELEVISION =
- CAMERA =
- CAMERA\_WITH\_FLASH =
- VIDEO\_CAMERA =
- VIDEOCASSETTE =
- MAGNIFYING\_GLASS\_TILTED\_LEFT =
- MAGNIFYING\_GLASS\_TILTED\_RIGHT =
- CANDLE =
- LIGHT\_BULB =
- FLASHLIGHT =
- RED\_PAPER\_LANTERN =
- DIYA\_LAMP =
- NOTEBOOK\_WITH\_DECORATIVE\_COVER =
- CLOSED\_BOOK =
- OPEN\_BOOK =
- GREEN\_BOOK =
- BLUE\_BOOK =
- ORANGE\_BOOK =

- BOOKS =
- NOTEBOOK =
- LEDGER =
- PAGE\_WITH\_CURL =
- SCROLL =
- PAGE\_FACING\_UP =
- NEWSPAPER =
- ROLLED\_UP\_NEWSPAPER =
- BOOKMARK\_TABS =
- BOOKMARK =
- LABEL =
- MONEY\_BAG =
- COIN =
- YEN\_BANKNOTE =
- DOLLAR\_BANKNOTE =
- EURO\_BANKNOTE =
- POUND\_BANKNOTE =
- MONEY\_WITH\_WINGS =
- CREDIT\_CARD =
- RECEIPT =
- CHART\_INCREASING\_WITH\_YEN =
- ENVELOPE =
- E\_MAIL =
- INCOMING\_ENVELOPE =
- ENVELOPE\_WITH\_ARROW =
- OUTBOX\_TRAY =
- INBOX\_TRAY =
- PACKAGE =
- CLOSED\_MAILBOX\_WITH\_RAISED\_FLAG =
- CLOSED\_MAILBOX\_WITH\_LOWERED\_FLAG =
- OPEN\_MAILBOX\_WITH\_RAISED\_FLAG =
- OPEN\_MAILBOX\_WITH\_LOWERED\_FLAG =
- POSTBOX =
- BALLOT\_BOX\_WITH\_BALLOT =
- PENCIL =
- BLACK\_NIB =

- FOUNTAIN\_PEN =
- PEN =
- PAINTBRUSH =
- CRAYON =
- MEMO =
- BRIEFCASE =
- FILE\_FOLDER =
- OPEN\_FILE\_FOLDER =
- CARD\_INDEX\_DIVIDERS =
- CALENDAR =
- TEAR\_OFF\_CALENDAR =
- SPIRAL\_NOTEPAD =
- SPIRAL\_CALENDAR =
- CARD\_INDEX =
- CHART\_INCREASING =
- CHART\_DECREASING =
- BAR\_CHART =
- CLIPBOARD =
- PUSHPIN =
- ROUND\_PUSHPIN =
- PAPERCLIP =
- LINKED\_PAPERCLIPS =
- STRAIGHT\_RULER =
- TRIANGULAR\_RULER =
- SCISSORS =
- CARD\_FILE\_BOX =
- FILE\_CABINET =
- WASTEBASKET =
- LOCKED =
- UNLOCKED =
- LOCKED\_WITH\_PEN =
- LOCKED\_WITH\_KEY =
- KEY =
- OLD\_KEY =
- HAMMER =
- AXE =

- PICK =
- HAMMER\_AND\_PICK =
- HAMMER\_AND\_WRENCH =
- DAGGER =
- CROSSED\_SWORDS =
- PISTOL =
- BOOMERANG =
- BOW\_AND\_ARROW =
- SHIELD =
- CARPENTRY\_SAW =
- WRENCH =
- SCREWDRIVER =
- NUT\_AND\_BOLT =
- GEAR =
- CLAMP =
- BALANCE\_SCALE =
- WHITE\_CANE =
- LINK =
- CHAINS =
- HOOK =
- TOOLBOX =
- MAGNET =
- LADDER =
- ALEMBIC =
- TEST\_TUBE =
- PETRI\_DISH =
- DNA =
- MICROSCOPE =
- TELESCOPE =
- SATELLITE\_ANTENNA =
- SYRINGE =
- DROP\_OF\_BLOOD =
- PILL =
- ADHESIVE\_BANDAGE =
- STETHOSCOPE =
- DOOR =

- ELEVATOR =
- MIRROR =
- WINDOW =
- BED =
- COUCH\_AND\_LAMP =
- CHAIR =
- TOILET =
- PLUNGER =
- SHOWER =
- BATHTUB =
- MOUSE\_TRAP =
- RAZOR =
- LOTION\_BOTTLE =
- SAFETY\_PIN =
- BROOM =
- BASKET =
- ROLL\_OF\_PAPER =
- BUCKET =
- SOAP =
- TOOTHBRUSH =
- SPONGE =
- FIRE\_EXTINGUISHER =
- SHOPPING\_CART =
- CIGARETTE =
- COFFIN =
- HEADSTONE =
- FUNERAL\_URN =
- MOAI =
- PLACARD =
- ATM\_SIGN =
- LITTER\_IN\_BIN\_SIGN =
- POTABLE\_WATER =
- WHEELCHAIR\_SYMBOL =
- MENS\_ROOM =
- WOMENS ROOM =
- RESTROOM =

- BABY\_SYMBOL =
- WATER\_CLOSET =
- PASSPORT\_CONTROL =
- CUSTOMS =
- BAGGAGE\_CLAIM =
- LEFT\_LUGGAGE =
- WARNING =
- CHILDREN\_CROSSING =
- NO\_ENTRY =
- PROHIBITED =
- NO\_BICYCLES =
- NO\_SMOKING =
- NO\_LITTERING =
- NON\_POTABLE\_WATER =
- NO\_PEDESTRIANS =
- NO\_MOBILE\_PHONES =
- NO\_ONE\_UNDER\_EIGHTEEN =
- RADIOACTIVE =
- BIOHAZARD =
- UP\_ARROW =
- UP\_RIGHT\_ARROW =
- RIGHT\_ARROW =
- DOWN\_RIGHT\_ARROW =
- DOWN\_ARROW =
- DOWN\_LEFT\_ARROW =
- LEFT\_ARROW =
- UP\_LEFT\_ARROW =
- UP\_DOWN\_ARROW =
- LEFT\_RIGHT\_ARROW =
- RIGHT\_ARROW\_CURVING\_LEFT =
- LEFT\_ARROW\_CURVING\_RIGHT =
- RIGHT\_ARROW\_CURVING\_UP =
- RIGHT\_ARROW\_CURVING\_DOWN =
- CLOCKWISE\_VERTICAL\_ARROWS =
- COUNTERCLOCKWISE\_ARROWS\_BUTTON =
- BACK\_ARROW =

- END\_ARROW =
- ON\_ARROW =
- SOON\_ARROW =
- TOP\_ARROW =
- PLACE\_OF\_WORSHIP =
- ATOM\_SYMBOL =
- OM =
- STAR\_OF\_DAVID =
- WHEEL\_OF\_DHARMA =
- YIN\_YANG =
- LATIN\_CROSS =
- ORTHODOX\_CROSS =
- STAR\_AND\_CRESCENT =
- PEACE\_SYMBOL =
- MENORAH =
- DOTTED\_SIX\_POINTED\_STAR =
- ARIES =
- TAURUS =
- GEMINI =
- CANCER =
- LEO =
- VIRGO =
- LIBRA =
- SCORPIO =
- SAGITTARIUS =
- CAPRICORN =
- AQUARIUS =
- PISCES =
- OPHIUCHUS =
- SHUFFLE\_TRACKS\_BUTTON =
- REPEAT\_BUTTON =
- REPEAT\_SINGLE\_BUTTON =
- PLAY\_BUTTON =
- FAST\_FORWARD\_BUTTON =
- NEXT\_TRACK\_BUTTON =
- PLAY\_OR\_PAUSE\_BUTTON =

- REVERSE\_BUTTON =
- FAST\_REVERSE\_BUTTON =
- LAST\_TRACK\_BUTTON =
- UPWARDS\_BUTTON =
- FAST\_UP\_BUTTON =
- DOWNWARDS\_BUTTON =
- FAST\_DOWN\_BUTTON =
- PAUSE\_BUTTON =
- STOP\_BUTTON =
- RECORD\_BUTTON =
- EJECT\_BUTTON =
- CINEMA =
- DIM\_BUTTON =
- BRIGHT\_BUTTON =
- ANTENNA\_BARS =
- VIBRATION MODE =
- MOBILE\_PHONE\_OFF =
- FEMALE\_SIGN =
- MALE\_SIGN =
- TRANSGENDER\_SYMBOL =
- MULTIPLY = x
- PLUS =
- MINUS =
- DIVIDE =
- INFINITY =
- DOUBLE\_EXCLAMATION\_MARK =
- EXCLAMATION\_QUESTION\_MARK =
- QUESTION\_MARK =
- WHITE\_QUESTION\_MARK =
- WHITE\_EXCLAMATION\_MARK =
- EXCLAMATION\_MARK =
- WAVY\_DASH =
- CURRENCY\_EXCHANGE =
- HEAVY\_DOLLAR\_SIGN =
- MEDICAL\_SYMBOL =
- RECYCLING\_SYMBOL =

- FLEUR\_DE\_LIS =
- TRIDENT\_EMBLEM =
- NAME\_BADGE =
- JAPANESE\_SYMBOL\_FOR\_BEGINNER =
- HOLLOW\_RED\_CIRCLE =
- CHECK\_MARK\_BUTTON =
- CHECK\_BOX\_WITH\_CHECK =
- CHECK\_MARK = ✓
- CROSS\_MARK =
- CROSS\_MARK\_BUTTON =
- CURLY\_LOOP =
- DOUBLE\_CURLY\_LOOP =
- PART\_ALTERNATION\_MARK =
- EIGHT\_SPOKED\_ASTERISK =
- EIGHT\_POINTED\_STAR =
- SPARKLE =
- COPYRIGHT = ©
- REGISTERED = ®
- TRADE\_MARK =  $^{TM}$
- INPUT\_LATIN\_UPPERCASE =
- INPUT\_LATIN\_LOWERCASE =
- INPUT\_NUMBERS =
- INPUT\_SYMBOLS =
- INPUT\_LATIN\_LETTERS =
- A\_BUTTON\_BLOOD\_TYPE =
- AB\_BUTTON\_BLOOD\_TYPE =
- B\_BUTTON\_BLOOD\_TYPE =
- CL\_BUTTON =
- COOL\_BUTTON =
- FREE\_BUTTON =
- INFORMATION =
- ID\_BUTTON =
- CIRCLED\_M =
- NEW\_BUTTON =
- NG\_BUTTON =
- O\_BUTTON\_BLOOD\_TYPE =

- OK\_BUTTON =
- P\_BUTTON =
- SOS\_BUTTON =
- UP\_BUTTON =
- VS\_BUTTON =
- JAPANESE\_HERE\_BUTTON =
- JAPANESE\_SERVICE\_CHARGE\_BUTTON =
- JAPANESE\_MONTHLY\_AMOUNT\_BUTTON =
- JAPANESE\_NOT\_FREE\_OF\_CHARGE\_BUTTON =
- JAPANESE\_RESERVED\_BUTTON =
- JAPANESE\_BARGAIN\_BUTTON =
- JAPANESE\_DISCOUNT\_BUTTON =
- JAPANESE\_FREE\_OF\_CHARGE\_BUTTON =
- JAPANESE\_PROHIBITED\_BUTTON =
- JAPANESE\_ACCEPTABLE\_BUTTON =
- JAPANESE APPLICATION BUTTON =
- JAPANESE\_PASSING\_GRADE\_BUTTON =
- JAPANESE\_VACANCY\_BUTTON =
- JAPANESE\_CONGRATULATIONS\_BUTTON =
- JAPANESE\_SECRET\_BUTTON =
- JAPANESE\_OPEN\_FOR\_BUSINESS\_BUTTON =
- JAPANESE\_NO\_VACANCY\_BUTTON =
- RED\_CIRCLE =
- ORANGE\_CIRCLE =
- YELLOW\_CIRCLE =
- GREEN\_CIRCLE =
- BLUE\_CIRCLE =
- PURPLE\_CIRCLE =
- BROWN\_CIRCLE =
- BLACK\_CIRCLE =
- WHITE\_CIRCLE =
- RED\_SQUARE =
- ORANGE\_SQUARE =
- YELLOW\_SQUARE =
- GREEN\_SQUARE =
- BLUE\_SQUARE =

- PURPLE\_SQUARE =
- BROWN\_SQUARE =
- BLACK\_LARGE\_SQUARE =
- WHITE\_LARGE\_SQUARE =
- BLACK\_MEDIUM\_SQUARE =
- WHITE\_MEDIUM\_SQUARE =
- BLACK\_MEDIUM\_SMALL\_SQUARE =
- WHITE\_MEDIUM\_SMALL\_SQUARE =
- BLACK\_SMALL\_SQUARE =
- WHITE\_SMALL\_SQUARE =
- LARGE\_ORANGE\_DIAMOND =
- LARGE\_BLUE\_DIAMOND =
- SMALL\_ORANGE\_DIAMOND =
- SMALL\_BLUE\_DIAMOND =
- RED\_TRIANGLE\_POINTED\_UP =
- RED\_TRIANGLE\_POINTED\_DOWN =
- DIAMOND\_WITH\_A\_DOT =
- RADIO\_BUTTON =
- WHITE\_SQUARE\_BUTTON =
- BLACK\_SQUARE\_BUTTON =
- CHEQUERED\_FLAG =
- TRIANGULAR\_FLAG =
- CROSSED\_FLAGS =
- BLACK\_FLAG =
- WHITE\_FLAG =

\_\_init\_\_()

## **Methods**

\_\_init\_\_()

## **Attributes**

**ABACUS** AB\_BUTTON\_BLOOD\_TYPE ACCORDION ADHESIVE\_BANDAGE ADMISSION\_TICKETS AERIAL\_TRAMWAY **AIRPLANE** AIRPLANE\_ARRIVAL AIRPLANE\_DEPARTURE ALARM\_CLOCK ALEMBIC ALIEN ALIEN\_MONSTER **AMBULANCE** AMERICAN\_FOOTBALL **AMPHORA** ANATOMICAL\_HEART **ANCHOR** ANGER\_SYMBOL ANGRY\_FACE ANGRY\_FACE\_WITH\_HORNS ANGUISHED\_FACE ANT ANTENNA\_BARS ANXIOUS\_FACE\_WITH\_SWEAT

continues on next page

Table 5 – continued from previous page

Table 5 – continued from previous page
AQUARIUS
ARIES
ARTICULATED_LORRY
ARTIST_PALETTE
ASTONISHED_FACE
ATM_SIGN
ATOM_SYMBOL
AUTOMOBILE
AUTO_RICKSHAW
AVOCADO
AXE
A_BUTTON_BLOOD_TYPE
BABY
BABY_ANGEL
BABY_BOTTLE
BABY_CHICK
BABY_SYMBOL
BACKHAND_INDEX_POINTING_DOWN
BACKHAND_INDEX_POINTING_LEFT
BACKHAND_INDEX_POINTING_RIGHT
BACKHAND_INDEX_POINTING_UP
BACKPACK
BACK_ARROW
BACON
BADGER
BADMINTON
continues on next page

Table 5 – continued from previous page	Table	5 –	continued	from	previous	page
--	-------	-----	-----------	------	----------	------

Т	Table 5 – continued from previous page	
BAGEL		
BAGGAGE_CLAIM		
BAGUETTE_BREAD		
BALANCE_SCALE		
BALD		
BALL		
BALLET_SHOES		
BALLOON		
BALLOT_BOX_WITH_BALLOT		
BANANA		
BANJO		
BANK		
BARBER_POLE		
BAR_CHART		
BASEBALL		
BASKET		
BASKETBALL		
BAT		
BATHTUB		
BATTERY		
BEACH_WITH_UMBRELLA		
BEAMING_FACE_WITH_SMILING	G_EYES	
BEAR		
BEATING_HEART		
BEAVER		
BED		

Table 5 – continued from previous page

Table 5 – continued from previous page	
BEER_MUG	
BEETLE	
BELL	
BELLHOP_BELL	
BELL_PEPPER	
BELL_WITH_SLASH	
BENTO_BOX	
BEVERAGE_BOX	
BICYCLE	
BIKINI	
BILLED_CAP	
BIOHAZARD	
BIRD	
BIRTHDAY_CAKE	
BISON	
BLACK_CIRCLE	
BLACK_FLAG	
BLACK_HEART	
BLACK_LARGE_SQUARE	
BLACK_MEDIUM_SMALL_SQUARE	
BLACK_MEDIUM_SQUARE	
BLACK_NIB	
BLACK_SMALL_SQUARE	
BLACK_SQUARE_BUTTON	
BLOSSOM	
BLOWFISH	
	continues on next page

Table 5 – continued from previous page

Table 5 – continued from previous page
BLUEBERRIES
BLUE_BOOK
BLUE_CIRCLE
BLUE_HEART
BLUE_SQUARE
BOAR
BOMB
BONE
BOOKMARK
BOOKMARK_TABS
BOOKS
BOOMERANG
BOTTLE_WITH_POPPING_CORK
BOUQUET
BOWLING
BOWL_WITH_SPOON
BOW_AND_ARROW
BOXING_GLOVE
BOY
BRAIN
BREAD
BREAST_FEEDING
BRICK
BRIDGE_AT_NIGHT
BRIEFCASE
BRIEFS
continues on next nage

Table 5 – continued from previous page

	Table 5 – continued from previous page	
BRIGHT_BUTTON		
BROCCOLI		
BROKEN_HEART		
BROOM		
BROWN_CIRCLE		
BROWN_HEART		
BROWN_SQUARE		
BUBBLE_TEA		
BUCKET		
BUG		
BUILDING_CONSTRUCTION		
BULLET_TRAIN		
BURRITO		
BUS		
BUSTS_IN_SILHOUETTE		
BUST_IN_SILHOUETTE		
BUS_STOP		
BUTTER		
BUTTERFLY		
B_BUTTON_BLOOD_TYPE		
CACTUS		
CALENDAR		
CALL_ME_HAND		
CAMEL		
CAMERA		
CAMERA_WITH_FLASH		
	000	ntinues on next page

continues on next page

Table 5	5 – continued	from previous page	è
---------	---------------	--------------------	---

Table 5 – continued from previous page	
CAMPING	
CANCER	
CANDLE	
CANDY	
CANNED_FOOD	
CANOE	
CAPRICORN	
CARD_FILE_BOX	
CARD_INDEX	
CARD_INDEX_DIVIDERS	
CAROUSEL_HORSE	
CARPENTRY_SAW	
CARP_STREAMER	
CARROT	
CASTLE	
CAT	
CAT_FACE	
CAT_WITH_TEARS_OF_JOY	
CAT_WITH_WRY_SMILE	
CHAINS	
CHAIR	
CHART_DECREASING	
CHART_INCREASING	
CHART_INCREASING_WITH_YEN	
CHECK_BOX_WITH_CHECK	
CHECK_MARK	
	continues on next page

Table 5 – continued from previous page

CHECK\_MARK\_BUTTON CHEESE\_WEDGE CHEQUERED\_FLAG CHERRIES CHERRY\_BLOSSOM CHESS\_PAWN CHESTNUT CHICKEN CHILD CHILDREN\_CROSSING CHIPMUNK CHOCOLATE\_BAR CHOPSTICKS CHRISTMAS\_TREE **CHURCH** CIGARETTE CINEMA CIRCLED\_M CIRCUS\_TENT CITYSCAPE CITYSCAPE\_AT\_DUSK CLAMPCLAPPER\_BOARD CLAPPING\_HANDS CLASSICAL\_BUILDING CLINKING\_BEER\_MUGS

continues on next page

Table	5 –	continued	from	previous	page
IUDIC	•	COLLINIACA	110111	picvious	page

Table 5 – continued from previous page	
CLINKING_GLASSES	
CLIPBOARD	
CLOCKWISE_VERTICAL_ARROWS	
CLOSED_BOOK	
CLOSED_MAILBOX_WITH_LOWERED_FLAG	
CLOSED_MAILBOX_WITH_RAISED_FLAG	
CLOSED_UMBRELLA	
CLOUD	
CLOUD_WITH_LIGHTNING	
CLOUD_WITH_LIGHTNING_AND_RAIN	
CLOUD_WITH_RAIN	
CLOUD_WITH_SNOW	
CLOWN_FACE	
CLUB_SUIT	
CLUTCH_BAG	
CL_BUTTON	
COAT	
COCKROACH	
COCKTAIL_GLASS	
COCONUT	
COFFIN	
COIN	
COLD_FACE	
COLLISION	
COMET	
COMPASS	
	continues on next page

Table 5 – continued from previous page

Table 5 – continued from previous page	
COMPUTER_DISK	
COMPUTER_MOUSE	
CONFETTI_BALL	
CONFOUNDED_FACE	
CONFUSED_FACE	
CONSTRUCTION	
CONSTRUCTION_WORKER	
CONTROL_KNOBS	
CONVENIENCE_STORE	
COOKED_RICE	
COOKIE	
COOKING	
COOL_BUTTON	
COPYRIGHT	
COUCH_AND_LAMP	
COUNTERCLOCKWISE_ARROWS_BUTTON	
COUPLE_WITH_HEART	
COW	
COWBOY_HAT_FACE	
COW_FACE	
CRAB	
CRAYON	
CREDIT_CARD	
CRESCENT_MOON	
CRICKET	
CRICKET_GAME	
	continues on next page

continues on next page

Table	5 –	continued	from	previous	page

	Table 5 – continued from previous page	
CROCODILE		
CROISSANT		
CROSSED_FINGERS		
CROSSED_FLAGS		
CROSSED_SWORDS		
CROSS_MARK		
CROSS_MARK_BUTTON		
CROWN		
CRYING_CAT		
CRYING_FACE		
CRYSTAL_BALL		
CUCUMBER		
CUPCAKE		
CUP_WITH_STRAW		
CURLING_STONE		
CURLY_HAIR		
CURLY_LOOP		
CURRENCY_EXCHANGE		
CURRY_RICE		
CUSTARD		
CUSTOMS		
CUT_OF_MEAT		
CYCLONE		
DAGGER		
DANGO		
DARK_SKIN_TONE		
	continues on ne	ext nage

Table 5 – continued from previous page

	Table 5 – continued from previous page	
DASHING_AWAY		
DEAF_PERSON		
DECIDUOUS_TREE		
DEER		
DELIVERY_TRUCK		
DEPARTMENT_STORE		
DERELICT_HOUSE		
DESERT		
DESERT_ISLAND		
DESKTOP_COMPUTER		
DETECTIVE		
DIAMOND_SUIT		
DIAMOND_WITH_A_DOT		
DIM_BUTTON		
DIRECT_HIT		
DISAPPOINTED_FACE		
DISGUISED_FACE		
DIVIDE		
DIVING_MASK		
DIYA_LAMP		
DIZZY		
DIZZY_FACE		
DNA		
DODO		
DOG		
DOG_FACE		
DOG_FACE		
	continues on ne	ADCO IV

continues on next page

Table 5	5 – continued	from previous page	è
---------	---------------	--------------------	---

DOLLAR_BANKNOTE		
DOLPHIN		
DOOR		
DOTTED_SIX_POINTED_STAR	R	
DOUBLE_CURLY_LOOP		
DOUBLE_EXCLAMATION_MARK	ζ	
DOUGHNUT		
DOVE		
DOWNCAST_FACE_WITH_SWEA	AT .	
DOWNWARDS_BUTTON		
DOWN_ARROW		
DOWN_LEFT_ARROW		
DOWN_RIGHT_ARROW		
DRAGON		
DRAGON_FACE		
DRESS		
DROOLING_FACE		
DROPLET		
DROP_OF_BLOOD		
DRUM		
DUCK		
DUMPLING		
DVD		
EAGLE		
EAR		
EAR_OF_CORN		
		continues on next page

Table 5 – continued from previous page

EAR_WITH_HEARING_AID  EGG  EGGPLANT  EIGHT_OCLOCK  EIGHT_POINTED_STAR  EIGHT_SPOKED_ASTERISK  EIGHT_THIRTY  EJECT_BUTTON  ELECTRIC_PLUG  ELEPHANT  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE  ENVELOPE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_MARK  EXCLAMATION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES  CONTINUES ON NEXT PAGE	Table 5 – continued from previous page
EGGPLANT EIGHT_OCLOCK EIGHT_POINTED_STAR EIGHT_SPOKED_ASTERISK EIGHT_THIRTY EJECT_BUTTON ELECTRIC_PLUG ELEPHANT ELEVATOR ELEVEN_OCLOCK ELEVEN_THIRTY ELF END_ARROW ENVELOPE ENVELOPE_WITH_ARROW EURO_BANKNOTE EVERGREEN_TREE EWE EXCLAMATION_MARK EXCLAMATION_QUESTION_MARK EXPLODING_HEAD EXPRESSIONLESS_FACE EYE EYES	EAR_WITH_HEARING_AID
EIGHT_POINTED_STAR  EIGHT_SPOKED_ASTERISK  EIGHT_THIRTY  EJECT_BUTTON  ELECTRIC_PLUG  ELEPHANT  ELEVATOR  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE  ENVELOPE_NITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYE	EGG
EIGHT_POINTED_STAR  EIGHT_SPOKED_ASTERISK  EIGHT_THIRTY  EJECT_BUTTON  ELECTRIC_PLUG  ELEPHANT  ELEVATOR  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	EGGPLANT
EIGHT_SPOKED_ASTERISK  EIGHT_THIRTY  EJECT_BUTTON  ELECTRIC_PLUG  ELEPHANT  ELEVATOR  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	EIGHT_OCLOCK
EIGHT_THIRTY  EJECT_BUTION  ELECTRIC_PLUG  ELEPHANT  ELEVATOR  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE  ENVELOPE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	EIGHT_POINTED_STAR
EJECT_BUTTON  ELECTRIC_PLUG  ELEPHANT  ELEVATOR  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	EIGHT_SPOKED_ASTERISK
ELECTRIC_PLUG  ELEPHANT  ELEVATOR  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	EIGHT_THIRTY
ELEVATOR  ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	EJECT_BUTTON
ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_MARK  EXCLAMATION_GUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	ELECTRIC_PLUG
ELEVEN_OCLOCK  ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	ELEPHANT
ELEVEN_THIRTY  ELF  END_ARROW  ENVELOPE  ENVELOPE  ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	ELEVATOR
ELF END_ARROW ENVELOPE ENVELOPE_WITH_ARROW  EURO_BANKNOTE EVERGREEN_TREE EWE EXCLAMATION_MARK EXCLAMATION_QUESTION_MARK EXPLODING_HEAD EXPRESSIONLESS_FACE EYE EYES	ELEVEN_OCLOCK
ENVELOPE ENVELOPE_WITH_ARROW  EURO_BANKNOTE EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	ELEVEN_THIRTY
ENVELOPE ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	ELF
ENVELOPE_WITH_ARROW  EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	END_ARROW
EURO_BANKNOTE  EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYE	ENVELOPE
EVERGREEN_TREE  EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYE	ENVELOPE_WITH_ARROW
EWE  EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYE	EURO_BANKNOTE
EXCLAMATION_MARK  EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYE	EVERGREEN_TREE
EXCLAMATION_QUESTION_MARK  EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYE	EWE .
EXPLODING_HEAD  EXPRESSIONLESS_FACE  EYE  EYES	EXCLAMATION_MARK
EXPRESSIONLESS_FACE  EYE  EYES	EXCLAMATION_QUESTION_MARK
EYES EYES	EXPLODING_HEAD
EYES	EXPRESSIONLESS_FACE
	EYE
	EYES
CHIMINAL IN SAMMING	

Table	5 –	continued	from	previous	page
IUDIC	•	COLLINIACA	110111	picvious	page

Table	e 5 – continued from previous page
E_MAIL	
FACE_BLOWING_A_KISS	
FACE_SAVORING_FOOD	
FACE_SCREAMING_IN_FEAR	
FACE_VOMITING	
FACE_WITHOUT_MOUTH	
FACE_WITH_HAND_OVER_MOUTH	
FACE_WITH_HEAD_BANDAGE	
FACE_WITH_MEDICAL_MASK	
FACE_WITH_MONOCLE	
FACE_WITH_OPEN_MOUTH	
FACE_WITH_RAISED_EYEBROW	
FACE_WITH_ROLLING_EYES	
FACE_WITH_STEAM_FROM_NOSE	
FACE_WITH_SYMBOLS_ON_MOUTH	
FACE_WITH_TEARS_OF_JOY	
FACE_WITH_THERMOMETER	
FACE_WITH_TONGUE	
FACTORY	
FAIRY	
FALAFEL	
FALLEN_LEAF	
FAMILY	
FAST_DOWN_BUTTON	
FAST_FORWARD_BUTTON	
FAST_REVERSE_BUTTON	
	continues on next page

Table 5 – continued from previous page

	Table 5 – continued from previous page	
FAST_UP_BUTTON		
FAX_MACHINE		
FEARFUL_FACE		
FEATHER		
FEMALE_SIGN		
FERRIS_WHEEL		
FERRY		
FIELD_HOCKEY		
FILE_CABINET		
FILE_FOLDER		
FILM_FRAMES		
FILM_PROJECTOR		
FIRE		
FIRECRACKER		
FIREWORKS		
FIRE_ENGINE		
FIRE_EXTINGUISHER		
FIRST_PLACE_MEDAL		
FIRST_QUARTER_MOON		
FIRST_QUARTER_MOON_FACE		
FISH		
FISHING_POLE		
FISH_CAKE_WITH_SWIRL		
FIVE_OCLOCK		
FIVE_THIRTY		
FLAG_IN_HOLE		
		continues on next page

continues on next page

Table 5	- continued	from	previous page	
---------	-------------	------	---------------	--

Table 5 – continued from previous page
FLAMINGO
FLASHLIGHT
FLATBREAD
FLAT_SHOE
FLEUR_DE_LIS
FLEXED_BICEPS
FLOPPY_DISK
FLOWER_PLAYING_CARDS
FLUSHED_FACE
FLY
FLYING_DISC
FLYING_SAUCER
FOG
FOGGY
FOLDED_HANDS
FONDUE
FOOT
FOOTPRINTS
FORK_AND_KNIFE
FORK_AND_KNIFE_WITH_PLATE
FORTUNE_COOKIE
FOUNTAIN
FOUNTAIN_PEN
FOUR_LEAF_CLOVER
FOUR_OCLOCK
FOUR_THIRTY
continues on next nage

Table 5 – continued from previous page

lable 5 – continued from previous page
FOX
FRAMED_PICTURE
FREE_BUTTON
FRENCH_FRIES
FRIED_SHRIMP
FROG
FRONT_FACING_BABY_CHICK
FROWNING_FACE
FROWNING_FACE_WITH_OPEN_MOUTH
FUEL_PUMP
FULL_MOON
FULL_MOON_FACE
FUNERAL_URN
GAME_DIE
GARLIC
GEAR
GEMINI
GEM_STONE
GENIE
GHOST
GIRAFFE
GIRL
GLASSES
GLASS_OF_MILK
GLOBE_SHOWING_AMERICAS
GLOBE_SHOWING_ASIA_AUSTRALIA
continues on next page
Continues on next page

Table 5 – continued from previous page

GLOBE\_SHOWING\_EUROPE\_AFRICA GLOBE\_WITH\_MERIDIANS **GLOVES** GLOWING\_STAR GOAL\_NET GOATGOBLIN **GOGGLES GORILLA** GRADUATION\_CAP **GRAPES** GREEN\_APPLE GREEN\_BOOK GREEN\_CIRCLE GREEN\_HEART GREEN\_SALAD GREEN\_SQUARE GRIMACING\_FACE GRINNING\_CAT GRINNING\_CAT\_WITH\_SMILING\_EYES GRINNING\_FACE GRINNING\_FACE\_WITH\_BIG\_EYES GRINNING\_FACE\_WITH\_SMILING\_EYES GRINNING\_FACE\_WITH\_SWEAT GRINNING\_SQUINTING\_FACE GROWING\_HEART

Table	5 –	continued	from	previous	page
IUDIC	•	COLLINIACA	110111	picvious	page

Table 5 – continued from previous page
GUARD
GUIDE_DOG
GUITAR
HAMBURGER
HAMMER
HAMMER_AND_PICK
HAMMER_AND_WRENCH
HAMSTER
HANDBAG
HANDSHAKE
HAND_WITH_FINGERS_SPLAYED
HATCHING_CHICK
HEADPHONE
HEADSTONE
HEART_DECORATION
HEART_EXCLAMATION
HEART_SUIT
HEART_WITH_ARROW
HEART_WITH_RIBBON
HEAR_NO_EVIL_MONKEY
HEAVY_DOLLAR_SIGN
HEDGEHOG
HELICOPTER
HERB
HIBISCUS
HIGH_HEELED_SHOE
continues on next page

Table 5 – continued from previous page

Table 5 – continued from previous page
HIGH_SPEED_TRAIN
HIGH_VOLTAGE
HIKING_BOOT
HINDU_TEMPLE
HIPPOPOTAMUS
HOLE
HOLLOW_RED_CIRCLE
HONEYBEE
HONEY_POT
НООК
HORIZONTAL_TRAFFIC_LIGHT
HORSE
HORSE_FACE
HORSE_RACING
HOSPITAL
HOTEL
HOT_BEVERAGE
HOT_DOG
HOT_FACE
HOT_PEPPER
HOT_SPRINGS
HOURGLASS_DONE
HOURGLASS_NOT_DONE
HOUSE
HOUSES
HOUSE_WITH_GARDEN
continues on next page

Table 5 – continued from previous page

Table 5 – continued from previous page
HUGGING_FACE
HUNDRED_POINTS
HUSHED_FACE
HUT
ICE
ICE_CREAM
ICE_HOCKEY
ICE_SKATE
ID_BUTTON
INBOX_TRAY
INCOMING_ENVELOPE
INDEX_POINTING_UP
INFINITY
INFORMATION
INPUT_LATIN_LETTERS
INPUT_LATIN_LOWERCASE
INPUT_LATIN_UPPERCASE
INPUT_NUMBERS
INPUT_SYMBOLS
JACK_O_LANTERN
JAPANESE_ACCEPTABLE_BUTTON
JAPANESE_APPLICATION_BUTTON
JAPANESE_BARGAIN_BUTTON
JAPANESE_CASTLE
JAPANESE_CONGRATULATIONS_BUTTON
JAPANESE_DISCOUNT_BUTTON
continues on next page
commisse on none page

Table 5	<ul><li>continued</li></ul>	from	previous	page
---------	-----------------------------	------	----------	------

Table 5 – continued from previous page	
JAPANESE_DOLLS	
JAPANESE_FREE_OF_CHARGE_BUTTON	
JAPANESE_HERE_BUTTON	
JAPANESE_MONTHLY_AMOUNT_BUTTON	
JAPANESE_NOT_FREE_OF_CHARGE_BUTTON	
JAPANESE_NO_VACANCY_BUTTON	
JAPANESE_OPEN_FOR_BUSINESS_BUTTON	
JAPANESE_PASSING_GRADE_BUTTON	
JAPANESE_POST_OFFICE	
JAPANESE_PROHIBITED_BUTTON	
JAPANESE_RESERVED_BUTTON	
JAPANESE_SECRET_BUTTON	
JAPANESE_SERVICE_CHARGE_BUTTON	
JAPANESE_SYMBOL_FOR_BEGINNER	
JAPANESE_VACANCY_BUTTON	
JEANS	
JOKER	
JOYSTICK	
KAABA	
KANGAROO	
KEY	
KEYBOARD	
KICK_SCOOTER	
KIMONO	
KISS	
KISSING_CAT	
	continues on next page

Tahla	5_	continued	from	previous page
rabie	$\circ$ –	continuea	Irom	previous bade

lable 5 – continued from previous page
KISSING_FACE
KISSING_FACE_WITH_CLOSED_EYES
KISSING_FACE_WITH_SMILING_EYES
KISS_MARK
KITCHEN_KNIFE
KITE
KIWI_FRUIT
KNOT
KOALA
LABEL
LAB_COAT
LACROSSE
LADDER
LADY_BEETLE
LAPTOP
LARGE_BLUE_DIAMOND
LARGE_ORANGE_DIAMOND
LAST_QUARTER_MOON
LAST_QUARTER_MOON_FACE
LAST_TRACK_BUTTON
LATIN_CROSS
LEAFY_GREEN
LEAF_FLUTTERING_IN_WIND
LEDGER
LEFT_ARROW
LEFT_ARROW_CURVING_RIGHT
continues on next page
Continues on next page

Table 5 – continued from p	previous page
LEFT_FACING_FIST	
LEFT_LUGGAGE	
LEFT_RIGHT_ARROW	
LEFT_SPEECH_BUBBLE	
LEG	
LEMON	
LEO	
LEOPARD	
LEVEL_SLIDER	
LIBRA	
LIGHT_BULB	
LIGHT_RAIL	
LIGHT_SKIN_TONE	
LINK	
LINKED_PAPERCLIPS	
LION	
LIPSTICK	
LITTER_IN_BIN_SIGN	
LIZARD	
LLAMA	
LOBSTER	
LOCKED	
LOCKED_WITH_KEY	
LOCKED_WITH_PEN	
LOCOMOTIVE	
LOLLIPOP	
	continues on next page

Table 5 – continued from previous page	Table	5 –	continued	from	previous	page
--	-------	-----	-----------	------	----------	------

Table 5 – continued from previous page	
LONG_DRUM	
LOTION_BOTTLE	
LOUDLY_CRYING_FACE	
LOUDSPEAKER	
LOVE_HOTEL	
LOVE_LETTER	
LOVE_YOU_GESTURE	
LUGGAGE	
LUNGS	
LYING_FACE	
MAGE	
MAGIC_WAND	
MAGNET	
MAGNIFYING_GLASS_TILTED_LEFT	
MAGNIFYING_GLASS_TILTED_RIGHT	
MAHJONG_RED_DRAGON	
MALE_SIGN	
MAMMOTH	
MAN	
MANGO	
MANS_SHOE	
MANTELPIECE_CLOCK	
MANUAL_WHEELCHAIR	
MAN_BEARD	
MAN_DANCING	
MAPLE_LEAF	
HAL EL_ELAT	
	continues on next page

Table 5	5 – continued	from previous page	è
---------	---------------	--------------------	---

MAP_OF_JAPAN  MARTIAL_ARTS_UNIFORM  MATE  MEAT_ON_BONE  MECHANICAL_ARM  MECHANICAL_LEG  MEDICAL_SYMBOL  MEDIUM_DARK_SKIN_TONE  MEDIUM_LIGHT_SKIN_TONE	ble 5 – continued from previous page
MATE  MEAT_ON_BONE  MECHANICAL_ARM  MECHANICAL_LEG  MEDICAL_SYMBOL  MEDIUM_DARK_SKIN_TONE	
MEAT_ON_BONE  MECHANICAL_ARM  MECHANICAL_LEG  MEDICAL_SYMBOL  MEDIUM_DARK_SKIN_TONE	
MECHANICAL_ARM  MECHANICAL_LEG  MEDICAL_SYMBOL  MEDIUM_DARK_SKIN_TONE	
MECHANICAL_LEG  MEDICAL_SYMBOL  MEDIUM_DARK_SKIN_TONE	
MEDICAL_SYMBOL  MEDIUM_DARK_SKIN_TONE	
MEDIUM_DARK_SKIN_TONE	
MEDIUM_LIGHT_SKIN_TONE	
MEDIUM_SKIN_TONE	
MEGAPHONE	
MELON	
MEMO	
MENORAH	
MENS_ROOM	
MEN_HOLDING_HANDS	
MERPERSON	
METRO	
MICROBE	
MICROPHONE	
MICROSCOPE	
MIDDLE_FINGER	
MILITARY_HELMET	
MILITARY_MEDAL	
MILKY_WAY	
MINIBUS	

Table 5 – continued from previous page	Table	5 –	continued	from	previous	page
--	-------	-----	-----------	------	----------	------

	Table 5 – continued from previous page
MINUS	
MIRROR	
MOAI	
MOBILE_PHONE	
MOBILE_PHONE_OFF	
MOBILE_PHONE_WITH_ARROW	
MONEY_BAG	
MONEY_MOUTH_FACE	
MONEY_WITH_WINGS	
MONKEY	
MONKEY_FACE	
MONORAIL	
MOON_CAKE	
MOON_VIEWING_CEREMONY	
MOSQUE	
MOSQUITO	
MOTORCYCLE	
MOTORIZED_WHEELCHAIR	
MOTORWAY	
MOTOR_BOAT	
MOTOR_SCOOTER	
MOUNTAIN	
MOUNTAIN_CABLEWAY	
MOUNTAIN_RAILWAY	
MOUNT_FUJI	
MOUSE	
	continues on next page

Table 5	5 – continued	from previous page	è
---------	---------------	--------------------	---

	Table 5 – continued from previous page	
MOUSE_FACE		
MOUSE_TRAP		
MOUTH		
MOVIE_CAMERA		
MRS_CLAUS		
MULTIPLY		
MUSHROOM		
MUSICAL_KEYBOARD		
MUSICAL_NOTE		
MUSICAL_NOTES		
MUSICAL_SCORE		
MUTED_SPEAKER		
NAIL_POLISH		
NAME_BADGE		
NATIONAL_PARK		
NAUSEATED_FACE		
NAZAR_AMULET		
NECKTIE		
NERD_FACE		
NESTING_DOLLS		
NEUTRAL_FACE		
NEWSPAPER		
NEW_BUTTON		
NEW_MOON		
NEW_MOON_FACE		
NEXT_TRACK_BUTTON		
		continues on next page

Table 5 – continued from previous page	Table	5 –	continued	from	previous	page
--	-------	-----	-----------	------	----------	------

Table 5 – continued from previous page
NG_BUTTON
NIGHT_WITH_STARS
NINE_OCLOCK
NINE_THIRTY
NINJA
NON_POTABLE_WATER
NOSE
NOTEBOOK
NOTEBOOK_WITH_DECORATIVE_COVER
NO_BICYCLES
NO_ENTRY
NO_LITTERING
NO_MOBILE_PHONES
NO_ONE_UNDER_EIGHTEEN
NO_PEDESTRIANS
NO_SMOKING
NUT_AND_BOLT
OCTOPUS
ODEN
OFFICE_BUILDING
OGRE
OIL_DRUM
OK_BUTTON
OK_HAND
OLDER_PERSON
OLD_KEY
continues on next page

Table 5	<ul><li>continued</li></ul>	from	previous	page
---------	-----------------------------	------	----------	------

Table 5 – continued from previous page	
OLD_MAN	
OLD_WOMAN	
OLIVE	
OM	
ONCOMING_AUTOMOBILE	
ONCOMING_BUS	
ONCOMING_FIST	
ONCOMING_POLICE_CAR	
ONCOMING_TAXI	
ONE_OCLOCK	
ONE_PIECE_SWIMSUIT	
ONE_THIRTY	
ONION	
ON_ARROW	
OPEN_BOOK	
OPEN_FILE_FOLDER	
OPEN_HANDS	
OPEN_MAILBOX_WITH_LOWERED_FLAG	
OPEN_MAILBOX_WITH_RAISED_FLAG	
OPHIUCHUS	
OPTICAL_DISK	
ORANGE_BOOK	
ORANGE_CIRCLE	
ORANGE_HEART	
ORANGE_SQUARE	
ORANGUTAN	
	continues on next page

Table 5 – continued from previous page

	Table 5 – continued from previous page
ORTHODOX_CROSS	
OTTER	
OUTBOX_TRAY	
OWL	
OX	
OYSTER	
O_BUTTON_BLOOD_TYPE	
PACKAGE	
PAGER	
PAGE_FACING_UP	
PAGE_WITH_CURL	
PAINTBRUSH	
PALMS_UP_TOGETHER	
PALM_TREE	
PANCAKES	
PANDA	
PAPERCLIP	
PARACHUTE	
PARROT	
PARTYING_FACE	
PARTY_POPPER	
PART_ALTERNATION_MARK	
PASSENGER_SHIP	
PASSPORT_CONTROL	
PAUSE_BUTTON	
PAW_PRINTS	
	continues on next page

Table 5 – continued from previous page

	Table 5 – continued from previous page	
PEACE_SYMBOL		
PEACH		
PEACOCK		
PEANUTS		
PEAR		
PEN		
PENCIL		
PENGUIN		
PENSIVE_FACE		
PEOPLE_HUGGING		
PEOPLE_WITH_BUNNY_EARS		
PEOPLE_WRESTLING		
PERFORMING_ARTS		
PERSEVERING_FACE		
PERSON		
PERSON_BIKING		
PERSON_BLOND_HAIR		
PERSON_BOUNCING_BALL		
PERSON_BOWING		
PERSON_CARTWHEELING		
PERSON_CLIMBING		
PERSON_FACEPALMING		
PERSON_FENCING		
PERSON_FROWNING		
PERSON_GESTURING_NO		
PERSON_GESTURING_OK		
	continue	s on next page

Table 5 – continued from previous page

PERSON\_GETTING\_HAIRCUT PERSON\_GETTING\_MASSAGE PERSON\_GOLFING PERSON\_IN\_BED PERSON\_IN\_LOTUS\_POSITION PERSON\_IN\_STEAMY\_ROOM PERSON\_IN\_SUIT\_LEVITATING PERSON\_IN\_TUXEDO PERSON\_JUGGLING PERSON\_KNEELING PERSON\_LIFTING\_WEIGHTS PERSON\_MOUNTAIN\_BIKING PERSON\_PLAYING\_HANDBALL PERSON\_PLAYING\_WATER\_POLO PERSON\_POUTING PERSON\_RAISING\_HAND PERSON\_ROWING\_BOAT PERSON\_RUNNING PERSON\_SHRUGGING PERSON\_STANDING PERSON\_SURFING PERSON\_SWIMMING PERSON\_TAKING\_BATH PERSON\_TIPPING\_HAND PERSON\_WALKING PERSON\_WEARING\_TURBAN

continues on next page

Table 5 – continued from previous page

	Table 5 – continued from previous page	
PERSON_WITH_SKULLCAP		
PERSON_WITH_VEIL		
PETRI_DISH		
PICK		
PICKUP_TRUCK		
PIE		
PIG		
PIG_FACE		
PIG_NOSE		
PILE_OF_POO		
PILL		
PINCHED_FINGERS		
PINCHING_HAND		
PINEAPPLE		
PINE_DECORATION		
PING_PONG		
PISCES		
PISTOL		
PIZZA		
PIñATA		
PLACARD		
PLACE_OF_WORSHIP		
PLAY_BUTTON		
PLAY_OR_PAUSE_BUTTON		
PLEADING_FACE		
PLUNGER		
		continues on next page

Table 5 – continued from previous page

	rable 5 – continued from previous page
PLUS	
POLICE_CAR	
POLICE_CAR_LIGHT	
POLICE_OFFICER	
POODLE	
POPCORN	
POSTAL_HORN	
POSTBOX	
POST_OFFICE	
POTABLE_WATER	
POTATO	
POTTED_PLANT	
POT_OF_FOOD	
POULTRY_LEG	
POUND_BANKNOTE	
POUTING_CAT	
POUTING_FACE	
PRAYER_BEADS	
PREGNANT_WOMAN	
PRETZEL	
PRINCE	
PRINCESS	
PRINTER	
PROHIBITED	
PURPLE_CIRCLE	
PURPLE_HEART	
	continues on next page

Table 5 – continu	led from previous page
PURPLE_SQUARE	
PURSE	
PUSHPIN	
PUZZLE_PIECE	
P_BUTTON	
QUESTION_MARK	
RABBIT	
RABBIT_FACE	
RACCOON	
RACING_CAR	
RADIO	
RADIOACTIVE	
RADIO_BUTTON	
RAILWAY_CAR	
RAILWAY_TRACK	
RAINBOW	
RAISED_BACK_OF_HAND	
RAISED_FIST	
RAISED_HAND	
RAISING_HANDS	
RAM	
RAT	
RAZOR	
RECEIPT	
RECORD_BUTTON	
RECYCLING_SYMBOL	
	continues on next page

Table	5 –	continued	from	previous	nage
IUDIC	•	COLLINIACA	110111	picvious	page

Table 5 – continued from previous page	
RED_APPLE	
RED_CIRCLE	
RED_ENVELOPE	
RED_HAIR	
RED_HEART	
RED_PAPER_LANTERN	
RED_SQUARE	
RED_TRIANGLE_POINTED_DOWN	
RED_TRIANGLE_POINTED_UP	
REGISTERED	
RELIEVED_FACE	
REMINDER_RIBBON	
REPEAT_BUTTON	
REPEAT_SINGLE_BUTTON	
RESCUE_WORKERS_HELMET	
RESTROOM	
REVERSE_BUTTON	
REVOLVING_HEARTS	
RHINOCEROS	
RIBBON	
RICE_BALL	
RICE_CRACKER	
RIGHT_ANGER_BUBBLE	
RIGHT_ARROW	
RIGHT_ARROW_CURVING_DOWN	
RIGHT_ARROW_CURVING_LEFT	
	continues on next page

Table 5	5 – continued	from previous page	è
---------	---------------	--------------------	---

 $RIGHT\_ARROW\_CURVING\_UP$ RIGHT\_FACING\_FIST RING RINGED\_PLANET ROASTED\_SWEET\_POTATO ROBOTROCK ROCKET ROLLED\_UP\_NEWSPAPER ROLLER\_COASTER ROLLER\_SKATE ROLLING\_ON\_THE\_FLOOR\_LAUGHING ROLL\_OF\_PAPER ROOSTER **ROSE** ROSETTE ROUND\_PUSHPIN RUGBY\_FOOTBALL RUNNING\_SHIRT RUNNING\_SHOE  $SAD\_BUT\_RELIEVED\_FACE$ SAFETY\_PIN SAFETY\_VEST **SAGITTARIUS** SAILBOAT SAKE

Table 5 – continued from previous page

	rable 5 – continued from previous page	
SALT		
SANDWICH		
SANTA_CLAUS		
SARI		
SATELLITE		
SATELLITE_ANTENNA		
SAUROPOD		
SAXOPHONE		
SCARF		
SCHOOL		
SCISSORS		
SCORPIO		
SCORPION		
SCREWDRIVER		
SCROLL		
SEAL		
SEAT		
SECOND_PLACE_MEDAL		
SEEDLING		
SEE_NO_EVIL_MONKEY		
SELFIE		
SEVEN_OCLOCK		
SEVEN_THIRTY		
SEWING_NEEDLE		
SHALLOW_PAN_OF_FOOD		
SHAMROCK		
		continues on next page

	Table 5 – continued from previous page
SHARK	
SHAVED_ICE	
SHEAF_OF_RICE	
SHIELD	
SHINTO_SHRINE	
SHIP	
SHOOTING_STAR	
SHOPPING_BAGS	
SHOPPING_CART	
SHORTCAKE	
SHORTS	
SHOWER	
SHRIMP	
SHUFFLE_TRACKS_BUTTON	
SHUSHING_FACE	
SIGN_OF_THE_HORNS	
SIX_OCLOCK	
SIX_THIRTY	
SKATEBOARD	
SKIER	
SKIS	
SKULL	
SKULL_AND_CROSSBONES	
SKUNK	
SLED	
SLEEPING_FACE	
	continues on next page

Table 5 – continued from previous page

SLEEPY\_FACE SLIGHTLY\_FROWNING\_FACE SLIGHTLY\_SMILING\_FACE SLOTH SLOT\_MACHINE SMALL\_AIRPLANE SMALL\_BLUE\_DIAMOND SMALL\_ORANGE\_DIAMOND SMILING\_CAT\_WITH\_HEART\_EYES SMILING\_FACE SMILING\_FACE\_WITH\_HALO SMILING\_FACE\_WITH\_HEARTS SMILING\_FACE\_WITH\_HEART\_EYES SMILING\_FACE\_WITH\_HORNS SMILING\_FACE\_WITH\_SMILING\_EYES SMILING\_FACE\_WITH\_SUNGLASSES SMILING\_FACE\_WITH\_TEAR SMIRKING FACE SNAIL **SNAKE** SNEEZING\_FACE **SNOWBOARDER** SNOWFLAKE **SNOWMAN** SNOWMAN\_WITHOUT\_SNOW SNOW\_CAPPED\_MOUNTAIN

continues on next page

Table 5	5 – continued	from previous page	è
---------	---------------	--------------------	---

	Table 5 – continued from previous page	
SOAP		
SOCCER_BALL		
SOCKS		
SOFTBALL		
SOFT_ICE_CREAM		
SOON_ARROW		
SOS_BUTTON		
SPADE_SUIT		
SPAGHETTI		
SPARKLE		
SPARKLER		
SPARKLES		
SPARKLING_HEART		
SPEAKER_HIGH_VOLUME		
SPEAKER_LOW_VOLUME		
SPEAKER_MEDIUM_VOLUME		
SPEAKING_HEAD		
SPEAK_NO_EVIL_MONKEY		
SPEECH_BALLOON		
SPEEDBOAT		
SPIDER		
SPIDER_WEB		
SPIRAL_CALENDAR		
SPIRAL_NOTEPAD		
SPIRAL_SHELL		
SPONGE		
	continues on next of	200

Table 5 – continued from previous page

Table 5 – continued from previous page	
SPOON	
SPORTS_MEDAL	
SPORT_UTILITY_VEHICLE	
SPOUTING_WHALE	
SQUID	
SQUINTING_FACE_WITH_TONGUE	
STADIUM	
STAR	
STAR_AND_CRESCENT	
STAR_OF_DAVID	
STAR_STRUCK	
STATION	
STATUE_OF_LIBERTY	
STEAMING_BOWL	
STETHOSCOPE	
STOPWATCH	
STOP_BUTTON	
STOP_SIGN	
STRAIGHT_RULER	
STRAWBERRY	
STUDIO_MICROPHONE	
STUFFED_FLATBREAD	
SUN	
SUNFLOWER	
SUNGLASSES	
SUNRISE	
continu	es on next page

Table	5 –	continued	from	previous	page

SUNRISE\_OVER\_MOUNTAINS **SUNSET**  $SUN\_BEHIND\_CLOUD$ SUN\_BEHIND\_LARGE\_CLOUD SUN\_BEHIND\_RAIN\_CLOUD SUN\_BEHIND\_SMALL\_CLOUD SUN\_WITH\_FACE **SUPERHERO** SUPERVILLAIN SUSHI SUSPENSION\_RAILWAY **SWAN** SWEAT\_DROPLETS **SYNAGOGUE SYRINGE TACO** TAKEOUT\_BOX **TAMALE** TANABATA\_TREE **TANGERINE TAURUS** TAXI TEACUP\_WITHOUT\_HANDLE **TEAPOT** TEAR\_OFF\_CALENDAR TEDDY\_BEAR

Table 5 – continued from previous page

	Table 5 – continued from previous page	
TELEPHONE		
TELEPHONE_RECEIVER		
TELESCOPE		
TELEVISION		
TENNIS		
TENT		
TEN_OCLOCK		
TEN_THIRTY		
TEST_TUBE		
THERMOMETER		
THINKING_FACE		
THIRD_PLACE_MEDAL		
THONG_SANDAL		
THOUGHT_BALLOON		
THREAD		
THREE_OCLOCK		
THREE_THIRTY		
THUMBS_DOWN		
THUMBS_UP		
TICKET		
TIGER		
TIGER_FACE		
TIMER_CLOCK		
TIRED_FACE		
TOILET		
TOKYO_TOWER		
	C	ontinues on next page

Table 5 – continued from previous page

	Table 5 – continued from previous page	
TOMATO		
TONGUE		
TOOLBOX		
ТООТН		
TOOTHBRUSH		
TOP_ARROW		
TOP_HAT		
TORNADO		
TRACKBALL		
TRACTOR		
TRADE_MARK		
TRAIN		
TRAM		
TRAM_CAR		
TRANSGENDER_SYMBOL		
TRIANGULAR_FLAG		
TRIANGULAR_RULER		
TRIDENT_EMBLEM		
TROLLEYBUS		
ТПОРНУ		
TROPICAL_DRINK		
TROPICAL_FISH		
TRUMPET		
TULIP		
TUMBLER_GLASS		
TURKEY		
		continues on next page

Table 5 – continued from previous page

	Table 3 – Continued Ironi previot	ao pago
TURTLE		
TWELVE_OCLOCK		
TWELVE_THIRTY		
TWO_HEARTS		
TWO_HUMP_CAMEL		
TWO_OCLOCK		
TWO_THIRTY		
T_REX		
T_SHIRT		
UMBRELLA		
UMBRELLA_ON_GROUND		
UMBRELLA_WITH_RAIN_DROPS		
UNAMUSED_FACE		
UNICORN		
UNLOCKED		
UPSIDE_DOWN_FACE		
UPWARDS_BUTTON		
UP_ARROW		
UP_BUTTON		
UP_DOWN_ARROW		
UP_LEFT_ARROW		
UP_RIGHT_ARROW		
VAMPIRE		
VERTICAL_TRAFFIC_LIGHT		
VIBRATION_MODE		
VICTORY_HAND		
		continues on next page

continues on next page

Table 5 – continued from previous page

	Table 5 – continued from previous page	
VIDEOCASSETTE		
VIDEO_CAMERA		
VIDEO_GAME		
VIOLIN		
VIRGO		
VOLCANO		
VOLLEYBALL		
VS_BUTTON		
VULCAN_SALUTE		
WAFFLE		
WANING_CRESCENT_MOON		
WANING_GIBBOUS_MOON		
WARNING		
WASTEBASKET		
WATCH		
WATERMELON		
WATER_BUFFALO		
WATER_CLOSET		
WATER_WAVE		
WAVING_HAND		
WAVY_DASH		
WAXING_CRESCENT_MOON		
WAXING_GIBBOUS_MOON		
WEARY_CAT		
WEARY_FACE		
WEDDING		
		continues on next page

continues on next page

Table 5 – continued from previous page

WHEELCHAIR_SYMBOL WHEEL_OF_DHARMA WHITE_CANE WHITE_CIRCLE WHITE_EXCLAMATION_MARK WHITE_FLAG WHITE_FLOWER WHITE_HAIR WHITE_HAIR WHITE_HARF WHITE_HARF WHITE_HARF WHITE_LARGE_SQUARE WHITE_MEDIUM_SWALL_SQUARE WHITE_MEDIUM_SQUARE WHITE_SMALL_SQUARE WHITE_SVARE_BUTTON WILTED_FLOWER WHOOW WIND_CHIME WIND_FACE WINE_GLASS WINKING_FACE_WITH_TONGUE WOOMAN WOMANS_BOOT	Table 5	- continued from previous page	
WHEEL_OF_DHARMA WHITE_CANE WHITE_CIRCLE WHITE_EXCLAMATION_MARK WHITE_FLAG WHITE_FLAG WHITE_FLOWER WHITE_HAIR WHITE_HARR WHITE_HEART WHITE_LARGE_SQUARE WHITE_MEDIUM_SMALL_SQUARE WHITE_MEDIUM_SQUARE WHITE_MEDIUM_SQUARE WHITE_SQUARE_BUTTON WILTED_FLOWER WINDOW WIND_CHIME WIND_FACE WINE_GLASS WINKING_FACE_WITH_TONGUE WOLF WOMAN WOMANS_BOOT	WHALE		
WHITE_CANE WHITE_EXCLAMATION_MARK WHITE_FLAG WHITE_FLAG WHITE_FLOWER WHITE_HAIR WHITE_HEART WHITE_HEART WHITE_MEDIUM_SNALL_SQUARE WHITE_MEDIUM_SNALL_SQUARE WHITE_MEDIUM_SQUARE WHITE_SMALL_SQUARE WHITE_SMALL_SQUARE WHITE_SQUARE_BUTTON WILTED_FLOWER WINDOW WIND_CHIME WIND_CHIME WIND_FACE WINKING_FACE_WITH_TONGUE WOMAN WOMANS_BOOT	WHEELCHAIR_SYMBOL		
WHITE_EXCLAMATION_MARK WHITE_FLAG WHITE_FLOWER WHITE_HAIR WHITE_HAIR WHITE_HEART WHITE_HEART WHITE_MEDIUM_SMALL_SQUARE WHITE_MEDIUM_SMALL_SQUARE WHITE_MEDIUM_SQUARE WHITE_SMALL_SQUARE WHITE_SMALL_SQUARE WHITE_SMALL_SQUARE WHITE_SQUARE_BUTTON WILTED_FLOWER WINDOW WIND_CHIME WIND_FACE WINE_GLASS WINKING_FACE_WITH_TONGUE WOLF WOMAN WOMANS_BOOT	WHEEL_OF_DHARMA		
WHITE_FLAG WHITE_FLOWER WHITE_HAIR WHITE_HAAR WHITE_HEART WHITE_LARGE_SQUARE WHITE_MEDIUM_SMALL_SQUARE WHITE_MEDIUM_SQUARE WHITE_MEDIUM_SQUARE WHITE_SQUARE WHITE_SQUARE_BUTTON WILTED_FLOWER WINDOW WIND_CHIME WIND_FACE WINE_GLASS WINKING_FACE_WITH_TONGUE WOMAN WOMANS_BOOT	WHITE_CANE		
WHITE_FLAG  WHITE_HAIR  WHITE_HEART  WHITE_HEART  WHITE_MEDIUM_SMALL_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_SMALL_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE_WITH_TONGUE  WOMAN  WOMANS_BOOT	WHITE_CIRCLE		
WHITE_FLOWER  WHITE_HEART  WHITE_LARGE_SQUARE  WHITE_MEDIUM_SMALL_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE_WITH_TONGUE  WOMANN  WOMANS_BOOT	WHITE_EXCLAMATION_MARK		
WHITE_HAIR  WHITE_HEART  WHITE_LARGE_SQUARE  WHITE_MEDIUM_SMALL_SQUARE  WHITE_QUESTION_MARK  WHITE_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE_WITH_TONGUE  WOMAN  WOMANS_BOOT	WHITE_FLAG		
WHITE_HEART  WHITE_LARGE_SQUARE  WHITE_MEDIUM_SMALL_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_QUESTION_MARK  WHITE_SMALL_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE_WITH_TONGUE  WOMAN  WOMANS_BOOT	WHITE_FLOWER		
WHITE_LARGE_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_MEDIUM_SQUARE  WHITE_QUESTION_MARK  WHITE_SMALL_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE_WITH_TONGUE  WOMAN  WOMANS_BOOT	WHITE_HAIR		
WHITE_MEDIUM_SQUARE  WHITE_QUESTION_MARK  WHITE_SMALL_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE_WITH_TONGUE  WOMAN  WOMANS_BOOT	WHITE_HEART		
WHITE_MEDIUM_SQUARE  WHITE_SMALL_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE_WITH_TONGUE  WOMAN  WOMANS_BOOT	WHITE_LARGE_SQUARE		
WHITE_QUESTION_MARK  WHITE_SMALL_SQUARE  WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE  WOLF  WOMAN  WOMANS_BOOT	WHITE_MEDIUM_SMALL_SQUARE		
WHITE_SQUARE WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE  WINKING_FACE  WOMAN  WOMANS_BOOT	WHITE_MEDIUM_SQUARE		
WHITE_SQUARE_BUTTON  WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE  WOLF  WOMAN  WOMANS_BOOT	WHITE_QUESTION_MARK		
WILTED_FLOWER  WINDOW  WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE  WOLF  WOMAN  WOMANS_BOOT	WHITE_SMALL_SQUARE		
WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE  WOMAN  WOMANS_BOOT	WHITE_SQUARE_BUTTON		
WIND_CHIME  WIND_FACE  WINE_GLASS  WINKING_FACE  WINKING_FACE_WITH_TONGUE  WOLF  WOMAN  WOMANS_BOOT	WILTED_FLOWER		
WIND_FACE WINE_GLASS WINKING_FACE WINKING_FACE_WITH_TONGUE WOLF WOMAN WOMANS_BOOT	WINDOW		
WINE_GLASS  WINKING_FACE  WINKING_FACE_WITH_TONGUE  WOLF  WOMAN  WOMANS_BOOT	WIND_CHIME		
WINKING_FACE WINKING_FACE_WITH_TONGUE  WOLF  WOMAN  WOMANS_BOOT	WIND_FACE		
WINKING_FACE_WITH_TONGUE  WOLF  WOMAN  WOMANS_BOOT	WINE_GLASS		
WOMAN WOMANS_BOOT	WINKING_FACE		
WOMANS_BOOT	WINKING_FACE_WITH_TONGUE		
WOMANS_BOOT	WOLF		
	WOMAN		
continues on next neces	WOMANS_BOOT		
			continues on next page

continues on next page

Table	5 –	continued	from	previous	nage
IUDIC	•	COLLINIACA	110111	picvious	page

Table 5 – continued from previous page	
WOMANS_CLOTHES	
WOMANS_HAT	
WOMANS_SANDAL	
WOMAN_AND_MAN_HOLDING_HANDS	
WOMAN_DANCING	
WOMAN_WITH_HEADSCARF	
WOMENS_ROOM	
WOMEN_HOLDING_HANDS	
WOOD	
WOOZY_FACE	
WORLD_MAP	
WORM	
WORRIED_FACE	
WRAPPED_GIFT	
WRENCH	
WRITING_HAND	
YARN	
YAWNING_FACE	
YELLOW_CIRCLE	
YELLOW_HEART	
YELLOW_SQUARE	
YEN_BANKNOTE	
YIN_YANG	
YO_YO	
ZANY_FACE	
ZEBRA	
CC	entinues on next page

Table 5 – continued from previous page

```
ZIPPER_MOUTH_FACE

ZOMBIE

ZZZ
```

```
ABACUS = ''
AB_BUTTON_BLOOD_TYPE = ''
ACCORDION = ''
ADHESIVE_BANDAGE = ''
ADMISSION_TICKETS = ''
AERIAL_TRAMWAY = ''
AIRPLANE = ''
AIRPLANE_ARRIVAL = ''
AIRPLANE_DEPARTURE = ''
ALARM_CLOCK = ''
ALEMBIC = ''
ALIEN = ''
ALIEN_MONSTER = ''
AMBULANCE = ''
AMERICAN_FOOTBALL = ''
AMPHORA = ''
ANATOMICAL_HEART = ''
ANCHOR = ''
ANGER_SYMBOL = ''
ANGRY_FACE = ''
ANGRY_FACE_WITH_HORNS = ''
ANGUISHED_FACE = ''
ANT = ''
ANTENNA_BARS = ''
ANXIOUS_FACE_WITH_SWEAT = ''
AQUARIUS = ''
```

```
ARIES = ''
ARTICULATED_LORRY = ''
ARTIST_PALETTE = ''
ASTONISHED_FACE = ''
ATM_SIGN = ''
ATOM_SYMBOL = ''
AUTOMOBILE = ''
AUTO_RICKSHAW = ''
AVOCADO = ''
AXE = ''
A_BUTTON_BLOOD_TYPE = ''
BABY = ''
BABY_ANGEL = ''
BABY_BOTTLE = ''
BABY_CHICK = ''
BABY_SYMBOL = ''
BACKHAND_INDEX_POINTING_DOWN = ''
BACKHAND_INDEX_POINTING_LEFT = ''
BACKHAND_INDEX_POINTING_RIGHT = ''
BACKHAND_INDEX_POINTING_UP = ''
BACKPACK = ''
BACK_ARROW = ''
BACON = ''
BADGER = ''
BADMINTON = ''
BAGEL = ''
BAGGAGE_CLAIM = ''
BAGUETTE\_BREAD = ''
BALANCE_SCALE = ''
BALD = ''
BALL = ''
```

```
BALLET_SHOES = ''
BALLOON = ''
BALLOT_BOX_WITH_BALLOT = ''
BANANA = ''
BANJO = ''
BANK = ''
BARBER_POLE = ''
BAR_CHART = ''
BASEBALL = ''
BASKET = ''
BASKETBALL = ''
BAT = ''
BATHTUB = ''
BATTERY = ''
BEACH_WITH_UMBRELLA = ''
BEAMING_FACE_WITH_SMILING_EYES = ''
BEAR = ''
BEATING_HEART = ''
BEAVER = ''
BED = ''
BEER_MUG = ''
BEETLE = ''
BELL = ''
BELLHOP_BELL = ''
BELL_PEPPER = ''
BELL_WITH_SLASH = ''
BENTO_BOX = ''
BEVERAGE_BOX = ''
BICYCLE = ''
BIKINI = ''
BILLED_CAP = ''
```

```
BIOHAZARD = ''
BIRD = ''
BIRTHDAY_CAKE = ''
BISON = ''
BLACK_CIRCLE = ''
BLACK_FLAG = ''
BLACK_HEART = ''
BLACK_LARGE_SQUARE = ''
BLACK_MEDIUM_SMALL_SQUARE = ''
BLACK_MEDIUM_SQUARE = ''
BLACK_NIB = ''
BLACK_SMALL_SQUARE = ''
BLACK_SQUARE_BUTTON = ''
BLOSSOM = ''
BLOWFISH = ''
BLUEBERRIES = ''
BLUE_BOOK = ''
BLUE_CIRCLE = ''
BLUE_HEART = ''
BLUE_SQUARE = ''
BOAR = ''
BOMB = ''
BONE = ''
BOOKMARK = ''
BOOKMARK_TABS = ''
BOOKS = ''
BOOMERANG = ''
BOTTLE_WITH_POPPING_CORK = ''
BOUQUET = ''
BOWLING = ''
BOWL_WITH_SPOON = ''
```

```
BOW_AND_ARROW = ''
BOXING_GLOVE = ''
BOY = ''
BRAIN = ''
BREAD = ''
BREAST_FEEDING = ''
BRICK = ''
BRIDGE_AT_NIGHT = ''
BRIEFCASE = ''
BRIEFS = ''
BRIGHT_BUTTON = ''
BROCCOLI = ''
BROKEN_HEART = ''
BROOM = ''
BROWN_CIRCLE = ''
BROWN_HEART = ''
BROWN_SQUARE = ''
BUBBLE_TEA = ''
BUCKET = ''
BUG = ''
BUILDING_CONSTRUCTION = ''
BULLET_TRAIN = ''
BURRITO = ''
BUS = ''
BUSTS_IN_SILHOUETTE = ''
BUST_IN_SILHOUETTE = ''
BUS_STOP = ''
BUTTER = ''
BUTTERFLY = ''
B_BUTTON_BLOOD_TYPE = ''
CACTUS = ''
```

```
CALENDAR = ''
CALL_ME_HAND = ''
CAMEL = ''
CAMERA = ''
CAMERA_WITH_FLASH = ''
CAMPING = ''
CANCER = ''
CANDLE = ''
CANDY = ''
CANNED_FOOD = ''
CANOE = ''
CAPRICORN = ''
CARD_FILE_BOX = ''
CARD_INDEX = ''
CARD_INDEX_DIVIDERS = ''
CAROUSEL_HORSE = ''
CARPENTRY_SAW = ''
CARP_STREAMER = ''
CARROT = ''
CASTLE = ''
CAT = ''
CAT_FACE = ''
CAT_WITH_TEARS_OF_JOY = ''
CAT_WITH_WRY_SMILE = ''
CHAINS = ''
CHAIR = ''
CHART_DECREASING = ''
CHART_INCREASING = ''
CHART_INCREASING_WITH_YEN = ''
CHECK_BOX_WITH_CHECK = ''
CHECK_MARK = '√'
```

```
CHECK_MARK_BUTTON = ''
CHEESE_WEDGE = ''
CHEQUERED_FLAG = ''
CHERRIES = ''
CHERRY_BLOSSOM = ''
CHESS_PAWN = ''
CHESTNUT = ''
CHICKEN = ''
CHILD = ''
CHILDREN_CROSSING = ''
CHIPMUNK = ''
CHOCOLATE_BAR = ''
CHOPSTICKS = ''
CHRISTMAS_TREE = ''
CHURCH = ''
CIGARETTE = ''
CINEMA = ''
CIRCLED_M = ''
CIRCUS_TENT = ''
CITYSCAPE = ''
CITYSCAPE_AT_DUSK = ''
CLAMP = ''
CLAPPER_BOARD = ''
CLAPPING_HANDS = ''
CLASSICAL_BUILDING = ''
CLINKING_BEER_MUGS = ''
CLINKING_GLASSES = ''
CLIPBOARD = ''
CLOCKWISE_VERTICAL_ARROWS = ''
CLOSED_BOOK = ''
CLOSED_MAILBOX_WITH_LOWERED_FLAG = ''
```

```
CLOSED_MAILBOX_WITH_RAISED_FLAG = ''
CLOSED_UMBRELLA = ''
CLOUD = ''
CLOUD_WITH_LIGHTNING = ''
CLOUD_WITH_LIGHTNING_AND_RAIN = ''
CLOUD_WITH_RAIN = ''
CLOUD_WITH_SNOW = ''
CLOWN_FACE = ''
CLUB_SUIT = ''
CLUTCH_BAG = ''
CL_BUTTON = ''
COAT = ''
COCKROACH = ''
COCKTAIL_GLASS = ''
COCONUT = ''
COFFIN = ''
COIN = ''
COLD_FACE = ''
COLLISION = ''
COMET = ''
COMPASS = ''
COMPUTER_DISK = ''
COMPUTER_MOUSE = ''
CONFETTI_BALL = ''
CONFOUNDED_FACE = ''
CONFUSED_FACE = ''
CONSTRUCTION = ''
CONSTRUCTION_WORKER = ''
CONTROL_KNOBS = ''
CONVENIENCE_STORE = ''
COOKED_RICE = ''
```

```
COOKIE = ''
COOKING = ''
COOL_BUTTON = ''
COPYRIGHT = '0'
COUCH_AND_LAMP = ''
COUNTERCLOCKWISE_ARROWS_BUTTON = ''
COUPLE_WITH_HEART = ''
COW = ''
COWBOY_HAT_FACE = ''
COW_FACE = ''
CRAB = ''
CRAYON = ''
CREDIT_CARD = ''
CRESCENT_MOON = ''
CRICKET = ''
CRICKET_GAME = ''
CROCODILE = ''
CROISSANT = ''
CROSSED_FINGERS = ''
CROSSED_FLAGS = ''
CROSSED_SWORDS = ''
CROSS_MARK = ''
CROSS_MARK_BUTTON = ''
CROWN = ''
CRYING_CAT = ''
CRYING_FACE = ''
CRYSTAL_BALL = ''
CUCUMBER = ''
CUPCAKE = ''
CUP_WITH_STRAW = ''
CURLING_STONE = ''
```

```
CURLY_HAIR = ''
CURLY_LOOP = ''
CURRENCY_EXCHANGE = ''
CURRY_RICE = ''
CUSTARD = ''
CUSTOMS = ''
CUT_OF_MEAT = ''
CYCLONE = ''
DAGGER = ''
DANGO = ''
DARK_SKIN_TONE = ''
DASHING_AWAY = ''
DEAF_PERSON = ''
DECIDUOUS_TREE = ''
DEER = ''
DELIVERY_TRUCK = ''
DEPARTMENT_STORE = ''
DERELICT_HOUSE = ''
DESERT = ''
DESERT_ISLAND = ''
DESKTOP_COMPUTER = ''
DETECTIVE = ''
DIAMOND_SUIT = ''
DIAMOND_WITH_A_DOT = ''
DIM_BUTTON = ''
DIRECT_HIT = ''
DISAPPOINTED_FACE = ''
DISGUISED_FACE = ''
DIVIDE = ''
DIVING_MASK = ''
DIYA_LAMP = ''
```

```
DIZZY = ''
DIZZY_FACE = ''
DNA = ''
DODO = ''
DOG = ''
DOG_FACE = ''
DOLLAR_BANKNOTE = ''
DOLPHIN = ''
DOOR = ''
DOTTED_SIX_POINTED_STAR = ''
DOUBLE_CURLY_LOOP = ''
DOUBLE_EXCLAMATION_MARK = ''
DOUGHNUT = ''
DOVE = ''
DOWNCAST_FACE_WITH_SWEAT = ''
DOWNWARDS_BUTTON = ''
DOWN_ARROW = ''
DOWN_LEFT_ARROW = ''
DOWN_RIGHT_ARROW = ''
DRAGON = ''
DRAGON_FACE = ''
DRESS = ''
DROOLING_FACE = ''
DROPLET = ''
DROP_OF_BLOOD = ''
DRUM = ''
DUCK = ''
DUMPLING = ''
DVD = ''
EAGLE = ''
EAR = ''
```

```
EAR_OF_CORN = ''
EAR_WITH_HEARING_AID = ''
EGG = ''
EGGPLANT = ''
EIGHT_OCLOCK = ''
EIGHT_POINTED_STAR = ''
EIGHT_SPOKED_ASTERISK = ''
EIGHT_THIRTY = ''
EJECT_BUTTON = ''
ELECTRIC_PLUG = ''
ELEPHANT = ''
ELEVATOR = ''
ELEVEN_OCLOCK = ''
ELEVEN_THIRTY = ''
ELF = ''
END_ARROW = ''
ENVELOPE = ''
ENVELOPE_WITH_ARROW = ''
EURO_BANKNOTE = ''
EVERGREEN_TREE = ''
EWE = ''
EXCLAMATION_MARK = ''
EXCLAMATION_QUESTION_MARK = ''
EXPLODING_HEAD = ''
EXPRESSIONLESS_FACE = ''
EYE = ''
EYES = ''
E_MAIL = ''
FACE_BLOWING_A_KISS = ''
FACE_SAVORING_FOOD = ''
FACE_SCREAMING_IN_FEAR = ''
```

```
FACE_VOMITING = ''
FACE_WITHOUT_MOUTH = ''
FACE_WITH_HAND_OVER_MOUTH = ''
FACE_WITH_HEAD_BANDAGE = ''
FACE_WITH_MEDICAL_MASK = ''
FACE_WITH_MONOCLE = ''
FACE_WITH_OPEN_MOUTH = ''
FACE_WITH_RAISED_EYEBROW = ''
FACE_WITH_ROLLING_EYES = ''
FACE_WITH_STEAM_FROM_NOSE = ''
FACE_WITH_SYMBOLS_ON_MOUTH = ''
FACE_WITH_TEARS_OF_JOY = ''
FACE_WITH_THERMOMETER = ''
FACE_WITH_TONGUE = ''
FACTORY = ''
FAIRY = ''
FALAFEL = ''
FALLEN_LEAF = ''
FAMILY = ''
FAST_DOWN_BUTTON = ''
FAST_FORWARD_BUTTON = ''
FAST_REVERSE_BUTTON = ''
FAST_UP_BUTTON = ''
FAX_MACHINE = ''
FEARFUL_FACE = ''
FEATHER = ''
FEMALE_SIGN = ''
FERRIS_WHEEL = ''
FERRY = ''
FIELD_HOCKEY = ''
FILE_CABINET = ''
```

```
FILE_FOLDER = ''
FILM_FRAMES = ''
FILM_PROJECTOR = ''
FIRE = ''
FIRECRACKER = ''
FIREWORKS = ''
FIRE_ENGINE = ''
FIRE_EXTINGUISHER = ''
FIRST_PLACE_MEDAL = ''
FIRST_QUARTER_MOON = ''
FIRST_QUARTER_MOON_FACE = ''
FISH = ''
FISHING_POLE = ''
FISH_CAKE_WITH_SWIRL = ''
FIVE_OCLOCK = ''
FIVE_THIRTY = ''
FLAG_IN_HOLE = ''
FLAMINGO = ''
FLASHLIGHT = ''
FLATBREAD = ''
FLAT_SHOE = ''
FLEUR_DE_LIS = ''
FLEXED_BICEPS = ''
FLOPPY_DISK = ''
FLOWER_PLAYING_CARDS = ''
FLUSHED_FACE = ''
FLY = ''
FLYING_DISC = ''
FLYING_SAUCER = ''
FOG = ''
FOGGY = ''
```

```
FOLDED_HANDS = ''
FONDUE = ''
FOOT = ''
FOOTPRINTS = ''
FORK_AND_KNIFE = ''
FORK_AND_KNIFE_WITH_PLATE = ''
FORTUNE_COOKIE = ''
FOUNTAIN = ''
FOUNTAIN_PEN = ''
FOUR_LEAF_CLOVER = ''
FOUR_OCLOCK = ''
FOUR_THIRTY = ''
FOX = ''
FRAMED_PICTURE = ''
FREE_BUTTON = ''
FRENCH_FRIES = ''
FRIED_SHRIMP = ''
FROG = ''
FRONT_FACING_BABY_CHICK = ''
FROWNING_FACE = ''
FROWNING_FACE_WITH_OPEN_MOUTH = ''
FUEL_PUMP = ''
FULL_MOON = ''
FULL_MOON_FACE = ''
FUNERAL_URN = ''
GAME_DIE = ''
GARLIC = ''
GEAR = ''
GEMINI = ''
GEM_STONE = ''
GENIE = ''
```

```
GHOST = ''
GIRAFFE = ''
GIRL = ''
GLASSES = ''
GLASS_OF_MILK = ''
GLOBE_SHOWING_AMERICAS = ''
GLOBE_SHOWING_ASIA_AUSTRALIA = ''
GLOBE_SHOWING_EUROPE_AFRICA = ''
GLOBE_WITH_MERIDIANS = ''
GLOVES = ''
GLOWING_STAR = ''
GOAL_NET = ''
GOAT = ''
GOBLIN = ''
GOGGLES = ''
GORILLA = ''
GRADUATION_CAP = ''
GRAPES = ''
GREEN_APPLE = ''
GREEN_BOOK = ''
GREEN_CIRCLE = ''
GREEN_HEART = ''
GREEN_SALAD = ''
GREEN_SQUARE = ''
GRIMACING_FACE = ''
GRINNING_CAT = ''
GRINNING_CAT_WITH_SMILING_EYES = ''
GRINNING_FACE = ''
GRINNING_FACE_WITH_BIG_EYES = ''
GRINNING_FACE_WITH_SMILING_EYES = ''
GRINNING_FACE_WITH_SWEAT = ''
```

```
GRINNING_SQUINTING_FACE = ''
GROWING_HEART = ''
GUARD = ''
GUIDE_DOG = ''
GUITAR = ''
HAMBURGER = ''
HAMMER = ''
HAMMER_AND_PICK = ''
HAMMER_AND_WRENCH = ''
HAMSTER = ''
HANDBAG = ''
HANDSHAKE = ''
HAND_WITH_FINGERS_SPLAYED = ''
HATCHING_CHICK = ''
HEADPHONE = ''
HEADSTONE = ''
HEART_DECORATION = ''
HEART_EXCLAMATION = ''
HEART_SUIT = ''
HEART_WITH_ARROW = ''
HEART_WITH_RIBBON = ''
HEAR_NO_EVIL_MONKEY = ''
HEAVY_DOLLAR_SIGN = ''
HEDGEHOG = ''
HELICOPTER = ''
HERB = ''
HIBISCUS = ''
HIGH_HEELED_SHOE = ''
HIGH_SPEED_TRAIN = ''
HIGH_VOLTAGE = ''
HIKING_BOOT = ''
```

```
HINDU_TEMPLE = ''
HIPPOPOTAMUS = ''
HOLE = ''
HOLLOW_RED_CIRCLE = ''
HONEYBEE = ''
HONEY_POT = ''
HOOK = ''
HORIZONTAL_TRAFFIC_LIGHT = ''
HORSE = ''
HORSE_FACE = ''
HORSE_RACING = ''
HOSPITAL = ''
HOTEL = ''
HOT_BEVERAGE = ''
HOT_DOG = ''
HOT_FACE = ''
HOT_PEPPER = ''
HOT_SPRINGS = ''
HOURGLASS_DONE = ''
HOURGLASS_NOT_DONE = ''
HOUSE = ''
HOUSES = ''
HOUSE_WITH_GARDEN = ''
HUGGING_FACE = ''
HUNDRED_POINTS = ''
HUSHED_FACE = ''
HUT = ''
ICE = ''
ICE_CREAM = ''
ICE_HOCKEY = ''
ICE_SKATE = ''
```

```
ID_BUTTON = ''
INBOX_TRAY = ''
INCOMING_ENVELOPE = ''
INDEX_POINTING_UP = ''
INFINITY = ''
INFORMATION = ''
INPUT_LATIN_LETTERS = ''
INPUT_LATIN_LOWERCASE = ''
INPUT_LATIN_UPPERCASE = ''
INPUT_NUMBERS = ''
INPUT_SYMBOLS = ''
JACK_O_LANTERN = ''
JAPANESE_ACCEPTABLE_BUTTON = ''
JAPANESE_APPLICATION_BUTTON = ''
JAPANESE_BARGAIN_BUTTON = ''
JAPANESE_CASTLE = ''
JAPANESE_CONGRATULATIONS_BUTTON = ''
JAPANESE_DISCOUNT_BUTTON = ''
JAPANESE_DOLLS = ''
JAPANESE_FREE_OF_CHARGE_BUTTON = ''
JAPANESE_HERE_BUTTON = ''
JAPANESE_MONTHLY_AMOUNT_BUTTON = ''
JAPANESE_NOT_FREE_OF_CHARGE_BUTTON = ''
JAPANESE_NO_VACANCY_BUTTON = ''
JAPANESE_OPEN_FOR_BUSINESS_BUTTON = ''
JAPANESE_PASSING_GRADE_BUTTON = ''
JAPANESE_POST_OFFICE = ''
JAPANESE_PROHIBITED_BUTTON = ''
JAPANESE_RESERVED_BUTTON = ''
JAPANESE_SECRET_BUTTON = ''
JAPANESE_SERVICE_CHARGE_BUTTON = ''
```

```
JAPANESE_SYMBOL_FOR_BEGINNER = ''
JAPANESE_VACANCY_BUTTON = ''
JEANS = ''
JOKER = ''
JOYSTICK = ''
KAABA = ''
KANGAROO = ''
KEY = ''
KEYBOARD = ''
KICK_SCOOTER = ''
KIMONO = ''
KISS = ''
KISSING_CAT = ''
KISSING_FACE = ''
KISSING_FACE_WITH_CLOSED_EYES = ''
KISSING_FACE_WITH_SMILING_EYES = ''
KISS_MARK = ''
KITCHEN_KNIFE = ''
KITE = ''
KIWI_FRUIT = ''
KNOT = ''
KOALA = ''
LABEL = ''
LAB\_COAT = ''
LACROSSE = ''
LADDER = ''
LADY_BEETLE = ''
LAPTOP = ''
LARGE_BLUE_DIAMOND = ''
LARGE_ORANGE_DIAMOND = ''
LAST_QUARTER_MOON = ''
```

```
LAST_QUARTER_MOON_FACE = ''
LAST_TRACK_BUTTON = ''
LATIN_CROSS = ''
LEAFY_GREEN = ''
LEAF_FLUTTERING_IN_WIND = ''
LEDGER = ''
LEFT_ARROW = ''
LEFT_ARROW_CURVING_RIGHT = ''
LEFT_FACING_FIST = ''
LEFT_LUGGAGE = ''
LEFT_RIGHT_ARROW = ''
LEFT_SPEECH_BUBBLE = ''
LEG = ''
LEMON = ''
LEO = ''
LEOPARD = ''
LEVEL_SLIDER = ''
LIBRA = ''
LIGHT_BULB = ''
LIGHT_RAIL = ''
LIGHT_SKIN_TONE = ''
LINK = ''
LINKED_PAPERCLIPS = ''
LION = ''
LIPSTICK = ''
LITTER_IN_BIN_SIGN = ''
LIZARD = ''
LLAMA = ''
LOBSTER = ''
LOCKED = ''
LOCKED_WITH_KEY = ''
```

```
LOCKED_WITH_PEN = ''
LOCOMOTIVE = ''
LOLLIPOP = ''
LONG_DRUM = ''
LOTION_BOTTLE = ''
LOUDLY_CRYING_FACE = ''
LOUDSPEAKER = ''
LOVE_HOTEL = ''
LOVE_LETTER = ''
LOVE_YOU_GESTURE = ''
LUGGAGE = ''
LUNGS = ''
LYING_FACE = ''
MAGE = ''
MAGIC_WAND = ''
MAGNET = ''
MAGNIFYING_GLASS_TILTED_LEFT = ''
MAGNIFYING_GLASS_TILTED_RIGHT = ''
MAHJONG_RED_DRAGON = ''
MALE_SIGN = ''
MAMMOTH = ''
MAN = ''
MANGO = ''
MANS_SHOE = ''
MANTELPIECE_CLOCK = ''
MANUAL_WHEELCHAIR = ''
MAN_BEARD = ''
MAN_DANCING = ''
MAPLE_LEAF = ''
MAP_OF_JAPAN = ''
MARTIAL_ARTS_UNIFORM = ''
```

```
MATE = ''
MEAT_ON_BONE = ''
MECHANICAL_ARM = ''
MECHANICAL_LEG = ''
MEDICAL_SYMBOL = ''
MEDIUM_DARK_SKIN_TONE = ''
MEDIUM_LIGHT_SKIN_TONE = ''
MEDIUM_SKIN_TONE = ''
MEGAPHONE = ''
MELON = ''
MEMO = ''
MENORAH = ''
MENS_ROOM = ''
MEN_HOLDING_HANDS = ''
MERPERSON = ''
METRO = ''
MICROBE = ''
MICROPHONE = ''
MICROSCOPE = ''
MIDDLE_FINGER = ''
MILITARY_HELMET = ''
MILITARY_MEDAL = ''
MILKY_WAY = ''
MINIBUS = ''
MINUS = ''
MIRROR = ''
MOAI = ''
MOBILE_PHONE = ''
MOBILE_PHONE_OFF = ''
MOBILE_PHONE_WITH_ARROW = ''
MONEY_BAG = ''
```

```
MONEY_MOUTH_FACE = ''
MONEY_WITH_WINGS = ''
MONKEY = ''
MONKEY_FACE = ''
MONORAIL = ''
MOON_CAKE = ''
MOON_VIEWING_CEREMONY = ''
MOSQUE = ''
MOSQUITO = ''
MOTORCYCLE = ''
MOTORIZED_WHEELCHAIR = ''
MOTORWAY = ''
MOTOR_BOAT = ''
MOTOR_SCOOTER = ''
MOUNTAIN = ''
MOUNTAIN_CABLEWAY = ''
MOUNTAIN_RAILWAY = ''
MOUNT_FUJI = ''
MOUSE = ''
MOUSE_FACE = ''
MOUSE_TRAP = ''
MOUTH = ''
MOVIE_CAMERA = ''
MRS_CLAUS = ''
MULTIPLY = '×'
MUSHROOM = ''
MUSICAL_KEYBOARD = ''
MUSICAL_NOTE = ''
MUSICAL_NOTES = ''
MUSICAL_SCORE = ''
MUTED_SPEAKER = ''
```

204

```
NAIL_POLISH = ''
NAME_BADGE = ''
NATIONAL_PARK = ''
NAUSEATED_FACE = ''
NAZAR_AMULET = ''
NECKTIE = ''
NERD_FACE = ''
NESTING_DOLLS = ''
NEUTRAL_FACE = ''
NEWSPAPER = ''
NEW_BUTTON = ''
NEW_MOON = ''
NEW_MOON_FACE = ''
NEXT_TRACK_BUTTON = ''
NG_BUTTON = ''
NIGHT_WITH_STARS = ''
NINE_OCLOCK = ''
NINE_THIRTY = ''
NINJA = ''
NON_POTABLE_WATER = ''
NOSE = ''
NOTEBOOK = ''
NOTEBOOK_WITH_DECORATIVE_COVER = ''
NO_BICYCLES = ''
NO_ENTRY = ''
NO_LITTERING = ''
NO_MOBILE_PHONES = ''
NO_ONE_UNDER_EIGHTEEN = ''
NO_PEDESTRIANS = ''
NO_SMOKING = ''
NUT_AND_BOLT = ''
```

```
OCTOPUS = ''
ODEN = ''
OFFICE_BUILDING = ''
OGRE = ''
OIL_DRUM = ''
OK_BUTTON = ''
OK_HAND = ''
OLDER_PERSON = ''
OLD_KEY = ''
OLD_MAN = ''
OLD_WOMAN = ''
OLIVE = ''
OM = ''
ONCOMING_AUTOMOBILE = ''
ONCOMING_BUS = ''
ONCOMING_FIST = ''
ONCOMING_POLICE_CAR = ''
ONCOMING_TAXI = ''
ONE_OCLOCK = ''
ONE_PIECE_SWIMSUIT = ''
ONE_THIRTY = ''
ONION = ''
ON_ARROW = ''
OPEN_BOOK = ''
OPEN_FILE_FOLDER = ''
OPEN_HANDS = ''
OPEN_MAILBOX_WITH_LOWERED_FLAG = ''
OPEN_MAILBOX_WITH_RAISED_FLAG = ''
OPHIUCHUS = ''
OPTICAL_DISK = ''
ORANGE_BOOK = ''
```

```
ORANGE_CIRCLE = ''
ORANGE_HEART = ''
ORANGE_SQUARE = ''
ORANGUTAN = ''
ORTHODOX_CROSS = ''
OTTER = ''
OUTBOX_TRAY = ''
OWL = ''
OX = ''
OYSTER = ''
O_BUTTON_BLOOD_TYPE = ''
PACKAGE = ''
PAGER = ''
PAGE_FACING_UP = ''
PAGE_WITH_CURL = ''
PAINTBRUSH = ''
PALMS_UP_TOGETHER = ''
PALM_TREE = ''
PANCAKES = ''
PANDA = ''
PAPERCLIP = ''
PARACHUTE = ''
PARROT = ''
PARTYING_FACE = ''
PARTY_POPPER = ''
PART_ALTERNATION_MARK = ''
PASSENGER_SHIP = ''
PASSPORT_CONTROL = ''
PAUSE_BUTTON = ''
PAW_PRINTS = ''
PEACE_SYMBOL = ''
```

```
PEACH = ''
PEACOCK = ''
PEANUTS = ''
PEAR = ''
PEN = ''
PENCIL = ''
PENGUIN = ''
PENSIVE_FACE = ''
PEOPLE_HUGGING = ''
PEOPLE_WITH_BUNNY_EARS = ''
PEOPLE_WRESTLING = ''
PERFORMING_ARTS = ''
PERSEVERING_FACE = ''
PERSON = ''
PERSON_BIKING = ''
PERSON_BLOND_HAIR = ''
PERSON_BOUNCING_BALL = ''
PERSON_BOWING = ''
PERSON_CARTWHEELING = ''
PERSON_CLIMBING = ''
PERSON_FACEPALMING = ''
PERSON_FENCING = ''
PERSON_FROWNING = ''
PERSON_GESTURING_NO = ''
PERSON_GESTURING_OK = ''
PERSON_GETTING_HAIRCUT = ''
PERSON_GETTING_MASSAGE = ''
PERSON_GOLFING = ''
PERSON_IN_BED = ''
PERSON_IN_LOTUS_POSITION = ''
PERSON_IN_STEAMY_ROOM = ''
```

```
PERSON_IN_SUIT_LEVITATING = ''
PERSON_IN_TUXEDO = ''
PERSON_JUGGLING = ''
PERSON_KNEELING = ''
PERSON_LIFTING_WEIGHTS = ''
PERSON_MOUNTAIN_BIKING = ''
PERSON_PLAYING_HANDBALL = ''
PERSON_PLAYING_WATER_POLO = ''
PERSON_POUTING = ''
PERSON_RAISING_HAND = ''
PERSON_ROWING_BOAT = ''
PERSON_RUNNING = ''
PERSON_SHRUGGING = ''
PERSON_STANDING = ''
PERSON_SURFING = ''
PERSON_SWIMMING = ''
PERSON_TAKING_BATH = ''
PERSON_TIPPING_HAND = ''
PERSON_WALKING = ''
PERSON_WEARING_TURBAN = ''
PERSON_WITH_SKULLCAP = ''
PERSON_WITH_VEIL = ''
PETRI_DISH = ''
PICK = ''
PICKUP_TRUCK = ''
PIE = ''
PIG = ''
PIG_FACE = ''
PIG_NOSE = ''
PILE_OF_POO = ''
PILL = ''
```

```
PINCHED_FINGERS = ''
PINCHING_HAND = ''
PINEAPPLE = ''
PINE_DECORATION = ''
PING_PONG = ''
PISCES = ''
PISTOL = ''
PIZZA = ''
PIÑATA = ''
PLACARD = ''
PLACE_OF_WORSHIP = ''
PLAY_BUTTON = ''
PLAY_OR_PAUSE_BUTTON = ''
PLEADING_FACE = ''
PLUNGER = ''
PLUS = ''
POLICE_CAR = ''
POLICE_CAR_LIGHT = ''
POLICE_OFFICER = ''
POODLE = ''
POPCORN = ''
POSTAL_HORN = ''
POSTBOX = ''
POST_OFFICE = ''
POTABLE_WATER = ''
POTATO = ''
POTTED_PLANT = ''
POT_OF_FOOD = ''
POULTRY_LEG = ''
POUND_BANKNOTE = ''
POUTING_CAT = ''
```

```
POUTING_FACE = ''
PRAYER_BEADS = ''
PREGNANT_WOMAN = ''
PRETZEL = ''
PRINCE = ''
PRINCESS = ''
PRINTER = ''
PROHIBITED = ''
PURPLE_CIRCLE = ''
PURPLE_HEART = ''
PURPLE_SQUARE = ''
PURSE = ''
PUSHPIN = ''
PUZZLE_PIECE = ''
P_BUTTON = ''
QUESTION_MARK = ''
RABBIT = ''
RABBIT_FACE = ''
RACCOON = ''
RACING_CAR = ''
RADIO = ''
RADIOACTIVE = ''
RADIO_BUTTON = ''
RAILWAY_CAR = ''
RAILWAY_TRACK = ''
RAINBOW = ''
RAISED_BACK_OF_HAND = ''
RAISED_FIST = ''
RAISED_HAND = ''
RAISING_HANDS = ''
RAM = ''
```

```
RAT = ''
RAZOR = ''
RECEIPT = ''
RECORD_BUTTON = ''
RECYCLING_SYMBOL = ''
RED_APPLE = ''
RED_CIRCLE = ''
RED_ENVELOPE = ''
RED_HAIR = ''
RED_HEART = ''
RED_PAPER_LANTERN = ''
RED_SQUARE = ''
RED_TRIANGLE_POINTED_DOWN = ''
RED_TRIANGLE_POINTED_UP = ''
REGISTERED = '®'
RELIEVED_FACE = ''
REMINDER_RIBBON = ''
REPEAT_BUTTON = ''
REPEAT_SINGLE_BUTTON = ''
RESCUE_WORKERS_HELMET = ''
RESTROOM = ''
REVERSE_BUTTON = ''
REVOLVING_HEARTS = ''
RHINOCEROS = ''
RIBBON = ''
RICE_BALL = ''
RICE_CRACKER = ''
RIGHT_ANGER_BUBBLE = ''
RIGHT_ARROW = ''
RIGHT_ARROW_CURVING_DOWN = ''
RIGHT_ARROW_CURVING_LEFT = ''
```

```
RIGHT_ARROW_CURVING_UP = ''
RIGHT_FACING_FIST = ''
RING = ''
RINGED_PLANET = ''
ROASTED_SWEET_POTATO = ''
ROBOT = ''
ROCK = ''
ROCKET = ''
ROLLED_UP_NEWSPAPER = ''
ROLLER_COASTER = ''
ROLLER_SKATE = ''
ROLLING_ON_THE_FLOOR_LAUGHING = ''
ROLL_OF_PAPER = ''
ROOSTER = ''
ROSE = ''
ROSETTE = ''
ROUND_PUSHPIN = ''
RUGBY_FOOTBALL = ''
RUNNING_SHIRT = ''
RUNNING_SHOE = ''
SAD_BUT_RELIEVED_FACE = ''
SAFETY_PIN = ''
SAFETY_VEST = ''
SAGITTARIUS = ''
SAILBOAT = ''
SAKE = ''
SALT = ''
SANDWICH = ''
SANTA_CLAUS = ''
SARI = ''
SATELLITE = ''
```

```
SATELLITE_ANTENNA = ''
SAUROPOD = ''
SAXOPHONE = ''
SCARF = ''
SCHOOL = ''
SCISSORS = ''
SCORPIO = ''
SCORPION = ''
SCREWDRIVER = ''
SCROLL = ''
SEAL = ''
SEAT = ''
SECOND_PLACE_MEDAL = ''
SEEDLING = ''
SEE_NO_EVIL_MONKEY = ''
SELFIE = ''
SEVEN_OCLOCK = ''
SEVEN_THIRTY = ''
SEWING_NEEDLE = ''
SHALLOW_PAN_OF_FOOD = ''
SHAMROCK = ''
SHARK = ''
SHAVED_ICE = ''
SHEAF_OF_RICE = ''
SHIELD = ''
SHINTO_SHRINE = ''
SHIP = ''
SHOOTING_STAR = ''
SHOPPING_BAGS = ''
SHOPPING_CART = ''
SHORTCAKE = ''
```

```
SHORTS = ''
SHOWER = ''
SHRIMP = ''
SHUFFLE_TRACKS_BUTTON = ''
SHUSHING_FACE = ''
SIGN_OF_THE_HORNS = ''
SIX_OCLOCK = ''
SIX_THIRTY = ''
SKATEBOARD = ''
SKIER = ''
SKIS = ''
SKULL = ''
SKULL_AND_CROSSBONES = ''
SKUNK = ''
SLED = ''
SLEEPING_FACE = ''
SLEEPY_FACE = ''
SLIGHTLY_FROWNING_FACE = ''
SLIGHTLY_SMILING_FACE = ''
SLOTH = ''
SLOT_MACHINE = ''
SMALL_AIRPLANE = ''
SMALL_BLUE_DIAMOND = ''
SMALL_ORANGE_DIAMOND = ''
SMILING_CAT_WITH_HEART_EYES = ''
SMILING_FACE = ''
SMILING_FACE_WITH_HALO = ''
SMILING_FACE_WITH_HEARTS = ''
SMILING_FACE_WITH_HEART_EYES = ''
SMILING_FACE_WITH_HORNS = ''
SMILING_FACE_WITH_SMILING_EYES = ''
```

```
SMILING_FACE_WITH_SUNGLASSES = ''
SMILING_FACE_WITH_TEAR = ''
SMIRKING_FACE = ''
SNAIL = ''
SNAKE = ''
SNEEZING_FACE = ''
SNOWBOARDER = ''
SNOWFLAKE = ''
SNOWMAN = ''
SNOWMAN_WITHOUT_SNOW = ''
SNOW_CAPPED_MOUNTAIN = ''
SOAP = ''
SOCCER_BALL = ''
SOCKS = ''
SOFTBALL = ''
SOFT_ICE_CREAM = ''
SOON_ARROW = ''
SOS_BUTTON = ''
SPADE_SUIT = ''
SPAGHETTI = ''
SPARKLE = ''
SPARKLER = ''
SPARKLES = ''
SPARKLING_HEART = ''
SPEAKER_HIGH_VOLUME = ''
SPEAKER_LOW_VOLUME = ''
SPEAKER_MEDIUM_VOLUME = ''
SPEAKING_HEAD = ''
SPEAK_NO_EVIL_MONKEY = ''
SPEECH_BALLOON = ''
SPEEDBOAT = ''
```

```
SPIDER = ''
SPIDER_WEB = ''
SPIRAL_CALENDAR = ''
SPIRAL_NOTEPAD = ''
SPIRAL_SHELL = ''
SPONGE = ''
SPOON = ''
SPORTS\_MEDAL = ''
SPORT_UTILITY_VEHICLE = ''
SPOUTING_WHALE = ''
SQUID = ''
SQUINTING_FACE_WITH_TONGUE = ''
STADIUM = ''
STAR = ''
STAR_AND_CRESCENT = ''
STAR_OF_DAVID = ''
STAR_STRUCK = ''
STATION = ''
STATUE_OF_LIBERTY = ''
STEAMING_BOWL = ''
STETHOSCOPE = ''
STOPWATCH = ''
STOP_BUTTON = ''
STOP_SIGN = ''
STRAIGHT_RULER = ''
STRAWBERRY = ''
STUDIO_MICROPHONE = ''
STUFFED_FLATBREAD = ''
SUN = ''
SUNFLOWER = ''
SUNGLASSES = ''
```

```
SUNRISE = ''
SUNRISE_OVER_MOUNTAINS = ''
SUNSET = ''
SUN_BEHIND_CLOUD = ''
SUN_BEHIND_LARGE_CLOUD = ''
SUN_BEHIND_RAIN_CLOUD = ''
SUN_BEHIND_SMALL_CLOUD = ''
SUN_WITH_FACE = ''
SUPERHERO = ''
SUPERVILLAIN = ''
SUSHI = ''
SUSPENSION_RAILWAY = ''
SWAN = ''
SWEAT_DROPLETS = ''
SYNAGOGUE = ''
SYRINGE = ''
TACO = ''
TAKEOUT_BOX = ''
TAMALE = ''
TANABATA_TREE = ''
TANGERINE = ''
TAURUS = ''
TAXI = ''
TEACUP_WITHOUT_HANDLE = ''
TEAPOT = ''
TEAR_OFF_CALENDAR = ''
TEDDY_BEAR = ''
TELEPHONE = ''
TELEPHONE_RECEIVER = ''
TELESCOPE = ''
TELEVISION = ''
```

218

```
TENNIS = ''
TENT = ''
TEN_OCLOCK = ''
TEN_THIRTY = ''
TEST_TUBE = ''
THERMOMETER = ''
THINKING_FACE = ''
THIRD_PLACE_MEDAL = ''
THONG_SANDAL = ''
THOUGHT_BALLOON = ''
THREAD = ''
THREE_OCLOCK = ''
THREE_THIRTY = ''
THUMBS_DOWN = ''
THUMBS_UP = ''
TICKET = ''
TIGER = ''
TIGER_FACE = ''
TIMER_CLOCK = ''
TIRED_FACE = ''
TOILET = ''
TOKYO_TOWER = ''
TOMATO = ''
TONGUE = ''
TOOLBOX = ''
TOOTH = ''
TOOTHBRUSH = ''
TOP_ARROW = ''
TOP\_HAT = ''
TORNADO = ''
TRACKBALL = ''
```

```
TRACTOR = ''
TRADE_MARK = 'TR'
TRAIN = ''
TRAM = ''
TRAM_CAR = ''
TRANSGENDER_SYMBOL = ''
TRIANGULAR_FLAG = ''
TRIANGULAR_RULER = ''
TRIDENT_EMBLEM = ''
TROLLEYBUS = ''
TROPHY = ''
TROPICAL_DRINK = ''
TROPICAL_FISH = ''
TRUMPET = ''
TULIP = ''
TUMBLER_GLASS = ''
TURKEY = ''
TURTLE = ''
TWELVE_OCLOCK = ''
TWELVE_THIRTY = ''
TWO_HEARTS = ''
TWO_HUMP_CAMEL = ''
TWO_OCLOCK = ''
TWO_THIRTY = ''
T_REX = ''
T_SHIRT = ''
UMBRELLA = ''
UMBRELLA_ON_GROUND = ''
UMBRELLA_WITH_RAIN_DROPS = ''
UNAMUSED_FACE = ''
UNICORN = ''
```

220

```
UNLOCKED = ''
UPSIDE_DOWN_FACE = ''
UPWARDS_BUTTON = ''
UP_ARROW = ''
UP_BUTTON = ''
UP_DOWN_ARROW = ''
UP_LEFT_ARROW = ''
UP_RIGHT_ARROW = ''
VAMPIRE = ''
VERTICAL_TRAFFIC_LIGHT = ''
VIBRATION_MODE = ''
VICTORY_HAND = ''
VIDEOCASSETTE = ''
VIDEO_CAMERA = ''
VIDEO_GAME = ''
VIOLIN = ''
VIRGO = ''
VOLCANO = ''
VOLLEYBALL = ''
VS_BUTTON = ''
VULCAN_SALUTE = ''
WAFFLE = ''
WANING_CRESCENT_MOON = ''
WANING_GIBBOUS_MOON = ''
WARNING = ''
WASTEBASKET = ''
WATCH = ''
WATERMELON = ''
WATER_BUFFALO = ''
WATER_CLOSET = ''
WATER_WAVE = ''
```

```
WAVING_HAND = ''
WAVY_DASH = ''
WAXING_CRESCENT_MOON = ''
WAXING_GIBBOUS_MOON = ''
WEARY_CAT = ''
WEARY_FACE = ''
WEDDING = ''
WHALE = ''
WHEELCHAIR_SYMBOL = ''
WHEEL_OF_DHARMA = ''
WHITE_CANE = ''
WHITE_CIRCLE = ''
WHITE_EXCLAMATION_MARK = ''
WHITE_FLAG = ''
WHITE_FLOWER = ''
WHITE_HAIR = ''
WHITE_HEART = ''
WHITE_LARGE_SQUARE = ''
WHITE_MEDIUM_SMALL_SQUARE = ''
WHITE_MEDIUM_SQUARE = ''
WHITE_QUESTION_MARK = ''
WHITE_SMALL_SQUARE = ''
WHITE_SQUARE_BUTTON = ''
WILTED_FLOWER = ''
WINDOW = ''
WIND_CHIME = ''
WIND_FACE = ''
WINE_GLASS = ''
WINKING_FACE = ''
WINKING_FACE_WITH_TONGUE = ''
WOLF = ''
```

```
WOMAN = ''
WOMANS_BOOT = ''
WOMANS_CLOTHES = ''
WOMANS_HAT = ''
WOMANS_SANDAL = ''
WOMAN_AND_MAN_HOLDING_HANDS = ''
WOMAN_DANCING = ''
WOMAN_WITH_HEADSCARF = ''
WOMENS_ROOM = ''
WOMEN_HOLDING_HANDS = ''
WOOD = ''
WOOZY_FACE = ''
WORLD_MAP = ''
WORM = ''
WORRIED_FACE = ''
WRAPPED_GIFT = ''
WRENCH = ''
WRITING_HAND = ''
YARN = ''
YAWNING_FACE = ''
YELLOW_CIRCLE = ''
YELLOW_HEART = ''
YELLOW_SQUARE = ''
YEN_BANKNOTE = ''
YIN_YANG = ''
YO_YO = ''
ZANY_FACE = ''
ZEBRA = ''
ZIPPER_MOUTH_FACE = ''
ZOMBIE = ''
ZZZ = ''
```

# 3.2.2 Fonts

Fonts in the pygamelib are nothing more than a specially organized sprite collection.

The way to use it is extremely simple: you instantiate a Font object and ask it to load the data from a specific font.

For example to load the 8bits font, you do:

# Example::

```
from pygamelib.gfx import core
my_font = core.Font('8bits')
```

That's it! The you can use it to format Text objects.

# 8bits

New in version 1.3.0.

# How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("8bits")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?



# figlet-caligraphy

New in version 1.3.0.

### How to use?

Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-caligraphy")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?

# More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-doom

New in version 1.3.0.

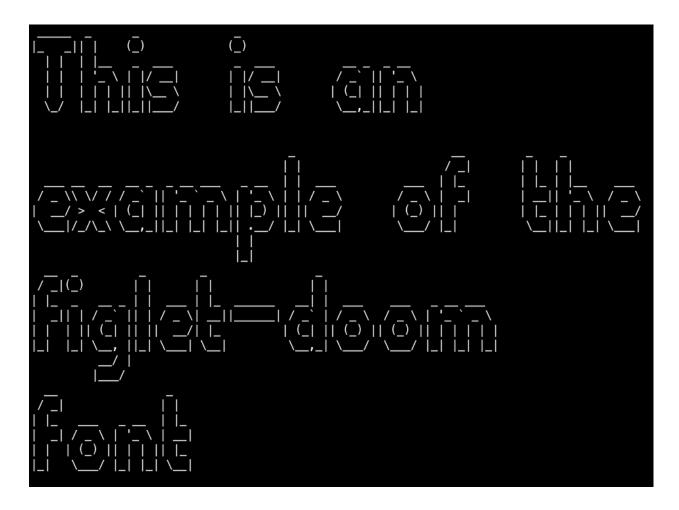
# How to use?

Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-doom")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?



# More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-graffiti

New in version 1.3.0.

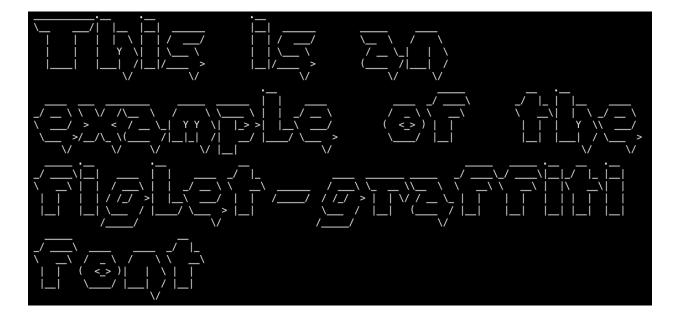
### How to use?

Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-graffiti")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

### What does it look like?



#### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-mirror

New in version 1.3.0.

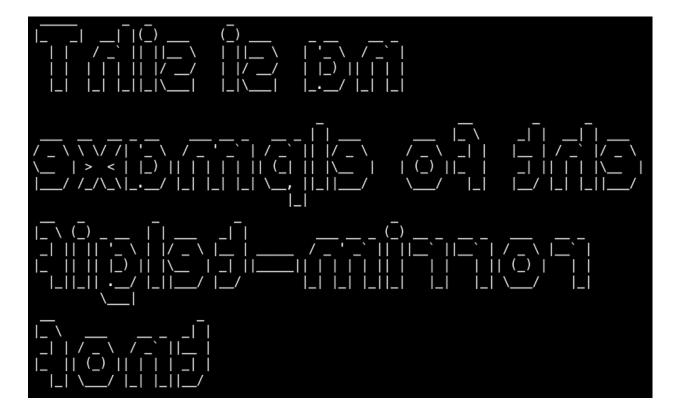
### How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-mirror")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?



#### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-pepper

New in version 1.3.0.

## How to use?

#### Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-pepper")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

#### What does it look like?

### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-poison

New in version 1.3.0.

# How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-poison")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?

@@@@@@ @@! !@!	@@@ @@@ @@@ @@@@@@@ @@@ @@@ @@@ @@@@@@@ @@! @@@ @@! !@@ !@! @!@ !@! !@!	@@@ @@@@@@ @@@ @@@@@@@ @@! !@@ !@! !@@!! !!@ !!@@!! !!! !!@!!! !!! !!!!!!!!		
@@@@@@@@@@@@@@@@@@! !@! !@!!!!!! !!!!!!!	QGG   QGG   QGGGGGGGGGGGGGGGGGGGGGGGG	@@@@@@@@@@@@@@@@@@@@@@@@@@! !@! @@@ @@! !@! @!@ !@! @!@@!@! !!! !!@!!! !!! !!: !!!		966
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@	@@@ @@@@@@@@ @@@ @@@@@@ @@@ @@@@@@@@ @@! @@!	30 0000000 00! 0!! 0!0!0!0! !!! !!!0!0! !!:		
@@@@@@@@@@@@@@@@@@@! !@! @!!!!!! !!!!!! !!!				

# More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-puffy

New in version 1.3.0.

### How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-puffy")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?



### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-rounded

New in version 1.3.0.

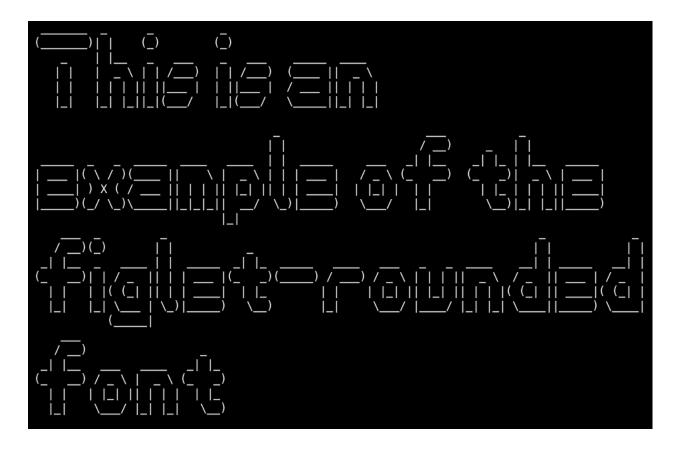
# How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-rounded")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

### What does it look like?



### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-stampatello

New in version 1.3.0.

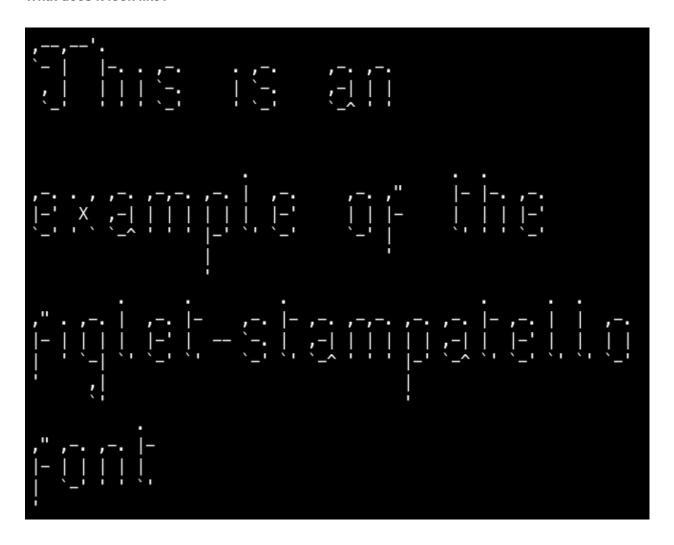
# How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-stampatello")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?



### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-univers

New in version 1.3.0.

### How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-univers")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

# What does it look like?

#### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# figlet-wavy

New in version 1.3.0.

# How to use?

# Example:

```
from pygamelib.gfx.core import Font
from pygamelib.base import Text

my_font = Font("figlet-wavy")
my_text = Text("Lorem Ipsum", font=my_font)
my_text.print_formatted()
```

### What does it look like?

#### More

Please visit the FIGlet project: http://www.figlet.org/

The fonts prefixed by "figlet-" are but a small portion of what's available here.

To easily convert FIGlet fonts to the pygamelib's format you can you the figlet-to-pygamelib script available here: https://github.com/pygamelib/figlet-to-pygamelib

# 3.3 base

The base module provide basic objects and exceptions that are used by the entire library.

# 3.3.1 Console

### class pygamelib.base.Console

Bases: object

The Console class is a singleton wrapper around the blessed. Terminal() class. Since the library is using Terminal a lot, it is both useful and efficient to have a quick access to a single instance of the class.

This class only expose one method: instance() that returns the singleton instance.

#### **Methods**

instance()

Returns the instance of the blessed. Terminal object.

### classmethod instance()

Returns the instance of the blessed. Terminal object.

New in version 1.3.0.

The pygamelib extensively use the Terminal object from the blessed module. However we find ourselves in need of a Terminal instance a lot, so to help with memory and execution time we just encapsulate the Terminal object in a singleton so any object can use it without instantiating it many times (and messing up with the contexts).

# Returns

Instance of blessed. Terminal object

Example:

term = Console.instance()

# 3.3.2 History

## class pygamelib.base.History

Bases: object

New in version 1.4.0.

History is a general history management object. It works by managing a time structure with past, current and future actions. Undoing and redoing is simply the action of moving along this timeline.

This object is data agnostic, it can be used with any type of objects. Internally, it uses a concept of "actions". An "action" consist of any python object that can be stored in an array.

Undoing or redoing over the limit of the past and future actions is not doing anything.

When add() is called, it clears the future actions.

This object provides a singleton interface through the *instance()*. It can be created as a normal instantiated object or as a global manager depending on your needs.

## Example:

```
global_history = History.instance()
global_history.add('Hel')
global_history.add('Hello')
global_history.add('Hello Wo')
global_history.add('Hello World')
print(global_history.current) # print "Hello World"
global_history.undo()
print(global_history.current) # print "Hello Wo"
global_history.undo()
print(global_history.current) # print "Hello"
global_history.redo()
global_history.redo()
global_history.redo() # This one does nothing as we called undo only twice.
print(global_history.current) # print "Hello World"
# Now an example of reset through add()
global_history.undo()
global_history.undo()
print(global_history.current) # print "Hello"
global_history.add("Hello there!")
print(global_history.current) # print "Hello there!"
global_history.redo() # does nothing as the future was reset by add()
```

```
__init__() \rightarrow None
```

The constructor takes no parameters. In the future, it could take some, like the maximum size of the history for example.

3.3. base 241

### **Methods**

init()	The constructor takes no parameters.
add(action)	Add an action (i.e: object) to the history.
<pre>instance(*args, **kwargs)</pre>	Returns/creates the instance of the History object
redo()	Step forward into the actions' timeline.
reset()	Reset the history to its initial state.
undo()	Step backward into the actions' timeline.

#### **Attributes**

current	current is a read-only property to access the current
	action of the history.

 $add(action: object) \rightarrow None$ 

Add an action (i.e: object) to the history.

This action becomes the current one.

Adding an action reset the future actions (i.e: you cannot redo what was undone before adding the action).

#### **Parameters**

**action** (*object*) – The action to add to the history.

Example:

```
global_history = History.instance()
global_history.add('Hel')
global_history.add('Hello')
print(global_history.current) # print "Hello"
```

#### property current: object

current is a read-only property to access the current action of the history.

To add an action and set it as current, use the add() method.

## classmethod instance(\*args, \*\*kwargs)

Returns/creates the instance of the History object

Creates an History object on first call an then returns the same instance on further calls

#### Returns

Instance of the History object

### **Return type**

History

# $redo() \rightarrow None$

Step forward into the actions' timeline.

This method takes the current action and puts it in the past queue. It then pop the last action of the future queue and set it as current.

If there's no more future actions, it does nothing.

Example:

```
global_history = History.instance()
global_history.add('Hel')
global_history.add('Hello')
print(global_history.current) # print "Hello"
global_history.undo()
print(global_history.current) # print "Hel"
global_history.redo()
print(global_history.current) # print "Hello"
```

#### $reset() \rightarrow None$

Reset the history to its initial state. It clears all the past and future actions. It also set the current action to None.

#### **undo()** $\rightarrow$ None

Step backward into the actions' timeline.

This method takes the current action and puts it in the future queue. It then pop the last action of the past queue and set it as current.

If there's no more past actions, it does nothing.

### Example:

```
global_history = History.instance()
global_history.add('Hel')
global_history.add('Hello')
print(global_history.current) # print "Hello"
global_history.undo()
print(global_history.current) # print "Hel"
```

## 3.3.3 Math

## class pygamelib.base.Math

Bases: object

The math class regroup math functions required for game development.

New in version 1.2.0.

For the moment there is only static methods in that class but it will evolve in the future.

```
__init__()
```

## **Methods**

```
__init__()

distance(row1, column1, row2, column2) Return the euclidean distance between to points.

intersect(row1, column1, width1, height1, ...) This function check if 2 rectangles intersect.

lerp(a, b, t) Return the linear interpolation between 2 values relative to a third value.
```

3.3. base 243

**static distance**(row1: int, column1: int, row2: int, column2: int)  $\rightarrow$  float

Return the euclidean distance between to points.

Points are identified by their row and column. If you want the distance in number of cells, you need to round the result (see example).

#### **Parameters**

- **row1** (*int*) the row number (coordinate) of the first point.
- **column1** (*int*) the column number (coordinate) of the first point.
- row2 (int) the row number (coordinate) of the second point.
- column2 (int) the column number (coordinate) of the second point.

#### Returns

The distance between the 2 points.

## Return type

float

# Example:

**static intersect**(row1: int, column1: int, width1: int, height1: int, row2: int, column2: int, width2: int, height2: int)  $\rightarrow$  bool

This function check if 2 rectangles intersect.

The 2 rectangles are defined by their positions (row, column) and dimension (width and height).

## **Parameters**

- row1 (int) The row of the first rectangle
- **column1** (*int*) The column of the first rectangle
- width1 (int) The width of the first rectangle
- **height1** (*int*) The height of the first rectangle
- row2 (int) The row of the second rectangle
- **column2** The column of the second rectangle
- width2 (int) The width of the second rectangle
- **height2** (*int*) The height of the second rectangle

#### **Returns**

A boolean, True if the rectangles intersect False, otherwise.

## Example:

**static lerp**(a: float, b: float, t: float)  $\rightarrow$  float

Return the linear interpolation between 2 values relative to a third value.

New in version 1.3.0.

#### **Parameters**

- a (float) Start value of the interpolation. Returned if t is 0.
- **b** (*float*) End value of the interpolation. Returned if t is 1.
- t (float) A value between 0 and 1 used to interpolate between a and b.

Example:

```
value = lerp(0, 100, 0.5) # 50
```

# 3.3.4 PglBaseObject

## class pygamelib.base.PglBaseObject

Bases: object

The base object of most of the pygamelib's classes.

New in version 1.3.0.

The PglBaseObject has 2 goals:

- Store the object's screen position.
- Implements a modified observer design pattern.

It is "modified" as it acts both as the observer and the client. The idea behind it is that any object can observe and be observed by any other objects.

The base logic of the pattern is already implemented and probably does not require re-implementation on the child object. However, the <code>handle\_notification()</code> method needs to be implemented in each client. The actual processing of the notification is indeed specific to each object.

Storing the screen position is particularly useful for *BoardItem* subclasses as they only know their position relative to the *Board* but might need to know their absolute screen coordinates.

This is a lightweight solution to that issue. It is not foolproof however! The screen\_row and screen\_column attributes are not wrapped properties and can be modified to mess up things. It shouldn't be done lightly. You have been warned!

```
__init__() \rightarrow None
```

Like the object class, this class constructor takes no parameter.

3.3. base 245

### **Methods**

init()	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
<pre>detach(observer)</pre>	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

# **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

# Returns

True or False depending on the success of the operation.

#### Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### property screen\_column: int

A property to get/set the screen column.

## **Parameters**

value (int) – the screen column

# Return type

int

#### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (*int*) – the screen row

# Return type

int

#### $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

3.3. base 247

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

an\_object.store\_screen\_coordinate(3,8)

# 3.3.5 PglException

exception pygamelib.base.PglException(error, message)

Exception raised for non specific errors in the pygamelib.

# 3.3.6 PglInvalidLevelException

exception pygamelib.base.PglInvalidLevelException(message)

Exception raised if a level is not associated to a board in Game().

# 3.3.7 PglInvalidTypeException

exception pygamelib.base.PglInvalidTypeException(message)

Exception raised for invalid types.

# 3.3.8 PglInventoryException

exception pygamelib.base.PglInventoryException(error, message)

Exception raised for issue related to the inventory. The error is an explicit string, and the message explains the error.

# 3.3.9 PglObjectIsNotMovableException

exception pygamelib.base.PglObjectIsNotMovableException(message)

Exception raised if the object that is being moved is not a subclass of Movable.

# 3.3.10 PglOutOfBoardBoundException

exception pygamelib.base.PglOutOfBoardBoundException(message)

Exception for out of the board's boundaries operations.

# 3.3.11 Text

class pygamelib.base.Text(text=",fg\_color=None, bg\_color=None, style=",font=None)

Bases: PglBaseObject

An object to manipulate and display text in multiple contexts.

New in version 1.2.0.

The Text class is a collection of text formatting and display static methods.

You can either instantiate an object or use the static methods.

The Text object allow for easy text manipulation through its collection of independent attributes. They help to set the text, its style and the foreground and background colors.

The Text object can be converted to a *Sprite* through the Sprite.from\_text() method. This is particularly useful to the place text on the game *Board*.

\_\_init\_\_(text=", fg\_color=None, bg\_color=None, style=", font=None)

#### **Parameters**

- **text** (*str*) The text to manipulate
- **fg\_color** (*Color*) The foreground color for the text.
- **bg\_color** (*Color*) The background color for the text.
- **style** (*str*) The style for the text.
- **font** (*Font*) The font in which the text is going to be displayed (only works when using Screen.place() and Screen.update())

## **Methods**

init([text, fg_color, bg_color, style, font])	
	param text
	The text to manipulate
attach(observer)	Attach an observer to this instance.
<pre>handle_notification(target[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
black(message)	This method works exactly the way green_bright() work with different color.
black_bright(message)	This method works exactly the way green_bright() work with different color.
<pre>black_dim(message)</pre>	This method works exactly the way green_bright() work with different color.
blue(message)	This method works exactly the way green_bright() work with different color.
blue_bright(message)	This method works exactly the way green_bright() work with different color.
blue_dim(message)	This method works exactly the way green_bright() work with different color.
cyan(message)	This method works exactly the way green_bright() work with different color.

continues on next page

3.3. base 249

Table 6 – continued from previous page

cyan_bright(message)  cyan_dim(message)  this method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  Print a debug message.  detach(observer) Detach an observer from this instance.  fatal(message) green(message)  This method works exactly the way green_bright() work with different color.  green_bright(message)  This method works exactly the way green_bright() work with different color.  green_bright(message)  This method works exactly the way green_bright() work with different color.  Info(message)  magenta(message)  This method works exactly the way green_bright() work with different color.  magenta_bright(message)  This method works exactly the way green_bright() work with different color.  magenta_dim(message)  This method works exactly the way green_bright() work with different color.  notify([modifier, attribute, value]) print_white_on_red(message)  This method works exactly the way green_bright() work with different color.  red_bright(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the wa	Table 6 Continues	i iloili pievious page
debug(message)         work with different color.           detach(boserver)         Detach an observer from this instance.           fatal(message)         Print a fatal message.           green(message)         This method works exactly the way green_bright() work with different color.           green_bright(message)         Return a string formatted to be bright green           info(message)         Print an informative message.           magenta(message)         Print an informative message.           magenta(message)         This method works exactly the way green_bright() work with different color.           magenta_bright(message)         This method works exactly the way green_bright() work with different color.           magenta_dim(message)         This method works exactly the way green_bright() work with different color.           notify([modifier, attribute, value])         Notify all the observers that a change occurred.           print_white_on_red(message)         Print a white message over a red background.           red(message)         This method works exactly the way green_bright() work with different color.           red_bright(message)         This method works exactly the way green_bright() work with different color.           red_dim(message)         This method works exactly the way green_bright() work with different color.           red_dim(message)         This method works exactly the way green_bright() work with different color.	cyan_bright(message)	
debug(message)         Print a debug message.           detach(observer)         Detach an observer from this instance.           fatal(message)         Print a fatal message.           green_bright(message)         This method works exactly the way green_bright() work with different color.           green_bright(message)         Return a string formatted to be bright green           jinfo(message)         Print an informative message.           magenta(message)         This method works exactly the way green_bright() work with different color.           magenta_bright(message)         This method works exactly the way green_bright() work with different color.           magenta_dim(message)         This method works exactly the way green_bright() work with different color.           notify([modifier, attribute, value])         Notify all the observers that a change occurred.           print_white_on_red(message)         Print a white message over a red background.           red(message)         This method works exactly the way green_bright() work with different color.           red_bright(message)         This method works exactly the way green_bright() work with different color.           red_dim(message)         This method works exactly the way green_bright() work with different color.           render_to_buffer(buffer, row, column,)         Render the Text object from the display buffer to the frame buffer.           warn(message)         This method works	cyan_dim(message)	
detach(observer)         Detach an observer from this instance.           fatal(message)         Print a fatal message.           green(message)         This method works exactly the way green_bright() work with different color.           green_bright(message)         Return a string formatted to be bright green           info(message)         This method works exactly the way green_bright() work with different color.           magenta(message)         Print an informative message.           magenta_bright(message)         This method works exactly the way green_bright() work with different color.           magenta_dim(message)         This method works exactly the way green_bright() work with different color.           notify([modifier, attribute, value])         Notify all the observers that a change occurred.           print_white_on_red(message)         Print a white message over a red background.           red(message)         This method works exactly the way green_bright() work with different color.           red_bright(message)         This method works exactly the way green_bright() work with different color.           red_dim(message)         This method works exactly the way green_bright() work with different color.           warn(message)         Print a warning message.           white(message)         This method works exactly the way green_bright() work with different color.           white_bright(message)         This method works exactly the way gr	debug(message)	
### Frint a fatal message.  ### green(message)  ### This method works exactly the way green_bright()  ### work with different color.  ### Return a string formatted to be bright green  ### This method works exactly the way green_bright()  ### work with different color.  ### magenta_bright(message)  ### magenta_bright(message)  ### magenta_bright(message)  ### magenta_dim(message)		
This method works exactly the way green_bright() work with different color.  Return a string formatted to be bright green green_dim(message) This method works exactly the way green_bright() work with different color.  magenta(message) Print an informative message.  magenta(message) This method works exactly the way green_bright() work with different color.  magenta_bright(message) This method works exactly the way green_bright() work with different color.  magenta_dim(message) This method works exactly the way green_bright() work with different color.  notify([modifier, attribute, value]) Notify all the observers that a change occurred. Print a white message over a red background.  red(message) This method works exactly the way green_bright() work with different color.  red_bright(message) This method works exactly the way green_bright() work with different color.  red_dim(message) This method works exactly the way green_bright() work with different color.  red_dim(message) This method works exactly the way green_bright() work with different color.  render_to_buffer(buffer, row, column,) Render the Text object from the display buffer to the frame buffer.  warn(message) White(message) This method works exactly the way green_bright() work with different color.  white_bright(message) This method works exactly the way green_bright() work with different color.  white_dim(message) This method works exactly the way green_bright() work with different color.  white_dim(message) This method works exactly the way green_bright() work with different color.  yellow(message) This method works exactly the way green_bright() work with different color.  yellow(message) This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work	· · · · · · · · · · · · · · · · · · ·	
work with different color.  Return a string formatted to be bright green  This method works exactly the way green_bright() work with different color.  Info(message)  magenta(message)  Print an informative message.  This method works exactly the way green_bright() work with different color.  magenta_bright(message)  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  Notify([modifier, attribute, value]) Print_white_on_red(message)  Print a white message over a red background.  red(message)  This method works exactly the way green_bright() work with different color.  red_bright(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  red_dim(message)  This method works exactly the way green_bright() work with different color.  render_to_buffer(buffer, row, column,)  Render the Text object from the display buffer to the frame buffer.  warn(message)  White(message)  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.	`	e
This method works exactly the way green_bright() work with different color.  Print an informative message.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  notify([modifier, attribute, value])     print_white_on_red(message)		work with different color.
work with different color.  Info(message)  magenta(message)  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  Indify([modifier, attribute, value])  Indifferent color.  Indifferent colo		
magenta(message)This method works exactly the way green_bright() work with different color.magenta_bright(message)This method works exactly the way green_bright() work with different color.magenta_dim(message)This method works exactly the way green_bright() work with different color.notify([modifier, attribute, value])Notify all the observers that a change occurred.print_white_on_red(message)Print a white message over a red background.red(message)This method works exactly the way green_bright() work with different color.red_bright(message)This method works exactly the way green_bright() work with different color.red_dim(message)This method works exactly the way green_bright() work with different color.render_to_buffer(buffer, row, column,)Render the Text object from the display buffer to the frame buffer.warn(message)Print a warning message.white(message)This method works exactly the way green_bright() work with different color.white_bright(message)This method works exactly the way green_bright() work with different color.white_dim(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.	<pre>green_dim(message)</pre>	
work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  Notify([modifier, attribute, value]) Notify all the observers that a change occurred.  Print_white_on_red(message) Print a white message over a red background.  red(message) This method works exactly the way green_bright() work with different color.  red_bright(message) This method works exactly the way green_bright() work with different color.  red_dim(message) This method works exactly the way green_bright() work with different color.  render_to_buffer(buffer, row, column,) Render the Text object from the display buffer to the frame buffer.  warn(message) Print a warning message.  white(message) This method works exactly the way green_bright() work with different color.  white_bright(message) This method works exactly the way green_bright() work with different color.  white_dim(message) This method works exactly the way green_bright() work with different color.  white_dim(message) This method works exactly the way green_bright() work with different color.  yellow_bright(message) This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.	info(message)	Print an informative message.
work with different color.  magenta_dim(message)  This method works exactly the way green_bright() work with different color.  Notify([modifier, attribute, value]) Notify all the observers that a change occurred.  Print_white_on_red(message) Print a white message over a red background.  This method works exactly the way green_bright() work with different color.  red_bright(message) This method works exactly the way green_bright() work with different color.  red_dim(message) This method works exactly the way green_bright() work with different color.  render_to_buffer(buffer, row, column,) Render the Text object from the display buffer to the frame buffer.  warn(message) Print a warning message. White(message) This method works exactly the way green_bright() work with different color.  white_bright(message) This method works exactly the way green_bright() work with different color.  white_dim(message) This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.	magenta(message)	
work with different color.  notify([modifier, attribute, value])  print_white_on_red(message)  red(message)  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  red_dim(message)  This method works exactly the way green_bright()  work with different color.  Render the Text object from the display buffer to the frame buffer.  Warn(message)  Print a warning message.  White(message)  This method works exactly the way green_bright()  work with different color.  White_bright(message)  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.	<pre>magenta_bright(message)</pre>	• • • • • • • • • • • • • • • • • • • •
print_white_on_red(message)Print a white message over a red background.red(message)This method works exactly the way green_bright() work with different color.red_bright(message)This method works exactly the way green_bright() work with different color.red_dim(message)This method works exactly the way green_bright() work with different color.render_to_buffer(buffer, row, column,)Render the Text object from the display buffer to the frame buffer.warn(message)Print a warning message.white(message)This method works exactly the way green_bright() work with different color.white_bright(message)This method works exactly the way green_bright() work with different color.white_dim(message)This method works exactly the way green_bright() work with different color.yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.	<pre>magenta_dim(message)</pre>	, , , , , , , , , , , , , , , , , , , ,
print_white_on_red(message)Print a white message over a red background.red(message)This method works exactly the way green_bright() work with different color.red_bright(message)This method works exactly the way green_bright() work with different color.red_dim(message)This method works exactly the way green_bright() work with different color.render_to_buffer(buffer, row, column,)Render the Text object from the display buffer to the frame buffer.warn(message)Print a warning message.white(message)This method works exactly the way green_bright() work with different color.white_bright(message)This method works exactly the way green_bright() work with different color.white_dim(message)This method works exactly the way green_bright() work with different color.yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.	<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
red(message)This method works exactly the way green_bright() work with different color.red_bright(message)This method works exactly the way green_bright() work with different color.red_dim(message)This method works exactly the way green_bright() work with different color.render_to_buffer(buffer, row, column,)Render the Text object from the display buffer to the frame buffer.warn(message)Print a warning message.white(message)This method works exactly the way green_bright() work with different color.white_bright(message)This method works exactly the way green_bright() work with different color.white_dim(message)This method works exactly the way green_bright() work with different color.yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright()		
red_bright(message)This method works exactly the way green_bright() work with different color.red_dim(message)This method works exactly the way green_bright() work with different color.render_to_buffer(buffer, row, column,)Render the Text object from the display buffer to the frame buffer.warn(message)Print a warning message.white(message)This method works exactly the way green_bright() work with different color.white_bright(message)This method works exactly the way green_bright() work with different color.white_dim(message)This method works exactly the way green_bright() work with different color.yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright() work with different color.		This method works exactly the way green_bright()
This method works exactly the way green_bright() work with different color.  Render the Text object from the display buffer to the frame buffer.  Print a warning message.  White(message)  Print a warning message.  This method works exactly the way green_bright() work with different color.  White_bright(message)  This method works exactly the way green_bright() work with different color.  White_dim(message)  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  Yellow_bright(message)  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  Yellow_dim(message)  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.  This method works exactly the way green_bright() work with different color.	red_bright(message)	
work with different color.  render_to_buffer(buffer, row, column,)  Render the Text object from the display buffer to the frame buffer.  Print a warning message.  White(message)  Print a warning message.  This method works exactly the way green_bright() work with different color.  White_bright(message)  This method works exactly the way green_bright() work with different color.  White_dim(message)  This method works exactly the way green_bright() work with different color.  Yellow(message)  This method works exactly the way green_bright() work with different color.  Yellow_bright(message)  This method works exactly the way green_bright() work with different color.  Yellow_bright(message)  This method works exactly the way green_bright() work with different color.  Yellow_dim(message)  This method works exactly the way green_bright() work with different color.		
frame buffer.  warn(message)  white(message)  This method works exactly the way green_bright()  work with different color.  white_bright(message)  This method works exactly the way green_bright()  work with different color.  This method works exactly the way green_bright()  work with different color.  yellow(message)  This method works exactly the way green_bright()  work with different color.  yellow_bright(message)  This method works exactly the way green_bright()  work with different color.  yellow_bright(message)  This method works exactly the way green_bright()  work with different color.  yellow_dim(message)  This method works exactly the way green_bright()  This method works exactly the way green_bright()	red_dim(message)	
white(message)This method works exactly the way green_bright() work with different color.white_bright(message)This method works exactly the way green_bright() work with different color.white_dim(message)This method works exactly the way green_bright() work with different color.yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright()This method works exactly the way green_bright()	<pre>render_to_buffer(buffer, row, column,)</pre>	
white(message)This method works exactly the way green_bright() work with different color.white_bright(message)This method works exactly the way green_bright() work with different color.white_dim(message)This method works exactly the way green_bright() work with different color.yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright()This method works exactly the way green_bright()	warn(message)	Print a warning message.
work with different color.  White_dim(message)  This method works exactly the way green_bright() work with different color.  Yellow(message)  This method works exactly the way green_bright() work with different color.  Yellow_bright(message)  This method works exactly the way green_bright() work with different color.  Yellow_dim(message)  This method works exactly the way green_bright()  This method works exactly the way green_bright()	white(message)	This method works exactly the way green_bright()
white_dim(message)This method works exactly the way green_bright() work with different color.yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright()	white_bright(message)	
yellow(message)This method works exactly the way green_bright() work with different color.yellow_bright(message)This method works exactly the way green_bright() work with different color.yellow_dim(message)This method works exactly the way green_bright()	<pre>white_dim(message)</pre>	This method works exactly the way green_bright()
yellow_bright(message)This method works exactly the way green_bright()work with different color.yellow_dim(message)This method works exactly the way green_bright()	yellow(message)	This method works exactly the way green_bright()
yellow_dim(message)  This method works exactly the way green_bright()	<pre>yellow_bright(message)</pre>	This method works exactly the way green_bright()
	yellow_dim(message)	This method works exactly the way green_bright()

250 Chapter 3. Tutorials

## **Attributes**

bg_color	The bg_color attribute sets the background color.
fg_color	The fg_color attribute sets the foreground color.
length	Return the true length of the text.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
text	The text attribute.
style	The style attribute sets the style of the text.
parent	This object's parent.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property bg\_color

The bg\_color attribute sets the background color. It needs to be a *Color*.

New in version 1.3.0.

When the background color is changed, the observers are notified of the change with the pygamelib.base.Text.bg\_color:changed event. The new color is passed as the *value* parameter.

## static black(message)

This method works exactly the way green\_bright() work with different color.

## static black\_bright(message)

This method works exactly the way green\_bright() work with different color.

#### static black\_dim(message)

This method works exactly the way green\_bright() work with different color.

## static blue(message)

This method works exactly the way green\_bright() work with different color.

## static blue\_bright(message)

This method works exactly the way green\_bright() work with different color.

3.3. base 251

```
static blue_dim(message)
```

This method works exactly the way green\_bright() work with different color.

## static cyan(message)

This method works exactly the way green\_bright() work with different color.

## static cyan\_bright(message)

This method works exactly the way green\_bright() work with different color.

#### static cyan\_dim(message)

This method works exactly the way green\_bright() work with different color.

#### static debug(message)

Print a debug message.

The debug message is a regular message prefixed by INFO in blue on a green background.

#### **Parameters**

**message** (str) – The message to print.

Example:

```
base.Text.debug("This is probably going to success, eventually...")
```

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## static fatal(message)

Print a fatal message.

The fatal message is a regular message prefixed by FATAL in white on a red background.

#### Parameters

**message** (str) – The message to print.

Example:

```
base.Text.fatal("|x_x|")
```

# property fg\_color

The fg\_color attribute sets the foreground color. It needs to be a *Color*.

New in version 1.3.0.

When the foreground color is changed, the observers are notified of the change with the pygamelib.base.Text.fg\_color:changed event. The new color is passed as the *value* parameter.

#### static green(message)

This method works exactly the way green\_bright() work with different color.

## static green\_bright(message)

Return a string formatted to be bright green

#### **Parameters**

**message** (str) – The message to format.

#### Returns

The formatted string

# Return type

str

#### Example:

```
print( Text.green_bright("This is a formatted message") )
```

## static green\_dim(message)

This method works exactly the way green\_bright() work with different color.

## handle\_notification(target, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# static info(message)

Print an informative message.

The info is a regular message prefixed by INFO in white on a blue background.

#### **Parameters**

**message** (str) – The message to print.

Example:

```
base.Text.info("This is a very informative message.")
```

## property length

Return the true length of the text.

New in version 1.3.0.

With UTF8 and emojis the length of a string as returned by python's len() function is often very wrong. For example, the len("x1b[48;2;139;22;19mx1b[38;2;160;26;23mx1b[0m") returns 39 when it should return 1.

This method returns the actual printing/display size of the text.

3.3. base 253

Note: This is a read only value. It is automatically updated when the text property is changed.

```
Example:
```

```
game.screen.place(my_text, 0, game.screen.width - my_text.length)
```

#### classmethod load(data: dict = None)

Load data and create a new Text object out of it.

New in version 1.3.0.

#### **Parameters**

**data** (dict) – Data to create a new actuator (usually generated by serialize())

#### Returns

A new Text object.

#### Return type

**Text** 

#### Example:

```
title = base.Text.load( previous_title.serialize() )
```

## static magenta(message)

This method works exactly the way green\_bright() work with different color.

## static magenta\_bright(message)

This method works exactly the way green\_bright() work with different color.

## static magenta\_dim(message)

This method works exactly the way green\_bright() work with different color.

```
notify(modifier = None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## parent

This object's parent. It needs to be a BoardItem.

## print\_formatted()

Print the text with the current font activated.

New in version 1.3.0.

If the font is not set, it is strictly equivalent to use Python's print(text\_object).

## static print\_white\_on\_red(message)

Print a white message over a red background.

#### **Parameters**

**message** (str) – The message to print.

Example:

```
base.Text.print_white_on_red("This is bright!")
```

#### static red(message)

This method works exactly the way green\_bright() work with different color.

## static red\_bright(message)

This method works exactly the way green bright() work with different color.

## static red\_dim(message)

This method works exactly the way green\_bright() work with different color.

```
render_to_buffer(buffer, row, column, buffer_height, buffer_width)
```

Render the Text object from the display buffer to the frame buffer.

New in version 1.3.0.

This method is automatically called by *pygamelib*.engine.Screen.render().

# **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (*int*) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (int) – the screen column

## Return type

int

# property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

## Return type

int

3.3. base 255

#### serialize()

Return a dictionary with all the attributes of this object.

New in version 1.3.0.

#### Returns

A dictionary with all the attributes of this object.

## Return type

dict

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

#### style

The style attribute sets the style of the text. It needs to be a str.

## property text

The text attribute. It needs to be a str.

New in version 1.3.0.

When the text is changed, the observers are notified of the change with the pygamelib.base.Text.text:changed event. The new text is passed as the *value* parameter.

## static warn(message)

Print a warning message.

The warning is a regular message prefixed by WARNING in black on a yellow background.

## **Parameters**

**message** (str) – The message to print.

Example:

```
base.Text.warn("This is a warning.")
```

## static white(message)

This method works exactly the way green\_bright() work with different color.

## static white\_bright(message)

This method works exactly the way green\_bright() work with different color.

## static white\_dim(message)

This method works exactly the way green\_bright() work with different color.

#### static yellow(message)

This method works exactly the way green\_bright() work with different color.

```
static yellow_bright(message)
```

This method works exactly the way green\_bright() work with different color.

```
static yellow_dim(message)
```

This method works exactly the way green\_bright() work with different color.

## 3.3.12 Vector2D

class pygamelib.base.Vector2D(row=0.0, column=0.0)

Bases: object

A 2D vector class.

New in version 1.2.0.

Contrary to the rest of the library Vector2D uses floating point numbers for its coordinates/direction/orientation. However since the rest of the library uses integers, the numbers are rounded to 2 decimals. You can alter that behavior by increasing or decreasing the rounding\_precision parameter (if you want integer for example).

Vector2D use the row/column internal naming convention as it is easier to visualize for developers that are still learning python or the pygamelib. If it is a concept that you already understand and are more familiar with the x/y coordinate system you can also use x and y.

- x is equivalent to column
- y is equivalent to row

Everything else is the same.

Vectors can be printed and supports basic operations:

- addition
- substraction
- · multiplication

Let's elaborate a bit more on the multiplication. The product behaves in 2 different ways:

If you multiply a vector with a scalar (int or float), the return value is a Vector2D with each vector component multiplied by said scalar.

If you multiply a Vector2D with another Vector2D you ask for the the cross product of vectors. This is an undefined mathematical operation in 2D as the cross product is supposed to be perpendicular to the 2 other vectors (along the z axis in our case). Since we don't have depth (z) in 2D, this will return the magnitude of the signed cross product of the 2 vectors.

Example of products:

```
v1 = base.Vector2D(1,2)
v2 = base.Vector2D(3,4)
# This returns -2
mag = v1 * v2
# This returns a Vector2D with values (-1, -2)
inv = v1 * -1
# This return a Vector2D with values (2.85, 3.8) or 95% of v2
dim = v2 * 0.95
```

## **Parameters**

3.3. base 257

- row (int) The row/y parameter.
- **column** (*int*) The column/x parameter.

## Example:

```
gravity = Vector2D(9.81, 0)
# Remember that minus on row is up.
speed = Vector2D(-0.123, 0.456)
# In that case you might want to increase the rounding precision
speed.rounding_precision = 3
```

```
__init__(row=0.0, column=0.0)
```

## **Methods**

init([row, column])	
<pre>from_direction(direction, step)</pre>	Build and return a Vector2D from a direction.
length()	Returns the length of a vector.
load(data)	Loads a vector from a dictionary.
serialize()	Returns a dictionary with the attributes of the vector.
unit()	Returns a normalized unit vector.

## **Attributes**

column	The column component of the vector.
row	The row component of the vector.
X	x is an alias for column.
у	y is an alias for row.
rounding_precision	The rounding_precision attributes is used when vectors values are calculated and the result rounded for convenience.

# property column

The column component of the vector.

```
classmethod from_direction(direction: Direction, step)
```

Build and return a Vector2D from a direction.

Directions are from the constants module.

## **Parameters**

- **direction** (*Direction*) A direction from the constants module.
- **step** (*int*) The number of cell to cross in one movement.

## Example:

```
v2d_up = Vector2D.from_direction(Direction.UP, 1)
```

## length()

Returns the length of a vector.

## **Return type**

float

Example:

```
if speed.length() == 0.0:
    print('We are not moving... at all...')
```

## classmethod load(data)

Loads a vector from a dictionary.

New in version 1.3.0.

## **Parameters**

**data** (*dict*) – A dictionary with the attributes of the vector.

#### **Returns**

A vector.

## Return type

Vector2D

Example:

```
gravity_dict = {"row": 9.81, "column": 0}
gravity = Vector2D.load(gravity_dict)
```

## rounding\_precision

The rounding\_precision attributes is used when vectors values are calculated and the result rounded for convenience. It can be changed anytime to increase or decrease the precision anytime.

## property row

The row component of the vector.

## serialize()

Returns a dictionary with the attributes of the vector.

New in version 1.3.0.

#### Returns

A dictionary with the attributes of the vector.

## Return type

dict

Example:

```
gravity = Vector2D(9.81, 0)
gravity_dict = gravity.serialize()
print(gravity_dict)
```

## unit()

Returns a normalized unit vector.

#### **Returns**

A unit vector

3.3. base 259

#### **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

#### property x

x is an alias for column.

## property y

y is an alias for row.

# 3.3.13 Deprecated objects

These are the deprecated objects of the pygamelib.base module. They should not be used as they are going to be removed in future versions.

## **HacException**

```
exception pygamelib.base.HacException(error, message)
```

A simple forward to PglException

Deprecated since version 1.3.0.

## HacInvalidLevelException

## exception pygamelib.base.HacInvalidLevelException(message)

Forward to PglInvalidLevelException

Deprecated since version 1.3.0.

# HacInvalidTypeException

# exception pygamelib.base.HacInvalidTypeException(message)

A simple forward to PglInvalidTypeException

Deprecated since version 1.3.0.

## **HacObjectIsNotMovableException**

## exception pygamelib.base.HacObjectIsNotMovableException(message)

Simple forward to PglObjectIsNotMovableException

Deprecated since version 1.3.0.

260

## HacOutOfBoardBoundException

exception pygamelib.base.HacOutOfBoardBoundException(message)

Simple forward to PglOutOfBoardBoundException

Deprecated since version 1.3.0.

# 3.4 board items

# 3.4.1 Actionable

class pygamelib.board\_items.Actionable(action=None, action\_parameters=None, perm=None, \*\*kwargs)
 Bases: Immovable

This class derives *Immovable*. It adds the ability to an Immovable BoardItem to be triggered and execute some code.

If an actionable board item is activated by an item (this mechanism is taken care of by the Board class), the function passed as the *action* parameter is called with *action\_parameters* as parameters. Subclass may implement a different mechanism for activation so please read their documentations.

#### **Parameters**

- **action** (*function*) the reference to a function (Attention: no parentheses at the end of the function name). It needs to be callable.
- **action\_parameters** (*list*) the parameters to the action function.
- **perm** (*constants*) The permission that defines what types of items can actually activate the actionable. The permission has to be one of the permissions defined in *constants*. By default it is set to Permission.PLAYER AUTHORIZED.

On top of these parameters Actionable accepts all parameters from Immovable and therefor from BoardItem.

**Note:** The common way to use this class is to use GenericActionableStructure. Please refer to GenericActionableStructure for more details.

**Important:** There's a complete tutorial about Actionable items on the pygamelib wiki

\_\_init\_\_(action=None, action\_parameters=None, perm=None, \*\*kwargs)

Like the object class, this class constructor takes no parameter.

3.4. board items 261

# Methods

init([action, action_parameters, perm])	Like the object class, this class constructor takes no
	parameter.
activate()	This function is calling the action function with the
	action_parameters.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
<pre>debug_info()</pre>	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
_	observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
	wise.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this ob-
V	ject.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
store_position(row, column[, layer])	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.
	1 J

262 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column
	of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as
	a 2 dimensions list.
width	Convenience method to get the width of the item.

#### activate()

This function is calling the action function with the action\_parameters.

The *action* callback function should therefor have a signature like:

def my\_callback\_function(actionable, action\_parameters)

With actionable being the Actionable current reference to self.

Usually it's automatically called by move() when a Player or NPC (see board\_items)

## property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

Example:

3.4. board\_items 263

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

#### Returns

False

## Return type

bool

**collides\_with**(other, projection\_offset: Vector2D = None)

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# Return type

bool

#### Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

## Return type

int

# Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

## **Return type**

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### **Returns**

True or False depending on the success of the operation.

#### Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

# Return type

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*Pg1Base0bject*) The object that has changed.
- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

3.4. board items 265

#### property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

## Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

## Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

# property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

#### Returns

The size of the item.

# Return type

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

## Returns

The layer number

## Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

#### **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

#### Returns

A new item.

# Return type

~pygamelib.board\_items.BoardItem

#### property model

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

#### pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

3.4. board items 267

```
position_as_vector()
```

Returns the current item position as a Vector2D

#### **Returns**

The position as a 2D vector

#### **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

# render\_to\_buffer(buffer, row, column, height, width)

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

## property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

## Returns

The row coordinate

## Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (int) – the screen column

## Return type

int

# property screen\_row: int A property to get/set the screen row. Parameters

**value** (*int*) – the screen row

**Return type** 

int

 $serialize() \rightarrow dict$ 

Return a dictionary with all the attributes of this object.

Returns

A dictionary with all the attributes of this object.

Return type

dict

set\_can\_move(value)

Set the value of the can\_move property to value.

**Parameters** 

**value** (*boo1*) – The value to set.

Example:

```
item.set_can_move(False)
```

set\_overlappable(value)

Set the value of the overlappable property to value.

**Parameters** 

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

set\_pickable(value)

Set the value of the pickable property to value.

**Parameters** 

**value** (*boo1*) – The value to set.

Example:

```
item.set_pickable(False)
```

set\_restorable(value)

Set the value of the restorable property to value.

**Parameters** 

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

3.4. board\_items 269

#### property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

## **Return type**

list

## Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
        f"{item.name} cannot be placed at {item.pos}."
    )
```

```
store_position(row: int, column: int, layer: int = 0)
```

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

## Example:

```
item.store_position(3,4)
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- column(int) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

#### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

# Return type

int

# Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

## 3.4.2 ActionableTile

class pygamelib.board\_items.ActionableTile(\*\*kwargs)

Bases: Actionable, Tile

The ActionableTile is the complex (i.e: multi-cells items) version of the *GenericActionableStructure*. It allows you to create any type of in game object that is represented with more than one character in the terminal and that is *Actionable*. Actionable object have a callback system that is automatically called when the player collide with the object.

**Important:** There's a complete tutorial about Actionable items on the pygamelib wiki

```
__init__(**kwargs)
```

Please have a look at the documentation for *Tile* and *Actionable* for the list of possible constructor's parameters.

3.4. board\_items 271

# Methods

Please have a look at the documentation for <i>Tile</i> and <i>Actionable</i> for the list of possible constructor's parameters.
This function is calling the action function with the action_parameters.
Attach an observer to this instance.
A Tile cannot move.
Tells if this item collides with another item.
Return a string with the list of the attributes and their current value.
Detach an observer from this instance.
Print the model WITHOUT carriage return.
Calculates the distance with an item.
A virtual method that needs to be implemented by the observer.
Return the item component at the row, column position if it is within the complex item's boundaries.
Load data and create a new Tile out of it.
Notify all the observers that a change occurred.
Returns True if the item is overlappable, False otherwise.
Returns True if the item is pickable, False otherwise.
Returns the current item position as a Vector2D
Render the complex board item from the display buffer to the frame buffer.
Returns True if the item is restorable, False otherwise.
Return a dictionary with all the attributes of this object.
Set the value of the can_move property to value.
Set the value of the overlappable property to value.
Set the value of the pickable property to value.
Set the value of the restorable property to value.
Store the BoardItem position for self access.
Store the screen position of the object.
Update the complex item with the current sprite.

272 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column
	of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of
	the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as
	a 2 dimensions list.
sprite	A property to easily access and update a complex item's sprite.
width	Convenience method to get the width of the item.

## activate()

This function is calling the action function with the action\_parameters.

The action callback function should therefor have a signature like:

def my\_callback\_function(actionable, action\_parameters)

With actionable being the Actionable current reference to self.

Usually it's automatically called by move() when a Player or NPC (see board\_items)

## property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

# **Return type**

bool

3.4. board\_items 273

#### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## can\_move()

A Tile cannot move.

#### Returns

False

#### Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- projection\_offset (Vector2D) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## Return type

bool

## Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

## Return type

int

## Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

## **Return type**

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### **Returns**

True or False depending on the success of the operation.

#### Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

# Return type

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*Pg1Base0bject*) The object that has changed.
- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

3.4. board items 275

#### property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

## Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### **Returns**

The height

#### Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

## Returns

The size of the item.

## Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

## **Return type**

~pygamelib.board\_items.BoardItem

#### Raises

**Pg10ut0fBoardBoundException** – if row or column are out of bound.

#### property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

## Return type

int

## Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

## classmethod load(data)

Load data and create a new Tile out of it.

#### **Parameters**

**data** (dict) – Data to create a new tile (usually generated by serialize())

#### Returns

A new complex npc.

## **Return type**

~pygamelib.board\_items.Tile

## property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

## **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

## property particle\_emitter

3.4. board items 277

#### pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
   print('The item is pickable')
```

#### position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

## render\_to\_buffer(buffer, row, column, height, width)

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

#### restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

## property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

## Returns

The row coordinate

## **Return type**

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

```
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize() \rightarrow dict
     Return a dictionary with all the attributes of this object.
         Returns
             A dictionary with all the attributes of this object.
         Return type
             dict
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
set_overlappable(value)
     Set the value of the overlappable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_overlappable(False)
set_pickable(value)
     Set the value of the pickable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_pickable(False)
```

3.4. board\_items 279

set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

## Return type

list

Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
        f"{item.name} cannot be placed at {item.pos}."
    )
```

## property sprite

A property to easily access and update a complex item's sprite.

## **Parameters**

**new\_sprite** (Sprite) – The sprite to set

Example:

```
npc1 = board_items.ComplexNpc(
                                sprite=npc_sprite_collection['npc1_idle']
                            )
# to access the sprite:
if npc1.sprite.width * npc1.sprite.height > CONSTANT_BIG_GUY:
   game.screen.place(
       base.Text(
            'Big boi detected!!!',
            core.Color(255,0,0),
            style=TextStyle.BOLD,
       notifications.row,
       notifications.column,
   )
# And to set it:
if game.player in game.neighbors(3, npc1):
   npc1.sprite = npc_sprite_collection['npc1_fight']
```

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

#### Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

#### Example:

```
an_object.store_screen_coordinate(3,8)
```

## update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

## Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

#### Returns

The width

## Return type

int

Example:

3.4. board items 281

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.3 BoardComplexItem

Bases: BoardItem

New in version 1.2.0.

A BoardComplexItem is the base item for multi cells elements. It inherits from *BoardItem* and accepts all its parameters.

The main difference is that a complex item can use *Sprite* as representation.

You can see a complex item as a collection of other items that are ruled by the same laws. They behave as one but a complex item is actually made of complex components. At first it is not important but you may want to exploit that as a feature for your game.

On top of *BoardItem* the constructor accepts the following parameters:

#### **Parameters**

- **sprite** (*Sprite*) A sprite representing the item.
- **size** (*array[int]*) The size of the item as [WIDTH, HEIGHT]. It impact movement and collision detection amongst other things. If it is left empty the Sprite size is used. If no sprite is given to the constructor the default size is 2x2.
- base\_item\_type (BoardItemComplexComponent) the building block of the complex item. The complex item is built from a 2D array of base items.

#### Null\_sprixel

The null\_sprixel is a bit of a special parameter: during construction a null sprixel is replaced by a BoardItemVoid. This is a trick to show the background (i.e transparency). A sprixel can take the color of the background but a complex item with a null\_sprixel that correspond to transparent zone of a sprite will really be transparent and show the background.

## Null\_sprixel

Sprixe1

**\_\_init\_\_**(sprite=None, size=None, null\_sprixel=None, base\_item\_type=None, \*\*kwargs)

Like the object class, this class constructor takes no parameter.

# Methods

init([sprite, size, null_sprixel,])	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Returns True if the item can move, False otherwise.
collides_with(other[, projection_offset])	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their
debug_Info()	current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>item(row, column)</pre>	Return the item component at the row, column position if it is within the complex item's boundaries.
load(data)	Load data and create a new BoardComplexItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the complex board item from the display buffer to the frame buffer.
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
store_position(row, column[, layer])	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.
update_sprite()	Update the complex item with the current sprite.

3.4. board\_items 283

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
sprite	A property to easily access and update a complex item's sprite.
width	Convenience method to get the width of the item.

## property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

284 Chapter 3. Tutorials

#### can\_move()

Returns True if the item can move, False otherwise.

Example:

```
if board.item(4,5).can_move():
    print('The item can move')
```

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

## **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

## Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

## Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

## Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

## **Return type**

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

#### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

#### Returns

The size of the item.

## Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

## Return type

~pygamelib.board\_items.BoardItem

#### Raises

**PglOutOfBoardBoundException** – if row or column are out of bound.

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

The layer number

# Return type

int

## Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

## classmethod load(data)

Load data and create a new BoardComplexItem out of it.

#### **Parameters**

**data** (dict) – Data to create a new complex item (usually generated by serialize())

#### Returns

A new complex item.

## **Return type**

~pygamelib.board\_items.BoardComplexItem

## property model

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
   print('The item is overlappable')
```

## property particle\_emitter

# pickable()

Returns True if the item is pickable, False otherwise.

```
if board.item(4,5).pickable():
   print('The item is pickable')
```

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

#### **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

#### render\_to\_buffer(buffer, row, column, height, width)

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
   print('The item is restorable')
```

#### property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

# Returns

The row coordinate

#### Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

## property screen\_column: int

A property to get/set the screen column.

**Parameters** 

Example:

**value** (int) – the screen column

```
Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize() \rightarrow dict
     Return a dictionary with all the attributes of this object.
         Returns
             A dictionary with all the attributes of this object.
         Return type
             dict
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
set_overlappable(value)
     Set the value of the overlappable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_overlappable(False)
set_pickable(value)
     Set the value of the pickable property to value.
         Parameters
             value (boo1) – The value to set.
     Example:
     item.set_pickable(False)
set_restorable(value)
     Set the value of the restorable property to value.
         Parameters
             value (bool) – The value to set.
```

290 Chapter 3. Tutorials

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

## Return type

list

## Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
          f"{item.name} cannot be placed at {item.pos}."
    )
```

## property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

**new\_sprite** (Sprite) – The sprite to set

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.

• **layer** – the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

#### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

#### Return type

int

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.4 BoardItemComplexComponent

class pygamelib.board\_items.BoardItemComplexComponent(\*\*kwargs)

Bases: BoardItem

The default component of a complex item.

It is literally just a BoardItem but is subclassed for easier identification.

It is however scanning its parent for the item's basic properties (overlappable, restorable, etc.)

A component can never be pickable by itself.

\_\_init\_\_(\*\*kwargs)

Like the object class, this class constructor takes no parameter.

#### **Methods**

init(**kwargs)	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Returns True if the item can move, False otherwise.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns False.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
store_position(row, column[, layer])	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.
	z z z z z z z z z z z z z z z z z z z

## **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

#### property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## can\_move()

Returns True if the item can move, False otherwise.

```
if board.item(4,5).can_move():
    print('The item can move')
```

**collides\_with**(other, projection\_offset: Vector2D = None)

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## Return type

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

#### property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

#### Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

#### Return type

str

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

#### Return type

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

## Returns

The heading of the item.

#### **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

#### property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

## Returns

The size of the item.

## Return type

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

#### **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

A new item.

# Return type

~pygamelib.board\_items.BoardItem

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

## property particle\_emitter

## pickable()

Returns False. A component is never pickable by itself (either the whole complex item is pickable or not, but not partially)

Example:

```
if item.item(4,5).pickable():
    print('The item is pickable')
```

# position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

#### Return type

Vector2D

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
```

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

## property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### Returns

The row coordinate

#### **Return type**

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

## property screen\_column: int

A property to get/set the screen column.

## **Parameters**

**value** (int) – the screen column

# Return type

int

# property screen\_row: int

A property to get/set the screen row.

## **Parameters**

**value** (int) – the screen row

# Return type

int

#### **serialize()** $\rightarrow$ dict

Return a dictionary with all the attributes of this object.

A dictionary with all the attributes of this object.

# Return type

dict

## set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

#### Return type

list

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

#### Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## **Returns**

The width

## Return type

int

# Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.5 BoardItem

Bases: PglBaseObject

Base class for any item that will be placed on a Board.

#### **Parameters**

- **type** (*str*) A type you want to give your item. It can be any string. You can then use the type for sorting or grouping for example.
- **name** (*str*) A name for this item. For identification purpose.
- **pos** (*list*) the position of this item. When the item is managed by the Board and Game engine this member hold the last updated position of the item. It is not updated if you manually move the item. It must be an array of 2 integers [row,column]
- **model** (*str*) The model to use to display this item on the Board. Be mindful of the space it will require. Default value is '\*'. This parameter is now deprecated in favor of "sprixel". If both "sprixel" and "model" are specified, "model" is ignored.
- parent The parent object of the board item. Usually a Board or Game object.
- **sprixel** (*Sprixel*) The sprixel that will represent the item on the Board.
- **pickable** (*bool*) Represent the capacity for a BoardItem to be pick-up by player or NPC. This parameter is True or False. If sets to None, it'll be set to False.
- **overlappable** (*bool*) Represent to be overlapped by another BoardItem. This parameter is True or False. If sets to None, it'll be set to False.
- **restorable** (*bool*) Represent the capacity for an Immovable BoardItem to be restored by the board if the item is overlappable and has been overlapped by another BoardItem. This parameter is True or False. If sets to None, it'll be set to False.
- **can\_move** (*bool*) Represent the ability of the BoardItem to move on the Board. If this parameter is False, the Board.move() method will not allow the item to move. This parameter is True or False. If sets to None, it'll be set to False.
- **pos** The position of the BoardItem on a *Board*. Please make sure that you understand what you do before changing that parameter. The position of an item is managed by the Board object and will be updated. In most cases you don't need to use that parameter. The position is a list of 2 or 3 int: [row, column, layer].
- **value** (*int | float*) The value of an item. It can be used for any game purpose: a score indicator, a trade value, the amount of XP to grant to a player on a kill, etc.
- **inventory\_space** (*int*) The space that the item takes in the *pygamelib.engine*. *Inventory*. This parameter used to be available only for *Immovable* items but since 1.3.0, every BoardItem can be configured to be pickable, so every BoardItem can now take space in the inventory. Default value is 1.
- **animation** (*Animation*) An animation to animate the item sprixel.
- particle\_emitter (ParticleEmitter) A particle emitter that is attached to this item.

**Note:** Starting with version 1.2.0 and introduction of complex items, BoardItems have a size. That size **CANNOT** be set. It is always 1x1. This is because a BoardItem always takes 1 cell, regardless of its actual number of characters. The size is a read-only property.

**Important:** In version 1.3.0 the BoardItem object has been reworked to make sure that the pickable, restorable, overlappable and can\_move properties are configurable for all items independently of their type. This fixes an issue with restorable: only *Immovable* objects could be restorable. Now all items can be any combination of these properties. As a developer you are now encouraged to use the corresponding functions to determine the abilities of an item.

**Warning:** An item cannot be restorable and pickable at the same time. If it's pickable, it's put into the inventory of the item overlapping it. Therefor, it cannot be restored. If both restorable and pickable are set to True, one of the 2 is set to False depending on the value of overlappable: if True restorable is set to True and pickable to False and the contrary if overlappable is False.

\_\_init\_\_(sprixel=None, model=None, name=None, item\_type=None, parent=None, pickable=False, overlappable=False, restorable=False, can\_move=False, pos=None, value=None, inventory\_space=1, animation: Animation = None, particle\_emitter=None)

Like the object class, this class constructor takes no parameter.

# Methods

init([sprixel, model, name, item_type,])	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Returns True if the item can move, False otherwise.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
<pre>debug_info()</pre>	Return a string with the list of the attributes and their
	current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
	observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
	wise.
pickable()	Returns True if the item is pickable, False otherwise.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the board item into a display buffer (not a
	screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this ob-
	ject.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

304 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

## property animation

A property to get and set an *Animation* for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

## Returns

True or False depending on the success of the operation.

#### **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## can\_move()

Returns True if the item can move, False otherwise.

Example:

```
if board.item(4,5).can_move():
    print('The item can move')
```

**collides\_with**(other, projection\_offset: Vector2D = None)

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## Return type

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

#### property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

## Returns

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

#### Return type

str

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBase0bject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

#### Return type

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

## Returns

The heading of the item.

#### **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

#### property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

## Returns

The size of the item.

# Return type

int

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

#### **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

A new item.

# Return type

~pygamelib.board\_items.BoardItem

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

# pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
     Render the board item into a display buffer (not a screen buffer).
     This method is automatically called by pygamelib.engine.Screen.render().
         Parameters
             • buffer (numpy.array) – A screen buffer to render the item into.
             • row (int) – The row to render in.
             • column (int) – The column to render in.
             • height (int) – The total height of the display buffer.
             • width (int) – The total width of the display buffer.
restorable()
     Returns True if the item is restorable, False otherwise.
     Example:
     if board.item(4,5).restorable():
         print('The item is restorable')
property row
     Convenience method to get the current stored row of the item.
     This is absolutely equivalent to access to item.pos[0].
         Returns
             The row coordinate
         Return type
             int
     Example:
     if item.row != item.pos[0]:
         print('Something extremely unlikely just happened...')
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
```

Return a dictionary with all the attributes of this object.

value (int) – the screen row

**Parameters** 

Return type int

**serialize()**  $\rightarrow$  dict

A dictionary with all the attributes of this object.

# Return type

dict

## set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

# **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

# **Return type**

list

Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
          f"{item.name} cannot be placed at {item.pos}."
    )
```

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

#### Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## Returns

The width

## Return type

int

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.6 BoardItemVoid

class pygamelib.board\_items.BoardItemVoid(\*\*kwargs)

Bases: BoardItem

A class that represent a void cell.

\_\_init\_\_(\*\*kwargs)

Like the object class, this class constructor takes no parameter.

# **Methods**

init(**kwargs)	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Returns True if the item can move, False otherwise.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	A BoardItemVoid is obviously overlappable (so player and NPC can walk over).
pickable()	A BoardItemVoid is not pickable, therefor this method return false.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

## **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

#### property animation

A property to get and set an *Animation* for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

## **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## can\_move()

Returns True if the item can move, False otherwise.

```
if board.item(4,5).can_move():
    print('The item can move')
```

**collides\_with**(other, projection\_offset: Vector2D = None)

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## Return type

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

#### property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

#### Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

#### Return type

str

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

#### Return type

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

## Returns

The heading of the item.

#### **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

## Returns

The size of the item.

## Return type

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

#### **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

A new item.

# Return type

~pygamelib.board\_items.BoardItem

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

#### overlappable()

A BoardItemVoid is obviously overlappable (so player and NPC can walk over).

## **Returns**

True

## property particle\_emitter

# pickable()

A BoardItemVoid is not pickable, therefor this method return false.

#### **Returns**

False

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## Return type

Vector2D

## Example:

318

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

# render\_to\_buffer(buffer, row, column, height, width)

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

```
• buffer (numpy.array) – A screen buffer to render the item into.
```

- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

#### restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
   print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### Returns

The row coordinate

## Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

## property screen\_column: int

A property to get/set the screen column.

## **Parameters**

**value** (int) – the screen column

# Return type

int

#### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

## Return type

int

## $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

#### Returns

A dictionary with all the attributes of this object.

## Return type

dict

```
set_can_move(value)
```

Set the value of the can\_move property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_can_move(False)
```

## set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_restorable(False)
```

# property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

## **Return type**

list

```
store_position(row: int, column: int, layer: int = 0)
```

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

## Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

### Returns

The width

# Return type

int

### Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

### 3.4.7 Camera

## class pygamelib.board\_items.Camera(actuator=None, \*\*kwargs)

Bases: Movable

New in version 1.3.0.

A Camera is a special item: it does not appear on the Board and actually is not even registered on it. It is only an item that you can center the board on (when using partial display). It helps for cut scenes for example.

The main difference with a regular BoardItem is that the row and column properties are writable. This means that you can directly manipulate its coordinates and partially render a huge board around that focal point.

The *Screen* buffer rendering system introduced in version 1.3.0 require a board item to be declared as the focus point of the board if partial display is enabled.

The Camera object inherits from Movable and can accept an actuator parameter. However, it is up to the developer to activate the actuators mechanics as the Camera object does not register as a NPC or a Player. The support for actuators is mainly thought for pre-scripted cut-scenes.

### Example:

```
# This example leverage the Screen buffer system introduced in v1.3.0.
# It pans the camera over a huge map. The Screen.update() method automatically
# uses the Board.partial_display_focus coordinates to adjust the displayed area.
camera = Camera()
huge_board.partial_display_focus = camera
while camera.column < huge_board.width:
    camera.column += 1
    game.screen.update()</pre>
```

```
__init__(actuator=None, **kwargs)
```

Like the object class, this class constructor takes no parameter.

## **Methods**

init([actuator])	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
collides_with(other[, projection_offset])	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
has_inventory()	This is a virtual method that must be implemented in deriving class.
load(data)	Load data and create a new Movable out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Serialize the Immovable object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

322 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### $can\_move() \rightarrow bool$

Movable implements can\_move().

### Returns

True

# Return type

Boolean

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### **Returns**

The column coordinate

## Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

### debug\_info()

Return a string with the list of the attributes and their current value.

## Return type

str

# detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

### **Returns**

The distance between this item and the other.

### Return type

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

### property dtmove

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

# **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## $has\_inventory() \rightarrow bool$

This is a virtual method that must be implemented in deriving class. This method has to return True or False. This represent the capacity for a Movable to have an inventory.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

### Returns

The size of the item.

## Return type

int

### property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

## classmethod load(data)

Load data and create a new Movable out of it.

### **Parameters**

**data** (dict) – Data to create a new movable item (usually generated by serialize())

#### Returns

A new complex item.

### Return type

~pygamelib.board\_items.Movable

# property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

## property particle\_emitter

## pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

### position\_as\_vector()

Returns the current item position as a Vector2D

## Returns

The position as a 2D vector

### Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

render\_to\_buffer(buffer, row, column, height, width)

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by pygamelib.engine.Screen.render().

### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

## Returns

The row coordinate

# Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

### **Parameters**

**value** (*int*) – the screen column

# Return type

int

### property screen\_row: int

A property to get/set the screen row.

## **Parameters**

value (int) – the screen row

### **Return type**

int

### **serialize()** $\rightarrow$ dict

Serialize the Immovable object.

This returns a dictionary that contains all the key/value pairs that makes up the object.

# set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

### **Returns**

The size.

### Return type

list

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## Returns

The width

## Return type

int

Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.8 Character

Bases: Movable

A base class for a character (playable or not)

#### **Parameters**

- **agility** (*int*) Represent the agility of the character
- **attack\_power** (*int*) Represent the attack power of the character.
- **defense\_power** (*int*) Represent the defense\_power of the character
- **hp** (*int*) Represent the hp (Health Point) of the character
- intelligence (int) Represent the intelligence of the character
- max\_hp (int) Represent the max\_hp of the character
- max\_mp (int) Represent the max\_mp of the character
- mp (int) Represent the mp (Mana/Magic Point) of the character
- **remaining\_lives** (*int*) Represent the remaining\_lives of the character. For a NPC it is generally a good idea to set that to 1. Unless the NPC is a multi phased boss.
- **strength** (*int*) Represent the strength of the character

These characteristics are here to be used by the game logic but very few of them are actually used by the Game (*pygamelib.engine*) engine.

\_\_init\_\_(max\_hp=None, hp=None, max\_mp=None, mp=None, remaining\_lives=None, attack\_power=None, defense\_power=None, strength=None, intelligence=None, agility=None, \*\*kwargs)

Like the object class, this class constructor takes no parameter.

# Methods

init([max_hp, hp, max_mp, mp,])	Like the object class, this class constructor takes no
1/1	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
<pre>debug_info()</pre>	Return a string with the list of the attributes and their current value.
<pre>detach(observer)</pre>	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
has_inventory()	This is a virtual method that must be implemented in deriving class.
load(data)	Load data and create a new Character out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Serialize the Character object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
store_position(row, column[, layer])	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

332 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### $can\_move() \rightarrow bool$

Movable implements can\_move().

### Returns

True

# Return type

Boolean

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### **Returns**

The column coordinate

## Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

### debug\_info()

Return a string with the list of the attributes and their current value.

## Return type

str

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

# **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

### Returns

The distance between this item and the other.

### Return type

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

### property dtmove

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

# **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## $has\_inventory() \rightarrow bool$

This is a virtual method that must be implemented in deriving class. This method has to return True or False. This represent the capacity for a Movable to have an inventory.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

### Returns

The size of the item.

## Return type

int

### property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

## Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

## classmethod load(data)

Load data and create a new Character out of it.

#### **Parameters**

**data** (dict) – Data to create a new character item (usually generated by serialize())

### Returns

A new character item.

### Return type

~pygamelib.board\_items.Character

### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

## property particle\_emitter

## pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

### position\_as\_vector()

Returns the current item position as a Vector2D

## Returns

The position as a 2D vector

### Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

render\_to\_buffer(buffer, row, column, height, width)

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

## Returns

The row coordinate

# Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

### **Parameters**

value(int) – the screen column

# Return type

int

### property screen\_row: int

A property to get/set the screen row.

# **Parameters**

value (int) – the screen row

### **Return type**

int

### $serialize() \rightarrow dict$

Serialize the Character object.

This returns a dictionary that contains all the key/value pairs that makes up the object.

# set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

### **Returns**

The size.

### Return type

list

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## Returns

The width

## Return type

int

Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.9 ComplexDoor

class pygamelib.board\_items.ComplexDoor(\*\*kwargs)

Bases: Door, BoardComplexItem

New in version 1.2.0.

A complex door is nothing more than a *Door* mashed with a *BoardComplexItem*.

It supports all parameters of both with inheritance going first to Door and second to BoardComplexItem.

The main interest is of course the multiple cell representation and the Sprites support.

Example:

```
castle_door = ComplexDoor(
    sprite=sprite_castle_door
)
```

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init(**kwargs)	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
handle_notification(subject[, attribute, value])	A virtual method that needs to be implemented by the
nature_notification(subject[, attribute, value])	observer.
<pre>item(row, column)</pre>	Return the item component at the row, column posi-
	tion if it is within the complex item's boundaries.
load(data)	Load data and create a new ComplexDoor out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
position_as_vector()	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the complex board item from the display buffer to the frame buffer.
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this ob-
	ject.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
update_sprite()	Update the complex item with the current sprite.

342 Chapter 3. Tutorials

# **Attributes**

	A company of the second and the design of the Constitutions
animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column
	of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of
	the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as
	a 2 dimensions list.
sprite	A property to easily access and update a complex
	item's sprite.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

#### Returns

False

## Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## Return type

bool

## Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### Returns

The column coordinate

### Return type

int

## Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

### **Return type**

str

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

## Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

### distance\_to(other)

Calculates the distance with an item.

### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

## **Return type**

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

## Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

### Returns

The size of the item.

## Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

## Return type

~pygamelib.board\_items.BoardItem

### Raises

**PglOutOfBoardBoundException** – if row or column are out of bound.

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

# Return type

int

## Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

## classmethod load(data)

Load data and create a new ComplexDoor out of it.

### **Parameters**

**data** (dict) – Data to create a new complex door (usually generated by serialize())

### Returns

A new complex npc.

## Return type

~pygamelib.board\_items.ComplexDoor

## property model

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
   print('The item is overlappable')
```

## property particle\_emitter

# pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

### **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

### render\_to\_buffer(buffer, row, column, height, width)

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
   print('The item is restorable')
```

### property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

### Returns

The row coordinate

### Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

## property screen\_column: int

A property to get/set the screen column.

```
Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize() \rightarrow dict
     Return a dictionary with all the attributes of this object.
         Returns
             A dictionary with all the attributes of this object.
         Return type
             dict
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
set_overlappable(value)
     Set the value of the overlappable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_overlappable(False)
set_pickable(value)
     Set the value of the pickable property to value.
         Parameters
             value (boo1) – The value to set.
     Example:
     item.set_pickable(False)
set_restorable(value)
     Set the value of the restorable property to value.
         Parameters
             value (bool) – The value to set.
```

3.4. board\_items 349

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

### Return type

list

## Example:

### property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

**new\_sprite** (*Sprite*) – The sprite to set

Example:

```
npc1 = board_items.ComplexNpc(
                                sprite=npc_sprite_collection['npc1_idle']
                            )
# to access the sprite:
if npc1.sprite.width * npc1.sprite.height > CONSTANT_BIG_GUY:
   game.screen.place(
        base.Text(
            'Big boi detected!!!',
            core.Color(255,0,0),
            style=TextStyle.BOLD,
        notifications.row,
       notifications.column,
   )
# And to set it:
if game.player in game.neighbors(3, npc1):
   npc1.sprite = npc_sprite_collection['npc1_fight']
```

## $store_position(row: int, column: int, layer: int = 0)$

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.

• **layer** – the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

# Return type

int

Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.10 ComplexNPC

class pygamelib.board\_items.ComplexNPC(\*\*kwargs)

```
Bases: NPC, BoardComplexItem
```

New in version 1.2.0.

A complex NPC is nothing more than a NPC mashed with a BoardComplexItem.

It supports all parameters of both with inheritance going first to NPC and second to BoardComplexItem.

The main interest is of course the multiple cell representation and the Sprites support.

Example:

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init(**kwargs)	Like the object class, this class constructor takes no
1111 t(	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
collides_with(other[, projection_offset])	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their
	current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
has_inventory()	Define if the NPC has an inventory.
item(row, column)	Return the item component at the row, column posi-
z com(row, coramin)	tion if it is within the complex item's boundaries.
load(data)	Load data and create a new ComplexNPC out of it.
notify([modifier, attribute, value])	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
	wise.
pickable()	Define if the NPC is pickable.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the complex board item from the display
	buffer to the frame buffer.
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Serialize the NPC object.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
<pre>update_sprite()</pre>	Update the complex item with the current sprite.

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
4	of the item.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
sprite	A property to easily access and update a complex item's sprite.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

```
can\_move() \rightarrow bool
```

Movable implements can\_move().

### Returns

True

## Return type

Boolean

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

# Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

## Returns

The column coordinate

### Return type

int

### Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

### Return type

str

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

### **Returns**

True or False depending on the success of the operation.

### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

### distance\_to(other)

Calculates the distance with an item.

### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

## **Return type**

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## property dtmove

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# has\_inventory()

Define if the NPC has an inventory.

This method returns false because the game engine doesn't manage NPC inventory yet but it could be in the future. It's a good habit to check the value returned by this function.

#### Returns

False

#### Return type

Boolean

Example:

```
if mynpc.has_inventory():
    print("Cool: we can pickpocket that NPC!")
else:
    print("No pickpocketing XP for us today :(")
```

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

# **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

#### Returns

The size of the item.

#### Return type

int

## item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

## Return type

~pygamelib.board\_items.BoardItem

#### Raises

**PglOutOfBoardBoundException** – if row or column are out of bound.

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

# classmethod load(data)

Load data and create a new ComplexNPC out of it.

#### **Parameters**

data (dict) - Data to create a new complex npc (usually generated by serialize())

# Returns

A new complex npc.

# Return type

~pygamelib.board\_items.ComplexNPC

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

## pickable()

Define if the NPC is pickable.

Obviously this method always return False.

#### Returns

False

## Return type

Boolean

Example:

```
if mynpc.pickable():
    Utils.warn("Something is fishy, that NPC is pickable"
        "but is not a Pokemon...")
```

# position\_as\_vector()

Returns the current item position as a Vector2D

### **Returns**

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

## render\_to\_buffer(buffer, row, column, height, width)

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.

```
• height (int) – The total height of the display buffer.
```

```
• width (int) – The total width of the display buffer.
```

# restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

#### property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

## Returns

The row coordinate

# Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

value (int) – the screen column

## Return type

int

## property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

# Return type

int

## $serialize() \rightarrow dict$

Serialize the NPC object.

This returns a dictionary that contains all the key/value pairs that makes up the object.

#### set\_can\_move(value)

Set the value of the can\_move property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

## set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_restorable(False)
```

# property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

# Return type

list

Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
        f"{item.name} cannot be placed at {item.pos}."
    )
```

# property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

```
new_sprite (Sprite) – The sprite to set
```

Example:

```
npc1 = board_items.ComplexNpc(
                                sprite=npc_sprite_collection['npc1_idle']
                            )
# to access the sprite:
if npc1.sprite.width * npc1.sprite.height > CONSTANT_BIG_GUY:
   game.screen.place(
        base.Text(
            'Big boi detected!!!',
            core.Color(255,0,0),
            style=TextStyle.BOLD,
        notifications row,
        notifications.column,
   )
# And to set it:
if game.player in game.neighbors(3, npc1):
   npc1.sprite = npc_sprite_collection['npc1_fight']
```

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

## **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

#### update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

## Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

### Returns

The width

# Return type

int

# Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.11 ComplexPlayer

class pygamelib.board\_items.ComplexPlayer(\*\*kwargs)

```
Bases: Player, BoardComplexItem
```

New in version 1.2.0.

A complex player is nothing more than a *Player* mashed with a *BoardComplexItem*.

It supports all parameters of both with inheritance going first to Player and second to BoardComplexItem.

The main interest is of course the multiple cell representation and the Sprites support.

# Example:

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init(**kwargs)	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
<pre>debug_info()</pre>	Return a string with the list of the attributes and their
	current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
	observer.
has_inventory()	This method returns True (a player has an inventory).
<pre>item(row, column)</pre>	Return the item component at the row, column posi-
	tion if it is within the complex item's boundaries.
load(data)	Load data and create a new ComplexPlayer out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
	wise.
pickable()	This method returns False (a player is obviously not
	pickable).
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the complex board item from the display
	buffer to the frame buffer.
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Serialize the Character object.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
<pre>set_pickable(value)</pre>	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.
<pre>update_sprite()</pre>	Update the complex item with the current sprite.

364 Chapter 3. Tutorials

# **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
4	of the item.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
sprite	A property to easily access and update a complex item's sprite.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*Pg1Base0bject*) – An observer to attach to this object.

# Returns

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

# Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

# Returns

The column coordinate

# Return type

int

#### Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

#### Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### **Returns**

True or False depending on the success of the operation.

## Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

## **Return type**

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## property dtmove

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## has\_inventory()

This method returns True (a player has an inventory).

#### property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

# Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### **Returns**

The height

#### Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

# property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

## Returns

The size of the item.

## Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

#### Return type

~pygamelib.board\_items.BoardItem

#### Raises

**PglOutOfBoardBoundException** – if row or column are out of bound.

#### property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

## Return type

int

## Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

# classmethod load(data)

Load data and create a new ComplexPlayer out of it.

#### **Parameters**

**data** (dict) – Data to create a new complex player (usually generated by serialize())

### Returns

A new complex npc.

## **Return type**

~pygamelib.board\_items.ComplexPlayer

### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

# **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

#### overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

### property particle\_emitter

```
pickable()
```

This method returns False (a player is obviously not pickable).

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
```

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

## property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### Returns

The row coordinate

## **Return type**

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

```
Parameters
             value (int) – the screen column
         Return type
              int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
              int
serialize() \rightarrow dict
     Serialize the Character object.
     This returns a dictionary that contains all the key/value pairs that makes up the object.
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
     Set the value of the overlappable property to value.
```

# set\_overlappable(value)

# **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

# set\_pickable(value)

Set the value of the pickable property to value.

## **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

## **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_restorable(False)
```

#### property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

# Return type

list

# Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
        f"{item.name} cannot be placed at {item.pos}."
    )
```

## property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

**new\_sprite** (Sprite) – The sprite to set

Example:

```
npc1 = board_items.ComplexNpc(
                                sprite=npc_sprite_collection['npc1_idle']
# to access the sprite:
if npc1.sprite.width * npc1.sprite.height > CONSTANT_BIG_GUY:
   game.screen.place(
        base.Text(
            'Big boi detected!!!',
            core.Color(255,0,0),
            style=TextStyle.BOLD,
       ),
       notifications.row,
       notifications.column,
   )
# And to set it:
if game.player in game.neighbors(3, npc1):
   npc1.sprite = npc_sprite_collection['npc1_fight']
```

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- $\mathbf{row}$  (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

## Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

#### Example:

```
an_object.store_screen_coordinate(3,8)
```

# update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

# Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

# property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## Returns

The width

# **Return type**

int

## Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.12 ComplexTreasure

class pygamelib.board\_items.ComplexTreasure(\*\*kwargs)

Bases: Treasure, BoardComplexItem

New in version 1.2.0.

A complex treasure is nothing more than a *Treasure* mashed with a *BoardComplexItem*.

It supports all parameters of both with inheritance going first to Treasure and second to BoardComplexItem.

The main interest is of course the multiple cell representation and the Sprites support.

Example:

```
chest = ComplexTreasure(
    sprite=sprite_chest
)
```

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init(**kwargs)	Like the object class, this class constructor takes no
(80)	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>item(row, column)</pre>	Return the item component at the row, column position if it is within the complex item's boundaries.
load(data)	Load data and create a new ComplexTreasure out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	This represent the capacity for a Treasure to be over- lapped by player or NPC.
pickable()	This represent the capacity for a Treasure to be picked-up by player or NPC.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the complex board item from the display buffer to the frame buffer.
restorable()	This represent the capacity for a Treasure to be restored after being overlapped.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
update_sprite()	Update the complex item with the current sprite.

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column
	of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of
	the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as
	a 2 dimensions list.
sprite	A property to easily access and update a complex item's sprite.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

376 Chapter 3. Tutorials

#### can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

#### Returns

False

## Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# Return type

bool

## Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

### Return type

int

# Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

### Return type

str

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### **Returns**

True or False depending on the success of the operation.

#### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

## **Return type**

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

#### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

#### Returns

The size of the item.

## Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

## Return type

~pygamelib.board\_items.BoardItem

#### Raises

**PglOutOfBoardBoundException** – if row or column are out of bound.

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

# Return type

int

# Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

# classmethod load(data)

Load data and create a new ComplexTreasure out of it.

#### **Parameters**

**data** (dict) – Data to create a new complex treasure (usually generated by serialize())

### Returns

A new complex npc.

## Return type

~pygamelib.board\_items.ComplexTreasure

## property model

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

This represent the capacity for a Treasure to be overlapped by player or NPC.

A treasure is not overlappable.

## Returns

False

## Return type

bool

## property particle\_emitter

# pickable()

This represent the capacity for a Treasure to be picked-up by player or NPC.

A treasure is obviously pickable by the player and potentially NPCs. *Board* puts the Treasure in the *Inventory* if the picker implements has\_inventory()

#### Returns

True

## Return type

bool

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

# render\_to\_buffer(buffer, row, column, height, width)

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

## **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (*int*) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

This represent the capacity for a Treasure to be restored after being overlapped.

A treasure is not overlappable, therefor is not restorable.

#### Returns

False

#### Return type

bool

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

### **Returns**

The row coordinate

## **Return type**

int

```
Example:
```

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

## property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (int) – the screen column

## Return type

in

#### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

# Return type

int

### $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

#### Returns

A dictionary with all the attributes of this object.

## Return type

dict

#### set\_can\_move(value)

Set the value of the can\_move property to value.

### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

## **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

## **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### **Returns**

The size.

#### Return type

list

Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
         f"{item.name} cannot be placed at {item.pos}."
    )
```

# property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

```
new_sprite (Sprite) – The sprite to set
```

Example:

```
npc1 = board_items.ComplexNpc(
                                sprite=npc_sprite_collection['npc1_idle']
                            )
# to access the sprite:
if npc1.sprite.width * npc1.sprite.height > CONSTANT_BIG_GUY:
   game.screen.place(
       base.Text(
            'Big boi detected!!!',
            core.Color(255,0,0),
            style=TextStyle.BOLD,
       ),
       notifications.row,
       notifications.column,
   )
# And to set it:
if game.player in game.neighbors(3, npc1):
   npc1.sprite = npc_sprite_collection['npc1_fight']
```

```
store_position(row: int, column: int, layer: int = 0)
```

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

## Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

#### update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

#### Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

# property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

#### **Returns**

The width

```
Return type int
```

Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.13 ComplexWall

```
class pygamelib.board_items.ComplexWall(**kwargs)
```

```
Bases: Wall, BoardComplexItem
```

New in version 1.2.0.

A complex wall is nothing more than a Wall mashed with a BoardComplexItem.

It supports all parameters of both with inheritance going first to Wall and second to BoardComplexItem.

The main interest is of course the multiple cell representation and the Sprites support.

Example:

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init(**kwargs)	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>item(row, column)</pre>	Return the item component at the row, column posi-
	tion if it is within the complex item's boundaries.
load(data)	Load data and create a new ComplexWall out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	This represent the capacity for a <i>BoardItem</i> to be overlapped by player or NPC.
pickable()	This represent the capacity for a <i>BoardItem</i> to be pick-up by player or NPC.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the complex board item from the display buffer to the frame buffer.
restorable()	This represent the capacity for an <i>Immovable</i> Movable item.
serialize()	Return a dictionary with all the attributes of this object.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
<pre>set_overlappable(value)</pre>	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.
update_sprite()	Update the complex item with the current sprite.

386 Chapter 3. Tutorials

# **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
sprite	A property to easily access and update a complex item's sprite.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

#### Returns

False

## Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## Return type

bool

## Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

### Return type

int

## Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

### Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### **Returns**

True or False depending on the success of the operation.

#### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

## **Return type**

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

#### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

#### Returns

The size of the item.

## Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

### Return type

~pygamelib.board\_items.BoardItem

#### Raises

**PglOutOfBoardBoundException** – if row or column are out of bound.

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

# Return type

int

# Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

# classmethod load(data)

Load data and create a new ComplexWall out of it.

#### **Parameters**

**data** (dict) – Data to create a new complex wall item (usually generated by serialize())

### Returns

A new complex npc.

## Return type

~pygamelib.board\_items.ComplexWall

## property model

**notify**(modifier=None, attribute: <math>str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

This represent the capacity for a *BoardItem* to be overlapped by player or NPC.

#### Returns

False

#### Return type

bool

# property particle\_emitter

## pickable()

This represent the capacity for a *BoardItem* to be pick-up by player or NPC.

#### Returns

False

#### **Return type**

bool

Example:

```
if mywall.pickable():
    print('Whoaa this wall is really light... and small...')
else:
    print('Really? Trying to pick-up a wall?')
```

# position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
```

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

#### restorable()

This represent the capacity for an *Immovable* Movable item. A wall is not overlappable.

## Returns

False

# Return type

bool

## property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### Returns

The row coordinate

#### **Return type**

int

Example:

```
if item.row != item.pos[0]:
         print('Something extremely unlikely just happened...')
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize() \rightarrow dict
     Return a dictionary with all the attributes of this object.
         Returns
             A dictionary with all the attributes of this object.
         Return type
             dict
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
set_overlappable(value)
     Set the value of the overlappable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_overlappable(False)
set_pickable(value)
     Set the value of the pickable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_pickable(False)
```

```
set_restorable(value)
```

Set the value of the restorable property to value.

#### **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### **Returns**

The size.

#### Return type

list

Example:

# property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

**new\_sprite** (*Sprite*) – The sprite to set

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

# Example:

```
item.store_position(3,4)
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

## update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

# Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

# Return type

int

Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.14 Door

class pygamelib.board\_items.Door(\*\*kwargs)

Bases: GenericStructure

A Door is a *GenericStructure* that is not pickable, overlappable and restorable. It has a value of 0 and a size of 1 by default. It is an helper class that allows to focus on game design and mechanics instead of small building blocks.

## **Parameters**

- **model** (*str*) The model that will represent the door on the map
- value (int) The value of the door, it is useless in that case. The default value is 0.
- **inventory\_space** (*int*) The size of the door in the inventory. Unless you make the door pickable (I have no idea why you would do that...), this parameter is not used.
- **type** (*str*) The type of the door. It is often used as a type identifier for your game main loop. For example: unlocked\_door or locked\_door.
- pickable (Boolean) Is this door pickable by the player? Default value is False.
- overlappable (Boolean) Is this door overlappable by the player? Default value is True.
- restorable (Boolean) Is this door restorable after being overlapped? Default value is True.

**Note:** All the options from *GenericStructure* are also available to this constructor.

Example:

```
door1 = Door(model=graphics.Models.DOOR,type='locked_door')
```

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

L L Globs A	
init(**kwargs)	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
Store_Sereen_posteron(row, commit)	Store the serven position of the object.

# **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

# **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

# Returns

True or False depending on the success of the operation.

#### **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

False

# Return type

bool

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# Return type

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

## **Returns**

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

# **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

# **Return type**

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

# Returns

The heading of the item.

## **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

# Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

# property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

# Returns

The size of the item.

# Return type

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

# Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

# **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

A new item.

# Return type

~pygamelib.board\_items.BoardItem

## property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

# pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

# position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

# **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
```

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

# restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
   print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### Returns

The row coordinate

## **Return type**

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

# **Parameters**

**value** (*int*) – the screen column

# Return type

int

# property screen\_row: int

A property to get/set the screen row.

# **Parameters**

**value** (*int*) – the screen row

# Return type

int

## $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

A dictionary with all the attributes of this object.

# Return type

dict

# set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

# set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

# **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

# property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

# Returns

The size.

#### Return type

list

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

## Example:

```
item.store_position(3,4)
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

# property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

# Return type

int

# Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.15 GenericActionableStructure

class pygamelib.board\_items.GenericActionableStructure(\*\*kwargs)

Bases: GenericStructure, Actionable

A GenericActionableStructure is the combination of a *GenericStructure* and an *Actionable*. It is only a helper combination.

Please see the documentation for *GenericStructure* and *Actionable* for more information.

Important: There's a complete tutorial about Actionable items on the pygamelib wiki

\_\_init\_\_(\*\*kwargs)

Like the object class, this class constructor takes no parameter.

# **Methods**

init(**kwargs)	Like the object class, this class constructor takes no parameter.
activate()	•
activate()	This function is calling the action function with the action_parameters.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
collides_with(other[, projection_offset])	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their
debug_IIIIO()	current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
-	observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
· ·	wise.
pickable()	Returns True if the item is pickable, False otherwise.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this ob-
Serrarize()	ject.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
5 core_sereen_posteron(row, commit)	Store the sereon position of the object.

406 Chapter 3. Tutorials

# **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

## activate()

This function is calling the action function with the action\_parameters.

The *action* callback function should therefor have a signature like:

def my\_callback\_function(actionable, action\_parameters)

With actionable being the Actionable current reference to self.

Usually it's automatically called by move() when a Player or NPC (see board\_items)

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

# **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

#### Returns

False

# Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- projection\_offset (Vector2D) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# Return type

bool

#### Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

# Return type

int

# Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

# **Return type**

str

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### **Returns**

True or False depending on the success of the operation.

## Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

# Return type

float

# Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

# **Parameters**

- **subject** (*Pg1Base0bject*) The object that has changed.
- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

# Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

## Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

# property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

#### Returns

The size of the item.

# Return type

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

# Returns

The layer number

# Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

#### **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

#### Returns

A new item.

# Return type

~pygamelib.board\_items.BoardItem

## property model

**notify**(modifier = None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

## pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

```
position_as_vector()
```

Returns the current item position as a Vector2D

#### **Returns**

The position as a 2D vector

#### **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

# render\_to\_buffer(buffer, row, column, height, width)

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

# restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

# Returns

The row coordinate

# Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (int) – the screen column

# Return type

int

# property screen\_row: int

A property to get/set the screen row.

# **Parameters**

**value** (int) – the screen row

# **Return type**

int

# $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

# Returns

A dictionary with all the attributes of this object.

# Return type

dict

# set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

# **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

# set\_pickable(value)

Set the value of the pickable property to value.

## **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_restorable(False)
```

#### property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

# Return type

list

# Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
        f"{item.name} cannot be placed at {item.pos}."
    )
```

```
store_position(row: int, column: int, layer: int = 0)
```

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

## **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

# Example:

```
item.store_position(3,4)
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

#### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

# Return type

int

# Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.16 GenericStructureComplexComponent

class pygamelib.board\_items.GenericStructureComplexComponent(\*\*kwargs)

```
Bases: \textit{GenericStructure}, \textit{BoardItemComplexComponent}
```

A ComplexComponent specifically for generic structures.

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# **Methods**

i i calcid	THE A SECOND SECOND
init(**kwargs)	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
<pre>debug_info()</pre>	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns False.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

# **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an *Animation* for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

# **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

# Returns

True or False depending on the success of the operation.

#### **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

False

# Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

## **Returns**

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

# **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

# **Return type**

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

# Returns

The heading of the item.

## **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

# Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

# property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

# Returns

The size of the item.

# Return type

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

# Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

# **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

A new item.

# Return type

~pygamelib.board\_items.BoardItem

## property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

# pickable()

Returns False. A component is never pickable by itself (either the whole complex item is pickable or not, but not partially)

Example:

```
if item.item(4,5).pickable():
   print('The item is pickable')
```

# position\_as\_vector()

Returns the current item position as a Vector2D

## Returns

The position as a 2D vector

# Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
```

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

# restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### Returns

The row coordinate

## **Return type**

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

# **Parameters**

**value** (int) – the screen column

# Return type

int

# property screen\_row: int

A property to get/set the screen row.

# **Parameters**

**value** (*int*) – the screen row

# Return type

int

## $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

A dictionary with all the attributes of this object.

# Return type

dict

# set\_can\_move(value)

Set the value of the can\_move property to value.

## **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

# set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

# **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

# property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

# Returns

The size.

#### Return type

list

Example:

422 Chapter 3. Tutorials

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

#### Example:

```
item.store_position(3,4)
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

# property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

# Return type

int

# Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.4.17 GenericStructure

class pygamelib.board\_items.GenericStructure(value=0, \*\*kwargs)

Bases: Immovable

A GenericStructure is as the name suggest, a generic object to create all kind of structures.

It can be tweaked with all the properties of *BoardItem*, *Immovable* and it can be made pickable, overlappable or restorable or any combination of these.

If you need an action to be done when a Player and/or a NPC touch the structure please have a look at *pygamelib*. board\_items.GenericActionableStructure.

#### **Parameters**

- pickable (bool) Define if the structure can be picked-up by a Player or NPC.
- **overlappable** (*bool*) Define if the structure can be overlapped by a Player or NPC.
- **restorable** (*bool*) Define if the structure can be restored by the Board after a Player or NPC passed through. For example, you want a door or an activator structure (see Generic ActionableStructure for that) to remain on the board after it's been overlapped by a player. But you could also want to develop some kind of Space Invaders game were the protection block are overlappable but not restorable.
- **value** (*int | float*) The value of the structure. It can be used for scoring, resource spending, etc.

On top of these, this object takes all parameters of BoardItem and Immovable

**Important:** If you need a structure with a permission system please have a look at *GenericActionableStructure*. This class has a permission system for activation.

\_\_init\_\_(value=0, \*\*kwargs)

Like the object class, this class constructor takes no parameter.

# Methods

init([value])	Like the object class, this class constructor takes no
([value])	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
collides_with(other[, projection_offset])	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their
debug_IIIIO()	current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
	observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

# **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

# **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

# Returns

True or False depending on the success of the operation.

#### **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

False

# Return type

bool

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

## **Returns**

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

## debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

## Returns

The distance between this item and the other.

## **Return type**

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

# Returns

The heading of the item.

#### **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

### property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

# Returns

The size of the item.

## Return type

int

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

## **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

A new item.

# Return type

~pygamelib.board\_items.BoardItem

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

# pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

430 Chapter 3. Tutorials

```
render_to_buffer(buffer, row, column, height, width)
```

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

## property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### Returns

The row coordinate

#### **Return type**

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

## property screen\_column: int

A property to get/set the screen column.

## **Parameters**

**value** (*int*) – the screen column

# Return type

int

## property screen\_row: int

A property to get/set the screen row.

## **Parameters**

**value** (*int*) – the screen row

## Return type

int

#### $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

A dictionary with all the attributes of this object.

# Return type

dict

## set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

# **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

## Returns

The size.

#### Return type

list

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

#### Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

# property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# **Returns**

The width

## Return type

int

# Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.18 Immovable

class pygamelib.board\_items.Immovable(inventory\_space: int = None, \*\*kwargs)

Bases: BoardItem

This class derive <code>BoardItem</code> and describe an object that cannot move or be moved (like a wall). <code>can\_move()</code> cannot be configured and return False. The other properties can be configured. They have the same default values than <code>BoardItem</code>.

#### **Parameters**

**inventory\_space** (int) – The space the immovable item takes into an *Inventory* (in case the item is pickable). By default it is 0.

```
__init__(inventory_space: int = None, **kwargs)
```

Like the object class, this class constructor takes no parameter.

#### **Methods**

init([inventory_space])	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
<pre>set_overlappable(value)</pre>	Set the value of the overlappable property to value.
<pre>set_pickable(value)</pre>	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

434 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

## property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

## Returns

True or False depending on the success of the operation.

#### **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

False

# Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

#### Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### **Returns**

The column coordinate

## Return type

int

### Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

#### debug\_info()

Return a string with the list of the attributes and their current value.

## Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

# distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

### **Return type**

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

## Returns

The heading of the item.

#### **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

### Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

## Returns

The size of the item.

## Return type

int

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

## **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

A new item.

# Return type

~pygamelib.board\_items.BoardItem

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

# pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
     Render the board item into a display buffer (not a screen buffer).
     This method is automatically called by pygamelib.engine.Screen.render().
         Parameters
             • buffer (numpy.array) – A screen buffer to render the item into.
             • row (int) – The row to render in.
             • column (int) – The column to render in.
             • height (int) – The total height of the display buffer.
             • width (int) – The total width of the display buffer.
restorable()
     Returns True if the item is restorable, False otherwise.
     Example:
     if board.item(4,5).restorable():
         print('The item is restorable')
property row
     Convenience method to get the current stored row of the item.
     This is absolutely equivalent to access to item.pos[0].
         Returns
             The row coordinate
         Return type
             int
     Example:
     if item.row != item.pos[0]:
         print('Something extremely unlikely just happened...')
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
```

Return a dictionary with all the attributes of this object.

**value** (int) – the screen row

**Parameters** 

Return type int

**serialize()**  $\rightarrow$  dict

A dictionary with all the attributes of this object.

# Return type

dict

## set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

Returns

The size.

#### Return type

list

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

### Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

## Return type

int

# Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# **3.4.19 Movable**

Bases: BoardItem

A class representing BoardItem capable of movements.

Movable subclasses BoardItem.

#### **Parameters**

- **step** (*int*) the amount of cell a movable can cross in one turn. Default value: 1.
- **step\_vertical** (*int*) the amount of cell a movable can vertically cross in one turn. Default value: step value.
- **step\_horizontal** (*int*) the amount of cell a movable can horizontally cross in one turn. Default value: step value.
- movement\_speed (int/float) The time (in seconds) between 2 movements of a Movable. It is used by all the Game's actuation methods to enforce move speed of NPC and projectiles.

The movement\_speed parameter is only used when the Game is configured with MODE\_RT. Additionally the dtmove property is used to accumulate time between frames. It is entirely managed by the Game object and most of the time you shouldn't mess up with it. Unless you want to manage movements by yourself. If so, have fun! That's the point of the pygamelib to let you do whatever you like.

This class derive BoardItem and describe an object that can move or be moved (like a player or NPC). Thus this class implements BoardItem.can\_move(). However it does not implement BoardItem.pickable() or BoardItem.overlappable()

\_\_init\_\_(step: int = None, step\_vertical: int = None, step\_horizontal: int = None, movement\_speed: float = None, \*\*kwargs)

Like the object class, this class constructor takes no parameter.

# Methods

init([step, step_vertical,])	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
has_inventory()	This is a virtual method that must be implemented in deriving class.
load(data)	Load data and create a new Movable out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Serialize the Immovable object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

444 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### $can\_move() \rightarrow bool$

Movable implements can\_move().

True

# Return type

Boolean

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### **Returns**

The column coordinate

## Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

#### debug\_info()

Return a string with the list of the attributes and their current value.

## Return type

str

# detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### **Returns**

The distance between this item and the other.

### Return type

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

#### property dtmove

#### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

# **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## $has\_inventory() \rightarrow bool$

This is a virtual method that must be implemented in deriving class. This method has to return True or False. This represent the capacity for a Movable to have an inventory.

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

#### **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

## Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

### Returns

The size of the item.

## Return type

int

#### property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

## classmethod load(data)

Load data and create a new Movable out of it.

#### **Parameters**

**data** (dict) – Data to create a new movable item (usually generated by serialize())

#### Returns

A new complex item.

#### Return type

~pygamelib.board\_items.Movable

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

## property particle\_emitter

## pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

### position\_as\_vector()

Returns the current item position as a Vector2D

## Returns

The position as a 2D vector

#### Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

render\_to\_buffer(buffer, row, column, height, width)

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

## Returns

The row coordinate

# Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

### **Parameters**

value(int) – the screen column

# Return type

int

### property screen\_row: int

A property to get/set the screen row.

## **Parameters**

**value** (int) – the screen row

#### **Return type**

int

#### **serialize()** $\rightarrow$ dict

Serialize the Immovable object.

This returns a dictionary that contains all the key/value pairs that makes up the object.

# set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### **Returns**

The size.

# Return type

list

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## Returns

The width

## Return type

int

Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.20 NPC

class pygamelib.board\_items.NPC(actuator=None, \*\*kwargs)

Bases: Character

A class that represent a non playable character controlled by the computer. For the NPC to be successfully managed by the Game, you need to set an actuator.

None of the parameters are mandatory, however it is advised to make good use of some of them (like type or name) for game design purpose.

In addition to its own member variables, this class inherits all members from:

- pygamelib.board\_items.Character
- pygamelib.board\_items.Movable
- pygamelib.board\_items.BoardItem

This class sets a couple of variables to default values:

- max hp: 10
- hp: 10
- remaining\_lives: 1
- attack\_power: 5
- movement\_speed: 0.25 (one movement every 0.25 second). Only useful if the game mode is set to MODE\_RT.

## **Parameters**

**actuator** (pygamelib.actuators.Actuator) – An actuator, it can be any class but it need to implement pygamelib.actuators.Actuator.

## Example:

```
mynpc = NPC(name='Idiot McStupid', type='dumb_enemy')
mynpc.step = 1
mynpc.actuator = RandomActuator()
```

```
__init__(actuator=None, **kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init([actuator])	Like the object class, this class constructor takes no
a + + a a la ( a la a a mana m)	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
has_inventory()	Define if the NPC has an inventory.
load(data)	Load data and create a new NPC out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
**	wise.
pickable()	Define if the NPC is pickable.
position_as_vector()	Returns the current item position as a Vector2D
render_to_buffer(buffer, row, column,)	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Serialize the NPC object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

454 Chapter 3. Tutorials

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### $can\_move() \rightarrow bool$

Movable implements can\_move().

True

# Return type

Boolean

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

#### Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### **Returns**

The column coordinate

## Return type

int

### Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

#### debug\_info()

Return a string with the list of the attributes and their current value.

## Return type

str

# detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

True or False depending on the success of the operation.

# **Return type**

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

### Return type

float

Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## property dtmove

#### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

# **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## has\_inventory()

Define if the NPC has an inventory.

This method returns false because the game engine doesn't manage NPC inventory yet but it could be in the future. It's a good habit to check the value returned by this function.

#### **Returns**

False

#### Return type

Boolean

Example:

```
if mynpc.has_inventory():
    print("Cool: we can pickpocket that NPC!")
else:
    print("No pickpocketing XP for us today :(")
```

## property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

# **Return type**

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

#### Returns

The size of the item.

# Return type

int

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### Returns

The layer number

## **Return type**

int

## Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new NPC out of it.

#### **Parameters**

**data** (dict) – Data to create a new npc (usually generated by serialize())

#### Returns

A new npc.

## Return type

~pygamelib.board\_items.NPC

## property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

## property particle\_emitter

## pickable()

Define if the NPC is pickable.

Obviously this method always return False.

#### Returns

False

## Return type

Boolean

Example:

```
if mynpc.pickable():
    Utils.warn("Something is fishy, that NPC is pickable"
        "but is not a Pokemon...")
```

# position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

#### Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
```

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

#### restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

```
The row coordinate
         Return type
             int
     Example:
     if item.row != item.pos[0]:
          print('Something extremely unlikely just happened...')
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize() \rightarrow dict
     Serialize the NPC object.
     This returns a dictionary that contains all the key/value pairs that makes up the object.
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
set_overlappable(value)
     Set the value of the overlappable property to value.
         Parameters
             value (boo1) – The value to set.
     Example:
     item.set_overlappable(False)
set_pickable(value)
     Set the value of the pickable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
```

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### **Returns**

The size.

## Return type

list

Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
        f"{item.name} cannot be placed at {item.pos}."
    )
```

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

## **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- layer the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

# **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

#### Returns

The width

# Return type

int

Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# **3.4.21 Player**

class pygamelib.board\_items.Player(inventory=None, \*\*kwargs)

Bases: Character

A class that represent a player controlled by a human.

This can take all parameter from *Character*, *Movable* and obviously *BoardItem*.

It is a specific board item as the whole Game class assumes only one player. Aside from the wrapper functions (like Game.move\_player for example), there is no reel limitations to use more than one player.

The player also has a couple of attributes that are added for your convenience. You are free to use them or not. They are (name and default value):

- max\_hp: 100
- hp: 100
- remaining\_lives: 3
- attack\_power: 10
- movement\_speed: 0.1 (one movement every 0.1 second). Only useful if the game mode is set to MODE\_RT.
- inventory: A *Inventory* object. If none is provided, one is created automatically.

A player can be animated by providing a *Animation* object to its *animation* attribute.

Like all other board items, you can specify a *sprixel* attribute that will be the representation of the player on the board.

Example:

```
player = Player(
   name="Player",
   # A sprixel with "@" as the model, no background color, a cyan foreground
   # color and we set the background to be transparent.
   sprixel=core.Sprixel("@", None, core.Color(0, 255, 255), True),
```

3.4. board items 463

(continues on next page)

(continued from previous page)

```
max_hp=200,
```

```
__init__(inventory=None, **kwargs)
```

Like the object class, this class constructor takes no parameter.

# **Methods**

init([inventory])	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
has_inventory()	This method returns True (a player has an inventory).
load(data)	Load data and create a new Character out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
	wise.
pickable()	This method returns False (a player is obviously not pickable).
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Serialize the Character object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
store_position(row, column[, layer])	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

464 Chapter 3. Tutorials

# **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### $can\_move() \rightarrow bool$

Movable implements can\_move().

### Returns

True

# Return type

Boolean

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### **Returns**

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

### debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

# detach(observer)

466

Detach an observer from this instance. If observer is not in the list this returns False.

### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

### distance\_to(other)

Calculates the distance with an item.

### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

### **Returns**

The distance between this item and the other.

## Return type

float

## Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## property dtmove

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

# **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# has\_inventory()

This method returns True (a player has an inventory).

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

# property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

## Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

### Returns

The size of the item.

## Return type

int

### property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

## classmethod load(data)

Load data and create a new Character out of it.

### **Parameters**

**data** (dict) – Data to create a new character item (usually generated by serialize())

### Returns

A new character item.

## Return type

~pygamelib.board\_items.Character

# property model

```
notify(modifier = None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### overlappable()

Returns True if the item is overlappable, False otherwise.

### Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

# property particle\_emitter

## pickable()

This method returns False (a player is obviously not pickable).

# position\_as\_vector()

Returns the current item position as a Vector2D

## Returns

The position as a 2D vector

## Return type

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
     next_position = item.position_as_vector() + gravity.unit()
render_to_buffer(buffer, row, column, height, width)
     Render the board item into a display buffer (not a screen buffer).
     This method is automatically called by pygamelib.engine.Screen.render().
         Parameters
             • buffer (numpy.array) – A screen buffer to render the item into.
             • row (int) - The row to render in.
             • column (int) – The column to render in.
             • height (int) – The total height of the display buffer.
             • width (int) – The total width of the display buffer.
restorable()
     Returns True if the item is restorable, False otherwise.
     Example:
     if board.item(4,5).restorable():
         print('The item is restorable')
property row
     Convenience method to get the current stored row of the item.
     This is absolutely equivalent to access to item.pos[0].
         Returns
             The row coordinate
         Return type
             int
     Example:
     if item.row != item.pos[0]:
         print('Something extremely unlikely just happened...')
property screen_column: int
     A property to get/set the screen column.
         Parameters
            value (int) – the screen column
         Return type
             int
property screen_row: int
```

A property to get/set the screen row.

value (int) – the screen row

**Parameters** 

Return type int

```
serialize() \rightarrow dict
```

Serialize the Character object.

This returns a dictionary that contains all the key/value pairs that makes up the object.

## set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_can_move(False)
```

## set\_overlappable(value)

Set the value of the overlappable property to value.

### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

# set\_pickable(value)

Set the value of the pickable property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

## **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

# Return type

list

Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

# property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

## Return type

int

Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.22 Projectile

Bases: Movable

A class representing a projectile type board item. That class can be sub-classed to represent all your needs (fireballs, blasters shots, etc.).

That class support the 2 types of representations: model and animations. The animation cases are slightly more evolved than the regular item.animation. It does use the item.animation but with more finesse as a projectile can travel in many directions. So it also keeps track of models and animation per travel direction.

You probably want to subclass Projectile. It is totally ok to use it as it, but it is easier to create a subclass that contains all your Projectile information and let the game engine deal with orientation, range keeping, etc. Please see examples/07\_projectiles.py for a good old fireball example.

By default, Projectile travels in straight line in one direction. This behavior can be overwritten by setting a specific actuator (a projectile is a *Movable* so you can use my\_projectile.actuator).

The general way to use it is as follow:

- Create a factory object with your static content (usually the static models, default direction and hit callback)
- Add the direction related models and/or animation (keep in mind that animation takes precedence over static models)
- deep copy that object when needed and add it to the projectiles stack of the game object.
- use Game.actuate\_projectiles(level) to let the Game engine do the heavy lifting.

The Projectile constructor takes the following parameters:

### **Parameters**

- direction (Direction) A direction from the constants module
- range (int) The maximum range of the projectile in number of cells that can be crossed. When range is attained the hit\_callback is called with a BoardItemVoid as a collision object.
- **step** (*int*) the amount of cells a projectile can cross in one turn
- model(str) the default model of the projectile.
- movement\_animation (Animation) the default animation of a projectile. If a projectile is sent in a direction that has no explicit and specific animation, then movement\_animation is used if defined.
- **hit\_animation** (*Animation*) the animation used when the projectile collide with something.
- **hit\_model** (*str*) the model used when the projectile collide with something.
- hit\_callback (function) A reference to a function that will be called upon collision. The hit\_callback is receiving the object it collides with as first parameter.
- **is\_aoe** (*bool*) Is this an 'area of effect' type of projectile? Meaning, is it doing something to everything around (mass heal, exploding rocket, fireball, etc.)? If yes, you must set that parameter to True and set the aoe\_radius. If not, the Game object will only send the colliding object in front of the projectile.

- **aoe\_radius** (*int*) the radius of the projectile area of effect. This will force the Game object to send a list of all objects in that radius.
- **callback\_parameters** (*list*) A list of parameters to pass to hit\_callback.
- movement\_speed (int | float) The movement speed of the projectile
- **collision\_exclusions** (*list*) A list of **TYPES** of objects that should not collides with that projectile. It is usually a good idea to put the projectile type in the exclusion list. This prevent the projectile to collide with other instances of itself. Adding the projectile's emitter is also a valid idea.
- parent The parent object (usually a Board object or some sort of BoardItem).

**Important:** The effects of a Projectile are determined by the callback. No callback == no effect!

# Example:

Like the object class, this class constructor takes no parameter.

# **Methods**

init([name, direction, step, range,])	Like the object class, this class constructor takes no parameter.
add_directional_animation(direction, anima-	Add an animation for a specific direction.
tion)	
<pre>add_directional_model(direction, model)</pre>	Add an model for a specific direction.
attach(observer)	Attach an observer to this instance.
can_move()	Movable implements can_move().
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
directional_animation(direction)	Return the animation for a specific direction.
<pre>directional_model(direction)</pre>	Return the model for a specific direction.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.

continues on next page

Table 7 – continued from previous page

<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
has_inventory()	Projectile cannot have inventory by default.
hit(objects)	A method that is called when the projectile hit something.
load(data)	Load data and create a new Movable out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Projectile are overlappable by default.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>remove_directional_animation(direction)</pre>	Remove an animation for a specific direction.
<pre>remove_directional_model(direction)</pre>	Remove the model for a specific direction.
render_to_buffer(buffer, row, column,)	Render the board item into a display buffer (not a screen buffer).
restorable()	We assume that by default, Projectiles are restorable.
serialize()	Serialize the Immovable object.
<pre>set_can_move(value)</pre>	Set the value of the can_move property to value.
<pre>set_direction(direction)</pre>	Set the direction of a projectile
<pre>set_overlappable(value)</pre>	Set the value of the overlappable property to value.
<pre>set_pickable(value)</pre>	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

# **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column
	of the item.
direction	The direction of the projectile.
dtmove	
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem
	takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer
	number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of
	the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as
	a 2 dimensions list.
width	Convenience method to get the width of the item.

# add\_directional\_animation(direction, animation)

Add an animation for a specific direction.

### **Parameters**

- **direction** (*int*) A direction from the constants module.
- **animation** (*Animation*) The animation for the direction

Example:

```
fireball.add_directional_animation(Direction.UP, Direction.UP, animation)
```

### add\_directional\_model(direction, model)

Add an model for a specific direction.

### **Parameters**

- **direction** (*int*) A direction from the constants module.
- model (str) The model for the direction

Example:

```
fireball add_directional_animation(Direction UP, upward_animation)
```

## property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

### **Returns**

True or False depending on the success of the operation.

# Return type

bool

Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## $can\_move() \rightarrow bool$

Movable implements can\_move().

### Returns

True

## Return type

Boolean

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- projection\_offset (Vector2D) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# **Return type**

bool

### Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### Returns

The column coordinate

# Return type

int

# Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

### Return type

str

# detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBase0bject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

## Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## property direction

The direction of the projectile.

Updating this property also updates the Unidirectional Actuator's direction.

### **Parameters**

**value** (int | *Vector2D*) – some param

**Warning:** If your projectile uses directional model and/or animation you should use  $set\_direction()$  to set the projectile direction.

Example:

```
bullet.direction = Vector2D(0, 1)
```

## directional\_animation(direction)

Return the animation for a specific direction.

#### **Parameters**

**direction** (*int*) – A direction from the constants module.

# **Return type**

Animation

Example:

```
# No more animation for the UP direction
fireball.directional_animation(Direction.UP)
```

## directional\_model(direction)

Return the model for a specific direction.

### **Parameters**

**direction** (*int*) – A direction from the constants module.

### Return type

str

Example:

```
fireball.directional_model(Direction.UP)
```

# display()

Print the model WITHOUT carriage return.

## distance\_to(other)

Calculates the distance with an item.

## **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

### Returns

The distance between this item and the other.

# Return type

float

# Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

## property dtmove

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

### has\_inventory()

Projectile cannot have inventory by default.

### Returns

False

## Return type

Boolean

### property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

# Return type

Vector2D

### Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

### Return type

int

### Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

# hit(objects)

A method that is called when the projectile hit something.

That method is automatically called by the Game object when the Projectile collide with another object or is at the end of its range.

Here are the call cases covered by the Game object:

- range is reached without collision and projectile IS NOT an AoE type: hit() is called with a single BoardItemVoid in the objects list.
- range is reached without collision and projectile IS an AoE type: hit() is called with the list of all objects within aoe\_radius (including structures).
- projectile collide with something and IS NOT an AoE type: hit() is called with the single colliding object in the objects list.
- projectile collide with something and IS an AoE type: hit() is called with the list of all objects within aoe\_radius (including structures).

In turn, that method calls the hit\_callback with the following parameters (in that order):

- 1. the projectile object
- 2. the list of colliding objects (that may contain only one object)
- 3. the callback parameters (from the constructor callback\_parameters)

## **Parameters**

**objects** – A list of objects hit by or around the projectile.

### Example:

```
my_projectile.hit([npc1])
```

### property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

### Returns

The size of the item.

### **Return type**

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

### Return type

int

# Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

### classmethod load(data)

Load data and create a new Movable out of it.

#### Parameters

**data** (dict) – Data to create a new movable item (usually generated by serialize())

#### Returns

A new complex item.

### **Return type**

~pygamelib.board\_items.Movable

## property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Projectile are overlappable by default.

#### Returns

True

## Return type

Boolean

## property particle\_emitter

## pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

# position\_as\_vector()

Returns the current item position as a Vector2D

### Returns

The position as a 2D vector

## **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

# remove\_directional\_animation(direction)

Remove an animation for a specific direction.

#### **Parameters**

**direction** (int) – A direction from the constants module.

Example:

```
# No more animation for the UP direction
fireball.remove_directional_animation(Direction.UP)
```

# remove\_directional\_model(direction)

Remove the model for a specific direction.

### **Parameters**

**direction** (*int*) – A direction from the constants module.

Example:

```
fireball.directional_model(Direction.UP)
```

# render\_to\_buffer(buffer, row, column, height, width)

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by *pygamelib.engine.Screen.render()*.

### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

```
restorable()
     We assume that by default, Projectiles are restorable.
         Returns
             True
         Return type
             bool
property row
     Convenience method to get the current stored row of the item.
     This is absolutely equivalent to access to item.pos[0].
         Returns
             The row coordinate
         Return type
             int
     Example:
     if item.row != item.pos[0]:
          print('Something extremely unlikely just happened...')
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize() \rightarrow dict
     Serialize the Immovable object.
     This returns a dictionary that contains all the key/value pairs that makes up the object.
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
```

### set\_direction(direction)

Set the direction of a projectile

This method will set a UnidirectionalActuator with the direction. It will also take care of updating the model and animation for the given direction if they are specified.

### **Parameters**

**direction** (*int*) – A direction from the constants module.

Example:

```
fireball.set_direction(Direction.UP)
```

## set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_overlappable(False)
```

## set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

## set\_restorable(value)

Set the value of the restorable property to value.

## **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_restorable(False)
```

# property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

### Returns

The size.

## Return type

list

Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
```

(continues on next page)

(continued from previous page)

```
f"{item.name} cannot be placed at {item.pos}."
)
```

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (int) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

## Example:

```
item.store_position(3,4)
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row(int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

# property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## Returns

The width

# Return type

int

## Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.23 TextItem

```
class pygamelib.board_items.TextItem(text=None, **kwargs)
```

Bases: BoardComplexItem

New in version 1.2.0.

The text item is a board item that can contains text. The text can then be manipulated and placed on a Board.

It is overall a *BoardComplexItem* (so it takes all the parameters of that class). The big difference is that the first parameter is the text you want to display.

The text parameter can be either a regular string or a *Text* object (in case you want formatting and colors).

### **Parameters**

**text** (str | *Text*) – The text you want to display.

Example:

```
city_name = TextItem('Super City')
fancy_city_name = TextItem(text=base.Text('Super City', base.Fore.GREEN,
    base.Back.BLACK,
    base.Style.BRIGHT
))
my_board.place_item(city_name, 0, 0)
my_board.place_item(fancy_city_name, 1, 0)
```

```
__init__(text=None, **kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init([text])	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Returns True if the item can move, False otherwise.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>item(row, column)</pre>	Return the item component at the row, column position if it is within the complex item's boundaries.
load(data)	Load data and create a new TextItem out of it.
notify([modifier, attribute, value])	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False other-
overlappable()	wise.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the complex board item from the display buffer to the frame buffer.
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
update_sprite()	Update the complex item with the current sprite.

## **Attributes**

animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	A property to get and set the size that the BoardItem takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
sprite	A property to easily access and update a complex item's sprite.
text	The text within the item.
width	Convenience method to get the width of the item.

# property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

488 Chapter 3. Tutorials

### can\_move()

Returns True if the item can move, False otherwise.

Example:

```
if board.item(4,5).can_move():
    print('The item can move')
```

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

## **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

# Return type

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### Returns

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

# debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

# Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## display()

Print the model WITHOUT carriage return.

### distance\_to(other)

Calculates the distance with an item.

### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

## **Return type**

float

# Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

# property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

#### Returns

The heading of the item.

### Return type

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

## property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

### Returns

The height

## Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

## property inventory\_space

A property to get and set the size that the BoardItem takes in the *Inventory*.

### Returns

The size of the item.

## Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

# Return type

~pygamelib.board\_items.BoardItem

#### Raises

**PglOutOfBoardBoundException** – if row or column are out of bound.

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

# Return type

int

# Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

# classmethod load(data)

Load data and create a new TextItem out of it.

#### **Parameters**

**data** (dict) – Data to create a new text item (usually generated by serialize())

### Returns

A new complex npc.

## Return type

~pygamelib.board\_items.TextItem

## property model

**notify**(modifier = None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
   print('The item is overlappable')
```

# property particle\_emitter

# pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
   print('The item is pickable')
```

## position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

#### **Return type**

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

### render\_to\_buffer(buffer, row, column, height, width)

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
   print('The item is restorable')
```

### property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

# Returns

The row coordinate

### Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

**Parameters** 

Example:

**value** (int) – the screen column

```
Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize() \rightarrow dict
     Return a dictionary with all the attributes of this object.
         Returns
             A dictionary with all the attributes of this object.
         Return type
             dict
set_can_move(value)
     Set the value of the can_move property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_can_move(False)
set_overlappable(value)
     Set the value of the overlappable property to value.
         Parameters
             value (bool) – The value to set.
     Example:
     item.set_overlappable(False)
set_pickable(value)
     Set the value of the pickable property to value.
         Parameters
             value (boo1) – The value to set.
     Example:
     item.set_pickable(False)
set_restorable(value)
     Set the value of the restorable property to value.
         Parameters
             value (bool) – The value to set.
```

494 Chapter 3. Tutorials

```
item.set_restorable(False)
```

## property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

## Return type

list

## Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
          f"{item.name} cannot be placed at {item.pos}."
    )
```

### property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

**new\_sprite** (Sprite) – The sprite to set

Example:

## $store_position(row: int, column: int, layer: int = 0)$

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.

• **layer** – the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

### property text

The text within the item.

TextItem.text can be set to either a string or a Text object.

It will always return a Text object.

Internally it translate the text to a *Sprite* to display it correctly on a *Board*. If print()-ed it will do so like the *Text* object.

# update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

### Returns

The width

### Return type

int

# Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.24 Tile

class pygamelib.board\_items.Tile(\*\*kwargs)

 $Bases: {\it BoardComplexItem}, {\it GenericStructure}$ 

New in version 1.2.0.

A Tile is a standard *BoardComplexItem* configured by default to:

- be overlappable
- be restorable
- be not pickable
- be immovable.

Aside from the movable attributes (it inherit from GenericStructure so it's an Immovable object), everything else is configurable.

It is particularly useful to display a Sprite on the background or to create terrain.

## Example:

```
grass_sprite = Sprite.load_from_ansi_file('textures/grass.ans')
for pos in grass_positions:
   outdoor_level.place_item( Tile(sprite=grass_sprite), pos[0], pos[1] )
```

```
__init__(**kwargs)
```

# **Parameters**

- **overlappable** (*bool*) Defines if the Tile can be overlapped.
- **restorable** (*bool*) Defines is the Tile should be restored after being overlapped.
- pickable (bool) Defines if the Tile can be picked up by the Player or NPC.

Please see BoardComplexItem for additional parameters.

# Methods

init(**kwargs)	
<b>,</b> ,	param overlappable
	Defines if the Tile can be overlapped.
attach(observer)	Attach an observer to this instance.
can_move()	A Tile cannot move.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>item(row, column)</pre>	Return the item component at the row, column position if it is within the complex item's boundaries.
load(data)	Load data and create a new Tile out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	Returns True if the item is overlappable, False otherwise.
pickable()	Returns True if the item is pickable, False otherwise.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the complex board item from the display buffer to the frame buffer.
restorable()	Returns True if the item is restorable, False otherwise.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
update_sprite()	Update the complex item with the current sprite.

498 Chapter 3. Tutorials

# **Attributes**

	A company of the second and the design of the Constitutions
animation	A property to get and set an <i>Animation</i> for this item.
column	Convenience method to get the current stored column
	of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of
	the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as
	a 2 dimensions list.
sprite	A property to easily access and update a complex
	item's sprite.
width	Convenience method to get the width of the item.

## property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### can\_move()

A Tile cannot move.

#### Returns

False

# Return type

bool

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

## **Return type**

bool

### Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

# property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

### **Returns**

The column coordinate

# Return type

int

# Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

### debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

### Return type

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

### property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

### property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

### Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
   print('The item is too big for the board.')
```

### property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

### Returns

The size of the item.

### Return type

int

item(row, column)

Return the item component at the row, column position if it is within the complex item's boundaries.

# Return type

 ${\it \sim} pygame lib.board\_items. Board Item$ 

#### **Raises**

**Pg10ut0fBoardBoundException** – if row or column are out of bound.

## property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

#### **Returns**

The layer number

### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

# classmethod load(data)

Load data and create a new Tile out of it.

#### **Parameters**

**data** (dict) – Data to create a new tile (usually generated by serialize())

#### Returns

A new complex npc.

#### Return type

~pygamelib.board\_items.Tile

# property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# overlappable()

Returns True if the item is overlappable, False otherwise.

Example:

```
if board.item(4,5).overlappable():
    print('The item is overlappable')
```

## property particle\_emitter

### pickable()

Returns True if the item is pickable, False otherwise.

Example:

```
if board.item(4,5).pickable():
    print('The item is pickable')
```

### position\_as\_vector()

Returns the current item position as a Vector2D

### Returns

The position as a 2D vector

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

render\_to\_buffer(buffer, row, column, height, width)

Render the complex board item from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

### restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if board.item(4,5).restorable():
    print('The item is restorable')
```

# property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

### Returns

The row coordinate

# Return type

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

# property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (*int*) – the screen column

# Return type

int

### property screen\_row: int

A property to get/set the screen row.

### **Parameters**

value (int) – the screen row

int

### $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

#### Returns

A dictionary with all the attributes of this object.

### **Return type**

dict

#### set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

### set\_pickable(value)

Set the value of the pickable property to value.

### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

### property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

### Return type

list

#### Example:

#### property sprite

A property to easily access and update a complex item's sprite.

#### **Parameters**

**new\_sprite** (*Sprite*) – The sprite to set

Example:

```
npc1 = board_items.ComplexNpc(
                                sprite=npc_sprite_collection['npc1_idle']
# to access the sprite:
if npc1.sprite.width * npc1.sprite.height > CONSTANT_BIG_GUY:
   game.screen.place(
        base.Text(
            'Big boi detected!!!',
            core.Color(255,0,0),
            style=TextStyle.BOLD,
       ),
       notifications.row,
       notifications.column,
   )
# And to set it:
if game.player in game.neighbors(3, npc1):
   npc1.sprite = npc_sprite_collection['npc1_fight']
```

 $store\_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- **row** (*int*) the row of the item in the *Board*.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

Example:

```
item.store_position(3,4)
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

**Parameters** 

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

### update\_sprite()

Update the complex item with the current sprite.

**Note:** This method use to need to be called every time the sprite was changed. Starting with version 1.3.0, it is no longer a requirement as BoardComplexItem.sprite was turned into a property that takes care of calling update\_sprite().

# Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    # This is not only no longer required but also wasteful as
    # update_sprite() is called twice here.
    item.sprite = s
    item.update_sprite()
    board.move(item, Direction.RIGHT, 1)
    time.sleep(0.2)
```

#### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

## Returns

The width

# Return type

int

#### Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.25 Treasure

class pygamelib.board\_items.Treasure(value=10, \*\*kwargs)

Bases: Immovable

A Treasure is an *Immovable* that is pickable and with a non zero value. It is an helper class that allows to focus on game design and mechanics instead of small building blocks.

### **Parameters**

- **model** (str) The model that will represent the treasure on the map
- value (int) The value of the treasure, it is usually used to calculate the score.

• **inventory\_space** (*int*) – The space occupied by the treasure. It is used by *Inventory* as a measure of space. If the treasure's size exceed the Inventory size (or the cumulated size of all items + the treasure exceed the inventory max\_size()) the Inventory will refuse to add the treasure.

**Note:** All the options from *Immovable* are also available to this constructor.

# Example:

```
money_bag = Treasure(
    model=graphics.Models.MONEY_BAG,value=100,inventory_space=2
)
print(f"This is a money bag {money_bag}")
player.inventory.add_item(money_bag)
print(f"The inventory value is {player.inventory.value()} and is at
    {player.inventory.size()}/{player.inventory.max_size}")
```

```
__init__(value=10, **kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init([value])	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
<pre>debug_info()</pre>	Return a string with the list of the attributes and their current value.
detach(observer)	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	This represent the capacity for a Treasure to be over- lapped by player or NPC.
pickable()	This represent the capacity for a Treasure to be picked-up by player or NPC.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	This represent the capacity for a Treasure to be restored after being overlapped.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

### **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

### property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

#### **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

#### Returns

False

# Return type

bool

```
collides_with(other, projection offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

### **Return type**

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### **Returns**

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

#### debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

# **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

### Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

#### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

### Return type

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

### property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

### property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

### Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

### property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

### Returns

The size of the item.

# Return type

int

# property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

### **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

#### Returns

A new item.

### **Return type**

~pygamelib.board\_items.BoardItem

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### overlappable()

This represent the capacity for a Treasure to be overlapped by player or NPC.

A treasure is not overlappable.

# Returns

False

# Return type

bool

### property particle\_emitter

### pickable()

This represent the capacity for a Treasure to be picked-up by player or NPC.

A treasure is obviously pickable by the player and potentially NPCs. *Board* puts the Treasure in the *Inventory* if the picker implements has\_inventory()

#### Returns

True

### Return type

bool

# position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

# Return type

Vector2D

# Example:

```
gravity = Vector2D(9.81, 0)
     next_position = item.position_as_vector() + gravity.unit()
render_to_buffer(buffer, row, column, height, width)
     Render the board item into a display buffer (not a screen buffer).
     This method is automatically called by pygamelib.engine.Screen.render().
         Parameters
             • buffer (numpy.array) – A screen buffer to render the item into.
             • row (int) - The row to render in.
             • column (int) – The column to render in.
             • height (int) – The total height of the display buffer.
             • width (int) – The total width of the display buffer.
restorable()
     This represent the capacity for a Treasure to be restored after being overlapped.
     A treasure is not overlappable, therefor is not restorable.
         Returns
             False
         Return type
             bool
property row
     Convenience method to get the current stored row of the item.
     This is absolutely equivalent to access to item.pos[0].
         Returns
             The row coordinate
         Return type
             int
     Example:
     if item.row != item.pos[0]:
         print('Something extremely unlikely just happened...')
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value(int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
```

3.4. board items 515

**Parameters** 

**value** (int) – the screen row

int

### $serialize() \rightarrow dict$

Return a dictionary with all the attributes of this object.

#### Returns

A dictionary with all the attributes of this object.

### **Return type**

dict

#### set\_can\_move(value)

Set the value of the can\_move property to value.

#### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

# set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_overlappable(False)
```

### set\_pickable(value)

Set the value of the pickable property to value.

### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_restorable(False)
```

### property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

#### Returns

The size.

### Return type

list

### Example:

```
# This is a silly example because the Board object does not allow
# that use case.
if item.column + item.size[0] >= board.width:
    Game.instance().screen.display_line(
        f"{item.name} cannot be placed at {item.pos}."
    )
```

**store\_position**(row: int, column: int, layer: int = 0)

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

### **Parameters**

- row (int) the row of the item in the Board.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

### Example:

```
item.store_position(3,4)
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

# Example:

```
an_object.store_screen_coordinate(3,8)
```

### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

# Returns

The width

# Return type

int

#### Example:

```
if item.width > board.width:
   print('The item is too big for the board.')
```

# 3.4.26 Wall

class pygamelib.board\_items.Wall(\*\*kwargs)

Bases: Immovable

A Wall is a specialized *Immovable* object that as unmodifiable characteristics:

- It is not pickable (and cannot be).
- It is not overlappable (and cannot be).
- It is not restorable (and cannot be).

As such it's an object that cannot be moved, cannot be picked up or modified by Player or NPC and block their ways. It is therefor advised to create one per board and reuse it in many places.

#### **Parameters**

- **model** (*str*) The representation of the Wall on the Board.
- name (str) The name of the Wall.
- **size** (*int*) The size of the Wall. This parameter will probably be deprecated as size is only used for pickable objects.

```
__init__(**kwargs)
```

Like the object class, this class constructor takes no parameter.

# Methods

init(**kwargs)	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
can_move()	Return the capability of moving of an item.
<pre>collides_with(other[, projection_offset])</pre>	Tells if this item collides with another item.
<pre>debug_info()</pre>	Return a string with the list of the attributes and their current value.
<pre>detach(observer)</pre>	Detach an observer from this instance.
display()	Print the model WITHOUT carriage return.
<pre>distance_to(other)</pre>	Calculates the distance with an item.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load data and create a new BoardItem out of it.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
overlappable()	This represent the capacity for a <i>BoardItem</i> to be overlapped by player or NPC.
pickable()	This represent the capacity for a <i>BoardItem</i> to be pick-up by player or NPC.
<pre>position_as_vector()</pre>	Returns the current item position as a Vector2D
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board item into a display buffer (not a screen buffer).
restorable()	This represent the capacity for an <i>Immovable</i> Movable item.
serialize()	Return a dictionary with all the attributes of this object.
set_can_move(value)	Set the value of the can_move property to value.
set_overlappable(value)	Set the value of the overlappable property to value.
set_pickable(value)	Set the value of the pickable property to value.
set_restorable(value)	Set the value of the restorable property to value.
<pre>store_position(row, column[, layer])</pre>	Store the BoardItem position for self access.
store_screen_position(row, column)	Store the screen position of the object.

### **Attributes**

animation	A property to get and set an Animation for this item.
column	Convenience method to get the current stored column of the item.
heading	Return the heading of the item.
height	Convenience method to get the height of the item.
inventory_space	Return the size that the Immovable item takes in the <i>Inventory</i> .
layer	Convenience method to get the current stored layer number of the item.
model	
particle_emitter	
row	Convenience method to get the current stored row of the item.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size	A read-only property that gives the size of the item as a 2 dimensions list.
width	Convenience method to get the width of the item.

### property animation

A property to get and set an Animation for this item.

**Important:** When an animation is set, the item is setting the animation's parent to itself.

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### can\_move()

Return the capability of moving of an item.

Obviously an Immovable item is not capable of moving. So that method always returns False.

#### Returns

False

# Return type

bool

```
collides_with(other, projection_offset: Vector2D = None)
```

Tells if this item collides with another item.

**Important:** collides\_with() does not take the layer into account! It is not desirable for the pygamelib to assume that 2 items on different layers wont collide. For example, if a player is over a door, they are on different layers, but logically speaking they are colliding. The player is overlapping the door. Therefor, it is the responsibility of the developer to check for layers in collision, if it is important to the game logic.

#### **Parameters**

- **other** (*BoardItem*) The item you want to check for collision.
- **projection\_offset** (*Vector2D*) A vector to offset this board item's position (not the position of the *other* item). Use this to detect a collision before moving the board item. You can pass the movement vector before moving to check if a collision will occur when moving.

### Return type

bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

## property column

Convenience method to get the current stored column of the item.

This is absolutely equivalent to access to item.pos[1].

#### **Returns**

The column coordinate

# Return type

int

Example:

```
if item.column != item.pos[1]:
    print('Something extremely unlikely just happened...')
```

#### debug\_info()

Return a string with the list of the attributes and their current value.

# Return type

str

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### display()

Print the model WITHOUT carriage return.

#### distance\_to(other)

Calculates the distance with an item.

### **Parameters**

**other** (*BoardItem*) – The item you want to calculate the distance to.

#### Returns

The distance between this item and the other.

### **Return type**

float

#### Example:

```
if npc.distance_to(game.player) <= 2.0:
    npc.seek_and_destroy = True</pre>
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

### property heading

Return the heading of the item.

This is a read only property that is updated by *store\_position()*.

The property represent the orientation and movement of the item in the board. It gives the difference between the item's centroid current and previous position. Thus, giving you both the direction and the distance of the movement. You can get the angle from here.

One of the possible usage of that property is to set the sprite/sprixel/model of a moving item.

### Returns

The heading of the item.

Vector2D

Example:

```
if my_item.heading.column > 0:
    my_item.sprixel.model = item_models["heading_right"]
```

**Warning:** Just after placing an item on the board, and before moving it, the heading cannot be trusted! The heading represent the direction and orientation of the **movement**, therefore, it is not reliable before the item moved.

### property height

Convenience method to get the height of the item.

This is absolutely equivalent to access to item.size[1].

#### Returns

The height

# Return type

int

Example:

```
if item.height > board.height:
    print('The item is too big for the board.')
```

### property inventory\_space

Return the size that the Immovable item takes in the *Inventory*.

### Returns

The size of the item.

### Return type

int

### property layer

Convenience method to get the current stored layer number of the item.

This is absolutely equivalent to access to item.pos[2].

### Returns

The layer number

#### Return type

int

Example:

```
if item.layer != item.pos[2]:
    print('Something extremely unlikely just happened...')
```

#### classmethod load(data)

Load data and create a new BoardItem out of it.

### **Parameters**

**data** (dict) – Data to create a new item (usually generated by serialize())

#### Returns

A new item.

### **Return type**

~pygamelib.board\_items.BoardItem

#### property model

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### overlappable()

This represent the capacity for a *BoardItem* to be overlapped by player or NPC.

### **Returns**

False

### Return type

bool

# property particle\_emitter

### pickable()

This represent the capacity for a *BoardItem* to be pick-up by player or NPC.

### Returns

False

### Return type

bool

### Example:

```
if mywall.pickable():
    print('Whoaa this wall is really light... and small...')
else:
    print('Really? Trying to pick-up a wall?')
```

### position\_as\_vector()

Returns the current item position as a Vector2D

#### Returns

The position as a 2D vector

Vector2D

Example:

```
gravity = Vector2D(9.81, 0)
next_position = item.position_as_vector() + gravity.unit()
```

```
render_to_buffer(buffer, row, column, height, width)
```

Render the board item into a display buffer (not a screen buffer).

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

### restorable()

This represent the capacity for an *Immovable* Movable item. A wall is not overlappable.

### Returns

False

#### Return type

bool

### property row

Convenience method to get the current stored row of the item.

This is absolutely equivalent to access to item.pos[0].

#### **Returns**

The row coordinate

#### **Return type**

int

Example:

```
if item.row != item.pos[0]:
    print('Something extremely unlikely just happened...')
```

### property screen\_column: int

A property to get/set the screen column.

# **Parameters**

**value** (*int*) – the screen column

# Return type

int

#### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

# Return type

int

### **serialize()** $\rightarrow$ dict

Return a dictionary with all the attributes of this object.

#### **Returns**

A dictionary with all the attributes of this object.

# Return type

dict

# set\_can\_move(value)

Set the value of the can\_move property to value.

### **Parameters**

**value** (bool) – The value to set.

Example:

```
item.set_can_move(False)
```

### set\_overlappable(value)

Set the value of the overlappable property to value.

#### **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_overlappable(False)
```

# set\_pickable(value)

Set the value of the pickable property to value.

#### **Parameters**

**value** (*bool*) – The value to set.

Example:

```
item.set_pickable(False)
```

# set\_restorable(value)

Set the value of the restorable property to value.

#### **Parameters**

**value** (*boo1*) – The value to set.

Example:

```
item.set_restorable(False)
```

# property size

A read-only property that gives the size of the item as a 2 dimensions list. The first element is the width and the second the height.

### Returns

The size.

526 Chapter 3. Tutorials

list

# Example:

 $store_position(row: int, column: int, layer: int = 0)$ 

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self position. It is a redundant information and might not be synchronized.

#### **Parameters**

- **row** (*int*) the row of the item in the *Board*.
- **column** (*int*) the column of the item in the *Board*.
- **layer** the layer of the item in the *Board*. By default layer is set to 0.

#### Example:

```
item.store_position(3,4)
```

### $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

# **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

### Example:

```
an_object.store_screen_coordinate(3,8)
```

### property width

Convenience method to get the width of the item.

This is absolutely equivalent to access to item.size[0].

#### Returns

The width

#### Return type

int

# Example:

```
if item.width > board.width:
    print('The item is too big for the board.')
```

# 3.5 constants

Accessible constants are the following:

General purpose:

PYGAMELIB\_VERSION

```
enum pygamelib.constants.Alignment(value)
```

Alignment regroup constants that used for various alignment purpose when organizing UI elements or other such graphical elements.

V\_CENTER and H\_CENTER respectively stand for Vertical center and Horizontal Center.

```
Member Type
```

int

Valid values are as follows:

```
LEFT = <Alignment.LEFT: 30000011>
```

```
RIGHT = <Alignment.RIGHT: 30000100>
```

CENTER = <Alignment.CENTER: 30000101>

```
TOP = <Alignment.TOP: 30000110>
```

BOTTOM = <Alignment.BOTTOM: 30000111>

V\_CENTER = <Alignment.V\_CENTER: 30001000>

H\_CENTER = <Alignment.H\_CENTER: 30001001>

```
enum pygamelib.constants.Algorithm(value)
```

A set of constants to identify the different algorithms used in the library (when a choice is possible). For now, it's only the path finding algorithm.

```
Member Type
```

int

Valid values are as follows:

```
BFS = <Algorithm.BFS: 90000100>
```

```
ASTAR = <Algorithm.ASTAR: 90000101>
```

```
enum pygamelib.constants.Direction(value)
```

Direction hold the basic constants for directions in the pygamelib. It is used for a wide variety of use cases from moving a player or NPC to indicate the direction of the movement of a cursor in the UI module!

```
Member Type
```

int

Valid values are as follows:

```
NO_DIR = <Direction.NO_DIR: 10000000>
```

UP = <Direction.UP: 10000001>

DOWN = <Direction.DOWN: 10000010>

```
LEFT = <Direction.LEFT: 10000011>
     RIGHT = <Direction.RIGHT: 10000100>
     DRUP = <Direction.DRUP: 10000101>
     DRDOWN = <Direction.DRDOWN: 10000110>
     DLUP = <Direction.DLUP: 10000111>
     DLDOWN = <Direction.DLDOWN: 10001000>
enum pygamelib.constants.EngineConstant(value)
     A couple of constants that controls the behavior of the engine itself.
          Member Type
              int
     Valid values are as follows:
     NO_PLAYER = <EngineConstant.NO_PLAYER: 90000001>
enum pygamelib.constants.EngineMode(value)
     Constants that controls the mode of the engine. So far, it's a choice between real time and turn by turn, but in the
     future there could be additional modes.
          Member Type
              int
     Valid values are as follows:
     MODE_REAL_TIME = <EngineMode.MODE_REAL_TIME: 90000002>
     MODE_TURN_BY_TURN = <EngineMode.MODE_TURN_BY_TURN: 90000003>
enum pygamelib.constants.InputValidator(value)
     Input Validators are used in the UI module to indicate what type of inputs are valid and/or accepted from the user.
          Member Type
              int
     Valid values are as follows:
     INTEGER_FILTER = <InputValidator.INTEGER_FILTER: 50000001>
     PRINTABLE_FILTER = <InputValidator.PRINTABLE_FILTER: 50000002>
enum pygamelib.constants.Orientation(value)
     Orientation regroup constants that are used to describe the orientation of graphical elements. The best example,
     is the BoxLayout: it can be organized vertically or horizontally.
          Member Type
              int
     Valid values are as follows:
     HORIZONTAL = <Orientation.HORIZONTAL: 30000001>
```

3.5. constants 529

VERTICAL = <Orientation.VERTICAL: 30000010>

```
enum pygamelib.constants.Permission(value)
```

Permission constants to separate what objects are allowed to interact with others. Mostly used to separate NPCs from Players.

```
Member Type
```

int

Valid values are as follows:

```
PLAYER_AUTHORIZED = <Permission.PLAYER_AUTHORIZED: 20000001>
```

```
NPC_AUTHORIZED = <Permission.NPC_AUTHORIZED: 20000010>
```

```
ALL_CHARACTERS_AUTHORIZED = <Permission.ALL_CHARACTERS_AUTHORIZED: 20000011>
```

```
ALL_MOVABLE_AUTHORIZED = <Permission.ALL_MOVABLE_AUTHORIZED: 20000100>
```

```
NONE_AUTHORIZED = <Permission.NONE_AUTHORIZED: 20000101>
```

enum pygamelib.constants.SizeConstraint(value)

SizeConstraint regroup constants that are used in element which the size can vary depending on context.

```
Member Type
```

int

Valid values are as follows:

```
DEFAULT_SIZE = <SizeConstraint.DEFAULT_SIZE: 60000001>
```

MINIMUM\_SIZE = <SizeConstraint.MINIMUM\_SIZE: 60000002>

MAXIMUM\_SIZE = <SizeConstraint.MAXIMUM\_SIZE: 60000003>

```
EXPAND = <SizeConstraint.EXPAND: 60000004>
```

```
enum pygamelib.constants.State(value)
```

A set of constants that describe the internal state of something. For example the, state of the *Game* engine that are used to process or not some events.

```
Member Type
```

int

Valid values are as follows:

```
RUNNING = <State.RUNNING: 40000001>
```

```
PAUSED = <State.PAUSED: 40000010>
```

STOPPED = <State.STOPPED: 40000011>

```
enum pygamelib.constants.TextStyle(value)
```

TextStyling is used to format characters or text. It is mostly used by Text.

```
Member Type
```

str

Valid values are as follows:

```
BOLD = <TextStyle.BOLD: '\x1b[1m'>
```

### UNDERLINE = <TextStyle.UNDERLINE: '\x1b[4m'>

The following constants are used in versions <= 1.3.0 and have been deprecated starting version 1.4.0.

### Directions:

- NO\_DIR: This one is used when no direction can be provided by an actuator (destination reached for a PathFinder for example)
- UP
- DOWN
- LEFT
- RIGHT
- DRUP: Diagonal right up
- DRDOWN: Diagonal right down
- DLUP: Diagonal Left up
- DLDOWN: Diagonal left down

#### Permissions:

- PLAYER AUTHORIZED
- NPC\_AUTHORIZED
- ALL\_PLAYABLE\_AUTHORIZED (deprecated in 1.2.0 in favor of ALL\_CHARACTERS\_AUTHORIZED)
- ALL\_CHARACTERS\_AUTHORIZED
- ALL\_MOVABLE\_AUTHORIZED
- NONE\_AUTHORIZED

# UI positions:

- ORIENTATION\_HORIZONTAL
- ORIENTATION\_VERTICAL
- ALIGN\_LEFT
- ALIGN\_RIGHT
- ALIGN CENTER

Actions states (for Actuators for example):

- RUNNING
- PAUSED
- STOPPED

Accepted input (mainly used in pygamelib.gfx.ui for input dialogs): \* INTEGER\_FILTER \* PRINTABLE\_FILTER Path Finding Algorithm Constants:

- ALGO\_BFS
- ALGO\_ASTAR

Text styling constants:

• BOLD

3.5. constants 531

UNDERLINE

Special constants:

- NO\_PLAYER: That constant is used to tell the Game object not to manage the player.
- MODE\_RT: Set the game object to Real Time mode. The game runs independently from the user input.
- MODE\_TBT : Set the game object to Turn By Turn mode. The game runs turn by turn and pause between each user input.

# 3.6 engine

# 3.6.1 Board

Bases: PglBaseObject

A class that represent a game board.

The board object is a 2D matrix of board items. This means that you can visualize it as a chessboard for example. All board items are positioned on this chessboard-like object and can be moved around.

The Board object is the base object to build a level. Once created to your liking you can add items from the *board\_items* module. You can also derived *BoardItem* to create your own board items, specific to your game.

If you want a detailed introduction to the Board object, go the the pygamelib wiki and read the "Getting started: the Board" article.

**Note:** In version 1.3.0 a new screen rendering stack was introduced. With this came the need for some object to hold more information about their state. This is the case for Board. To use partial display with the *Screen* buffer system the board itself needs to hold the information about were to draw and on what to focus on. The existing code will still work as the *Game* object takes care of forwarding the information to the Board. However, it is now possible to exploit the *Camera* object to create cut scenes and more interesting movements.

**Important:** Partial display related parameters are information used by the *display\_around()* method and the *Screen* object to either display directly the board (display\_around) or render the Board in the frame buffer. **You have to make sure that the focus element's position is updated**. If you use the player, you have nothing to do but the Camera object needs to be manually updated for example.

**Warning:** in 1.3.0 the notion of layers was added to the Board object. Layers are used to better manage items overlapping. For the moment, layers are automatically managed to expand and shrink on demand (or on a need basis). You can use the layer system to add some depth to your game but you should be warned that you may experience some issues. If it is the case please report them on the Github issues page. For existing code, the entire Board object behaves exactly like in version 1.2.x.

\_\_init\_\_(name: str = 'Board', size: list = None, ui\_borders: str = None, ui\_border\_bottom: str = '-',
 ui\_border\_top: str = '-', ui\_border\_left: str = '|', ui\_border\_right: str = '|', ui\_board\_void\_cell=' ',
 ui\_board\_void\_cell\_sprixel: Sprixel = None, player\_starting\_position: list = None,
 DISPLAY\_SIZE\_WARNINGS=False, parent=None, partial\_display\_viewport=None,
 partial\_display\_focus=None, enable\_partial\_display=False)

#### **Parameters**

- name (str) the name of the Board
- **size** (*list*) array [width,height] with width and height being int. The size of the board. If layers is not specified it is set to 5.
- player\_starting\_position (list) array [row,column] with row and column being int. The coordinates at which Game will place the player on change\_level().
- **ui\_borders** (*str*) To set all the borders to the same value
- **ui\_border\_left** (*str*) A string that represents the left border.
- **ui\_border\_right** (*str*) A string that represents the right border.
- **ui\_border\_top** (*str*) A string that represents the top border.
- ui\_border\_bottom (str) A string that represents the bottom border.
- ui\_board\_void\_cell (str) A string that represents an empty cell. This option is going to be the model of the BoardItemVoid (see pygamelib.board\_items.BoardItemVoid)
- parent (*Game*) The parent object (usually the Game object).
- **DISPLAY\_SIZE\_WARNINGS** (*boo1*) A boolean to show or hide the warning about boards bigger than 80 rows and/or columns.
- **enable\_partial\_display** (*bool*) A boolean to tell the Board to enable or not partial display of boards. Default: False.
- partial\_display\_viewport (list) A 2 int elements array that gives the radius of the partial display in number of row and column. Please see display\_around().
- partial\_display\_focus (BoardItem or Vector2D) An item to focus (i.e center) the view on. When partial display is enabled the rendered view will be centered on this focus point/item. It can be an item or a vector.

3.6. engine 533

# Methods

init([name, size, ui_borders,])	param name
	the name of the Board
attach(observer)	Attach an observer to this instance.
<pre>check_sanity()</pre>	Check the board sanity.
<pre>clear_cell(row, column[, layer])</pre>	Clear cell (row, column, layer)
detach(observer)	Detach an observer from this instance.
display()	Display the entire board.
<pre>display_around(item, row_radius, col- umn_radius)</pre>	Display only a part of the board.
<pre>generate_void_cell()</pre>	This method return a void cell.
<pre>get_immovables(**kwargs)</pre>	Return a list of all the Immovable objects in the Board.
<pre>get_movables(**kwargs) handle_notification(subject[, attribute, value])</pre>	Return a list of all the Movable objects in the Board. A virtual method that needs to be implemented by the observer.
<pre>init_board()</pre>	Initialize the board with BoardItemVoid that uses ui_board_void_cell_sprixel or ui_board_void_cell (in that order of preference) as model.
<pre>init_cell(row, column[, layer])</pre>	Initialize a specific cell of the board with Board-ItemVoid that uses ui_board_void_cell as model.
<pre>instantiate_item(data)</pre>	Instantiate a BoardItem from its serialized data.
<pre>item(row, column[, layer])</pre>	Return the item at the row, column, layer position if within board's boundaries.
layers(row, column)	A method to get the number of layers at the Board's given coordinates.
load([data])	Create a new Board object based on serialized data.
<pre>move(item, direction[, step])</pre>	Board.move() is a routing function.
neighbors(obj[, radius])	Returns a list of neighbors (non void item) around an object.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>place_item(item, row, column[, layer,])</pre>	Place an item at coordinates row, column and layer.
remove_item(item)	Remove an item from the board.
render_cell(row, column)	New in version 1.3.0.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the board into from the display buffer to the frame buffer.
serialize()	Return a serialized version of the board.
store_screen_position(row, column)	Store the screen position of the object.

534 Chapter 3. Tutorials

### **Attributes**

height	A convenience read only property to get the height of the Board.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
width	A convenience read only property to get the width of the Board.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

## Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# $check\_sanity() \rightarrow None$

Check the board sanity.

This is essentially an internal method called by the constructor.

# clear\_cell(row, column, layer=0)

Clear cell (row, column, layer)

This method clears a cell, meaning it position a void\_cell BoardItemVoid at these coordinates.

It also removes the items from the the list of movables and immovables.

#### **Parameters**

- $\mathbf{row}(int)$  The row of the item to remove
- **column** (*int*) The column of the item to remove
- **layer** (*int*) The layer of the item to remove. The default value is 0 to remain coherent with previous version of the library.

# Example:

```
myboard.clear_cell(3,4,0)
```

Warning: This method does not check the content before, it will overwrite the content.

3.6. engine 535

**Important:** In the case of a BoardComplexItem derivative (Tile, ComplexPlayer, ComplexNPC, etc.) clearing one cell of the entire item is enough to remove the entire item from the list of movables or immovables.

**Note:** Starting in 1.3.0 and the addition of board's layers, there is no more overlapping matrix. With no more moving items around this method should be a little faster. It also means that the layer parameter is really important (a wrong layer means that you'll clear the wrong cell). Be ready to catch an IndexError exception

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

#### Return type

bool

### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### $display() \rightarrow None$

Display the entire board.

This method display the Board (as in print()), taking care of displaying the borders, and everything inside.

It uses the \_\_str\_\_ method of the item, which by default uses (in order) BoardItem.sprixel and (if no sprixel is defined) BoardItem.model. If you want to override this behavior you have to subclass BoardItem.

```
display\_around(item, row\_radius, column\_radius) \rightarrow None
```

Display only a part of the board.

This method behaves like display() but only display a part of the board around an item (usually the player). Example:

```
# This will display only a total of 30 cells vertically and
# 60 cells horizontally.
board.display_around(player, 15, 30)
```

#### **Parameters**

- item (BoardItem) an item to center the view on (it has to be a subclass of BoardItem)
- **row\_radius** (*int*) The radius of display in number of rows showed. Remember that it is a radius not a diameter...
- **column\_radius** (*int*) The radius of display in number of columns showed. Remember that... Well, same thing.

It uses the same display algorithm than the regular display() method.

## generate\_void\_cell()

This method return a void cell.

If ui\_board\_void\_cell\_sprixel is defined it uses it, otherwise use ui\_board\_void\_cell to generate the void item.

#### Returns

A void board item

# **Return type**

BoardItemVoid

### Example:

```
board.generate_void_cell()
```

# get\_immovables(\*\*kwargs)

Return a list of all the Immovable objects in the Board.

See pygamelib.board\_items.Immovable for more on
 an Immovable object.

### **Parameters**

\*\*kwargs – an optional dictionnary with keys matching Immovables class members and value being something **contained** in that member.

#### Returns

A list of Immovable items

### Example:

```
for m in myboard.get_immovables():
    print(m.name)

# Get all the Immovable objects that type contains "wall"
    AND name contains fire
walls = myboard.get_immovables(type="wall",name="fire")
```

### get\_movables(\*\*kwargs)

Return a list of all the Movable objects in the Board.

See pygamelib.board\_items.Movable for more on a Movable object.

# **Parameters**

\*\*kwargs – an optional dictionnary with keys matching Movables class members and value being something contained in that member.

#### Returns

A list of Movable items

# Example:

```
for m in myboard.get_movables():
    print(m.name)

# Get all the Movable objects that has a type that contains "foe"
foes = myboard.get_movables(type="foe")
```

### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

### property height

A convenience read only property to get the height of the Board.

It is absolutely equivalent to access to board.size[1].

#### Returns

The height of the board.

# Return type

int

Example:

```
if board.size[1] != board.height:
    print('Houston, we have a problem...')
```

## init\_board()

Initialize the board with BoardItemVoid that uses ui\_board\_void\_cell\_sprixel or ui\_board\_void\_cell (in that order of preference) as model.

This method is automatically called by the Board's constructor.

Example:

```
myboard.init_board()
```

### $init_cell(row, column, layer=0) \rightarrow None$

Initialize a specific cell of the board with BoardItemVoid that uses ui\_board\_void\_cell as model.

# **Parameters**

- **row** (*int*) the row coordinate.
- **column** (*int*) the column coordinate.

Example:

```
myboard.init_cell(2,3,0)
```

### static instantiate\_item(data: dict)

Instantiate a BoardItem from its serialized data.

## **Parameters**

**data** (*dict*) – The data to use to build the item.

### Returns

an instance of a BoardItem.

**Important:** The actual object depends on the serialized data. It can be any derivative of BoardItem (even custom objects as long as they inherit from BoardItem) as long as they are importable by this class.

### Example:

```
# First get some board item serialization data. For example:
data = super_duper_npc.serialize()
# Then instantiate a new one:
another_super_duper_npc = Board.instantiate_item(data)
```

item(row, column, layer=-1)

Return the item at the row, column, layer position if within board's boundaries.

#### **Parameters**

- row (int) The row to probe.
- **column** (*int*) The column to probe.
- **layer** (*int*) The layer to probe (default: -1 i.e the top item).

### Return type

pygamelib.board\_items.BoardItem

#### Raises

PglOutOfBoardBoundException - if row, column or layer are out of bound.

```
layers(row, column) \rightarrow int
```

A method to get the number of layers at the Board's given coordinates.

## Returns

The number of layers of the board.

### Return type

int

## Example:

```
if board.layers(game.player.row, game.player.column) > 1:
    print('The player is stomping on something!')
```

## classmethod load(data: dict = None)

Create a new Board object based on serialized data.

If data is None, None is returned.

If a color component is missing from data, it is set to 0 (see examples).

Raises an exception if the color components are not integer.

### **Parameters**

**data** (*dict*) – Data loaded from JSON data (serialized).

#### Returns

Either a Board object or None if data where empty.

## **Return type**

Board | NoneType

### Raise

PglInvalidTypeException

Example:

```
# Loading from parsed JSON data
new_board = Board.load(json.load("board_lvl_01.json"))
```

move(item, direction, step=1)

Board.move() is a routing function. It does 2 things:

**1 - If the direction is a** *Vector2D***, round the** values to the nearest integer (as move works with entire board cells, i.e integers).

2 - route toward the right moving function depending if the item is complex or not.

Move an item in the specified direction for a number of steps.

#### **Parameters**

- item (pygamelib.board\_items.Movable) an item to move (it has to be a subclass of Movable)
- direction (Direction or Vector2D) a direction from constants
- **step** (*int*) the number of steps to move the item.

If the number of steps is greater than the Board, the item will be move to the maximum possible position.

If the item is not a subclass of Movable, an PglObjectIsNotMovableException exception (see *pygamelib*. base.PglObjectIsNotMovableException).

Example:

```
board.move(player,Direction.UP,1)
```

**Important:** if the move is successful, an empty BoardItemVoid (see pygamelib.boards\_item. BoardItemVoid) will be put at the departure position (unless the movable item is over an overlappable item). If the movable item is over an overlappable item, the overlapped item is restored.

**Important:** Also important: If the direction is a *Vector2D*, the values will be rounded to the nearest integer (as move works with entire board cells). It allows for movement accumulation before actually moving. The step parameter is not used in that case.

```
neighbors(obj, radius: int = 1)
```

Returns a list of neighbors (non void item) around an object.

This method returns a list of objects that are all around an object between the position of an object and all the cells at **radius**.

#### **Parameters**

- radius (int) The radius in which non void item should be included
- **obj** (BoardItem) The central object. The neighbors are calculated for that object.

### Returns

A list of BoardItem. No BoardItemVoid is included.

### **Raises**

**PglInvalidTypeException** – If radius is not an int.

Example:

```
for item in game.neighbors(npc, 2):
    print(f'{item.name} is around {npc.name} at coordinates '
        '({item.pos[0]},{item.pos[1]})')
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

### **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

place\_item(item, row: int, column: int, layer: int = 0, auto\_layer: bool = True)

Place an item at coordinates row, column and layer.

If row, column or layer are out of the board boundaries, a PglOutOfBoardBoundException is raised.

If the item is not a subclass of BoardItem, a PglInvalidTypeException

The observers are notified of a successful placement with the pygamelib.engine.Board.place\_item:item\_placed event. The item that was deleted is passed as the value of the event.

**Warning:** Nothing prevents you from placing an object on top of another. Be sure to check that. This method will check for items that are both overlappable **and** restorable to save them, but that's the extend of it.

## remove\_item(item)

Remove an item from the board.

If the item is a single BoardItem, this method is absolutely equivalent to calling <code>clear\_cell()</code>. If item is a derivative of BoardComplexItem, it is not as clear\_cell() only clears a specific cell (that can be part of a complex item). This method actually remove the entire item and clears all its cells.

The observers are notified of a successful removal with the pygamelib.engine.Board.remove\_item:item\_removed event. The item that was deleted is passed as the value of the event.

### **Parameters**

**item** (*BoardItem*) – The item to remove.

Example:

```
game.current_board().remove_item(game.player)
```

```
render_cell(row, column)
```

New in version 1.3.0.

Render the cell at given position.

This method always return a *Sprixel* (it could be an empty one though). It automatically render the highest item (if items are overlapping for example). If the rendered *Sprixel* is configured to have transparent background, this method is going to go through the layers to make sure that it is rendering the sprixels correctly (i.e. with the right background color).

For basic usage of the library it is unlikely that you will use it. It is part of the screen rendering stack introduced in version 1.3.0. Actually unless you need to write a different rendering system you won't use that method.

#### **Parameters**

- row (int) The row to render.
- **column** (*int*) The column to render.

### Return type

Sprixel

#### Raises

PglOutOfBoardBoundException – if row or column are out of bound.

## Example:

```
# This renders the board from the top left corner of the screen.
for row in range(0, myboard.height):
    for column in range(0, myboard.height):
        myscreen.place(
            myboard.render_cell(row, column)
        ),
        row,
        column,
```

**render\_to\_buffer**(*buffer*, *row*, *column*, *buffer\_height*, *buffer\_width*) → None

Render the board into from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

### **Parameters**

- **buffer** (*numpy.array*) A frame buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (int) – the screen column

## Return type

int

## property screen\_row: int

A property to get/set the screen row.

### **Parameters**

**value** (int) – the screen row

# Return type

int

### serialize()

Return a serialized version of the board.

### **Returns**

A dictionary containing the board's attributes.

Example:

```
serialized_board_data = myboard.serialize()
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

# property width

A convenience read only property to get the width of the Board.

It is absolutely equivalent to access to board.size[0].

## Returns

The width of the board.

# Return type

int

Example:

```
if board.size[0] != board.width:
    print('Houston, we have a problem...')
```

# 3.6.2 Game

Bases: PglBaseObject

A class that serve as a game engine.

This object is the central system that allow the management of a game. It holds boards (see *pygamelib.engine*. *Board*), associate it to level, takes care of level changing, etc.

**Note:** The game object has an object\_library member that is always an empty array except just after loading a board. In this case, if the board have a "library" field, it is going to be used to populate object\_library. This library is accessible through the Game object mainly so people have access to it across different Boards during level design in the editor. That architecture decision is debatable.

Note: The constructor of Game takes care of initializing the terminal to properly render the colors on Windows.

**Important:** The Game object automatically assumes ownership over the Player.

\_\_init\_\_(name: str = 'Game', player: Player = None, boards={}, current\_level=None, enable\_partial\_display=False, partial\_display\_viewport=None, partial\_display\_focus=None, mode=EngineMode.MODE\_TURN\_BY\_TURN, user\_update=None, input\_lag=0.01, user\_update\_paused=None)

# **Parameters**

- name (str) The Game name.
- boards (dict) A dictionary of boards with the level number as key and a board reference as value.
- **current\_level** (*int*) The current level.
- **enable\_partial\_display** (*bool*) A boolean to tell the Game object to enable or not partial display of boards. Default: False.
- partial\_display\_viewport (list) A 2 int elements array that gives the radius of the partial display in number of row and column. Please see display\_around().
- partial\_display\_focus (BoardItem) The object that is going to be the center of the view when the board is displayed.
- mode (EngineMode) The mode parameter configures the way the run() method is going to behave. The default value is EngineMode.MODE\_TURN\_BY\_TURN. In that mode, the Game object wait for an user input before looping. Exactly like when you wait for user input with get\_key(). The other possible value is EngineMode.MODE\_TURN\_BY\_TURN. RT stands for "Real Time". In that mode, the Game object waits for a minimal amount of time (0.01 i.e 100 FPS, configurable through the input\_lag parameter) in order to get the input from the user and call the update function right away. This parameter is only useful if you use Game.run().

- **user\_update** (*function*) A reference to the main program update function. The update function is called for each new frame. It is called with 3 parameters: the game object, the user input (can be None) and the elapsed time since last frame.
- user\_update\_paused (function) A reference to the update function called when the game is paused. It is called with the same 3 parameters than the regular update function: the game object, the user input (can be None) and the elapsed time since last frame. If not specified, the regular update function is called but nothing is done regarding NPCs, projectiles, animations, etc.
- **input\_lag** (*float | int*) The amount of time the run() function is going to wait for a user input before returning None and calling the update function. Default is 0.01.

### **Methods**

init([name, player, boards,])	
	param name
	The Game name.
<pre>actuate_npcs(level_number[, elapsed_time])</pre>	Actuate all NPCs on a given level
<pre>actuate_projectiles(level_number[,</pre>	Actuate all Projectiles on a given level
elapsed_time])	
<pre>add_board(level_number, board)</pre>	Add a board for the level number.
<pre>add_npc(level_number, npc[, row, column,])</pre>	Add a NPC to the game.
<pre>add_projectile(level_number, projectile[,])</pre>	Add a Projectile to the game.
<pre>animate_items(level_number[, elapsed_time])</pre>	That method goes through all the BoardItems of a
	given map and call Animation.next_frame().
attach(observer)	Attach an observer to this instance.
<pre>change_level(level_number)</pre>	Change the current level, load the board and place the
	player to the right place.
<pre>clear_screen()</pre>	Clear the whole screen (i.e: remove everything writ-
	ten in terminal)
<pre>clear_session_logs()</pre>	Delete all the log lines from the logs.
config([section])	Get the content of a previously loaded configuration
	section.
<pre>create_config(section)</pre>	Initialize a new config section.
<pre>current_board()</pre>	This method return the board object corresponding to
	the current_level.
<pre>delete_all_levels()</pre>	Delete all boards and their associated levels from the
	game object.
<pre>delete_level([lvl_number])</pre>	Delete a level and its associated Board from the game
	object.
detach(observer)	Detach an observer from this instance.
<pre>display_board()</pre>	Display the current board.
<pre>display_player_stats([life_model,</pre>	Display the player name and health.
void_model])	
<pre>get_board(level_number)</pre>	This method returns the board associated with a level
	number.
<pre>get_key()</pre>	Reads the next key-stroke returning it as a string.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
	observer.
<pre>insert_board(level_number, board)</pre>	Insert a board for the level number.

continues on next page

Table 8 – continued from previous page

instance(*args, **kwargs)	Returns the instance of the Game object
<pre>load_board(filename[, lvl_number])</pre>	Load a saved board
<pre>load_config(filename[, section])</pre>	Load a configuration file from the disk.
<pre>move_player(direction[, step])</pre>	Easy wrapper for Board.move().
neighbors([radius, obj])	Get a list of neighbors (non void item) around an object.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
pause()	Set the game engine state to PAUSE.
remove_npc(level_number, npc)	This methods remove the NPC from the level in pa-
	rameter.
run()	New in version 1.2.0.
<pre>save_board(lvl_number, filename)</pre>	Save a board to a JSON file
<pre>save_config([section, filename, append])</pre>	Save a configuration section.
session_log(line)	Add a line to the session logs.
session_logs()	Return the complete session logs since instantiation.
start()	Set the game engine state to RUNNING.
stop()	Set the game engine state to STOPPED.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

# **Attributes**

screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
state	Get/set the state of the game.

# actuate\_npcs(level\_number, elapsed\_time=0.0)

Actuate all NPCs on a given level

This method actuate all NPCs on a board associated with a level. At the moment it means moving the NPCs but as the Actuators become more capable this method will evolve to allow more choice (like attack use objects, etc.)

When all NPCs have been successfully actuated, the observers are notified of the change with the pygamelib.engine.Game.actuate\_npcs:npcs\_actuated event. Their is value passed for that event.

### **Parameters**

- **level\_number** (*int*) The number of the level to actuate NPCs in.
- **elapsed\_time** (*float*) The amount of time that passed since last call. This parameter is not mandatory.

# Example:

ygame.actuate_npcs(1)
-----------------------

**Note:** This method only move NPCs when their actuator state is RUNNING. If it is PAUSED or STOPPED, the NPC is not moved.

546 Chapter 3. Tutorials

**Note:** Since version 1.2.0 it's possible for a Movable item to have different vertical and horizontal movement steps, so actuate\_npc respect that by integrating the steps with a unit direction vector. It should be completely transparent and you should not expect any change. Just more movement freedom. If you do experience issues, please report a bug.

**Note:** Since version 1.2.0 and the appearance of the realtime mode, we have to account for movement speed. This method does it.

## actuate\_projectiles(level\_number, elapsed\_time=0.0)

Actuate all Projectiles on a given level

This method actuate all Projectiles on a board associated with a level. This method differs from actuate\_npcs() as some logic is involved with projectiles that NPC do not have. This method decrease the available range by projectile.step each time it's called. It also detects potential collisions. If the available range falls to 0 or a collision is detected the projectile hit\_callback is called.

This method respects the Projectile.collision\_exclusions parameter and does not register collisions with objects of a type present in that list.

**Important:** In this method, projectiles do not collide with overlappable items. If you want to detect collisions with overlappable objects, please implement your own projectile actuation method.

#### **Parameters**

- **level\_number** (*int*) The number of the level to actuate Projectiles in.
- **elapsed\_time** (*float*) The amount of time that passed since last call. This parameter is not mandatory.

When all Projectiles have been successfully actuated, the observers are notified of the change with the pygamelib.engine.Game.actuate\_projectiles:projectiles\_actuated event. Their is value passed for that event.

Example:

```
mygame.actuate_projectiles(1)
```

**Note:** This method only move Projectiles when their actuator state is RUNNING. If it is PAUSED or STOPPED, the Projectile is not moved.

**Important:** Please have a look at the *pygamelib.board\_items.Projectile.hit()* method for more information on the projectile hit mechanic.

 $add\_board(level\ number:\ int,\ board:\ Board) \rightarrow None$ 

Add a board for the level number.

This method associate a Board (pygamelib.engine.Board) to a level number.

If the partial display is enabled at Game level (i.e: partial\_display\_viewport is not None and enable\_partial\_display is True), this method propagate the settings to the board automatically. Same for partial\_display\_focus.

#### Example:

```
game.add_board(1,myboard)
```

#### **Parameters**

- **level\_number** (*int*) the level number to associate the board to.
- **board** (pygamelib.engine.Board) a Board object corresponding to the level number.

#### Raises

**PglInvalidTypeException** – If either of these parameters are not of the correct type.

add\_npc(level\_number, npc, row=None, column=None, layer=None, auto\_layer=True)

Add a NPC to the game. It will be placed on the board corresponding to the level\_number. If row and column are not None, the NPC is placed at these coordinates. Else, it's randomly placed in an empty cell.

Example:

```
game.add_npc(1,my_evil_npc,5,2)
```

#### **Parameters**

- **level\_number** (*int*) the level number of the board.
- npc (pygamelib.board\_items.NPC) the NPC to place.
- row (int) the row coordinate to place the NPC at.
- **column** (*int*) the column coordinate to place the NPC at.

If either of these parameters are not of the correct type, a PglInvalidTypeException exception is raised.

**Important:** If the NPC does not have an actuator, this method is going to affect a pygamelib.actuators.RandomActuator() to npc.actuator. And if npc.step == None, this method sets it to 1

add\_projectile(level\_number, projectile, row=None, column=None)

Add a Projectile to the game. It will be placed on the board corresponding to level\_number. Neither row nor column can be None.

Example:

```
game.add_projectile(1, fireball, 5, 2)
```

#### **Parameters**

- **level\_number** (*int*) the level number of the board.
- **projectile** (*Projectile*) the Projectile to place.
- **row** (*int*) the row coordinate to place the Projectile at.
- **column** (*int*) the column coordinate to place the Projectile at.

If either of these parameters are not of the correct type, a PglInvalidTypeException exception is raised.

**Important:** If the Projectile does not have an actuator, this method is going to affect pygamelib.actuators.RandomActuator(moveset=[RIGHT]) to projectile.actuator. And if projectile.step == None, this method sets it to 1.

```
animate_items(level_number, elapsed_time=0.0)
```

That method goes through all the BoardItems of a given map and call Animation.next\_frame().

When all items have been successfully animated, the observers are notified of the change with the pygamelib.engine.Game.animate\_items\_items\_animated event. Their is value passed for that event.

#### **Parameters**

- **level\_number** (*int*) The number of the level to animate items in.
- **elapsed\_time** (*float*) The amount of time that passed since last call. This parameter is not mandatory.

#### Raise

PglInvalidLevelException PglInvalidTypeException

## Example:

```
mygame.animate_items(1)
```

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

# **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

## Returns

True or False depending on the success of the operation.

### Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## **change\_level**( $level\_number: int$ ) $\rightarrow$ None

Change the current level, load the board and place the player to the right place.

# Example:

```
game.change_level(1)
```

# **Parameters**

**level\_number** (*int*) – the level number to change to.

#### Raises

base.PglInvalidTypeException - If parameter is not an int.

#### clear\_screen()

Clear the whole screen (i.e: remove everything written in terminal)

Deprecated since version 1.2.0: Starting 1.2.0 we are using the pygamelib.engine. Screen object to manage the screen. That function is a simple forward and is kept for backward compatibility only. You should use Game.screen.clear()

# $clear\_session\_logs() \rightarrow None$

Delete all the log lines from the logs.

Example:

```
game = Game.instance()
game.clear_logs()
```

**Note:** The session log system is nothing more than a list to do your "debug prints". If you want a real logging system, please use Python logging module.

```
config(section: str = 'main') \rightarrow dict
```

Get the content of a previously loaded configuration section.

#### **Parameters**

**section** (str) – The name of the section.

Example:

```
if mygame.config('main')['pgl-version-required'] < 10200:
    print('The pygamelib version 1.2.0 or greater is required.')
    exit()</pre>
```

## $create\_config(section: str) \rightarrow None$

Initialize a new config section.

The new section is a dictionary.

#### **Parameters**

**section** (str) – The name of the new section.

Example:

```
if mygame.config('high_scores') is None:
    mygame.create_config('high_scores')
mygame.config('high_scores')['first_place'] = mygame.player.name
```

# $current\_board() \rightarrow Board$

This method return the board object corresponding to the current\_level.

Example:

```
game.current_board().display()
```

If current\_level is set to a value with no corresponding board a PglException exception is raised with an invalid\_level error.

### delete\_all\_levels()

Delete all boards and their associated levels from the game object.

You might want to think twice before using that function...

Example:

```
game.delete_all_levels()
```

# delete\_level(lvl\_number: int = None)

Delete a level and its associated Board from the game object.

Both the level and the board can't be used after that (unless they are reloaded or replaced of course).

#### **Parameters**

**lvl\_number** (*int*) – The number of the level to remove.

#### Raises

- base.PglInvalidTypeException If parameter is not an int.
- base.PglInvalidLevelException If parameter is not a valid level.

Example:

```
my_game.delete_level(1)
```

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

### Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### display\_board()

Display the current board.

The behavior of that function is dependant on how you configured this object. If you set enable\_partial\_display to True AND partial\_display\_viewport is set to a correct value, it will call Game.current\_board().display\_around() with the correct parameters. The partial display will be centered on the player (Game.player). Otherwise it will just call Game.current\_board().display().

If the player is not set or is set to EngineConstant.NO\_PLAYER partial display won't activate automatically.

Example:

```
mygame.enable_partial_display = True
# Number of rows, number of column (on each side, total viewport
# will be 20x20 in that case).
mygame.partial_display_viewport = [10, 10]
```

(continues on next page)

(continued from previous page)

```
# This will call Game.current_board().display_around()
mygame.display()
mygame.enable_partial_display = False
# This will call Game.current_board().display()
mygame.display()
```

 $\label{likelihood} \mbox{display-player\_stats} (\mbox{\it life\_model} = \mbox{\it $1b$} [41m \mbox{\it $1b$} [0m', void\_model = \mbox{\it $1b$} [40m \mbox{\it $1b$} [0m')$$ 

Display the player name and health.

Deprecated since version This: method is completely deprecated and not even compatible with the Screen Buffer system. It will be removed in 1.4.0.

This method print the Player name, a health bar (20 blocks of life\_model). When life is missing the complement (20-life missing) is printed using void model. It also display the inventory value as "Score".

### **Parameters**

- $life_model(str)$  The character(s) that should be used to represent the *remaining* life.
- **void\_model** (*str*) The character(s) that should be used to represent the *lost* life.

Note: This method might change in the future. Particularly it could take a template of what to display.

```
get\_board(level\ number:\ int) \rightarrow Board
```

This method returns the board associated with a level number. :param level\_number: The number of the level. :type level\_number: int

### Raises

**PglInvalidTypeException** – if the level\_number is not an int.

Example:

```
level1_board = mygame.get_board(1)
```

### static get\_key()

Reads the next key-stroke returning it as a string.

Example:

```
key = Utils.get_key()
if key == Utils.key.UP:
    print("Up")
elif key == "q"
    exit()
```

**Note:** See *readkey* documentation in *readchar* package.

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Anv*) The new value of the attribute. This can be None.

```
insert_board(level_number: int, board: Board) → None
```

Insert a board for the level number.

This method does basically the same thing than add\_board() except that if the level number is already associated it re-affect the numbers down.

#### Example:

```
game.insert_board(1,myboard_1)
# level number 1 is associated with myboard_1
game.insert_board(2,myboard_2)
# level number 1 is associated with myboard_1
# level number 2 is associated with myboard_2
game.insert_board(2,myboard_3)
# level number 1 is associated with myboard_1
# level number 2 is now associated with myboard_3
# level number 3 is associated with myboard_2
```

## **Parameters**

- **level\_number** (*int*) the level number to associate the board to.
- board (pygamelib.engine.Board) a Board object corresponding to the level number.

### Raises

**PglInvalidTypeException** – If either of these parameters are not of the correct type.

```
classmethod instance(*args, **kwargs)
```

Returns the instance of the Game object

Creates a Game object on first call an then returns the same instance on further calls

#### Returns

Instance of Game object

# load\_board(filename, lvl\_number=0)

Load a saved board

Load a Board saved on the disk as a JSON file. This method creates a new Board object, populate it with all the elements (except a Player) and then return it.

If the filename argument is not an existing file, the open function is going to raise an exception.

This method, load the board from the JSON file, populate it with all BoardItem included, check for sanity, init the board with BoardItemVoid and then associate the freshly created board to a lvl\_number. It then create the NPCs and add them to the board.

# **Parameters**

• **filename** (str) – The file to load

• **lvl\_number** (*int*) – The level number to associate the board to. Default is 0.

#### Returns

a newly created board (see pygamelib.engine.Board)

### Example:

```
mynewboard = game.load_board( 'awesome_level.json', 1 )
game.change_level( 1 )
```

```
load_config(filename: str, section: str = 'main') \rightarrow dict
```

Load a configuration file from the disk. The configuration file must respect the INI syntax. The goal of these methods is to simplify configuration files management.

### **Parameters**

- **filename** (*str*) The filename to load. does not check for existence.
- **section** (*str*) The section to put the read config file into. This allow for multiple files for multiple purpose. Section is a human readable unique identifier.

#### Raises

- FileNotFoundError If filename is not found on the disk.
- json.decoder.JSONDecodeError If filename could not be decoded as JSON.

#### Returns

The parsed data.

## Return type

dict

**Warning:** breaking changes: before v1.1.0 that method use to load file using the configparser module. This have been dumped in favor of json files. Since that methods was apparently not used, there is no backward compatibility.

# Example:

```
mygame.load_config('game_controls.json','game_control')
```

### move\_player(direction, step=1)

Easy wrapper for Board.move().

Example:

```
mygame.move_player(Direction.RIGHT,1)
```

## neighbors(radius=1, obj=None)

Get a list of neighbors (non void item) around an object.

This method returns a list of objects that are all around an object between the position of an object and all the cells at **radius**.

### **Parameters**

- radius (int) The radius in which non void item should be included
- **object** (pygamelib.board\_items.BoardItem) The central object. The neighbors are calculated for that object. If None, the player is the object.

### Returns

A list of BoardItem, No BoardItemVoid is included.

#### Raises

PglInvalidTypeException – If radius is not an int.

## Example:

```
for item in game.neighbors(2):
    print(f'{item.name} is around player at coordinates '
        '({item.pos[0]},{item.pos[1]})')
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

## **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## pause()

Set the game engine state to PAUSE.

Example:

```
mygame.pause()
```

remove\_npc(level\_number, npc)

This methods remove the NPC from the level in parameter.

### **Parameters**

- **level** (*int*) The number of the level from where the NPC is to be removed.
- **npc** (*NPC*) The NPC object to remove.

Example:

```
mygame.remove_npc(1, dead_npc)
```

#### run()

New in version 1.2.0.

The run() method act as the main game loop and does a number of things for you:

1. It grabs the user input. If the Game object is configured with MODE\_TBT (the default), nothing happen until the user hit a key. If the mode is set to MODE\_RT, it will wait for input\_lag secondes for a user input before going to step 3.

- 2. It calculate the elapsed time between 2 frames.
- 3. Accumulates the elapsed time in the player dtmove variable (if there is a player object configured)
- 4. It sets the cursor position to 0,0 (meaning that your user\_update function will draw on top of the previously drawn window). The Board.display() and Board.display\_around() method clean the end of their line.
- 5. It calls the user\_update function with 3 parameters: the game object, the key hit by the user (it can be None) and the elapsed time between to calls.
- 6. Clears the end of the screen.
- 7. Actuates NPCs (If there is at least one Board manage by Game).
- 8. Actuates projectiles (If there is at least one Board manage by Game).
- 9. Animates items (If there is at least one Board manage by Game).

On the subject of particle emitters, the *Board* object automatically update the ones that are attached to BoardItems. For all other particle emitters you need to call the update method of the emitters yourself (for now).

In version 1.2.X, there was a bug when the game was paused. In that case nothing was happening anymore. The user update function was not called and events were not processed. On top of that it was impossible to use run() without associating a board object with a level. Starting with version 1.3.0, it is now possible to use run() without associating a board object with a level. There is also a new parameter to the constructor (user\_update\_paused) that allows you to specify a function that will be called when the game is paused. This function will be called with the same 3 parameters than the regular update function: the game object, the user input (can be None) and the elapsed time since last frame.

**Important:** If you try to set the game state to PAUSED and the user\_update\_paused function is not defined, a notification will be issued and the game will continue to run. The notification message is pygamelib.engine.Game.run:PauseNotAvailable

### Raises

PglInvalidTypeException, PglInvalidTypeException

# Example:

mygame.run()

### save\_board(lvl number, filename)

Save a board to a JSON file

This method saves a Board and everything in it but the BoardItemVoid.

Not check are done on the filename, if anything happen you get the exceptions from open().

## **Parameters**

- **lvl\_number** (*int*) The level number to get the board from.
- **filename** (*str*) The path to the file to save the data to.

# Raises

- **PglInvalidTypeException** If any parameter is not of the right type
- *PglInvalidLevelException* If the level is not associated with a Board.

## Example:

```
game.save_board( 1, 'hac-maps/level1.json')
```

If Game.object\_library is not an empty array, it will be saved also.

**Warning:** In version 1.3.0 the *Board* class changed a lot and a layer system has been added. Therefor, boards saved from version 1.3.0+ are *not* compatible with previous version. Previous boards can be loaded (*Game.load\_board(*) is backward compatible), but when saved they will be converted to the new format.

**save\_config**( $section: str = None, filename: str = None, append: bool = False) <math>\rightarrow$  None Save a configuration section.

#### **Parameters**

- **section** (*str*) The name of the section to save on disk.
- **filename** (*str*) The file to write in. If not provided it will write in the file that was used to load the given section. If section was not loaded from a file, save will raise an exception.
- **append** (*bool*) Do we need to append to the file or replace the content (True = append, False = replace)

Example:

```
mygame.save_config('game_controls', 'data/game_controls.json')
```

## property screen\_column: int

A property to get/set the screen column.

# **Parameters**

**value** (int) – the screen column

# **Return type**

int

## property screen\_row: int

A property to get/set the screen row.

# **Parameters**

**value** (int) – the screen row

# Return type

int

### **session\_log**(line: str) $\rightarrow$ None

Add a line to the session logs.

Session logs needs to be activated first.

### **Parameters**

**line** (str) – The line to add to the logs.

Example:

```
game = Game.instance()
game.ENABLE_SESSION_LOGS = True
game.session_log('Game engine initialized')
```

**Note:** The session log system is nothing more than a list to do your "debug prints". If you want a real logging system, please use Python logging module.

```
session_logs() \rightarrow list
```

Return the complete session logs since instantiation.

Example:

```
game = Game.instance()
game.ENABLE_SESSION_LOGS = True
for line in game.logs():
    print(line)
```

**Note:** The session log system is nothing more than a list to do your "debug prints". If you want a real logging system, please use Python logging module.

#### start()

Set the game engine state to RUNNING.

The game has to be RUNNING for actuate\_npcs() and move\_player() to do anything.

Example:

```
mygame.start()
```

### property state

Get/set the state of the game.

### **Parameters**

**value** (*State*) – The new state of the game (from the constants module).

## Returns

The state of the game.

# Return type

State

The observers are notified of a change of state with the pygamelib.engine.Game.state event. The new state is passed as the value of the event.

# stop()

Set the game engine state to STOPPED.

Example:

```
mygame.stop()
```

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

### Example:

```
an_object.store_screen_coordinate(3,8)
```

# 3.6.3 Inventory

class pygamelib.engine.Inventory(max size=10, parent=None)

Bases: PglBaseObject

A class that represent the Player (or NPC) inventory.

This class is pretty straightforward: it is an object container, you can add, get and remove items and you can get a value from the objects in the inventory.

On top of that, starting with version 1.3.0, a constraints system has been added. It allows to specify a certain amount of constraints that will be applied to the items when they are added to the inventory.

For the moment, constraints are limited to the number of items with a given type/ name/value (any combination of these three).

When a constraint is violated, the item is not added to the inventory and a notification is broadcasted to the observers of the inventory. A PglInventoryException is also raised with name "constraint\_violation" and the constraint details in description.

**Note:** You can print() the inventory. This is mostly useful for debug as you want to have a better display in your game.

**Warning:** The *Game* engine and *Player* takes care to initiate an inventory for the player, you don't need to do it.

**\_\_init\_\_**(max\_size=10, parent=None)

The constructor takes two parameters: the maximum size of the inventory. And the Inventory owner/parent.

Each *BoardItem* that is going to be put in the inventory has a size (default is 1), the total addition of all these size cannot exceed max\_size.

#### **Parameters**

- max\_size (int) The maximum size of the inventory. Default value: 10.
- parent The parent object (usually a BoardItem).

# **Methods**

init([max_size, parent])	The constructor takes two parameters: the maximum size of the inventory.
<pre>add_constraint(constraint_name[, item_type,])</pre>	, , , , , , , , , , , , , , , , , , ,
add_item(item)	Add an item to the inventory.
attach(observer)	Attach an observer to this instance.
available_space()	Return the available space in the inventory.
<pre>clear_constraints()</pre>	Remove all constraints from the inventory.
<pre>delete_item(name)</pre>	Delete THE FIRST item matching the name given in argument.
<pre>delete_items(name)</pre>	Delete ALL items matching the name given in argument.
detach(observer)	Detach an observer from this instance.
empty()	Empty the inventory.
<pre>get_item(name)</pre>	Return the FIRST item corresponding to the name given in argument.
<pre>get_items(name)</pre>	Return ALL items matching the name given in argument.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>items_name()</pre>	Return the list of all items names in the inventory.
load(data)	Load serialized data into a new Inventory object.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>remove_constraint(constraint_name)</pre>	
search(query)	Search for objects in the inventory.
serialize()	Serialize the inventory in a dictionary.
size()	Return the cumulated size of the inventory.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
value()	Return the cumulated value of the inventory.

# **Attributes**

constraints	
items	Return the list of all items in the inventory.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

 $add\_constraint\_name: str, item\_type: str = None, item\_name: str = None, item\_value: int = None, max\_number: int = 1)$ 

Add a constraint to the inventory.

## **Parameters**

- $constraint_name(str)$  the name of the constraint.
- **item\_type** (*str*) the type of the item.
- **item\_name** (*int*) the name of the item.
- item\_value the value of the item.

560 Chapter 3. Tutorials

• max\_number (int) – the maximum number of items that match the item\_\* parameters that can be in the inventory.

The observers are notified of the addition of the constraint with the pygamelib.engine.Inventory.add\_constraint event. The constraint that was added is passed as the value of the event as a dictionnary.

New in version 1.3.0.

### add\_item(item)

Add an item to the inventory.

This method will add an item to the inventory unless:

- it is not an instance of BoardItem,
- you try to add an item that is not pickable,
- there is no more space left in the inventory (i.e. the cumulated size of the inventory + your item.inventory\_space is greater than the inventory max\_size)
- An existing constraint is violated.

### **Parameters**

```
item (BoardItem) – the item you want to add
```

#### Returns

The index of the newly added item in the inventory or None if the item could not be added.

### Return type

int|None

### Raise

```
PglInventoryException, PglInvalidTypeException
```

When an item is successfully added, the observers are notified of the change with the pygamelib.engine.Inventory.add\_item event. The item that was added is passed as the value of the event.

When something goes wrong exceptions are raised. The following exceptions can be raised (*PglInventoryException*):

- not\_pickable: The item you try to add is not pickable.
- not\_enough\_space: There is not enough space left in the inventory.
- constraint\_violation: A constraint is violated.

A PglInvalidTypeException is raised when the item you try to add is not a BoardItem.

## Example:

```
item = Treasure(model=graphics.Models.MONEY_BAG,size=2,name='Money bag')
try:
    mygame.player.inventory.add_item(item)
expect PglInventoryException as e:
    if e.error == 'not_enough_space':
        print(f"Impossible to add {item.name} to the inventory, there is no"
        "space left in it!")
        print(e.message)
    elif e.error == 'not_pickable':
        print(e.message)
```

**Note:** In versions prior to 1.3.0, the inventory object was changing the name of the item if another item with the same name was already in the inventory. This is (fortunately) not the case anymore. The Inventory class does NOT modify the items that are stored into it anymore.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

#### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# $\textbf{available\_space()} \rightarrow \text{int}$

Return the available space in the inventory.

That is to say, Inventory.max\_size - Inventory.size().

The returned number is comprised between 0 and Inventory.max\_size.

### Returns

The size as an int.

# Return type

int

Example:

```
method()
```

# clear\_constraints()

Remove all constraints from the inventory.

The observers are notified with the pygamelib.engine.Inventory.clear\_constraints event. The value is set to None for this event.

New in version 1.3.0.

## property constraints

Return the list of all constraints in the inventory.

#### **Returns**

a list of constraints (dict)

### **Return type**

list

## Example:

```
for cstr in game.player.inventory.constraints:
    print(f" - {cstr[name]}")
```

### delete\_item(name)

Delete THE FIRST item matching the name given in argument.

#### **Parameters**

**name** (str) – the name of the items you want to delete.

When an item is successfully removed, the observers are notified of the change with the pygamelib.engine.Inventory.delete\_item event. The item that was deleted is passed as the value of the event.

Example:

```
mygame.player.inventory.delete_item('heart_1')
```

**Important:** Starting with version 1.3.0 this method does not raise exceptions anymore. It's behavior also changed from deleting a precise item to deleting the first one that matches the name.

## delete\_items(name)

Delete ALL items matching the name given in argument.

#### **Parameters**

**name** (str) – the name of the items you want to delete.

The observers are notified of each deletion with the pygamelib.engine.Inventory.delete\_item event. The item that was deleted is passed as the value of the event.

Example:

```
mygame.player.inventory.delete_items('heart_1')
```

New in version 1.3.0.

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*Pg1Base0bject*) – An observer to detach from this object.

## Returns

True or False depending on the success of the operation.

### Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# empty()

Empty the inventory.

The observers are notified that the Inventory has been emptied with the pygamelib.engine.Inventory.empty event. Nothing is passed as the value.

### Example:

```
if inventory.size() > 0:
  inventory.empty()
```

## get\_item(name)

Return the FIRST item corresponding to the name given in argument.

#### **Parameters**

**name** (str) – the name of the item you want to get.

### Returns

An item.

# Return type

BoardItem | None

# Example:

```
life_container = mygame.player.inventory.get_item('heart_1')
if isinstance(life_container,GenericActionableStructure):
    life_container.action(life_container.action_parameters)
```

**Note:** Please note that the item object reference is returned but nothing is changed in the inventory. The item hasn't been removed.

**Important:** Starting with version 1.3.0 this method does not raise exceptions anymore. Instead it returns None if no item is found. It's behavior also changed from returning a precise item to the first one that matches the name.

### get\_items(name)

Return ALL items matching the name given in argument.

#### **Parameters**

**name** (str) – the name of the item you want to get.

#### Returns

An array of items.

# Return type

list

# Example:

```
for life_container in mygame.player.inventory.get_items('heart_1'):
    if isinstance(life_container,GenericActionableStructure):
        life_container.action(life_container.action_parameters)
```

**Note:** Please note that the item object reference is returned but nothing is changed in the inventory. The item hasn't been removed.

New in version 1.3.0.

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## property items

Return the list of all items in the inventory.

#### Returns

a list of BoardItem

### **Return type**

list

## Example:

```
for item in game.player.inventory.items:
    print(f"This is a mighty item: {item.name}")
```

# items\_name()

Return the list of all items names in the inventory.

### Returns

a list of string representing the items names.

#### Return type

list

## classmethod load(data: dict)

Load serialized data into a new Inventory object.

#### **Parameters**

data (dict) – The serialized data

### **Returns**

A new Inventory object.

## **Return type**

Inventory

New in version 1.3.0.

Example:

```
my_player.inventory = Inventory.load(data)
```

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## remove\_constraint(constraint\_name: str)

Remove a constraint from the inventory.

#### **Parameters**

 $constraint_name(str)$  – the name of the constraint.

The observers are notified of the removal of the constraint with the pygamelib.engine.Inventory.remove\_constraint event. The constraint that was removed is passed as the value of the event as a dictionnary.

New in version 1.3.0.

# property screen\_column: int

A property to get/set the screen column.

### **Parameters**

value (int) – the screen column

#### Return type

int

## property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

## Return type

int

## search(query)

Search for objects in the inventory.

All objects that matches the query are going to be returned. Search is performed on the name and type of the object.

### **Parameters**

query – the query that items in the inventory have to match to be returned

### Returns

a list of BoardItems.

## Return type

list

Example:

```
for item in game.player.inventory.search('mighty'):
    print(f"This is a mighty item: {item.name}")
```

## serialize()

Serialize the inventory in a dictionary.

### Returns

The serialized data.

### **Return type**

dict

New in version 1.3.0.

Example:

```
json.dump(my_inventory.serialize(), out_file)
```

### size()

Return the cumulated size of the inventory. It can be used in the UI to display the size compared to max\_size for example.

### Returns

size of inventory

# Return type

int

Example:

```
print(f"Inventory: {mygame.player.inventory.size()}/"
  "{mygame.player.inventory.max_size}")
```

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## value()

Return the cumulated value of the inventory. It can be used for scoring for example.

#### Returns

value of inventory

# Return type

int

Example:

```
if inventory.value() >= 10:
    print('Victory!')
    break
```

## 3.6.4 Screen

```
class pygamelib.engine.Screen(width: int = None, height: int = None)
Bases: PglBaseObject
```

The screen object is pretty straightforward: it is an object that allow manipulation of the screen.

**Warning:** Starting with version 1.3.0 the terminal parameter has been removed. The Screen object now takes advantage of base.Console.instance() to get a reference to a blessed.Terminal object.

Version 1.3.0 introduced a new way of managing the screen. It rely on an internally managed display buffer that allows for easier positioning and more regular rendering. This comes at a cost though as the performances takes a hit. The screen should still be able to be refreshed between 50 and 60+ times per seconds (and still around 30 times per second within a virtual machine). These numbers obviously depends on the terminal used, the screen size and the content to display.

This change introduce two ways of displaying things on the screen:

- The **Improved Screen Management** stack (referred to as ISM later in the doc).
- The Legacy Direct Display stack.

It is safer to consider them mutually incompatible. In reality the **Improved Screen Management** will always use the whole display but you can use the methods from the **Direct Display** stack to write over the buffer. It is really **NOT** advised.

We introduced the **Improved Screen Management** stack because the direct display is messy and does not allow us to do what we want in term of positioning, UI, etc.

A typical usage consist of:

- Placing elements on the screen with place()
- Update the screen with update()

That's it! The screen maintain its own state and knows when to re-render the display buffer. You don't need to manually call *render()*. This helps with performances as the frame buffer is only rendered when needed.

Example:

```
screen = Screen()
# The next 3 lines do the same thing: display a message centered on the screen.
# Screen Buffer style
screen.place('This is centered', screen.vcenter, screen.hcenter)
screen.update()
# Direct Display style
screen.display_at('This is centered', screen.vcenter, screen.hcenter)
# The rest of this example uses the Screen Buffer (because placing a Board
# anywhere on the Screen is not supported by the Direct Display stack).
# delete the previous message and place a Board at the center of the screen
```

(continues on next page)

(continued from previous page)

```
screen.delete(screen.vcenter, screen.hcenter)
screen.place(
   my_awesome_board,
   screen.vcenter - int(my_awesome_board.height/2),
   screen.hcenter - int(my_awesome_board.width/2)
)
screen.update()
```

### Precisions about the Improved Screen Management stack:

You don't need to know how the frame buffer works to use it. However, if you are interested in more details, here they are.

The Improved Screen Management stacks uses a double numpy buffer to represent the screen. One buffer is used to place elements as objects (that's the buffer managed by place() or delete()). It is never directly printed to the screen. It is here to simplify screen maintenance. This buffer is called the **display buffer**. It is practical to use to place, move and delete elements on the screen space. But as said before it cannot be directly printed to the screen. It needs to be rendered first.

For example, if you want to use a sprite on a title screen and want to move it around (or animate the screen). Normally (i.e with Direct Display) you would display the sprite at a specific position and then would either call <code>clear()</code> or overwrite all the sprite with spaces to erase and replace and/or move it. And that's very slow.

With the **Improved Screen Management** you *place()* the sprite and then just *delete()* it. And since it is only one object reference it is a very fast operation (we only place or delete one cell of the buffer).

When *update()* is called, it first look at the state of the buffers and call *render()* if needed (i.e. if something has change in the display buffer). The buffers are only rendered when needed.

When *render()* is called it goes through the display buffer and render each elements transforming it into a printable sequence that is stored in the frame buffer. The rendering is done from the bottom right corner of the screen to the top left corner. This allows for cleaning junk characters at no additional cost.

**TL;DR:** The **display buffer** hold the objects placed on the screen while the **frame buffer** hold the rendered representation of the display buffer.

The Screen object also inherits from the *PglBaseObject* and if the object that is *place()*-ed is an instance of *PglBaseObject*, the screen will automatically attach itself to the object. When notified of a change it will trigger a render cycle before the next update.

In terms of performances, depending on your terminal emulator and CPU you will most certainly achieve over 30 FPS. Here are a couple of benchmark results:

- On an Intel Core i7 @ 4.20 GHz: 50 to 70 FPS.
- On an AMD Ryzen 9 5950X @ 4.80 GHz: 60 to 100 FPS.

The new **Improved Screen Management** is faster than the legacy stack in most of the cases. The only case when the legacy Direct Display stack might be faster is in the case of a game or application with only simple ASCII characters and not a lot of things to display.

Here are some compiled benchmark results of both of systems over 150 runs:

Benchmark	Improved Screen Management	Legacy Direct Display
Sprite (place, render and update screen), Sprite size: 155x29	10.0 msec. or 71 FPS	380.0 msec. or 3 FPS
Sprite 200 updates	620.0 msec. or 76 FPS	9830.0 msec. or 20 FPS
Phase 1 - 500 frames. Single board avg load	11.02 msec. per frame or 91 FPS	12.65 msec. per frame or 79 FPS
Phase 2 - 500 frames. Dual board high load	18.18 msec. per frame or 55 FPS	28.34 msec. per frame or 35 FPS
Overall - 1000 frames.	14.60 msec. per frame or 68 FPS	20.49 msec. per frame or 49 FPS

You can use the 2 benchmark scripts to compare on your system:

- benchmark-screen-buffer.py
- benchmark-screen-direct-display.py

The frame buffer system has been tested on the following terminals:

- xterm-256color
- Konsole
- Kitty
- · Alacritty
- GNOME Terminal

Performances are consistent across the different terminals. The only exception is the GNOME Terminal, which is slower than the others (about  $20 \sim 30 \%$  slower).

```
\_init\_(width: int = None, height: int = None)
```

The constructor takes the following (optional) parameters.

### **Parameters**

- width (int) The width of the screen.
- **height** (*int*) The height of the screen.

Setting any of these parameters fixes the screen size regardless of the actual console/terminal resolution. Leaving any of these parameters unset will let the constructor use the actual console/terminal resolution instead.

Please have a look at the examples for more on this topic.

# Example:

```
# Let's assume a terminal resolution of 170(width)x75(height).
screen = Screen()
# Next line display: "Screen width=170 height=75"
print(f"Screen width={screen.width} height={screen.height}")
screen = Screen(50)
# Next line display: "Screen width=50 height=75"
print(f"Screen width={screen.width} height={screen.height}")
screen = Screen(height=50)
# Next line display: "Screen width=170 height=50"

(continues on next page)
```

(continued from previous page)

```
print(f"Screen width={screen.width} height={screen.height}")
screen = Screen(50, 50)
# Next line display: "Screen width=50 height=50"
print(f"Screen width={screen.width} height={screen.height}")
```

# **Methods**

init([width, height])	The constructor takes the following (optional) param-
	eters.
attach(observer)	Attach an observer to this instance.
clear()	This methods clear the screen.
<pre>clear_buffers()</pre>	This methods clear the Screen's buffers (both display and frame buffer).
<pre>clear_frame_buffer()</pre>	This methods clear the frame buffer (but not the display buffer).
<pre>delete([row, column])</pre>	Delete a element on screen.
detach(observer)	Detach an observer from this instance.
<pre>display_at(text[, row, column, clear_eol,])</pre>	Displays text at a given position.
<pre>display_line(*text[, end, file, flush])</pre>	A wrapper to Python's print() builtin function except it will always add an ANSI sequence to clear the end of the line.
<pre>display_sprite(sprite[, filler, file, flush])</pre>	Displays a sprite at the current cursor position.
<pre>display_sprite_at(sprite[, row, column,])</pre>	Displays a sprite at a given position.
<pre>force_render()</pre>	Force the immediate rendering of the display buffer.
<pre>force_update()</pre>	Same as <i>force_render()</i> but also force the immediate screen update.
<pre>get(row, column)</pre>	Get an element from the display buffer at the specified screen coordinates.
<pre>handle_notification(subject[, attribute, value])</pre>	When a Screen object is notified, it set the display buffer to be rendered before the next update.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>place([element, row, column, rendering_pass])</pre>	Place an element on the screen.
render()	Render the display buffer into the frame buffer.
store_screen_position(row, column)	Store the screen position of the object.
trigger_rendering()	Trigger the frame buffer for rendering at the next update.
update()	Update the screen.

### **Attributes**

buffer	The buffer property return a numpy.array as a writable frame buffer.
hcenter	Return the horizontal center of the screen as an int.
height	This property returns the height of the terminal window in number of characters.
need_rendering	This property return True if the display buffer has been updated since the last rendering cycle and the screen needs to re-render the frame buffer.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
vcenter	Return the vertical center of the screen as an int.
width	This property returns the width of the terminal window in number of characters.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### Parameters

**observer** (*PglBaseObject*) – An observer to attach to this object.

### **Returns**

True or False depending on the success of the operation.

## Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## property buffer

The buffer property return a numpy.array as a writable frame buffer.

The buffer is a 2D plane (like a screen) and anything can render in it. However, it is recommended to place objects through Screen.place() and update the screen with Screen.update() (update calls render() if needed and do the actual display).

**Warning:** Everything that is stored in the buffer *must* be printable. Each cell of the frame buffer represent a single character on screen, so you need to take care of that when you write into that buffer or you will corrupt the display. If *need\_rendering* returns True, you need to manually call *render()* before writing anything into the frame buffer. Or else it will be squashed in the next rendering cycle.

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

## clear()

This methods clear the screen.

## clear\_buffers()

This methods clear the Screen's buffers (both display and frame buffer).

Make sure that you really want to clear the buffers before doing so, because this is a slow operation.

Once the buffer is cleared nothing is left in it, you have to reposition (place) everything.

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

#### clear\_frame\_buffer()

This methods clear the frame buffer (but not the display buffer). This means that the next time *update()* is called, rendering will be triggered.

Make sure that you really want to clear the buffers before doing so, because this is a slow operation. It might however be faster than manually update screen cells.

Once the buffer is cleared nothing is left in it, it sets the Screen for a rendering update.

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

#### delete(row=None, column=None)

Delete a element on screen.

It is important to note that if you placed an element that occupies more than 1 cell, you only have to erase that specific position not the entire area.

#### **Parameters**

- **row** (*int*) The row coordinate of the element to delete.
- **column** (*int*) The column coordinate of the element to delete.

## Example:

```
board = Board(size=[20,20])
screen.place(board, 2, 2)
# With this we have placed a board at screen coordinates 2,2 and the board
# will display on screen coordinates from 2,2 to 22,22.
# However, to delete the board we don't need to clean all these cells.
# Just the one where we placed the board:
screen.delete(2, 2)
```

3.6. engine 573

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### **Returns**

True or False depending on the success of the operation.

#### **Return type**

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

Displays text at a given position. If clear\_eol is True, also clear the end of line. Additionally you can specify all the parameters of a regular print() if you need to.

#### **Parameters**

- **text** (*str*) The text to display. Please note that in that case text is a single string.
- **row** (*int*) The row position in the terminal window.
- **column** (*int*) The column position in the terminal window.
- **clear\_eol** (*bool*) If True this clears the end of the line (everything after the last character displayed by that method).
- end (str) end sub string added to the printed text. Usually a carriage return.
- file (stream) -
- flush (bool) –

**Important:** The cursor is only moved for printing the text. It is returned to its previous position after.

**Note:** The position respect the row/column convention across the library. It is reversed compared to the blessed module.

## Example:

(continues on next page)

(continued from previous page)

```
end=''
```

**Note:** This method is part of the **Legacy Direct Display** rendering stack and is incompatible with the methods identified as being part of the **Improved Screen Management** stack.

**display\_line**(\*text, end=\n', file=<colorama.ansitowin32.StreamWrapper object>, flush=False)

A wrapper to Python's print() builtin function except it will always add an ANSI sequence to clear the end of the line. Making it more suitable to use in a user\_update callback.

The reason is that with line with variating length, if you use run() but not clear(), some characters will remain on screen because run(), for performances concerns does not clear the entire screen. It just bring the cursor back to the top left corner of the screen. So if you want to benefit from the increase performances you should use display\_line().

#### **Parameters**

- \*text (str/objects) objects that can serialize to str. The ANSI sequence to clear the end of the line is *always* appended to the text.
- end (str) end sub string added to the printed text. Usually a carriage return.
- file (stream) -
- flush (bool) -

#### Example:

```
screen.display_line(f'This line will display correctly: {elapsed_time}')
# That line will have trailing characters that are not cleared after redraw
# if you don't use clear().
print(f'That one won't: {elapsed_time}')
```

New in version 1.2.0.

**Note:** This method is part of the **Legacy Direct Display** rendering stack and is incompatible with the methods identified as being part of the **Improved Screen Management** stack.

**display\_sprite**(sprite, filler=[0m, file=<colorama.ansitowin32.StreamWrapper object>, flush=False)

Displays a sprite at the current cursor position. If a *Sprixel* is empty, then it's going to be replaced by filler.

### **Parameters**

- **sprite** (*Sprite*) The sprite object to display.
- **filler** (*Sprixe1*) A sprixel object to replace all empty sprixels in sprite.
- file (stream) -
- **flush** print() parameter to flush the stream after printing

## Examples:

```
screen.display_sprite(panda_sprite)
```

3.6. engine 575

New in version 1.3.0.

**Note:** This method is part of the **Legacy Direct Display** rendering stack and is incompatible with the methods identified as being part of the **Improved Screen Management** stack.

Displays a sprite at a given position. If a *Sprixel* is empty, then it's going to be replaced by filler.

#### **Parameters**

- **sprite** (*Sprite*) The sprite object to display.
- **row** (*int*) The row position in the terminal window.
- **column** (*int*) The column position in the terminal window.
- **filler** (*Sprixe1*) A sprixel object to replace all empty sprixels in sprite.
- file (stream) -
- **flush** (bool) print() parameter to flush the stream after printing

## Example:

New in version 1.3.0.

**Note:** This method is part of the **Legacy Direct Display** rendering stack and is incompatible with the methods identified as being part of the **Improved Screen Management** stack.

## force\_render()

Force the immediate rendering of the display buffer.

If you just want to mark the frame buffer for rendering before the next update use  $trigger\_rendering()$  instead.

Example:

```
screen.force_render()
```

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

## force\_update()

Same as *force\_render()* but also force the immediate screen update.

Example:

```
screen.force_update()
```

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

```
get(row: int, column: int)
```

Get an element from the display buffer at the specified screen coordinates.

The element is returned from the display buffer (pre-rendering).

#### **Parameters**

- row (int) The row of the element to get.
- **column** (*int*) The column of the element to get.

## Example:

```
board = Board(size=[20,20])
screen.place(board, 2, 2)
my_board = screen.get(2,2)
```

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

## handle\_notification(subject, attribute=None, value=None)

When a Screen object is notified, it set the display buffer to be rendered before the next update.

## property hcenter

Return the horizontal center of the screen as an int.

Example:

```
screen.place('horizontally centered', 0, screen.hcenter)
```

## property height

This property returns the height of the terminal window in number of characters.

## property need\_rendering

This property return True if the display buffer has been updated since the last rendering cycle and the screen needs to re-render the frame buffer.

It returns False otherwise.

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

3.6. engine 577

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

#### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

place(element=None, row=None, column=None, rendering\_pass=1)

Place an element on the screen.

This method places an element in the screen display buffer. The element is then going to be rendered in the frame buffer before being printed on screen.

The following elements can be placed on screen:

- · All BoardItem derivatives.
- All BoardComplexItem derivatives.
- Board object.
- Text objects.
- Sprite objects.
- Sprixel objects.
- · Regular Python str.
- Any object that expose a render\_to\_buffer() method.

Here is the required signature for render\_to\_buffer:

## render\_to\_buffer(self, buffer, row, column, buffer\_height, buffer\_width)

The buffer parameter will always be a numpy array, row and column are the position to render to. Finally buffer\_height and buffer\_width are the dimension of the buffer.

The buffer is rendered in 2 passes. By default all elements are rendered in pass 1. But if for some reason something needs to be drawn over other elements (like if a dialog/popup is needed for example), the element can be set to be rendered only during the second pass.

### **Parameters**

- **element** (*various*) The element to place.
- row (int) The row to render to.
- **column** (*int*) The column to render to.
- **rendering\_pass** (*int*) When to render the element. You can have any number of rendering passes but you have to be careful of performances. Higher passes render on top of lower passes. You can see the render passes as plane to write on. The default pass is 1.

**Warning:** to be rendered on the second+ pass an element *needs* to implement render\_to\_buffer(...). This excludes all standard types (but not *Text*). Regular Python strings and object that can be print() can still be used in the first pass.

Example:

```
screen.place(my_sprite, 0, 0)
```

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

#### render()

Render the display buffer into the frame buffer.

Example:

```
screen.render()
screen.update()
```

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

## property screen\_column: int

A property to get/set the screen column.

### **Parameters**

**value** (int) – the screen column

## Return type

int

## property screen\_row: int

A property to get/set the screen row.

## **Parameters**

**value** (int) – the screen row

## Return type

int

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

3.6. engine 579

Example:

```
an_object.store_screen_coordinate(3,8)
```

## trigger\_rendering()

Trigger the frame buffer for rendering at the next update.

Example:

```
screen.trigger_rendering()
```

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Direct Display** stack.

## update()

Update the screen. Update means write the frame buffer on screen.

Example:

```
mygame = Game()
sc = core.SpriteCollection.load_json_file('title_screens.spr')
mygame.screen.place(sc['welcome_screen'], 0, 0)
mygame.screen.update()
```

New in version 1.3.0.

**Note:** This method is part of the **Improved Screen Management** rendering stack and is incompatible with the methods identified as being part of the **Legacy Direct Display** stack.

## property vcenter

Return the vertical center of the screen as an int.

Example:

```
screen.place('vertically centered', screen.vcenter, 0)
```

#### property width

This property returns the width of the terminal window in number of characters.

# 3.7 gfx

The gfx (for graphics) sub-module holds all the classes related to the graphics system.

## 3.7.1 core

#### **Animation**

Bases: object

The Animation class is used to give the ability to have more than one model for a BoardItem. A BoardItem can have an animation and all of them that are available to the Game object can be animated through Game.animate\_items(lvl\_number). To benefit from that, BoardItem.animation must be set explicitly. An animation is controlled via the same state system than the Actuators.

The frames are all stored in a list called frames, that you can access through Animation.frames.

#### **Parameters**

- **display\_time** (*float*) The time each frame is displayed
- **auto\_replay** (*bool*) controls the auto replay of the animation, if false once the animation is played it stays on the last frame of the animation.
- **frames** (array[str| *Sprixel* | *Sprite*] | *SpriteCollection*) an array of "frames" (string, sprixel, sprite) or a sprite collection
- animated\_object (BoardItem) The object to animate. This parameter is deprecated. Please use parent instead. It is only kept for backward compatibility. The parent parameter always takes precedence over this one.
- **parent** (*BoardItem*) The parent object. It is also the object to animate. Important: We cannot animate anything else that BoardItems and subclasses.
- **refresh\_screen** (*function*) The callback function that controls the redrawing of the screen. This function reference should come from the main game.

**Important:** When a *SpriteCollection* is used as the *frames* parameter the sprites' names are ordered so the frames are displayed in correct order. This means that 'walk\_1' is going to be displayed before 'walk\_2'. Otherwise SpriteCollection is un-ordered.

## Example

\_\_init\_\_(display\_time=0.05, auto\_replay=True, frames=None, animated\_object=None, refresh\_screen=None, initial\_index=None, parent=None)

3.7. qfx 581

## **Methods**

init([display_time, auto_replay,])	
<pre>add_frame(frame)</pre>	Add a frame to the animation.
<pre>current_frame()</pre>	Return the current frame.
load(data)	Load a serialized Animation object.
next_frame()	Update the parent's model, sprixel or sprite with the next frame of the animation.
pause()	Set the animation state to PAUSED.
play_all()	Play the entire animation once.
remove_frame(index)	Remove a frame from the animation.
reset()	Reset the Animation to the first frame.
search_frame(frame)	Search a frame in the animation.
serialize()	Serialize the Animation object.
start()	Set the animation state to State.RUNNING.
stop()	Set the animation state to STOPPED.

## **Attributes**

dtanimate The time elapsed since the last frame was displayed.
--

## add\_frame(frame)

Add a frame to the animation.

The frame has to be a string (that includes sprites from the Sprite module and squares from the Utils module).

Raise an exception if frame is not a string.

## **Parameters**

**frame** (str|:class:*Sprite* '|:*class*: '*Sprixel*) – The frame to add to the animation.

## Raise

pygamelib.base.PglInvalidTypeException

## Example:

```
item.animation.add_frame(Sprite.ALIEN)
item.animation.add_frame(Sprite.ALIEN_MONSTER)
```

## current\_frame()

Return the current frame.

## Example:

```
item.model = item.animation.current_frame()
```

## property dtanimate

The time elapsed since the last frame was displayed.

### classmethod load(data)

Load a serialized Animation object.

#### **Parameters**

data (dict) – The serialized Animation object.

#### **Returns**

The loaded Animation object.

## Return type

Animation

#### next\_frame()

Update the parent's model, sprixel or sprite with the next frame of the animation.

That method takes care of automatically resetting the animation if the last frame is reached if the state is State.RUNNING.

If the state is PAUSED it still update the parent.model and returning the current frame. It does NOT actually go to next frame.

If parent is not a sub class of *BoardItem* an exception is raised.

#### Raise

PglInvalidTypeException

Example:

```
item.animation.next_frame()
```

**Warning:** If you use Sprites as frames, you need to make sure your Animation is attached to a *BoardComplexItem*.

## pause()

Set the animation state to PAUSED.

Example:

```
item.animation.pause()
```

## play\_all()

Play the entire animation once.

That method plays the entire animation only once, there is no auto replay as it blocks the game (for the moment).

If the the state is PAUSED or STOPPED, the animation does not play and the method return False.

If parent is not a sub class of BoardItem an exception is raised.

If screen\_refresh is not defined or is not a function an exception is raised.

## Raise

PglInvalidTypeException

Example:

```
item.animation.play_all()
```

```
remove_frame(index)
```

Remove a frame from the animation.

That method remove the frame at the specified index and return it if it exists.

If the index is out of bound an exception is raised. If the index is not an int an exception is raised.

#### **Parameters**

**index** (*int*) – The index of the frame to remove.

## Return type

str

## Raise

IndexError, PglInvalidTypeException

## Example:

```
item.animation.remove_frame( item.animation.search_frame(
    Sprite.ALIEN_MONSTER)
)
```

#### reset()

Reset the Animation to the first frame.

Example:

```
item.animation.reset()
```

## search\_frame(frame)

Search a frame in the animation.

That method is returning the index of the first occurrence of "frame".

Raise an exception if frame is not a string.

## **Parameters**

**frame** (str) – The frame to find.

## Return type

int

## Raise

PglInvalidTypeException

## Example:

```
item.animation.remove_frame(
   item.animation.search_frame(Sprite.ALIEN_MONSTER)
)
```

## serialize()

584

Serialize the Animation object.

The *refresh\_screen* callback function is not serialized. Neither is the parent.

## Returns

A dictionary containing the Animation object's data.

### Return type

dict

#### start()

Set the animation state to State.RUNNING.

If the animation state is not State.RUNNING, animation's next\_frame() function return the last frame returned.

Example:

```
item.animation.start()
```

## stop()

Set the animation state to STOPPED.

Example:

```
item.animation.stop()
```

#### **Font**

**class** pygamelib.gfx.core.**Font**(font\_name: str = None, search\_directories: list = None)

Bases: object

New in version 1.3.0.

The Font class allow to load and manipulate a pygamelib "font". A font consist of a sprite collection and a configuration file.

If you want to create your own font, please have a look at the font creation tutorial.

In general the Font class is not used directly but passed to a *Text* object. The text is then rendered using the font.

For performance consideration, it is advised to load the font once and to reuse the object in multiple text objects.

Glyphs are cached (particularly if you change the colors) so it is always beneficial to reuse a font object.

Example:

```
myfont = Font("8bits")
# If you print() mytext, it will use the terminal font and print in cyan.
# But if you Sreen.place() it, it will render using the 8bits sprite font.
mytext = Text("Here's a cool text", fg_color = Color(0,255,255), font=myfont)
```

```
__init__(font\_name: str = None, search\_directories: list = None) <math>\rightarrow None
```

## **Parameters**

- $font_name(str)$  The name of the font to load upon object construction.
- **search\_directories** (*list*) A list of directories to search for the font. The items of the list are strings representing a relative or absolute path.

**Important:** The search directories **must** contain a "fonts" directory, that itself contains the font at the correct format.

**Note:** Version 1.3.0 comes with a pygamelib specific font called 8bits. It also comes with a handfull of fonts imported from the figlet fonts. Please go to http://www.figlet.org/ for more information.

The conversion script will be made available in the Pygamelib Github organization (https://github.com/pygamelib).

## Example:

```
myfont = Font("8bits")
```

#### **Methods**

init([font_name, search_directories])	param font_name  The name of the font to load upon object construction.
<pre>glyph([glyph_name, fg_color, bg_color])</pre>	This method take a glyph name in parameter and returns its representation as a <i>Sprite</i> .
load([font_name])	Load a font by name.

#### **Attributes**

colorable	Returns the "colorability" of the font as specified in the font config file.
glyphs_map	Returns the glyph map of the font as specified in the font config file.
height	Returns the height of the font as specified in the font config file.
horizontal_spacing	Returns the horizontal spacing recommended by the font (as specified in the font config file).
monospace	Returns if the font is monospace as specified in the font config file.
name	Return the name of the font.
scalable	Returns the scalability of the font as specified in the font config file.
vertical_spacing	Returns the vertical spacing recommended by the font (as specified in the font config file).

## property colorable: bool

Returns the "colorability" of the font as specified in the font config file.

# Return type

bool

```
{\tt glyph}(glyph\_name: str = None, fg\_color: \ {\tt Color} = None, bg\_color: \ {\tt Color} = None) \rightarrow Sprite
```

This method take a glyph name in parameter and returns its representation as a *Sprite*.

The glyph name is usually the name of a character (like "a") but it is not mandatory and can be anything. The default glyph (returned when no glyph matches the requested glyph) is called "default" for example.

#### **Parameters**

```
glyph_name (str) – The glyph name
```

586 Chapter 3. Tutorials

#### Returns

A glyphe as a Sprite

## Return type

Sprite

## Example:

```
myfont = Font("8bits")
row = 5
column = 10
for letter in "this is a text":
    glyph = myfont.glyph(letter)
    screen.place(glyph, row, column)
    column += glyph.width + myfont.horizontal_spacing()

# Please note that in real life you would just do this
mytext = Text("this is a text", font=myfont)
screen.place(mytext, row, column)
```

#### property glyphs\_map: dict

Returns the glyph map of the font as specified in the font config file.

## Return type

dict

## property height: int

Returns the height of the font as specified in the font config file.

#### Return type

int

Example:

```
screen.place(text, last_row + myfont.height, first_text_column)
```

### property horizontal\_spacing: int

Returns the horizontal spacing recommended by the font (as specified in the font config file).

As a user of the font class using the Font class to change the look of some text, you will rarely use that value directly (it is directly used by Text.render\_to\_buffer()).

If your goal is to use the Font class to do glyph rendering as you see fit, use the horizontal spacing value to place each glyph relatively to the one on its left or right.

### Return type

int

## **load**( $font\_name: str = None$ ) $\rightarrow$ None

Load a font by name. Once the font is loaded glyphs can be accessed through the glyph() method.

This method is automatically called is the Font constructor is called with a font name.

#### Parameters

**font\_name** (str) – The name of the font to load upon object construction.

Example:

3.7. qfx 587

```
# The 2 following examples do exactly the same thing.
# Example 1: instantiate and load
myfont = Font()
myfont.load("8bits")
# Example 2: load from instantiation
myfont2 = Font("8bits")
# At that point myfont and myfont2 are exactly the same (and there is no
# good justification to instantiate or load the font twice).
```

## property monospace: bool

Returns if the font is monospace as specified in the font config file.

## **Return type**

bool

#### property name: str

Return the name of the font. The name is the string that was used to load the font.

Example:

```
myfont = Font("8bits")
if myfont.name() != "8bits":
    print("Something very wrong just occurred!")
```

### property scalable: bool

Returns the scalability of the font as specified in the font config file.

## Return type

bool

## property vertical\_spacing: int

Returns the vertical spacing recommended by the font (as specified in the font config file).

## Return type

int

Example:

```
screen.place(
   text,
   last_row + myfont.height() + myfont.vertical_spacing(),
   first_text_column
)
```

## **SpriteCollection**

## class pygamelib.gfx.core.SpriteCollection(data=None)

Bases: UserDict

SpriteCollection is a dictionnary class that derives collections. UserDict.

Its main goal is to provide an easy to use object to load and save sprite files. On top of traditional dict method, it provides the following capabilities:

- loading and writing from and to JSON files,
- · data serialization,

• shortcut to add sprites to the dictionnary.

A SpriteCollection is an unordered indexed list of Sprites (i.e a dictionnary).

Sprites are indexed by their names in that collection.

Example:

```
# Load a sprite file
sprites_village1 = SpriteCollection.load_json_file('gfx/village1.spr')
# display the Sprites with their name
for sprite_name in sprites_village1:
    print(f'{sprite_name}:\n{sprites_village1[sprite_name]}')
# Add an empty sprite with name 'house_placeholder'
sprites_village1.add( Sprite(name='house_placeholder') )
# This is absolutely equivalent to:
sprites_village1['house_placeholder'] = Sprite(name='house_placeholder')
# And now rewrite the sprite file with the new placeholder house
sprites_village1.to_json_file('gfx/village1.spr')
```

\_\_init\_\_(data=None)

## **Methods**

init([data])	
add(sprite)	Add a Sprite to the collection.
clear()	
copy()	
fromkeys(iterable[, value])	
<pre>get(k[,d])</pre>	
<pre>items()</pre>	
keys()	
load(data)	Load serialized data and return a new SpriteCollection object.
<pre>load_json_file(filename)</pre>	Load a JSON sprite file into a new SpriteCollection object.
<i>pop</i> (k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised.
<pre>popitem()</pre>	as a 2-tuple; but raise KeyError if D is empty.
rename(old_key, new_key)	Rename a key in the collection.
<pre>serialize() setdefault(k[,d])</pre>	Return a serialized version of the SpriteCollection.
to_json_file(filename)	Export the SpriteCollection object in JSON and writes it on the disk.
update([E, ]**F)	If E present and has a .keys() method, does: for k in E: $D[k] = E[k]$ If E present and lacks .keys() method, does: for $(k, v)$ in E: $D[k] = v$ In either case, this is followed by: for k, v in F.items(): $D[k] = v$
values()	

## add(sprite)

Add a Sprite to the collection. This method is simply a shortcut to the usual dictionnary affectation. The collection requires the name of the Sprite to be the key. That method does that automatically.

## **Parameters**

**sprite** (*Sprite*) – A Sprite object to add to the collection.

**Warning:** As SpriteCollection index Sprites by their name if you change the Sprite's name *after* adding it to the collection you will need to manually update the keys.

## Example:

590 Chapter 3. Tutorials

(continued from previous page)

```
new_village.add( copy.deepcopy( sprites_village1.get('bakery') ) )
              print( new_village['bakery'] )
clear() \rightarrow None. Remove all items from D.
copy()
classmethod fromkeys(iterable, value=None)
get(k[,d]) \rightarrow D[k] if k in D, else d. d defaults to None.
items() \rightarrow a set-like object providing a view on D's items
keys() \rightarrow a set-like object providing a view on D's keys
classmethod load(data)
              Load serialized data and return a new SpriteCollection object.
                                    data (str) – Serialized data that need to be expanded into objects.
                         Returns
                                    A new SpriteCollection object.
                         Return type
                                     SpriteCollection 5 contraction 5 contraction
              Example:
              sprites_village1 = SpriteCollection.load(
                           sprites_village_template.serialize()
              )
static load_json_file(filename)
              Load a JSON sprite file into a new SpriteCollection object.
                         Parameters
                                    filename (str) – The complete path (relative or absolute) to the sprite file.
                         Returns
                                    A new SpriteCollection object.
                         Return type
                                     SpriteCollection
              Example:
              sprites_village1 = SpriteCollection.load_json_file('gfx/village1.spr')
pop(k[,d]) \rightarrow v, remove specified key and return the corresponding value.
              If key is not found, d is returned if given, otherwise KeyError is raised.
```

3.7. qfx 591

This methods also takes care of renaming the Sprite associated with the old key name.

 $popitem() \rightarrow (k, v)$ , remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.

rename(old key, new key)

Rename a key in the collection.

#### **Parameters**

- **old\_key** (str) The key to rename
- **new\_key** (str) The new key name

Example:

```
my_collection.rename('panda', 'panda walk 01')
```

#### serialize()

Return a serialized version of the SpriteCollection. The serialized data can be pass to the JSON module to export.

#### Returns

The SpriteCollection object serialized as a dictionnary.

## Return type

dict

Example:

```
data = sprites_village1.serialize()
```

**setdefault**(k[, d])  $\rightarrow$  D.get(k,d), also set D[k]=d if k not in D

## to\_json\_file(filename)

Export the SpriteCollection object in JSON and writes it on the disk.

#### **Parameters**

**filename** (*str*) – The complete path (relative or absolute) to the sprite file to write.

Example:

```
sprites_village1.to_json_file('gfx/village1.spr')
```

**update**( $[E,]^{**F}$ )  $\rightarrow$  None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

**values()**  $\rightarrow$  an object providing a view on D's values

## **Sprite**

Bases: PglBaseObject

The Sprite object represent a 2D "image" that can be used to represent any complex item. Obviously, a sprite in the pygamelib is not really an image, it is a series of glyphs (or characters) with colors (foreground and background) information.

A Sprite object is a 2D array of Sprixe1.

If you use the climage python module, you can load the generated result into a Sprite through Sprite.load\_from\_ansi\_file().

#### **Parameters**

• **sprixels** (list) – A 2D array of Sprixel.

- **default\_sprixel** (*Sprixel*) A default Sprixel to complete lines that are not long enough. By default, it's an empty Sprixel.
- parent (BoardComplexItem (suggested)) The parent object of this Sprite. If it's left to None, the BoardComplexItem constructor takes ownership of the sprite.
- **size** (*list*) A 2 elements list that represent the width and height ([width, height]) of the Sprite. It is only needed if you create an empty Sprite. If you load from a file or provide an array of sprixels it's obviously calculated automatically. Default value: [2, 2].
- name(str) The name of sprite. If none is given, an UUID will be automatically generated.

## Example:

```
void = Sprixel()
# This represent a panda
panda_sprite = Sprite(
    sprixels=[
        [void, void, void, void, void, void, void],
            Sprixel.black_rect(),
            Sprixel.black_rect(),
            void,
            void,
            void,
            void,
            Sprixel.black_rect(),
            Sprixel.black_rect(),
        ],
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
        ],
        Γ
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.black_rect(),
            Sprixel.black_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.black_rect(),
            Sprixel.black_rect(),
        ],
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.red_rect(),
```

(continues on next page)

(continued from previous page)

```
Sprixel.red_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
        ],
        Γ
            void,
            void,
            Sprixel.black_rect(),
            Sprixel.black_rect(),
            Sprixel.black_rect(),
            Sprixel.black_rect(),
            void,
            void,
        ],
            void,
            void,
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
            Sprixel.black_rect(),
            Sprixel.black_rect(),
        ],
            void,
            void,
            Sprixel.black_rect(),
            Sprixel.black_rect(),
            void,
            void,
            void,
            void,
        ],
    ],
)
```

\_\_init\_\_(sprixels=None, default\_sprixel=None, parent=None, size=[2, 2], name=None)
Like the object class, this class constructor takes no parameter.

594 Chapter 3. Tutorials

## **Methods**

init([sprixels, default_sprixel,])	Like the object class, this class constructor takes no
	parameter.
attach(observer)	Attach an observer to this instance.
<pre>calculate_size()</pre>	Calculate the size of the sprite and update the size variable.
copy()	Returns a (deep) copy of the sprite.
detach(observer)	Detach an observer from this instance.
empty()	Empty the sprite and fill it with default sprixels.
<pre>flip_horizontally()</pre>	Flip the sprite horizontally.
<pre>flip_vertically()</pre>	Flip the sprite vertically (i.e upside/down).
<pre>from_text(text_object)</pre>	Create a Sprite from a <i>Text</i> object.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Create a new Sprite object based on serialized data.
<pre>load_from_ansi_file(filename[, de-</pre>	Load an ANSI encoded file into a Sprite object.
fault_sprixel])	
<pre>modulate(color[, ratio])</pre>	Modulate the sprite colors with the color in parame-
	ters.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the sprite from the display buffer to the frame buffer.
scale([ratio])	Scale a sprite up and down using the nearest neighbor algorithm.
serialize()	Serialize a Sprite into a dictionary.
<pre>set_sprixel(row, column, value)</pre>	Set a specific sprixel in the sprite to the given value.
set_transparency(state)	This method enable transparent background to all the sprite's sprixels.
<pre>sprixel([row, column])</pre>	Return a sprixel at a specific position within the sprite.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
tint(color[, ratio])	Tint a copy of the sprite with the color.

## **Attributes**

height	Property that returns the height of the Sprite.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
width	Property that returns the width of the Sprite.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

#### **Return type**

bool

Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

#### calculate\_size()

Calculate the size of the sprite and update the size variable.

The size is immediately returned.

It is done separately for concerns about performances of doing that everytime the size is requested.

## Return type

list

Example:

## copy()

Returns a (deep) copy of the sprite.

New in version 1.3.0.

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### Parameters

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### empty()

Empty the sprite and fill it with default sprixels.

Example:

```
player_sprite.empty()
```

## flip\_horizontally()

Flip the sprite horizontally.

This method performs a symmetry versus the vertical axis.

At the moment, glyph are not inverted. Only the position of the sprixels.

The flipped sprite is returned (original sprite is not modified).

## Return type

Sprite

Example:

```
reflection_sprite = player_sprite.flip_horizontally()
```

## flip\_vertically()

Flip the sprite vertically (i.e upside/down).

At the moment, glyph are not inverted. Only the position of the sprixels. There is one exception however, as climage uses the "utf8 glyph as a marker, that specific glyph is inverted to "and vice versa.

The flipped sprite is returned (original sprite is not modified).

## **Return type**

Sprite

Example:

```
reflection_sprite = player_sprite.flip_vertically()
```

## classmethod from\_text(text\_object)

Create a Sprite from a Text object.

## **Parameters**

**text\_object** (*Text*) – A text object to transform into Sprite.

Example:

```
# The Text object allow for easy manipulation of text
village_name = base.Text('Khukdale',fg_red, bg_green)
# It can be converted into a Sprite to be displayed on the Board
village_sign = board_items.Tile(sprite=Sprite.from_text(village_name))
# And can be used as formatted text
notifications.push( f'You enter the dreaded village of {village_name}' )
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.

• **value** (*Any*) – The new value of the attribute. This can be None.

## property height

Property that returns the height of the Sprite.

New in version 1.3.0.

Contrary to Sprite.size[1], this property always calls Sprite.calculate\_size() before returning the height.

#### classmethod load(data)

Create a new Sprite object based on serialized data.

New in version 1.3.0.

#### **Parameters**

**data** (*dict*) – Data loaded from a JSON sprite file (deserialized).

#### Return type

Sprite

Example:

```
new_sprite = Sprite.load(json_parsed_data)
```

## classmethod load\_from\_ansi\_file(filename, default sprixel=None)

Load an ANSI encoded file into a Sprite object.

This class method can load a file produced by the climage python module and load it into a Sprite class. Each character is properly decoded into a *Sprixe1* with model, background and foreground colors.

A Sprite is rectangular (at least for the moment), so in case the file is not shaped as a rectangle, this method automatically fills the void with a default sprixel (to make sure all lines in the sprite have the same length). By default, it fills the table with None "values" but you can specify a default sprixel.

The reasons the default sprixel is set to None is because None values in a sprite are not translated into a component in <code>BoardComplexItem</code> (i.e no sub item is generated).

#### **Parameters**

- **filename** (*str*) The path to a file to load.
- **default\_sprixel** (None | *Sprixel*) The default Sprixel to fill a non rectangular shaped sprite.

Example:

```
player_sprite = gfx_core.Sprite.load_from_ansi_file('gfx/models/player.ans')
```

```
modulate(color: Color, ratio: float = 0.5)
```

Modulate the sprite colors with the color in parameters.

New in version 1.3.0.

This method tint all the sprixels of the sprite with the color at the specified ratio. **The original sprite IS modified**.

If you want to keep the original sprite intact consider using tint().

#### **Parameters**

- **color** (*Color*) The modulation color.
- ratio (float) The modulation ratio between 0.0 and 1.0 (default: 0.5)

#### Returns

None

When this method is called, the observers are notified of the change with the pygamelib.core.Sprite.color:modulated event. No arguments are passed along this event.

#### Example:

```
player_sprites = core.SpriteCollection.load_json_file("gfx/player.spr")
# After that, the sprite is quite not "normal" anymore...
player_sprites["normal"].modulate(core.Color(0, 255, 0), 0.3)
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

render\_to\_buffer(buffer, row, column, buffer\_height, buffer\_width)

Render the sprite from the display buffer to the frame buffer.

New in version 1.3.0.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

### scale(ratio=1.0)

Scale a sprite up and down using the nearest neighbor algorithm.

New in version 1.3.0.

### **Parameters**

**ratio** (*float*) – The scaling ration.

#### Returns

An upscaled/downscaled sprite.

## **Return type**

Sprite

3.7. qfx 599

Note: The sprites generated with pgl-converter.py don't scale well yet if the –unicode flag is active

Example:

```
bigger_sprite = original_sprite.scale(2)
```

#### property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

value (int) – the screen column

#### **Return type**

int

## property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

## **Return type**

int

## serialize()

Serialize a Sprite into a dictionary.

New in version 1.3.0.

## **Returns**

The class as a dictionary

## Return type

dict

Example:

```
json.dump( sprite.serialize() )
```

## set\_sprixel(row, column, value)

Set a specific sprixel in the sprite to the given value.

## **Parameters**

- **row** (*int*) The row of the sprite (WARNING: internal sprite coordinates)
- **column** (*int*) The column of the sprite (same warning)
- value (Sprixe1) The sprixel to set at [row, column]

When a sprixel is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprite.sprixel:changed event. A structure is passed as the *value* parameter. This structure has 3 members: row, column and sprixel.

Example:

```
my_sprite.set_sprixel(1, 2, Sprixel("#",fg_color=green))
```

600 Chapter 3. Tutorials

#### set\_transparency(state)

This method enable transparent background to all the sprite's sprixels.

New in version 1.3.0.

#### **Parameters**

**state** – a boolean to enable or disable background transparency

When the transparency is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprite.transparency:changed event. The new transparency state is passed as the *value* parameter.

Example:

```
player_sprite.set_transparency(True)
```

**Warning:** This set background transparency on all sprixels, make sure you are not using background colors as part of your sprite before doing that. It can also be used as a game/rendering mechanic. Just make sure you know what you do. As a reminder, by default, sprixels with no background have transparent background enable.

## sprixel(row=0, column=None)

Return a sprixel at a specific position within the sprite.

If the column is set to None, the whole row is returned.

## **Parameters**

- row (int) The row to access within the sprite.
- **column** (*int*) The column to access within the sprite.

## Returns

**Sprixel** 

Example:

```
# Return the entire line at row index 2
scanline = house_sprite.sprixel(2)
# Return the specific sprixel at sprite internal coordinate 2,3
house_sprixel = house_sprite.sprixel(2, 3)
```

**Warning:** For performance consideration sprixel() does not check the size of its matrix. This method is called many times during rendering and 2 calls to len() in a row are adding up pretty quickly. It checks the boundary of the sprite using the cached size. Make sure it is up to date!

 $store\_screen\_position(row: int, column: int) \rightarrow bool$ 

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

3.7. qfx 601

#### Example:

```
an_object.store_screen_coordinate(3,8)
```

tint(color: Color, ratio: float = 0.5)

Tint a copy of the sprite with the color.

New in version 1.3.0.

This method creates a copy of the sprite and tint all its sprixels with the color at the specified ratio. It then returns the new sprite. **The original sprite is NOT modified**.

## **Parameters**

- **color** (*Color*) The tint color.
- ratio (float) The tint ration between 0.0 and 1.0 (default: 0.5)

### Returns

Sprite

Example:

#### property width

Property that returns the width of the Sprite.

New in version 1.3.0.

Contrary to Sprite.size[0], this property *always* calls Sprite.calculate\_size() before returning the width.

## **Sprixel**

```
class pygamelib.gfx.core.Sprixel(model=", bg_color=None, fg_color=None, is_bg_transparent=None)
    Bases: PglBaseObject
```

A sprixel is the representation of 1 cell of the sprite or one cell on the Board. It is not really a pixel but it is the closest notion we'll have. A Sprixel has a background color, a foreground color and a model. All regular BoardItems can now use a sprixel instead of a model (but simple model is still supported of course).

In the terminal, a sprixel is represented by a single character.

If the background color and the is\_bg\_transparent are None, the sprixel will be automatically configured with transparent background. In that case, as we cannot really achieve transparency in the console, the sprixel will take the background color of whatever it is overlapping.

**Important: BREAKING CHANGE**: in version 1.3.0 background and foreground colors use the new *Color* object. Therefor, Sprixel does not accept ANSI sequences anymore for the bg\_color and fg\_color parameters.

Example:

**\_\_init\_\_**(model=", bg\_color=None, fg\_color=None, is\_bg\_transparent=None)

## **Parameters**

- model(str) The model, it can be any string. Preferrably a single character.
- $bg\_color(Color)$  A Color object to configure the background color.
- **fg\_color** (*Color*) A Color object to configure the foreground color.
- **is\_bg\_transparent** (*bool*) Set the background of the Sprixel to be transparent. It tells the engine to replace the background of the Sprixel by the background color of the overlapped sprixel.

## Methods

init([model, bg_color, fg_color,])	
, 00, 100,]/	param model
	The model, it can be any string. Pre-
	ferrably a single character.
attach(observer)	Attach an observer to this instance.
black_rect()	This class method returns a sprixel that is the equiv-
	alent of pygamelib.assets.graphics.BLACK_RECT.
black_square()	This class method returns a
	sprixel that is the equivalent of
	pygamelib.assets.graphics.BLACK_SQUARE.
blue_rect()	This class method returns a sprixel that is the equiv-
1-7	alent of pygamelib.assets.graphics.BLUE_RECT.
blue_square()	This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE_SQUARE.
copy()	Returns a (deep) copy of the sprixel.
cyan_rect()	This class method returns a sprixel that is the equiv-
	alent of pygamelib.assets.graphics.CYAN_RECT.
cyan_square()	This class method returns a sprixel that is the equiva-
	lent of pygamelib.assets.graphics.CYAN_SQUARE.
detach(observer)	Detach an observer from this instance.
<pre>from_ansi(string[, model])</pre>	Takes an ANSI string, parse it and return a Sprixel.
<pre>green_rect()</pre>	This class method returns a sprixel that is the equiv-
araan aayara()	alent of pygamelib.assets.graphics.GREEN_RECT. This class method returns a
<pre>green_square()</pre>	This class method returns a sprixel that is the equivalent of
	pygamelib.assets.graphics.GREEN_SQUARE.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
, <b>,</b> , , , , , , , , , , , , , , , , ,	observer.
load(data)	Create a new Sprixel object based on serialized data.
<pre>magenta_rect()</pre>	This class method returns a
	sprixel that is the equivalent of
<pre>magenta_square()</pre>	pygamelib.assets.graphics.MAGENTA_RECT. This class method returns a
magerita_square()	This class method returns a sprixel that is the equivalent of
	pygamelib.assets.graphics.MAGENTA_SQUARE.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
red_rect()	This class method returns a sprixel that is the equiv-
	alent of pygamelib.assets.graphics.RED_RECT.
red_square()	This class method returns a sprixel that is the equiv-
1	alent of pygamelib.assets.graphics.RED_SQUARE.
render_to_buffer(buffer, row, column,)	Render the sprixel from the display buffer to the frame buffer.
serialize()	Serialize a Sprixel into a dictionary.
store_screen_position(row, column)	Store the screen position of the object.
white_rect()	This class method returns a sprixel that is the equiv-
·	alent of pygamelib.assets.graphics.WHITE_RECT.
white_square()	This class method returns a
	sprixel that is the equivalent of
	pygamelib.assets.graphics.WHITE_SQUARE.
yellow_rect()	This class method returns a sprixel that is the equiva-
yellow_square()	lent of pygamelib.assets.graphics.YELLOW_RECT.  This class method returns a
	sprixel that is the Chapter valer Tutorials
	pygamelib.assets.graphics.YELLOW_SQUARE.

## **Attributes**

bg_color	A property to get/set the background color of the Sprixel.
fg_color	A property to get/set the foreground color of the Sprixel.
length	Return the true length of the model.
mode1	A property to get/set the model of the Sprixel.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

#### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## property bg\_color

A property to get/set the background color of the Sprixel.

#### **Parameters**

```
value (Color) – The new color
```

When the bg\_color is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprixel.bg\_color:changed event. The new bg\_color is passed as the *value* parameter.

Example:

```
# Access the sprixel's color
sprix.bg_color
# Set the sprixel's background color to some blue
sprix.bg_color = Color(0,128,255)
```

## classmethod black\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLACK\_RECT. The difference is that BLACK\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.black_rect()
```

#### classmethod black\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLACK\_SQUARE. The difference is that BLACK\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.black_square()
```

#### classmethod blue\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE\_RECT. The difference is that BLUE\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.blue_rect()
```

## classmethod blue\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE\_SQUARE. The difference is that BLUE\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.blue_square()
```

#### copy()

Returns a (deep) copy of the sprixel.

New in version 1.3.0.

### classmethod cyan\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.CYAN\_RECT. The difference is that CYAN\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.cyan_rect()
```

## classmethod cyan\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.CYAN\_SQUARE. The difference is that CYAN\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.cyan_square()
```

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

606 Chapter 3. Tutorials

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## property fg\_color

A property to get/set the foreground color of the Sprixel.

#### **Parameters**

```
value (Color) – The new color
```

When the fg\_color is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprixel.fg\_color:changed event. The new fg\_color is passed as the *value* parameter.

Example:

```
# Access the sprixel's color
sprix.fg_color
# Set the sprixel's foreground color to some green
sprix.fg_color = Color(0,255,128)
```

#### static from\_ansi(string, model=")

Takes an ANSI string, parse it and return a Sprixel.

#### **Parameters**

- **string** (*str*) The ANSI string to parse.
- **model** (*str*) The character used to represent the sprixel in the ANSI sequence. Default is ""

Example:

```
new_sprixel = Sprixel.from_ansi(
    "\x1b[48;2;139;22;19m\x1b[38;2;160;26;23m\x1b[0m"])
```

**Warning:** This has mainly be tested with ANSI string generated by climage. If you find any issue, please report it

### classmethod green\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.GREEN\_RECT. The difference is that GREEN\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.green_rect()
```

## classmethod green\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.GREEN\_SQUARE. The difference is that GREEN\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

3.7. qfx 607

```
sprixel = Sprixel.green_square()
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

## property length

Return the true length of the model.

New in version 1.3.0.

With UTF8 and emojis the length of a string as returned by python's len() function is often very wrong. For example, the len("x1b[48;2;139;22;19mx1b[38;2;160;26;23mx1b[0m") returns 39 when it should return 1.

This method returns the actual printing/display size of the sprixel's model.

**Note:** This is a read only value. It is automatically updated when the model is changed.

Example:

```
if sprix.length > 2:
    print(
         f"Warning: that sprixel {sprix} will break the rest of the "
         "board's alignement"
        )
```

## classmethod load(data)

Create a new Sprixel object based on serialized data.

New in version 1.3.0.

#### **Parameters**

**data** (*dict*) – Data loaded from JSON data (deserialized).

# Return type

Sprixel

Example:

```
new_sprite = Sprixel.load(json_parsed_data['default_sprixel'])
```

### classmethod magenta\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.MAGENTA\_RECT. The difference is that MAGENTA\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.magenta_rect()
```

## classmethod magenta\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.MAGENTA\_SQUARE. The difference is that MAGENTA\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.magenta_square()
```

## property model

A property to get/set the model of the Sprixel.

#### **Parameters**

**value** (str) – The new model

When the model is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprixel.model:changed event. The new model is passed as the *value* parameter.

Example:

```
# Get the sprixel's model
sprix.model
# Set the sprixel's model to "@"
sprix.model = "@"
```

**notify**(modifier = None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

## **Parameters**

- **modifier** (*PglBaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# classmethod red\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.RED\_RECT. The difference is that RED\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.red_rect()
```

# classmethod red\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.RED\_SQUARE. The difference is that RED\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.red_square()
```

render\_to\_buffer(buffer, row, column, buffer height, buffer width)

Render the sprixel from the display buffer to the frame buffer.

New in version 1.3.0.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property screen\_column: int

A property to get/set the screen column.

```
Parameters
```

**value** (*int*) – the screen column

# Return type

int

# property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

# Return type

int

# serialize()

Serialize a Sprixel into a dictionary.

New in version 1.3.0.

### Returns

The class as a dictionary

# Return type

dict

## Example:

```
json.dump( sprixel.serialize() )
```

610 Chapter 3. Tutorials

### $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

#### classmethod white\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.WHITE\_RECT. The difference is that WHITE\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.white_rect()
```

# classmethod white\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.WHITE\_SQUARE. The difference is that WHITE\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.white_square()
```

### classmethod yellow\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.YELLOW\_RECT. The difference is that YELLOW\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Note: Yellow is often rendered as brown.

Example:

```
sprixel = Sprixel.yellow_rect()
```

# classmethod yellow\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.YELLOW\_SQUARE. The difference is that YELLOW\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

**Note:** Yellow is often rendered as brown.

Example:

```
sprixel = Sprixel.yellow_square()
```

# Color

**class** pygamelib.gfx.core.**Color**(r=0, g=0, b=0)

Bases: PglBaseObject

New in version 1.3.0.

A color represented by red, green and blue (RGB) components. Values are integer between 0 and 255 (both included).

# **Parameters**

- **r** (*int*) The red component of the color.
- g(int) The green component of the color.
- **b** (*int*) The blue component of the color.

# Example:

```
# color is blue
color = Color(0, 0, 255)
# and now color is pink
color.r = 255
```

```
__init__(r=0, g=0, b=0)
```

Like the object class, this class constructor takes no parameter.

### **Methods**

init([r, g, b])	Like the object class, this class constructor takes no parameter.
attach(observer)	Attach an observer to this instance.
<pre>blend(other_color[, fraction])</pre>	Blend the color with another one.
copy()	Returns a (deep) copy of this color.
detach(observer)	Detach an observer from this instance.
<pre>from_ansi(string)</pre>	Create and return a Color object based on an ANSI color string.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Create a new Color object based on serialized data.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
random()	Create and return a new random color.
randomize()	Set a random value for each of the components of an
	existing color.
serialize()	Serialize a Color into a dictionary.
store_screen_position(row, column)	Store the screen position of the object.

612 Chapter 3. Tutorials

## **Attributes**

b	The b property controls the intensity of the blue color.
g	The g property controls the intensity of the green color.
r	The r property controls the intensity of the red color.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

# Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property b

The b property controls the intensity of the blue color. You can set it to an integer between 0 and 255 (both included).

When this property is set, the observers are notified with the pygamelib.gfx.core.Color.b:changed event. The value of the event is the new value of the property.

### Example:

```
color = Color(128, 128, 0)
print(f"Value for b is {color.b}")
color.b = 255
print(f"New value for b is {color.b}")
```

# **blend**(other\_color, fraction=0.5)

Blend the color with another one. Fraction controls the amount of other\_color that is included (0 means no inclusion at all).

# **Parameters**

- other\_color (*Color*) The color to blend with.
- **fraction** (*float*) The blending modulation factor between 0 and 1.

#### **Returns**

A new Color object that contains the blended color.

### Return type

Color

# Example:

```
a = Color(200, 200, 200)
b = Color(25, 25, 25)
# c is going to be Color(112, 112, 112)
c = a.blend(b, 0.5)
```

# copy()

Returns a (deep) copy of this color.

Example:

```
red = Color(255, 0, 0)
red2 = red.copy()
```

# detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# classmethod from\_ansi(string)

Create and return a Color object based on an ANSI color string.

**Important:** The string must be RGB, i.e '[38;2;RED;GREEN;BLUEm' or '[48;2;RED;GREEN;BLUEm' for foreground and background colors. This method will return None if the color string is not RGB. It is also important to understand that Color is independent from the foreground of background, it is just a color. Therefor '[38;2;89;32;93m' and '[48;2;89;32;93m' will both be parsed into Color(89, 32, 93).

# **Parameters**

**string** (str) – The ANSI color string to convert.

Example:

```
color = Color.from_ansi()
```

## property g

The g property controls the intensity of the green color. You can set it to an integer between 0 and 255 (both included).

When this property is set, the observers are notified with the pygamelib.gfx.core.Color.g:changed event. The value of the event is the new value of the property.

614 Chapter 3. Tutorials

# Example:

```
color = Color(128, 128, 0)
print(f"Value for g is {color.g}")
color.g = 255
print(f"New value for g is {color.g}")
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

#### classmethod load(data)

Create a new Color object based on serialized data.

If data is None, None is returned.

If a color component is missing from data, it is set to 0 (see examples).

Raises an exception if the color components are not integer.

# **Parameters**

```
data (dict) – Data loaded from JSON data (deserialized).
```

#### Returns

Either a Color object or None if data where empty.

#### **Return type**

```
Color | NoneType
```

### Raise

PglInvalidTypeException

## Example:

```
# Loading from parsed JSON data
new_color = Color.load(json_parsed_data['default_sprixel']['fg_color'])
# Loading from incomplete data
color = Color.load({'red':25,'green':35})
# Result in the following Color object:
# Color(25, 35, 0)
```

**notify**(modifier = None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

# **Parameters**

- **modifier** (*PglBaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

#### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### property r

The r property controls the intensity of the red color. You can set it to an integer between 0 and 255 (both included).

When this property is set, the observers are notified with the pygamelib.gfx.core.Color.r:changed event. The value of the event is the new value of the property.

## Example:

```
color = Color(128, 128, 0)
print(f"Value for r is {color.r}")
color.r = 255
print(f"New value for r is {color.r}")
```

### classmethod random()

Create and return a new random color.

#### Return type

Color

# Example:

```
my_color = Color.random()
```

### randomize()

Set a random value for each of the components of an existing color.

When this method is called, the observers are notified with the pygamelib.gfx.core.Color.randomized event. The value of the event is the new color.

# Returns

None

# Return type

NoneType

#### Example:

```
color = Color()
color.randomize()
```

# property screen\_column: int

A property to get/set the screen column.

```
Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize()
     Serialize a Color into a dictionary.
         Returns
             The class as a dictionary
         Return type
             dict
     Example:
     json.dump( color.serialize() )
```

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

# **Parameters**

- **row** (*int*) The row (or y) coordinate.
- column(int) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

# 3.7.2 ui

Warning: The UI module is in alpha version. Some things might change over time.

The ui module contains the classes to easily build full screen Terminal User Interface (TUI) for your games (or applications).

**Important:** It works exclusively with the screen buffer system (place, delete, render, update, etc.). It doesn't work with Screen functions tagged "direct display" like display\_at().

#### Box

Bases: object

A simple object to draw a box on screen.

The Box object's looks and feel is highly configurable through the *UiConfig* object.

```
__init__(width: int, height: int, title: str = ", config: UiConfig = None, fill: bool = False, filling_sprixel: Sprixel = None, title_alignment: Alignment = Alignment.CENTER)
```

The box constructor takes the following parameters.

#### **Parameters**

- width (int) The width of the box.
- **height** (*int*) The height of the box.
- **title** (str | Text) The title of the box (encased in the top border).
- **config** (*UiConfig*) The configuration object.
- **fill** (bool) A tag to tell the box object to fill its inside (or not).
- **filling\_sprixel** (Sprixel) If fill is True, the filling Sprixel is used to fill the inside of the box.
- **title\_alignment** (*int*) The alignment of the title in the top bar. It is a constant from the constant module and can be ALIGN\_LEFT, ALIGN\_RIGHT and ALIGN\_CENTER. THIS FEATURE IS NOT YET IMPLEMENTED.

**Todo:** Implement the title alignment.

### Example:

```
config = UiConfig(bg_color=None)
box = Box(30, 10, 'This is a box')
screen.place(box, 20, 20)
screen.update()
```

# **Methods**

```
__init__(width, height[, title, config, ...]) The box constructor takes the following parameters.

render_to_buffer(buffer, row, column, ...) Render the box from the display buffer to the frame buffer.
```

618 Chapter 3. Tutorials

## **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
height	Get and set the height of the box, only accept int.
title	Get and set the title, only accepts str or Text.
width	Get and set the width of the box, only accept int.

# property config: UiConfig

Get and set the config object (*UiConfig*).

## property height: int

Get and set the height of the box, only accept int.

 $\textbf{render\_to\_buffer}(\textit{buffer}, \textit{row}, \textit{column}, \textit{buffer\_height}, \textit{buffer\_width}) \rightarrow \textit{None}$ 

Render the box from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property title

Get and set the title, only accepts str or Text.

# property width: int

Get and set the width of the box, only accept int.

# **BoxLayout**

Bases: Layout

The box layout lines up child widgets horizontally or vertically. The orientation of the layout is controlled using the *Orientation* constants.

**\_\_init\_\_**(orientation: Orientation | None = None, size\_constraint: SizeConstraint | None = None, parent: Widget | None = None)  $\rightarrow$  None

### **Parameters**

- **orientation** (*Orientation*) The orientation of the layout.
- **size\_constraint** (*SizeConstraint*) The size constraint policy for managed widgets.
- parent (Widget) The parent object, ie: the one in which the GridLayout reside.

Example:

```
parent_widget = Widget(45, 30, config=config)
# Add a GridLayout to a widget
parent_widget.layout = BoxLayout(orientation=Orientation.VERTICAL)
parent_widget.layout.add_widget(LineInput())
```

#### **Methods**

init([orientation, size_constraint, parent])	param orientation  The orientation of the layout.
add_widget(w)	Add a widget to the BoxLayout.
attach(observer)	Attach an observer to this instance.
count()	Returns the amount (the count) of widgets in the BoxLayout.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
widgets()	Returns the list of widgets that are managed by the GridLayout as a set.

# **Attributes**

height	Get the layout's height (including spacing).
orientation	Get and set the layout's orientation.
parent	This property get/set the parent of the Layout (if any).
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size_constraint	Get and set the layout's size constraint policy.
spacing	This property get/set the inter-widgets spacing of the
	Layout.
width	Get the layout's width (including spacing).

# $add\_widget(w: Widget) \rightarrow bool$

Add a widget to the BoxLayout. If the widget is correctly added to the layout this method returns True, otherwise it returns False.

#### **Parameters**

**widget** (class: *Widget*) – The widget to add to the layout.

# Example:

(continues on next page)

(continued from previous page)

```
# Then add children widgets to the parent's layout
# Step 1: create a widget (here just a generic widget)
child_widget1 = Widget(config=UiConfig.instance())
# Step 2: add it to the layout.
parent_widget.layout.add_widget(child_widget1)
# That's it!
```

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

#### Returns

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

```
count() \rightarrow int
```

Returns the amount (the count) of widgets in the BoxLayout.

### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (Pg1Base0bject) – An observer to detach from this object.
```

# Returns

True or False depending on the success of the operation.

# Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

### property height: int

Get the layout's height (including spacing).

Returns an int.

```
notify (modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

### **Parameters**

- **modifier** (*PglBaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

#### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# property orientation: Orientation

Get and set the layout's orientation.

The orientation of the BoxLayout can be changed dynamically and will take effect immediately.

It has to be a Orientation.

## property parent: Widget | None

This property get/set the parent of the Layout (if any).

 $\textbf{render\_to\_buffer}: \textit{numpy.array}, \textit{row}: \textit{int}, \textit{column}: \textit{int}, \textit{buffer\_height}: \textit{int}, \textit{buffer\_width}: \textit{int}) \rightarrow \\ \textbf{None}$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

```
property screen_column: int
     A property to get/set the screen column.
         Parameters
            value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
```

# property size\_constraint: SizeConstraint

Get and set the layout's size constraint policy. It has to be a *SizeConstraint*.

# property spacing: int

This property get/set the inter-widgets spacing of the Layout.

When the spacing is changed the pygamelib.gfx.ui.Layout.spacing:changed event is sent to the observers.

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

```
widgets() \rightarrow Set[Widget]
```

Returns the list of widgets that are managed by the GridLayout as a set. This set is not guaranteed to be ordered!

## property width: int

Get the layout's width (including spacing).

Returns an int.

# ColorPickerDialog

```
class pygamelib.gfx.ui.ColorPickerDialog(title: str = None, config: UiConfig = None)
```

Bases: Dialog

The ColorPickerDialog is a dialog wrapper around the ColorPicker widget.

It serves the same purpose: present a way to easily select a custom color to the user.

It does it as an immediately usable dialog.

The show() method returns the *Color* selected by the user. If the user pressed the ESC key, it returns None.

# **Key mapping:**

- ESC: Exit from the show() method and return None.
- ENTER: Exit from the show() method. Returns the currently selected color.
- UP / DOWN: Increase/decrease the currently selected channel by 1.
- PAGE UP / PAGE DOWN: Increase/decrease the currently selected channel by 10.
- LEFT / RIGHT: Navigate between color channels.

Like all dialogs, it is automatically destroyed on exit of the *show()* method. It is also deleted from the screen buffer.

```
__init__(title: str = None, config: UiConfig = None) \rightarrow None
```

The constructor only take the configuration as parameter.

#### **Parameters**

**config** (*UiConfig*) – The configuration object.

## Example:

```
color_dialog = ColorPickerDialog(conf)
color_dialog.set_color( core.Color(128, 128, 128) )
screen.place(color_dialog, 10, 10)
new_color = color_dialog.show()
```

## **Methods**

init([title, config])	The constructor only take the configuration as parameter.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
set_color(color)	Set the color shown in the dialog.
<pre>set_selection([selection])</pre>	Set the channel selection.
show()	Show the dialog and execute the event loop.

### **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
title	Get / set the dialog title, it needs to be a str.
user_input	Facility to store and retrieve the user input.

# property config

Get and set the config object (*UiConfig*).

**render\_to\_buffer**(buffer, row: int, column: int,  $buffer\_height$ : int,  $buffer\_width$ : int)  $\rightarrow$  None Render the object from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

### **Parameters**

• **buffer** (*numpy.array*) – A screen buffer to render the item into.

- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

```
set\_color(color: Color) \rightarrow None
```

Set the color shown in the dialog.

#### **Parameters**

**color** (*Color*) – The color to edit.

Example:

```
color_dialog.set_color( core.Color(128, 128, 128) )
```

# $set_selection(selection: int = 0)$

Set the channel selection.

## **Parameters**

**selection** (int) – The number of the channel to select (0 = red, 1 = green and 2 = blue).

Example:

```
color_dialog.set_selection(1)
```

## show()

Show the dialog and execute the event loop. Until this method returns, all keyboards event are processed by the local event loop. This is also true if called from the main event loop.

This event loop returns the edited *Color* or None if the user pressed the ESC key.

## Returns

The editor color.

# Return type

Color

Example:

```
new_color = color_dialog.show()
```

### property title

Get / set the dialog title, it needs to be a str.

# property user\_input

Facility to store and retrieve the user input.

# ColorPicker

**class** pygamelib.gfx.ui.**ColorPicker**(*orientation: int = None*, *config:* UiConfig = *None*)

Bases: object

The ColorPicker widget is a simple object to select the red, green and blue components of a color.

It provides the API to set/get each color channel independently as well as the mechanism to select and draw a selection box around one specific channel to give the user a visual cue about what he is modifying.

```
__init__(orientation: int = None, config: UiConfig = None) \rightarrow None
```

The constructor is really simple and takes only 2 arguments.

#### **Parameters**

- **orientation** (*Orientation*) One of the orientation constants.
- **config** (*UiConfig*) The configuration object.

The default orientation is horizontal.

**Warning:** The orientation parameter is ignored for the moment.

## Example:

```
color_picker = ColorPicker(constants.Orientation.HORIZONTAL, conf)
screen.place(color_picker, 10, 10)
screen.update()
```

# **Methods**

init([orientation, config])	The constructor is really simple and takes only 2 ar-
	guments.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.

# **Attributes**

blue	Get / set the blue component of the color, the value needs to be an int between 0 and 255.
color	Get / set the edited color.
green	Get / set the green component of the color, the value needs to be an int between 0 and 255.
red	Get / set the red component of the color, the value needs to be an int between 0 and 255.
selection	Get / set the selection, it needs to be an int between 0 and 2 included.

#### property blue

Get / set the blue component of the color, the value needs to be an int between 0 and 255.

### property color

Get / set the edited color.

The setter automatically forward the individual red, green and blue values to to the proper properties of that widget.

#### **Parameters**

```
value (Color) – The color object.
```

Example:

```
current_color = color_picker.color
current_color.r += 10
color_picker.color = current_color
```

## property green

Get / set the green component of the color, the value needs to be an int between 0 and 255.

# property red

Get / set the red component of the color, the value needs to be an int between 0 and 255.

**render\_to\_buffer**(buffer, row: int, column: int, buffer\_height: int, buffer\_width: int)  $\rightarrow$  None Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

#### property selection

Get / set the selection, it needs to be an int between 0 and 2 included.

0 correspond to the red channel, 1 to the green channel and 2 to the blue channel.

When this widget is rendered a *Box* will be rendered around the specified channel.

### Cursor

Bases: PglBaseObject

New in version 1.4.0.

The Cursor class represent a typing cursor on screen.

**Warning:** The Cursor **need** to be rendered last! For example, in a LineInput widget, the cursor is rendered when the rest of the LineInput is already rendered. The reason being that the Cursor need to know what is already on screen in case it overlap something.

**\_\_init\_\_**(relative\_row: int | None = 0, relative\_column: int | None = 0, blink\_time: float | None = 0.4, sprixel: Sprixel | None = None, parent: Widget | None = None)  $\rightarrow$  None

#### **Parameters**

- **relative\_row** (*int*) The relative row position inside the parent widget.
- **relative\_column** (*int*) The relative column position inside the parent widget.
- **blink\_time** (*float*) The time interval between blinks. Default is 0.4 seconde. If you want to keep the cursor solid (i.e: not blinking) set this parameter to 0.
- **sprixel** (Sprixel) The cursor's sprixel (or representation) as a Sprixel.
- **parent** (*Widget*) The parent object, ie: the one in which the cursor reside. It's optional because you can share a cursor with multiple widgets.

### Example:

```
# Create a cyan cursor rapidly blinking.
custom_cursor = Cursor(
    blink_time=0.1,
    sprixel=Sprixel(
        "|", bg_color=config.input_bg_color, fg_color=Color(0, 255, 255)
    ),
)
line_input = LineInput(
    "Test of the LineInput widget",
    config=UiConfig.instance(),
    minimum_width=6,
    maximum_width=200,
    cursor=custom_cursor,
)
```

#### **Methods**

init([relative_row, relative_column,])	
	param relative_row
	The relative row position inside the
	parent widget.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
	observer.
<pre>lock_position()</pre>	Prevent the cursor's relative position to be changed.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame
	buffer.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
unlock_position()	Authorize the cursor's relative position to be changed.

628 Chapter 3. Tutorials

## **Attributes**

sprixel	Get and set the sprixel of the cursor, it has to be core. Sprixel.
parent	Get and set the parent widget of the cursor, it has to be a <i>Widget</i> or None.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
relative_column	Get and set the relative_column of the cursor, it has to be an int.
relative_row	Get and set the relative_row of the cursor, it has to be an int.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

# Return type

bool

#### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

### Returns

True or False depending on the success of the operation.

## Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# lock\_position()

Prevent the cursor's relative position to be changed. It is useful for objects that manipulate the cursor depending on their content.

# Example:

```
my_cursor = Cursor()
my_lineedit = LineEdit(cursor=my_cursor)
my_cursor.lock_position()
my_lineedit.delete()
my_cursor.unlock_position()
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### property parent: Widget | None

Get and set the parent widget of the cursor, it has to be a Widget or None.

#### property relative\_column: int

Get and set the relative\_column of the cursor, it has to be an int. This value cannot be negative (as it makes no sense in our coordinate referential).

# property relative\_row: int

Get and set the relative\_row of the cursor, it has to be an int. This value cannot be negative (as it makes no sense in our coordinate referential).

```
render_to_buffer(buffer, row, column, buffer_height, buffer_width) → None
```

Render the object from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (*int*) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (*int*) – the screen column

### **Return type**

int

#### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

#### Return type

int

# property sprixel: Sprixel

Get and set the sprixel of the cursor, it has to be core. Sprixel.

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

# $unlock\_position() \rightarrow None$

Authorize the cursor's relative position to be changed. It is useful for objects that manipulate the cursor depending on their content.

Example:

```
my_cursor = Cursor()
my_lineedit = LineEdit(cursor=my_cursor)
my_cursor.lock_position()
```

(continues on next page)

(continued from previous page)

```
my_lineedit.delete()
my_cursor.unlock_position()
```

# **Dialog**

class pygamelib.gfx.ui.Dialog(config=None)

Bases: object

Dialog is a virtual class that can be subclassed to create actual dialogs.

All classes that inherits from Dialog have the following constraints:

- They need to implement a show() method.
- They are automatically rendered on the second pass by the *Screen* object.

It stores the *UiConfig* object and provide a helper attribute for user inputs.

```
__init__(config=None) \rightarrow None
```

This constructor takes only one parameter.

#### **Parameters**

config (UiConfig.) - The config object.

## **Methods**

init([config])	This constructor takes only one parameter.
show()	This is a virtual method, calling it directly will only
	raise a NotImplementedError.

### **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
user_input	Facility to store and retrieve the user input.

# property config

Get and set the config object (UiConfig).

## show()

This is a virtual method, calling it directly will only raise a NotImplementedError. Each class that inheritate Dialog needs to implement show().

# property user\_input

Facility to store and retrieve the user input.

# **FileDialog**

```
class pygamelib.gfx.ui.FileDialog(path: Path = None, width: int = 20, height: int = 10, title: str = 'File dialog', show\_hidden\_files: bool = False, filter: str = '*', config: UiConfig = None)
```

Bases: Dialog

The FileDialog is a file selection dialog: it allow the user to select a file on disk in a relatively easy way. File can then be use for any purpose by the program, like for "save as" or "open" features.

The show() method returns the path selected by the user.

# Key mapping:

- ESC: set the path to None and exit from the *show()* method.
- ENTER: Exit from the *show()* method. Returns the currently selected path.
- BACKSPACE / DELETE: delete a character (both keys have the same result).
- UP / DOWN: Navigate between the files.
- LEFT / RIGHT: Navigate between the directories.
- All other keys input characters in the input field.

In all cases, when the dialog is closed, a path is returned. It can be a file name entered by the user or an existing file. The returned value can also be None if the user pressed ESC. There is no guarantee that the returned path is correct. Please, check it before doing anything with it.

Like all dialogs, it is automatically destroyed on exit of the *show()* method. It is also deleted from the screen buffer.

```
__init__(path: Path = None, width: int = 20, height: int = 10, title: str = 'File\ dialog', show\_hidden\_files: bool = False, filter: str = '*', config: UiConfig = None) \rightarrow None
```

#### **Parameters**

- path (pathlib.Path) The path to start in. This path is made absolute by the constructor.
- width (int) The width of the file dialog widget (in number of screen cells).
- **height** (*int*) The height of the file dialog widget (in number of screen cells).
- **title** (*str*) The title of the dialog (written in the upper border).
- **show\_hidden\_files** (*boo1*) Does the file dialog needs to show the hidden files or not.
- **filter** (*str*) A string that will be used to filter the files shown to the user. For example "\*.spr".
- **config** (*UiConfig*) The configuration object.

# Example:

```
file_dialog = FileDialog( Path("."), 30, 10, "Open file", False, conf)
screen.place(file_dialog, 10, 10)
file = file_dialog.show()
```

# **Methods**

init([path, width, height, title,])	
	param path  The path to start in. This path is made absolute by the
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
show()	Show the dialog and execute the event loop.

# **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
filter	Get/set the current file filter.
path	Get/set the current path.
show_hidden_files	Get/set the property, if True the file dialog is going to show hidden files, and , if False, it won't.
user_input	Facility to store and retrieve the user input.

# property config

Get and set the config object (*UiConfig*).

# property filter

Get/set the current file filter.

## Returns

The dialog's current filter.

## Return type

str

# property path

Get/set the current path.

# Returns

The dialog's current path.

#### **Return type**

pathlib.Path

 $\textbf{render\_to\_buffer}(\textit{buffer}, \textit{row: int, column: int, buffer\_height: int, buffer\_width: int)} \rightarrow \textbf{None}$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.

• width (int) – The total width of the display buffer.

```
show() \rightarrow Path
```

Show the dialog and execute the event loop. Until this method returns, all keyboards event are processed by the local event loop. This is also true if called from the main event loop.

This event loop returns a pathlib.Path object or None if the user pressed the ESC key. The path can point to an existing file or not.

Example:

```
fields = multi_input.show()
```

## property show\_hidden\_files

Get/set the property, if True the file dialog is going to show hidden files, and, if False, it won't.

#### Returns

The dialog's current show\_hidden\_files value.

# Return type

bool

# property user\_input

Facility to store and retrieve the user input.

# **FormLayout**

```
class pygamelib.gfx.ui.FormLayout(parent: Widget | None = None)
    Bases: GridLayout
    __init__(parent: Widget | None = None) → None
```

# **Parameters**

parent (Widget) - The parent object, ie: the one in which the GridLayout reside.

Example:

```
parent_widget = Widget(45, 30, config=config)
# Add a GridLayout to a widget
parent_widget.layout = GridLayout()
# You could also do:
parent_widget.layout = GridLayout(parent_widget)
# But it is not necessary because the Widget.layout property is going to do
# it for you.
```

# **Methods**

init([parent])	
	param parent  The parent object, ie: the one in which the GridLayout reside.
add_row(label, widget)	
<pre>add_widget(widget[, row, column])</pre>	Add a widget to the GridLayout.
attach(observer)	Attach an observer to this instance.
count()	Returns the amount (the count) of widgets in the GridLayout.
<pre>count_columns()</pre>	Returns the number of columns in the GridLayout.
count_rows()	Returns the number of rows in the GridLayout.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	This is an implementation of the notification handling system that is necessary for this class to handle cor- rectly widget's resizing.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
remove_row([row])	
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
widgets()	Returns the list of widgets that are managed by the GridLayout as a set.

# **Attributes**

column_minimum_width	Get and set the column's minimum width of the layout.
height	Get the layout's height (including spacing).
horizontal_spacing	Get and set the horizontal spacing between the widgets in the layout.
parent	This property get/set the parent of the Layout (if any).
row_minimum_height	Get and set the row's minimum width of the layout.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
spacing	Get and set the spacing between the widgets in the layout.
vertical_spacing	Get and set the vertical spacing between the widgets in the layout.
width	Get the layout's width (including spacing).

 $add\_row(label: Text, widget: Widget) \rightarrow int$ 

 $add\_widget(widget: Widget, row: int = None, column: int = None) \rightarrow bool$ 

Add a widget to the GridLayout. If the widget is correctly added to the layout this method returns True, otherwise it returns False.

636 Chapter 3. Tutorials

#### **Parameters**

- widget (class: Widget) The widget to add to the layout.
- **row** (*int*) The row in the layout at which the widget should be added.
- **column** (*int*) The column in the layout at which the widget should be added.

#### Example:

```
parent_widget = Widget(45, 30, config=config)
# Add a GridLayout to a widget
parent_widget.layout = GridLayout()
# Then add children widgets to the parent's layout
# Step 1: create a widget (here just a generic widget)
child_widget1 = Widget(config=UiConfig.instance())
# Step 2: add it to the layout.
parent_widget.layout.add_widget(child_widget1, 2, 3)
# That's it!
```

If either of the row or column (or both) are None, the method will find the first unused cell to put the widget in.

**Important:** If there's no space within the existing grid, a new line will be added. For now, the expansion policy cannot be chosen and it is vertically. In the future an expand policy could be added.

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### Parameters

```
observer (PglBaseObject) – An observer to attach to this object.
```

## Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### property column\_minimum\_width: int

Get and set the column's minimum width of the layout. This will apply to all columns. It has to be an int.

```
count() \rightarrow int
```

Returns the amount (the count) of widgets in the GridLayout.

```
count\_columns() \rightarrow int
```

Returns the number of columns in the GridLayout.

```
count\_rows() \rightarrow int
```

Returns the number of rows in the GridLayout.

# detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

### **Return type**

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# handle\_notification(subject, attribute=None, value=None)

This is an implementation of the notification handling system that is necessary for this class to handle correctly widget's resizing. If you subclass GridLayout and you need to overload that method, please keep in mind that you need to let GridLayout.handle\_notification() do its job if you want to benefit from its capabilities.

In other words, do not forget to call super().handle\_notification(subject, attribute, value)!

# property height: int

Get the layout's height (including spacing).

Returns an int.

This is a read-only property.

# property horizontal\_spacing: int

Get and set the horizontal spacing between the widgets in the layout. It has to be an int.

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

# **Parameters**

- **modifier** (*PglBaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute(str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## property parent: Widget | None

This property get/set the parent of the Layout (if any).

```
remove\_row(row: int = None)
```

 $\textbf{render\_to\_buffer}: \textit{numpy.array}, \textit{row}: \textit{int}, \textit{column}: \textit{int}, \textit{buffer\_height}: \textit{int}, \textit{buffer\_width}: \textit{int}) \rightarrow \\ \textit{None}$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property row\_minimum\_height: int

Get and set the row's minimum width of the layout. This will apply to all row. It has to be an int.

### property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

value (int) - the screen column

#### Return type

int

## property screen\_row: int

A property to get/set the screen row.

## **Parameters**

value (int) – the screen row

# Return type

int

# property spacing: int

Get and set the spacing between the widgets in the layout. This property sets both the horizontal and vertical spacing. It has to be an int.

**Warning:** if you try to retrieve the spacing and the horizontal and vertical spacing are not identical this property returns -1.

### $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

# **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

# property vertical\_spacing: int

Get and set the vertical spacing between the widgets in the layout. It has to be an int.

```
widgets() \rightarrow Set[Widget]
```

Returns the list of widgets that are managed by the GridLayout as a set. This set is not guaranteed to be ordered!

## property width: int

Get the layout's width (including spacing).

Returns an int.

This is a read-only property.

# GridLayout

```
class pygamelib.gfx.ui.GridLayout(parent: Widget | None = None)
```

Bases: Layout

New in version 1.4.0.

The GridLayout is a layout to organize the widgets in a grid (shocking right?). All widgets are managed in a grid, one per cell. Layouts can be nested of course to adapt to your need.

```
__init__(parent: Widget \mid None = None) \rightarrow None
```

#### **Parameters**

parent (Widget) - The parent object, ie: the one in which the GridLayout reside.

Example:

```
parent_widget = Widget(45, 30, config=config)
# Add a GridLayout to a widget
parent_widget.layout = GridLayout()
# You could also do:
parent_widget.layout = GridLayout(parent_widget)
# But it is not necessary because the Widget.layout property is going to do
# it for you.
```

# **Methods**

init([parent])	
	param parent
	The parent object, ie: the one in
	which the GridLayout reside.
<pre>add_widget(widget[, row, column])</pre>	Add a widget to the GridLayout.
attach(observer)	Attach an observer to this instance.
count()	Returns the amount (the count) of widgets in the
	GridLayout.
<pre>count_columns()</pre>	Returns the number of columns in the GridLayout.
<pre>count_rows()</pre>	Returns the number of rows in the GridLayout.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	This is an implementation of the notification handling
	system that is necessary for this class to handle cor-
	rectly widget's resizing.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame
	buffer.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
widgets()	Returns the list of widgets that are managed by the
	GridLayout as a set.

# **Attributes**

column_minimum_width	Get and set the column's minimum width of the layout.
height	Get the layout's height (including spacing).
horizontal_spacing	Get and set the horizontal spacing between the widgets in the layout.
parent	This property get/set the parent of the Layout (if any).
row_minimum_height	Get and set the row's minimum width of the layout.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
spacing	Get and set the spacing between the widgets in the layout.
vertical_spacing	Get and set the vertical spacing between the widgets in the layout.
width	Get the layout's width (including spacing).

 $\textbf{add\_widget}(\textit{widget}: \ \textit{Widget}, \textit{row}: \textit{int} = \textit{None}, \textit{column}: \textit{int} = \textit{None}) \rightarrow \textit{bool}$ 

Add a widget to the GridLayout. If the widget is correctly added to the layout this method returns True, otherwise it returns False.

# **Parameters**

- widget (class: Widget) The widget to add to the layout.
- row (int) The row in the layout at which the widget should be added.
- **column** (*int*) The column in the layout at which the widget should be added.

### Example:

```
parent_widget = Widget(45, 30, config=config)
# Add a GridLayout to a widget
parent_widget.layout = GridLayout()
# Then add children widgets to the parent's layout
# Step 1: create a widget (here just a generic widget)
child_widget1 = Widget(config=UiConfig.instance())
# Step 2: add it to the layout.
parent_widget.layout.add_widget(child_widget1, 2, 3)
# That's it!
```

If either of the row or column (or both) are None, the method will find the first unused cell to put the widget in

**Important:** If there's no space within the existing grid, a new line will be added. For now, the expansion policy cannot be chosen and it is vertically. In the future an expand policy could be added.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property column\_minimum\_width: int

Get and set the column's minimum width of the layout. This will apply to all columns. It has to be an int.

```
\textbf{count()} \to int
```

Returns the amount (the count) of widgets in the GridLayout.

```
count\_columns() \rightarrow int
```

Returns the number of columns in the GridLayout.

```
count\_rows() \rightarrow int
```

Returns the number of rows in the GridLayout.

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## handle\_notification(subject, attribute=None, value=None)

This is an implementation of the notification handling system that is necessary for this class to handle correctly widget's resizing. If you subclass GridLayout and you need to overload that method, please keep in mind that you need to let GridLayout.handle\_notification() do its job if you want to benefit from its capabilities.

In other words, do not forget to call super().handle\_notification(subject, attribute, value)!

### property height: int

Get the layout's height (including spacing).

Returns an int.

This is a read-only property.

# property horizontal\_spacing: int

Get and set the horizontal spacing between the widgets in the layout. It has to be an int.

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

# **Parameters**

- **modifier** (*Pg1Base0bject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## property parent: Widget | None

This property get/set the parent of the Layout (if any).

 $render_{to\_buffer: numpy.array, row: int, column: int, buffer\_height: int, buffer\_width: int) \rightarrow None$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

• **buffer** (*numpy.array*) – A screen buffer to render the item into.

```
• row (int) – The row to render in.
```

- **column** (*int*) The column to render in.
- height (int) The total height of the display buffer.
- width (int) The total width of the display buffer.

# property row\_minimum\_height: int

Get and set the row's minimum width of the layout. This will apply to all row. It has to be an int.

## property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

value(int) – the screen column

## Return type

int

# property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

# **Return type**

int

# property spacing: int

Get and set the spacing between the widgets in the layout. This property sets both the horizontal and vertical spacing. It has to be an int.

**Warning:** if you try to retrieve the spacing and the horizontal and vertical spacing are not identical this property returns -1.

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

### **Parameters**

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

# property vertical\_spacing: int

Get and set the vertical spacing between the widgets in the layout. It has to be an int.

```
widgets() \rightarrow Set[Widget]
```

Returns the list of widgets that are managed by the GridLayout as a set. This set is not guaranteed to be ordered!

#### property width: int

Get the layout's width (including spacing).

Returns an int.

This is a read-only property.

#### GridSelectorDialog

```
class pygamelib.gfx.ui.GridSelectorDialog(choices: list = None, max\_height: int = None, max\_width: int = None, title: str = None, config: UiConfig = None)
```

Bases: Dialog

The GridSelectorDialog is an easy wrapper around the *GridSelector* object. It offers a simple interface for the programmer to present a *GridSelector* to the user and retrieve its selection.

The show() method returns the path selected by the user.

## Key mapping:

- ESC: set the selected item to an empty Sprixel and exit from the show() method.
- ENTER: Exit from the show() method. Returns the currently selected sprixel.
- UP / DOWN / LEFT / RIGHT: Navigate between the files.
- PAGE\_UP / PAGE\_DOWN: Go to previous / next page if there's any.

In all cases, when the dialog is closed, a Sprixel is returned.

Like all dialogs, it is automatically destroyed on exit of the *show()* method. It is also deleted from the screen buffer.

```
__init__(choices: list = None, max_height: int = None, max_width: int = None, title: str = None, config: UiConfig = None) \rightarrow None
```

## **Parameters**

- **choices** (*list*) A list of choices to present to the user. The elements of the list needs to be str or *Sprixel*.
- max\_height (int) The maximum height of the grid selector.
- max\_width (int) The maximum width of the grid selector.
- **config** (*UiConfig*) The configuration object.

## Example:

```
choices = ["@","#","$","%","&","*","[","]"]
grid_dialog = GridSelector(choices, 10, 30, conf)
screen.place(grid_dialog, 10, 10)
grid_dialog.show()
```

init([choices, max_height, max_width,])	
	param choices  A list of choices to present to the user.  The elements of the
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
show()	Show the dialog and execute the event loop.

### **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
grid_selector	Get / set the GridSelector object, it has to be a <i>GridSelector</i> object.
title	Get / set the title of the dialog, it needs to be a str.
user_input	Facility to store and retrieve the user input.

## property config

Get and set the config object (*UiConfig*).

## property grid\_selector

Get / set the GridSelector object, it has to be a *GridSelector* object.

**render\_to\_buffer**(buffer, row: int, column: int, buffer\_height: int, buffer\_width: int)  $\rightarrow$  None Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (*int*) The row to render in.
- column(int) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

### show()

Show the dialog and execute the event loop. Until this method returns, all keyboards event are processed by the local event loop. This is also true if called from the main event loop.

This event loop returns the selected item as a *Sprixel* or None if the user pressed the ESC key.

#### **Returns**

The selected item.

#### **Return type**

Sprixel

Example:

646 Chapter 3. Tutorials

```
item = grid_dialog.show()
```

## property title

Get / set the title of the dialog, it needs to be a str.

## property user\_input

Facility to store and retrieve the user input.

#### **GridSelector**

```
class pygamelib.gfx.ui.GridSelector(choices: list = None, max_height: int = None, max_width: int = None, config: UiConfig = None)
```

Bases: object

The GridSelector is a widget that present a list of elements as a grid to the user.

It also provides the API to draw and manage the cursor and to retrieve the selected element.

**Warning:** In the first version of that widget, only the characters that have a length of 1 are supported. This excludes some UTF8 characters and most of the emojis.

```
\__init\_(choices: list = None, max_height: int = None, max_width: int = None, config: UiConfig = None) \rightarrow None
```

#### **Parameters**

- **choices** (*list*) A list of choices to present to the user. The elements of the list needs to be str or *Sprixel*.
- max\_height (int) The maximum height of the grid selector.
- max\_width (int) The maximum width of the grid selector.
- **config** (*UiConfig*) The configuration object.

## Example:

```
choices = ["@","#","$","%","&","*","[","]"]
grid_selector = GridSelector(choices, 10, 30, conf)
screen.place(grid_selector, 10, 10)
screen.update()
```

init([choices, max_height, max_width,])	param choices  A list of choices to present to the user.  The elements of the
<pre>current_sprixel()</pre>	Returns the currently selected sprixel.
cursor_down()	Move the selection cursor one row down.
cursor_left()	Move the selection cursor one column to the left.
<pre>cursor_right()</pre>	Move the selection cursor one column to the right.
cursor_up()	Move the selection cursor one row up.
<pre>items_per_page()</pre>	Returns the number of items per page.
nb_pages()	Returns the number of pages.
<pre>page_down()</pre>	Change the current page to the one immediately down (current_page + 1).
page_up()	Change the current page to the one immediately up (current_page - 1).
render_to_buffer(buffer, row, column,)	Render the object from the display buffer to the frame buffer.

#### **Attributes**

choices	Get and set the list of choices, it has to be a list of <i>Sprixel</i> or str.
current_choice	Get and set the currently selected item's index (the current choice), it needs to be an int.
current_page	Get and set the current page of the grid selector, it needs to be an int.
max_height	Get and set the maximum height of the grid selector, it needs to be an int.
max_width	Get and set the maximum width of the grid selector, it needs to be an int.

## property choices: int

Get and set the list of choices, it has to be a list of Sprixel or str.

## property current\_choice: int

Get and set the currently selected item's index (the current choice), it needs to be an int. Use <code>current\_sprixel()</code> to get the actual current item.

# property current\_page: int

Get and set the current page of the grid selector, it needs to be an int.

## $current\_sprixel() \rightarrow Sprixel$

Returns the currently selected sprixel.

## $\textbf{cursor\_down()} \rightarrow None$

Move the selection cursor one row down.

Chapter 3. Tutorials

```
cursor\_left() \rightarrow None
     Move the selection cursor one column to the left.
cursor_right() → None
     Move the selection cursor one column to the right.
cursor\_up() \rightarrow None
     Move the selection cursor one row up.
items\_per\_page() \rightarrow int
     Returns the number of items per page.
property max_height: int
     Get and set the maximum height of the grid selector, it needs to be an int.
property max_width: int
```

Get and set the maximum width of the grid selector, it needs to be an int.

```
nb_pages() \rightarrow int
```

Returns the number of pages.

```
page\_down() \rightarrow None
```

Change the current page to the one immediately down (current\_page + 1).

```
page\_up() \rightarrow None
```

Change the current page to the one immediately up (current\_page - 1).

**render\_to\_buffer**(buffer, row: int, column: int, buffer\_height: int, buffer\_width: int)  $\rightarrow$  None Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (*int*) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

#### Layout

```
class pygamelib.gfx.ui.Layout(parent: Widget | None = None)
     Bases: PglBaseObject
```

New in version 1.4.0.

The Layout class is mostly a virtual class. It implements a few properties but all of the methods and properties marked with the virtual method tag need to be implemented in the inheriting object.

By convention, a layout will always use the maximum space available in a rendering buffer. That means that in render\_to\_buffer() it will try to use the entire buffer\_width and buffer\_height while respecting the layout's constraints.

It is therefore the responsibility of the widget or layout that triggers the rendering loop to confine said layout inside its own rendering space. Most of the time it involves passing a different set of argument to render\_to\_buffer() (like the position or size of the buffer).

```
__init__(parent: Widget \mid None = None) \rightarrow None
```

The Layout constructor takes the following parameters.

#### **Parameters**

**parent** – The parent widget. If set, it will set the parent's layout.

### **Methods**

init([parent])	The Layout constructor takes the following parameters.
add_widget(w)	virtual method
attach(observer)	Attach an observer to this instance.
count()	virtual method
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	virtual method
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
widgets()	virtual method

#### **Attributes**

height	virtual attribute
parent	This property get/set the parent of the Layout (if any).
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
spacing	This property get/set the inter-widgets spacing of the
	Layout.
width	virtual attribute

## $add\_widget(w: Widget) \rightarrow bool$

virtual method

This method is purely virtual and needs to be implemented in the inheriting class.

It must allow adding a *Widget* to the layout. Adding can mean different things depending on the type of layout. For example, a *GridLayout* need a row and a column to place the widget. However, these parameters are optional. All layouts should be able to add a *Widget* in the first available space.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

#### Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

```
count() \rightarrow int
```

virtual method

This method is purely virtual and needs to be implemented in the inheriting class.

It must count and returns as an integer the number of widgets in the layout.

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

### Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## property height: int

virtual attribute

This property is purely virtual and needs to be implemented in the inheriting class.

It must return the total height of the Layout.

3.7. qfx 651

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

#### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

### property parent: Widget | None

This property get/set the parent of the Layout (if any).

```
\textbf{render\_to\_buffer}(\textit{buffer: numpy.array, row: int, column: int, buffer\_height: int, buffer\_width: int)} \rightarrow \\ \textbf{None}
```

virtual method

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

This method is purely virtual and needs to be implemented in the inheriting class.

It must render the object from the display buffer to the frame buffer.

By convention, a layout will always use the maximum space available in a rendering buffer. That means that in  $render\_to\_buffer()$  it will try to use the entire buffer\_width and buffer\_height while respecting the layout's constraints.

It is therefore the responsibility of the widget or layout that triggers the rendering loop to confine said layout inside its own rendering space. Most of the time it involves passing a different set of argument to  $render_{to\_buffer}$ () (like the position or size of the buffer).

#### property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

value (int) – the screen column

#### Return type

int

#### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

### **Return type**

int

#### property spacing: int

This property get/set the inter-widgets spacing of the Layout.

When the spacing is changed the pygamelib.gfx.ui.Layout.spacing:changed event is sent to the observers.

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## $widgets() \rightarrow List[Widget]$

virtual method

#### Returns

A list of widgets

## Return type

List[Widget]

This method is purely virtual and needs to be implemented in the inheriting class.

It must returns a list of widgets that are contained in the layout.

### property width: int

virtual attribute

This property is purely virtual and needs to be implemented in the inheriting class.

It must return the total width of the Layout.

# LineInput

Bases: Widget

New in version 1.4.0.

The LineInput widget allows the user to enter and edit a single line of text.

3.7. qfx 653

This widget can be configured to accept either anything printable or only digits.

Contrary to its dialog version that widget does not have any key binding. It provides all the tools to manipulate it but it is the user's (developer) responsibility to bind keys to specific actions.

```
__init___(default: str = ", filter: InputValidator = InputValidator.PRINTABLE_FILTER, width: int = 0, height: int = 0, minimum_width: int = 0, minimum_height: int = 1, maximum_width: int = 20, maximum_height: int = 1, config: UiConfig | None = None, history: History | None = None, cursor: Cursor | None = None) → None
```

#### **Parameters**

- **default** (*str*) The default value in the input field.
- **filter** (*InputValidator*) Sets the type of accepted input. It comes from the constants module.
- width (int) The width of the LineInput.
- **height** (*int*) The height of the LineInput.
- **minimum\_width** (*int*) The minimum width of the LineInput.
- minimum\_height (int) The minimum height of the LineInput.
- maximum\_width (int) The maximum width of the LineInput.
- maximum\_height (int) The maximum height of the LineInput.
- **config** (*UiConfig*) The configuration object.
- **history** (*History*) The history object. If none is provided, the LineInput will use the global instance of History.

### Example:

```
line_input = LineInput(
    "Test of the LineInput widget",
    config=UiConfig.instance(),
    minimum_width=6,
    maximum_width=200,
)
screen.place(line_input, 10, 10)

# Somewhere else in your code you can access the content with LineInput.text
pet_name = line_input.text
```

init([default, filter, width, height,])	
([0010010, 11101, 111011, 11018, 111])	param default
	The default value in the input field.
	The delant value in the input note.
attach(observer)	Attach an observer to this instance.
backspace()	Erase the character immediately before the <i>Cursor</i> .
clear()	Clear everything from the LineInput.
cursor	A read-only property that gives access to the cursor.
delete()	Delete the character immediately under the <i>Cursor</i> .
<pre>detach(observer)</pre>	Detach an observer from this instance.
end()	Set the <i>Cursor</i> 's relative column to the length of the
	content (i.e: put the cursor at the end of the LineEdit).
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
	observer.
home()	Set the <i>Cursor</i> 's relative column to 0 (i.e. put the
	cursor at the beginning of the LineEdit).
<pre>insert_characters([character, position])</pre>	Insert one or more characters at a given position.
length()	Return the length of the content of the LineInput wid-
	get.
move_cursor(direction)	Move the <i>Cursor</i> in the specified direction.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
redo()	If a <i>History</i> is available, redo previously undone
	changes.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
undo()	If a <i>History</i> is available, undo the last changes.

### **Attributes**

bg_color	This property get/set the background color of the widget.
children	This read only property property returns the list of children widgets.
cursor	A read-only property that gives access to the cursor.
filter	Get and set the filter of the line input, it has to be an <i>InputValidator</i> .
focus	This property get/set the focus property.
height	This property get/set the height of the widget.
layout	This property get/set the layout of the widget.
maximum_height	This property get/set the maximum height of the widget.
maximum_width	This property get/set the maximum width of the widget.
minimum_height	This property get/set the minimum height of the widget.
minimum_width	This property get/set the minimum width of the widget.
parent	This property get/set the parent widget of the widget.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size_constraint	This property get/set the size constraints of the widget.
text	Get and set the text of the line input, it has to be a str.
width	This property get/set the width of the widget.
X	This property get/set the x position of the widget on screen.
У	This property get/set the y position of the widget on screen.

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

### Returns

True or False depending on the success of the operation.

## **Return type**

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

656 Chapter 3. Tutorials

#### **backspace()** $\rightarrow$ None

Erase the character immediately before the Cursor.

The modification is reported to the history (i.e: can be undone)

Example:

```
# If the LineInput contains "Hello"
line_input.backspace()
# Will modify it to "Hell" if the cursor is at the end of the line.
```

## property bg\_color: Color

This property get/set the background color of the widget.

When the color is changed the pygamelib.gfx.ui.Widget.bg\_color:changed event is sent to the observers.

## property children: Set[Widget]

This read only property property returns the list of children widgets.

```
clear() \rightarrow None
```

Clear everything from the LineInput. If a *History* is available, it will also clear the history.

#### property cursor: Cursor

A read-only property that gives access to the cursor.

#### $delete() \rightarrow None$

Delete the character immediately under the Cursor.

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

## **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

#### Return type

bool

#### Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

```
\textbf{end()} \rightarrow None
```

Set the *Cursor*'s relative column to the length of the content (i.e: put the cursor at the end of the LineEdit).

# property filter: InputValidator

Get and set the filter of the line input, it has to be an InputValidator.

# property focus: bool

This property get/set the focus property. It is a boolean.

At the moment it is mostly an informational property, to tell the programmer and potentially the Widget user (i.e: the class inheriting from Widget) about its own state.

3.7. qfx 657

#### handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBase0bject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

### property height: int

This property get/set the height of the widget. This property respects the boundaries set by the *maxi-mum\_height* and *minimum\_height* properties.

When the height is changed the pygamelib.gfx.ui.Widget.resizeEvent:height event is sent to the observers.

#### **home()** $\rightarrow$ None

Set the Cursor's relative column to 0 (i.e: put the cursor at the beginning of the LineEdit).

```
insert\_characters(character: str = None, position: int | None = None) \rightarrow None
```

Insert one or more characters at a given position.

If no position is given, the characters are inserted at the cursor's position.

## **Parameters**

- **character** (*str*) The character to insert.
- $\bullet$   $\,$  position ( int) The position at which a character must be inserted.

### Example:

```
# Insert a character at position 3 of the LineInput widget (if it exists,
# otherwise insert at the end)
line_input.insert_character("a", 3)
# Insert a character at the cursor's position
line_input.insert_character("a")
```

### property layout: Layout

This property get/set the layout of the widget. You can then add sub widgets to the layout.

This must be a Layout or a class that inherits from it.

When the layout is changed the pygamelib.gfx.ui.Widget.layout:changed event is sent to the observers.

#### $length() \rightarrow int$

658

Return the length of the content of the LineInput widget.

### property maximum\_height: int

This property get/set the maximum height of the widget. This property is used when changing the size constraints and the height property.

#### property maximum\_width: int

This property get/set the maximum width of the widget. This property is used when changing the size constraints and the width property.

#### property minimum\_height: int

This property get/set the minimum height of the widget. This property is used when changing the size constraints and the height property.

### property minimum\_width: int

This property get/set the minimum width of the widget. This property is used when changing the size constraints and the width property.

```
move_cursor(direction: Direction) \rightarrow None
```

Move the *Cursor* in the specified direction.

#### **Parameters**

**direction** (*Direction*) – The direction to move the cursor to.

Example:

```
line_edit.move_cursor(constants.Direction.LEFT)
```

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

#### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

#### property parent: Widget | None

This property get/set the parent widget of the widget.

```
redo() \rightarrow None
```

If a *History* is available, redo previously undone changes.

```
\textbf{render\_to\_buffer}(\textit{buffer}, \textit{row}, \textit{column}, \textit{buffer\_height}, \textit{buffer\_width}) \rightarrow \textit{None}
```

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

## **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.

3.7. qfx 659

• width (int) – The total width of the display buffer.

#### property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (int) – the screen column

## **Return type**

int

#### property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (*int*) – the screen row

## Return type

int

#### property size\_constraint: SizeConstraint

This property get/set the size constraints of the widget. Changing the size constraints immediately resize the widget.

#### $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

#### property text: str

Get and set the text of the line input, it has to be a str. When setting this property tries to set the LineInput.width to the size of the content if the content's length is greater than the width.

Obviously the width is constrained by the maximum\_width property.

## **undo()** $\rightarrow$ None

If a *History* is available, undo the last changes.

### property width: int

This property get/set the width of the widget. This property respects the boundaries set by the *maxi-mum\_width* and *minimum\_width* properties.

When the width is changed the pygamelib.gfx.ui.Widget.resizeEvent:width event is sent to the observers.

# property x: int

This property get/set the x position of the widget on screen. Since a Widget is a *PglBaseObject* this is an alias for the *screen\_column* property.

#### property y: int

This property get/set the y position of the widget on screen. Since a Widget is a *PglBaseObject* this is an alias for the *screen\_row* property.

## LineInputDialog

Bases: Dialog

The LineInputDialog allows the user to enter and edit a single line of text.

This dialog can be configured to accept either anything printable or only digits.

The show() method returns the user input.

## **Key mapping:**

- ESC: set the user input to "" and exit from the show() method.
- ENTER: Exit from the show() method. Returns the user input.
- BACKSPACE / DELETE: delete a character (both keys have the same result)
- All other keys input characters in the input field.

In all cases, when the dialog is closed, the user input is returned.

Like all dialogs, it is automatically destroyed on exit of the *show()* method. It is also deleted from the screen buffer.

```
\__init\_(title=None, label='Input a value:', default='', filter=InputValidator.PRINTABLE_FILTER, config=None) \rightarrow None
```

#### **Parameters**

- **title** (*str*) The short title of the dialog. Only used when the dialog is not borderless.
- label (str | base.Text) The label of the dialog (usually a one line instruction).
- $\mathbf{default}(str)$  The default value in the input field.
- **filter** (*InputValidator*) Sets the type of accepted input. It comes from the constants module.
- **config** (*UiConfig*) The configuration object.

# Example:

```
line_input = LineInputDialog(
    "Name the pet",
    "Enter the name of your pet:",
    "Stupido",
)
screen.place(line_input, 10, 10)
pet_name = line_input.show()
```

init([title, label, default, filter, config])	
	param title  The short title of the dialog. Only used when the dialog is not
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
show()	Show the dialog and execute the event loop.

### **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
label	Get and set the label of the dialog, it has to be a str or base. Text.
title	Get and set the title of the dialog, it has to be a str.
user_input	Facility to store and retrieve the user input.

## property config

Get and set the config object (*UiConfig*).

### property label: Text

Get and set the label of the dialog, it has to be a str or base. Text.

 $render_to\_buffer$  (buffer, row, column, buffer\_height, buffer\_width)  $\rightarrow$  None

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (*int*) The row to render in.
- column(int) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

### show()

Show the dialog and execute the event loop. Until this method returns, all keyboards event are processed by the local event loop. This is also true if called from the main event loop.

This event loop returns the either "" or what is displayed in the input field.

Example:

```
value = line_input.show()
```

## property title: str

Get and set the title of the dialog, it has to be a str.

#### property user\_input

Facility to store and retrieve the user input.

#### Menu

Bases: object

The Menu object consists of a list of other Menu objects and/or MenuAction objects.

It has a title that is used in a MenuBar and the list of its entries is displayed when the menu is expanded.

A Menu object can contains an arbitrary number of entries with an arbitrary depth of submenus.

**\_\_init\_\_**(*title*: Text = *None*, *entries*: *list* = *None*, *padding*: int = 1, *config*: UiConfig = *None*)  $\rightarrow$  None The constructor takes the following parameters.

#### **Parameters**

- title (str | Text) The title of the action (i.e. its label)
- entries (list) A list of MenuAction or other Menu objects.
- **padding** (*int*) The horizontal padding, i.e the number of space characters added to the left and right of the title.
- **config** (*UiConfig*) The configuration object.

## Example

```
menubar = MenuBar(config=UiConfig.instance(game=Game.instance()))
file_menu = Menu(
    "File",
    [
         MenuAction("Open", open_file),
         MenuAction("Save", save_file),
         MenuAction("Save as", save_file_as),
         MenuAction("Quit", exit_application),
    ]
)
menubar.add_entry( file_menu )
menubar.add_entry( MenuAction("Help", display_help) )
screen.place(menubar, 0, 0)
screen.update()
```

init([title, entries, padding, config])	The constructor takes the following parameters.
activate()	Activates the menu.
add_entry(entry)	Add an entry to the menu.
collapse()	Collapse the menu.
current_entry()	Return the currently selected menu entry.
expand()	Expand the menu.
<pre>menu_width()</pre>	Calculate and return the maximum width of the menu
	based on the widest element.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame
	buffer.
select_next()	Select the next entry in the menu.
<pre>select_previous()</pre>	Select the previous entry in the menu.
title_width()	Return the actual width of the menu title.

### **Attributes**

config	Get / set the config of the Menu, it needs to be a <i>UiConfig</i> .
entries	Get / set the entries of the Menu, it needs to be a list of MenuAction objects.
padding	Get / set the padding before and after the menu, it needs to be an int.
selected	Get / set the selected status of the Menu, it needs to be a boolean.
title	Get / set the title of the Menu, it needs to be a <i>Text</i> object or a python str.

### activate()

Activates the menu. This method contains its own event loop a bit like the show() methods of Dialogs. It expands the menu if it wasn't already the case and listen to keyboard key strokes.

- SPACE or ENTER activates (i.e execute) menu actions.
- DOWN select the next entry.
- UP select the previous entry.
- ESC or LEFT close the menu.
- RIGHT activate (i.e expand) a submenu.

# Example:

menu.activate()

# add\_entry(entry)

Add an entry to the menu. An entry can be a *MenuAction* or a *Menu*. Entries are displayed in the order of there additions from left to right.

664 Chapter 3. Tutorials

**Important:** The config of the entry is overwritten by the config of the Menu. That is why it's not mandatory for *Menu* and *MenuAction*.

#### **Parameters**

**entry** (MenuAction | Menu) – The entry to add.

Example:

```
menu.add_entry( Menu('File') )
menu.add_entry( MenuAction('Exit', quit_application) )
```

## collapse()

Collapse the menu. A menu is automatically collapsed after activation.

Example:

```
file_menu.collapse()
```

#### property config

Get / set the config of the Menu, it needs to be a UiConfig.

## current\_entry()

Return the currently selected menu entry.

It can be either a Menu object or a MenuAction object.

#### property entries: list

Get / set the entries of the Menu, it needs to be a list of MenuAction objects.

#### expand()

Expand the menu. A menu is automatically expanded when activated.

Example:

```
file_menu.expand()
```

## $menu\_width() \rightarrow int$

Calculate and return the maximum width of the menu based on the widest element. This includes the padding.

## Returns

the menu width.

## Return type

int

# property padding

Get / set the padding before and after the menu, it needs to be an int.

The padding is only used when the menu is nested into another menu.

 $\textbf{render\_to\_buffer}(\textit{buffer}, \textit{row: int, column: int, buffer\_height: int, buffer\_width: int)} \rightarrow \textit{None}$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (int) The row to render in.
- **column** (*int*) The column to render in.
- height (int) The total height of the display buffer.
- width (int) The total width of the display buffer.

#### select\_next()

Select the next entry in the menu.

The selected entry is rendered differently to give a visual feedback to the user. Please see the *UiConfig* class for the styling option available to the Menu object.

Example:

```
menu.select_next()
```

#### select\_previous()

Select the previous entry in the menu.

The selected entry is rendered differently to give a visual feedback to the user. Please see the *UiConfig* class for the styling option available to the Menu object.

Example:

```
menu.select_previous()
```

### property selected: bool

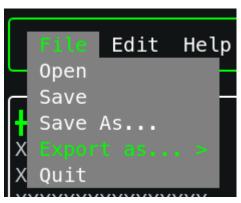
Get / set the selected status of the Menu, it needs to be a boolean.

This changes the representation (way it's drawn) of the menu entry.

## property title: Text

Get / set the title of the Menu, it needs to be a Text object or a python str.

The title is used in the MenuBar. In the following image, the title of the expanded menu is "File".



## $title\_width() \rightarrow int$

Return the actual width of the menu title. This takes into account the padding.

Example:

```
menu.title_width()
```

666 Chapter 3. Tutorials

### **MenuAction**

Bases: object

A menu action is a menu entry that executes a callback when activated. Usually a Menuaction represents an action from the user interface like open file, save, quit, etc.

Therefor a MenuAction is fairly simple, at its simplest it has a title and a callable reference to a function.

An action cannot be used by itself but can be added to a MenuBar or a Menu.

Like everything in the UI module, MenuAction are styled through a *UiConfig* object. Unlike the other classes of that module however, the configuration object is not mandatory when instanciating this class. The reason is that the *MenuBar* object impose the configuration to its managed *MenuAction* and *Menu*.

```
__init__(title: Text = None, action=None, parameter=None, padding: int = 1, config: UiConfig = None) →
None
```

The constructor takes the following parameters.

#### **Parameters**

- **title** (str | *Text*) The title of the action (i.e: its label)
- **action** (*callable*) A reference to a callable function that is going to be executed when the action is activated. If set to None, nothing will happen when the action is activated.
- parameter (Any) A parameter that is passed to the callback action if not None.
- **padding** (*int*) The horizontal padding, i.e the number of space characters added to the left and right of the action.
- **config** (*UiConfig*) The configuration object.

# Example

3.7. qfx 667

init([title, action, parameter,])	The constructor takes the following parameters.
<pre>activate()</pre>	Execute and return the result of the callback.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
title_width()	Return the actual width of the action's title.

#### **Attributes**

action	Get / set the action's callback, it needs to be a callable.
config	Get / set the config of the MenuAction, it needs to be a <i>UiConfig</i> .
padding	Get / set the padding before and after the menu action, it needs to be an int.
selected	Get / set the selected of the MenuAction, it needs to be a boolean.
title	Get / set the title of the action, it needs to be a str or a <i>Text</i> object.

# property action

Get / set the action's callback, it needs to be a callable.

#### activate()

Execute and return the result of the callback.

Example:

```
file_save_action.activate()
```

### property config

Get / set the config of the MenuAction, it needs to be a UiConfig.

## property padding

Get / set the padding before and after the menu action, it needs to be an int.

 $\textbf{render\_to\_buffer}(\textit{buffer}, \textit{row: int, column: int, buffer\_height: int, buffer\_width: int)} \rightarrow \textit{None}$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (*int*) The row to render in.
- column (int) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

# property selected: bool

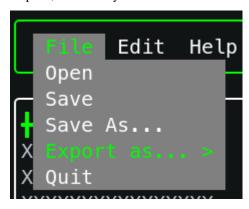
Get / set the selected of the MenuAction, it needs to be a boolean.

This changes the representation (way it's drawn) of the menu entry.

#### property title: Text

Get / set the title of the action, it needs to be a str or a *Text* object.

The title is used in the *Menu*. In the following image, the title of the first action in the expanded menu is "Open", followed by "Save".



#### title\_width()

Return the actual width of the action's title. This takes into account the padding.

Example:

```
menu_action.title_width()
```

### **MenuBar**

class pygamelib.gfx.ui.MenuBar(entries: list = None, spacing: int = 2, config: UiConfig = None)
Bases: object

The MenuBar widget is exactly that: an horizontal bar that can hold Menu or MenuAction objects.

Contrary to these 2 classes, MenuBar does *not* have an activate() method. The reason is that the menubar cannot block rendering with its own event loop as it is supposed to be showned at all times. So the management of interactions are left to the programmer to implement.

A typical implementation would look like this:

Example:

3.7. qfx 669

(continued from previous page)

```
MenuAction("Quit", exit_application),
    ]
menubar.add_entry( file_menu )
menubar.add_entry( MenuAction("Help", display_help) )
# Place the menubar on screen
screen.place(menubar, 0, 0)
screen.update()
# Then, somewhere in an event loop, manage the inputs for example in the user
# update function
def user_update(game, inkey, elapsed_time):
    if inkey == engine.key.DOWN:
        if menubar.current_entry() is not None:
            menubar.current_entry().activate()
    elif inkey == engine.key.LEFT:
        menubar.select_previous()
    elif inkey == engine.key.RIGHT:
        menubar.select_next()
    elif inkey.name == "KEY_ENTER":
        if menubar.current_entry() is not None:
            menubar.current_entry().activate()
    elif inkey.name == "KEY_ESCAPE":
        menubar.close()
```

**\_\_init\_\_**(entries: list = None, spacing: int = 2, config: UiConfig = None)  $\rightarrow$  None

The constructor takes the following parameters.

#### **Parameters**

- entries (list) A list of MenuAction or Menu objects.
- **spacing** The horizontal spacing between entries.
- **config** (*UiConfig*) The configuration object.

# **Methods**

init([entries, spacing, config])	The constructor takes the following parameters.
<pre>add_entry(entry)</pre>	Add an entry to the menu bar.
close()	Close and unselect menu entries/submenu.
<pre>current_entry()</pre>	Return the currently selected menu entry.
length()	Returns the total length of the menubar.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
select_next()	Select the next element in the menubar.
select_previous()	Select the previous element in the menubar.

670 Chapter 3. Tutorials

### **Attributes**

config	Get / set the config of the MenuBar, it needs to be a <i>UiConfig</i> .
current_index	Get / set the currently selected menu entry, it needs to be an int.
entries	Get / set the entries of the MenuBar, it needs to be a list of MenuAction or Menu objects.
spacing	Get / set the spacing between menu entries, it needs to be an int.

#### add\_entry(entry)

Add an entry to the menu bar. An entry can be a *MenuAction* or a *Menu*. Entries are displayed in the order of there additions from left to right.

**Important:** The config of the entry is overwritten by the config of the MenuBar. That is why it's not mandatory for *Menu* and *MenuAction*.

#### **Parameters**

**entry** (MenuAction | Menu) – The entry to add.

## Example:

```
menubar.add_entry( Menu('File') )
menubar.add_entry( MenuAction('Exit', quit_application) )
```

#### close()

Close and unselect menu entries/submenu.

Please call that method when the menu bar loses focus.

## property config

Get / set the config of the MenuBar, it needs to be a UiConfig.

**Important:** The MenuBar's config is imposed on the managed items (Menu and MenuAction).

# current\_entry()

Return the currently selected menu entry.

It can be either a Menu object or a MenuAction object.

#### property current\_index

Get / set the currently selected menu entry, it needs to be an int. When setting the current\_index, if the previous index was corresponding to a selected entry, said entry is first unselected.

## property entries: list

Get / set the entries of the MenuBar, it needs to be a list of MenuAction or Menu objects.

#### $length() \rightarrow int$

Returns the total length of the menubar. This is computed everytime the method is called and it includes the spacing.

**render\_to\_buffer**(buffer, row: int, column: int,  $buffer\_height$ : int,  $buffer\_width$ : int)  $\rightarrow$  None Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

# select\_next()

Select the next element in the menubar.

Example

```
if user_input.name == 'KEY_RIGHT':
    menubar.select_next()
```

### select\_previous()

Select the previous element in the menubar.

Example

```
if user_input.name == 'KEY_RIGHT':
    menubar.select_previous()
```

## property spacing

Get / set the spacing between menu entries, it needs to be an int.

#### MessageDialog

```
class pygamelib.gfx.ui.MessageDialog(data: list = None, width: int = 20, height: int = None, adaptive_height: bool = True, alignment: Alignment = None, title: <math>str = None, config: UiConfig = None)
```

Bases: Dialog

The message dialog is a popup that can display multiple lines of text.

It supports formatted text (base.Text), python strings, <code>pygamelib.gfx.core.Sprixel</code>, <code>core.Sprite</code> and more generally anything that can be rendered on screen (i.e: posess a render\_to\_buffer(self, buffer , row, column, buffer\_height, buffer\_width) method).

Each line can be aligned separately using one of the Alignment constants. Please see add\_line().

It also implements the *show()* virtual method of *Dialog*. This method is blocking and has its own event loop. It does not return anything.

ESC or ENTER close the dialog.

For the moment, the full message dialog needs to be displayed on screen. There is no pagination, but it is going to be implemented in a future release.

As all dialogs it also has a *user\_input* property that reflects the user input. It is not used here however.

Like all dialogs, it is automatically destroyed on exit of the *show()* method. It is also deleted from the screen buffer.

**Todo:** Implements pagination.

```
__init__(data: list = None, width: int = 20, height: int = None, adaptive_height: bool = True, alignment: Alignment = None, title: <math>str = None, config: UiConfig = None) \rightarrow None
```

#### **Parameters**

- data (list) A list of data to display inside the MessageDialog. Elements in the list
  can contain various data types like base. Text, python strings, pygamelib.gfx.core.
  Sprixel, core.Sprite
- width (int) The width of the message dialog widget (in number of screen cells).
- **height** (*int*) The height of the message dialog widget (in number of screen cells).
- **adaptive\_height** (*boo1*) If True, the dialog height will be automatically adapted to match the content size.
- alignment (Alignment) The alignment to apply to the data parameter. Please use the Alignment constants. The default value is pygamelib.constants.Alignment.LEFT
- **title** (*str*) The short title of the dialog. Only used when the dialog is not borderless.
- **config** (*UiConfig*) The configuration object.

#### Example:

init([data, width, height,])	
	param data
	A list of data to display inside the
	MessageDialog. Elements in
add_line(data[, alignment])	Add a line to the message dialog.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
show()	Show the dialog and execute the event loop.

### **Attributes**

config height	Get and set the config object ( <i>UiConfig</i> ). Get and set the height of the message dialog, it has to be an int.
title	Get and set the title of the dialog, it has to be a str.
user_input	Facility to store and retrieve the user input.

 $add\_line(data, alignment: Alignment = Alignment.LEFT) \rightarrow None$ 

Add a line to the message dialog.

The line can be any type of data that can be rendered on screen. This means that any object that expose a render\_to\_buffer(self, buffer, row, column, buffer\_height, buffer\_width) method can be added as a "line". Python strings are also obviously accepted.

Here is a non-exhaustive list of supported types:

- Text,
- python strings (str),
- Sprixel,
- Sprite,
- · most board items,
- etc.

## **Parameters**

- data (various) The data to add to the message dialog.
- alignment (Alignment) The alignment of the line to add.

## Example:

```
msg.add_line(
   base.Text(
     'This is centered and very red',
     core.Color(255,0,0),
   ),
```

(continues on next page)

(continued from previous page)

```
constants.Alignment.CENTER,
```

### property config

Get and set the config object (*UiConfig*).

#### property height: int

Get and set the height of the message dialog, it has to be an int.

```
render_to\_buffer(buffer, row, column, buffer\_height, buffer\_width) \rightarrow None
```

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

### $show() \rightarrow None$

Show the dialog and execute the event loop. Until this method returns, all keyboards event are processed by the local event loop. This is also true if called from the main event loop.

This event loop returns the key pressed.

Example:

```
key_pressed = msg.show()
if key_pressed.name = 'KEY_ENTER':
    // do something
else:
    print('Good bye')
```

#### property title: str

Get and set the title of the dialog, it has to be a str.

#### property user\_input

Facility to store and retrieve the user input.

# MultiLineInputDialog

Bases: Dialog

The MultiLineInputDialog behave essentially like the *LineInputDialog* but is more configurable to allow the user to enter and edit a multiple lines of text.

Each field of this dialog can be individually configured to accept either anything printable or only digits.

The show() method returns the user input.

3.7. qfx 675

#### **Key mapping:**

- ESC: set the user input to "" and exit from the show() method.
- ENTER: Exit from the show() method. Returns the user input.
- BACKSPACE / DELETE: delete a character (both keys have the same result).
- TAB: cycle through the fields.
- All other keys input characters in the input field.

In all cases, when the dialog is closed, the user input is returned.

Like all dialogs, it is automatically destroyed on exit of the *show()* method. It is also deleted from the screen buffer.

```
\__init\_(fields=[{'default': ", 'filter': InputValidator.PRINTABLE_FILTER, 'label': 'Input a value:'}], title: str = None, config=None) \rightarrow None
```

#### **Parameters**

- **fields** (*list*) A list of dictionnary that represent the fields to present to the user. Please see bellow for a description of the dictionnary.
- **title** (*str*) The short title of the dialog. Only used when the dialog is not borderless.
- **config** (*UiConfig*) The configuration object.

The fields needs to be a list that contains dictionaries. Each of the dictionaries needs to contain 3 fields:

- "label": A one line instruction displayed over the field. This is a string.
- "default": A string that is going to pre-fill the input field.
- "filter": A filter to configure the acceptable inputs.

The filters needs to be a InputValidator.

## Example:

```
fields = \Gamma
    {
        "label": "Enter the height of the new sprite:",
        "default": "",
        "filter": constants.InputValidator.INTEGER_FILTER,
    },
        "label": "Enter the width of the new sprite:",
        "default": "",
        "filter": constants.InputValidator.INTEGER_FILTER,
    },
        "label": "Enter the name of the new sprite:",
        "default": f"Sprite {len(sprite_list)}",
        "filter": constants.InputValidator.PRINTABLE_FILTER,
    },
multi_input = MultiLineInput(fields, conf)
screen.place(multi_input, 10, 10)
completed_fields = multi_input.show()
```

init([fields, title, config])	
	param fields  A list of dictionnary that represent the fields to present to the
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
show()	Show the dialog and execute the event loop.

### **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
fields	Get and set the fields of the dialog, see the constructor for the format or this list.
title	Get and set the title of the dialog, it has to be a str.
user_input	Facility to store and retrieve the user input.

#### property config

Get and set the config object (*UiConfig*).

## property fields

Get and set the fields of the dialog, see the constructor for the format or this list.

 $render_to\_buffer(buffer, row, column, buffer\_height, buffer\_width) \rightarrow None$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- height (int) The total height of the display buffer.
- width (int) The total width of the display buffer.

### show()

Show the dialog and execute the event loop. Until this method returns, all keyboards event are processed by the local event loop. This is also true if called from the main event loop.

This event loop returns a list of dictionaries with the content of each fields. The list of dictionaries is the same than the fields constructor parameter but each key has an additional 'user\_input' field that contains the user input.

If the fields parameter was:

```
[ {
    "label": "Input a value:",
    (continues on next page)
```

(continued from previous page)

```
"default": "",
    "filter": constants.InputValidator.PRINTABLE_FILTER,
}
]
```

The returned value would be:

Example:

```
fields = multi_input.show()
```

#### property title: str

Get and set the title of the dialog, it has to be a str.

#### property user\_input

Facility to store and retrieve the user input.

# **ProgressBar**

Bases: object

A simple horizontal progress bar widget.

```
__init__(value=0, maximum=100, width=20, progress_marker=", empty_marker='', config=None)
```

#### **Parameters**

- **value** (*int*) The initial value parameter. It represents the progression.
- maximum (int) The maximum value held by the progress bar. Any value over the maximum is ignored.
- width (int) The width of the progress bar widget (in number of screen cells).
- **progress\_marker** (*pygamelib.gfx.core.Sprixel*) The progress marker is displayed on progression. It is the sprixel that fills the bar. Please see below.
- **empty\_marker** (*pygamelib.gfx.core.Sprixel*) The empty marker is displayed instead of the progress marker when the bar should be empty (when the value is too low to fill the bar for example). Please see below.
- **config** (*UiConfig*) The configuration object.

Here is a representation of were the progress and empty markers are used.

```
Progress marker

[====----]

|
Empty marker
```

## Example:

```
# Create a default progress bar with the default configuration
progress_bar = ProgressBar(config=UiConfig.instance())
# Place the progress bar in the middle of the screen
screen.place(
    progress_bar, screen.vcenter, screen.hcenter - int(progress_bar.width)
)
for progress in range(progress_bar.maximum + 1):
    # Do something useful
    progress_bar.value = progress
    screen.update()
```

## **Methods**

init([value, maximum, width,])	
	param value  The initial value parameter. It represents the progression.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.

## **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
empty_marker	Get and set the empty marker, preferrably a Sprixel but could be a str.
maximum	Get and set the maximum possible progress, it has to be an int.
progress_marker	Get and set the progress marker, preferrably a Sprixel but could be a str.
value	Get and set the current progress value, it has to be an int.

## property config

Get and set the config object (*UiConfig*).

### property empty\_marker

Get and set the empty marker, preferrably a Sprixel but could be a str.

#### property maximum

Get and set the maximum possible progress, it has to be an int.

### property progress\_marker

Get and set the progress marker, preferrably a Sprixel but could be a str.

```
render_to_buffer(buffer, row, column, buffer_height, buffer_width)
```

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property value

Get and set the current progress value, it has to be an int.

## **ProgressDialog**

Bases: Dialog

ProgressDialog is a progress bar widget as a dialog (or popup). The main difference with a progress bar with borders is that it is automatically rendered on the second pass by the screen object (therefore, is visible on top of other graphical elements).

This dialog requires external interactions so it is the only dialog widget that does not provide a useful show() implementation. As a matter of fact, show do nothing at all.

ProgressDialog is mainly a label, a box and a *ProgressBar* bundled together.

```
__init__(label=Progress dialog[0m, value=0, maximum=100, width=20, progress_marker=", empty_marker=" ', adaptive_width=True, destroy_on_complete=True, config=None)
```

The constructor accepts the following parameters.

## **Parameters**

- label (str | base.Text) A label to display on top of the progress bar.
- **value** (*int*) The initial value parameter. It represents the progression.
- maximum (int) The maximum value held by the progress bar. Any value over the maximum is ignored.
- width (int) The width of the progress bar widget (in number of screen cells).
- **progress\_marker** (*pygamelib.gfx.core.Sprixel*) The progress marker is displayed on progression. It is the sprixel that fills the bar. Please see below.
- **empty\_marker** (*pygamelib.gfx.core.Sprixel*) The empty marker is displayed instead of the progress marker when the bar should be empty (when the value is too low to fill the bar for example). Please see below.

- adaptive\_width (bool) If True, the dialog will automatically adapt to the size of the label.
- **destroy\_on\_complete** If True, the dialog will remove itself from the screen when complete (i.e: when value == maximum)
- **config** (*UiConfig*) The configuration object.

## Example:

```
# Create a default progress bar with the default configuration
progress_dial = ProgressDialog(
    "Please wait while I'm doing something super duper important",
    config=UiConfig.instance(),
)
# Place the progress bar in the middle of the screen
screen.place(
    progress_dial, screen.vcenter, screen.hcenter - int(progress_bar.width)
)
for progress in range(progress_dial.maximum + 1):
    # Do something useful
    progress_dial.value = progress
    screen.update()
```

## **Methods**

init([label, value, maximum, width,])	The constructor accepts the following parameters.
render_to_buffer(buffer, row, column,)	Render the object from the display buffer to the frame buffer.
show()	The show method does nothing in the ProgressDialog.

## **Attributes**

config	Get and set the config object ( <i>UiConfig</i> ).
label	Get and set the label of the dialog, it has to be a str or
	base.Text.
maximum	Get and set the maximum possible progress, it has to
	be an int.
user_input	Facility to store and retrieve the user input.
value	Get and set the current progress value, it has to be an
	int.

# property config

Get and set the config object (*UiConfig*).

# property label

Get and set the label of the dialog, it has to be a str or base. Text.

# property maximum

Get and set the maximum possible progress, it has to be an int.

3.7. gfx 681

```
render_to_buffer(buffer, row, column, buffer_height, buffer_width)
```

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- **row** (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## show()

The show method does nothing in the ProgressDialog. It is a notable exception and the only dialog widget in the UI module to do that.

#### property user\_input

Facility to store and retrieve the user input.

#### property value

Get and set the current progress value, it has to be an int.

# **UiConfig**

```
class pygamelib.gfx.ui.UiConfig(game=None, box_vertical_border='|', box_horizontal_border='-', box_top_left_corner='', box_top_right_corner='', box_bottom_left_corner='', box_bottom_right_corner='', box_vertical_and_left='', fg_color: Color = Color(255, 255, 255), bg_color: Color = Color(0, 128, 128), fg_color_inactive: Color = Color(128, 128, 128), bg_color_selected: Color = Color(128, 128, 128), bg_color_not_selected=None, fg_color_selected: Color = Color(0, 255, 0), fg_color_not_selected: Color = Color(255, 255, 255), bg_color_menu_not_selected: Color = Color(128, 128, 128), border_fg_color: Color = Color(255, 255, 255), border_bg_color: Color = None, borderless_dialog: bool = True, widget_bg_color: Color = Color(0, 128, 128), input_fg_color: Color = Color(255, 255, 255), input_bg_color: Color = Color(163, 163, 163))
```

Bases: object

A configuration object for the UI module. TEST

This object's purpose is to configure the look and feel of the UI widgets. It does nothing by itself.

# **Parameters**

- **game** (*Game*) The game object.
- **box\_vertical\_border** (*str*) The vertical border of a box.
- **box\_horizontal\_border** (*str*) The horizontal border of a box.
- **box\_top\_left\_corner** (*str*) The top left corner of a box.
- **box\_top\_right\_corner** (*str*) The top right corner of a box.
- box\_bottom\_left\_corner (str) The bottom left corner of a box.

- **box\_bottom\_right\_corner** (*str*) The bottom right corner of a box.
- box\_vertical\_and\_right (str) The left junction between two boxes.
- box\_vertical\_and\_left (str) The right junction between two boxes.
- **fg\_color** (*Color*) The foreground color (for text and content).
- **bg\_color** (*Color*) The background color (for text and content).
- **fg\_color\_inactive** (*Color*) The foreground color for inactive items like menu entries.
- **bg\_color\_selected** (*Color*) The background color (for selected text and content).
- **bg\_color\_not\_selected** (*Color*) The background color (for non selected text and content).
- **fg\_color\_selected** (*Color*) The foreground color (for selected text and content).
- **fg\_color\_not\_selected** (*Color*) The foreground color (for non selected text and content).
- **bg\_color\_menu\_not\_selected** (*Color*) The menu background color (for expanded menu items).
- **border\_fg\_color** (*Color*) The foreground color (for borders).
- **border\_bg\_color** (*Color*) The background color (for borders).
- **borderless\_dialog** (*bool*) Is the dialog borderless or not.
- widget\_bg\_color (Color) The background color of a widget.
- input\_fg\_color (Color) The foreground color (i.e the text color) of a LineInput widget.
- input\_bg\_color (*Color*) The background color of a LineInput widget.

# Example:

```
config_ui_red = UiConfig(
   fg_color=Color(255,0,0),
   border_fg_color=Color(255,0,0)
)
```

\_\_init\_\_(game=None, box\_vertical\_border='|', box\_horizontal\_border='-', box\_top\_left\_corner='', box\_top\_right\_corner='', box\_bottom\_left\_corner='', box\_bottom\_right\_corner='', box\_vertical\_and\_right='|-', box\_vertical\_and\_left='', fg\_color: Color = Color(255, 255, 255), bg\_color: Color = Color(0, 128, 128), fg\_color\_inactive: Color = Color(128, 128, 128), bg\_color\_selected: Color = Color(128, 128, 128), bg\_color\_not\_selected=None, fg\_color\_selected: Color = Color(0, 255, 0), fg\_color\_not\_selected: Color = Color(255, 255, 255), bg\_color\_menu\_not\_selected: Color = Color(128, 128, 128), border\_fg\_color: Color = Color(255, 255, 255), border\_bg\_color: Color = None, borderless\_dialog: bool = True, widget\_bg\_color: Color = Color(0, 128, 128), input\_fg\_color: Color = Color(255, 255, 255), input\_bg\_color: Color = Color(163, 163, 163))

# **Methods**

```
__init__([game, box_vertical_border, ...])

instance(*args, **kwargs)

Returns the instance of the UiConfig object
```

#### classmethod instance(\*args, \*\*kwargs)

Returns the instance of the UiConfig object

Creates an UiConfig object on first call an then returns the same instance on further calls. Useful for a default configuration. It accepts all the parameters from the constructor.

#### **Returns**

Instance of UiConfig object

# Widget

Bases: PglBaseObject

New in version 1.4.0.

The Widget object is the base for all UI elements (or should be). By itself it does not do anything functionally useful. What it does however, is taking care of the geometry logic.

It enforces the geometry constraints and takes care of sending resize events messages.

```
__init__(width: int = 0, height: int = 0, minimum_width: int = 0, minimum_height: int = 0, maximum_width: int = 20, maximum_height: int = 10, layout: Layout | None = None, bg_color: Color | None = None, config: UiConfig | None = None) \rightarrow None
```

#### **Parameters**

- width (int) The width of the widget.
- **height** (*int*) The height of the widget.
- **minimum\_width** (*int*) The minimum\_width of the widget.
- minimum\_height (int) The minimum\_height of the widget.
- maximum\_width (int) The maximum\_width of the widget.
- $maximum_height(int)$  The maximum\_height of the widget.
- **layout** (*Layout*) The layout of the widget.
- **bg\_color** (*Color*) The default background color of the widget. This property overrides the *widget\_bg\_color* from the *UiConfig* class (in case you want to create a specific widget with a different background color than the default one).
- **config** (*UiConfig*) The configuration object.

Example:

```
my_widget = Widget(6, 3, minimum_width=6, minimum_height=3)
my_widget.bg_color = Color(255, 255, 255)
screen.place(my_widget, 5, 2)
```

# **Methods**

init([width, height, minimum_width,])	
	param width
	The width of the widget.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render the object from the display buffer to the frame buffer.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.

# **Attributes**

$bg\_color$	This property get/set the background color of the widget.
children	This read only property property returns the list of children widgets.
focus	This property get/set the focus property.
height	This property get/set the height of the widget.
layout	This property get/set the layout of the widget.
maximum_height	This property get/set the maximum height of the widget.
maximum_width	This property get/set the maximum width of the widget.
minimum_height	This property get/set the minimum height of the widget.
minimum_width	This property get/set the minimum width of the widget.
parent	This property get/set the parent widget of the widget.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
size_constraint	This property get/set the size constraints of the widget.
width	This property get/set the width of the widget.
X	This property get/set the x position of the widget on screen.
у	This property get/set the y position of the widget on screen.

3.7. gfx 685

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

#### Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property bg\_color: Color

This property get/set the background color of the widget.

When the color is changed the pygamelib.gfx.ui.Widget.bg\_color:changed event is sent to the observers.

## property children: Set[Widget]

This read only property property returns the list of children widgets.

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

# **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# property focus: bool

This property get/set the focus property. It is a boolean.

At the moment it is mostly an informational property, to tell the programmer and potentially the Widget user (i.e: the class inheriting from Widget) about its own state.

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

#### property height: int

This property get/set the height of the widget. This property respects the boundaries set by the *maximum\_height* and *minimum\_height* properties.

When the height is changed the pygamelib.gfx.ui.Widget.resizeEvent:height event is sent to the observers.

# property layout: Layout

This property get/set the layout of the widget. You can then add sub widgets to the layout.

This must be a Layout or a class that inherits from it.

When the layout is changed the pygamelib.gfx.ui.Widget.layout:changed event is sent to the observers.

## property maximum\_height: int

This property get/set the maximum height of the widget. This property is used when changing the size constraints and the height property.

# property maximum\_width: int

This property get/set the maximum width of the widget. This property is used when changing the size constraints and the width property.

# property minimum\_height: int

This property get/set the minimum height of the widget. This property is used when changing the size constraints and the height property.

#### property minimum\_width: int

This property get/set the minimum width of the widget. This property is used when changing the size constraints and the width property.

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

```
property parent: Widget | None
```

This property get/set the parent widget of the widget.

 $\textbf{render\_to\_buffer}(\textit{buffer: numpy.array, row: int, column: int, buffer\_height: int, buffer\_width: int)} \rightarrow \\ \textbf{None}$ 

Render the object from the display buffer to the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

# property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (*int*) – the screen column

### Return type

int

# property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

# Return type

int

## property size\_constraint: SizeConstraint

This property get/set the size constraints of the widget. Changing the size constraints immediately resize the widget.

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- **row** (*int*) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## property width: int

This property get/set the width of the widget. This property respects the boundaries set by the *maxi-mum\_width* and *minimum\_width* properties.

When the width is changed the pygamelib.gfx.ui.Widget.resizeEvent:width event is sent to the observers.

#### property x: int

This property get/set the x position of the widget on screen. Since a Widget is a *PglBaseObject* this is an alias for the *screen\_column* property.

#### property y: int

This property get/set the y position of the widget on screen. Since a Widget is a *PglBaseObject* this is an alias for the *screen\_row* property.

# 3.7.3 particles

New in version 1.3.0.

Starting with version 1.3.0, the pygamelib now provides a particle system. It is for now a first limited version and it has a number of limitations.

First, the particles are "non interactive" objects. They are not affected by board items or anything drawn on screen nor can they affect them. All particles are drawn on top of an already rendered screen.

This means no fancy particle physics out of the box. It doesn't means that it is not doable. It just means that it is not existing out of the box.

Second, although I did my best to make the particle system as efficient as possible, drawing a lot of moving elements in the terminal is very slow. So be mindful of the performances when using it.

Now despite the limitations, the particle system still allow to do some very cool stuff. Here is a video example:

This is the benchmark of the particle system, the code is available on Ghithub.

**Important:** Like the UI module, the particles system works exclusively with the screen buffer system (place, delete, render, update, etc.). It doesn't work with Screen functions tagged "direct display" like display\_at().

#### **CircleEmitter**

class pygamelib.gfx.particles.CircleEmitter(emitter\_properties: EmitterProperties = None)

Bases: ParticleEmitter

The CircleEmitter differs from the *ParticleEmitter* in only one thing: it emits its particle in a circular shape, like this:

Aside from that specificity it's exactly the same as a regular particle emitter.

**\_\_init\_\_**(*emitter\_properties:* EmitterProperties = None)  $\rightarrow$  None

The CircleEmitter takes the same parameters than the *ParticleEmitter* and make use of EmitterProperties.radius.

The radius is used as the initial distance from the center of the circle (i.e the emitter's position).

# **Methods**

init([emitter_properties])	The CircleEmitter takes the same parameters than the <i>ParticleEmitter</i> and make use of EmitterProperties.radius.
apply_force(force)	Apply a force to all alive particles.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
<pre>emit([amount])</pre>	Emit a certain amount of particles.
finished()	Returns True if the emitter is finished.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a particle emitter from serialized data.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render all the particles of that emitter in the frame buffer.
resize_pool([new_size])	In substance, this method is an alias for ParticleEmitter.particle_pool.resize().
serialize()	Serialize the particle emitter.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
toggle_active()	Toggle the emitter's state between active and inactive.
update()	Update all the particles in the pool.

# **Attributes**

active column	Access and set the active property. Access and set the column property (i.e. x).
particle_pool	This property holds this emitter's instance of a <i>ParticlePool</i> .
row	Access and set the row property (i.e: y).
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property (i.e: column).
У	Access and set the y property (i.e: row).

# property active

Access and set the active property.

An emitter only emits particles if he is active. Emitted particles keeps being updated even if the emitter is not active anymore, for obvious reasons.

# apply\_force(force: Vector2D)

Apply a force to all alive particles.

The force needs to be a *Vector2D*.

# **Parameters**

**force** (*Vector2D*) – The force to apply to the particles.

Example:

690 Chapter 3. Tutorials

```
my_emitter.apply_force(base.Vector2D(0,0.3)) # slight wind.
```

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*Pg1Base0bject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

#### Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

### property column

Access and set the column property (i.e. x).

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*Pg1BaseObject*) – An observer to detach from this object.

# Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## emit(amount: int = None)

Emit a certain amount of particles.

The emitter will request particles from the particle pool. This in turn will trigger the recycling of dead particles if needed.

Calling this method faster than the configured emit\_rate is not going to emit more particles. An emitter cannot emit particles faster than its emit\_rate.

If amount is None, the emitter emits emit\_number particles.

### **Parameters**

**amount** (*int*) – The amount (number) of particles to be emitted.

Example:

```
my_emitter.emit(50)
```

# finished()

Returns True if the emitter is finished.

A finished emitter has both:

- Reach the end of its lifespan (i.e lifespan == 0)
- And all particles are finished too.

This means that an emitter will, in most cases, not be finished as soon as its lifespan reaches 0 but a bit after. When all of its managed particles are dead.

This is on purpose for both aesthetic reasons (avoiding particles sudden removal) and for optimization (counting active particles is a O(n) operation and can be very long when there's a lot of particles so we want to do it only when necessary).

Example:

```
if my_emitter.finished():
    screen.delete(my_emitter.row, my_emitter.column)
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# classmethod load(data)

Load a particle emitter from serialized data.

### **Parameters**

data (dict) – The serialized data.

#### Returns

The loaded particle emitter.

#### **Return type**

ParticleEmitter

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.

• value (Any) – An optional parameter that identify the new value of the attribute.

### Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

#### property particle\_pool

This property holds this emitter's instance of a ParticlePool.

```
render_to_buffer(buffer, row, column, buffer_height, buffer_width)
```

Render all the particles of that emitter in the frame buffer.

This method is automatically called by pygamelib.engine.Screen.render().

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

```
resize_pool(new_size: int = None)
```

In substance, this method is an alias for ParticleEmitter.particle\_pool.resize(). However, called without parameter, it will try to resize the particle pool to emit\_number \* particle\_lifespan. It will do so only if the resulting number is greater than the current particle pool size.

#### **Parameters**

```
new_size (int) – The desired new size of the pool.
```

# Example:

```
my_emitter.resize_pool(3000)
```

# property row

Access and set the row property (i.e: y).

# property screen\_column: int

A property to get/set the screen column.

# **Parameters**

 $\textbf{value}\ (int)-the\ screen\ column$ 

#### **Return type**

int

# property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

# **Return type**

int

### serialize()

Serialize the particle emitter.

#### Returns

A dictionary containing all the emitter's properties.

## **Return type**

dict

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- **row** (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## toggle\_active()

Toggle the emitter's state between active and inactive.

An inactive emitter does not emit new particles but keeps processing particles that have already been emitted.

Example:

```
if not my_emitter.active:
    my_emitter.toggle_active()
```

# update()

Update all the particles in the pool.

Updating a particle means applying particle\_acceleration to every particle and then call *Particle. update()*.

Example:

```
my_emitter.update()
```

## property x

Access and set the x property (i.e. column).

# property y

Access and set the y property (i.e: row).

## **ColorParticle**

```
class pygamelib.gfx.particles.ColorParticle(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, sprixel: ParticleSprixel = None, start\_color: Color = None, start\_color: Color = None, start\_color: Color = None)
```

Bases: Particle

This class is an extension of *Particle*. It adds the possibility to gradually go from a starting color to an end color over time. It is linked with the lifespan of the particle.

```
__init__(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, sprixel: ParticleSprixel = None, start_color: Color = None, stop_color: Color = None) \rightarrow None
```

The constructor takes the following parameters.

#### **Parameters**

- row (int) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.
- **sprixel** (*Sprixel*) The sprixel that represent the particle when drawn on screen.
- **start\_color** (*Color*) The color of the particle at the beginning of its lifespan.
- **stop\_color** (*Color*) The color of the particle at the end of its lifespan.

# Example:

```
single_particle = ColorParticle(
    row=5,
    column=5,
    velocity=base.Vector2D(-0.5, 0.0),
    lifespan=10,
    start_color=core.Color(255, 0, 0),
    stop_color=core.Color(0, 255, 0),
)
```

3.7. gfx 695

# **Methods**

init ([	The construction to least the full purious and another
init([row, column, velocity, lifespan,])	The constructor takes the following parameters.
<pre>apply_force(force)</pre>	Apply a force to the particle's acceleration vector.
attach(observer)	Attach an observer to this instance.
<pre>detach(observer)</pre>	Detach an observer from this instance.
finished()	Return True if the particle is done living (i.e its lifespan is lesser or equal to 0).
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a ColorParticle from a dictionary.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
render([sprixel])	Render the particle as a <i>Sprixel</i> .
<pre>reset([row, column, velocity, lifespan])</pre>	Reset a particle in its initial state.
reset_lifespan([lifespan])	Reset the particle lifespan (including the initial lifespan).
serialize()	Serialize a ColorParticle into a dictionary.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
terminate()	Terminate a particle, i.e sets its lifespan to -1.
update()	The update method perform the calculations required
	to process the new particle position.

#### **Attributes**

column	Access and set the column property.
row	Access and set the row property.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property.
У	Access and set the y property.

# $apply_force(force: Vector2D) \rightarrow None$

Apply a force to the particle's acceleration vector.

You are more likely to apply forces to all particles of an emitter through the apply\_force() method of the emitter class.

# **Parameters**

**force** (*Vector2D*) – The force to apply.

## Example:

```
gravity = Vector2D(-0.2, 0.0)
my_particle.apply_force(gravity)
```

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property column

Access and set the column property. Equivalent to the x property.

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

#### Return type

bool

# Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### $finished() \rightarrow bool$

Return True if the particle is done living (i.e its lifespan is lesser or equal to 0). It returns False otherwise.

# Return type

bool

# Example:

```
if not my_particle.finished():
    my_particle.update()
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.

• **value** (*Any*) – The new value of the attribute. This can be None.

#### classmethod load(data)

Load a ColorParticle from a dictionary.

#### **Parameters**

**data** (*dict*) – The dictionary to load from

#### Returns

The loaded ColorParticle

#### **Return type**

ColorParticle

# Example:

```
particle = ColorParticle.load( json.load( open("particle.json") )
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

```
render(sprixel: Sprixel = None)
```

Render the particle as a *Sprixel*. This method is called by the *ParticleEmitter* render\_to\_buffer method.

It takes a *Sprixel* as a parameter. This Sprixel is given by the ParticleEmitter.render\_to\_buffer() method and if it is not None, the particle will render itself into that *Sprixel* and return it.

**Important:** This method must be called after everything else as rendered or else there will be *Sprixe1* that will be overwritten during their rendering cycle. Other elements could also have their *Sprixe1* corrupted and replaced by the particle's one.

# **Parameters**

**sprixel** (*Sprixel*) – A sprixel already rendered in the screen buffer.

## Example:

```
p = my_particle
buffer[p.row][p.column] = p.render(buffer[p.row][p.column])
```

```
reset(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None)
```

Reset a particle in its initial state. This is particularly useful for the reuse of particles.

This method takes almost the same parameters than the constructor.

#### **Parameters**

- row (int) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.

# Example:

```
single_particle.reset(
   row=5,
   column=5,
   velocity=base.Vector2D(-0.5, 0.0),
   lifespan=10,
)
```

# **reset\_lifespan**(lifespan: int = 20) $\rightarrow$ None

Reset the particle lifespan (including the initial lifespan).

#### **Parameters**

**lifespan** (*int*) – The particle lifespan in number of movements/turns.

#### Example:

```
my_particle.reset_lifespan(10)
```

## property row

Access and set the row property. Equivalent to the y property.

# property screen\_column: int

A property to get/set the screen column.

#### **Parameters**

**value** (int) – the screen column

# Return type

int

# property screen\_row: int

A property to get/set the screen row.

# **Parameters**

**value** (int) – the screen row

# Return type

int

# serialize()

Serialize a ColorParticle into a dictionary.

## Returns

The class as a dictionary

3.7. gfx 699

#### Return type

dict

Example:

```
json.dump( particle.serialize() )
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

## **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## $terminate() \rightarrow None$

Terminate a particle, i.e sets its lifespan to -1.

In that case the ParticleEmitter and ParticlePool will recycle it. That is *IF* you are managing the particle through an emitter and/or a pool of course.

Example:

```
p = my_particle
if p.row >= screen,height or p.column >= screen.width:
    p.terminate()
```

#### update()

The update method perform the calculations required to process the new particle position. It mainly adds the acceleration to the velocity vector and update the position accordingly.

After calling update() the acceleration is "consumed" in the velocity and therefor reset.

The update() method takes no parameters and returns nothing.

Example:

```
my_particle.update()
```

#### property x

Access and set the x property. Equivalent to the column property.

### property y

Access and set the y property. Equivalent to the row property.

## ColorPartitionParticle

Bases: PartitionParticle

This class is basically the same as *ColorParticle* but its base class is *PartitionParticle* instead of *Particle*. Everything else is the same.

It serves the same purpose as the ColorParticle with the added partition particle capabilities.

```
__init__(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, partition: list = None, partition_blending_table: list = None, start_color: Color = None, stop_color: Color = None) \rightarrow None
```

The constructor takes the following parameters.

#### **Parameters**

- row (int) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.
- **partition** (*list*) The partition of the particle.
- $partition\_blending\_table$  (list) The blending table of the particle.
- **start\_color** (*Color*) The color of the particle at the beginning of its lifespan.
- **stop\_color** (*Color*) The color of the particle at the end of its lifespan.

# Example:

```
single_particle = RandomColorPartitionParticle(
   row=5,
   column=5,
   velocity=base.Vector2D(-0.5, 0.0),
   lifespan=10,
)
```

3.7. gfx 701

# **Methods**

init([row, column, velocity, lifespan,])	The constructor takes the following parameters.
apply_force(force)	Apply a force to the particle's acceleration vector.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
finished()	Return True if the particle is done living (i.e its lifespan is lesser or equal to 0).
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a ColorPartitionParticle from a dictionary.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
render([sprixel])	This method first calls the Particle.render() method.
<pre>reset([row, column, velocity, lifespan])</pre>	Reset a particle in its initial state.
reset_lifespan([lifespan])	Reset the particle lifespan (including the initial lifespan).
serialize()	Serialize a ColorPartitionParticle into a dictionary.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
terminate()	Terminate a particle, i.e sets its lifespan to -1.
update()	This method first calls the Particle.update() method,
	then calculates the quadrant position, i.e: the actual
	position of the particle within a console character.

# **Attributes**

column	Access and set the column property.
row	Access and set the row property.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property.
У	Access and set the y property.

# $apply_force(force: Vector2D) \rightarrow None$

Apply a force to the particle's acceleration vector.

You are more likely to apply forces to all particles of an emitter through the apply\_force() method of the emitter class.

### **Parameters**

**force** (*Vector2D*) – The force to apply.

# Example:

```
gravity = Vector2D(-0.2, 0.0)
my_particle.apply_force(gravity)
```

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

702 Chapter 3. Tutorials

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

#### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property column

Access and set the column property. Equivalent to the x property.

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*Pg1Base0bject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# $finished() \rightarrow bool$

Return True if the particle is done living (i.e its lifespan is lesser or equal to 0). It returns False otherwise.

# Return type

bool

## Example:

```
if not my_particle.finished():
    my_particle.update()
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

• **subject** (*PglBase0bject*) – The object that has changed.

- attribute (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

# classmethod load(data)

Load a ColorPartitionParticle from a dictionary.

#### **Parameters**

**data** (*dict*) – The dictionary to load from

#### Returns

The loaded ColorPartitionParticle

# Return type

ColorPartitionParticle

#### Example:

```
particle = ColorPartitionParticle.load( json.load( open("particle.json") )
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# render(sprixel: Sprixel = None)

This method first calls the Particle.render() method. Then it updates the rendered particle's model based on the blending table.

### **Parameters**

**sprixel** (*Sprixel*) – A sprixel already rendered in the screen buffer.

# Example:

```
p = my_particle
buffer[p.row][p.column] = p.render(buffer[p.row][p.column])
```

```
reset(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None)
```

Reset a particle in its initial state. This is particularly useful for the reuse of particles.

This method takes almost the same parameters than the constructor.

#### **Parameters**

• **row** (*int*) – The initial row position of the particle on the screen.

- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.

Example:

```
single_particle.reset(
    row=5,
    column=5,
    velocity=base.Vector2D(-0.5, 0.0),
    lifespan=10,
)
```

```
reset_lifespan(lifespan: int = 20) \rightarrow None
```

Reset the particle lifespan (including the initial lifespan).

#### **Parameters**

**lifespan** (*int*) – The particle lifespan in number of movements/turns.

Example:

```
my_particle.reset_lifespan(10)
```

## property row

Access and set the row property. Equivalent to the y property.

# property screen\_column: int

A property to get/set the screen column.

## **Parameters**

value (int) – the screen column

## **Return type**

int

# property screen\_row: int

A property to get/set the screen row.

### **Parameters**

**value** (*int*) – the screen row

# Return type

int

# serialize()

Serialize a ColorPartitionParticle into a dictionary.

#### **Returns**

The class as a dictionary

# Return type

dict

Example:

```
json.dump( particle.serialize() )
```

3.7. gfx 705

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

## Example:

```
an_object.store_screen_coordinate(3,8)
```

#### $terminate() \rightarrow None$

Terminate a particle, i.e sets its lifespan to -1.

In that case the ParticleEmitter and ParticlePool will recycle it. That is *IF* you are managing the particle through an emitter and/or a pool of course.

## Example:

```
p = my_particle
if p.row >= screen,height or p.column >= screen.width:
    p.terminate()
```

#### update()

This method first calls the Particle.update() method, then calculates the quadrant position, i.e: the actual position of the particle within a console character. It then updates the particle's model based on this internal position.

Example:

```
my_particle.update()
```

#### property x

Access and set the x property. Equivalent to the column property.

# property y

Access and set the y property. Equivalent to the row property.

# **EmitterProperties**

Bases: object

EmitterProperties is a class that hold configuration variables for a particle emitter. The idea is that it's easier to carry around for multiple emitters with the same configuration than multiple values in the emitter's constructor.

It holds all possible parmeters for all types of emitters. Emitters uses only the ones that they really need.

**Important:** In most cases these values are copied by the emitter's constructor. So changing the values during an emitter's alive cycle is not going to do anything.

**Note:** This class should be a @dataclass. However, support for keyword only data classes is specific to python 3.10+. So for now, it is a regular class.

\_\_init\_\_(row: int = 0, column: int = 0, variance: float = 2.0, emit\_number: int = 1, emit\_rate: float = 0.1, lifespan: int = 200, parent=None, particle\_velocity=None, particle\_acceleration=None, particle\_lifespan: float = 5.0, radius: float = 1.0, particle: Particle = None)  $\rightarrow$  None

#### **Parameters**

- **row** (*int*) The row where the emitter is. It is only important for the first rendering cycle. After that, the emitter will know its position on screen.
- **column** (*int*) The row where the emitter is. It is only important for the first rendering cycle. After that, the emitter will know its position on screen.
- **variance** (*float*) The variance is the amount of randomness that is allowed when emitting a particle. The exact use of this parameter is specific to each emitter.
- **emit\_number** (*int*) The number of particle emitted at each timer tick.
- **emit\_rate** (*float*) The rate of emission in seconds. This value needs to be understood as "the emitter will emit **emit\_number** particles every **emit\_rate** seconds".
- **lifespan** (*int*) The lifespan of the emitter in number of emission cycle. If lifespan is set to 1 for example, the emitter will only emit one burst of particles.
- parent (BoardItem) A parent board item. If you do that manually, you will probably want to set it specifically for each emitter.
- particle\_velocity (*Vector2D*) The initial particle velocity. Please read the documentation of each emitter for the specific use of particle velocity.
- particle\_acceleration (*Vector2D*) The initial particle acceleration. Please read the documentation of each emitter for the specific use of particle acceleration.
- particle\_lifespan (int) The lifespan of the particle in number of cycles.
- **radius** (*float*) For emitter that supports it (like the CircleEmitter), sets the radius of emission (which translate into a velocity vector for each particle).
- **particle** (*Particle*) The particle that the emitter will emit. This can be a class reference or a fully instantiated particle. Emitters will copy it in the particle pool.

#### Example:

```
props = EmitterProperties(emit_number=10, emit_rate=0.1, lifespan=10)
```

## **Methods**

```
__init__([row, column, variance, ...])

param row

The row where the emitter is. It is only important for the first

load(data)

Load an EmitterProperties from a dictionary.

serialize()

Serialize an EmitterProperties into a dictionary.
```

#### classmethod load(data)

Load an EmitterProperties from a dictionary.

#### **Parameters**

**data** (*dict*) – The dictionary to load from.

#### Returns

The EmitterProperties object

### **Return type**

**EmitterProperties** 

#### Example:

#### serialize()

Serialize an EmitterProperties into a dictionary.

### Returns

The class as a dictionary

# Return type

dict

#### Example:

```
json.dump( emitter_properties.serialize() )
```

## **ParticleEmitter**

class pygamelib.gfx.particles.ParticleEmitter(emitter\_properties=None)

Bases: PglBaseObject

The particle emitter is a key piece of the pygamelib's particle system: it's the part that actually do something!

The emitter takes care of managing the particles' life cycle. It emits, move, apply forces, update and draw particles on screen. It also provide convenient methods to manage the particle pool or apply forces to all active particles in the pool.

Particle emitters are configured with *EmitterProperties*. This is a convenient way to place multiple emitters with the same configuration. For example, if you create a "torch fire" emitter, you can use the same properties to create multiple emitters. It's less cumbersome than having the parameters tied to an instance of the emitter.

Here is an example of that taken from examples/benchmark-particle-system:

Example:

```
# The torch fire properties
emt_props = particles.EmitterProperties(
    screen.vcenter, # Position is not important as it will be updated by the
    screen.hcenter, # ParticleEmitter.render_to_buffer method.
   lifespan=150,
    variance=0.3,
    emit_number=10,
    emit_rate=0.1.
    particle=particles.ColorPartitionParticle(
        start_color=core.Color(45, 151, 227),
        stop_color=core.Color(7, 2, 40),
    ),
    particle_lifespan=5,
   radius=0.4,
# Now create multiple emitters at different position with the same properties.
for c in [[20, 24], [20, 35], [20, 122], [20, 133]]:
    bench_state.particle_emitters.append(particles.CircleEmitter(emt_props))
    screen.place(
        bench_state.particle_emitters[-1],
        screen.vcenter - int(bench_state.altar_sprite.height / 2) + c[0],
        2, # Always set your emitters to be rendered on the second pass.
    )
```

**Important:** The entire particle system is build around the **Screen Buffer** system and is completely incompatible with the direct display system. If you want to use the particle system you **have to** use Screen.place() and the other methods of the **Screen Buffer** system.

An emitter should always be placed on screen and set to render on the second rendering pass.

It is important if you want to avoid artifacts (like particles being rendered only under the position of the emitter).

The particles by themselves are not able to render on screen, the emitter is doing that job for them.

It also means that the particles are rendered and displayed over a screen that is already rendered. Therefor, by default and for the moment, they cannot interact with elements on screen or items in a board. It also means that there is no built in particle physics (for the moment).

```
__init__(emitter properties=None) \rightarrow None
```

The constructor takes the following parameter:

## **Parameters**

 $\begin{tabular}{ll} \bf emitter\_properties \ (\it EmitterProperties) - The properties \ of that particle \ emitter. \end{tabular}$ 

# **Methods**

init([emitter_properties])	The constructor takes the following parameter:
apply_force(force)	Apply a force to all alive particles.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
emit([amount])	Emit a certain amount of particles.
finished()	Returns True if the emitter is finished.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a particle emitter from serialized data.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
<pre>render_to_buffer(buffer, row, column,)</pre>	Render all the particles of that emitter in the frame buffer.
resize_pool([new_size])	In substance, this method is an alias for ParticleEmitter.particle_pool.resize().
serialize()	Serialize the particle emitter.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
toggle_active()	Toggle the emitter's state between active and inactive.
update()	Update all the particles in the pool.

## **Attributes**

active	Access and set the active property.
column	Access and set the column property (i.e: x).
particle_pool	This property holds this emitter's instance of a <i>ParticlePool</i> .
row	Access and set the row property (i.e: y).
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property (i.e: column).
у	Access and set the y property (i.e: row).

# property active

Access and set the active property.

An emitter only emits particles if he is active. Emitted particles keeps being updated even if the emitter is not active anymore, for obvious reasons.

# apply\_force(force: Vector2D)

Apply a force to all alive particles.

The force needs to be a *Vector2D*.

#### **Parameters**

**force** (*Vector2D*) – The force to apply to the particles.

# Example:

```
my_emitter.apply_force(base.Vector2D(0,0.3)) # slight wind.
```

710 Chapter 3. Tutorials

### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### **Returns**

True or False depending on the success of the operation.

#### Return type

bool

# Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property column

Access and set the column property (i.e. x).

#### detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

```
emit(amount: int = None) \rightarrow None
```

Emit a certain amount of particles.

The emitter will request particles from the particle pool. This in turn will trigger the recycling of dead particles if needed.

Calling this method faster than the configured emit\_rate is not going to emit more particles. An emitter cannot emit particles faster than its emit\_rate.

If amount is None, the emitter emits emit\_number particles.

## **Parameters**

**amount** (*int*) – The amount (number) of particles to be emitted.

Example:

```
my_emitter.emit(50)
```

3.7. gfx 711

### finished()

Returns True if the emitter is finished.

A finished emitter has both:

- Reach the end of its lifespan (i.e lifespan == 0)
- And all particles are finished too.

This means that an emitter will, in most cases, not be finished as soon as its lifespan reaches 0 but a bit after. When all of its managed particles are dead.

This is on purpose for both aesthetic reasons (avoiding particles sudden removal) and for optimization (counting active particles is a O(n) operation and can be very long when there's a lot of particles so we want to do it only when necessary).

### Example:

```
if my_emitter.finished():
    screen.delete(my_emitter.row, my_emitter.column)
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

#### classmethod load(data)

Load a particle emitter from serialized data.

# **Parameters**

**data** (*dict*) – The serialized data.

### Returns

The loaded particle emitter.

#### **Return type**

ParticleEmitter

```
notify(modifier=None, attribute: str = None, value: Any = None) \rightarrow None
```

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- **value** (*Any*) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## property particle\_pool

This property holds this emitter's instance of a *ParticlePool*.

```
render_to_buffer(buffer, row, column, buffer_height, buffer_width)
```

Render all the particles of that emitter in the frame buffer.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

```
resize_pool(new_size: int = None)
```

In substance, this method is an alias for ParticleEmitter.particle\_pool.resize(). However, called without parameter, it will try to resize the particle pool to emit\_number \* particle\_lifespan. It will do so only if the resulting number is greater than the current particle pool size.

#### Parameters 4 8 1

```
new_size (int) – The desired new size of the pool.
```

Example:

```
my_emitter.resize_pool(3000)
```

### property row

Access and set the row property (i.e: y).

# property screen\_column: int

A property to get/set the screen column.

## **Parameters**

**value** (int) – the screen column

#### Return type

int

# property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

#### Return type

int

#### serialize()

Serialize the particle emitter.

### Returns

A dictionary containing all the emitter's properties.

## **Return type**

dict

# $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- **row** (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## toggle\_active()

Toggle the emitter's state between active and inactive.

An inactive emitter does not emit new particles but keeps processing particles that have already been emitted.

Example:

```
if not my_emitter.active:
    my_emitter.toggle_active()
```

# update()

Update all the particles in the pool.

Updating a particle means applying particle\_acceleration to every particle and then call *Particle. update()*.

Example:

```
my_emitter.update()
```

## property x

Access and set the x property (i.e. column).

# property y

Access and set the y property (i.e: row).

## **ParticlePool**

Bases: object

The particle pool is a structure that holds a large number of particles and make them available to the emitters.

Its main role is to optimize the performances (both speed and memory usage). It works by pre-instantiating a desired number of particles according to the *EmitterProperties* that is given to the constructor.

The particle pool is optimized to avoid searching for available particles. It sets its own size to avoid relying on anything but its last known particle made available to the emitter. So unless for specific behavior, it is probably a good idea to let it sets its own size.

It also recycle particles that are *finished()* to avoid a constant cycle of creation/destruction of a large amount of particle objects.

**\_\_init\_\_**(*size:* int = None,  $emitter\_properties$ : EmitterProperties = None)  $\rightarrow$  None The constructor takes the following parameters:

# Parameters

- **size** (*int*) The size of the pool in number of particles. For this to be efficient, be sure to have enough particles to cover for enough cycles before your first emitted particles are finished. The *ParticleEmitter* uses the following rule to size the pool: emit\_rate \* particle\_lifespan. It is the default value if size is not specified.
- **emitter\_properties** (*EmitterProperties*) The properties of the particles that needs to be pre-instantiated.

#### Example:

```
my_particle_pool = ParticlePool(500, my_properties)
```

#### **Methods**

init([size, emitter_properties])	The constructor takes the following parameters:
<pre>count_active_particles()</pre>	Returns the number of active particle (i.e not finished)
	in the pool.
<pre>get_particles([amount])</pre>	Returns the requested amount of particles.
resize(new_size)	Resize the particle pool to a new size.

# **Attributes**

pool	A read-only property that returns the particle pool tu-
	ple.

#### $count_active_particles() \rightarrow int$

Returns the number of active particle (i.e not finished) in the pool.

**Important:** The only way to know the amount of alive particles is to go through the entire pool. Be aware of the performance impact on large particle pools.

#### Returns

the number of active particles.

# Return type

int

## Example:

```
if emitter.particles.count_active_particles() > 0:
    emitter.apply_force(gravity)
```

```
get_particles(amount: int = None) \rightarrow tuple
```

Returns the requested amount of particles.

It is important to know that no particle is created during that call. This method returns available particles in the pool. Particles are recycled after they "died".

If amount is not specified the pool returns EmitterProperties.emit\_number particles.

#### **Parameters**

**amount** (*int*) – The amount of particles to return.

#### Returns

A tuple containing the desired amount of particles.

## Return type

tuple

Example:

```
fresh_particles = my_particle_pool.get_particles(30)
```

# property pool: tuple

A read-only property that returns the particle pool tuple.

```
resize(new_size: int)
```

Resize the particle pool to a new size.

If the new size is greater than the old one, the pool will be filled by pre-instanciated particles. If it's shorter however, the extra particles will be destroyed.

#### **Parameters**

```
new_size (int) – The new size of the pool.
```

Example:

```
# Resize the particle pool to hold 100 particles.
my_particle_pool.resize(100)
```

# **Particle**

class pygamelib.gfx.particles.Particle(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, sprixel: ParticleSprixel = None)

Bases: *PglBaseObject*New in version 1.3.0.

The Particle class is the base class that is inherited from by all other particles. It is mostly a "data class" in the sense that it is a class used for calculations but is not able to render on screen by itself. All operations are pure data operations until the emitter draw the particles.

Altought the Particle class can be used on its own, it is most likely to be used as a template for a particle emitter.

```
__init__(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, sprixel: ParticleSprixel = None) \rightarrow None
```

The constructor takes the following parameters.

#### **Parameters**

- **row** (*int*) The initial row position of the particle on the screen.
- column(int) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.
- **sprixel** (*Sprixe1*) The sprixel that represent the particle when drawn on screen.

# Example:

```
single_particle = Particle(
   row=5,
   column=5,
   velocity=base.Vector2D(-0.5, 0.0),
   lifespan=10,
   sprixel=core.Sprixel(graphics.GeometricShapes.BLACK_CIRCLE)
)
```

# Methods

init([row, column, velocity, lifespan,])	The constructor takes the following parameters.
<pre>apply_force(force)</pre>	Apply a force to the particle's acceleration vector.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
finished()	Return True if the particle is done living (i.e its lifespan is lesser or equal to 0).
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a Particle from a dictionary.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
render([sprixel])	Render the particle as a Sprixe1.
<pre>reset([row, column, velocity, lifespan])</pre>	Reset a particle in its initial state.
reset_lifespan([lifespan])	Reset the particle lifespan (including the initial lifespan).
serialize()	Serialize a Particle into a dictionary.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
terminate()	Terminate a particle, i.e sets its lifespan to -1.
update()	The update method perform the calculations required
	to process the new particle position.

#### **Attributes**

column	Access and set the column property.
row	Access and set the row property.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property.
У	Access and set the y property.

# $apply\_force(force: Vector2D) \rightarrow None$

Apply a force to the particle's acceleration vector.

You are more likely to apply forces to all particles of an emitter through the apply\_force() method of the emitter class.

## **Parameters**

**force** (*Vector2D*) – The force to apply.

## Example:

```
gravity = Vector2D(-0.2, 0.0)
my_particle.apply_force(gravity)
```

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

# **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

# Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property column

Access and set the column property. Equivalent to the x property.

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

### Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

### $finished() \rightarrow bool$

Return True if the particle is done living (i.e its lifespan is lesser or equal to 0). It returns False otherwise.

# Return type

bool

# Example:

```
if not my_particle.finished():
   my_particle.update()
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.

3.7. qfx 719

• **value** (*Any*) – The new value of the attribute. This can be None.

#### classmethod load(data)

Load a Particle from a dictionary.

#### **Parameters**

data (dict) – The dictionary to load from

#### Returns

The loaded Particle

## **Return type**

Particle

## Example:

```
particle = Particle.load( json.load( open("particle.json") ) )
```

**notify** (*modifier=None*, *attribute*: str = None, *value*: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## render(sprixel: Sprixel = None)

Render the particle as a *Sprixel*. This method is called by the *ParticleEmitter* render\_to\_buffer method.

It takes a *Sprixel* as a parameter. This Sprixel is given by the ParticleEmitter.render\_to\_buffer() method and if it is not None, the particle will render itself into that *Sprixel* and return it.

**Important:** This method must be called after everything else as rendered or else there will be *Sprixel* that will be overwritten during their rendering cycle. Other elements could also have their *Sprixel* corrupted and replaced by the particle's one.

## **Parameters**

**sprixel** (*Sprixel*) – A sprixel already rendered in the screen buffer.

## Example:

```
p = my_particle
buffer[p.row][p.column] = p.render(buffer[p.row][p.column])
```

```
reset(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None)
```

Reset a particle in its initial state. This is particularly useful for the reuse of particles.

This method takes almost the same parameters than the constructor.

#### **Parameters**

- row (int) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.

# Example:

```
single_particle.reset(
   row=5,
   column=5,
   velocity=base.Vector2D(-0.5, 0.0),
   lifespan=10,
)
```

# **reset\_lifespan**(lifespan: int = 20) $\rightarrow$ None

Reset the particle lifespan (including the initial lifespan).

#### **Parameters**

**lifespan** (*int*) – The particle lifespan in number of movements/turns.

#### Example:

```
my_particle.reset_lifespan(10)
```

## property row

Access and set the row property. Equivalent to the y property.

## property screen\_column: int

A property to get/set the screen column.

### **Parameters**

value (int) – the screen column

## Return type

int

## property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

value (int) - the screen row

# Return type

int

# serialize()

Serialize a Particle into a dictionary.

## Returns

The class as a dictionary

3.7. qfx 721

### Return type

dict

Example:

```
json.dump( particle.serialize() )
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

## **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

## $terminate() \rightarrow None$

Terminate a particle, i.e sets its lifespan to -1.

In that case the ParticleEmitter and ParticlePool will recycle it. That is *IF* you are managing the particle through an emitter and/or a pool of course.

Example:

```
p = my_particle
if p.row >= screen,height or p.column >= screen.width:
    p.terminate()
```

## $update() \rightarrow None$

The update method perform the calculations required to process the new particle position. It mainly adds the acceleration to the velocity vector and update the position accordingly.

After calling update() the acceleration is "consumed" in the velocity and therefor reset.

The update() method takes no parameters and returns nothing.

Example:

```
my_particle.update()
```

## property x

Access and set the x property. Equivalent to the column property.

## property y

Access and set the y property. Equivalent to the row property.

# **ParticleSprixel**

Bases: Sprixel

New in version 1.3.0.

The ParticleSprixel is nothing more than a *Sprixel*. Its only role is to help differentiate rendered sprixels for Partition Particles.

**\_\_init\_\_**(model=", bg\_color=None, fg\_color=None, is\_bg\_transparent=None)

## **Parameters**

- **model** (*str*) The model, it can be any string. Preferrably a single character.
- **bg\_color** (Color) A Color object to configure the background color.
- **fg\_color** (Color) A Color object to configure the foreground color.
- **is\_bg\_transparent** (*bool*) Set the background of the Sprixel to be transparent. It tells the engine to replace the background of the Sprixel by the background color of the overlapped sprixel.

# Methods

init([model, bg_color, fg_color,])	
, 00, 100,]/	param model
	The model, it can be any string. Pre-
	ferrably a single character.
attach(observer)	Attach an observer to this instance.
black_rect()	This class method returns a sprixel that is the equiv-
	alent of pygamelib.assets.graphics.BLACK_RECT.
black_square()	This class method returns a
	sprixel that is the equivalent of
	pygamelib.assets.graphics.BLACK_SQUARE.
blue_rect()	This class method returns a sprixel that is the equiv-
1-7	alent of pygamelib.assets.graphics.BLUE_RECT.
blue_square()	This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE_SQUARE.
copy()	Returns a (deep) copy of the sprixel.
cyan_rect()	This class method returns a sprixel that is the equiv-
	alent of pygamelib.assets.graphics.CYAN_RECT.
cyan_square()	This class method returns a sprixel that is the equiva-
	lent of pygamelib.assets.graphics.CYAN_SQUARE.
detach(observer)	Detach an observer from this instance.
<pre>from_ansi(string[, model])</pre>	Takes an ANSI string, parse it and return a Sprixel.
<pre>green_rect()</pre>	This class method returns a sprixel that is the equiv-
araan aayara()	alent of pygamelib.assets.graphics.GREEN_RECT. This class method returns a
<pre>green_square()</pre>	This class method returns a sprixel that is the equivalent of
	pygamelib.assets.graphics.GREEN_SQUARE.
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the
, <b>,</b> , , , , , , , , , , , , , , , , ,	observer.
load(data)	Create a new Sprixel object based on serialized data.
<pre>magenta_rect()</pre>	This class method returns a
	sprixel that is the equivalent of
<pre>magenta_square()</pre>	pygamelib.assets.graphics.MAGENTA_RECT. This class method returns a
magerita_square()	This class method returns a sprixel that is the equivalent of
	pygamelib.assets.graphics.MAGENTA_SQUARE.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
red_rect()	This class method returns a sprixel that is the equiv-
	alent of pygamelib.assets.graphics.RED_RECT.
red_square()	This class method returns a sprixel that is the equiv-
1	alent of pygamelib.assets.graphics.RED_SQUARE.
render_to_buffer(buffer, row, column,)	Render the sprixel from the display buffer to the frame buffer.
serialize()	Serialize a Sprixel into a dictionary.
store_screen_position(row, column)	Store the screen position of the object.
white_rect()	This class method returns a sprixel that is the equiv-
·	alent of pygamelib.assets.graphics.WHITE_RECT.
white_square()	This class method returns a
	sprixel that is the equivalent of
	pygamelib.assets.graphics.WHITE_SQUARE.
yellow_rect()	This class method returns a sprixel that is the equiva-
yellow_square()	lent of pygamelib.assets.graphics.YELLOW_RECT.  This class method returns a
	sprixel that is the Chapter valer Tutorials
	pygamelib.assets.graphics.YELLOW_SQUARE.

## **Attributes**

bg_color	A property to get/set the background color of the Sprixel.
$fg\_color$	A property to get/set the foreground color of the Sprixel.
length	Return the true length of the model.
mode1	A property to get/set the model of the Sprixel.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.

#### attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

# property bg\_color

A property to get/set the background color of the Sprixel.

#### **Parameters**

```
value (Color) - The new color
```

When the bg\_color is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprixel.bg\_color:changed event. The new bg\_color is passed as the *value* parameter.

Example:

```
# Access the sprixel's color
sprix.bg_color
# Set the sprixel's background color to some blue
sprix.bg_color = Color(0,128,255)
```

## classmethod black\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLACK\_RECT. The difference is that BLACK\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.black_rect()
```

### classmethod black\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLACK\_SQUARE. The difference is that BLACK\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.black_square()
```

## classmethod blue\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE\_RECT. The difference is that BLUE\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.blue_rect()
```

## classmethod blue\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE\_SQUARE. The difference is that BLUE\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.blue_square()
```

### copy()

Returns a (deep) copy of the sprixel.

New in version 1.3.0.

## classmethod cyan\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.CYAN\_RECT. The difference is that CYAN\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.cyan_rect()
```

## classmethod cyan\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.CYAN\_SQUARE. The difference is that CYAN\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.cyan_square()
```

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

## **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

## **Return type**

bool

726 Chapter 3. Tutorials

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## property fg\_color

A property to get/set the foreground color of the Sprixel.

#### **Parameters**

```
value (Color) - The new color
```

When the fg\_color is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprixel.fg\_color:changed event. The new fg\_color is passed as the *value* parameter.

## Example:

```
# Access the sprixel's color
sprix.fg_color
# Set the sprixel's foreground color to some green
sprix.fg_color = Color(0,255,128)
```

### static from\_ansi(string, model=")

Takes an ANSI string, parse it and return a Sprixel.

#### **Parameters**

- **string** (*str*) The ANSI string to parse.
- **model** (*str*) The character used to represent the sprixel in the ANSI sequence. Default is ""

#### Example:

```
new_sprixel = Sprixel.from_ansi(
    "\x1b[48;2;139;22;19m\x1b[38;2;160;26;23m\x1b[0m"])
```

**Warning:** This has mainly be tested with ANSI string generated by climage. If you find any issue, please report it

## classmethod green\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.GREEN\_RECT. The difference is that GREEN\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.green_rect()
```

# classmethod green\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.GREEN\_SQUARE. The difference is that GREEN\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

3.7. qfx 727

```
sprixel = Sprixel.green_square()
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

## property length

Return the true length of the model.

New in version 1.3.0.

With UTF8 and emojis the length of a string as returned by python's len() function is often very wrong. For example, the len("x1b[48;2;139;22;19mx1b[38;2;160;26;23mx1b[0m") returns 39 when it should return 1.

This method returns the actual printing/display size of the sprixel's model.

**Note:** This is a read only value. It is automatically updated when the model is changed.

Example:

```
if sprix.length > 2:
    print(
         f"Warning: that sprixel {sprix} will break the rest of the "
         "board's alignement"
        )
```

## classmethod load(data)

Create a new Sprixel object based on serialized data.

New in version 1.3.0.

#### **Parameters**

**data** (*dict*) – Data loaded from JSON data (deserialized).

# Return type Sprixel

Example:

```
new_sprite = Sprixel.load(json_parsed_data['default_sprixel'])
```

### classmethod magenta\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.MAGENTA\_RECT. The difference is that MAGENTA\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.magenta_rect()
```

## classmethod magenta\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.MAGENTA\_SQUARE. The difference is that MAGENTA\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.magenta_square()
```

# property model

A property to get/set the model of the Sprixel.

#### **Parameters**

**value** (str) – The new model

When the model is changed, the observers are notified of the change with the pygamelib.gfx.core.Sprixel.model:changed event. The new model is passed as the *value* parameter.

Example:

```
# Get the sprixel's model
sprix.model
# Set the sprixel's model to "@"
sprix.model = "@"
```

**notify**(modifier = None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

## **Parameters**

- **modifier** (*PglBaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- attribute (str) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## classmethod red\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.RED\_RECT. The difference is that RED\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

3.7. qfx 729

```
sprixel = Sprixel.red_rect()
```

# classmethod red\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.RED\_SQUARE. The difference is that RED\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.red_square()
```

render\_to\_buffer(buffer, row, column, buffer\_height, buffer\_width)

Render the sprixel from the display buffer to the frame buffer.

New in version 1.3.0.

This method is automatically called by *pygamelib.engine.Screen.render()*.

#### **Parameters**

- **buffer** (*numpy.array*) A screen buffer to render the item into.
- row (int) The row to render in.
- **column** (*int*) The column to render in.
- **height** (*int*) The total height of the display buffer.
- width (int) The total width of the display buffer.

## property screen\_column: int

A property to get/set the screen column.

```
Parameters
```

**value** (*int*) – the screen column

# Return type

int

## property screen\_row: int

A property to get/set the screen row.

#### **Parameters**

**value** (int) – the screen row

## Return type

int

# serialize()

Serialize a Sprixel into a dictionary.

New in version 1.3.0.

# Returns

The class as a dictionary

# Return type

dict

## Example:

```
json.dump( sprixel.serialize() )
```

### $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

#### classmethod white\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.WHITE\_RECT. The difference is that WHITE\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.white_rect()
```

## classmethod white\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.WHITE\_SQUARE. The difference is that WHITE\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

Example:

```
sprixel = Sprixel.white_square()
```

## classmethod yellow\_rect()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.YELLOW\_RECT. The difference is that YELLOW\_RECT is a string and this one is a Sprixel that can be manipulated more easily.

Note: Yellow is often rendered as brown.

Example:

```
sprixel = Sprixel.yellow_rect()
```

# classmethod yellow\_square()

This class method returns a sprixel that is the equivalent of pygamelib.assets.graphics.YELLOW\_SQUARE. The difference is that YELLOW\_SQUARE is a string and this one is a Sprixel that can be manipulated more easily.

**Note:** Yellow is often rendered as brown.

Example:

```
sprixel = Sprixel.yellow_square()
```

#### **PartitionParticle**

```
class pygamelib.gfx.particles.PartitionParticle(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, partition: list = None, partition\_blending\_table: list = None)
```

Bases: *Particle*New in version 1.3.0.

The PartitionParticle is a more precise *Particle*. Its main difference is that it is additive. This means that the PartitionParticle posess the ability to complement a sprixel that is already drawn. Or to add to a sprixel that is already drawn.

As a matter of facts, the primary goal of the PartitionParticle is to modify an already drawn sprixel to improve the visuals/graphical effects.

For example, if two particles occupy the same space on screen, with a regular *Particle* the last to render is the one that will be displayed. If one particle is represented by "and the other by ", only the second will be displayed.

In the case of PartitionParticles, an addition of the 2 sprixels will be displayed! So in the previous example the addition of the 2 particles would result in "because " + " = ".

It comes at a cost though as the PartitionParticle is slower to render than the Particle class.

The partition particle achieve that by using a partition and a blending table. The blending table is crucial for the performances to be not too catastrophic. The size of the blending table is directly linked to the performances of the PartitionParticle (the bigger the blending table the slower the rendering).

The blending table is a dictionnary of strings that covers all possible operations.

Example:

```
partition_blending_table = {
    gb.QUADRANT_UPPER_LEFT
    + gb.QUADRANT_UPPER_RIGHT: gb.UPPER_HALF_BLOCK,
    gb.QUADRANT_UPPER_LEFT + gb.QUADRANT_LOWER_LEFT: gb.LEFT_HALF_BLOCK,
    gb.QUADRANT_UPPER_LEFT
    + gb.QUADRANT_LOWER_RIGHT: gb.QUADRANT_UPPER_LEFT_AND_LOWER_RIGHT,
    # it goes on for many lines...
}
```

By default, the PartitionParticle has a blending table that is using the UTF8 Blocks.QUADRANT\_\* characters. If you want to use a different one, you need to define a new blending table and pass it as parameter to the constructor.

The partition itself is a 2x2 array that contains the 4 quadrants of a character displayed in the terminal.

As an example, if a full character were a block: "the partition would be: [[", "], [", "]].

You can conceive the partition as the exploded version of the character/sprixel and the blending table as the rules to blend them together.

The PartitionParticle can also be used to create reinforcement effects. For example, if the partition is composed solely of "and the partition table only define one rule: "+" = ". It is a powerful particle that can be used to create a lot of different effects.

**Important:** A limit of the current implementation is that the partition table must be a 2x2 array. It cannot be otherwise. Even if all the quadrants are the same.

**\_\_init\_\_**(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, partition: list = None, partition\_blending\_table: list = None)  $\rightarrow$  None

The constructor takes the following parameters.

#### **Parameters**

- $\mathbf{row}(int)$  The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.
- partition (list) The 2x2 array that defines the partition of the sprixel.
- partition\_blending\_table (list) The blending table that defines the rules to blend the 2 sprixels.

## Example:

# **Methods**

init([row, column, velocity, lifespan,])	The constructor takes the following parameters.
apply_force(force)	Apply a force to the particle's acceleration vector.
	* * *
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
finished()	Return True if the particle is done living (i.e its lifespan is lesser or equal to 0).
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a PartitionParticle from a dictionary.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
render([sprixel])	This method first calls the Particle.render() method.
<pre>reset([row, column, velocity, lifespan])</pre>	Reset a particle in its initial state.
reset_lifespan([lifespan])	Reset the particle lifespan (including the initial lifespan).
serialize()	Serialize a PartitionParticle into a dictionary.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
terminate()	Terminate a particle, i.e sets its lifespan to -1.
update()	This method first calls the Particle.update() method, then calculates the quadrant position, i.e: the actual position of the particle within a console character.

# **Attributes**

column	Access and set the column property.
row	Access and set the row property.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property.
у	Access and set the y property.

# $apply_force(force: Vector2D) \rightarrow None$

Apply a force to the particle's acceleration vector.

You are more likely to apply forces to all particles of an emitter through the apply\_force() method of the emitter class.

## **Parameters**

**force** (*Vector2D*) – The force to apply.

# Example:

```
gravity = Vector2D(-0.2, 0.0)
my_particle.apply_force(gravity)
```

# attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

#### Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## property column

Access and set the column property. Equivalent to the x property.

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

# $finished() \rightarrow bool$

Return True if the particle is done living (i.e its lifespan is lesser or equal to 0). It returns False otherwise.

# Return type

bool

## Example:

```
if not my_particle.finished():
    my_particle.update()
```

# handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

#### **Parameters**

• **subject** (*PglBase0bject*) – The object that has changed.

3.7. qfx 735

- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- **value** (*Any*) The new value of the attribute. This can be None.

## classmethod load(data)

Load a PartitionParticle from a dictionary.

#### **Parameters**

**data** (*dict*) – The dictionary to load from

#### Returns

The loaded PartitionParticle

# Return type

**PartitionParticle** 

#### Example:

```
particle = PartitionParticle.load( json.load( open("particle.json") )
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

#### **Parameters**

- **modifier** (*Pg1BaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

# render(sprixel: Sprixel = None)

This method first calls the Particle.render() method. Then it updates the rendered particle's model based on the blending table.

# **Parameters**

**sprixel** (*Sprixel*) – A sprixel already rendered in the screen buffer.

# Example:

```
p = my_particle
buffer[p.row][p.column] = p.render(buffer[p.row][p.column])
```

```
reset(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None)
```

Reset a particle in its initial state. This is particularly useful for the reuse of particles.

This method takes almost the same parameters than the constructor.

#### **Parameters**

• **row** (*int*) – The initial row position of the particle on the screen.

- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.

Example:

```
single_particle.reset(
    row=5,
    column=5,
    velocity=base.Vector2D(-0.5, 0.0),
    lifespan=10,
)
```

```
reset_lifespan(lifespan: int = 20) \rightarrow None
```

Reset the particle lifespan (including the initial lifespan).

## **Parameters**

**lifespan** (*int*) – The particle lifespan in number of movements/turns.

Example:

```
my_particle.reset_lifespan(10)
```

## property row

Access and set the row property. Equivalent to the y property.

# property screen\_column: int

A property to get/set the screen column.

## **Parameters**

value (int) – the screen column

## **Return type**

int

# property screen\_row: int

A property to get/set the screen row.

## **Parameters**

**value** (int) – the screen row

## Return type

int

# serialize()

Serialize a PartitionParticle into a dictionary.

### **Returns**

The class as a dictionary

# Return type

dict

Example:

```
json.dump( particle.serialize() )
```

```
store\_screen\_position(row: int, column: int) \rightarrow bool
```

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

### Example:

```
an_object.store_screen_coordinate(3,8)
```

#### $terminate() \rightarrow None$

Terminate a particle, i.e sets its lifespan to -1.

In that case the ParticleEmitter and ParticlePool will recycle it. That is *IF* you are managing the particle through an emitter and/or a pool of course.

Example:

```
p = my_particle
if p.row >= screen,height or p.column >= screen.width:
    p.terminate()
```

#### update()

This method first calls the Particle.update() method, then calculates the quadrant position, i.e: the actual position of the particle within a console character. It then updates the particle's model based on this internal position.

Example:

```
my_particle.update()
```

## property x

Access and set the x property. Equivalent to the column property.

## property y

Access and set the y property. Equivalent to the row property.

## **RandomColorParticle**

```
class pygamelib.gfx.particles.RandomColorParticle(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, sprixel: ParticleSprixel = None, color: Color = None)
```

Bases: Particle

This class is a *Particle* that has a random foreground color.

By default, if both the sprixel and color parameters are not specified, the model of the *Sprixel* is going to be '•' and the color will be randomly chosen.

You can also specify a color and a model.

```
__init__(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, sprixel: ParticleSprixel = None, color: Color = None) \rightarrow None
```

The constructor takes the following parameters.

#### **Parameters**

- **row** (*int*) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.
- **sprixel** (*Sprixel*) The sprixel that represent the particle when drawn on screen.
- **color** (*Color*) The color of the particle (if you want a specific color instead of a random one).

# Example:

```
single_particle = RandomColorParticle(
   row=5,
   column=5,
   velocity=base.Vector2D(-0.5, 0.0),
   lifespan=10,
)
```

## **Methods**

init([row, column, velocity, lifespan,])	The constructor takes the following parameters.
apply_force(force)	Apply a force to the particle's acceleration vector.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
finished()	Return True if the particle is done living (i.e its lifespan is lesser or equal to 0).
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a PartitionParticle from a dictionary.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
render([sprixel])	Render the particle as a Sprixel.
<pre>reset([row, column, velocity, lifespan])</pre>	Reset a particle in its initial state.
reset_lifespan([lifespan])	Reset the particle lifespan (including the initial lifespan).
serialize()	Serialize a RandomColorParticle into a dictionary.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
terminate()	Terminate a particle, i.e sets its lifespan to -1.
update()	The update method perform the calculations required
	to process the new particle position.

## **Attributes**

column	Access and set the column property.
row	Access and set the row property.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property.
У	Access and set the y property.

## $apply_force(force: Vector2D) \rightarrow None$

Apply a force to the particle's acceleration vector.

You are more likely to apply forces to all particles of an emitter through the apply\_force() method of the emitter class.

#### **Parameters**

**force** (Vector2D) – The force to apply.

## Example:

```
gravity = Vector2D(-0.2, 0.0)
my_particle.apply_force(gravity)
```

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

**observer** (*PglBaseObject*) – An observer to attach to this object.

## Returns

True or False depending on the success of the operation.

## Return type

bool

## Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## property column

Access and set the column property. Equivalent to the x property.

## detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

## **Parameters**

**observer** (*PglBaseObject*) – An observer to detach from this object.

#### Returns

True or False depending on the success of the operation.

### **Return type**

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

#### **finished()** $\rightarrow$ bool

Return True if the particle is done living (i.e its lifespan is lesser or equal to 0). It returns False otherwise.

# Return type

bool

Example:

```
if not my_particle.finished():
    my_particle.update()
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

## **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- attribute (str) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

## classmethod load(data)

Load a PartitionParticle from a dictionary.

#### **Parameters**

**data** (*dict*) – The dictionary to load from

## Returns

The loaded PartitionParticle

## Return type

**PartitionParticle** 

Example:

```
particle = RandomColorParticle.load( json.load( open("particle.json") ) )
```

**notify**(modifier=None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

## **Parameters**

• **modifier** (*Pg1BaseObject*) – An optional parameter that identify the modifier object to exclude it from the notified objects.

3.7. qfx 741

- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

# Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

## **render**(*sprixel*: Sprixel = *None*)

Render the particle as a *Sprixel*. This method is called by the *ParticleEmitter* render\_to\_buffer method.

It takes a *Sprixe1* as a parameter. This Sprixel is given by the ParticleEmitter.render\_to\_buffer() method and if it is not None, the particle will render itself into that *Sprixe1* and return it.

**Important:** This method must be called after everything else as rendered or else there will be *Sprixel* that will be overwritten during their rendering cycle. Other elements could also have their *Sprixel* corrupted and replaced by the particle's one.

#### **Parameters**

**sprixel** (*Sprixel*) – A sprixel already rendered in the screen buffer.

#### Example:

```
p = my_particle
buffer[p.row][p.column] = p.render(buffer[p.row][p.column])
```

**reset**(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None)

Reset a particle in its initial state. This is particularly useful for the reuse of particles.

This method takes almost the same parameters than the constructor.

#### **Parameters**

- row (int) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.

#### Example:

```
single_particle.reset(
    row=5,
    column=5,
    velocity=base.Vector2D(-0.5, 0.0),
    lifespan=10,
)
```

```
reset_lifespan(lifespan: int = 20) \rightarrow None
     Reset the particle lifespan (including the initial lifespan).
         Parameters
             lifespan (int) – The particle lifespan in number of movements/turns.
     Example:
     my_particle.reset_lifespan(10)
property row
     Access and set the row property. Equivalent to the y property.
property screen_column: int
     A property to get/set the screen column.
         Parameters
             value (int) – the screen column
         Return type
             int
property screen_row: int
     A property to get/set the screen row.
         Parameters
             value (int) – the screen row
         Return type
             int
serialize()
     Serialize a RandomColorParticle into a dictionary.
             The class as a dictionary
         Return type
             dict
     Example:
     json.dump( particle.serialize() )
store\_screen\_position(row: int, column: int) \rightarrow bool
     Store the screen position of the object.
     This method is automatically called by Screen.place().
         Parameters
```

3.7. gfx 743

• row (int) – The row (or y) coordinate.

an\_object.store\_screen\_coordinate(3,8)

Example:

• **column** (*int*) – The column (or x) coordinate.

```
terminate() \rightarrow None
```

Terminate a particle, i.e sets its lifespan to -1.

In that case the ParticleEmitter and ParticlePool will recycle it. That is *IF* you are managing the particle through an emitter and/or a pool of course.

Example:

```
p = my_particle
if p.row >= screen,height or p.column >= screen.width:
    p.terminate()
```

```
update() \rightarrow None
```

The update method perform the calculations required to process the new particle position. It mainly adds the acceleration to the velocity vector and update the position accordingly.

After calling update() the acceleration is "consumed" in the velocity and therefor reset.

The update() method takes no parameters and returns nothing.

Example:

```
my_particle.update()
```

#### property x

Access and set the x property. Equivalent to the column property.

## property y

Access and set the y property. Equivalent to the row property.

## **RandomColorPartitionParticle**

Bases: PartitionParticle

This class is basically the same as *RandomColorParticle* but its base class is *PartitionParticle* instead of *Particle*. Everything else is the same.

```
__init__(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None, partition: list = None, partition_blending_table: list = None, color: Color = None) \rightarrow None
```

The constructor takes the following parameters.

## **Parameters**

- row (int) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.
- partition (list) The partition of the particle.
- partition\_blending\_table (list) The blending table of the particle.

• **color** (*Color*) – The color of the particle (if you want a specific color instead of a random one).

# Example:

```
single_particle = RandomColorPartitionParticle(
   row=5,
   column=5,
   velocity=base.Vector2D(-0.5, 0.0),
   lifespan=10,
)
```

# **Methods**

init([row, column, velocity, lifespan,])	The constructor takes the following parameters.
apply_force(force)	Apply a force to the particle's acceleration vector.
attach(observer)	Attach an observer to this instance.
detach(observer)	Detach an observer from this instance.
finished()	Return True if the particle is done living (i.e its lifes-
()	pan is lesser or equal to 0).
<pre>handle_notification(subject[, attribute, value])</pre>	A virtual method that needs to be implemented by the observer.
load(data)	Load a RandomColorPartitionParticle from a dictionary.
<pre>notify([modifier, attribute, value])</pre>	Notify all the observers that a change occurred.
render([sprixel])	This method first calls the Particle.render() method.
<pre>reset([row, column, velocity, lifespan])</pre>	Reset a particle in its initial state.
reset_lifespan([lifespan])	Reset the particle lifespan (including the initial lifespan).
serialize()	Serialize a RandomColorPartitionParticle into a dictionary.
<pre>store_screen_position(row, column)</pre>	Store the screen position of the object.
terminate()	Terminate a particle, i.e sets its lifespan to -1.
update()	This method first calls the Particle.update() method, then calculates the quadrant position, i.e: the actual position of the particle within a console character.

# **Attributes**

column	Access and set the column property.
row	Access and set the row property.
screen_column	A property to get/set the screen column.
screen_row	A property to get/set the screen row.
X	Access and set the x property.
У	Access and set the y property.

 $apply\_force(force: Vector2D) \rightarrow None$ 

Apply a force to the particle's acceleration vector.

You are more likely to apply forces to all particles of an emitter through the apply\_force() method of the emitter class.

#### **Parameters**

```
force (Vector2D) – The force to apply.
```

Example:

```
gravity = Vector2D(-0.2, 0.0)
my_particle.apply_force(gravity)
```

## attach(observer)

Attach an observer to this instance. It means that until it is detached, it will be notified every time that a notification is issued (usually on changes).

An object cannot add itself to the list of observers (to avoid infinite recursions).

## **Parameters**

```
observer (PglBaseObject) – An observer to attach to this object.
```

#### Returns

True or False depending on the success of the operation.

# **Return type**

bool

Example:

```
myboard = Board()
screen = Game.instance().screen
# screen will be notified of all changes in myboard
myboard.attach(screen)
```

## property column

Access and set the column property. Equivalent to the x property.

# detach(observer)

Detach an observer from this instance. If observer is not in the list this returns False.

#### **Parameters**

```
observer (PglBaseObject) – An observer to detach from this object.
```

#### Returns

True or False depending on the success of the operation.

# Return type

bool

Example:

```
# screen will no longer be notified of the changes in myboard.
myboard.detach(screen)
```

## $finished() \rightarrow bool$

Return True if the particle is done living (i.e its lifespan is lesser or equal to 0). It returns False otherwise.

# Return type

bool

Example:

```
if not my_particle.finished():
   my_particle.update()
```

## handle\_notification(subject, attribute=None, value=None)

A virtual method that needs to be implemented by the observer. By default it does nothing but each observer needs to implement it if something needs to be done when notified.

This method always receive the notifying object as first parameter. The 2 other parameters are optional and can be None.

You can use the attribute and value as you see fit. You are free to consider attribute as an event and value as the event's value.

### **Parameters**

- **subject** (*PglBaseObject*) The object that has changed.
- **attribute** (*str*) The attribute that has changed, it is usually a "FQDN style" string. This can be None.
- value (Any) The new value of the attribute. This can be None.

## classmethod load(data)

Load a RandomColorPartitionParticle from a dictionary.

#### **Parameters**

**data** (*dict*) – The dictionary to load from

#### Returns

The loaded RandomColorPartitionParticle

## Return type

RandomColorPartitionParticle

## Example:

**notify**(modifier = None, attribute: str = None, value: Any = None)  $\rightarrow$  None

Notify all the observers that a change occurred.

## **Parameters**

- **modifier** (*PglBaseObject*) An optional parameter that identify the modifier object to exclude it from the notified objects.
- **attribute** (*str*) An optional parameter that identify the attribute that has changed.
- value (Any) An optional parameter that identify the new value of the attribute.

## Example:

```
# This example is silly, you would usually notify other objects from inside
# an object that changes a value that's important for the observers.
color = Color(255,200,125)
color.attach(some_text_object)
color.notify()
```

```
render(sprixel: Sprixel = None)
```

This method first calls the Particle.render() method. Then it updates the rendered particle's model based on the blending table.

#### **Parameters**

**sprixel** (*Sprixel*) – A sprixel already rendered in the screen buffer.

Example:

```
p = my_particle
buffer[p.row][p.column] = p.render(buffer[p.row][p.column])
```

```
reset(row: int = 0, column: int = 0, velocity: Vector2D = None, lifespan: int = None)
```

Reset a particle in its initial state. This is particularly useful for the reuse of particles.

This method takes almost the same parameters than the constructor.

#### **Parameters**

- **row** (*int*) The initial row position of the particle on the screen.
- **column** (*int*) The initial column position of the particle on the screen.
- **velocity** (*Vector2D*) The initial velocity of the particle.
- **lifespan** (*int*) The particle lifespan in number of movements/turns. A particle with a lifespan of 3 will move for 3 turns before being finished.

Example:

```
single_particle.reset(
    row=5,
    column=5,
    velocity=base.Vector2D(-0.5, 0.0),
    lifespan=10,
)
```

# **reset\_lifespan**(lifespan: int = 20) $\rightarrow$ None

Reset the particle lifespan (including the initial lifespan).

#### **Parameters**

**lifespan** (*int*) – The particle lifespan in number of movements/turns.

Example:

```
my_particle.reset_lifespan(10)
```

## property row

Access and set the row property. Equivalent to the y property.

# property screen\_column: int

A property to get/set the screen column.

## **Parameters**

value (int) - the screen column

## Return type

int

#### property screen\_row: int

A property to get/set the screen row.

## **Parameters**

**value** (int) – the screen row

## **Return type**

int

## serialize()

Serialize a RandomColorPartitionParticle into a dictionary.

#### **Returns**

The class as a dictionary

## Return type

dict

Example:

```
json.dump( particle.serialize() )
```

## $store\_screen\_position(row: int, column: int) \rightarrow bool$

Store the screen position of the object.

This method is automatically called by Screen.place().

#### **Parameters**

- row (int) The row (or y) coordinate.
- **column** (*int*) The column (or x) coordinate.

Example:

```
an_object.store_screen_coordinate(3,8)
```

# $\textbf{terminate()} \rightarrow None$

Terminate a particle, i.e sets its lifespan to -1.

In that case the ParticleEmitter and ParticlePool will recycle it. That is *IF* you are managing the particle through an emitter and/or a pool of course.

Example:

```
p = my_particle
if p.row >= screen,height or p.column >= screen.width:
    p.terminate()
```

#### update()

This method first calls the Particle.update() method, then calculates the quadrant position, i.e: the actual position of the particle within a console character. It then updates the particle's model based on this internal position.

Example:

```
my_particle.update()
```

## property x

Access and set the x property. Equivalent to the column property.

3.7. qfx 749

## property y

Access and set the y property. Equivalent to the row property.

# 3.8 Credits

# 3.8.1 Development Lead

• Arnaud Dupuis (@arnauddupuis)

# 3.8.2 Contributors

- Kalil de Lima (@kaozdl)
- Muhammad Syuqri (@Dansyuqri)
- Ryan Brown (@grimmjow8)
- Chase Miller (@Arekenaten)
- Gunjan Rawal (@gunjanraval)
- Anshul Choudhary (@achoudh5)
- Raymond Beaudoin (@synackray)
- Felipe Rodrigues (@fbidu)
- Bastien Wirtz (@bwirtz)
- Franz Osorio (@f-osorio)
- Guillermo Eijo (@guilleijo)
- Diego Cáceres (@diego-caceres)
- Spassarop (@spassarop)
- Javier Hernán Caballero García (@caballerojavier13)
- Olle Lögdahl (@ollelogdahl)
- MaryEtta Morris (@morrme)
- Peter Szabo (@szabopeter)
- Frans Ramirez (@Frans06)
- Krunal Rank (@KRHero03)
- Juan Picca (@jumapico)
- Harshini (@harshiniwho)
- Tammysalmon (@tammysalmon)
- JayC (@jayc13)
- Rikil Gajarla (@RikilG)
- Melsaa (@melsaa)

# 3.9 Release notes

# 3.9.1 1.3.0 (2022-10-07)

This release is massive. Please read the documentation for specific changes to classes. It is available at https://pygamelib.readthedocs.io/en/latest/index.html.

**Important one:** the whole pygamelib has been migrated to its own Github organization: https://github.com/pygamelib please update your links! The library's repository is now available at https://github.com/pygamelib/pygamelib.

# Main updates

- New feature: A lot of new tools have been developed for the library and are all available on the organization's Github: https://github.com/pygamelib.
- New feature: The pygamelib.engine.Screen class now has a new Improved Screen Management double buffered system. This set of methods allow for a simplified management of the console screen. It is also faster than the Legacy Direct Display system. Please read the documentation (https://pygamelib.readthedocs.io/en/latest/pygamelib.engine.Screen.html) and the wiki on the Github repository for more about the differences. You will probably want to switch to the new stack as soon as possible. Both systems are clearly identified in the documentation by visible tags. Most of the new features of this release are NOT compatible with the Legacy Direct Display system. It still received updates and new features but will probably be deprecated in future updates.
- New feature: Introducing the pygamelib.gfx.ui module! The beginning of a module for all your game/application user interface needs. The module is in alpha for the moment, feel free to voice your feedback. This module is only compatible with the **Improved Screen Management**.
- New feature: A new tool has been added to the library: pgl-sprite-editor. An editor to create or edit sprites and sprite based animations.
- New feature: pygamelib.engine.Game can now be created as a Singleton through the instance() method.
- New feature: Add a particle system to the library! It includes a number of new classes that are located in the
  pygamelib.gfx.particles submodule. This module is only compatible with the Improved Screen Management.
- New feature: introducing pygamelib.gfx.core.Font, a Sprite based font system. This release come with an "8bits" font and a couple of font imported from FIGlet!
- New feature: Add a Color class (pygamelib.gfx.core.Color) to entirely abstract the color system.
- New feature: All objects can now be properly serialized and loaded through a streamlined process. Look for the *serialize()* and *load()* methods.
- **New feature:** New base object pygamelib.base.PglBaseObject, all objects that inherits from python's object are now inheriting from this new one. It implements a couple of base features but the most important is the modified *Observer* design pattern that is the base of a refactoring to event base communication within the library.
- New feature: Added a new board item: pygamelib.board\_item.Camera. It is a specific item that is not shown on the board. It can be used for cinematic for example. Please read the documentation for more information.
- New feature/improvement: The *Board* object has been reworked to allow for a third dimension. It now has a new property called *layer*. Layers are automatically added and removed to fit the need of overlapping items. *Board.place\_item()* also accept a new layer parameter to set the layer (if you want to put stuff over the player for example). An example is visible here: https://www.youtube.com/watch?v=9cOt63ZAJOk.
- *Improvement:* Most resources intensive array/list have been replaced by numpy arrays. This brings better performances for pygamelib.engine.Board and for pygamelib.engine.Screen.

3.9. Release notes 751

- *Improvement*: Add a new algorithm to the PathFinder actuator: A\*.
- Improvement: pygamelib.gfx.core.Sprite can now be tinted or modulated with a color. Both operation do the same thing: change the color of the sprite by applying a color at a given ratio. However, tint() returns a new sprite and does not modify the original sprite while modulate() returns nothing and modify the sprite directly.

# **Breaking changes**

- pygamelib.board\_items.BoardItem constructor parameter changed: type is now item\_type.
- pygamelib.board\_items.BoardItem: there was a conflict with inventory\_space. It was defined both as a property and a method. The method has been removed and *BoardItem.inventory\_space* is now a proper python property. Concretely: you might have to remove parenthesis when using any\_item.inventory\_space (vs the old any\_item.inventory\_space()).
- The new pygamelib.gfx.core.Color replaces Terminal.on\_color\_rgb() and Terminal.color\_rgb(). It is much easier to use (just use the Color object and the pygamelib will manage foreground and background differences) but it requires to change the initialization of every Sprixel and Text of your game (sorry...).
- When using the new Improved Screen Management stack and partial display at the same time, you now have to set Board.partial\_display\_focus. It is not breaking anything in existing code but it will not behave as you want is you just Screen.place() your board (that uses partial display) without setting the partial\_display\_focus to the player first.

## Other changes

- *Improvement*: pgl-editor now uses Sprixels instead of regular characters allowing for more possible customization and features in the Board and Screen.
- *Improvement*: in pgl-editor it is now possible to generate a random color in the color editor.
- *Improvement*: All actuators now return pygamelib.constants.NO\_DIR if there is no direction available to next\_move(). This makes the actuators behavior more consistent particularly when they are overloaded.
- *Improvement*: The RandomActuator behavior has been reworked. It now choose a direction and follow it for a certain distance before choosing a new direction. It also detect when it is stuck an, in that case, pick a new direction.
- *Improvement*: Add display\_sprite() and display\_sprite\_at() method to Screen. These methods can display a pygamelib.gfx.core.Sprite on screen.
- *Improvement*: Inventory has been improved to be more versatile and less limited. It now behaves like an enhanced list of objects. A rudimentary constraints system was added (for example to limit the number of certain types of items). The new inventory is also fully plugged into the observer/notifications system.
- Improvement: All BoardItem now have configurable properties for restorable, overlappable, pickable and can\_move.
- *Improvement*: pygamelib.board\_items.BoardComplexItem.sprite is now a @property instead of a class variable. That property automatically call update\_sprite().
- Improvement: When Game.mode is set to pygamelib.constants.MODE\_RT, all pygamelib.board\_items. Movable now accumulate movement vectors (when using vectors). This means that non unit movement patterns are now possible.
- Improvement: The new pygamelib.base.Console implements a Singleton design pattern. You can now get a unique reference to the blessed.Terminal (the object wrapped in Console) object by calling Console.instance().

- Fixed a bug in pygamelib.engine.Screen.display\_at(): it was not possible to display anything after (below a Board). It is now possible.
- *Improvement*: pygamelib.base.Text has improved a lot. It can now use the Font system, has new attributes and is now a *PglBaseObject*. Please read the documentation for more.
- *Improvement*: Sprixels and Sprites now have their own deepcopy operator: Sprixel.copy() and Sprite.copy().
- *Improvement*: It is now possible to set the transparency of all sprixels of a sprite by using Sprite. set\_transparency().
- Fixed a bug with *restorable* items: now all board items can be set to be restorable.
- Fixed a bug in pgl-editor when editing large boards that require partial display. The viewport was not correctly restored.
- Fixed issues with the library's inheritance graph.
- Fixed a bug in pygamelib.engine.Game where the partial display settings (when set at in the Game instance), were not correctly passed down to the Board.
- Fixed the sphinx dependencies (for building the doc).
- Fixed the mess in the sphinx files to generate the documentation.
- Fixed an issue with linting dependencies.
- Removed legacy files from older version of the library.

I would like to thank all the contributors (https://pygamelib.readthedocs.io/en/latest/authors.html) for their work on this massive update.

The new pygamelib logo was done by an awesome artist: Jack Tseng (https://hellojacktseng.carrd.co/ https://twitter.com/HelloJackTseng) please have a look at their amazing work!!

## 3.9.2 1.2.3 (2020-09-01)

Emergency release: fix a regression introduced by v1.2.2.

### 3.9.3 1.2.2 (2020-09-01)

- Fix issue with imports for Python 3.6
- Fix an issue with the way pygamelib.engine.Screen test the terminal on Windows.

### 3.9.4 1.2.0 (2020-08-29)

- Renamed the entire library from hac-game-lib to pygamelib.
- \*Breaking change:\* The library has been heavily refactored and this creates some issues. Please have a look at the migration notes
- New feature: Items that can be represented on more than one cell. We call them complex items. There's a lot of new complex items: ComplexPlayer and ComplexNPC of course, but also ComplexWall, ComplexDoor, ComplexTreasure and the general purpose Tile object.
- New feature: Going, with complex item we now have a proper sprite system with the gfx.core.Sprite class.

3.9. Release notes 753

- New feature: In addition to the regular model we now have a new concept: the Sprixel. A Sprite is made of many Sprixels.
- New feature: New JSON based file format to save, load and distribute sprites and/or sprixels.
- New feature: All these sprites can be grouped into a SpriteCollection that in turn can be saved in our new sprite file format.
- New feature: New Math library. This one starts small but will grow. It makes calculating the distance and intersections easier.
- New feature: New Vector2D class to represent forces and movement as a vector. It is now possible to give a vector to the move() method.
- New feature: Gave some love to text. There are now 2 objects dedicated to text: base. Text to manipulate text and board\_items. TextItem to easily place text on a board.
- New feature: A Screen object has been added to make the screen manipulation simpler.
- New feature: The Game object now has a run() method that act as the main game loop. It calls a user defined update function and takes care of a lot of things. It runs until the Game.state is set to STOPPED.
- New feature: The Game object can now turn by turn or real time. All movables can be configured to have time based or turn based movement speed.
- Improvement: The Animation class now support both regular strings (models), Sprixel and Sprite.
- *Improvement*: All complex items obviously support (actually requires) sprites but all regular board items now supports sprixels.
- *Improvement*: Test coverage dramatically improved. It has jumped from 25% to 98%.
- *Improvement*: Lots of objects now have attributes to easily access and/or set properties like position (mostly read only), width, height, etc.
- *Improvement*: Converted the editor to pygamelib and renamed it pgl-editor.py. Also added a multi page selector and integrated the new graphic assets.
- Improvement: All movables can now have different vertical and horizontal "steps" parameters.
- Cleaned up the repository (it was becoming seriously messy).
- Change the prefix of all exceptions from HAc to Pgl.
- Added a NO\_PLAYER constant to tell the game object that he should not expect a player object.
- Improve the generated documentation.
- Various improvements in exceptions raising across the library. Please see the documentation (that was also updated).
- Various bug fixing in the Suparex example.

I also need to give some kudos to the kids of the Hyrule Astronomy Club for thorough testing of Suparex. They found well hidden bug and exploitable bugs. Special thanks to Arthur who found many glitches. Congratulations to Arthur and Hadrien that successfully exploited them to achieve extremely high scores (up to 12000!!!).

### 3.9.5 1.1.1 (2020-07-18)

- Fix a bug in hgl-editor: when using previously recorded parameters to create a board the editor was crashing.
- *Improvement*: Automatically enable partial display and map bigger than 40x40.
- Fix a bug a coordinates in Board.item()

## 3.9.6 1.1.0 (2020-06-12)

- Fix many issues with strings all across the library.
- Fix many issues with variables interpolation in exceptions.
- Fix a bug in Game.load\_board() that was causing corruptions.
- Fix multiple typos in the documentation.
- · Fix an issue with the user directory in hgl-editor
- Fix many issues with the PatrolActuator.
- New feature: partial display (dynamically display only a part of a board)
- New feature: new mono directional actuator.
- New feature: projectiles (can be sent and completely managed by the game object)
- New feature: new assets module to hold many non core submodules.
- New feature: Assets.Graphics that add thousands of glyphs (including emojis) to the current capacities of the library.
- New feature: Add support for PatrolActuator in hgl-editor.
- New feature: Add support for PathFinder actuator in hgl-editor.
- New feature: Add an object parent system.
- New feature: Add a configuration system to hgl-editor.
- Improvement: Add full configuration features to the Game object.
- *Improvement*: Add a new example in the form of a full procedural generation platform game (see examples/suparex).
- *Improvement*: Improved performances particularly around the features that relies on Board.place\_item(). Up to 70 times faster.
- Improvement: It is now possible to specify the first frame index in Animation.
- *Improvement*: Formatted all the code with black.
- Improvement: PathFinder.add\_waypoint() now sets the destination if it wasn't set before.

3.9. Release notes 755

## 3.9.7 1.0.1 (2020-05-17)

- Fix a huge default save directory issue (see complete announcement) in hgl-editor.
- Fix lots of strings in hgl-editor.
- Fix a type issue in the Inventory class for the not\_enough\_space exception.
- Improve Board.display() performances by 15% (average).

## 3.9.8 1.0.0 (2020-03-20)

- Add AdvancedActuators.PathFinder @arnauddupuis
- Add test cases for BoardItem @grimmjow8 @Arekenaten
- Add test cases for Board @grimmjow8 @Arekenaten
- Add support to load files from the directories in directories.json @kaozdl
- Add a new SimpleActuators.PatrolActuator @kaozdl
- Add Animation capabilities @arnauddupuis
- Improve navigation in hgl-editor by using arrow keys @bwirtz
- Improve selection of maps in hgl-editor @gunjanraval @kaozdl
- Improve documentation for SimpleActuators.PathActuator @achoudh5
- Improve documentation for launching the test suite @bwirtz
- Migration from pip install to pipenv @kaozdl
- Fix board saving bug in hgl-editor @gunjanraval
- Fix back menu issues in hgl-editor @synackray
- Fix README and setup.py @fbidu
- Make the module compatible with Flake8: @bwirtz @arnauddupuis @kaozdl @f-osorio @guilleijo @diego-caceres @spassarop
- CircleCI integration @caballerojavier13 @bwirtz

### 3.9.9 2019.5

• Please see the official website.

### 3.9.10 pre-2019.5

• Please see the Github for history.

# CHAPTER

# **FOUR**

# **INDICES AND TABLES**

- genindex
- modindex
- search

# **PYTHON MODULE INDEX**

р

pygamelib.constants, 532

760 Python Module Index

# **INDEX**

Symbols	init() (pygamelib.board_items.Character
init() (pygamelib.actuators.Actuator method), 7	method), 331
init() (pygamelib.actuators.Behavioral method),	init() (pygamelib.board_items.ComplexDoor
11	method), 341
init() (pygamelib.actuators.PathActuator	init()
method), 14	method), 352
init() (pygamelib.actuators.PathFinder method),	init() (pygamelib.board_items.ComplexPlayer
23	method), 363
init() (pygamelib.actuators.PatrolActuator	init() (pygamelib.board_items.ComplexTreasure
method), 19	method), 374
init() (pygamelib.actuators.RandomActuator	init() (pygamelib.board_items.ComplexWall
method), 30	method), 385
init() (pygamelib.actuators.UnidirectionalActuator	init() (pygamelib.board_items.Door method), 396
method), 34	init() (pygamelib.board_items.GenericActionableStructure
init() (pygamelib.assets.graphics.Blocks method),	method), 406
41	init() (pygamelib.board_items.GenericStructure
init() (pygamelib.assets.graphics.BoxDrawings	method), 424
method), 47	init() (pygamelib.board_items.GenericStructureComplexComponen
init() (pygamelib.assets.graphics.GeometricShapes	method), 415
method), 59	init() (pygamelib.board_items.Immovable
init() (pygamelib.assets.graphics.Models	method), 434
method), 129	init() (pygamelib.board_items.Movable method),
init() (pygamelib.base.History method), 241	443
init() (pygamelib.base.Math method), 243	init() (pygamelib.board_items.NPC method), 453
init() (pygamelib.base.PglBaseObject method),	init() (pygamelib.board_items.Player method), 464
245	init() (pygamelib.board_items.Projectile method),
init() (pygamelib.base.Text method), 249	IIII c() (pygameuo.boara_nems.Frojectue meinoa), 474
init() (pygamelib.base.Vector2D method), 258	init() (pygamelib.board_items.TextItem method),
init() (pygamelib.board_items.Actionable	486
method), 261	init() (pygamelib.board_items.Tile method), 497
init() (pygamelib.board_items.ActionableTile	init() (pygamelib.board_items.Treasure method),
method), 271	508
init() (pygamelib.board_items.BoardComplexItem	init() (pygamelib.board_items.Wall method), 518
method), 282	init() (pygamelib.engine.Board method), 532
init() (pygamelib.board_items.BoardItem method), 303	init() (pygamelib.engine.Game method), 544
ini+ () (nyamalih hoard itams RoardItamComplex(	Compinitat () (pygamelib.engine.Inventory method), 559
method), 293	init() (pygamelib.engine.Screen method), 570
init() (pygamelib.board_items.BoardItemVoid	init() (pygamelib.gfx.core.Animation method),
method), 313	581
init() (pygamelib.board_items.Camera method),	init() (pygamelib.gfx.core.Color method), 612
322	init() (pygamelib.gfx.core.Font method), 585

init() (pygamelib.gfx.core.Sprite method), 594 init() (pygamelib.gfx.core.SpriteCollection	680init() (pygamelib.gfx.ui.UiConfig method), 683
method), 589	init() (pygamelib.gfx.ui.Widget method), 684
init() (pygamelib.gfx.core.Sprixel method), 603	٨
init() (pygamelib.gfx.particles.CircleEmitter	A
method), 689	A_BUTTON_BLOOD_TYPE
init() (pygamelib.gfx.particles.ColorParticle	(pygamelib.assets.graphics.Models attribute),
method), 695	182
init() (pygamelib.gfx.particles.ColorPartitionPartic	
method), 701	(pygamelib.assets.graphics.Models attribute),
init() (pygamelib.gfx.particles.EmitterProperties method), 707	181
init() (pygamelib.gfx.particles.Particle method),	ABACUS (pygamelib.assets.graphics.Models attribute),
	181
init() (pygamelib.gfx.particles.ParticleEmitter	AC_CURRENT (pygamelib.assets.graphics.MiscTechnicals
method), 709	attribute), 84
init() (pygamelib.gfx.particles.ParticlePool	ACCORDION (pygamelib.assets.graphics.Models at-
method), 715	tribute), 181
init() (pygamelib.gfx.particles.ParticleSprixel	action (pygamelib.gfx.ui.MenuAction property), 668
method), 723	Actionable (class in pygamelib.board_items), 261 ActionableTile (class in pygamelib.board_items), 271
init() (pygamelib.gfx.particles.PartitionParticle	activate() (pygamelib.board_items.Actionable
method), 732	method), 263
init() (pygamelib.gfx.particles.RandomColorPartic	leactivate() (pygamelib.board_items.ActionableTile
method), 738	method), 273
init() (pygamelib.gfx.particles.RandomColorPartite	method), 273 ioaEarwalee() (pygamelib.board_items.GenericActionableStructur
method), 744	method), 407
init() (pygamelib.gfx.ui.Box method), 618	activate() (pygamelib.gfx.ui.Menu method), 664
init() (pygamelib.gfx.ui.BoxLayout method), 619	activate() (pygamelib.gfx.ui.MenuAction method), 668
init() (pygamelib.gfx.ui.ColorPicker method), 626	active (pygamelib.gfx.particles.CircleEmitter property),
init() (pygamelib.gfx.ui.ColorPickerDialog	690
method), 624	active (pygamelib.gfx.particles.ParticleEmitter prop-
init() (pygamelib.gfx.ui.Cursor method), 627	erty), 710
init() (pygamelib.gfx.ui.Dialog method), 632	actuate_npcs() (pygamelib.engine.Game method), 546
init() (pygamelib.gfx.ui.FileDialog method), 633	actuate_projectiles() (pygamelib.engine.Game
init() (pygamelib.gfx.ui.FormLayout method), 635	method), 547
init() (pygamelib.gfx.ui.GridLayout method), 640	Actuator (class in pygamelib.actuators), 7
init() (pygamelib.gfx.ui.GridSelector method),	add() (pygamelib.base.History method), 242
647	add() (pygamelib.gfx.core.SpriteCollection method), 590
init() (pygamelib.gfx.ui.GridSelectorDialog	add_board() (pygamelib.engine.Game method), 547
method), 645	add_constraint() (pygamelib.engine.Inventory
init() (pygamelib.gfx.ui.Layout method), 649	method), 560
init() (pygamelib.gfx.ui.LineInput method), 654	<pre>add_directional_animation()</pre>
init() (pygamelib.gfx.ui.LineInputDialog method), 661	(pygamelib.board_items.Projectile method), 475
init() (pygamelib.gfx.ui.Menu method), 663	add_directional_model()
init() (pygamelib.gfx.ui.MenuAction method), 667	(pygamelib.board_items.Projectile method),
init() (pygamelib.gfx.ui.MenuBar method), 670	476
init() (pygamelib.gfx.ui.MessageDialog method),	add_entry() (pygamelib.gfx.ui.Menu method), 664
673	<pre>add_entry() (pygamelib.gfx.ui.MenuBar method), 671</pre>
init() (pygamelib.gfx.ui.MultiLineInputDialog	<pre>add_frame() (pygamelib.gfx.core.Animation method),</pre>
method), 676	582
init() (pygamelib.gfx.ui.ProgressBar method),	<pre>add_item() (pygamelib.engine.Inventory method), 561</pre>
678	<pre>add_line() (pygamelib.gfx.ui.MessageDialog method),</pre>
init() (pygamelib.gfx.ui.ProgressDialog method),	674

<pre>add_npc() (pygamelib.engine.Game method), 548 add_projectile() (pygamelib.engine.Game method),</pre>	181 ANGER_SYMBOL (pygamelib.assets.graphics.Models attribute), 181
add_row() (pygamelib.gfx.ui.FormLayout method), 636 add_waypoint() (pygamelib.actuators.PathFinder	ANGRY_FACE (pygamelib.assets.graphics.Models attribute), 181
<pre>method), 24 add_widget() (pygamelib.gfx.ui.BoxLayout method), 620</pre>	ANGRY_FACE_WITH_HORNS (pygamelib.assets.graphics.Models attribute), 181
<pre>add_widget() (pygamelib.gfx.ui.FormLayout method), 636</pre>	ANGUISHED_FACE (pygamelib.assets.graphics.Models attribute), 181
<pre>add_widget() (pygamelib.gfx.ui.GridLayout method), 641</pre>	<pre>animate_items() (pygamelib.engine.Game method), 549</pre>
add_widget() (pygamelib.gfx.ui.Layout method), 650 ADHESIVE_BANDAGE (pygamelib.assets.graphics.Models attribute), 181	Animation (class in pygamelib.gfx.core), 581 animation (pygamelib.board_items.Actionable prop- erty), 263
ADMISSION_TICKETS (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.ActionableTile property), 273
AERIAL_TRAMWAY (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.BoardComplexItem property), 284
AIRPLANE (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.BoardItem prop- erty), 305
AIRPLANE_ARRIVAL (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.BoardItemComplexComponent property), 294
AIRPLANE_DEPARTURE (pygamelib.assets.graphics.Models attribute), 181	
ALARM_CLOCK (pygamelib.assets.graphics.MiscTechnicals attribute), 84	
ALARM_CLOCK (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.Character prop- erty), 333
ALEMBIC (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.ComplexDoor prop- erty), 343
ALIEN (pygamelib.assets.graphics.Models attribute), 181 ALIEN_MONSTER (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.ComplexNPC prop- erty), 354 animation (pygamelib.board_items.ComplexPlayer
$\verb ALL_AROUND_PROFILE  (pygame lib. assets. graphics. Misc Teacher and the property of the pr$	chnicals property), 365
attribute), 84 ALL_CHARACTERS_AUTHORIZED	animation (pygamelib.board_items.ComplexTreasure property), 376
	animation (pygamelib.board_items.ComplexWall prop- erty), 387
ALL_MOVABLE_AUTHORIZED	animation (pygamelib.board_items.Door property), 398
(pygamelib.constants.Permission attribute), 530	animation (pygamelib.board_items.GenericActionableStructure property), 407
ALTERNATIVE_KEY_SYMBOL	animation (pygamelib.board_items.GenericStructure
(pygamelib.assets.graphics.MiscTechnicals attribute), 84	<pre>property), 426 animation(pygamelib.board_items.GenericStructureComplexComponent</pre>
AMBULANCE (pygamelib.assets.graphics.Models attribute), 181	property), 416 animation (pygamelib.board_items.Immovable prop-
${\tt AMERICAN\_FOOTBALL}\ (pygame lib. assets. graphics. Models$	erty), 435
attribute), 181  AMPHORA (pygamelib.assets.graphics.Models attribute),	animation (pygamelib.board_items.Movable property), 445
181 ANATOMICAL_HEART (pygamelib.assets.graphics.Models attribute), 181	animation (pygamelib.board_items.NPC property), 455 animation (pygamelib.board_items.Player property), 465
ANCHOR (pygamelib.assets.graphics.Models attribute),	animation (pygamelib.board_items.Projectile property),

- 476
- animation (pygamelib.board items.TextItem property), APL\_FUNCTIONAL\_SYMBOL\_DELTA\_UNDERBAR
- animation (pygamelib.board\_items.Tile property), 499
- animation (pygamelib.board\_items.Treasure property),
- animation (pygamelib.board items. Wall property), 520 ANT (pygamelib.assets.graphics.Models attribute), 181
- ANTENNA\_BARS (pygamelib.assets.graphics.Models attribute), 181
- ANXIOUS\_FACE\_WITH\_SWEAT
  - (pygamelib.assets.graphics.Models attribute),
- APL\_FUNCTIONAL\_SYMBOL\_ALPHA (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_ALPHA\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_BACKSLASH\_BAR (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_BACKSLASH (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_DIAERESIS (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_JOT (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_STAR (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_STILE (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_CIRCLE\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_COMMA\_BAR (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DEL\_DIAERESIS (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DEL\_STILE (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DEL\_TILDE (pygamelib. assets. graphics. Misc Technicalsattribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DELTA\_STILE (pygamelib.assets.graphics.MiscTechnicals

- attribute), 84
- (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DIAMOND\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DOWN\_CARET\_TILDE (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DOWN\_SHOE\_STILE (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DOWN\_TACK\_JOT (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_DOWN\_TACK\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_DOWNWARDS\_VANE (pygamelib.assets.graphics.MiscTechnicals attribute), 84
- APL\_FUNCTIONAL\_SYMBOL\_EPSILON\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_GREATER\_THAN\_DIAERESIS (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_I\_BEAM (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_IOTA (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_IOTA\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_JOT\_DIAERESIS (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_JOT\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_LEFT\_SHOE\_STILE (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_LEFTWARDS\_VANE (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_OMEGA (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_OMEGA\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals

- attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_BACKSLASH (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_CIRCLE (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_COLON (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DEL (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DELTA (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DIAMOND (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DIVIDE (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DOWN\_CARET (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_DOWNWARDS\_ARROW (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_EQUAL (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_GREATER\_THAN (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_JOT (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_LEFTWARDS\_ARROW (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_LESS\_THAN (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_NOT\_EQUAL (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_QUESTION (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_RIGHTWARDS\_ARROW (pygamelib.assets.graphics.MiscTechnicals

- attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_SLASH (pygamelib.assets.graphics.MiscTechnicals attribute), 85
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_UP\_CARET (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_QUAD\_UPWARDS\_ARROW (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_QUOTE\_QUAD (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_QUOTE\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_RHO (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_RIGHTWARDS\_VANE (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_SEMICOLON\_UNDERBAR (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_SLASH\_BAR (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_SQUISH\_QUAD (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_STAR\_DIAERESIS (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_STILE\_TILDE (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_TILDE\_DIAERESIS (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_UP\_CARET\_TILDE (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_UP\_SHOE\_JOT (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_DIAERESIS (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_JOT (pygamelib.assets.graphics.MiscTechnicals attribute), 86
- $$\label{eq:continuous} \begin{split} \texttt{APL\_FUNCTIONAL\_SYMBOL\_UP\_TACK\_OVERBAR} \\ (\textit{pygamelib.assets.graphics.MiscTechnicals} \end{split}$$

attribute), 86 APL_FUNCTIONAL_SYMBOL_UPWARDS_VANE	attach() (pygamelib.board_items.ActionableTile method), 273
(pygame lib. assets. graphics. Misc Technicals	attach() (pygamelib.board_items.BoardComplexItem
attribute), 86 APL_FUNCTIONAL_SYMBOL_ZILDE	method), 284 attach() (pygamelib.board_items.BoardItem method),
(pygamelib.assets.graphics.MiscTechnicals	305
attribute), 86	attach() (pygamelib.board_items.BoardItemComplexComponent
apply_force() (pygamelib.gfx.particles.CircleEmitter	method), 294
method), 690	attach() (pygamelib.board_items.BoardItemVoid
<pre>apply_force() (pygamelib.gfx.particles.ColorParticle</pre>	method), 314
method), 696	attach() (pygamelib.board_items.Camera method), 323
<pre>apply_force() (pygamelib.gfx.particles.ColorPartitionPa</pre>	uatclach() (pygamelib.board_items.Character_method),
method), 702	333
apply_force() (pygamelib.gfx.particles.Particle	attach() (pygamelib.board_items.ComplexDoor
method), 718	method), 343
apply_force() (pygamelib.gfx.particles.ParticleEmitter	
method), 710	method), 354
<pre>apply_force() (pygamelib.gfx.particles.PartitionParticle method), 734</pre>	attach() (pygamelib.board_items.ComplexPlayer method), 365
apply_force() (pygamelib.gfx.particles.RandomColorPa	
method), 740	method), 376
<pre>apply_force() (pygamelib.gfx.particles.RandomColorPa</pre>	rativaeh(i)cle (pygamelib.board_items.ComplexWall
method), 745	method), 387
AQUARIUS (pygamelib.assets.graphics.Models attribute),	<pre>attach() (pygamelib.board_items.Door method), 398</pre>
181	$\verb attach()  (pygame lib. board\_items. Generic Actionable Structure$
ARC (pygamelib.assets.graphics.MiscTechnicals at-	method), 407
tribute), 86	attach() (pygamelib.board_items.GenericStructure
ARIES (pygamelib.assets.graphics.Models attribute), 181	method), 426
ARTICULATED_LORRY (pygamelib.assets.graphics.Models	attach() (pygamelib.board_items.GenericStructureComplexComponent
attribute), 182	method), 416
ARTIST_PALETTE (pygamelib.assets.graphics.Models at-	attach() (pygamelib.board_items.Immovable method),
tribute), 182	435 attach() (pygamelib.board_items.Movable method),
ASTAR (pygamelib.constants.Algorithm attribute), 528 ASTONISHED_FACE (pygamelib.assets.graphics.Models	attach() (pygamelib.board_items.Movable method), 445
attribute), 182	attach() (pygamelib.board_items.NPC method), 455
ATM_SIGN (pygamelib.assets.graphics.Models attribute),	attach() (pygamelib.board_items.Player method), 465
182	attach() (pygamelib.board_items.Projectile method),
ATOM_SYMBOL (pygamelib.assets.graphics.Models at-	476
tribute), 182	<pre>attach() (pygamelib.board_items.TextItem method),</pre>
attach() (pygamelib.actuators.Actuator method), 8	488
attach() (pygamelib.actuators.Behavioral method), 11	attach() (pygamelib.board_items.Tile method), 499
<pre>attach() (pygamelib.actuators.PathActuator method),</pre>	<pre>attach() (pygamelib.board_items.Treasure method),</pre>
15	510
attach() (pygamelib.actuators.PathFinder method), 25	attach() (pygamelib.board_items.Wall method), 520
<pre>attach() (pygamelib.actuators.PatrolActuator method),</pre>	attach() (pygamelib.engine.Board method), 535
19	attach() (pygamelib.engine.Game method), 549
attach() (pygamelib.actuators.RandomActuator	attach() (pygamelib.engine.Inventory method), 562
method), 31	attach() (pygamelib.engine.Screen method), 572
attach() (pygamelib.actuators.UnidirectionalActuator	attach() (pygamelib.gfx.core.Color method), 613
method), 35	attach() (pygamelib.gfx.core.Sprite method), 595
attach() (pygamelib.base.PglBaseObject method), 246 attach() (pygamelib.base.Text method), 251	attach() (pygamelib.gfx.core.Sprixel method), 605 attach() (pygamelib.gfx.particles.CircleEmitter
attach() (pygamelib.board_items.Actionable method),	method), 691
263	attach() (pygamelib.gfx.particles.ColorParticle
	The state of the s

method), 696	BACKHAND_INDEX_POINTING_RIGHT
${\it attach()}~(pygamelib.gfx.particles. Color Partition Particle~method), 702$	(pygamelib.assets.graphics.Models attribute), 182
attach() (pygamelib.gfx.particles.Particle method), 718	BACKHAND_INDEX_POINTING_UP
attach() (pygamelib.gfx.particles.ParticleEmitter method), 710	(pygamelib.assets.graphics.Models attribute), 182
attach() (pygamelib.gfx.particles.ParticleSprixel method), 725	$\begin{array}{c} {\tt BACKPACK}\ (pygamelib. assets. graphics. Models\ attribute), \\ 182 \end{array}$
attach() (pygamelib.gfx.particles.PartitionParticle method), 734	backspace() (pygamelib.gfx.ui.LineInput method), 656 BACON (pygamelib.assets.graphics.Models attribute), 182
attach() (pygamelib.gfx.particles.RandomColorParticle method), 740	BADGER (pygamelib.assets.graphics.Models attribute), 182
attach() (pygamelib.gfx.particles.RandomColorPartition. method), 746	tribute), 182
attach() (pygamelib.gfx.ui.BoxLayout method), 621	BAGEL (pygamelib.assets.graphics.Models attribute), 182
attach() (pygamelib.gfx.ui.Cursor method), 629	BAGGAGE_CLAIM (pygamelib.assets.graphics.Models at-
attach() (pygamelib.gfx.ui.FormLayout method), 637	tribute), 182
attach() (pygamelib.gfx.ui.GridLayout method), 642 attach() (pygamelib.gfx.ui.Layout method), 650	BAGUETTE_BREAD (pygamelib.assets.graphics.Models attribute), 182
attach() (pygamelib.gfx.ui.LineInput method), 656	${\tt BALANCE\_SCALE}\ (pygamelib. assets. graphics. Models\ at-$
attach() (pygamelib.gfx.ui.Widget method), 685	tribute), 182
AUTO_RICKSHAW (pygamelib.assets.graphics.Models attribute), 182	BALD (pygamelib.assets.graphics.Models attribute), 182 BALL (pygamelib.assets.graphics.Models attribute), 182
AUTOMOBILE (pygamelib.assets.graphics.Models attribute), 182	BALLET_SHOES (pygamelib.assets.graphics.Models attribute), 182
available_space() (pygamelib.engine.Inventory method), 562	$\begin{array}{c} {\tt BALLOON} \ \ (pygamelib. assets. graphics. Models \ \ attribute), \\ 183 \end{array}$
AVOCADO (pygamelib.assets.graphics.Models attribute),	BALLOT_BOX_WITH_BALLOT
AVOCADO (pygamelib.assets.graphics.Models attribute), 182 AXE (pygamelib.assets.graphics.Models attribute), 182	BALLOT_BOX_WITH_BALLOT (pygamelib.assets.graphics.Models attribute), 183
182	(pygame lib. assets. graphics. Models  attribute),
182 AXE (pygamelib.assets.graphics.Models attribute), 182	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183
182  AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  tribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute),
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute),
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183  BAT (pygamelib.assets.graphics.Models attribute), 183  BAT (pygamelib.assets.graphics.Models attribute), 183  BAT (pygamelib.assets.graphics.Models attribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183  BAT (pygamelib.assets.graphics.Models attribute), 183  BAT (pygamelib.assets.graphics.Models attribute), 183  BATHTUB (pygamelib.assets.graphics.Models attribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183  BATHTUB (pygamelib.assets.graphics.Models attribute), 183  BATHTUB (pygamelib.assets.graphics.Models attribute), 183  BATTERY (pygamelib.assets.graphics.Models attribute), 183
AXE (pygamelib.assets.graphics.Models attribute), 182  B b (pygamelib.gfx.core.Color property), 613 B_BUTTON_BLOOD_TYPE	(pygamelib.assets.graphics.Models attribute), 183  BANANA (pygamelib.assets.graphics.Models attribute), 183  BANJO (pygamelib.assets.graphics.Models attribute), 183  BANK (pygamelib.assets.graphics.Models attribute), 183  BAR_CHART (pygamelib.assets.graphics.Models attribute), 183  BARBER_POLE (pygamelib.assets.graphics.Models attribute), 183  BASEBALL (pygamelib.assets.graphics.Models attribute), 183  BASKET (pygamelib.assets.graphics.Models attribute), 183  BASKETBALL (pygamelib.assets.graphics.Models attribute), 183  BAT (pygamelib.assets.graphics.Models attribute), 183  BAT (pygamelib.assets.graphics.Models attribute), 183  BATTERY (pygamelib.assets.graphics.Models attribute), 183  BATTERY (pygamelib.assets.graphics.Models attribute), 183  BATTERY (pygamelib.assets.graphics.Models attribute), 183  BEACH_WITH_UMBRELLA

183	BLACK_CIRCLE (pygamelib.assets.graphics.Models
BEAR (pygamelib.assets.graphics.Models attribute), 183	attribute), 184
BEATING_HEART (pygamelib.assets.graphics.Models at-	BLACK_CIRCLE_FOR_RECORD
tribute), 183	(pygamelib.assets.graphics.MiscTechnicals
BEAVER (pygamelib.assets.graphics.Models attribute),	attribute), 86
183	BLACK_DIAMOND (pygamelib.assets.graphics.GeometricShapes
BED (pygamelib.assets.graphics.Models attribute), 183 BEER_MUG (pygamelib.assets.graphics.Models attribute),	attribute), 63 black_dim() (pygamelib.base.Text static method), 251
183	BLACK_DOWN_POINTING_DOUBLE_TRIANGLE
BEETLE (pygamelib.assets.graphics.Models attribute),	(pygamelib.assets.graphics.MiscTechnicals
183	attribute), 86
Behavioral (class in pygamelib.actuators), 11	BLACK_DOWN_POINTING_SMALL_TRIANGLE
BELL (pygamelib.assets.graphics.Models attribute), 183	(pygame lib. assets. graphics. Geometric Shapes
BELL_PEPPER (pygamelib.assets.graphics.Models at-	attribute), 63
tribute), 183	BLACK_DOWN_POINTING_TRIANGLE
BELL_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 86	(pygamelib.assets.graphics.GeometricShapes attribute), 63
BELL_WITH_SLASH (pygamelib.assets.graphics.Models	BLACK_FLAG (pygamelib.assets.graphics.Models at-
attribute), 183	tribute), 184
BELLHOP_BELL (pygamelib.assets.graphics.Models	BLACK_HEART (pygamelib.assets.graphics.Models at-
attribute), 183	tribute), 184
BENTO_BOX (pygamelib.assets.graphics.Models at-	. 4.0
tribute), 183	attribute), 63
	BLACK_LARGE_SQUARE (pygamelib.assets.graphics.Models
attribute), 86	attribute), 184
BENZENE_RING_WITH_CIRCLE (pygamelib.assets.graphics.MiscTechnicals	BLACK_LEFT_POINTING_DOUBLE_TRIANGLE (pygamelib.assets.graphics.MiscTechnicals
attribute), 86	attribute), 86
BEVERAGE_BOX (pygamelib.assets.graphics.Models	BLACK_LEFT_POINTING_DOUBLE_TRIANGLE_WITH_VERTICAL_BAR
attribute), 183	(pygamelib.assets.graphics.MiscTechnicals at-
BFS (pygamelib.constants.Algorithm attribute), 528	tribute), 86
bg_color (pygamelib.base.Text property), 251	BLACK_LEFT_POINTING_POINTER
bg_color (pygamelib.gfx.core.Sprixel property), 605	(pygame lib. assets. graphics. Geometric Shapes
bg_color (pygamelib.gfx.particles.ParticleSprixel prop-	attribute), 63
erty), 725	BLACK_LEFT_POINTING_SMALL_TRIANGLE
bg_color (pygamelib.gfx.ui.LineInput property), 657 bg_color (pygamelib.gfx.ui.Widget property), 686	(pygamelib.assets.graphics.GeometricShapes attribute), 63
BICYCLE (pygamelib.assets.graphics.Models attribute),	
183	(pygamelib.assets.graphics.GeometricShapes
BIKINI (pygamelib.assets.graphics.Models attribute),	attribute), 63
183	BLACK_LOWER_LEFT_TRIANGLE
BILLED_CAP (pygamelib.assets.graphics.Models at-	(pygame lib. assets. graphics. Geometric Shapes
tribute), 183	attribute), 63
BIOHAZARD (pygamelib.assets.graphics.Models at-	BLACK_LOWER_RIGHT_TRIANGLE
tribute), 183	(pygamelib.assets.graphics.GeometricShapes
BIRD (pygamelib.assets.graphics.Models attribute), 184 BIRTHDAY_CAKE (pygamelib.assets.graphics.Models at-	attribute), 63 BLACK_MEDIUM_DOWN_POINTING_TRIANGLE
tribute), 184	(pygamelib.assets.graphics.MiscTechnicals
BISON (pygamelib.assets.graphics.Models attribute), 184	attribute), 86
black() (pygamelib.base.Text static method), 251	BLACK_MEDIUM_LEFT_POINTING_TRIANGLE
black_bright() (pygamelib.base.Text static method),	(pygamelib.assets.graphics.MiscTechnicals
251	attribute), 86
${\tt BLACK\_CIRCLE}\ (pygamelib. assets. graphics. Geometric Shapper and the property of the pr$	
attribute), 63	(pygamelib.assets.graphics.MiscTechnicals

attribute), 86 BLACK_MEDIUM_SMALL_SQUARE	class method), 725 BLACK_SQUARE_BUTTON
(pygame lib. assets. graphics. Geometric Shapes	(pygamelib.assets.graphics.Models attribute),
attribute), 63	184
BLACK_MEDIUM_SMALL_SQUARE	BLACK_SQUARE_FOR_STOP
(pygamelib.assets.graphics.Models attribute),	(pygamelib.assets.graphics.MiscTechnicals
184	attribute), 87
BLACK_MEDIUM_SQUARE	BLACK_UP_POINTING_DOUBLE_TRIANGLE
(pygamelib.assets.graphics.GeometricShapes	(pygamelib.assets.graphics.MiscTechnicals
attribute), 64	attribute), 87
BLACK_MEDIUM_SQUARE	BLACK_UP_POINTING_SMALL_TRIANGLE
(pygamelib.assets.graphics.Models attribute), 184	(pygamelib.assets.graphics.GeometricShapes attribute), 64
BLACK_MEDIUM_UP_POINTING_TRIANGLE	BLACK_UP_POINTING_TRIANGLE
(pygamelib.assets.graphics.MiscTechnicals attribute), 86	(pygamelib.assets.graphics.GeometricShapes attribute), 64
BLACK_NIB (pygamelib.assets.graphics.Models at-	BLACK_UPPER_LEFT_TRIANGLE
tribute), 184	(pygamelib.assets.graphics.GeometricShapes
BLACK_PARALLELOGRAM	attribute), 64
(pygame lib. assets. graphics. Geometric Shapes	BLACK_UPPER_RIGHT_TRIANGLE
attribute), 64	(pygame lib. assets. graphics. Geometric Shapes
<pre>black_rect() (pygamelib.gfx.core.Sprixel class</pre>	attribute), 64
method), 605	BLACK_VERTICAL_RECTANGLE
black_rect() (pygamelib.gfx.particles.ParticleSprixel class method), 725	(pygamelib.assets.graphics.GeometricShapes attribute), 64
$\verb+BLACK_RECTANGLE+ (pygamelib. assets. graphics. Geometric States) and the pygamelib. The pygamelib is a set of the pygamelib is a set of the pygamelib is a set of the pygamelib. The pygamelib is a set of the pygamelib is a $	Shalpend() (pygamelib.gfx.core.Color method), 613
attribute), 64	Blocks (class in pygamelib.assets.graphics), 40
BLACK_RIGHT_POINTING_DOUBLE_TRIANGLE	BLOSSOM (pygamelib.assets.graphics.Models attribute),
(pygame lib. assets. graphics. Misc Technicals	184
attribute), 87	BLOWFISH (pygamelib.assets.graphics.Models attribute),
BLACK_RIGHT_POINTING_DOUBLE_TRIANGLE_WITH_VER	
(pygamelib.assets.graphics.MiscTechnicals at-	blue (pygamelib.gfx.ui.ColorPicker property), 626
<i>tribute</i> ), 87	blue() (pygamelib.base.Text static method), 251
BLACK_RIGHT_POINTING_POINTER	BLUE_BOOK (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.GeometricShapes	tribute), 184
attribute), 64	blue_bright() (pygamelib.base.Text static method),
BLACK_RIGHT_POINTING_SMALL_TRIANGLE	251  PLUE CIPCLE (progradib assets graphics Models at
(pygamelib.assets.graphics.GeometricShapes attribute), 64	BLUE_CIRCLE (pygamelib.assets.graphics.Models at- tribute), 184
BLACK_RIGHT_POINTING_TRIANGLE	blue_dim() (pygamelib.base.Text static method), 251
(pygamelib.assets.graphics.GeometricShapes	BLUE_HEART (pygamelib.assets.graphics.Models at-
attribute), 64	tribute), 184
BLACK_RIGHT_POINTING_TRIANGLE_WITH_DOUBLE_VER	
(pygamelib.assets.graphics.MiscTechnicals at-	606
tribute), 87	blue_rect() (pygamelib.gfx.particles.ParticleSprixel
BLACK_SMALL_SQUARE (pygamelib.assets.graphics.Geometrics)	100 00 1
attribute), 64	BLUE_SQUARE (pygamelib.assets.graphics.Models at-
BLACK_SMALL_SQUARE (pygamelib.assets.graphics.Models	0 1
attribute), 184	blue_square() (pygamelib.gfx.core.Sprixel class
${\tt BLACK\_SQUARE}\ (pygamelib. assets. graphics. Geometric Shapper and the properties of the properti$	
attribute), 64	blue_square() (pygamelib.gfx.particles.ParticleSprixel
<pre>black_square() (pygamelib.gfx.core.Sprixel class</pre>	class method), 726
method), 605	BLUEBERRIES (pygamelib.assets.graphics.Models at-
$black\_square()$ (pygamelib.gfx.particles.ParticleSprixel	tribute), 184

BOAR (pygamelib.assets.graphics.Models attribute), 184	BOWLING (pygamelib.assets.graphics.Models attribute),
Board (class in pygamelib.engine), 532	184
BoardComplexItem (class in pygamelib.board_items),	Box (class in pygamelib.gfx.ui), 618
282 BoardItem (class in pygamelib.board_items), 302	BoxDrawings (class in pygamelib.assets.graphics), 43 BOXING_GLOVE (pygamelib.assets.graphics.Models
BoardItemComplexComponent (class in	attribute), 185
pygamelib.board_items), 293	BoxLayout (class in pygamelib.gfx.ui), 619
BoardItemVoid (class in pygamelib.board_items), 313	BOY (pygamelib.assets.graphics.Models attribute), 185
BOLD (pygamelib.constants.TextStyle attribute), 530	BRAIN (pygamelib.assets.graphics.Models attribute), 185
BOMB (pygamelib.assets.graphics.Models attribute), 184	BREAD (pygamelib.assets.graphics.Models attribute), 185
BONE (pygamelib.assets.graphics.Models attribute), 184	BREAST_FEEDING (pygamelib.assets.graphics.Models at-
BOOKMARK (pygamelib.assets.graphics.Models attribute),	tribute), 185
184	BRICK (pygamelib.assets.graphics.Models attribute), 185
BOOKMARK_TABS (pygamelib.assets.graphics.Models at-	BRIDGE_AT_NIGHT (pygamelib.assets.graphics.Models
tribute), 184 BOOKS (pygamelib.assets.graphics.Models attribute), 184	attribute), 185 BRIEFCASE (pygamelib.assets.graphics.Models at-
BOOMERANG (pygamelib.assets.graphics.Models at-	tribute), 185
tribute), 184	BRIEFS (pygamelib.assets.graphics.Models attribute),
BOTTLE_WITH_POPPING_CORK	185
(pygamelib.assets.graphics.Models attribute), 184	BRIGHT_BUTTON (pygamelib.assets.graphics.Models attribute), 185
BOTTOM (pygamelib.constants.Alignment attribute), 528	BROCCOLI (pygamelib.assets.graphics.Models attribute),
BOTTOM_CURLY_BRACKET	185
(pygame lib. assets. graphics. Misc Technicals	BROKEN_CIRCLE_WITH_NORTHWEST_ARROW
attribute), 87	(pygamelib.assets.graphics.MiscTechnicals
BOTTOM_HALF_INTEGRAL	attribute), 87
(pygamelib.assets.graphics.MiscTechnicals	BROKEN_HEART (pygamelib.assets.graphics.Models
attribute), 87 BOTTOM_LEFT_CORNER (pygamelib.assets.graphics.MiscTed	attribute), 185
attribute), 87	BROWN_CIRCLE (pygamelib.assets.graphics.Models
BOTTOM_LEFT_CROP (pygamelib.assets.graphics.MiscTechn	110 0 1
attribute), 87	BROWN_HEART (pygamelib.assets.graphics.Models at-
BOTTOM_PARENTHESIS (pygamelib.assets.graphics.MiscTed	
attribute), 87	BROWN_SQUARE (pygamelib.assets.graphics.Models
BOTTOM_RIGHT_CORNER	attribute), 185
(pygamelib.assets.graphics.MiscTechnicals	BUBBLE_TEA (pygamelib.assets.graphics.Models at-
attribute), 87	tribute), 185
BOTTOM_RIGHT_CROP (pygamelib.assets.graphics.MiscTechattribute), 87	http://disect.com/http://disec
BOTTOM_SQUARE_BRACKET	buffer (pygamelib.engine.Screen property), 572
(pygame lib. assets. graphics. Misc Technicals	BUG (pygamelib.assets.graphics.Models attribute), 185
attribute), 87	BUILDING_CONSTRUCTION
BOTTOM_SQUARE_BRACKET_OVER_TOP_SQUARE_BRACKET	
(pygamelib.assets.graphics.MiscTechnicals at-	185
tribute), 87	BULLET (pygamelib.assets.graphics.GeometricShapes at-
BOTTOM_TORTOISE_SHELL_BRACKET	tribute), 64
(pygamelib.assets.graphics.MiscTechnicals attribute), 87	BULLET_TRAIN (pygamelib.assets.graphics.Models attribute), 185
BOUQUET (pygamelib.assets.graphics.Models attribute),	BULLSEYE (pygamelib.assets.graphics.GeometricShapes
184	attribute), 64
BOW_AND_ARROW (pygamelib.assets.graphics.Models attribute), 184	BURRITO (pygamelib.assets.graphics.Models attribute), 185
BOWL_WITH_SPOON (pygamelib.assets.graphics.Models	BUS (pygamelib.assets.graphics.Models attribute), 185
attribute), 184	BUS_STOP (pygamelib.assets.graphics.Models attribute),

can\_move() (pygamelib.board items.Door method), 398 185 BUST\_IN\_SILHOUETTE (pygamelib.assets.graphics.Models can\_move() (pygamelib.board\_items.GenericActionableStructure attribute), 185 *method*), 408 BUSTS\_IN\_SILHOUETTE can\_move() (pygamelib.board\_items.GenericStructure (pygamelib.assets.graphics.Models attribute), method), 426 can\_move() (pygamelib.board items.GenericStructureComplexComponen BUTTER (pygamelib.assets.graphics.Models attribute), method), 416 (pygamelib.board\_items.Immovable 185 can\_move() (pygamelib.assets.graphics.Models BUTTERFLY method), 435 attribute), 185 can\_move() (pygamelib.board\_items.Movable method), C can\_move() (pygamelib.board\_items.NPC method), 455 can\_move() (pygamelib.board\_items.Player method), CACTUS (pygamelib.assets.graphics.Models attribute), 465 can\_move() (pygamelib.board\_items.Projectile method), calculate\_size() (pygamelib.gfx.core.Sprite method), 476 can\_move() (pygamelib.board\_items.TextItem method), CALENDAR (pygamelib.assets.graphics.Models attribute), 185 can\_move() (pygamelib.board\_items.Tile method), 499 CALL\_ME\_HAND (pygamelib.assets.graphics.Models can\_move() (pygamelib.board items.Treasure method), attribute), 186 510 CAMEL (pygamelib.assets.graphics.Models attribute), 186 can\_move() (pygamelib.board\_items.Wall method), 520 Camera (class in pygamelib.board items), 321 CANCER (pygamelib.assets.graphics.Models attribute), CAMERA (pygamelib.assets.graphics.Models attribute), 186 CANDLE CAMERA\_WITH\_FLASH (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Models attribute), attribute), 186 CANDY (pygamelib.assets.graphics.Models attribute), 186 CAMPING (pygamelib.assets.graphics.Models attribute), CANNED\_FOOD (pygamelib.assets.graphics.Models 186 tribute), 186 (pygamelib.board\_items.Actionable can\_move() CANOE (pygamelib.assets.graphics.Models attribute), 186 method), 264 CAPRICORN (pygamelib.assets.graphics.Models (pygamelib.board\_items.ActionableTile can\_move() tribute), 186 method), 274 CARD\_FILE\_BOX (pygamelib.assets.graphics.Models atcan\_move() (pygamelib.board\_items.BoardComplexItem tribute), 186 method), 284 CARD\_INDEX (pygamelib.assets.graphics.Models can\_move() (pygamelib.board\_items.BoardItem tribute), 186 method), 305 can\_move() (pygamelib.board\_items.BoardItemComplexCompDncINDEX\_DIVIDERS (pygamelib.assets.graphics.Models attribute), method), 294 can\_move() (pygamelib.board items.BoardItemVoid CAROUSEL\_HORSE (pygamelib.assets.graphics.Models at*method*), 314 tribute), 186 can\_move() (pygamelib.board items.Camera method), CARP\_STREAMER (pygamelib.assets.graphics.Models at-323 tribute), 186 can\_move() (pygamelib.board\_items.Character CARPENTRY\_SAW (pygamelib.assets.graphics.Models atmethod), 333 tribute), 186 (pygamelib.board\_items.ComplexDoor can\_move() CARROT (pygamelib.assets.graphics.Models attribute), *method*), 343 186 can\_move() (pygamelib.board\_items.ComplexNPC CASTLE (pygamelib.assets.graphics.Models attribute), method), 354 (pygamelib.board\_items.ComplexPlayer can\_move() CAT (pygamelib.assets.graphics.Models attribute), 186 *method*), 365 CAT\_FACE (pygamelib.assets.graphics.Models attribute), can\_move() (pygamelib.board\_items.ComplexTreasure 186 *method*), 376 CAT\_WITH\_TEARS\_OF\_JOY (pygamelib.board\_items.ComplexWall can\_move() (pygamelib.assets.graphics.Models attribute), method), 387

187

186 **CIGARETTE** (pygamelib.assets.graphics.Models tribute), 187 CAT\_WITH\_WRY\_SMILE (pygamelib.assets.graphics.Models attribute), 186 CINEMA (pygamelib.assets.graphics.Models attribute), CENTER (pygamelib.constants.Alignment attribute), 528 187 (pygamelib.assets.graphics.Models attribute), CIRCLE\_WITH\_ALL\_BUT\_UPPER\_LEFT\_QUADRANT\_BLACK (pygamelib.assets.graphics.GeometricShapes CHAIR (pygamelib.assets.graphics.Models attribute), 186 attribute), 64 change\_level() (pygamelib.engine.Game method), 549 CIRCLE\_WITH\_LEFT\_HALF\_BLACK (pygamelib.assets.graphics.GeometricShapes Character (class in pygamelib.board items), 331 CHART\_DECREASING (pygamelib.assets.graphics.Models attribute), 64 attribute), 186 CIRCLE\_WITH\_LOWER\_HALF\_BLACK CHART\_INCREASING (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.GeometricShapes attribute), 186 attribute), 64 CHART\_INCREASING\_WITH\_YEN CIRCLE\_WITH\_RIGHT\_HALF\_BLACK (pygamelib.assets.graphics.Models attribute), (pygamelib.assets.graphics.GeometricShapes 186 attribute), 64 CHECK\_BOX\_WITH\_CHECK CIRCLE\_WITH\_UPPER\_HALF\_BLACK (pygamelib.assets.graphics.Models attribute), (pygamelib.assets.graphics.GeometricShapes attribute), 64 CIRCLE\_WITH\_UPPER\_RIGHT\_QUADRANT\_BLACK CHECK\_MARK (pygamelib.assets.graphics.Models tribute), 186 (pygamelib.assets.graphics.GeometricShapes CHECK\_MARK\_BUTTON (pygamelib.assets.graphics.Models attribute), 64 attribute), 186 CIRCLE\_WITH\_VERTICAL\_FILL check\_sanity() (pygamelib.engine.Board method), 535 (pygamelib.assets.graphics.GeometricShapes CHEESE\_WEDGE (pygamelib.assets.graphics.Models attribute), 64 attribute), 187 CIRCLED\_HORIZONTAL\_BAR\_WITH\_NOTCH CHEQUERED\_FLAG (pygamelib.assets.graphics.Models at-(pygamelib.assets.graphics.MiscTechnicals tribute), 187 attribute), 87 CHERRIES (pygamelib.assets.graphics.Models attribute), CIRCLED\_M (pygamelib.assets.graphics.Models attribute), 187 CHERRY\_BLOSSOM (pygamelib.assets.graphics.Models at-CIRCLED\_TRIANGLE\_DOWN tribute), 187 (pygamelib.assets.graphics.MiscTechnicals CHESS\_PAWN (pygamelib.assets.graphics.Models attribute), 87 CircleEmitter (class in pygamelib.gfx.particles), 689 tribute), 187 CHESTNUT (pygamelib.assets.graphics.Models attribute), CIRCUS\_TENT (pygamelib.assets.graphics.Models tribute), 187 CHICKEN (pygamelib.assets.graphics.Models attribute), **CITYSCAPE** (pygamelib.assets.graphics.Models attribute), 187 CHILD (pygamelib.assets.graphics.Models attribute), 187 CITYSCAPE\_AT\_DUSK (pygamelib.assets.graphics.Models children (pygamelib.gfx.ui.LineInput property), 657 attribute), 187 children (pygamelib.gfx.ui.Widget property), 686 CL\_BUTTON (pygamelib.assets.graphics.Models CHILDREN\_CROSSING (pygamelib.assets.graphics.Models tribute), 188 CLAMP (pygamelib.assets.graphics.Models attribute), 187 attribute), 187 CHIPMUNK (pygamelib.assets.graphics.Models attribute), CLAPPER\_BOARD (pygamelib.assets.graphics.Models attribute), 187 CHOCOLATE\_BAR (pygamelib.assets.graphics.Models at-CLAPPING\_HANDS (pygamelib.assets.graphics.Models attribute), 187 tribute), 187 choices (pygamelib.gfx.ui.GridSelector property), 648 CLASSICAL\_BUILDING (pygamelib.assets.graphics.Models CHOPSTICKS (pygamelib.assets.graphics.Models attribute), 187 clear() (pygamelib.engine.Screen method), 573 tribute), 187 CHRISTMAS\_TREE (pygamelib.assets.graphics.Models atclear() (pygamelib.gfx.core.SpriteCollection method), tribute), 187 591 CHURCH (pygamelib.assets.graphics.Models attribute), clear() (pygamelib.gfx.ui.LineInput method), 657

772 Index

clear\_buffers() (pygamelib.engine.Screen method),

573	COCKTAIL_GLASS (pygamelib.assets.graphics.Models at-
<pre>clear_cell() (pygamelib.engine.Board method), 535</pre>	tribute), 188
clear_constraints() (pygamelib.engine.Inventory method), 562	COCONUT (pygamelib.assets.graphics.Models attribute), 188
clear_frame_buffer() (pygamelib.engine.Screen method), 573	COFFIN (pygamelib.assets.graphics.Models attribute), 188
clear_screen() (pygamelib.engine.Game method), 549 CLEAR_SCREEN_SYMBOL	COIN (pygamelib.assets.graphics.Models attribute), 188 COLD_FACE (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.MiscTechnicals	tribute), 188
attribute), 87	collapse() (pygamelib.gfx.ui.Menu method), 665
clear_session_logs() (pygamelib.engine.Game method), 550	<pre>collides_with() (pygamelib.board_items.Actionable method), 264</pre>
clear_waypoints() (pygamelib.actuators.PathFinder method), 25	<pre>collides_with() (pygamelib.board_items.ActionableTile method), 274</pre>
CLINKING_BEER_MUGS (pygamelib.assets.graphics.Models attribute), 187	<pre>collides_with() (pygamelib.board_items.BoardComplexItem method), 285</pre>
CLINKING_GLASSES (pygamelib.assets.graphics.Models attribute), 187	<pre>collides_with() (pygamelib.board_items.BoardItem     method), 306</pre>
CLIPBOARD (pygamelib.assets.graphics.Models attribute), 187	<pre>collides_with() (pygamelib.board_items.BoardItemComplexComponen</pre>
CLOCKWISE_VERTICAL_ARROWS (pygamelib.assets.graphics.Models attribute),	<pre>collides_with() (pygamelib.board_items.BoardItemVoid method), 315</pre>
187	collides_with() (pygamelib.board_items.Camera
close() (pygamelib.gfx.ui.MenuBar method), 671	method), 324
CLOSED_BOOK (pygamelib.assets.graphics.Models attribute), 187	<pre>collides_with() (pygamelib.board_items.Character method), 334</pre>
CLOSED_MAILBOX_WITH_LOWERED_FLAG	collides_with() (pygamelib.board_items.ComplexDoor
(pygamelib.assets.graphics.Models attribute),	method), 344
187	<pre>collides_with() (pygamelib.board_items.ComplexNPC</pre>
CLOSED_MAILBOX_WITH_RAISED_FLAG	method), 355
(pygamelib.assets.graphics.Models attribute), 187	<pre>collides_with() (pygamelib.board_items.ComplexPlayer</pre>
CLOSED_UMBRELLA (pygamelib.assets.graphics.Models attribute), 188	<pre>collides_with() (pygamelib.board_items.ComplexTreasure method), 377</pre>
CLOUD (pygamelib.assets.graphics.Models attribute), 188 CLOUD_WITH_LIGHTNING	<pre>collides_with() (pygamelib.board_items.ComplexWall method), 388</pre>
(pygamelib.assets.graphics.Models attribute), 188	<pre>collides_with()</pre>
CLOUD_WITH_LIGHTNING_AND_RAIN	$\verb collides_with()  (pygame lib.board_items. Generic Actionable Structure $
(pygamelib.assets.graphics.Models attribute), 188	method), 408 collides_with() (pygamelib.board_items.GenericStructure
CLOUD_WITH_RAIN (pygamelib.assets.graphics.Models attribute), 188	<pre>method), 427 collides_with() (pygamelib.board_items.GenericStructureComplexCom</pre>
CLOUD_WITH_SNOW (pygamelib.assets.graphics.Models attribute), 188	method), 417 collides_with() (pygamelib.board_items.Immovable
CLOWN_FACE (pygamelib.assets.graphics.Models at-	method), 436
tribute), 188	collides_with() (pygamelib.board_items.Movable
CLUB_SUIT (pygamelib.assets.graphics.Models at-	method), 446
tribute), 188	collides_with() (pygamelib.board_items.NPC
CLUTCH_BAG (pygamelib.assets.graphics.Models at-	method), 456
tribute), 188	collides_with() (pygamelib.board_items.Player
COCKPOACH (pygamelib assets graphics Models attribute), 188	method), 466
COCKROACH (pygamelib.assets.graphics.Models attribute), 188	collides_with() (pygamelib.board_items.Projectile method), 477
,,	,,,

collides_with() (pygamelib.board_items.TextItem method), 489	column (pygamelib.board_items.NPC property), 456 column (pygamelib.board_items.Player property), 466
	* * * * * * * * * * * * * * * * * * * *
collides_with() (pygamelib.board_items.Tile method), 500	column (pygamelib.board_items.Projectile property), 477
collides_with() (pygamelib.board_items.Treasure	column (pygamelib.board_items.TextItem property), 489
method), 511	column (pygamelib.board_items.Tile property), 500
<pre>collides_with() (pygamelib.board_items.Wall</pre>	column (pygamelib.board_items.Treasure property), 511
method), 521	column (pygamelib.board_items.Wall property), 521
COLLISION (pygamelib.assets.graphics.Models at-	column (pygamelib.gfx.particles.CircleEmitter property),
tribute), 188	691
Color (class in pygamelib.gfx.core), 612	column (pygamelib.gfx.particles.ColorParticle property),
color (pygamelib.gfx.ui.ColorPicker property), 626	697
colorable (pygamelib.gfx.core.Font property), 586	column (pygamelib.gfx.particles.ColorPartitionParticle
ColorParticle (class in pygamelib.gfx.particles), 695	property), 703
ColorPartitionParticle (class in	column (pygamelib.gfx.particles.Particle property), 719
pygamelib.gfx.particles), 701	column (pygamelib.gfx.particles.ParticleEmitter prop-
ColorPicker (class in pygamelib.gfx.ui), 626	erty), 711
ColorPickerDialog (class in pygamelib.gfx.ui), 623 column (pygamelib.base.Vector2D property), 258	column (pygamelib.gfx.particles.PartitionParticle property), 735
column (pygamelib.board_items.Actionable property),	column (pygamelib.gfx.particles.RandomColorParticle
264	property), 740
column (pygamelib.board_items.ActionableTile prop-	${\tt column} \ (pygame lib. gfx. particles. Random Color Partition Particle$
erty), 274	property), 746
column (pygamelib.board_items.BoardComplexItem	column_minimum_width (pygamelib.gfx.ui.FormLayout
property), 285	property), 637
column (pygamelib.board_items.BoardItem property),	column_minimum_width (pygamelib.gfx.ui.GridLayout
306	property), 642
$column (pygamelib.board\_items.BoardItemComplexCom$	OCOMET (pygamelib.assets.graphics.Models attribute), 188
property), 295	COMPASS (pygamelib.assets.graphics.Models attribute),
column (pygamelib.board_items.BoardItemVoid prop-	188
erty), 315	ComplexDoor (class in pygamelib.board_items), 341
column (pygamelib.board_items.Camera property), 324	ComplexNPC (class in pygametib.board_items), 352
column (pygamelib.board_items.Character property),	ComplexPlayer (class in pygamelib.board_items), 363
334	ComplexTreasure (class in pygamelib.board_items),
column (pygamelib.board_items.ComplexDoor prop-	374
erty), 344	ComplexWall (class in pygamelib.board_items), 385
<pre>column (pygamelib.board_items.ComplexNPC property),</pre>	${\tt COMPOSITION\_SYMBOL}\ (pygame lib. assets. graphics. Misc Technical Assets and Strategy and $
355	attribute), 87
column (pygamelib.board_items.ComplexPlayer prop-	COMPUTER_DISK (pygamelib.assets.graphics.Models at-
erty), 366	tribute), 188
column (pygamelib.board_items.ComplexTreasure prop-	
erty), 377	
	COMPUTER_MOUSE (pygamelib.assets.graphics.Models at-
•	tribute), 188
${\tt column}\ (pygame lib.board\_items. Complex Wall\ property),$	tribute), 188 CONFETTI_BALL (pygamelib.assets.graphics.Models at-
column (pygamelib.board_items.ComplexWall property), 388	tribute), 188 CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188
column (pygamelib.board_items.ComplexWall property), 388 column (pygamelib.board_items.Door property), 399	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619
column (pygamelib.board_items.ComplexWall property), 388	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619
column (pygamelib.board_items.ComplexWall property), 388 column (pygamelib.board_items.Door property), 399	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619
column (pygamelib.board_items.ComplexWall property), 388 column (pygamelib.board_items.Door property), 399 column (pygamelib.board_items.GenericActionableStructu property), 408	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619  areonfig (pygamelib.gfx.ui.ColorPickerDialog property), 624
column (pygamelib.board_items.ComplexWall property), 388  column (pygamelib.board_items.Door property), 399  column (pygamelib.board_items.GenericActionableStructuproperty), 408  column (pygamelib.board_items.GenericStructure prop-	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619  areonfig (pygamelib.gfx.ui.ColorPickerDialog property), 624  config (pygamelib.gfx.ui.Dialog property), 632
column (pygamelib.board_items.ComplexWall property), 388  column (pygamelib.board_items.Door property), 399  column (pygamelib.board_items.GenericActionableStructu	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619  treonfig (pygamelib.gfx.ui.ColorPickerDialog property), 624  config (pygamelib.gfx.ui.Dialog property), 632  config (pygamelib.gfx.ui.FileDialog property), 634
column (pygamelib.board_items.ComplexWall property), 388  column (pygamelib.board_items.Door property), 399  column (pygamelib.board_items.GenericActionableStructu- property), 408  column (pygamelib.board_items.GenericStructure property), 427  column (pygamelib.board_items.GenericStructureComplex)	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619  treonfig (pygamelib.gfx.ui.ColorPickerDialog property), 624  config (pygamelib.gfx.ui.Dialog property), 632  config (pygamelib.gfx.ui.FileDialog property), 634  config (pygamelib.gfx.ui.GridSelectorDialog property),
column (pygamelib.board_items.ComplexWall property), 388  column (pygamelib.board_items.Door property), 399  column (pygamelib.board_items.GenericActionableStructure property), 408  column (pygamelib.board_items.GenericStructure property), 427  column (pygamelib.board_items.GenericStructureComplex property), 417	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619  preonfig (pygamelib.gfx.ui.ColorPickerDialog property), 624  config (pygamelib.gfx.ui.Dialog property), 632  config (pygamelib.gfx.ui.FileDialog property), 634  config (pygamelib.gfx.ui.GridSelectorDialog property), 646
column (pygamelib.board_items.ComplexWall property), 388  column (pygamelib.board_items.Door property), 399  column (pygamelib.board_items.GenericActionableStructu- property), 408  column (pygamelib.board_items.GenericStructure property), 427  column (pygamelib.board_items.GenericStructureComplex)	tribute), 188  CONFETTI_BALL (pygamelib.assets.graphics.Models attribute), 188  config (pygamelib.gfx.ui.Box property), 619  treonfig (pygamelib.gfx.ui.ColorPickerDialog property), 624  config (pygamelib.gfx.ui.Dialog property), 632  config (pygamelib.gfx.ui.FileDialog property), 634  config (pygamelib.gfx.ui.GridSelectorDialog property),

config (pygamelib.gfx.ui.MenuAction property), 668 config (pygamelib.gfx.ui.MenuBar property), 671 config (pygamelib.gfx.ui.MessageDialog property), 675 config (pygamelib.gfx.ui.MultiLineInputDialog property), 677 config (pygamelib.gfx.ui.ProgressBar property), 679 config (pygamelib.gfx.ui.ProgressDialog property), 681 config() (pygamelib.engine.Game method), 550 CONFOUNDED\_FACE (pygamelib.assets.graphics.Models attribute), 188 CONFUSED\_FACE (pygamelib.assets.graphics.Models attribute), 188  ${\tt CONICAL\_TAPER}\ (pygame lib. assets. graphics. Misc Technicals$ attribute), 87 Console (class in pygamelib.base), 240 constraints (pygamelib.engine.Inventory property), 562 CONSTRUCTION (pygamelib.assets.graphics.Models attribute), 188 CONSTRUCTION\_WORKER (pygamelib.assets.graphics.Models attribute), CONTINUOUS\_UNDERLINE\_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 87 CONTROL\_KNOBS (pygamelib.assets.graphics.Models attribute), 188 CONVENIENCE\_STORE (pygamelib.assets.graphics.Models attribute), 188 COOKED\_RICE (pygamelib.assets.graphics.Models *tribute*), 188 COOKIE (pygamelib.assets.graphics.Models attribute), COOKING (pygamelib.assets.graphics.Models attribute), COOL\_BUTTON (pygamelib.assets.graphics.Models attribute), 189 copy() (pygamelib.gfx.core.Color method), 614 copy() (pygamelib.gfx.core.Sprite method), 596 copy() (pygamelib.gfx.core.SpriteCollection method), copy() (pygamelib.gfx.core.Sprixel method), 606 copy() (pygamelib.gfx.particles.ParticleSprixel method), 726 COPYRIGHT (pygamelib.assets.graphics.Models at*tribute*), 189 COUCH\_AND\_LAMP (pygamelib.assets.graphics.Models attribute), 189 count() (pygamelib.gfx.ui.BoxLayout method), 621 count() (pygamelib.gfx.ui.FormLayout method), 637 count() (pygamelib.gfx.ui.GridLayout method), 642 count() (pygamelib.gfx.ui.Layout method), 651

count\_active\_particles()

(pygamelib.gfx.particles.ParticlePool method),

715 count\_columns() (pygamelib.gfx.ui.FormLayout *method*), 637 count\_columns() (pygamelib.gfx.ui.GridLayout method), 642 count\_rows() (pygamelib.gfx.ui.FormLayout method), count\_rows() (pygamelib.gfx.ui.GridLayout method), 642 COUNTERBORE (pygamelib.assets.graphics.MiscTechnicals attribute), 87 COUNTERCLOCKWISE\_ARROWS\_BUTTON (pygamelib.assets.graphics.Models attribute), COUNTERSINK (pygamelib.assets.graphics.MiscTechnicals attribute), 87 COUPLE\_WITH\_HEART (pygamelib.assets.graphics.Models attribute), 189 COW (pygamelib.assets.graphics.Models attribute), 189 COW\_FACE (pygamelib.assets.graphics.Models attribute), 189 COWBOY\_HAT\_FACE (pygamelib.assets.graphics.Models attribute), 189 CRAB (pygamelib.assets.graphics.Models attribute), 189 CRAYON (pygamelib.assets.graphics.Models attribute), 189 create\_config() (pygamelib.engine.Game method), 550 CREDIT\_CARD (pygamelib.assets.graphics.Models tribute), 189 CRESCENT\_MOON (pygamelib.assets.graphics.Models attribute), 189 CRICKET (pygamelib.assets.graphics.Models attribute), 189 CRICKET\_GAME (pygamelib.assets.graphics.Models attribute), 189 CROCODILE (pygamelib.assets.graphics.Models attribute), 189 CROISSANT (pygamelib.assets.graphics.Models attribute), 189 CROSS\_MARK (pygamelib.assets.graphics.Models *tribute*), 189 CROSS\_MARK\_BUTTON (pygamelib.assets.graphics.Models attribute), 189 CROSSED\_FINGERS (pygamelib.assets.graphics.Models attribute), 189 CROSSED\_FLAGS (pygamelib.assets.graphics.Models attribute), 189 CROSSED\_SWORDS (pygamelib.assets.graphics.Models attribute), 189 CROWN (pygamelib.assets.graphics.Models attribute), 189 CRYING\_CAT (pygamelib.assets.graphics.Models

Index 775

*tribute*), 189

CRYING\_FACE (pygamelib.assets.graphics.Models

- tribute), 189
- CRYSTAL\_BALL (pygamelib.assets.graphics.Models attribute), 189
- CUCUMBER (pygamelib.assets.graphics.Models attribute), 189
- CUP\_WITH\_STRAW (pygamelib.assets.graphics.Models attribute), 189
- CUPCAKE (pygamelib.assets.graphics.Models attribute), 189
- CURLING\_STONE (pygamelib.assets.graphics.Models attribute), 189
- CURLY\_BRACKET\_EXTENSION
  - (pygamelib.assets.graphics.MiscTechnicals attribute), 87
- CURLY\_HAIR (pygamelib.assets.graphics.Models attribute), 189
- CURLY\_LOOP (pygamelib.assets.graphics.Models attribute), 190
- CURRENCY\_EXCHANGE (pygamelib.assets.graphics.Models attribute), 190
- current (pygamelib.base.History property), 242
- current\_choice (pygamelib.gfx.ui.GridSelector property), 648
- current\_entry() (pygamelib.gfx.ui.Menu method), 665 current\_entry() (pygamelib.gfx.ui.MenuBar method), 671
- current\_frame() (pygamelib.gfx.core.Animation method), 582
- current\_index (pygamelib.gfx.ui.MenuBar property),
  671
- current\_page (pygamelib.gfx.ui.GridSelector property),
  648
- current\_path() (pygamelib.actuators.PathFinder method), 25
- current\_waypoint() (pygamelib.actuators.PathFinder method), 25
- CURRY\_RICE (pygamelib.assets.graphics.Models attribute), 190
- Cursor (class in pygamelib.gfx.ui), 627
- cursor (pygamelib.gfx.ui.LineInput property), 657
- cursor\_down() (pygamelib.gfx.ui.GridSelector method), 648
- cursor\_left() (pygamelib.gfx.ui.GridSelector method), 648
- cursor\_right() (pygamelib.gfx.ui.GridSelector method), 649
- cursor\_up() (pygamelib.gfx.ui.GridSelector method),
- CUSTARD (pygamelib.assets.graphics.Models attribute), 190

- CUSTOMS (pygamelib.assets.graphics.Models attribute), 190
- CUT\_OF\_MEAT (pygamelib.assets.graphics.Models attribute), 190
- cyan() (pygamelib.base.Text static method), 252
- cyan\_dim() (pygamelib.base.Text static method), 252
- cyan\_rect() (pygamelib.gfx.core.Sprixel class method),
  606
- cyan\_rect() (pygamelib.gfx.particles.ParticleSprixel class method), 726
- cyan\_square() (pygamelib.gfx.core.Sprixel class method), 606
- cyan\_square() (pygamelib.gfx.particles.ParticleSprixel class method), 726
- CYCLONE (pygamelib.assets.graphics.Models attribute), 190
- CYLINDRICITY (pygamelib.assets.graphics.MiscTechnicals attribute), 87

### D

- DAGGER (pygamelib.assets.graphics.Models attribute), 190
- DANGO (pygamelib.assets.graphics.Models attribute), 190
- DARK\_SHADE (pygamelib.assets.graphics.Blocks attribute), 42
- DARK\_SKIN\_TONE (pygamelib.assets.graphics.Models attribute), 190
- DASHING\_AWAY (pygamelib.assets.graphics.Models attribute), 190
- DEAF\_PERSON (pygamelib.assets.graphics.Models attribute), 190
- debug() (pygamelib.base.Text static method), 252
- debug\_info() (pygamelib.board\_items.Actionable method), 264
- debug\_info() (pygamelib.board\_items.BoardComplexItem method), 285
- debug\_info() (pygamelib.board\_items.BoardItem
   method), 306
- ${\tt debug\_info()}\ (pygamelib.board\_items. Board Item Complex Component\ method), 295$
- debug\_info() (pygamelib.board\_items.BoardItemVoid method), 315
- debug\_info() (pygamelib.board\_items.Camera method), 324

- debuq\_info() (pygamelib.board items.ComplexPlayer DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTAL method), 366
- debug\_info() (pygamelib.board items.ComplexTreasure *method*), 377
- debug\_info() (pygamelib.board\_items.ComplexWall *method*), 388
- debug\_info() (pygamelib.board items.Door method),
- debug\_info() (pygamelib.board items.GenericActionableStructure tribute), 87 *method*), 408
- debug\_info() (pygamelib.board\_items.GenericStructure method), 427
- method), 417
- debug\_info() (pygamelib.board\_items.Immovable method), 436
- debug\_info() (pygamelib.board\_items.Movable method), 446
- debug\_info() (pygamelib.board\_items.NPC method),
- debug\_info() (pygamelib.board\_items.Player method), 466
- debug\_info() (pygamelib.board\_items.Projectile method), 477
- debug\_info() (pygamelib.board\_items.TextItem method), 489
- debug\_info() (pygamelib.board\_items.Tile method), 500
- debug\_info() (pygamelib.board\_items.Treasure method), 511
- debug\_info() (pygamelib.board\_items.Wall method), 521
- DECIDUOUS\_TREE (pygamelib.assets.graphics.Models attribute), 190
- DECIMAL\_EXPONENT\_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 87
- DECIMAL\_SEPARATOR\_KEY\_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 87
- DEER (pygamelib.assets.graphics.Models attribute), 190 DEFAULT\_SIZE (pygamelib.constants.SizeConstraint attribute), 530
- delete() (pygamelib.engine.Screen method), 573
- delete() (pygamelib.gfx.ui.LineInput method), 657
- delete\_all\_levels() (pygamelib.engine.Game method), 550
- delete\_item() (pygamelib.engine.Inventory method),
- delete\_items() (pygamelib.engine.Inventory method),
- delete\_level() (pygamelib.engine.Game method), 551 DELIVERY\_TRUCK (pygamelib.assets.graphics.Models attribute), 190

- (pygamelib.assets.graphics.MiscTechnicals attribute), 87
- DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTAL\_WITH\_CIRCLE (pygamelib.assets.graphics.MiscTechnicals attribute), 87
- DENTISTRY SYMBOL LIGHT DOWN AND HORIZONTAL WITH TRIANGLE (pygamelib.assets.graphics.MiscTechnicals at-
- DENTISTRY\_SYMBOL\_LIGHT\_DOWN\_AND\_HORIZONTAL\_WITH\_WAVE (pygamelib.assets.graphics.MiscTechnicals at-
- debug\_info() (pygamelib.board\_items.GenericStructureCDENTEXTRMIpSNMMPOL\_LIGHT\_UP\_AND\_HORIZONTAL (pygamelib.assets.graphics.MiscTechnicals attribute), 88

tribute), 88

- DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_WITH\_CIRCLE (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_WITH\_TRIANGLE (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_UP\_AND\_HORIZONTAL\_WITH\_WAVE (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_BOTTOM\_LEFT (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_BOTTOM\_RIGHT (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_TOP\_LEFT (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_TOP\_RIGHT (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_AND\_WAVE (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_WITH\_CIRCLE (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DENTISTRY\_SYMBOL\_LIGHT\_VERTICAL\_WITH\_TRIANGLE (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- DEPARTMENT\_STORE (pygamelib.assets.graphics.Models attribute), 190
- DERELICT\_HOUSE (pygamelib.assets.graphics.Models attribute), 190
- DESERT (pygamelib.assets.graphics.Models attribute),
- DESERT\_ISLAND (pygamelib.assets.graphics.Models at*tribute*), 190
- DESKTOP\_COMPUTER (pygamelib.assets.graphics.Models

attribute), 190	<pre>detach() (pygamelib.board_items.TextItem method),</pre>
detach() (pygamelib.actuators.Actuator method), 8	489
detach() (pygamelib.actuators.Behavioral method), 12	<pre>detach() (pygamelib.board_items.Tile method), 500</pre>
<pre>detach() (pygamelib.actuators.PathActuator method),</pre>	detach() (pygamelib.board_items.Treasure method),
15	511
detach() (pygamelib.actuators.PathFinder method), 26	<pre>detach() (pygamelib.board_items.Wall method), 521</pre>
<pre>detach() (pygamelib.actuators.PatrolActuator method),</pre>	detach() (pygamelib.engine.Board method), 536
20	detach() (pygamelib.engine.Game method), 551
detach() (pygamelib.actuators.RandomActuator	detach() (pygamelib.engine.Inventory method), 563
method), 32	detach() (pygamelib.engine.Screen method), 574
detach() (pygamelib.actuators.UnidirectionalActuator	detach() (pygamelib.gfx.core.Color method), 614
method), 36	detach() (pygamelib.gfx.core.Sprite method), 596
detach() (pygamelib.base.PglBaseObject method), 246	detach() (pygamelib.gfx.core.Sprixel method), 606
detach() (pygamelib.base.Text method), 252	detach() (pygamelib.gfx.particles.CircleEmitter
detach() (pygamelib.board_items.Actionable method),	method), 691
265	detach() (pygamelib.gfx.particles.ColorParticle
	method), 697
method), 275	detach() (pygamelib.gfx.particles.ColorPartitionParticle
detach() (pygamelib.board_items.BoardComplexItem	method), 703
method), 285	detach() (pygamelib.gfx.particles.Particle method), 719
detach() (pygamelib.board_items.BoardItem method),	detach() (pygamelib.gfx.particles.ParticleEmitter
306	method), 711
detach() (pygamelib.board_items.BoardItemComplexCom	
method), 295	method), 726
detach() (pygamelib.board_items.BoardItemVoid	detach() (pygamelib.gfx.particles.PartitionParticle
method), 315	method), 735
detach() (pygamelib.board_items.Camera method), 324	detach() (pygamelib.gfx.particles.RandomColorParticle
<pre>detach() (pygamelib.board_items.Character method),</pre>	method), 740
334	$\verb"detach" (pygame lib.gfx.particles.Random Color Partition Particles and on Color Partition Particles and on Color Partition Particles and on Color Partition Particles and Only Partition Partition Particles and Only Partition Partitio$
detach() (pygamelib.board_items.ComplexDoor	method), 746
method), 344	detach() (pygamelib.gfx.ui.BoxLayout method), 621
detach() (pygamelib.board_items.ComplexNPC	detach() (pygamelib.gfx.ui.Cursor method), 629
method), 355	<pre>detach() (pygamelib.gfx.ui.FormLayout method), 638</pre>
<pre>detach() (pygamelib.board_items.ComplexPlayer</pre>	<pre>detach() (pygamelib.gfx.ui.GridLayout method), 642</pre>
method), 366	detach() (pygamelib.gfx.ui.Layout method), 651
<pre>detach() (pygamelib.board_items.ComplexTreasure</pre>	detach() (pygamelib.gfx.ui.LineInput method), 657
method), 377	detach() (pygamelib.gfx.ui.Widget method), 686
detach() (pygamelib.board_items.ComplexWall	DETECTIVE (pygamelib.assets.graphics.Models at-
method), 388	tribute), 190
detach() (pygamelib.board_items.Door method), 399	Dialog (class in pygamelib.gfx.ui), 632
- 110	CDFAMETER_SIGN (pygamelib.assets.graphics.MiscTechnicals
method), 409	attribute), 88
detach() (pygamelib.board_items.GenericStructure	
method), 427	attribute), 190
	laxxAMnnDnwifTH_A_DOT (pygamelib.assets.graphics.Models
method), 417	attribute), 190
detach() (pygamelib.board_items.Immovable method),	DIM_BUTTON (pygamelib.assets.graphics.Models at-
436	tribute), 190
detach() (pygamelib.board_items.Movable method),	DIMENSION_ORIGIN (pygamelib.assets.graphics.MiscTechnicals
446	attribute), 88
detach() (pygamelib.board_items.NPC method), 456	DIRECT_CURRENT_SYMBOL_FORM_TWO
detach() (pygamelib.board_items.Player method), 466	(pygamelib.assets.graphics.MiscTechnicals
detach() (pygamelib.board_items.Projectile method),	attribute), 88
477	DIRECT_HIT (pygamelib.assets.graphics.Models at-

tribute), 190	display() (pygamelib.board_items.Player method), 467
<pre>direction (pygamelib.board_items.Projectile property),</pre>	display() (pygamelib.board_items.Projectile method), 478
<pre>directional_animation()</pre>	<pre>display() (pygamelib.board_items.TextItem method),</pre>
(pygamelib.board_items.Projectile method),	490
478	display() (pygamelib.board_items.Tile method), 501
<pre>directional_model()</pre>	display() (pygamelib.board_items.Treasure method),
(pygamelib.board_items.Projectile method),	512
478	display() (pygamelib.board_items.Wall method), 522
DISAPPOINTED_FACE (pygamelib.assets.graphics.Models	display() (pygamelib.engine.Board method), 536
attribute), 190	display_around() (pygamelib.engine.Board method),
DISCONTINUOUS_UNDERLINE_SYMBOL	536
(pygamelib.assets.graphics.MiscTechnicals attribute), 88	display_at() (pygamelib.engine.Screen method), 574 display_board() (pygamelib.engine.Game method),
DISGUISED_FACE (pygamelib.assets.graphics.Models at-	551
tribute), 190	display_line() (pygamelib.engine.Screen method),
display() (pygamelib.board_items.Actionable method),	575
265	display_player_stats() (pygamelib.engine.Game
display() (pygamelib.board_items.ActionableTile	method), 552
method), 275	<pre>display_sprite() (pygamelib.engine.Screen method),</pre>
<pre>display() (pygamelib.board_items.BoardComplexItem</pre>	575
method), 286	display_sprite_at() (pygamelib.engine.Screen
<pre>display() (pygamelib.board_items.BoardItem method),</pre>	method), 576
307	distance() (pygamelib.base.Math static method), 243
${\tt display()} \ (pygame lib.board\_items. Board Item Complex $	
method), 296	method), 265
display() (pygamelib.board_items.BoardItemVoid method), 316	<pre>distance_to() (pygamelib.board_items.ActionableTile</pre>
display() (pygamelib.board_items.Camera method),	distance_to() (pygamelib.board_items.BoardComplexItem
325	method), 286
display() (pygamelib.board_items.Character method),	distance_to() (pygamelib.board_items.BoardItem
335	method), 307
display() (pygamelib.board_items.ComplexDoor	${\tt distance\_to()}\ (pygame lib. board\_items. Board Item Complex Component$
method), 345	method), 296
display() (pygamelib.board_items.ComplexNPC method), 356	<pre>distance_to() (pygamelib.board_items.BoardItemVoid</pre>
display() (pygamelib.board_items.ComplexPlayer	distance_to() (pygamelib.board_items.Camera
method), 367	method), 325
display() (pygamelib.board_items.ComplexTreasure	distance_to() (pygamelib.board_items.Character
method), 378	method), 335
display() (pygamelib.board_items.ComplexWall method), 389	<pre>distance_to() (pygamelib.board_items.ComplexDoor method), 345</pre>
display() (pygamelib.board_items.Door method), 400	distance_to() (pygamelib.board_items.ComplexNPC
display() (pygamelib.board_items.GenericActionableStr	
method), 409	distance_to() (pygamelib.board_items.ComplexPlayer
display() (pygamelib.board_items.GenericStructure	method), 367
method), 428	distance_to() (pygamelib.board_items.ComplexTreasure
${\tt display()} \ (pygame lib. board\_items. Generic Structure Communications) \\$	plexCompo <b>meth</b> od), 378
method), 418	distance_to() (pygamelib.board_items.ComplexWall
${\tt display()}\ (pygame lib. board\_items. Immovable\ method),$	method), 389
437	distance_to() (pygamelib.board_items.Door method),
display() (pygamelib.board_items.Movable method),	400
dianlar () (magnetic beautitous NDC mathed) 457	distance_to() (pygamelib.board_items.GenericActionableStructure
display() (pygamelib.board_items.NPC method), 457	method), 409

<pre>distance_to() (pygamelib.board_items.GenericStructure</pre>	DOUBLE_DOWN_AND_LEFT
method), 428	(pygamelib.assets.graphics.BoxDrawings
$\verb distance_to()  (pygame lib.board\_items. Generic Structure) $	
method), 418	DOUBLE_DOWN_AND_RIGHT
distance_to() (pygamelib.board_items.Immovable	(pygamelib.assets.graphics.BoxDrawings
method), 437	attribute), 52
distance_to() (pygamelib.board_items.Movable	DOUBLE_EXCLAMATION_MARK
method), 447	(pygamelib.assets.graphics.Models attribute),
distance_to() (pygamelib.board_items.NPC method),	191
457	DOUBLE_HORIZONTAL (pygamelib.assets.graphics.BoxDrawings
distance_to() (pygamelib.board_items.Player	attribute), 52
method), 467	DOUBLE_UP_AND_HORIZONTAL
distance_to() (pygamelib.board_items.Projectile	(pygamelib.assets.graphics.BoxDrawings
method), 478	attribute), 52
distance_to() (pygamelib.board_items.TextItem	DOUBLE_UP_AND_LEFT (pygamelib.assets.graphics.BoxDrawings
<pre>method), 490 distance_to() (pygamelib.board_items.Tile method),</pre>	attribute), 52 DOUBLE_UP_AND_RIGHT
501	(pygamelib.assets.graphics.BoxDrawings
distance_to() (pygamelib.board_items.Treasure	attribute), 52
method), 512	DOUBLE_VERTICAL (pygamelib.assets.graphics.BoxDrawings
distance_to() (pygamelib.board_items.Wall method),	attribute), 52
522	DOUBLE_VERTICAL_AND_HORIZONTAL
DIVIDE (pygamelib.assets.graphics.Models attribute),	(pygamelib.assets.graphics.BoxDrawings
190	attribute), 52
DIVING_MASK (pygamelib.assets.graphics.Models at-	DOUBLE_VERTICAL_AND_LEFT
<i>tribute</i> ), 190	(pygamelib.assets.graphics.BoxDrawings
DIYA_LAMP (pygamelib.assets.graphics.Models at-	attribute), 52
tribute), 190	DOUBLE_VERTICAL_AND_RIGHT
DIZZY (pygamelib.assets.graphics.Models attribute), 190	(pygamelib.assets.graphics.BoxDrawings
DIZZY_FACE (pygamelib.assets.graphics.Models at-	attribute), 52
tribute), 191	DOUBLE_VERTICAL_BAR
DLDOWN (pygamelib.constants.Direction attribute), 529	(pygamelib.assets.graphics.MiscTechnicals
DLUP (pygamelib.constants.Direction attribute), 529	attribute), 88
DNA (pygamelib.assets.graphics.Models attribute), 191	DOUGHNUT (pygamelib.assets.graphics.Models attribute),
DODO (pygamelib.assets.graphics.Models attribute), 191	191
DOG (pygamelib.assets.graphics.Models attribute), 191	DOVE (pygamelib.assets.graphics.Models attribute), 191
DOG_FACE (pygamelib.assets.graphics.Models attribute),	DOWN (pygamelib.constants.Direction attribute), 528
191	DOWN_ARROW (pygamelib.assets.graphics.Models at-
DOLLAR_BANKNOTE (pygamelib.assets.graphics.Models	tribute), 191
attribute), 191	DOWN_ARROWHEAD (pygamelib.assets.graphics.MiscTechnicals
DOLPHIN (pygamelib.assets.graphics.Models attribute),	attribute), 88
191	DOWN_DOUBLE_AND_HORIZONTAL_SINGLE
Door (class in pygamelib.board_items), 396	(pygamelib.assets.graphics.BoxDrawings
DOOR (pygamelib.assets.graphics.Models attribute), 191	attribute), 52
DOTTED_CIRCLE (pygamelib.assets.graphics.GeometricSha	•
attribute), 64 DOTTED_SIX_POINTED_STAR	(pygamelib.assets.graphics.BoxDrawings attribute), 52
(pygamelib.assets.graphics.Models attribute),	DOWN_DOUBLE_AND_RIGHT_SINGLE
(pygameno.asseis.grapnics.modeis anribute), 191	
DOUBLE_CURLY_LOOP (pygamelib.assets.graphics.Models	(pygamelib.assets.graphics.BoxDrawings attribute), 52
attribute), 191	DOWN_HEAVY_AND_HORIZONTAL_LIGHT
DOUBLE_DOWN_AND_HORIZONTAL	(pygamelib.assets.graphics.BoxDrawings
(pygamelib.assets.graphics.BoxDrawings	attribute), 52
attribute) 52	DOWN HEAVY AND LEFT LIGHT

DRDOWN (pygamelib.constants.Direction attribute), 529 (pygamelib.assets.graphics.BoxDrawings attribute), 53 DRESS (pygamelib.assets.graphics.Models attribute), 191 DOWN\_HEAVY\_AND\_LEFT\_UP\_LIGHT DROOLING\_FACE (pygamelib.assets.graphics.Models at-(pygamelib.assets.graphics.BoxDrawings tribute), 191 attribute), 53 DROP\_OF\_BLOOD (pygamelib.assets.graphics.Models at-DOWN\_HEAVY\_AND\_RIGHT\_LIGHT tribute), 191 (pygamelib.assets.graphics.BoxDrawings DROPLET (pygamelib.assets.graphics.Models attribute), attribute), 53 DOWN\_HEAVY\_AND\_RIGHT\_UP\_LIGHT DRUM (pygamelib.assets.graphics.Models attribute), 191 (pygamelib.assets.graphics.BoxDrawings DRUP (pygamelib.constants.Direction attribute), 529 attribute), 53 dtanimate (pygamelib.gfx.core.Animation property), DOWN\_HEAVY\_AND\_UP\_HORIZONTAL\_LIGHT dtmove (pygamelib.board\_items.Camera property), 325 (pygamelib.assets.graphics.BoxDrawings dtmove (pygamelib.board\_items.Character property), attribute), 53 DOWN\_LEFT\_ARROW (pygamelib.assets.graphics.Models attribute), 191 dtmove (pygamelib.board\_items.ComplexNPC property), DOWN\_LIGHT\_AND\_HORIZONTAL\_HEAVY (pygamelib.assets.graphics.BoxDrawings dtmove (pygamelib.board\_items.ComplexPlayer propattribute), 53 erty), 367 DOWN\_LIGHT\_AND\_LEFT\_HEAVY dtmove (pygamelib.board items.Movable property), 447 (pygamelib.assets.graphics.BoxDrawings dtmove (pygamelib.board\_items.NPC property), 457 attribute), 53 dtmove (pygamelib.board items.Player property), 467 DOWN\_LIGHT\_AND\_LEFT\_UP\_HEAVY dtmove (pygamelib.board\_items.Projectile property), (pygamelib.assets.graphics.BoxDrawings attribute), 53 DUCK (pygamelib.assets.graphics.Models attribute), 191 DOWN\_LIGHT\_AND\_RIGHT\_HEAVY DUMPLING (pygamelib.assets.graphics.Models attribute), (pygamelib.assets.graphics.BoxDrawings attribute), 53 DVD (pygamelib.assets.graphics.Models attribute), 191 DOWN\_LIGHT\_AND\_RIGHT\_UP\_HEAVY Ε (pygamelib.assets.graphics.BoxDrawings attribute), 53  ${\tt E\_MAIL} \quad (pygamelib. assets. graphics. Models \quad attribute),$ DOWN\_LIGHT\_AND\_UP\_HORIZONTAL\_HEAVY (pygamelib.assets.graphics.BoxDrawings EAGLE (pygamelib.assets.graphics.Models attribute), 191 attribute), 53 EAR (pygamelib.assets.graphics.Models attribute), 191 DOWN\_RIGHT\_ARROW (pygamelib.assets.graphics.Models EAR\_OF\_CORN (pygamelib.assets.graphics.Models atattribute), 191 tribute), 191 DOWN\_SINGLE\_AND\_HORIZONTAL\_DOUBLE EAR\_WITH\_HEARING\_AID (pygamelib.assets.graphics.BoxDrawings (pygamelib.assets.graphics.Models attribute), attribute), 53 DOWN\_SINGLE\_AND\_LEFT\_DOUBLE EARTH\_GROUND (pygamelib.assets.graphics.MiscTechnicals (pygamelib.assets.graphics.BoxDrawings attribute), 88 attribute), 53 EGG (pygamelib.assets.graphics.Models attribute), 192 DOWN\_SINGLE\_AND\_RIGHT\_DOUBLE EGGPLANT (pygamelib.assets.graphics.Models attribute), (pygamelib.assets.graphics.BoxDrawings 192 attribute), 53 EIGHT\_OCLOCK (pygamelib.assets.graphics.Models DOWNCAST\_FACE\_WITH\_SWEAT attribute), 192 (pygamelib.assets.graphics.Models attribute),  ${\tt EIGHT\_POINTED\_STAR}\ (pygame lib. assets. graphics. Models$ attribute), 192 DOWNWARDS\_BUTTON (pygamelib.assets.graphics.Models EIGHT\_SPOKED\_ASTERISK attribute), 191 (pygamelib.assets.graphics.Models attribute), DRAGON (pygamelib.assets.graphics.Models attribute), 192

Index 781

DRAGON\_FACE (pygamelib.assets.graphics.Models at-

tribute), 191

**EIGHT\_THIRTY** 

attribute), 192

(pygamelib.assets.graphics.Models

EJECT_BUTTON (pygamelib.assets.graphics.Models attribute), 192	attribute), 192 EXCLAMATION_QUESTION_MARK
EJECT_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 88	
ELECTRIC_ARROW (pygamelib.assets.graphics.MiscTechnicattribute), 88	<b>aEXPAND</b> (pygamelib.constants.SizeConstraint attribute), 530
ELECTRIC_PLUG (pygamelib.assets.graphics.Models attribute), 192	expand() (pygamelib.gfx.ui.Menu method), 665 EXPLODING_HEAD (pygamelib.assets.graphics.Models at-
ELECTRICAL_INTERSECTION	tribute), 192
(pygamelib.assets.graphics.MiscTechnicals attribute), 88	EXPRESSIONLESS_FACE (pygamelib.assets.graphics.Models attribute),
ELEPHANT (pygamelib.assets.graphics.Models attribute), 192	192
ELEVATOR (pygamelib.assets.graphics.Models attribute),	EYE (pygamelib.assets.graphics.Models attribute), 192 EYES (pygamelib.assets.graphics.Models attribute), 192
192 ELEVEN_OCLOCK (pygamelib.assets.graphics.Models at-	F
tribute), 192	FACE_BLOWING_A_KISS
ELEVEN_THIRTY (pygamelib.assets.graphics.Models attribute), 192	(pygamelib.assets.graphics.Models attribute), 192
ELF (pygamelib.assets.graphics.Models attribute), 192 emit() (pygamelib.gfx.particles.CircleEmitter method),	FACE_SAVORING_FOOD (pygamelib.assets.graphics.Models
691	attribute), 192 FACE_SCREAMING_IN_FEAR
emit() (pygamelib.gfx.particles.ParticleEmitter method), 711	(pygamelib.assets.graphics.Models attribute), 192
EmitterProperties (class in pygamelib.gfx.particles), 706	FACE_VOMITING (pygamelib.assets.graphics.Models attribute), 192
EMPHASIS_SYMBOL (pygamelib.assets.graphics.MiscTechni	FACE_WITH_HAND_OVER_MOUTH
attribute), 88 empty() (pygamelib.engine.Inventory method), 563	(pygamelib.assets.graphics.Models attribute), 193
empty() (pygamelib.gfx.core.Sprite method), 596	FACE_WITH_HEAD_BANDAGE
empty_marker (pygamelib.gfx.ui.ProgressBar property), 679	(pygamelib.assets.graphics.Models attribute), 193
end() (pygamelib.gfx.ui.LineInput method), 657	FACE_WITH_MEDICAL_MASK
END_ARROW (pygamelib.assets.graphics.Models attribute), 192	(pygamelib.assets.graphics.Models attribute), 193
ENTER_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 88	FACE_WITH_MONOCLE (pygamelib.assets.graphics.Models attribute), 193
entries (pygamelib.gfx.ui.Menu property), 665	FACE_WITH_OPEN_MOUTH
entries (pygamelib.gfx.ui.MenuBar property), 671 ENVELOPE (pygamelib.assets.graphics.Models attribute),	(pygamelib.assets.graphics.Models attribute), 193
192 ENVELOPE_WITH_ARROW	FACE_WITH_RAISED_EYEBROW
(pygamelib.assets.graphics.Models attribute), 192	(pygamelib.assets.graphics.Models attribute), 193
ERASE_TO_THE_LEFT (pygamelib.assets.graphics.MiscTecl attribute), 88	(p)gameno.assers.grapmes.mears anniente);
ERASE_TO_THE_RIGHT (pygamelib.assets.graphics.MiscTed	193 Chaicalwith steam from nose
attribute), 88	(pygamelib.assets.graphics.Models attribute),
EURO_BANKNOTE (pygamelib.assets.graphics.Models attribute), 192	193
EVERGREEN_TREE (pygamelib.assets.graphics.Models attribute), 192	FACE_WITH_SYMBOLS_ON_MOUTH (pygamelib.assets.graphics.Models attribute), 193
EWE (pygamelib.assets.graphics.Models attribute), 192	FACE_WITH_TEARS_OF_JOY
EXCLAMATION_MARK (pygamelib.assets.graphics.Models	(pygamelib.assets.graphics.Models attribute).

193

## FACE\_WITH\_THERMOMETER

(pygamelib.assets.graphics.Models attribute), 193

- FACE\_WITH\_TONGUE (pygamelib.assets.graphics.Models attribute), 193
- FACE\_WITHOUT\_MOUTH (pygamelib.assets.graphics.Models finished() attribute), 193 me.
- FACTORY (pygamelib.assets.graphics.Models attribute),
- FAIRY (pygamelib.assets.graphics.Models attribute), 193 FALAFEL (pygamelib.assets.graphics.Models attribute),
- FALLEN\_LEAF (pygamelib.assets.graphics.Models attribute), 193
- FAMILY (pygamelib.assets.graphics.Models attribute), 193
- FAST\_DOWN\_BUTTON (pygamelib.assets.graphics.Models attribute), 193
- FAST\_FORWARD\_BUTTON

(pygamelib.assets.graphics.Models attribute),

### FAST\_REVERSE\_BUTTON

(pygamelib.assets.graphics.Models attribute), 193

- FAST\_UP\_BUTTON (pygamelib.assets.graphics.Models attribute), 193
- fatal() (pygamelib.base.Text static method), 252
- FAX\_MACHINE (pygamelib.assets.graphics.Models attribute), 193
- FEARFUL\_FACE (pygamelib.assets.graphics.Models attribute), 193
- FEATHER (pygamelib.assets.graphics.Models attribute), 193
- FEMALE\_SIGN (pygamelib.assets.graphics.Models at tribute), 193
- FERRIS\_WHEEL (pygamelib.assets.graphics.Models attribute), 193
- FERRY (pygamelib.assets.graphics.Models attribute), 193 fg\_color (pygamelib.base.Text property), 252
- fg\_color (pygamelib.gfx.core.Sprixel property), 607
- fg\_color (pygamelib.gfx.particles.ParticleSprixel property), 727
- FIELD\_HOCKEY (pygamelib.assets.graphics.Models attribute), 193
- fields (pygamelib.gfx.ui.MultiLineInputDialog property), 677
- FILE\_CABINET (pygamelib.assets.graphics.Models attribute), 193
- FILE\_FOLDER (pygamelib.assets.graphics.Models attribute), 193
- FileDialog (class in pygamelib.gfx.ui), 633
- FILM\_FRAMES (pygamelib.assets.graphics.Models at tribute), 194

- FILM\_PROJECTOR (pygamelib.assets.graphics.Models attribute). 194
- filter (pygamelib.gfx.ui.FileDialog property), 634
- filter (pygamelib.gfx.ui.LineInput property), 657
- find\_path() (pygamelib.actuators.PathFinder method),
  26
- finished() (pygamelib.gfx.particles.CircleEmitter method), 692
- finished() (pygamelib.gfx.particles.ColorParticle method), 697
- finished() (pygamelib.gfx.particles.ColorPartitionParticle method), 703
- finished() (pygamelib.gfx.particles.Particle method), 719
- finished() (pygamelib.gfx.particles.ParticleEmitter method), 711
- finished() (pygamelib.gfx.particles.PartitionParticle method), 735
- finished() (pygamelib.gfx.particles.RandomColorParticle method), 741
- finished() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 746
- FIRE (pygamelib.assets.graphics.Models attribute), 194
- FIRE\_ENGINE (pygamelib.assets.graphics.Models attribute), 194
- FIRE\_EXTINGUISHER (pygamelib.assets.graphics.Models attribute), 194
- FIRECRACKER (pygamelib.assets.graphics.Models attribute), 194
- FIREWORKS (pygamelib.assets.graphics.Models attribute), 194
- FIRST\_PLACE\_MEDAL (pygamelib.assets.graphics.Models attribute), 194
- FIRST\_QUARTER\_MOON (pygamelib.assets.graphics.Models attribute), 194
- FIRST\_QUARTER\_MOON\_FACE
  - (pygamelib.assets.graphics.Models attribute), 194
- FISH (pygamelib.assets.graphics.Models attribute), 194 FISH\_CAKE\_WITH\_SWIRL
  - (pygamelib.assets.graphics.Models attribute), 194
- FISHEYE (pygamelib.assets.graphics.GeometricShapes attribute), 64
- FISHING\_POLE (pygamelib.assets.graphics.Models attribute), 194
- FIVE\_OCLOCK (pygamelib.assets.graphics.Models attribute), 194
- FIVE\_THIRTY (pygamelib.assets.graphics.Models attribute), 194
- FLAG\_IN\_HOLE (pygamelib.assets.graphics.Models attribute), 194
- FLAMINGO (pygamelib.assets.graphics.Models attribute), 194

- FLASHLIGHT (pygamelib.assets.graphics.Models attribute), 194
- FLAT\_SHOE (pygamelib.assets.graphics.Models attribute), 194
- FLATBREAD (pygamelib.assets.graphics.Models attribute), 194
- FLATNESS (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- FLEUR\_DE\_LIS (pygamelib.assets.graphics.Models attribute), 194
- FLEXED\_BICEPS (pygamelib.assets.graphics.Models attribute), 194
- flip\_horizontally() (pygamelib.gfx.core.Sprite method), 596
- flip\_vertically() (pygamelib.gfx.core.Sprite method), 597
- FLOPPY\_DISK (pygamelib.assets.graphics.Models attribute), 194
- FLOWER\_PLAYING\_CARDS
  - (pygamelib.assets.graphics.Models attribute), 194
- FLUSHED\_FACE (pygamelib.assets.graphics.Models attribute), 194
- FLY (pygamelib.assets.graphics.Models attribute), 194
- FLYING\_DISC (pygamelib.assets.graphics.Models attribute), 194
- FLYING\_SAUCER (pygamelib.assets.graphics.Models attribute), 194
- focus (pygamelib.gfx.ui.LineInput property), 657
- focus (pygamelib.gfx.ui.Widget property), 686
- FOG (pygamelib.assets.graphics.Models attribute), 194
- FOGGY (pygamelib.assets.graphics.Models attribute), 194
- FOLDED\_HANDS (pygamelib.assets.graphics.Models attribute), 194
- FONDUE (pygamelib.assets.graphics.Models attribute), 195
- Font (class in pygamelib.gfx.core), 585
- FOOT (pygamelib.assets.graphics.Models attribute), 195
- FOOTPRINTS (pygamelib.assets.graphics.Models at tribute), 195
- force\_render() (pygamelib.engine.Screen method), 576
- FORK\_AND\_KNIFE (pygamelib.assets.graphics.Models attribute), 195
- FORK\_AND\_KNIFE\_WITH\_PLATE
  - (pygamelib.assets.graphics.Models attribute), 195
- FormLayout (class in pygamelib.gfx.ui), 635
- FORTUNE\_COOKIE (pygamelib.assets.graphics.Models attribute), 195
- FOUNTAIN (pygamelib.assets.graphics.Models attribute), 195

- FOUNTAIN\_PEN (pygamelib.assets.graphics.Models attribute), 195
- FOUR\_LEAF\_CLOVER (pygamelib.assets.graphics.Models attribute), 195
- FOUR\_OCLOCK (pygamelib.assets.graphics.Models attribute), 195
- FOUR\_THIRTY (pygamelib.assets.graphics.Models attribute), 195
- FOX (pygamelib.assets.graphics.Models attribute), 195
- FRAMED\_PICTURE (pygamelib.assets.graphics.Models attribute), 195
- FREE\_BUTTON (pygamelib.assets.graphics.Models attribute), 195
- FRENCH\_FRIES (pygamelib.assets.graphics.Models attribute), 195
- FRIED\_SHRIMP (pygamelib.assets.graphics.Models attribute), 195
- FROG (pygamelib.assets.graphics.Models attribute), 195
- from\_ansi() (pygamelib.gfx.core.Color class method),
  614
- from\_ansi() (pygamelib.gfx.core.Sprixel static method),
  607
- from\_ansi() (pygamelib.gfx.particles.ParticleSprixel static method), 727
- from\_direction() (pygamelib.base.Vector2D class
   method), 258
- from\_text() (pygamelib.gfx.core.Sprite class method),
  597
- fromkeys() (pygamelib.gfx.core.SpriteCollection class method), 591
- FRONT\_FACING\_BABY\_CHICK
  - (pygamelib.assets.graphics.Models attribute), 195
- FROWN (pygamelib.assets.graphics.MiscTechnicals attribute), 88
- FROWNING\_FACE (pygamelib.assets.graphics.Models attribute), 195
- FROWNING\_FACE\_WITH\_OPEN\_MOUTH
  - (pygamelib.assets.graphics.Models attribute), 195
- FUEL\_PUMP (pygamelib.assets.graphics.Models attribute), 195
- FULL\_BLOCK (pygamelib.assets.graphics.Blocks attribute), 42
- FULL\_MOON (pygamelib.assets.graphics.Models attribute), 195
- FULL\_MOON\_FACE (pygamelib.assets.graphics.Models attribute), 195
- FUNERAL\_URN (pygamelib.assets.graphics.Models attribute), 195
- FUSE (pygamelib.assets.graphics.MiscTechnicals attribute), 88

G	(
g (pygamelib.gfx.core.Color property), 614	
Game (class in pygamelib.engine), 544	(
GAME_DIE (pygamelib.assets.graphics.Models attribute),	
195	ç
GARLIC (pygamelib.assets.graphics.Models attribute),	ç
195	(
GEAR (pygamelib.assets.graphics.Models attribute), 195	
GEM_STONE (pygamelib.assets.graphics.Models at-	(
tribute), 195	(
GEMINI (pygamelib.assets.graphics.Models attribute),	(
195	
generate_void_cell() (pygamelib.engine.Board	(
method), 536	`
GenericActionableStructure (class in	(
pygamelib.board_items), 406	
GenericStructure (class in pygamelib.board_items),	,
424	(
${\tt GenericStructureComplexComponent}  ({\it class}  {\it in} $	
pygamelib.board_items), 415	Ç
GENIE (pygamelib.assets.graphics.Models attribute), 195	Ç
GeometricShapes (class in pygamelib.assets.graphics),	(
56	
get() (pygamelib.engine.Screen method), 577	(
get() (pygamelib.gfx.core.SpriteCollection method), 591	
get_board() (pygamelib.engine.Game method), 552	ç
get_immovables() (pygamelib.engine.Board method),	
537	(
<pre>get_item() (pygamelib.engine.Inventory method), 564</pre>	
get_items() (pygametib.engine.Inventory method), 564 get_items() (pygamelib.engine.Inventory method), 564	ç
	(
get_key() (pygamelib.engine.Game static method), 552	
get_movables() (pygamelib.engine.Board method), 537	ç
<pre>get_particles() (pygamelib.gfx.particles.ParticlePool</pre>	:
method), 716	,
GHOST (pygamelib.assets.graphics.Models attribute), 195	ç
${\tt GIRAFFE} \ \ (pygamelib. assets. graphics. Models \ \ attribute),$	(
196	•
GIRL (pygamelib.assets.graphics.Models attribute), 196	,
${\tt GLASS\_OF\_MILK}\ (pygamelib.assets.graphics.Models\ at-$	(
tribute), 196	
${\tt GLASSES}\ (pygamelib. assets. graphics. Models\ attribute),$	ç
196	
GLOBE_SHOWING_AMERICAS	Ç
(pygamelib.assets.graphics.Models attribute),	
196	ç
GLOBE_SHOWING_ASIA_AUSTRALIA	
(pygamelib.assets.graphics.Models attribute),	(
196	(
GLOBE_SHOWING_EUROPE_AFRICA	(
(pygamelib.assets.graphics.Models attribute),	(
(p) (Source to to to be seen a full to the little of the l	

(pygamelib.assets.graphics.Models attribute),

196

196

GLOBE\_WITH\_MERIDIANS

GLOVES (pygamelib.assets.graphics.Models attribute), 196 GLOWING\_STAR (pygamelib.assets.graphics.Models attribute), 196 glyph() (pygamelib.gfx.core.Font method), 586 glyphs\_map (pygamelib.gfx.core.Font property), 587 GOAL\_NET (pygamelib.assets.graphics.Models attribute), GOAT (pygamelib.assets.graphics.Models attribute), 196 GOBLIN (pygamelib.assets.graphics.Models attribute), GOGGLES (pygamelib.assets.graphics.Models attribute), GORILLA (pygamelib.assets.graphics.Models attribute), 196 GRADUATION\_CAP (pygamelib.assets.graphics.Models attribute), 196 GRAPES (pygamelib.assets.graphics.Models attribute), green (pygamelib.gfx.ui.ColorPicker property), 627 green() (pygamelib.base.Text static method), 252 GREEN\_APPLE (pygamelib.assets.graphics.Models tribute), 196 GREEN\_BOOK (pygamelib.assets.graphics.Models *tribute*), 196 green\_bright() (pygamelib.base.Text static method), 253 GREEN\_CIRCLE (pygamelib.assets.graphics.Models attribute), 196 green\_dim() (pygamelib.base.Text static method), 253 GREEN\_HEART (pygamelib.assets.graphics.Models attribute), 196 green\_rect() (pygamelib.gfx.core.Sprixel method), 607 green\_rect() (pygamelib.gfx.particles.ParticleSprixel class method), 727 GREEN\_SALAD (pygamelib.assets.graphics.Models attribute), 196 (pygamelib.assets.graphics.Models GREEN\_SQUARE attribute), 196 green\_square() (pygamelib.gfx.core.Sprixel method), 607 green\_square() (pygamelib.gfx.particles.ParticleSprixel class method), 727 grid\_selector (pygamelib.gfx.ui.GridSelectorDialog property), 646 GridLayout (class in pygamelib.gfx.ui), 640 GridSelector (class in pygamelib.gfx.ui), 647 GridSelectorDialog (class in pygamelib.gfx.ui), 645 GRIMACING\_FACE (pygamelib.assets.graphics.Models attribute), 196

(pygamelib.assets.graphics.Models

Index 785

GRINNING\_CAT

attribute), 196

GRINNING\_CAT\_WITH\_SMILING\_EYES

(pygamelib.assets.graphics.Models attribute), 196	handle_notification() (pygamelib.actuators.PathFinder method),
GRINNING_FACE (pygamelib.assets.graphics.Models attribute), 196	26 handle_notification()
tribute), 196 GRINNING_FACE_WITH_BIG_EYES	(pygamelib.actuators.PatrolActuator method),
(pygamelib.assets.graphics.Models attribute),	20
196	handle_notification()
GRINNING_FACE_WITH_SMILING_EYES	(pygamelib.actuators.RandomActuator
(pygamelib.assets.graphics.Models attribute),	method), 32
196	<pre>handle_notification()</pre>
GRINNING_FACE_WITH_SWEAT	(pygame lib. actuators. Unidirectional Actuator
(pygamelib.assets.graphics.Models attribute),	method), 36
196	handle_notification()
GRINNING_SQUINTING_FACE (pygamelib.assets.graphics.Models attribute),	(pygamelib.base.PglBaseObject method), 246
196	handle_notification() (pygamelib.base.Text
GROWING_HEART (pygamelib.assets.graphics.Models at-	method), 253
tribute), 197	handle_notification()
GUARD (pygamelib.assets.graphics.Models attribute), 197 GUIDE_DOG (pygamelib.assets.graphics.Models at-	(pygamelib.board_items.Actionable method), 265
tribute), 197	handle_notification()
GUITAR (pygamelib.assets.graphics.Models attribute),	(pygamelib.board_items.ActionableTile
197	method), 275
H	handle_notification()
	(pygamelib.board_items.BoardComplexItem
H_CENTER (pygamelib.constants.Alignment attribute),	<pre>method), 286 handle_notification()</pre>
528 HacException, 260	(pygamelib.board_items.BoardItem method),
HacInvalidLevelException, 260	307
HacInvalidTypeException, 260	handle_notification()
HacObjectIsNotMovableException, 260	(pygamelib.board_items.BoardItemComplexComponer
HacOutOfBoardBoundException, 261	method), 296
	handle_notification()
tribute), 197	(pygamelib.board_items.BoardItemVoid
HAMMER (pygamelib.assets.graphics.Models attribute),	method), 316
197	handle_notification()
HAMMER_AND_PICK (pygamelib.assets.graphics.Models attribute), 197	(pygamelib.board_items.Camera method), 325
HAMMER_AND_WRENCH (pygamelib.assets.graphics.Models	handle_notification()
attribute), 197	(pygamelib.board_items.Character method),
HAMSTER (pygamelib.assets.graphics.Models attribute),	335
197	handle_notification()
HAND_WITH_FINGERS_SPLAYED	(pygamelib.board_items.ComplexDoor
(pygamelib.assets.graphics.Models attribute),	<pre>method), 345 handle_notification()</pre>
197 HANDBAG (pygamelib.assets.graphics.Models attribute),	(pygamelib.board_items.ComplexNPC
197	method), 356
handle_notification()	handle_notification()
(pygamelib.actuators.Actuator method), 9 handle_notification()	(pygamelib.board_items.ComplexPlayer method), 367
(pygamelib.actuators.Behavioral method),	handle_notification()
12	(pygamelib.board_items.ComplexTreasure
handle_notification()	method), 378
(pygamelib.actuators.PathActuator method), 16	nanure_notification()

(pygamelib.board_items.ComplexWall method), 389	handle_notification() (pygamelib.gfx.particles.ColorParticle
handle_notification()	method), 697
$(pygamelib.board\_items.Door\ method), 400$	handle_notification()
handle_notification()	(pygame lib. gfx. particles. Color Partition Particle
$(pygame lib. board\_items. Generic Actionable Structure)$	ture method), 703
method), 409	handle_notification()
handle_notification()	(pygamelib.gfx.particles.Particle method),
(pygamelib.board_items.GenericStructure	719
method), 428	handle_notification()
handle_notification()	(pygamelib.gfx.particles.ParticleEmitter
(pygamelib.board_items.GenericStructureCompl	
method), 418	handle_notification()
handle_notification()	(pygamelib.gfx.particles.ParticleSprixel
(pygamelib.board_items.Immovable method),	method), 728
437	handle_notification()
handle_notification()	(pygamelib.gfx.particles.PartitionParticle
(pygamelib.board_items.Movable method),	method), 735
(pygameno.boara_nems.movable memoa), 447	handle_notification()
handle_notification()	
	(pygamelib.gfx.particles.RandomColorParticle
(pygamelib.board_items.NPC method), 457	method), 741
handle_notification()	handle_notification()
(pygamelib.board_items.Player method), 467	(pygamelib.gfx.particles.RandomColorPartitionParticle method), 747
handle_notification()	handle_notification() (pygamelib.gfx.ui.BoxLayout
(pygamelib.board_items.Projectile method),	method), 621
479	handle_notification() (pygamelib.gfx.ui.Cursor
handle_notification()	method), 629
(pygamelib.board_items.TextItem method),	handle_notification()
490	(pygamelib.gfx.ui.FormLayout method), 638
handle_notification() (pygamelib.board_items.Tile	handle_notification()
method), 501	(pygamelib.gfx.ui.GridLayout method), 643
handle_notification()	handle_notification() (pygamelib.gfx.ui.Layout
(pygamelib.board_items.Treasure method),	method), 651
512	handle_notification() (pygamelib.gfx.ui.LineInput
handle_notification() (pygamelib.board_items.Wall	method), 657
method), 522	1.0
handle_notification() (pygamelib.engine.Board	method), 686
method), 537	HANDSHAKE (pygamelib.assets.graphics.Models at-
handle_notification() (pygamelib.engine.Game	tribute), 197
method), 552	has_inventory() (pygamelib.board_items.Camera
handle_notification() (pygamelib.engine.Inventory	method), 325
method), 564	has_inventory() (pygamelib.board_items.Character
handle_notification() (pygamelib.engine.Screen	method), 335
method), 577	has_inventory()(pygamelib.board_items.ComplexNPC
handle_notification() (pygamelib.gfx.core.Color	method), 356
method), 615	has_inventory()(pygamelib.board_items.ComplexPlayer
handle_notification() (pygamelib.gfx.core.Sprite	method), 367
method), 597	has_inventory() (pygamelib.board_items.Movable
<pre>handle_notification() (pygamelib.gfx.core.Sprixel</pre>	method), 447
method), 608	has_inventory() (pygamelib.board_items.NPC
handle_notification()	method), 457
(pygamelib.gfx.particles.CircleEmitter	has_inventory() (pygamelib.board_items.Player
method), 692	method), 467

has_inventory() (pygamelib.board_items.Projectile method), 479	tribute), 197 HEAR_NO_EVIL_MONKEY
HATCHING_CHICK (pygamelib.assets.graphics.Models attribute), 197	(pygamelib.assets.graphics.Models attribute), 197
hcenter (pygamelib.engine.Screen property), 577	HEART_DECORATION (pygamelib.assets.graphics.Models
${\tt heading}\ ({\it pygame lib.board\_items.} Actionable\ {\it property}),$	attribute), 197
265	HEART_EXCLAMATION (pygamelib.assets.graphics.Models
heading (pygamelib.board_items.ActionableTile prop-	attribute), 197
erty), 275	HEART_SUIT (pygamelib.assets.graphics.Models at-
heading (pygamelib.board_items.BoardComplexItem property), 286	tribute), 197 HEART_WITH_ARROW (pygamelib.assets.graphics.Models
heading (pygamelib.board_items.BoardItem property),	attribute), 197
307	HEART_WITH_RIBBON (pygamelib.assets.graphics.Models
heading (pygamelib.board_items.BoardItemComplexCom	
property), 296	HEAVY_DOLLAR_SIGN (pygamelib.assets.graphics.Models
heading (pygamelib.board_items.BoardItemVoid prop-	attribute), 197
erty), 316	HEAVY_DOUBLE_DASH_HORIZONTAL
heading (pygamelib.board_items.Camera property), 325	(pygamelib.assets.graphics.BoxDrawings
heading (pygamelib.board_items.Character property),	attribute), 53
335	HEAVY_DOUBLE_DASH_VERTICAL
heading (pygamelib.board_items.ComplexDoor property), 345	(pygamelib.assets.graphics.BoxDrawings attribute), 53
heading (pygamelib.board_items.ComplexNPC prop-	HEAVY_DOWN (pygamelib.assets.graphics.BoxDrawings
erty), 357	attribute), 53
heading (pygamelib.board_items.ComplexPlayer prop-	HEAVY_DOWN_AND_HORIZONTAL
erty), 367	(pygamelib.assets.graphics.BoxDrawings
heading (pygamelib.board_items.ComplexTreasure	attribute), 53
property), 378	HEAVY_DOWN_AND_LEFT
${\tt heading} \ \ (pygamelib.board\_items. Complex Wall \ \ prop-$	(pygame lib. assets. graphics. Box Drawings
erty), 389	attribute), 53
heading (pygamelib.board_items.Door property), 400	HEAVY_DOWN_AND_RIGHT
heading (pygamelib.board_items.GenericActionableStruc property), 409	ture (pygamelib.assets.graphics.BoxDrawings attribute), 53
	HEAVY_HORIZONTAL (pygamelib.assets.graphics.BoxDrawing
property), 428	attribute), 53
${\tt heading} (pygame lib. board\_items. Generic Structure Complete St$	ex <b>HEANINOLLER</b> IT (pygamelib.assets.graphics.BoxDrawings
property), 418	attribute), 53
heading (pygamelib.board_items.Immovable property),	HEAVY_LEFT_AND_LIGHT_RIGHT
437	(pygamelib. assets. graphics. Box Drawings
heading (pygamelib.board_items.Movable property),	attribute), 53
447	HEAVY_QUADRUPLE_DASH_HORIZONTAL
heading (pygamelib.board_items.NPC property), 458	(pygamelib. assets. graphics. Box Drawings
heading (pygamelib.board_items.Player property), 467	attribute), 53
heading (pygamelib.board_items.Projectile property),	HEAVY_QUADRUPLE_DASH_VERTICAL
479	(pygamelib.assets.graphics.BoxDrawings
heading (pygamelib.board_items.TextItem property),	attribute), 53
490	HEAVY_RIGHT (pygamelib.assets.graphics.BoxDrawings
heading (pygamelib.board_items.Tile property), 501	attribute), 53
heading (pygamelib.board_items.Treasure property), 512	HEAVY_TRIPLE_DASH_HORIZONTAL
heading (pygamelib.board_items.Wall property), 522	(pygamelib.assets.graphics.BoxDrawings attribute), 53
HEADPHONE (pygamelib.assets.graphics.Models at-	HEAVY_TRIPLE_DASH_VERTICAL
tribute), 197	(pygamelib.assets.graphics.BoxDrawings
HEADSTONE (nyoamelih assets oranhics Models at-	attribute) 53

HEAVY_	UP (pygamelib.assets.graphics.BoxDrawings at-	erty), 429
	tribute), 53	$\label{lem:height} \textbf{height} (\textit{pygamelib.board\_items.} \textit{GenericStructureComplexComponent}$
HEAVY_	UP_AND_HORIZONTAL	property), 419
	(pygamelib.assets.graphics.BoxDrawings attribute), 53	height (pygamelib.board_items.Immovable property), 438
HEAVY_	UP_AND_LEFT (pygamelib.assets.graphics.BoxDrav	v <b>hæ</b> ight (pygamelib.board_items.Movable property), 448
	attribute), 53	height (pygamelib.board_items.NPC property), 458
HEAVY_	UP_AND_LIGHT_DOWN	height (pygamelib.board_items.Player property), 468
	(pygamelib.assets.graphics.BoxDrawings attribute), 54	height (pygamelib.board_items.Projectile property), 480
HEAVY_		whierigght (pygamelib.board_items.TextItem property), 491
	attribute), 54	height (pygamelib.board_items.Tile property), 502
HEAVY_	WERTICAL (pygamelib.assets.graphics.BoxDrawing	sheight (pygamelib.board_items.Treasure property), 513
	attribute), 54	height (pygamelib.board_items.Wall property), 523
HEAVY_	VERTICAL_AND_HORIZONTAL	height (pygamelib.engine.Board property), 538
	(pygamelib.assets.graphics.BoxDrawings	height (pygamelib.engine.Screen property), 577
	attribute), 54	height (pygamelib.gfx.core.Font property), 587
HEAVY_	VERTICAL_AND_LEFT	height (pygamelib.gfx.core.Sprite property), 598
	(pygamelib.assets.graphics.BoxDrawings	height (pygamelib.gfx.ui.Box property), 619
	attribute), 54	height (pygamelib.gfx.ui.BoxLayout property), 622
HEAVY_	VERTICAL_AND_RIGHT	height (pygamelib.gfx.ui.FormLayout property), 638
	(pygame lib. assets. graphics. Box Drawings	height (pygamelib.gfx.ui.GridLayout property), 643
	attribute), 54	height (pygamelib.gfx.ui.Layout property), 651
HEDGEH	OG (pygamelib.assets.graphics.Models attribute), 197	height (pygamelib.gfx.ui.LineInput property), 658 height (pygamelib.gfx.ui.MessageDialog property), 675
height		height (pygamelib.gfx.ui.Widget property), 687
	266	HELICOPTER (pygamelib.assets.graphics.Models at-
height	(pygamelib.board_items.ActionableTile property), 276	tribute), 197 HELM_SYMBOL (pygamelib.assets.graphics.MiscTechnicals
height	(pygamelib.board_items.BoardComplexItem	attribute), 88
	property), 287	HERB (pygamelib.assets.graphics.Models attribute), 197
height		HIBISCUS (pygamelib.assets.graphics.Models attribute),
	308	197
height	(pygamelib.board_items.BoardItemComplexComporproperty), 297	ohAGH_HEELED_SHOE (pygamelib.assets.graphics.Models attribute), 197
height	(pygamelib.board_items.BoardItemVoid property), 317	HIGH_SPEED_TRAIN (pygamelib.assets.graphics.Models attribute), 197
height	(pygamelib.board_items.Camera property), 326	HIGH_VOLTAGE (pygamelib.assets.graphics.Models
	(pygamelib.board_items.Character property), 336	attribute), 197 HIKING_BOOT (pygamelib.assets.graphics.Models at-
height		<i>tribute</i> ), 197
3	erty), 346	HINDU_TEMPLE (pygamelib.assets.graphics.Models
height	(pygamelib.board_items.ComplexNPC property), 357	attribute), 197 HIPPOPOTAMUS (pygamelib.assets.graphics.Models
height	(pygamelib.board_items.ComplexPlayer prop-	attribute), 198
nergne	erty), 368	History (class in pygamelib.base), 241
height	(pygamelib.board_items.ComplexTreasure prop-	hit() (pygamelib.board_items.Projectile method), 480
	erty), 379	HOLE (pygamelib.assets.graphics.Models attribute), 198
height	(pygamelib.board_items.ComplexWall property), 390	HOLLOW_RED_CIRCLE (pygamelib.assets.graphics.Models attribute), 198
_	(pygamelib.board_items.Door property), 401	home() (pygamelib.gfx.ui.LineInput method), 658
height	(pygamelib.board_items.GenericActionableStructu property), 410	rHONEY_POT (pygamelib.assets.graphics.Models at- tribute), 198
height	(pygamelib.board_items.GenericStructure prop-	1110utc), 170
	4 Jo	

HONEYBEE (pygamelib.assets.graphics.Models attribute), 198	(pygamelib.assets.graphics.MiscTechnicals attribute), 89
HOOK (pygamelib.assets.graphics.Models attribute), 198 HORIZONTAL (pygamelib.constants.Orientation attribute),	HOUSE (pygamelib.assets.graphics.MiscTechnicals attribute), 89
529	HOUSE (pygamelib.assets.graphics.Models attribute), 198
HORIZONTAL_LINE_EXTENSION	HOUSE_WITH_GARDEN (pygamelib.assets.graphics.Models
$(pygame lib. assets. graphics. {\it MiscTechnicals}$	attribute), 198
attribute), 88	HOUSES (pygamelib.assets.graphics.Models attribute),
HORIZONTAL_SCAN_LINE_1	198
(pygamelib.assets.graphics.MiscTechnicals attribute), 89	HUGGING_FACE (pygamelib.assets.graphics.Models attribute), 198
HORIZONTAL_SCAN_LINE_3	HUNDRED_POINTS (pygamelib.assets.graphics.Models at-
(pygame lib. assets. graphics. Misc Technicals	tribute), 198
attribute), 89	HUSHED_FACE (pygamelib.assets.graphics.Models at-
HORIZONTAL_SCAN_LINE_7	tribute), 198
(pygamelib.assets.graphics.MiscTechnicals attribute), 89	HUT (pygamelib.assets.graphics.Models attribute), 198 HYSTERESIS_SYMBOL (pygamelib.assets.graphics.MiscTechnicals
HORIZONTAL_SCAN_LINE_9	attribute), 89
(pygamelib.assets.graphics.MiscTechnicals attribute), 89	Í
horizontal_spacing (pygamelib.gfx.core.Font prop-	ICE (pygamelib.assets.graphics.Models attribute), 198
erty), 587	ICE_CREAM (pygamelib.assets.graphics.Models at-
horizontal_spacing (pygamelib.gfx.ui.FormLayout	tribute), 198
property), 638	ICE_HOCKEY (pygamelib.assets.graphics.Models at-
horizontal_spacing (pygamelib.gfx.ui.GridLayout	tribute), 198
property), 643	ICE_SKATE (pygamelib.assets.graphics.Models at-
HORIZONTAL_TRAFFIC_LIGHT	tribute), 198
(pygamelib.assets.graphics.Models attribute), 198	ID_BUTTON (pygamelib.assets.graphics.Models attribute), 198
HORSE (pygamelib.assets.graphics.Models attribute), 198	Immovable (class in pygamelib.board_items), 434
HORSE_FACE (pygamelib.assets.graphics.Models attribute), 198	INBOX_TRAY (pygamelib.assets.graphics.Models attribute), 199
HORSE_RACING (pygamelib.assets.graphics.Models	INCOMING_ENVELOPE (pygamelib.assets.graphics.Models
attribute), 198	attribute), 199
HOSPITAL (pygamelib.assets.graphics.Models attribute), 198	INDEX_POINTING_UP (pygamelib.assets.graphics.Models attribute), 199
HOT_BEVERAGE (pygamelib.assets.graphics.Models attribute), 198	INFINITY (pygamelib.assets.graphics.Models attribute), 199
HOT_DOG (pygamelib.assets.graphics.Models attribute),	info() (pygamelib.base.Text static method), 253
198	INFORMATION (pygamelib.assets.graphics.Models at-
HOT_FACE (pygamelib.assets.graphics.Models attribute),	tribute), 199
198	<pre>init_board() (pygamelib.engine.Board method), 538</pre>
HOT_PEPPER (pygamelib.assets.graphics.Models at-	<pre>init_cell() (pygamelib.engine.Board method), 538</pre>
tribute), 198	INPUT_LATIN_LETTERS
HOT_SPRINGS (pygamelib.assets.graphics.Models attribute), 198	(pygamelib.assets.graphics.Models attribute), 199
HOTEL (pygamelib.assets.graphics.Models attribute), 198	INPUT_LATIN_LOWERCASE
HOURGLASS (pygamelib.assets.graphics.MiscTechnicals attribute), 89	(pygamelib.assets.graphics.Models attribute), 199
HOURGLASS_DONE (pygamelib.assets.graphics.Models at-	INPUT_LATIN_UPPERCASE
tribute), 198	(pygamelib.assets.graphics.Models attribute),
${\tt HOURGLASS\_NOT\_DONE}\ (pygame lib. assets. graphics. Models$	199
attribute), 198	INPUT_NUMBERS (pygamelib.assets.graphics.Models at-
HOURGLASS_WITH_FLOWING_SAND	tribute), 199

INPUT_SYMBOLS (pygamelib.assets.graphics.Models at-	property), 419
tribute), 199	<pre>inventory_space (pygamelib.board_items.Immovable</pre>
<pre>insert_board() (pygamelib.engine.Game method), 553</pre>	property), 438
<pre>insert_characters() (pygamelib.gfx.ui.LineInput</pre>	inventory_space (pygamelib.board_items.Movable
method), 658	property), 448
INSERTION_SYMBOL (pygamelib.assets.graphics.MiscTechn	
attribute), 89	erty), 458
instance() (pygamelib.base.Console class method),	inventory_space (pygamelib.board_items.Player prop-
240	erty), 468
	* * * *
instance() (pygamelib.base.History class method), 242	
instance() (pygamelib.engine.Game class method), 553	property), 480
<pre>instance() (pygamelib.gfx.ui.UiConfig class method),</pre>	inventory_space (pygamelib.board_items.TextItem
684	property), 491
<pre>instantiate_item() (pygamelib.engine.Board static</pre>	<pre>inventory_space (pygamelib.board_items.Tile prop-</pre>
method), 538	erty), 502
INTEGER_FILTER (pygamelib.constants.InputValidator	<pre>inventory_space (pygamelib.board_items.Treasure</pre>
attribute), 529	property), 513
INTEGRAL_EXTENSION (pygamelib.assets.graphics.MiscTed	* * * * * * * * * * * * * * * * * * * *
attribute), 89	erty), 523
intersect() (pygamelib.base.Math static method), 244	INVERSE_BULLET (pygamelib.assets.graphics.GeometricShapes
Inventory (class in pygamelib.engine), 559	attribute), 64
inventory_space (pygamelib.board_items.Actionable	INVERSE_WHITE_CIRCLE
property), 266	(pygamelib.assets.graphics.GeometricShapes
inventory_space (pygamelib.board_items.ActionableTile	
property), 276	<pre>item() (pygamelib.board_items.ActionableTile method),</pre>
$\verb inventory_space   (pygame lib.board\_items. Board Complexity)   (pygame lib.board Compl$	
property), 287	item() (pygamelib.board_items.BoardComplexItem
<pre>inventory_space (pygamelib.board_items.BoardItem</pre>	method), 287
property), 308	<pre>item() (pygamelib.board_items.ComplexDoor method),</pre>
<pre>inventory_space (pygamelib.board_items.BoardItemCor</pre>	nplexCompositent
property), 297	item() (pygamelib.board_items.ComplexNPC method),
<pre>inventory_space(pygamelib.board_items.BoardItemVoid</pre>	
property), 317	item() (pygamelib.board_items.ComplexPlayer
inventory_space (pygamelib.board_items.Camera	method), 368
	item() (pygamelib.board_items.ComplexTreasure
property), 326	
inventory_space (pygamelib.board_items.Character	method), 379
property), 336	<pre>item() (pygamelib.board_items.ComplexWall method),</pre>
<pre>inventory_space (pygamelib.board_items.ComplexDoor</pre>	
property), 346	<pre>item() (pygamelib.board_items.TextItem method), 491</pre>
$\verb inventory_space   \textit{pygamelib.board\_items.ComplexNPC} $	
property), 358	item() (pygamelib.engine.Board method), 539
<pre>inventory_space(pygamelib.board_items.ComplexPlaye</pre>	ritems (pygamelib.engine.Inventory property), 565
property), 368	<pre>items() (pygamelib.gfx.core.SpriteCollection method),</pre>
<pre>inventory_space(pygamelib.board_items.ComplexTreas</pre>	1.0
property), 379	<pre>items_name() (pygamelib.engine.Inventory method),</pre>
<pre>inventory_space (pygamelib.board_items.ComplexWall</pre>	565
property), 390	<pre>items_per_page() (pygamelib.gfx.ui.GridSelector</pre>
inventory_space (pygamelib.board_items.Door prop-	method), 649
	memoa), 049
erty), 401	1.54
inventory_space(pygamelib.board_items.GenericAction	
property), 410	JACK_O_LANTERN (pygamelib.assets.graphics.Models at-
$\verb"inventory_space" (pygame lib.board\_items. Generic Struct")$	ure tribute), 199
property), 429	JAPANESE_ACCEPTABLE_BUTTON
$\verb"inventory_space" (pygame lib.board_items. Generic Struct")$	ureComplexComponentssets.graphics.Models attribute).

199	(pygamelib.assets.graphics.Models attribute),
JAPANESE_APPLICATION_BUTTON (pygamelib.assets.graphics.Models attribute),	JAPANESE_VACANCY_BUTTON
199	(pygamelib.assets.graphics.Models attribute),
JAPANESE_BARGAIN_BUTTON	200
(pygamelib.assets.graphics.Models attribute),	JEANS (pygamelib.assets.graphics.Models attribute), 200 JOKER (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_CASTLE (pygamelib.assets.graphics.Models attribute), 199	JOYSTICK (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_CONGRATULATIONS_BUTTON	
(pygamelib.assets.graphics.Models attribute),	KAABA (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_DISCOUNT_BUTTON	KANGAROO (pygamelib.assets.graphics.Models attribute),
(pygamelib.assets.graphics.Models attribute),	200 KEY (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_DOLLS (pygamelib.assets.graphics.Models attribute), 199	KEYBOARD (pygamelib.assets.graphics.MiscTechnicals attribute), 89
JAPANESE_FREE_OF_CHARGE_BUTTON	KEYBOARD (pygamelib.assets.graphics.Models attribute),
(pygamelib.assets.graphics.Models attribute),	200 keys() (pygamelib.gfx.core.SpriteCollection method),
JAPANESE_HERE_BUTTON	591
(pygamelib.assets.graphics.Models attribute),	KICK_SCOOTER (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_MONTHLY_AMOUNT_BUTTON	KIMONO (pygamelib.assets.graphics.Models attribute),
(pygamelib.assets.graphics.Models attribute),	200 KISS (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_NO_VACANCY_BUTTON	KISS_MARK (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.Models attribute),	<i>tribute</i> ), 200
199	KISSING_CAT (pygamelib.assets.graphics.Models at-
JAPANESE_NOT_FREE_OF_CHARGE_BUTTON	tribute), 200
(pygamelib.assets.graphics.Models attribute), 199	KISSING_FACE (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_OPEN_FOR_BUSINESS_BUTTON	KISSING_FACE_WITH_CLOSED_EYES
(pygamelib.assets.graphics.Models attribute), 199	(pygamelib.assets.graphics.Models attribute), 200
JAPANESE_PASSING_GRADE_BUTTON	KISSING_FACE_WITH_SMILING_EYES
(pygamelib.assets.graphics.Models attribute), 199	(pygamelib.assets.graphics.Models attribute), 200
JAPANESE_POST_OFFICE	KITCHEN_KNIFE (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.Models attribute), 199	tribute), 200 KITE (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_PROHIBITED_BUTTON	KIWI_FRUIT (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.Models attribute), 199	tribute), 200 KNOT (pygamelib.assets.graphics.Models attribute), 200
JAPANESE_RESERVED_BUTTON	KOALA (pygamelib.assets.graphics.Models attribute), 200
(pygamelib.assets.graphics.Models attribute), 199	L
JAPANESE_SECRET_BUTTON	LAB_COAT (pygamelib.assets.graphics.Models attribute),
(pygamelib.assets.graphics.Models attribute), 199	200
JAPANESE_SERVICE_CHARGE_BUTTON	LABEL (pygamelib.assets.graphics.Models attribute), 200 label (pygamelib.gfx.ui.LineInputDialog property), 662
(pygamelib.assets.graphics.Models attribute), 199	label (pygamelib.gfx.ui.ProgressDialog property), 681
JAPANESE_SYMBOL_FOR_BEGINNER	LACROSSE (pygamelib.assets.graphics.Models attribute), 200

LADDE	R (pygamelib.assets.graphics.Models attribute), 200	layer (pygamelib.board_items.Player property), 468 layer (pygamelib.board_items.Projectile property), 481
LADY_I	BEETLE (pygamelib.assets.graphics.Models attribute), 200	layer (pygamelib.board_items.TextItem property), 491 layer (pygamelib.board_items.Tile property), 502
LAPTO	P (pygamelib.assets.graphics.Models attribute), 200	layer (pygamelib.board_items.Treasure property), 513 layer (pygamelib.board_items.Wall property), 523
LARGE_	_BLUE_DIAMOND (pygamelib.assets.graphics.Models attribute), 200	layers() (pygamelib.engine.Board method), 539 Layout (class in pygamelib.gfx.ui), 649
	_CIRCLE (pygamelib.assets.graphics.GeometricShap attribute), 64	layout (pygamelib.gfx.ui.Widget property), 687
LARGE	_ORANGE_DIAMOND	LEAF_FLUTTERING_IN_WIND
	(pygamelib.assets.graphics.Models attribute), 200	(pygamelib.assets.graphics.Models attribute), 201
	QUARTER_MOON (pygamelib.assets.graphics.Models attribute), 200	LEAFY_GREEN (pygamelib.assets.graphics.Models attribute), 201
LAST_(	QUARTER_MOON_FACE (pygamelib.assets.graphics.Models attribute),	LEDGER (pygamelib.assets.graphics.Models attribute), 201
LAST_	200 IRACK_BUTTON (pygamelib.assets.graphics.Models	LEFT (pygamelib.constants.Alignment attribute), 528 LEFT (pygamelib.constants.Direction attribute), 528
	attribute), 201	LEFT_ARROW (pygamelib.assets.graphics.Models at-
	_CROSS (pygamelib.assets.graphics.Models at- tribute), 201	tribute), 201 LEFT_ARROW_CURVING_RIGHT
-	(pygamelib.board_items.Actionable property), 266	(pygamelib.assets.graphics.Models attribute),
	(pygamelib.board_items.ActionableTile property), 276	201 LEFT_CEILING (pygamelib.assets.graphics.MiscTechnicals
	(pygamelib.board_items.BoardComplexItem property), 287	attribute), 89 LEFT_CURLY_BRACKET_LOWER_HOOK
	(pygamelib.board_items.BoardItem property), 308	(pygame lib. assets. graphics. Misc Technicals
layer	(pygamelib.board_items.BoardItemComplexCompor	
,	property), 297	LEFT_CURLY_BRACKET_MIDDLE_PIECE
	(pygamelib.board_items.BoardItemVoid property), 317	(pygamelib.assets.graphics.MiscTechnicals attribute), 89
	(pygamelib.board_items.Camera property), 326	LEFT_CURLY_BRACKET_UPPER_HOOK
	(pygamelib.board_items.Character property), 336	(pygamelib.assets.graphics.MiscTechnicals
layer	(pygamelib.board_items.ComplexDoor property), 346	attribute), 89
lavor	(pygamelib.board_items.ComplexNPC property),	LEFT_DOWN_HEAVY_AND_RIGHT_UP_LIGHT (pygamelib.assets.graphics.BoxDrawings
layer	yygametio.boara_tiems.Complexivi C property), 358	attribute), 54
laver		LEFT_FACING_FIST (pygamelib.assets.graphics.Models
ruyer	erty), 368	attribute), 201
laver	(pygamelib.board_items.ComplexTreasure prop-	LEFT_FIVE_EIGHTHS_BLOCK
,	erty), 379	(pygamelib.assets.graphics.Blocks attribute),
layer	(pygamelib.board_items.ComplexWall property),	42
	390	LEFT_FLOOR (pygamelib.assets.graphics.MiscTechnicals
	(pygamelib.board_items.Door property), 401	attribute), 89
layer	$(pygamelib.board\_items. Generic Actionable Structure Generation and Generation $	
_	property), 410	(pygamelib.assets.graphics.GeometricShapes
layer	(pygamelib.board_items.GenericStructure prop-	attribute), 64
1	erty), 429	LEFT_HALF_BLOCK (pygamelib.assets.graphics.Blocks at-
ıayer	(pygamelib.board_items.GenericStructureComplexC	_
laver	property), 419 (pygamelib.board_items.Immovable property), 438	LEFT_HEAVY_AND_RIGHT_DOWN_LIGHT (pygamelib.assets.graphics.BoxDrawings
	(pygamelib.board_items.Movable property), 448	attribute), 54
	(pygamelib.board_items.NPC property), 459	LEFT_HEAVY_AND_RIGHT_UP_LIGHT

(pygamelib.assets.graphics.BoxDrawings attribute), 54	(pygamelib.assets.graphics.Blocks attribute), 42
LEFT_HEAVY_AND_RIGHT_VERTICAL_LIGHT	LEFT_UP_HEAVY_AND_RIGHT_DOWN_LIGHT
(pygamelib.assets.graphics.BoxDrawings attribute), 54	(pygamelib.assets.graphics.BoxDrawings attribute), 54
LEFT_LIGHT_AND_RIGHT_DOWN_HEAVY	LEFT_VERTICAL_BOX_LINE
(pygamelib.assets.graphics.BoxDrawings	(pygamelib.assets.graphics.MiscTechnicals
attribute), 54	attribute), 89
LEFT_LIGHT_AND_RIGHT_UP_HEAVY	LEG (pygamelib.assets.graphics.Models attribute), 201
(pygamelib.assets.graphics.BoxDrawings	LEMON (pygamelib.assets.graphics.Models attribute), 201
attribute), 54	length (pygamelib.base.Text property), 253
LEFT_LIGHT_AND_RIGHT_VERTICAL_HEAVY	length (pygamelib.gfx.core.Sprixel property), 608
(pygamelib.assets.graphics.BoxDrawings	length (pygamelib.gfx.particles.ParticleSprixel prop-
attribute), 54	erty), 728
LEFT_LUGGAGE (pygamelib.assets.graphics.Models	length() (pygamelib.base.Vector2D method), 258
attribute), 201	length() (pygamelib.gfx.ui.LineInput method), 658
LEFT_ONE_EIGHTH_BLOCK	length() (pygamelib.gfx.ui.MenuBar method), 671
(pygamelib.assets.graphics.Blocks attribute),	LEO (pygamelib.assets.graphics.Models attribute), 201
42	${\tt LEOPARD} \ \ (pygame lib. assets. graphics. Models \ \ attribute),$
LEFT_ONE_QUARTER_BLOCK	201
(pygamelib.assets.graphics.Blocks attribute),	lerp() (pygamelib.base.Math static method), 244
42	LEVEL_SLIDER (pygamelib.assets.graphics.Models
LEFT_PARENTHESIS_EXTENSION	attribute), 201
(pygamelib.assets.graphics.MiscTechnicals attribute), 89	LIBRA (pygamelib.assets.graphics.Models attribute), 201 LIGHT_ARC_DOWN_AND_LEFT
LEFT_PARENTHESIS_LOWER_HOOK	(pygamelib.assets.graphics.BoxDrawings
(pygamelib.assets.graphics.MiscTechnicals	attribute), 54
attribute), 89	LIGHT_ARC_DOWN_AND_RIGHT
LEFT_PARENTHESIS_UPPER_HOOK	(pygamelib.assets.graphics.BoxDrawings
(pygamelib.assets.graphics.MiscTechnicals	attribute), 54
attribute), 89 LEFT_POINTING_ANGLE_BRACKET	LIGHT_ARC_UP_AND_LEFT
(pygamelib.assets.graphics.MiscTechnicals	(pygamelib.assets.graphics.BoxDrawings attribute), 54
attribute), 89	LIGHT_ARC_UP_AND_RIGHT
LEFT_RIGHT_ARROW (pygamelib.assets.graphics.Models	(pygamelib.assets.graphics.BoxDrawings
attribute), 201	attribute), 54
LEFT_SEVEN_EIGHTHS_BLOCK	LIGHT_BULB (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.Blocks attribute),	tribute), 201
42	LIGHT_DIAGONAL_CROSS
LEFT_SPEECH_BUBBLE (pygamelib.assets.graphics.Models attribute), 201	(pygamelib.assets.graphics.BoxDrawings attribute), 54
LEFT_SQUARE_BRACKET_EXTENSION	LIGHT_DIAGONAL_UPPER_LEFT_TO_LOWER_RIGHT
(pygamelib.assets.graphics.MiscTechnicals	(pygamelib.assets.graphics.BoxDrawings
attribute), 89	attribute), 54
LEFT_SQUARE_BRACKET_LOWER_CORNER	LIGHT_DIAGONAL_UPPER_RIGHT_TO_LOWER_LEFT
(pygame lib. assets. graphics. Misc Technicals	(pygamelib.assets.graphics.BoxDrawings
attribute), 89	attribute), 54
LEFT_SQUARE_BRACKET_UPPER_CORNER	LIGHT_DOUBLE_DASH_HORIZONTAL
(pygamelib.assets.graphics.MiscTechnicals attribute), 89	(pygamelib.assets.graphics.BoxDrawings attribute), 54
LEFT_THREE_EIGHTHS_BLOCK	LIGHT_DOUBLE_DASH_VERTICAL
(pygamelib.assets.graphics.Blocks attribute), 42	(pygamelib.assets.graphics.BoxDrawings attribute), 54
LEFT_THREE_QUARTERS_BLOCK	LIGHT_DOWN (pygamelib.assets.graphics.BoxDrawings

attribute), 54	LIGHT_VERTICAL_AND_LEFT
LIGHT_DOWN_AND_HORIZONTAL	(pygamelib.assets.graphics.BoxDrawings
(pygamelib.assets.graphics.BoxDrawings	attribute), 55
attribute), 54	LIGHT_VERTICAL_AND_RIGHT
LIGHT_DOWN_AND_LEFT	(pygamelib.assets.graphics.BoxDrawings
(pygamelib.assets.graphics.BoxDrawings	attribute), 55
attribute), 54	LineInput (class in pygamelib.gfx.ui), 653
LIGHT_DOWN_AND_RIGHT	LineInputDialog (class in pygamelib.gfx.ui), 661
(pygamelib.assets.graphics.BoxDrawings	LINK (pygamelib.assets.graphics.Models attribute), 201
attribute), 54	LINKED_PAPERCLIPS (pygamelib.assets.graphics.Models
LIGHT_HORIZONTAL (pygamelib.assets.graphics.BoxDrawin	
attribute), 54	LION (pygamelib.assets.graphics.Models attribute), 201
	LIPSTICK (pygamelib.assets.graphics.Models attribute),
attribute), 54	201
LIGHT_LEFT_AND_HEAVY_RIGHT	LITTER_IN_BIN_SIGN (pygamelib.assets.graphics.Models
(pygamelib.assets.graphics.BoxDrawings	attribute), 201
attribute), 54	LIZARD (pygamelib.assets.graphics.Models attribute),
LIGHT_QUADRUPLE_DASH_HORIZONTAL	201
(pygamelib.assets.graphics.BoxDrawings	LLAMA (pygamelib.assets.graphics.Models attribute), 201
attribute), 54	load() (pygamelib.actuators.Actuator method), 9
LIGHT_QUADRUPLE_DASH_VERTICAL	load() (pygamelib.actuators.Behavioral method), 12
(pygamelib.assets.graphics.BoxDrawings attribute), 55	load() (pygamelib.actuators.PathActuator class method), 16
LIGHT_RAIL (pygamelib.assets.graphics.Models at-	load() (pygamelib.actuators.PathFinder class method),
tribute), 201	27
LIGHT_RIGHT (pygamelib.assets.graphics.BoxDrawings	load() (pygamelib.actuators.PatrolActuator class
attribute), 55	method), 20
LIGHT_SHADE (pygamelib.assets.graphics.Blocks attribute), 42	load() (pygamelib.actuators.RandomActuator class method), 32
LIGHT_SKIN_TONE (pygamelib.assets.graphics.Models attribute), 201	load() (pygamelib.actuators.UnidirectionalActuator class method), 36
LIGHT_TRIPLE_DASH_HORIZONTAL	load() (pygamelib.base.Text class method), 254
(pygamelib.assets.graphics.BoxDrawings	load() (pygamelib.base.Vector2D class method), 259
attribute), 55	load() (pygamelib.board_items.Actionable class
LIGHT_TRIPLE_DASH_VERTICAL	method), 267
(pygamelib.assets.graphics.BoxDrawings	load() (pygamelib.board_items.ActionableTile class
attribute), 55	method), 277
LIGHT_UP (pygamelib.assets.graphics.BoxDrawings at-	load() (pygamelib.board_items.BoardComplexItem
<i>tribute</i> ), 55	class method), 288
LIGHT_UP_AND_HEAVY_DOWN	load() (pygamelib.board_items.BoardItem class
(pygamelib.assets.graphics.BoxDrawings	method), 308
attribute), 55	load() (pygamelib.board_items.BoardItemComplexComponen
LIGHT_UP_AND_HORIZONTAL	class method), 297
(pygamelib.assets.graphics.BoxDrawings	load() (pygamelib.board_items.BoardItemVoid class
attribute), 55	method), 317
LIGHT_UP_AND_LEFT (pygamelib.assets.graphics.BoxDraw attribute), 55	
LIGHT_UP_AND_RIGHT (pygamelib.assets.graphics.BoxDra	
attribute), 55	method), 337
LIGHT_VERTICAL (pygamelib.assets.graphics.BoxDrawings	
attribute), 55	method), 347
LIGHT_VERTICAL_AND_HORIZONTAL	load() (pygamelib.board_items.ComplexNPC class
(pygamelib.assets.graphics.BoxDrawings	method), 358
attribute). 55	load() (nyoamelih hoard items ComplexPlayer class

method), 369	class method), 741
<pre>load() (pygamelib.board_items.ComplexTreasure class</pre>	
load() (pygamelib.board_items.ComplexWall class	load_board() (pygamelib.engine.Game method), 553
method), 391	load_config() (pygamelib.engine.Game method), 554
load() (pygamelib.board_items.Door class method), 401	<pre>load_from_ansi_file() (pygamelib.gfx.core.Sprite</pre>
<pre>load() (pygamelib.board_items.GenericActionableStruct</pre>	ure class method), 598
class method), 411	<pre>load_json_file() (pygamelib.gfx.core.SpriteCollection</pre>
<pre>load() (pygamelib.board_items.GenericStructure class</pre>	static method), 591
method), 429	LOBSTER (pygamelib.assets.graphics.Models attribute),
load() (pygamelib.board_items.GenericStructureComple	
class method), 419	lock_position() (pygamelib.gfx.ui.Cursor method),
load() (pygamelib.board_items.Immovable class	630
method), 438	LOCKED (pygamelib.assets.graphics.Models attribute),
<pre>load() (pygamelib.board_items.Movable class method),</pre>	201
449	LOCKED_WITH_KEY (pygamelib.assets.graphics.Models
load() (pygamelib.board_items.NPC class method), 459	attribute), 201
load() (pygamelib.board_items.Player class method),	LOCKED_WITH_PEN (pygamelib.assets.graphics.Models
	2.7.0
469	attribute), 201
load() (pygamelib.board_items.Projectile class	LOCOMOTIVE (pygamelib.assets.graphics.Models at-
method), 481	tribute), 202
<pre>load() (pygamelib.board_items.TextItem class method),</pre>	LOLLIPOP (pygamelib.assets.graphics.Models attribute),
492	202
load() (pygamelib.board_items.Tile class method), 503	LONG_DRUM (pygamelib.assets.graphics.Models at-
	tribute), 202
load() (pygamelib.board_items.Treasure class method),	
513	LOTION_BOTTLE (pygamelib.assets.graphics.Models at-
load() (pygamelib.board_items.Wall class method), 523	tribute), 202
load() (pygamelib.engine.Board class method), 539	LOUDLY_CRYING_FACE (pygamelib.assets.graphics.Models
load() (pygamelib.engine.Inventory class method), 565	attribute), 202
load() (pygamelib.gfx.core.Animation class method),	LOUDSPEAKER (pygamelib.assets.graphics.Models at-
582	tribute), 202
load() (pygamelib.gfx.core.Color class method), 615	LOVE_HOTEL (pygamelib.assets.graphics.Models at-
load() (pygamelib.gfx.core.Font method), 587	tribute), 202
load() (pygamelib.gfx.core.Sprite class method), 598	LOVE_LETTER (pygamelib.assets.graphics.Models at-
<pre>load() (pygamelib.gfx.core.SpriteCollection class</pre>	tribute), 202
method), 591	LOVE_YOU_GESTURE (pygamelib.assets.graphics.Models
load() (pygamelib.gfx.core.Sprixel class method), 608	attribute), 202
load() (pygamelib.gfx.particles.CircleEmitter class	LOWER_FIVE_EIGHTHS_BLOCK
method), 692	(pygamelib.assets.graphics.Blocks attribute),
load() (pygamelib.gfx.particles.ColorParticle class	42
method), 698	LOWER_HALF_BLOCK (pygamelib.assets.graphics.Blocks
${\tt load()}  (pygame lib. gfx. particles. Color Partition Particle$	attribute), 42
class method), 704	${\tt LOWER\_HALF\_CIRCLE}\ (pygamelib. assets. graphics. Geometric Shapes)$
<pre>load() (pygamelib.gfx.particles.EmitterProperties class</pre>	attribute), 64
method), 708	LOWER_HALF_INVERSE_WHITE_CIRCLE
<pre>load() (pygamelib.gfx.particles.Particle class method),</pre>	(pygamelib.assets.graphics.GeometricShapes
720	attribute), 64
load() (pygamelib.gfx.particles.ParticleEmitter class	LOWER_LEFT_QUADRANT_CIRCULAR_ARC
method), 712	(pygame lib. assets. graphics. Geometric Shapes
load() (pygamelib.gfx.particles.ParticleSprixel class	attribute), 64
method), 728	LOWER_LEFT_TRIANGLE
<pre>load() (pygamelib.gfx.particles.PartitionParticle class</pre>	(pygamelib.assets.graphics.GeometricShapes
method), 736	attribute), 65
load() (pygamelib.gfx.particles.RandomColorParticle	
(pygameno, gyaparneno, manaomeono il article	

(pygamelib.assets.graphics.Models attribute),

(pygamelib.assets.graphics.Blocks attribute). MAHJONG\_RED\_DRAGON (pygamelib.assets.graphics.Models 42 attribute), 202 LOWER\_ONE\_QUARTER\_BLOCK MALE\_SIGN (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Blocks attribute), tribute), 202 MAMMOTH (pygamelib.assets.graphics.Models attribute), LOWER\_RIGHT\_QUADRANT\_CIRCULAR\_ARC (pygamelib.assets.graphics.GeometricShapes MAN (pygamelib.assets.graphics.Models attribute), 202 attribute), 65 MAN\_BEARD (pygamelib.assets.graphics.Models attribute), 202 LOWER\_RIGHT\_TRIANGLE (pygamelib.assets.graphics.GeometricShapes MAN\_DANCING (pygamelib.assets.graphics.Models atattribute), 65 tribute), 202 LOWER\_SEVEN\_EIGHTHS\_BLOCK MANGO (pygamelib.assets.graphics.Models attribute), 202 (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Blocks attribute), MANS\_SHOE tribute), 202 42 LOWER\_THREE\_EIGHTHS\_BLOCK MANTELPIECE\_CLOCK (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Blocks attribute), attribute), 202 43 MANUAL\_WHEELCHAIR (pygamelib.assets.graphics.Models LOWER\_THREE\_QUARTERS\_BLOCK attribute), 202 (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Blocks MAP\_OF\_JAPAN attribute), attribute), 202 LOZENGE (pygamelib.assets.graphics.GeometricShapes MAPLE\_LEAF (pygamelib.assets.graphics.Models attribute), 65 tribute), 202 MARTIAL\_ARTS\_UNIFORM LUGGAGE (pygamelib.assets.graphics.Models attribute), (pygamelib.assets.graphics.Models attribute), LUNGS (pygamelib.assets.graphics.Models attribute), 202 202 LYING\_FACE (pygamelib.assets.graphics.Models MATE (pygamelib.assets.graphics.Models attribute), 202 tribute), 202 Math (class in pygamelib.base), 243 max\_height (pygamelib.gfx.ui.GridSelector property), M 649 max\_width (pygamelib.gfx.ui.GridSelector property), MAGE (pygamelib.assets.graphics.Models attribute), 202 649 magenta() (pygamelib.base.Text static method), 254 maximum (pygamelib.gfx.ui.ProgressBar property), 679 magenta\_bright() (pygamelib.base.Text static maximum (pygamelib.gfx.ui.ProgressDialog property), method), 254 magenta\_dim() (pygamelib.base.Text static method), maximum\_height (pygamelib.gfx.ui.LineInput property), 254 658 magenta\_rect() (pygamelib.gfx.core.Sprixel class maximum\_height (pygamelib.gfx.ui.Widget property), *method*), 608 magenta\_rect() (pygamelib.gfx.particles.ParticleSprixel MAXIMUM\_SIZE (pygamelib.constants.SizeConstraint atclass method), 728 *tribute*), 530 magenta\_square() (pygamelib.gfx.core.Sprixel class maximum\_width (pygamelib.gfx.ui.LineInput property), method), 609 magenta\_square() (pygamelib.gfx.particles.ParticleSprixel maximum\_width (pygamelib.gfx.ui.Widget property), 687 class method), 729 MEAT\_ON\_BONE (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Models MAGIC\_WAND attribute), 203 tribute), 202 MECHANICAL\_ARM (pygamelib.assets.graphics.Models at-(pygamelib.assets.graphics.Models attribute), MAGNET tribute), 203 202 MECHANICAL\_LEG (pygamelib.assets.graphics.Models at-MAGNIFYING\_GLASS\_TILTED\_LEFT tribute), 203 (pygamelib.assets.graphics.Models attribute), MEDICAL\_SYMBOL (pygamelib.assets.graphics.Models attribute), 203 MAGNIFYING\_GLASS\_TILTED\_RIGHT MEDIUM\_DARK\_SKIN\_TONE

Index 797

203

(pygamelib.assets.graphics.Models attribute),

202

MEDIUM_LIGHT_SKIN_TONE	MIDDLE_FINGER (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.Models attribute),	tribute), 203
203	MILITARY_HELMET (pygamelib.assets.graphics.Models
MEDIUM_SHADE (pygamelib.assets.graphics.Blocks	attribute), 203
attribute), 43	MILITARY_MEDAL (pygamelib.assets.graphics.Models at-
MEDIUM_SKIN_TONE (pygamelib.assets.graphics.Models	tribute), 203
attribute), 203	MILKY_WAY (pygamelib.assets.graphics.Models at-
MEGAPHONE (pygamelib.assets.graphics.Models at-	tribute), 203
tribute), 203	MINIBUS (pygamelib.assets.graphics.Models attribute),
MELON (pygamelib.assets.graphics.Models attribute), 203	203
MEMO (pygamelib.assets.graphics.Models attribute), 203	<pre>minimum_height (pygamelib.gfx.ui.LineInput property),</pre>
MEN_HOLDING_HANDS (pygamelib.assets.graphics.Models	659
attribute), 203	<pre>minimum_height (pygamelib.gfx.ui.Widget property),</pre>
MENORAH (pygamelib.assets.graphics.Models attribute),	687
203	MINIMUM_SIZE (pygamelib.constants.SizeConstraint at-
MENS_ROOM (pygamelib.assets.graphics.Models at-	tribute), 530
tribute), 203	minimum_width (pygamelib.gfx.ui.LineInput property),
Menu ( <i>class in pygamelib.gfx.ui</i> ), 663	659
menu_width() (pygamelib.gfx.ui.Menu method), 665	minimum_width (pygamelib.gfx.ui.Widget property), 687
MenuAction (class in pygamelib.gfx.ui), 667	MINUS (pygamelib.assets.graphics.Models attribute), 203
MenuBar (class in pygamelib.gfx.ui), 669	MIRROR (pygamelib.assets.graphics.Models attribute),
MERPERSON (pygamelib.assets.graphics.Models at-	203
tribute), 203	MiscTechnicals (class in pygamelib.assets.graphics),
MessageDialog (class in pygamelib.gfx.ui), 672	67
METRICAL_BREVE (pygamelib.assets.graphics.MiscTechnical	
attribute), 89	MOBILE_PHONE (pygamelib.assets.graphics.Models
METRICAL_LONG_OVER_SHORT	attribute), 203
(pygamelib.assets.graphics.MiscTechnicals	MOBILE_PHONE_OFF (pygamelib.assets.graphics.Models
attribute), 89	attribute), 203
METRICAL_LONG_OVER_TWO_SHORTS	MOBILE_PHONE_WITH_ARROW
(pygamelib.assets.graphics.MiscTechnicals	(pygamelib.assets.graphics.Models attribute),
attribute), 89	(pygametio.assets.graphics.modets airribute), 203
METRICAL_PENTASEME (pygamelib.assets.graphics.MiscTec	
attribute), 89	tribute), 529
METRICAL_SHORT_OVER_LONG	
(pygamelib.assets.graphics.MiscTechnicals	MODE_TURN_BY_TURN (pygamelib.constants.EngineMode attribute), 529
attribute), 89	model (pygamelib.board_items.Actionable property), 267
METRICAL_TETRASEME (pygamelib.assets.graphics.MiscTed	
attribute), 89	277
METRICAL_TRISEME (pygamelib.assets.graphics.MiscTechr	
attribute), 89	erty), 288
METRICAL_TWO_SHORTS_JOINED	model (pygamelib.board_items.BoardItem property), 309
(pygamelib.assets.graphics.MiscTechnicals	model (pygamelib.board_items.BoardItemComplexComponent
attribute), 90	property), 298
METRICAL_TWO_SHORTS_OVER_LONG	model (pygamelib.board_items.BoardItemVoid property),
(pygamelib.assets.graphics.MiscTechnicals	318
attribute), 90	model (pygamelib.board_items.Camera property), 327
METRO (pygamelib.assets.graphics.Models attribute), 203	model (pygamelib.board_items.Character property), 337
MICROBE (pygamelib.assets.graphics.Models attribute),	model (pygamelib.board_items.ComplexDoor property),
203	347
MICROPHONE (pygamelib.assets.graphics.Models at-	model (pygamelib.board_items.ComplexNPC property),
tribute), 203	358
MICROSCOPE (pygamelib.assets.graphics.Models at-	model (pygamelib.board_items.ComplexPlayer prop-
tribute), 203	erty), 369

<pre>model (pygamelib.board_items.ComplexTreasure prop-</pre>	tribute), 204
erty), 380	MOTORCYCLE (pygamelib.assets.graphics.Models at-
<pre>model (pygamelib.board_items.ComplexWall property),</pre>	tribute), 204
391	MOTORIZED_WHEELCHAIR
model (pygamelib.board_items.Door property), 402	(pygamelib.assets.graphics.Models attribute),
model (pygamelib.board_items.GenericActionableStructure	
property), 411	MOTORWAY (pygamelib.assets.graphics.Models attribute),
model (pygamelib.board_items.GenericStructure prop-	204
erty), 430	MOUNT_FUJI (pygamelib.assets.graphics.Models at-
model (pygamelib.board_items.GenericStructureComplexC	1.0
property), 420	MOUNTAIN (pygamelib.assets.graphics.Models attribute),
model (pygamelib.board_items.Immovable property), 439	204
model (pygametib.board_items.Movable property), 449  model (pygamelib.board_items.Movable property), 449	MOUNTAIN_CABLEWAY (pygamelib.assets.graphics.Models
	attribute), 204
model (pygamelib.board_items.NPC property), 459	
model (pygamelib.board_items.Player property), 469	MOUNTAIN_RAILWAY (pygamelib.assets.graphics.Models
model (pygamelib.board_items.Projectile property), 481	attribute), 204
model (pygamelib.board_items.TextItem property), 492	MOUSE (pygamelib.assets.graphics.Models attribute), 204
model (pygamelib.board_items.Tile property), 503	MOUSE_FACE (pygamelib.assets.graphics.Models at-
model (pygamelib.board_items.Treasure property), 514	tribute), 204
model (pygamelib.board_items.Wall property), 524	MOUSE_TRAP (pygamelib.assets.graphics.Models at-
model (pygamelib.gfx.core.Sprixel property), 609	tribute), 204
model (pygamelib.gfx.particles.ParticleSprixel property),	MOUTH (pygamelib.assets.graphics.Models attribute), 204
729	Movable (class in pygamelib.board_items), 443
Models (class in pygamelib.assets.graphics), 92	move() (pygamelib.engine.Board method), 540
<pre>modulate() (pygamelib.gfx.core.Sprite method), 598</pre>	<pre>move_cursor() (pygamelib.gfx.ui.LineInput method),</pre>
module	659
pygamelib.constants,532	<pre>move_player() (pygamelib.engine.Game method), 554</pre>
MONEY_BAG (pygamelib.assets.graphics.Models at-	moveset (pygamelib.actuators.RandomActuator prop-
tribute), 203	erty), 32
MONEY_MOUTH_FACE (pygamelib.assets.graphics.Models	MOVIE_CAMERA (pygamelib.assets.graphics.Models
attribute), 203	attribute), 204
MONEY_WITH_WINGS (pygamelib.assets.graphics.Models	MRS_CLAUS (pygamelib.assets.graphics.Models at-
attribute), 204	<i>tribute</i> ), 204
MONKEY (pygamelib.assets.graphics.Models attribute),	MultiLineInputDialog (class in pygamelib.gfx.ui),
204	675
MONKEY_FACE (pygamelib.assets.graphics.Models at-	MULTIPLY (pygamelib.assets.graphics.Models attribute),
tribute), 204	204
MONORAIL (pygamelib.assets.graphics.Models attribute),	
204	204
monospace (pygamelib.gfx.core.Font property), 588	MUSICAL_KEYBOARD (pygamelib.assets.graphics.Models
MONOSTABLE_SYMBOL (pygamelib.assets.graphics.MiscTech	
attribute), 90	MUSICAL_NOTE (pygamelib.assets.graphics.Models
MOON_CAKE (pygamelib.assets.graphics.Models at-	attribute), 204
tribute), 204	MUSICAL_NOTES (pygamelib.assets.graphics.Models at-
MOON_VIEWING_CEREMONY	tribute), 204
(pygamelib.assets.graphics.Models attribute), 204	MUSICAL_SCORE (pygamelib.assets.graphics.Models attribute), 204
MOSQUE (pygamelib.assets.graphics.Models attribute), 204	MUTED_SPEAKER (pygamelib.assets.graphics.Models attribute), 204
MOSQUITO (pygamelib.assets.graphics.Models attribute),	<i>"</i>
204	N
MOTOR_BOAT (pygamelib.assets.graphics.Models at-	NAIL_POLISH (pygamelib.assets.graphics.Models at-
<i>tribute</i> ), 204	tribute), 204
MOTOR_SCOOTER (pygamelib.assets.graphics.Models at-	

- NAME\_BADGE (pygamelib.assets.graphics.Models tribute), 205
- NATIONAL\_PARK (pygamelib.assets.graphics.Models attribute), 205
- NAUSEATED\_FACE (pygamelib.assets.graphics.Models attribute), 205
- NAZAR AMULET (pygamelib.assets.graphics.Models attribute), 205
- nb\_pages() (pygamelib.gfx.ui.GridSelector method),
- NECKTIE (pygamelib.assets.graphics.Models attribute),
- need\_rendering (pygamelib.engine.Screen property), 577
- neighbors() (pygamelib.engine.Board method), 540
- neighbors() (pygamelib.engine.Game method), 554
- NERD\_FACE (pygamelib.assets.graphics.Models attribute), 205
- NESTING\_DOLLS (pygamelib.assets.graphics.Models attribute), 205
- NEUTRAL\_FACE (pygamelib.assets.graphics.Models attribute), 205
- (pygamelib.assets.graphics.Models NEW\_BUTTON tribute), 205
- NEW\_MOON (pygamelib.assets.graphics.Models attribute),
- NEW\_MOON\_FACE (pygamelib.assets.graphics.Models attribute), 205
- NEWSPAPER (pygamelib.assets.graphics.Models attribute), 205
- next\_action() (pygamelib.actuators.Behavioral method), 12
- next\_action() (pygamelib.actuators.PathFinder method), 27
- next\_frame() (pygamelib.gfx.core.Animation method),
- next\_move() (pygamelib.actuators.Actuator method), 9
- next\_move() (pygamelib.actuators.Behavioral method), 13
- (pygamelib.actuators.PathActuator next\_move() method), 16
- next\_move() (pygamelib.actuators.PathFinder method),
- next\_move() (pygamelib.actuators.PatrolActuator method), 20
- (pygamelib.actuators.RandomActuator next\_move() method), 33
- next\_move() (pygamelib.actuators.UnidirectionalActuatornotify() method), 36
- NEXT\_PAGE attribute), 90
- NEXT\_TRACK\_BUTTON (pygamelib.assets.graphics.Models notify() (pygamelib.base.PglBaseObject method), 247 attribute), 205
- next\_waypoint()

- method), 28
- NG\_BUTTON (pygamelib.assets.graphics.Models attribute), 205
- NIGHT\_WITH\_STARS (pygamelib.assets.graphics.Models attribute), 205
- NINE\_OCLOCK (pygamelib.assets.graphics.Models attribute), 205
- NINE\_THIRTY (pygamelib.assets.graphics.Models attribute), 205
- NINJA (pygamelib.assets.graphics.Models attribute), 205 NO\_BICYCLES (pygamelib.assets.graphics.Models at-
- NO\_DIR (pygamelib.constants.Direction attribute), 528
- NO\_ENTRY (pygamelib.assets.graphics.Models attribute), 205
- NO\_LITTERING (pygamelib.assets.graphics.Models attribute), 205
- NO\_MOBILE\_PHONES (pygamelib.assets.graphics.Models attribute), 205
- NO\_ONE\_UNDER\_EIGHTEEN

tribute), 205

- (pygamelib.assets.graphics.Models attribute),
- NO\_PEDESTRIANS (pygamelib.assets.graphics.Models attribute), 205
- NO\_PLAYER (pygamelib.constants.EngineConstant attribute), 529
- NO\_SMOKING (pygamelib.assets.graphics.Models tribute), 205
- NON\_POTABLE\_WATER (pygamelib.assets.graphics.Models attribute), 205
- NONE\_AUTHORIZED (pygamelib.constants.Permission attribute), 530
- NOSE (pygamelib.assets.graphics.Models attribute), 205
- ${\tt NOT\_CHECK\_MARK}\ (pygamelib. assets. graphics. Misc Technicals$ attribute), 90
- NOTEBOOK (pygamelib.assets.graphics.Models attribute), 205
- NOTEBOOK\_WITH\_DECORATIVE\_COVER
  - (pygamelib.assets.graphics.Models attribute), 205
- notify() (pygamelib.actuators.Actuator method), 9
- notify() (pygamelib.actuators.Behavioral method), 13
- notify() (pygamelib.actuators.PathActuator method),
- notify() (pygamelib.actuators.PathFinder method), 28
- notify() (pygamelib.actuators.PatrolActuator method), 21
- (pygamelib.actuators.RandomActuator method), 33
- (pygamelib.assets.graphics.MiscTechnicals notify() (pygamelib.actuators.UnidirectionalActuator method), 37

  - notify() (pygamelib.base.Text method), 254
  - (pygamelib.actuators.PathFinder notify() (pygamelib.board items.Actionable method),

267	notify() (pygamelib.gfx.particles.ColorParticle
$\verb"notify"() \qquad \textit{(pygamelib.board\_items.ActionableTile}$	method), 698
method), 277	notify() (pygamelib.gfx.particles.ColorPartitionParticle
notify() (pygamelib.board_items.BoardComplexItem method), 288	method), 704 notify() (pygamelib.gfx.particles.Particle method), 720
notify() (pygamelib.board_items.BoardItem method),	notify() (pygamelib.gfx.particles.ParticleEmitter
309	method), 712
$\verb"notify()" (pygame lib.board\_items. Board Item Complex Comp$	n <b>port</b> anty() (pygamelib.gfx.particles.ParticleSprixel
method), 298	method), 729
<pre>notify() (pygamelib.board_items.BoardItemVoid</pre>	notify() (pygamelib.gfx.particles.PartitionParticle method), 736
<pre>notify() (pygamelib.board_items.Camera method), 327</pre>	$\verb"notify" () \textit{ (pygame lib. gfx. particles. Random Color Particle}$
<pre>notify() (pygamelib.board_items.Character method),</pre>	method), 741
337	$\verb"notify" () (\textit{pygamelib.gfx.particles.RandomColorPartitionParticle}$
notify() (pygamelib.board_items.ComplexDoor	method), 747
method), 347	notify() (pygamelib.gfx.ui.BoxLayout method), 622
notify() (pygamelib.board_items.ComplexNPC	notify() (pygamelib.gfx.ui.Cursor method), 630
method), 358  not if the Complex Player of the Complex Player	notify() (pygamelib.gfx.ui.FormLayout method), 638
notify() (pygamelib.board_items.ComplexPlayer method), 369	notify() (pygamelib.gfx.ui.GridLayout method), 643 notify() (pygamelib.gfx.ui.Layout method), 651
notify() (pygamelib.board_items.ComplexTreasure	notify() (pygamelib.gfx.ui.LineInput method), 659
method), 380	notify() (pygamelib.gfx.ui.Widget method), 687
notify() (pygamelib.board_items.ComplexWall	NPC (class in pygamelib.board_items), 453
method), 391	NPC_AUTHORIZED (pygamelib.constants.Permission at-
<pre>notify() (pygamelib.board_items.Door method), 402</pre>	tribute), 530
$\verb"notify" () \textit{ (pygame lib. board\_items. Generic Actionable Structure)} \\$	cMUT_AND_BOLT (pygamelib.assets.graphics.Models
method), 411	attribute), 205
notify() (pygamelib.board_items.GenericStructure method), 430	0
notify() (pygamelib.board_items.GenericStructureComp	ON COMPRESSION COD TYPE
method), 420	
notify() (pygamelib.board_items.Immovable method),	(pygamelib.assets.graphics.Models attribute), 207
439	OBSERVER_EYE_SYMBOL
<pre>notify() (pygamelib.board_items.Movable method),</pre>	(pygamelib.assets.graphics.MiscTechnicals
449	attribute), 90
<pre>notify() (pygamelib.board_items.NPC method), 459</pre>	OCTOPUS (pygamelib.assets.graphics.Models attribute),
notify() (pygamelib.board_items.Player method), 469	205
notify() (pygamelib.board_items.Projectile method),	ODEN (pygamelib.assets.graphics.Models attribute), 206
481	OFFICE_BUILDING (pygamelib.assets.graphics.Models
notify() (pygamelib.board_items.TextItem method),	attribute), 206
492	OGRE (pygamelib.assets.graphics.Models attribute), 206
<pre>notify() (pygamelib.board_items.Tile method), 503 notify() (pygamelib.board_items.Treasure method),</pre>	OIL_DRUM (pygamelib.assets.graphics.Models attribute),
514	206 OK_BUTTON (pygamelib.assets.graphics.Models at-
notify() (pygamelib.board_items.Wall method), 524	OK_BUTTON (pygamelib.assets.graphics.Models at- tribute), 206
notify() (pygamelib.engine.Board method), 541	OK_HAND (pygamelib.assets.graphics.Models attribute),
notify() (pygamelib.engine.Game method), 555	206
notify() (pygamelib.engine.Inventory method), 565	OLD_KEY (pygamelib.assets.graphics.Models attribute),
notify() (pygamelib.engine.Screen method), 577	206
notify() (pygamelib.gfx.core.Color method), 615	OLD_MAN (pygamelib.assets.graphics.Models attribute),
<pre>notify() (pygamelib.gfx.core.Sprite method), 599</pre>	206
notify() (pygamelib.gfx.core.Sprixel method), 609	
notify() (pygamelib.gfx.particles.CircleEmitter	OLD_WOMAN (pygamelib.assets.graphics.Models at- tribute), 206

method), 692

ORANGE\_HEART

OLDER\_PERSON (pygamelib.assets.graphics.Models attribute), 207 attribute), 206 ORANGE\_SQUARE (pygamelib.assets.graphics.Models at-OLIVE (pygamelib.assets.graphics.Models attribute), 206 tribute), 207 OM (pygamelib.assets.graphics.Models attribute), 206 ORANGUTAN (pygamelib.assets.graphics.Models at-ON\_ARROW (pygamelib.assets.graphics.Models attribute), tribute), 207 206 orientation (pygamelib.gfx.ui.BoxLayout property), ONCOMING AUTOMOBILE (pygamelib.assets.graphics.Models attribute), ORTHODOX\_CROSS (pygamelib.assets.graphics.Models at-206 tribute), 207 ONCOMING\_BUS OTTER (pygamelib.assets.graphics.Models attribute), 207 (pygamelib.assets.graphics.Models attribute), 206 OUTBOX\_TRAY (pygamelib.assets.graphics.Models at-ONCOMING\_FIST (pygamelib.assets.graphics.Models attribute), 207 (pygamelib.board\_items.Actionable tribute), 206 overlappable() ONCOMING\_POLICE\_CAR method), 267 (pygamelib.assets.graphics.Models attribute), overlappable() (pygamelib.board\_items.ActionableTile method), 277 ONCOMING\_TAXI (pygamelib.assets.graphics.Models atoverlappable() (pygamelib.board\_items.BoardComplexItem tribute), 206 method), 288 (pygamelib.assets.graphics.Models (pygamelib.board\_items.BoardItem ONE\_OCLOCK overlappable() tribute), 206 method), 309 ONE\_PIECE\_SWIMSUIT (pygamelib.assets.graphics.Models overlappable() (pygamelib.board\_items.BoardItemComplexComponent attribute), 206 method), 298 ONE\_THIRTY (pygamelib.assets.graphics.Models overlappable() (pygamelib.board\_items.BoardItemVoid tribute), 206 method), 318 ONION (pygamelib.assets.graphics.Models attribute), 206 overlappable() (pygamelib.board\_items.Camera (pygamelib.assets.graphics.Models OPEN\_BOOK atmethod), 327 tribute), 206 overlappable() (pygamelib.board\_items.Character OPEN\_CIRCUIT\_OUTPUT\_H\_TYPE\_SYMBOL method), 337 overlappable() (pygamelib.board\_items.ComplexDoor (pygamelib.assets.graphics.MiscTechnicals attribute), 90 method), 347 OPEN\_CIRCUIT\_OUTPUT\_L\_TYPE\_SYMBOL overlappable() (pygamelib.board\_items.ComplexNPC (pygamelib.assets.graphics.MiscTechnicals method), 359 attribute), 90 overlappable() (pygamelib.board\_items.ComplexPlayer OPEN\_FILE\_FOLDER (pygamelib.assets.graphics.Models method), 369 attribute), 206 overlappable() (pygamelib.board\_items.ComplexTreasure (pygamelib.assets.graphics.Models atmethod), 380 OPEN\_HANDS tribute), 206 overlappable() (pygamelib.board items.ComplexWall OPEN\_MAILBOX\_WITH\_LOWERED\_FLAG method), 391 (pygamelib.assets.graphics.Models attribute), overlappable() (pygamelib.board\_items.Door 206 method), 402 OPEN\_MAILBOX\_WITH\_RAISED\_FLAG overlappable() (pygamelib.board items.GenericActionableStructure (pygamelib.assets.graphics.Models attribute), method), 411  ${\tt overlappable()}\ (pygame lib. board\_items. Generic Structure$ **OPHIUCHUS** (pygamelib.assets.graphics.Models method), 430 atoverlappable() (pygamelib.board\_items.GenericStructureComplexComp tribute), 206 OPTICAL\_DISK (pygamelib.assets.graphics.Models method), 420 attribute), 206 overlappable() (pygamelib.board\_items.Immovable OPTION\_KEY (pygamelib.assets.graphics.MiscTechnicals method), 439 (pygamelib.board\_items.Movable attribute), 90 overlappable() ORANGE\_BOOK (pygamelib.assets.graphics.Models method), 449 tribute), 206 overlappable() (pygamelib.board\_items.NPC ORANGE\_CIRCLE (pygamelib.assets.graphics.Models atmethod), 459 tribute), 206 overlappable() (pygamelib.board items.Player

802 Index

method), 469

(pygamelib.assets.graphics.Models

overlappable() (pygamelib.board items.Projectile PART\_ALTERNATION\_MARK method), 481 (pygamelib.assets.graphics.Models attribute), overlappable() (pygamelib.board\_items.TextItem 207 method), 492 Particle (class in pygamelib.gfx.particles), 717 overlappable() (pygamelib.board\_items.Tile method), particle\_emitter (pygamelib.board\_items.Actionable property), 267 overlappable() (pygamelib.board items.Treasure particle\_emitter(pygamelib.board items.ActionableTile method), 514 property), 277 overlappable() (pygamelib.board\_items.Wall method),  $\verb"particle_emitter" (pygame lib.board\_items. Board Complex Item")$ property), 288 particle\_emitter (pygamelib.board\_items.BoardItem OWL (pygamelib.assets.graphics.Models attribute), 207 OX (pygamelib.assets.graphics.Models attribute), 207 property), 309  $\verb|particle_emitter|| (pygame lib. board\_items. Board Item Complex Components)| (pygame lib. board\_items. Board Items. Board It$ OYSTER (pygamelib.assets.graphics.Models attribute), 207 property), 298 particle\_emitter(pygamelib.board\_items.BoardItemVoid Р property), 318 particle\_emitter (pygamelib.board\_items.Camera P\_BUTTON (pygamelib.assets.graphics.Models attribute), property), 327 particle\_emitter (pygamelib.board\_items.Character PACKAGE (pygamelib.assets.graphics.Models attribute), property), 337 padding (pygamelib.gfx.ui.Menu property), 665 particle\_emitter(pygamelib.board\_items.ComplexDoor property), 347 padding (pygamelib.gfx.ui.MenuAction property), 668 particle\_emitter(pygamelib.board\_items.ComplexNPC page\_down() (pygamelib.gfx.ui.GridSelector method), property), 359 PAGE\_FACING\_UP (pygamelib.assets.graphics.Models atparticle\_emitter(pygamelib.board\_items.ComplexPlayer tribute), 207 property), 369  $\verb|particle_emitter| (pygame lib.board\_items. Complex Treasure$ page\_up() (pygamelib.gfx.ui.GridSelector method), 649 property), 380 PAGE\_WITH\_CURL (pygamelib.assets.graphics.Models atparticle\_emitter(pygamelib.board\_items.ComplexWall tribute), 207 property), 391 PAGER (pygamelib.assets.graphics.Models attribute), 207 PAINTBRUSH (pygamelib.assets.graphics.Models atparticle\_emitter (pygamelib.board\_items.Door proptribute), 207 erty), 402 (pygamelib.assets.graphics.Models particle\_emitter(pygamelib.board\_items.GenericActionableStructure PALM\_TREE atproperty), 411 tribute), 207 particle\_emitter(pygamelib.board\_items.GenericStructure PALMS\_UP\_TOGETHER (pygamelib.assets.graphics.Models property), 430 attribute), 207 PANCAKES (pygamelib.assets.graphics.Models attribute), particle\_emitter(pygamelib.board items.GenericStructureComplexCompl property), 420 particle\_emitter (pygamelib.board\_items.Immovable PANDA (pygamelib.assets.graphics.Models attribute), 207 **PAPERCLIP** (pygamelib.assets.graphics.Models property), 439 at-(pygamelib.board\_items.Movable tribute), 207 particle\_emitter property), 449 **PARACHUTE** (pygamelib.assets.graphics.Models atparticle\_emitter (pygamelib.board\_items.NPC proptribute), 207 erty), 459 parent (pygamelib.base.Text attribute), 254 parent (pygamelib.gfx.ui.BoxLayout property), 622 particle\_emitter (pygamelib.board\_items.Player property), 469 parent (pygamelib.gfx.ui.Cursor property), 630 parent (pygamelib.gfx.ui.FormLayout property), 638 particle\_emitter (pygamelib.board\_items.Projectile parent (pygamelib.gfx.ui.GridLayout property), 643 property), 481 (pygamelib.board\_items.TextItem parent (pygamelib.gfx.ui.Layout property), 652 particle\_emitter parent (pygamelib.gfx.ui.LineInput property), 659 property), 492 particle\_emitter (pygamelib.board\_items.Tile propparent (pygamelib.gfx.ui.Widget property), 687 PARROT (pygamelib.assets.graphics.Models attribute), erty), 503 particle\_emitter (pygamelib.board\_items.Treasure 207

Index 803

property), 514

- particle\_emitter (pygamelib.board\_items.Wall property), 524 particle\_pool (pygamelib.gfx.particles.CircleEmitter
- property), 693
  particle\_pool (pygamelib.gfx.particles.ParticleEmitter
- particle\_pool (pygametw.gjx.particles.FarticleEmiller property), 713
- ParticleEmitter (class in pygamelib.gfx.particles), 708
- ParticlePool (class in pygamelib.gfx.particles), 715 ParticleSprixel (class in pygamelib.gfx.particles), 723
- PartitionParticle (class in pygamelib.gfx.particles), 732
- PARTY\_POPPER (pygamelib.assets.graphics.Models attribute), 207
- PARTYING\_FACE (pygamelib.assets.graphics.Models attribute), 207
- PASSENGER\_SHIP (pygamelib.assets.graphics.Models attribute), 207
- PASSIVE\_PULL\_DOWN\_OUTPUT\_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 90
- PASSIVE\_PULL\_UP\_OUTPUT\_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 90
- PASSPORT\_CONTROL (pygamelib.assets.graphics.Models attribute), 207
- path (pygamelib.gfx.ui.FileDialog property), 634
- PathActuator (class in pygamelib.actuators), 14
- PathFinder (class in pygamelib.actuators), 23
- PatrolActuator (class in pygamelib.actuators), 18
- pause() (pygamelib.actuators.Actuator method), 9
- pause() (pygamelib.actuators.Behavioral method), 13
- pause() (pygamelib.actuators.PathActuator method), 17
- pause() (pygamelib.actuators.PathFinder method), 28
- pause() (pygamelib.actuators.PatrolActuator method), 21
- pause() (pygamelib.actuators.RandomActuator method), 33
- pause() (pygamelib.actuators.UnidirectionalActuator method), 37
- pause() (pygamelib.engine.Game method), 555
- pause() (pygamelib.gfx.core.Animation method), 583
- PAUSE\_BUTTON (pygamelib.assets.graphics.Models attribute), 207
- PAUSED (pygamelib.constants.State attribute), 530
- PAW\_PRINTS (pygamelib.assets.graphics.Models attribute), 207
- PEACE\_SYMBOL (pygamelib.assets.graphics.Models attribute), 207
- PEACH (pygamelib.assets.graphics.Models attribute), 207
- PEACOCK (pygamelib.assets.graphics.Models attribute), 208
- PEANUTS (pygamelib.assets.graphics.Models attribute),

- 208
- PEAR (pygamelib.assets.graphics.Models attribute), 208 PEN (pygamelib.assets.graphics.Models attribute), 208
- ${\tt PENCIL} \quad (pygamelib. assets. graphics. Models \quad attribute),$
- PENGUIN (pygamelib.assets.graphics.Models attribute), 208
- PENSIVE\_FACE (pygamelib.assets.graphics.Models attribute), 208
- PEOPLE\_HUGGING (pygamelib.assets.graphics.Models attribute), 208

# PEOPLE\_WITH\_BUNNY\_EARS

- (pygamelib.assets.graphics.Models attribute), 208
- PEOPLE\_WRESTLING (pygamelib.assets.graphics.Models attribute), 208
- PERFORMING\_ARTS (pygamelib.assets.graphics.Models attribute), 208
- PERSEVERING\_FACE (pygamelib.assets.graphics.Models attribute), 208
- PERSON (pygamelib.assets.graphics.Models attribute),
- PERSON\_BIKING (pygamelib.assets.graphics.Models attribute), 208
- PERSON\_BLOND\_HAIR (pygamelib.assets.graphics.Models attribute), 208

## PERSON\_BOUNCING\_BALL

- (pygamelib.assets.graphics.Models attribute), 208
- PERSON\_BOWING (pygamelib.assets.graphics.Models attribute), 208

#### PERSON\_CARTWHEELING

- (pygamelib.assets.graphics.Models attribute), 208
- PERSON\_CLIMBING (pygamelib.assets.graphics.Models attribute), 208
- PERSON\_FACEPALMING (pygamelib.assets.graphics.Models attribute), 208
- PERSON\_FENCING (pygamelib.assets.graphics.Models attribute), 208
- PERSON\_FROWNING (pygamelib.assets.graphics.Models attribute), 208

# PERSON\_GESTURING\_NO

(pygamelib.assets.graphics.Models attribute), 208

## PERSON\_GESTURING\_OK

(pygamelib.assets.graphics.Models attribute), 208

#### PERSON\_GETTING\_HAIRCUT

(pygamelib.assets.graphics.Models attribute), 208

## PERSON\_GETTING\_MASSAGE

(pygamelib.assets.graphics.Models attribute), 208

PERSON\_GOLFING (pygamelib.assets.graphics.Models attribute), 208 PERSON\_IN\_BED (pygamelib.assets.graphics.Models attribute), 208 PERSON\_IN\_LOTUS\_POSITION (pygamelib.assets.graphics.Models attribute), PERSON\_IN\_STEAMY\_ROOM (pygamelib.assets.graphics.Models attribute), PERSON\_IN\_SUIT\_LEVITATING (pygamelib.assets.graphics.Models attribute), PERSON\_IN\_TUXEDO (pygamelib.assets.graphics.Models attribute), 209 PERSON\_JUGGLING (pygamelib.assets.graphics.Models attribute), 209 PERSON\_KNEELING (pygamelib.assets.graphics.Models attribute), 209 PERSON\_LIFTING\_WEIGHTS (pygamelib.assets.graphics.Models attribute), PERSON\_MOUNTAIN\_BIKING (pygamelib.assets.graphics.Models attribute). 209 PERSON\_PLAYING\_HANDBALL (pygamelib.assets.graphics.Models attribute), PERSON\_PLAYING\_WATER\_POLO (pygamelib.assets.graphics.Models attribute), PERSON\_POUTING (pygamelib.assets.graphics.Models attribute), 209 PERSON\_RAISING\_HAND (pygamelib.assets.graphics.Models attribute),  ${\tt PERSON\_ROWING\_BOAT}\ (pygamelib. assets. graphics. Models$ attribute), 209 PERSON\_RUNNING (pygamelib.assets.graphics.Models attribute), 209 PERSON\_SHRUGGING (pygamelib.assets.graphics.Models attribute), 209  ${\tt PERSON\_STANDING} \quad (pygamelib. assets. graphics. Models$ attribute), 209 PERSON\_SURFING (pygamelib.assets.graphics.Models attribute), 209 PERSON\_SWIMMING (pygamelib.assets.graphics.Models attribute), 209 PERSON\_TAKING\_BATH (pygamelib.assets.graphics.Models pickable() (pygamelib.board\_items.GenericActionableStructure attribute), 209 PERSON\_TIPPING\_HAND pickable() (pygamelib.board\_items.GenericStructure

(pygamelib.assets.graphics.Models attribute),

PERSON\_WALKING (pygamelib.assets.graphics.Models at-

tribute), 209 PERSON\_WEARING\_TURBAN (pygamelib.assets.graphics.Models attribute), PERSON\_WITH\_SKULLCAP (pygamelib.assets.graphics.Models attribute), PERSON\_WITH\_VEIL (pygamelib.assets.graphics.Models attribute), 209  ${\tt PERSPECTIVE}\, (pygamelib. assets. graphics. Misc Technicals$ attribute), 90 PETRI\_DISH (pygamelib.assets.graphics.Models tribute), 209 PglBaseObject (class in pygamelib.base), 245 PglException, 248 PglInvalidLevelException, 248 PglInvalidTypeException, 248 PglInventoryException, 248 PglObjectIsNotMovableException, 248 PglOutOfBoardBoundException, 248 PICK (pygamelib.assets.graphics.Models attribute), 209 pickable() (pygamelib.board\_items.Actionable method), 267 pickable() (pygamelib.board items.ActionableTile method), 277  $\verb"pickable()" (pygame lib.board\_items. Board Complex Item"$ method), 288 pickable() (pygamelib.board\_items.BoardItem method), 309 pickable() (pygamelib.board\_items.BoardItemComplexComponent method), 298 (pygamelib.board\_items.BoardItemVoid pickable() method), 318 pickable() (pygamelib.board\_items.Camera method), 327 pickable() (pygamelib.board items.Character method), 337 pickable() (pygamelib.board\_items.ComplexDoor method), 347 pickable() (pygamelib.board\_items.ComplexNPC *method*), 359 pickable() (pygamelib.board\_items.ComplexPlayer *method*), 369 pickable() (pygamelib.board\_items.ComplexTreasure method), 380 pickable() (pygamelib.board\_items.ComplexWall method), 391 pickable() (pygamelib.board\_items.Door method), 402

pickable() (pygamelib.board items.GenericStructureComplexComponen

Index 805

*method*), 411

method), 430

method), 420

pickable() (pygamelib.board\_items.Immovable PLAY\_BUTTON (pygamelib.assets.graphics.Models atmethod), 439 tribute), 210 pickable() (pygamelib.board items.Movable method), PLAY\_OR\_PAUSE\_BUTTON (pygamelib.assets.graphics.Models attribute), pickable() (pygamelib.board\_items.NPC method), 460 pickable() (pygamelib.board items.Player method), Player (class in pygamelib.board items), 463 PLAYER\_AUTHORIZED (pygamelib.constants.Permission pickable() (pygamelib.board\_items.Projectile method), attribute), 530 PLEADING\_FACE (pygamelib.assets.graphics.Models atpickable() (pygamelib.board\_items.TextItem method), tribute), 210 PLUNGER (pygamelib.assets.graphics.Models attribute), pickable() (pygamelib.board\_items.Tile method), 503 210 pickable() (pygamelib.board\_items.Treasure method), PLUS (pygamelib.assets.graphics.Models attribute), 210 514 POLICE\_CAR (pygamelib.assets.graphics.Models pickable() (pygamelib.board\_items.Wall method), 524 tribute), 210 PICKUP\_TRUCK (pygamelib.assets.graphics.Models POLICE\_CAR\_LIGHT (pygamelib.assets.graphics.Models attribute), 209 attribute), 210 PIE (pygamelib.assets.graphics.Models attribute), 209 POLICE\_OFFICER (pygamelib.assets.graphics.Models at-PIG (pygamelib.assets.graphics.Models attribute), 209 tribute), 210 PIG\_FACE (pygamelib.assets.graphics.Models attribute), POODLE (pygamelib.assets.graphics.Models attribute), 209 PIG\_NOSE (pygamelib.assets.graphics.Models attribute), pool (pygamelib.gfx.particles.ParticlePool property), 209 716 PILE\_OF\_POO (pygamelib.assets.graphics.Models atpop() (pygamelib.gfx.core.SpriteCollection method), 591 tribute), 209 POPCORN (pygamelib.assets.graphics.Models attribute), PILL (pygamelib.assets.graphics.Models attribute), 209 210 PINCHED\_FINGERS (pygamelib.assets.graphics.Models popitem() (pygamelib.gfx.core.SpriteCollection method), 591 attribute), 209 PINCHING\_HAND (pygamelib.assets.graphics.Models atposition\_as\_vector() (pygamelib.board\_items.Actionable tribute), 210 method), 267 PINE\_DECORATION (pygamelib.assets.graphics.Models attribute), 210 position\_as\_vector() (pygamelib.board\_items.ActionableTile **PINEAPPLE** (pygamelib.assets.graphics.Models atmethod), 278 tribute), 210 PING\_PONG (pygamelib.assets.graphics.Models position\_as\_vector() attribute), 210 (pygamelib.board\_items.BoardComplexItem PISCES (pygamelib.assets.graphics.Models attribute), method), 289 position\_as\_vector() PISTOL (pygamelib.assets.graphics.Models attribute), (pygamelib.board\_items.BoardItem method), 309 PIZZA (pygamelib.assets.graphics.Models attribute), 210 position\_as\_vector() PIñATA (pygamelib.assets.graphics.Models attribute), (pygamelib.board items.BoardItemComplexComponent method), 298 PLACARD (pygamelib.assets.graphics.Models attribute), position\_as\_vector() (pygamelib.board\_items.BoardItemVoid method), 318 place() (pygamelib.engine.Screen method), 578 place\_item() (pygamelib.engine.Board method), 541 position\_as\_vector() PLACE\_OF\_INTEREST\_SIGN (pygamelib.board\_items.Camera method), (pygamelib.assets.graphics.MiscTechnicals 327 attribute), 90 position\_as\_vector() PLACE\_OF\_WORSHIP (pygamelib.assets.graphics.Models (pygamelib.board\_items.Character method), attribute), 210 play\_all() (pygamelib.gfx.core.Animation method), position\_as\_vector() (pygamelib.board items.ComplexDoor 583

<pre>method), 348 position_as_vector()</pre>	POSTBOX (pygamelib.assets.graphics.Models attribute), 210
(pygamelib.board_items.ComplexNPC method), 359	POT_OF_FOOD (pygamelib.assets.graphics.Models attribute), 210
position_as_vector() (pygamelib.board_items.ComplexPlayer	POTABLE_WATER (pygamelib.assets.graphics.Models attribute), 210
method), 370 position_as_vector()	POTATO (pygamelib.assets.graphics.Models attribute), 210
(pygamelib.board_items.ComplexTreasure method), 381	POTTED_PLANT (pygamelib.assets.graphics.Models attribute), 210
<pre>position_as_vector()</pre>	POULTRY_LEG (pygamelib.assets.graphics.Models attribute), 210
<pre>method), 392 position_as_vector() (pygamelib.board_items.Door</pre>	POUND_BANKNOTE (pygamelib.assets.graphics.Models attribute), 210
<pre>method), 402 position_as_vector()</pre>	POUTING_CAT (pygamelib.assets.graphics.Models attribute), 210
(pygamelib.board_items.GenericActionableStruc method), 411	attribute), 210
<pre>position_as_vector()</pre>	POWER_ON_OFF_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 90
position_as_vector()	POWER_ON_SYMBOL (pygamelib.assets.graphics.MiscTechnicals
position_as_vector()	attribute), 90
(pygamelib.board_items.Immovable method), 439	
<pre>position_as_vector()</pre>	PRAYER_BEADS (pygamelib.assets.graphics.Models attribute), 211 PREGNANT_WOMAN (pygamelib.assets.graphics.Models at-
<pre>position_as_vector() (pygamelib.board_items.NPC</pre>	tribute), 211 PRETZEL (pygamelib.assets.graphics.Models attribute),
<pre>position_as_vector()</pre>	211
(pygamelib.board_items.Player method), 469	PREVIOUS_PAGE (pygamelib.assets.graphics.MiscTechnicals attribute), 90
<pre>position_as_vector()</pre>	PRINCE (pygamelib.assets.graphics.Models attribute), 211
482	DDINGEGG ( 121 , 12 M 11 , 13 )
nosition as vector()	PRINCESS (pygamelib.assets.graphics.Models attribute),
<pre>position_as_vector()</pre>	PRINCESS (pygamelib.assets.graphics.Models attribute), 211 print_formatted() (pygamelib.base.Text method), 254 PRINT_SCREEN_SYMBOL
(pygamelib.board_items.TextItem method),	211 print_formatted() (pygamelib.base.Text method), 254
<pre>(pygamelib.board_items.TextItem method), 493 position_as_vector() (pygamelib.board_items.Tile     method), 503 position_as_vector()     (pygamelib.board_items.Treasure method),</pre>	print_formatted() (pygamelib.base.Text method), 254 PRINT_SCREEN_SYMBOL
<pre>(pygamelib.board_items.TextItem method),</pre>	print_formatted() (pygamelib.base.Text method), 254  PRINT_SCREEN_SYMBOL
(pygamelib.board_items.TextItem method), 493  position_as_vector() (pygamelib.board_items.Tile method), 503  position_as_vector() (pygamelib.board_items.Treasure method), 514  position_as_vector() (pygamelib.board_items.Wall method), 524  POSITION_INDICATOR (pygamelib.assets.graphics.MiscTell	print_formatted() (pygamelib.base.Text method), 254 PRINT_SCREEN_SYMBOL
(pygamelib.board_items.TextItem method), 493  position_as_vector() (pygamelib.board_items.Tile method), 503  position_as_vector()	print_formatted() (pygamelib.base.Text method), 254  PRINT_SCREEN_SYMBOL
(pygamelib.board_items.TextItem method), 493  position_as_vector() (pygamelib.board_items.Tile method), 503  position_as_vector() (pygamelib.board_items.Treasure method), 514  position_as_vector() (pygamelib.board_items.Wall method), 524  POSITION_INDICATOR (pygamelib.assets.graphics.MiscTenattribute), 90	print_formatted() (pygamelib.base.Text method), 254 PRINT_SCREEN_SYMBOL

tribute), 211	RABBIT (pygamelib.assets.graphics.Models attribute),
Projectile (class in pygamelib.board_items), 473	211
PROJECTIVE (pygamelib.assets.graphics.MiscTechnicals attribute), 90	RABBIT_FACE (pygamelib.assets.graphics.Models attribute), 211
PURPLE_CIRCLE (pygamelib.assets.graphics.Models attribute), 211	RACCOON (pygamelib.assets.graphics.Models attribute), 211
PURPLE_HEART (pygamelib.assets.graphics.Models attribute), 211	RACING_CAR (pygamelib.assets.graphics.Models attribute), 211
PURPLE_SQUARE (pygamelib.assets.graphics.Models at-	RADICAL_SYMBOL_BOTTOM
tribute), 211	(pygame lib. assets. graphics. Misc Technicals
PURSE (pygamelib.assets.graphics.Models attribute), 211	attribute), 90
PUSHPIN (pygamelib.assets.graphics.Models attribute),	RADIO (pygamelib.assets.graphics.Models attribute), 211 RADIO_BUTTON (pygamelib.assets.graphics.Models
PUZZLE_PIECE (pygamelib.assets.graphics.Models	attribute), 211
attribute), 211	${\tt RADIOACTIVE}  (pygamelib. assets. graphics. Models  at-$
pygamelib.constants	tribute), 211
module, 532	RAILWAY_CAR (pygamelib.assets.graphics.Models attribute), 211
Q	RAILWAY_TRACK (pygamelib.assets.graphics.Models at-
QUADRANT_LOWER_LEFT	tribute), 211
(pygamelib.assets.graphics.Blocks attribute), 43	RAINBOW (pygamelib.assets.graphics.Models attribute), 211
QUADRANT_LOWER_RIGHT	RAISED_BACK_OF_HAND
(pygamelib.assets.graphics.Blocks attribute), 43	(pygamelib.assets.graphics.Models attribute), 211
QUADRANT_UPPER_LEFT	RAISED_FIST (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.Blocks attribute), 43	tribute), 211 RAISED_HAND (pygamelib.assets.graphics.Models at-
QUADRANT_UPPER_LEFT_AND_LOWER_LEFT_AND_LOWER_	
(pygamelib.assets.graphics.Blocks attribute), 43	RAISING_HANDS (pygamelib.assets.graphics.Models attribute), 211
QUADRANT_UPPER_LEFT_AND_LOWER_RIGHT	RAM (pygamelib.assets.graphics.Models attribute), 211
(pygamelib.assets.graphics.Blocks attribute), 43	random() (pygamelib.gfx.core.Color class method), 616 RandomActuator (class in pygamelib.actuators), 30
QUADRANT_UPPER_LEFT_AND_UPPER_RIGHT_AND_LOWER	RampdomColorParticle (class in
(pygamelib.assets.graphics.Blocks attribute),	pygamelib.gfx.particles), 738 RandomColorPartitionParticle (class in
43 QUADRANT_UPPER_LEFT_AND_UPPER_RIGHT_AND_LOWER	•
(pygamelib.assets.graphics.Blocks attribute),	randomize() (pygamelib.gfx.core.Color method), 616
43	RAT (pygamelib.assets.graphics.Models attribute), 211
QUADRANT_UPPER_RIGHT	RAZOR (pygamelib.assets.graphics.Models attribute), 212
(pygamelib.assets.graphics.Blocks attribute), 43	RECEIPT (pygamelib.assets.graphics.Models attribute), 212
QUADRANT_UPPER_RIGHT_AND_LOWER_LEFT	RECORD_BUTTON (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.Blocks attribute),	tribute), 212 RECYCLING_SYMBOL (pygamelib.assets.graphics.Models
43 QUADRANT_UPPER_RIGHT_AND_LOWER_LEFT_AND_LOWER	
(pygamelib.assets.graphics.Blocks attribute),	red (pygamelib.gfx.ui.ColorPicker property), 627
43	red() (pygamelib.base.Text static method), 255
QUESTION_MARK (pygamelib.assets.graphics.Models attribute), 211	RED_APPLE (pygamelib.assets.graphics.Models attribute), 212
_	red_bright() (pygamelib.base.Text static method), 255
R	RED_CIRCLE (pygamelib.assets.graphics.Models at- tribute), 212
r (pygamelib.gfx.core.Color property), 616	tribute), 212

- RED\_HAIR (pygamelib.assets.graphics.Models attribute), 212.
- RED\_HEART (pygamelib.assets.graphics.Models at tribute), 212
- RED\_PAPER\_LANTERN (pygamelib.assets.graphics.Models attribute), 212
- red\_rect() (pygamelib.gfx.core.Sprixel class method),
  609
- red\_rect() (pygamelib.gfx.particles.ParticleSprixel class method), 729
- RED\_SQUARE (pygamelib.assets.graphics.Models attribute), 212
- red\_square() (pygamelib.gfx.core.Sprixel class method), 610
- red\_square() (pygamelib.gfx.particles.ParticleSprixel class method), 730
- RED\_TRIANGLE\_POINTED\_DOWN
  - (pygamelib.assets.graphics.Models attribute), 212
- RED\_TRIANGLE\_POINTED\_UP
  - (pygamelib.assets.graphics.Models attribute), 212
- redo() (pygamelib.base.History method), 242
- redo() (pygamelib.gfx.ui.LineInput method), 659
- REGISTERED (pygamelib.assets.graphics.Models attribute), 212
- relative\_column (pygamelib.gfx.ui.Cursor property),
- relative\_row (pygamelib.gfx.ui.Cursor property), 630
- RELIEVED\_FACE (pygamelib.assets.graphics.Models attribute), 212
- REMINDER\_RIBBON (pygamelib.assets.graphics.Models attribute), 212
- remove\_constraint() (pygamelib.engine.Inventory method), 566
- remove\_directional\_animation()
  - (pygamelib.board\_items.Projectile method), 482
- remove\_directional\_model()
  - (pygamelib.board\_items.Projectile method), 482
- remove\_frame() (pygamelib.gfx.core.Animation method), 583
- ${\tt remove\_item()}\ (pygame lib.engine. Board\ method),\ 541$
- remove\_npc() (pygamelib.engine.Game method), 555
- remove\_row() (pygamelib.gfx.ui.FormLayout method),
  638
- remove\_waypoint() (pygamelib.actuators.PathFinder method), 29
- rename() (pygamelib.gfx.core.SpriteCollection method),
  591

- render() (pygamelib.engine.Screen method), 579
- render() (pygamelib.gfx.particles.ColorParticle method), 698
- render() (pygamelib.gfx.particles.ColorPartitionParticle method), 704
- render() (pygamelib.gfx.particles.Particle method), 720
- render() (pygamelib.gfx.particles.PartitionParticle method), 736
- render() (pygamelib.gfx.particles.RandomColorParticle method), 742
- render() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 747
- render\_cell() (pygamelib.engine.Board method), 542
- render\_to\_buffer() (pygamelib.base.Text method), 255
- render\_to\_buffer() (pygamelib.board\_items.Actionable method), 268
- render\_to\_buffer() (pygamelib.board\_items.ActionableTile method), 278
- render\_to\_buffer() (pygamelib.board\_items.BoardComplexItem method), 289
- render\_to\_buffer() (pygamelib.board\_items.BoardItem method), 309

- render\_to\_buffer() (pygamelib.board\_items.ComplexWall method), 392
- render\_to\_buffer() (pygamelib.board\_items.Door method), 402
- render\_to\_buffer() (pygamelib.board\_items.GenericActionableStructur method), 412
- render\_to\_buffer() (pygamelib.board\_items.GenericStructure method), 430
- render\_to\_buffer() (pygamelib.board\_items.GenericStructureComplext method), 420

- render\_to\_buffer() (pygamelib.board items.NPC

<pre>method), 460 render_to_buffer()</pre>		method), 668 render_to_buffer() (pygamelib.gfx.ui.MenuBar
method), 470		method), 671
* *		render_to_buffer() (pygamelib.gfx.ui.MessageDialog
method), 482	= · ·	method), 675
* ·		render_to_buffer() (pygamelib.gfx.ui.MultiLineInputDialog
method), 493		method), 677
render_to_buffer()		
method), 504		method), 680 (pygameub.g/x.ui.ProgressBar
		render_to_buffer() (pygamelib.gfx.ui.ProgressDialog
method), 515		method), 681
render_to_buffer()		
method), 525		method), 688
render_to_buffer()		REPEAT_BUTTON (pygamelib.assets.graphics.Models at-
method), 542		tribute), 212
render_to_buffer()		
	1.0	
method), 599		(pygamelib.assets.graphics.Models attribute),
render_to_buffer()		212
method), 610		RESCUE_WORKERS_HELMET
	(pygamelib.gfx.particles.CircleEmi	tter (pygamelib.assets.graphics.Models attribute), 212
method), 693		
	=	miteset() (pygamelib.base.History method), 243
method), 713		reset() (pygamelib.gfx.core.Animation method), 584
	- · ·	riceset() (pygamelib.gfx.particles.ColorParticle method),
method), 730		698
render_to_buffer() 619	(pygamelib.gjx.ui.Box meinoa),	reset() (pygamelib.gfx.particles.ColorPartitionParticle
	(maamalih afr ui Rayl avout	method), 704
render_to_buffer()		reset() (pygamelib.gfx.particles.Particle method), 720
method), 622		reset() (pygamelib.gfx.particles.PartitionParticle
render_to_buffer()	2.7.0	method), 736
method), 627		reset() (pygamelib.gfx.particles.RandomColorParticle
	(pygamelib.gfx.ui.ColorPickerDialo	
method), 624		reset() (pygamelib.gfx.particles.RandomColorPartitionParticle
render_to_buffer()	2.0	method), 748
method), 630		reset_lifespan() (pygamelib.gfx.particles.ColorParticle
render_to_buffer()	(pygamelib.gfx.ui.FileDialog	
		method), 699
method), 634		$\verb"reset_lifespan"() (pygame lib.gfx.particles. Color Partition Particle$
render_to_buffer()	(pygamelib.gfx.ui.FormLayout	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705
render_to_buffer()  method), 639	(pygamelib.gfx.ui.FormLayout	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705 reset_lifespan() (pygamelib.gfx.particles.Particle
render_to_buffer()  method), 639  render_to_buffer()	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705 reset_lifespan() (pygamelib.gfx.particles.Particle method), 721
render_to_buffer()  method), 639  render_to_buffer()  method), 643	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705 reset_lifespan() (pygamelib.gfx.particles.Particle method), 721 reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705 reset_lifespan() (pygamelib.gfx.particles.Particle method), 721 reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737 reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector)  (pygamelib.gfx.ui.GridSelectorDial	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705 reset_lifespan() (pygamelib.gfx.particles.Particle method), 721 reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737 reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle method), 742
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705 reset_lifespan() (pygamelib.gfx.particles.Particle method), 721 reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737 reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle dog method), 742 reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle)
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705 reset_lifespan() (pygamelib.gfx.particles.Particle method), 721 reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737 reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle dog method), 742 reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()  method), 652	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737  reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle log method), 742  reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748  resize() (pygamelib.gfx.particles.ParticlePool method),
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()  method), 652  render_to_buffer()	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout  (pygamelib.gfx.ui.LineInput	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737  reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle log method), 742  reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748  resize() (pygamelib.gfx.particles.ParticlePool method), 716
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()  method), 652  render_to_buffer()  method), 659	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout  (pygamelib.gfx.ui.LineInput	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737  reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle log method), 742  reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748  resize() (pygamelib.gfx.particles.ParticlePool method), 716  resize_pool() (pygamelib.gfx.particles.CircleEmitter
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()  method), 652  render_to_buffer()  method), 659  render_to_buffer()	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout  (pygamelib.gfx.ui.LineInput	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737  reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle log method), 742  reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748  resize() (pygamelib.gfx.particles.ParticlePool method), 716  resize_pool() (pygamelib.gfx.particles.CircleEmitter method), 693
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()  method), 652  render_to_buffer()  method), 659  render_to_buffer()  method), 669	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout  (pygamelib.gfx.ui.LineInput  (pygamelib.gfx.ui.LineInputDialog	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737  reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle log method), 742  reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748  resize() (pygamelib.gfx.particles.ParticlePool method), 716  resize_pool() (pygamelib.gfx.particles.CircleEmitter method), 693  resize_pool() (pygamelib.gfx.particles.ParticleEmitter
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()  method), 652  render_to_buffer()  method), 659  render_to_buffer()  method), 662  render_to_buffer()	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout  (pygamelib.gfx.ui.LineInput	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737  reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle dog method), 742  reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748  resize() (pygamelib.gfx.particles.ParticlePool method), 716  resize_pool() (pygamelib.gfx.particles.CircleEmitter method), 693  resize_pool() (pygamelib.gfx.particles.ParticleEmitter method), 713
render_to_buffer()  method), 639  render_to_buffer()  method), 643  render_to_buffer()  method), 649  render_to_buffer()  method), 646  render_to_buffer()  method), 652  render_to_buffer()  method), 659  render_to_buffer()  method), 669	(pygamelib.gfx.ui.FormLayout  (pygamelib.gfx.ui.GridLayout  (pygamelib.gfx.ui.GridSelector  (pygamelib.gfx.ui.GridSelectorDial  (pygamelib.gfx.ui.Layout  (pygamelib.gfx.ui.LineInput  (pygamelib.gfx.ui.LineInputDialog  (pygamelib.gfx.ui.Menu method),	reset_lifespan() (pygamelib.gfx.particles.ColorPartitionParticle method), 705  reset_lifespan() (pygamelib.gfx.particles.Particle method), 721  reset_lifespan() (pygamelib.gfx.particles.PartitionParticle method), 737  reset_lifespan() (pygamelib.gfx.particles.RandomColorParticle log method), 742  reset_lifespan() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 748  resize() (pygamelib.gfx.particles.ParticlePool method), 716  resize_pool() (pygamelib.gfx.particles.CircleEmitter method), 693  resize_pool() (pygamelib.gfx.particles.ParticleEmitter

restorable() (pygamelib.board items.ActionableTile REVERSE\_BUTTON (pygamelib.assets.graphics.Models atmethod), 278 tribute), 212 restorable() (pygamelib.board items.BoardComplexItenREVERSED\_NOT\_SIGN (pygamelib.assets.graphics.MiscTechnicals attribute), 90 method), 289 restorable() (pygamelib.board\_items.BoardItem REVOLVING\_HEARTS (pygamelib.assets.graphics.Models method), 310 attribute), 212 restorable() (pygamelib.board items.BoardItemComplexICHINDGEROS (pygamelib.assets.graphics.Models method), 299 tribute), 212 (pygamelib.assets.graphics.Models attribute), restorable() (pygamelib.board items.BoardItemVoid RIBBON method), 319 212 restorable() (pygamelib.board\_items.Camera RICE\_BALL (pygamelib.assets.graphics.Models atmethod), 328 tribute), 212 (pygamelib.assets.graphics.Models (pygamelib.board\_items.Character restorable() RICE\_CRACKER attribute), 212 *method*), 338 restorable() (pygamelib.board\_items.ComplexDoor RIGHT (pygamelib.constants.Alignment attribute), 528 *method*), 348 RIGHT (pygamelib.constants.Direction attribute), 529 (pygamelib.board\_items.ComplexNPC RIGHT\_ANGER\_BUBBLE (pygamelib.assets.graphics.Models restorable() *method*), 360 attribute), 212 restorable() (pygamelib.board items.ComplexPlayer RIGHT\_ANGLE\_WITH\_DOWNWARDS\_ZIGZAG\_ARROW *method*), 370 (pygamelib.assets.graphics.MiscTechnicals restorable() (pygamelib.board\_items.ComplexTreasure attribute), 90 method), 381 RIGHT\_ARROW (pygamelib.assets.graphics.Models atrestorable() (pygamelib.board\_items.ComplexWall tribute), 212 method), 392 RIGHT\_ARROW\_CURVING\_DOWN restorable() (pygamelib.board\_items.Door method), (pygamelib.assets.graphics.Models attribute), restorable() (pygamelib.board\_items.GenericActionable %FGHFUrARROW\_CURVING\_LEFT (pygamelib.assets.graphics.Models *method*), 412 attribute), restorable() (pygamelib.board\_items.GenericStructure 212 RIGHT\_ARROW\_CURVING\_UP *method*), 431 restorable() (pygamelib.board\_items.GenericStructureComplexCo(ppygamelib.assets.graphics.Models attribute), method), 421 212  $RIGHT\_CEILING$  (pygamelib.assets.graphics.MiscTechnicals restorable() (pygamelib.board\_items.Immovable attribute), 90 method), 440 RIGHT\_CURLY\_BRACKET\_LOWER\_HOOK restorable() (pygamelib.board items.Movable method), 450 (pygamelib.assets.graphics.MiscTechnicals restorable() (pygamelib.board\_items.NPC method), attribute), 90 RIGHT\_CURLY\_BRACKET\_MIDDLE\_PIECE restorable() (pygamelib.board\_items.Player method), (pygamelib.assets.graphics.MiscTechnicals 470 attribute), 90 restorable() (pygamelib.board items.Projectile RIGHT\_CURLY\_BRACKET\_UPPER\_HOOK method), 482 (pygamelib.assets.graphics.MiscTechnicals (pygamelib.board\_items.TextItem attribute), 90 restorable() method), 493 RIGHT\_DOWN\_HEAVY\_AND\_LEFT\_UP\_LIGHT restorable() (pygamelib.board\_items.Tile method), (pygamelib.assets.graphics.BoxDrawings 504 attribute), 55 RIGHT\_FACING\_FIST (pygamelib.assets.graphics.Models restorable() (pygamelib.board\_items.Treasure method), 515 attribute), 213 restorable() (pygamelib.board\_items.Wall method), RIGHT\_FLOOR (pygamelib.assets.graphics.MiscTechnicals attribute), 90 RESTROOM (pygamelib.assets.graphics.Models attribute), RIGHT\_HALF\_BLACK\_CIRCLE (pygamelib.assets.graphics.GeometricShapes RETURN\_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 65 RIGHT\_HALF\_BLOCK (pygamelib.assets.graphics.Blocks attribute), 90

attribute), 43	ROASTED_SWEET_POTATO
RIGHT_HEAVY_AND_LEFT_DOWN_LIGHT	(pygamelib.assets.graphics.Models attribute),
(pygamelib.assets.graphics.BoxDrawings attribute), 55	213 ROBOT (pygamelib.assets.graphics.Models attribute), 213
RIGHT_HEAVY_AND_LEFT_UP_LIGHT	ROCK (pygamelib.assets.graphics.Models attribute), 213
(pygamelib.assets.graphics.BoxDrawings	ROCKET (pygamelib.assets.graphics.Models attribute),
attribute), 55	213
RIGHT_HEAVY_AND_LEFT_VERTICAL_LIGHT	ROLL_OF_PAPER (pygamelib.assets.graphics.Models at-
(pygamelib.assets.graphics.BoxDrawings	tribute), 213
attribute), 55	ROLLED_UP_NEWSPAPER
RIGHT_LIGHT_AND_LEFT_DOWN_HEAVY	(pygamelib.assets.graphics.Models attribute),
(pygamelib.assets.graphics.BoxDrawings	213
attribute), 55	ROLLER_COASTER (pygamelib.assets.graphics.Models at-
RIGHT_LIGHT_AND_LEFT_UP_HEAVY	tribute), 213
(pygamelib.assets.graphics.BoxDrawings	ROLLER_SKATE (pygamelib.assets.graphics.Models
attribute), 55	attribute), 213
RIGHT_LIGHT_AND_LEFT_VERTICAL_HEAVY	ROLLING_ON_THE_FLOOR_LAUGHING
(pygamelib.assets.graphics.BoxDrawings	(pygamelib.assets.graphics.Models attribute),
attribute), 55	213
RIGHT_ONE_EIGHTH_BLOCK	ROOSTER (pygamelib.assets.graphics.Models attribute),
(pygamelib.assets.graphics.Blocks attribute),	213
43	ROSE (pygamelib.assets.graphics.Models attribute), 213
RIGHT_PARENTHESIS_EXTENSION	ROSETTE (pygamelib.assets.graphics.Models attribute),
(pygame lib. assets. graphics. Misc Technicals	213
attribute), 90	ROUND_PUSHPIN (pygamelib.assets.graphics.Models at-
RIGHT_PARENTHESIS_LOWER_HOOK	tribute), 213
(pygamelib.assets.graphics.MiscTechnicals attribute), 91	rounding_precision (pygamelib.base.Vector2D attribute), 259
RIGHT_PARENTHESIS_UPPER_HOOK	row (pygamelib.base.Vector2D property), 259
(pygamelib.assets.graphics.MiscTechnicals	row (pygamelib.board_items.Actionable property), 268
attribute), 91	row (pygamelib.board_items.ActionableTile property),
RIGHT_POINTING_ANGLE_BRACKET	278
(pygamelib.assets.graphics.MiscTechnicals	row (pygamelib.board_items.BoardComplexItem prop-
attribute), 91	erty), 289
RIGHT_SQUARE_BRACKET_EXTENSION	row (pygamelib.board_items.BoardItem property), 310
(pygamelib.assets.graphics.MiscTechnicals	row (pygamelib.board_items.BoardItemComplexComponent
attribute), 91	property), 299
RIGHT_SQUARE_BRACKET_LOWER_CORNER	row (pygamelib.board_items.BoardItemVoid property),
(pygamelib.assets.graphics.MiscTechnicals	319
attribute), 91	row (pygamelib.board_items.Camera property), 328
RIGHT_SQUARE_BRACKET_UPPER_CORNER	row (pygamelib.board_items.Character property), 338
(pygamelib.assets.graphics.MiscTechnicals attribute), 91	<pre>row (pygamelib.board_items.ComplexDoor property),</pre>
RIGHT_UP_HEAVY_AND_LEFT_DOWN_LIGHT	row (pygamelib.board_items.ComplexNPC property),
(pygamelib.assets.graphics.BoxDrawings	360
attribute), 55	row (pygamelib.board_items.ComplexPlayer property),
RIGHT_VERTICAL_BOX_LINE	370
(pygamelib.assets.graphics.MiscTechnicals	row (pygamelib.board_items.ComplexTreasure property),
attribute), 91	381
RING (pygamelib.assets.graphics.Models attribute), 213	row (pygamelib.board_items.ComplexWall property), 392
RING_OPERATOR (pygamelib.assets.graphics.GeometricSha	
attribute), 65	row (pygamelib.board_items.GenericActionableStructure
RINGED_PLANET (pygamelib.assets.graphics.Models at-	property), 412
tribute), 213	row (pygamelib.board_items.GenericStructure property),

431	SANDWICH (pygamelib.assets.graphics.Models attribute),
row (pygamelib.board_items.GenericStructureComplexCom	
property), 421	SANTA_CLAUS (pygamelib.assets.graphics.Models at-
row (pygamelib.board_items.Immovable property), 440	tribute), 213
row (pygamelib.board_items.Movable property), 450	SARI (pygamelib.assets.graphics.Models attribute), 213
row (pygamelib.board_items.NPC property), 460	SATELLITE (pygamelib.assets.graphics.Models at-
row (pygamelib.board_items.Player property), 470	tribute), 213
row (pygamelib.board_items.Projectile property), 483	SATELLITE_ANTENNA (pygamelib.assets.graphics.Models
row (pygamelib.board_items.TextItem property), 493	attribute), 213
row (pygamelib.board_items.Tile property), 504	SAUROPOD (pygamelib.assets.graphics.Models attribute),
row (pygamelib.board_items.Treasure property), 515	214
row (pygamelib.board_items.Wall property), 525	<pre>save_board() (pygamelib.engine.Game method), 556</pre>
row (pygamelib.gfx.particles.CircleEmitter property), 693	<pre>save_config() (pygamelib.engine.Game method), 557</pre>
row (pygamelib.gfx.particles.ColorParticle property),	SAXOPHONE (pygamelib.assets.graphics.Models at-
699	tribute), 214
row (pygamelib.gfx.particles.ColorPartitionParticle	scalable (pygamelib.gfx.core.Font property), 588
property), 705	scale() (pygamelib.gfx.core.Sprite method), 599
row (pygamelib.gfx.particles.Particle property), 721	SCARF (pygamelib.assets.graphics.Models attribute), 214
row (pygamelib.gfx.particles.ParticleEmitter property),	SCHOOL (pygamelib.assets.graphics.Models attribute),
713	214
row (pygamelib.gfx.particles.PartitionParticle property), 737	SCISSORS (pygamelib.assets.graphics.Models attribute), 214
row (pygamelib.gfx.particles.RandomColorParticle prop-	SCORPIO (pygamelib.assets.graphics.Models attribute),
erty), 743	214
row (pygamelib.gfx.particles.RandomColorPartitionParticl	eSCORPION (pygamelib.assets.graphics.Models attribute),
property), 748	214
<pre>row_minimum_height (pygamelib.gfx.ui.FormLayout</pre>	Screen (class in pygamelib.engine), 568
property), 639	screen_column (pygamelib.actuators.Actuator prop-
<pre>row_minimum_height (pygamelib.gfx.ui.GridLayout</pre>	erty), 10
property), 644	screen_column (pygamelib.actuators.Behavioral prop-
RUGBY_FOOTBALL (pygamelib.assets.graphics.Models at-	erty), 13
tribute), 213	screen_column (pygamelib.actuators.PathActuator
run() (pygamelib.engine.Game method), 555	property), 17
RUNNING (pygamelib.constants.State attribute), 530	screen_column (pygamelib.actuators.PathFinder prop-
RUNNING_SHIRT (pygamelib.assets.graphics.Models at-	erty), 29
tribute), 213	screen_column (pygamelib.actuators.PatrolActuator
RUNNING_SHOE (pygamelib.assets.graphics.Models	property), 21
attribute), 213	screen_column (pygamelib.actuators.RandomActuator
	property), 33
S	screen_column (pygamelib.actuators.UnidirectionalActuator
SAD_BUT_RELIEVED_FACE	property), 37
(pygamelib.assets.graphics.Models attribute),	screen_column (pygamelib.base.PglBaseObject prop-
213	erty), 247
SAFETY_PIN (pygamelib.assets.graphics.Models at-	screen_column (pygamelib.base.Text property), 255
tribute), 213	screen_column (pygamelib.board_items.Actionable
SAFETY_VEST (pygamelib.assets.graphics.Models at-	property), 268
tribute), 213	screen_column (pygamelib.board_items.ActionableTile
SAGITTARIUS (pygamelib.assets.graphics.Models at-	property), 278
tribute), 213	screen_column(pygamelib.board_items.BoardComplexItem
SAILBOAT (pygamelib.assets.graphics.Models attribute),	property), 289
213	screen_column (pygamelib.board_items.BoardItem
SAKE (pygamelib.assets.graphics.Models attribute), 213	property), 310
SALT (pygamelib.assets.graphics.Models attribute), 213	screen_column(pygamelib.board_items.BoardItemComplexComponent
2 Wygamemoiassessigrapinesinioueis unitome), 213	property), 299

screen\_column (pygamelib.gfx.core.Sprite property),

screen\_column (pygamelib.gfx.core.Sprixel property),

610

screen\_column (pygamelib.board items.BoardItemVoid screen\_column (pygamelib.gfx.particles.CircleEmitter property), 319 property), 693 screen\_column (pygamelib.board items.Camera propscreen\_column (pygamelib.gfx.particles.ColorParticle property), 699 erty), 328 (pygamelib.board\_items.Character screen\_column  $screen\_column$  (pygamelib.gfx.particles.ColorPartitionParticle property), 705 property), 338 screen\_column (pygamelib.board items.ComplexDoor screen\_column (pygamelib.gfx.particles.Particle propproperty), 348 erty), 721  $screen\_column(pygamelib.gfx.particles.ParticleEmitter)$ screen\_column (pygamelib.board items.ComplexNPC property), 713 property), 360 screen\_column (pygamelib.board\_items.ComplexPlayer screen\_column (pygamelib.gfx.particles.ParticleSprixel property), 370 property), 730 screen\_column(pygamelib.board\_items.ComplexTreasurescreen\_column(pygamelib.gfx.particles.PartitionParticle property), 382 property), 737 screen\_column (pygamelib.board\_items.ComplexWall screen\_column (pygamelib.gfx.particles.RandomColorParticle property), 393 property), 743 screen\_column (pygamelib.board\_items.Door prop- screen\_column(pygamelib.gfx.particles.RandomColorPartitionParticle erty), 403 property), 748 screen\_column (pygamelib.board\_items.GenericActionablestreemreolumn (pygamelib.gfx.ui.BoxLayout property), property), 412 screen\_column (pygamelib.board\_items.GenericStructure screen\_column (pygamelib.gfx.ui.Cursor property), 631 screen\_column (pygamelib.gfx.ui.FormLayout propproperty), 431 screen\_column (pygamelib.board\_items.GenericStructureComplexControlare) screen\_column (pygamelib.gfx.ui.GridLayout propproperty), 421 screen\_column (pygamelib.board\_items.Immovable erty), 644 screen\_column (pygamelib.gfx.ui.Layout property), 652 property), 440 screen\_column (pygamelib.board\_items.Movable propscreen\_column (pygamelib.gfx.ui.LineInput property), erty), 450 660 screen\_column (pygamelib.board\_items.NPC propscreen\_column (pygamelib.gfx.ui.Widget property), 688 erty), 461 screen\_row (pygamelib.actuators.Actuator property), 10 screen\_column (pygamelib.board\_items.Player propscreen\_row (pygamelib.actuators.Behavioral property), erty), 470 13 screen\_column (pygamelib.board\_items.Projectile screen\_row (pygamelib.actuators.PathActuator propproperty), 483 *erty*), 17 screen\_column (pygamelib.board\_items.TextItem propscreen\_row (pygamelib.actuators.PathFinder property), erty), 493 screen\_column (pygamelib.board items.Tile property), screen\_row (pygamelib.actuators.PatrolActuator property), 21 (pygamelib.actuators.RandomActuator screen\_column (pygamelib.board\_items.Treasure propscreen\_row erty), 515 property), 33 screen\_column (pygamelib.board items.Wall property), screen\_row(pygamelib.actuators.UnidirectionalActuator property), 37 screen\_column (pygamelib.engine.Board property), 542 screen\_row (pygamelib.base.PglBaseObject property), screen\_column (pygamelib.engine.Game property), 557 247 screen\_row (pygamelib.base.Text property), 255 screen\_column (pygamelib.engine.Inventory property), 566 screen\_row (pygamelib.board\_items.Actionable propscreen\_column (pygamelib.engine.Screen property), erty), 268 579 (pygamelib.board\_items.ActionableTile screen\_row screen\_column (pygamelib.gfx.core.Color property), property), 279 616 screen\_row(pygamelib.board\_items.BoardComplexItem

814 Index

property), 290

erty), 310

screen\_row (pygamelib.board\_items.BoardItem prop-

screen\_row(pygamelib.board items.BoardItemComplexComponent

```
property), 299
                                                                property), 705
                (pygamelib.board items.BoardItemVoid
                                                       screen_row (pygamelib.gfx.particles.Particle property),
screen_row
         property), 319
                                                                721
screen_row (pygamelib.board_items.Camera property),
                                                                       (pygamelib.gfx.particles.ParticleEmitter
                                                       screen_row
         328
                                                                property), 713
screen_row (pygamelib.board items.Character prop-
                                                                       (pygamelib.gfx.particles.ParticleSprixel
                                                       screen_row
         ertv), 338
                                                                property), 730
screen_row
                 (pygamelib.board items.ComplexDoor
                                                       screen_row
                                                                      (pygamelib.gfx.particles.PartitionParticle
         property), 349
                                                                property), 737
screen_row
                 (pygamelib.board_items.ComplexNPC
                                                       screen\_row(pygamelib.gfx.particles.RandomColorParticle)
         property), 360
                                                                property), 743
                (pygamelib.board_items.ComplexPlayer
                                                       screen_row(pygamelib.gfx.particles.RandomColorPartitionParticle
screen_row
                                                                property), 748
         property), 371
screen_row (pygamelib.board_items.ComplexTreasure
                                                       screen_row (pygamelib.gfx.ui.BoxLayout property), 623
                                                       screen_row (pygamelib.gfx.ui.Cursor property), 631
         property), 382
                 (pygamelib.board_items.ComplexWall
                                                       screen_row (pygamelib.gfx.ui.FormLayout property),
screen_row
         property), 393
                                                                639
screen_row (pygamelib.board_items.Door property),
                                                       screen_row
                                                                     (pygamelib.gfx.ui.GridLayout property),
                                                                644
screen_row (pygamelib.board items.GenericActionableStrscmeen_row (pygamelib.gfx.ui.Layout property), 652
         property), 412
                                                       screen_row (pygamelib.gfx.ui.LineInput property), 660
screen_row (pygamelib.board items.GenericStructure
                                                       screen_row (pygamelib.gfx.ui.Widget property), 688
                                                       SCREWDRIVER (pygamelib.assets.graphics.Models at-
         property), 431
screen_row(pygamelib.board items.GenericStructureComplexComptoribentte), 214
                                                       SCROLL (pygamelib.assets.graphics.Models attribute),
         property), 421
screen_row (pygamelib.board_items.Immovable prop-
                                                                214
                                                       SEAL (pygamelib.assets.graphics.Models attribute), 214
         erty), 440
screen_row (pygamelib.board_items.Movable property),
                                                       search() (pygamelib.engine.Inventory method), 566
                                                       search_frame()
                                                                                (pygamelib.gfx.core.Animation
         450
              (pygamelib.board_items.NPC property),
                                                                method), 584
screen_row
         461
                                                       SEAT (pygamelib.assets.graphics.Models attribute), 214
screen_row (pygamelib.board_items.Player property),
                                                       {\tt SECOND\_PLACE\_MEDAL}\ (pygame lib. assets. graphics. Models
         470
                                                                attribute), 214
                                                                (pygamelib.assets.graphics.MiscTechnicals at-
screen_row (pygamelib.board_items.Projectile prop-
                                                       SECTOR
                                                                tribute), 91
         erty), 483
screen_row (pygamelib.board_items.TextItem property),
                                                       SEE_NO_EVIL_MONKEY (pygamelib.assets.graphics.Models
         494
                                                                attribute), 214
screen_row (pygamelib.board_items.Tile property), 504
                                                       SEEDLING (pygamelib.assets.graphics.Models attribute),
screen_row (pygamelib.board_items.Treasure property),
         515
                                                       SEGMENT (pygamelib.assets.graphics.MiscTechnicals at-
              (pygamelib.board items.Wall property),
screen_row
                                                                tribute), 91
         525
                                                       select_next() (pygamelib.gfx.ui.Menu method), 666
screen_row (pygamelib.engine.Board property), 543
                                                       select_next() (pygamelib.gfx.ui.MenuBar method),
screen_row (pygamelib.engine.Game property), 557
                                                                672
                                                       select_previous() (pygamelib.gfx.ui.Menu method),
screen_row (pygamelib.engine.Inventory property), 566
screen_row (pygamelib.engine.Screen property), 579
                                                                666
screen_row (pygamelib.gfx.core.Color property), 617
                                                       select_previous()
                                                                                   (pygamelib.gfx.ui.MenuBar
screen_row (pygamelib.gfx.core.Sprite property), 600
                                                                method), 672
screen\_row\ (pygamelib.gfx.core.Sprixel\ property),\ 610
                                                       selected (pygamelib.gfx.ui.Menu property), 666
                 (pygamelib.gfx.particles.CircleEmitter
                                                       selected (pygamelib.gfx.ui.MenuAction property), 668
screen_row
                                                       selection (pygamelib.gfx.ui.ColorPicker property), 627
         property), 693
screen_row
                 (pygamelib.gfx.particles.ColorParticle
                                                       SELFIE (pygamelib.assets.graphics.Models attribute),
         property), 699
                                                                214
screen_row(pygamelib.gfx.particles.ColorPartitionParticlerialize() (pygamelib.actuators.Actuator method),
```

serialize() (pygamelib.actuators. PathActuator method), 17  serialize() (pygamelib.actuators. PathActuator method), 17  serialize() (pygamelib.actuators. PathActuator method), 19  serialize() (pygamelib.actuators. PathActuator method), 29  serialize() (pygamelib.actuators. RandomActuator method), 39  serialize() (pygamelib.actuators. UnidirectionalActuator method), 31  serialize() (pygamelib.board_items. Complex (pygamelib.board_items. Prosertialize() (pygamelib.board_items. Soardlomboard_items. Soardlomboard_items. Soardlomboard_items. Soardlomboard_items. Complex (pygamelib.board_items. Complex (p	10	461
method), 17 serialize() (pygamelib.acutators.PathFinder method), 29 serialize() (pygamelib.acutators.RandomAcutator method), 31 serialize() (pygamelib.acutators.RandomAcutator method), 34 serialize() (pygamelib.acutators.RandomAcutator method), 37 serialize() (pygamelib.acutators.UnidirectionalAcutator method), 37 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Vector2D method), 259 serialize() (pygamelib.base.Vector2D method), 259 serialize() (pygamelib.board_items.ActionableTile method), 269 serialize() (pygamelib.board_items.ActionableTile method), 269 serialize() (pygamelib.board_items.BoardItemSourdItems.BoardItemSourdIte		* * * * * * * * * * * * * * * * * * * *
serialize() (pygamelib.actuators.PatrolActuator method), 31 serialize() (pygamelib.actuators.CandomActuator method), 34 serialize() (pygamelib.actuators.UnidirectionalActuator method), 37 serialize() (pygamelib.base.Text method), 255 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Text method), 269 serialize() (pygamelib.board_items.Actionable Tile method), 310 serialize() (pygamelib.board_items.BoardItemVonlextor method), 310 serialize() (pygamelib.board_items.BoardItemVonlextor method), 310 serialize() (pygamelib.board_items.BoardItemVonlextor method), 310 serialize() (pygamelib.board_items.Candera method), 329 serialize() (pygamelib.board_items.Candera method), 339 serialize() (pygamelib.board_items.Complex/PIOvor method), 360 serialize() (pygamelib.board_items.Complex/PIOvor method), 360 serialize() (pygamelib.board_items.Complex/PIOvor method), 360 serialize() (pygamelib.board_items.Complex/PIOvor method), 371 serialize() (pygamelib.board_items.Complex/PIOvor method), 383 serialize() (pygamelib.board_items.Complex/PIOvor method), 371 serialize() (pygamelib.board_items.Complex/PIOvor method), 371 serialize() (pygamelib.board_items.Complex/PIOvor method), 371 serialize() (pygamelib.board_items.Complex/PIOvor method), 373 serialize() (pygamelib.board_i	***	***
method), 31 serialize() (pygamelib.acutators.RandomActutator method), 34 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.board_items.Actionable method), 269 serialize() (pygamelib.board_items.ActionableTite method), 279 serialize() (pygamelib.board_items.RoandComplexItem method), 290 serialize() (pygamelib.board_items.BoardItemComplexItem method), 310 serialize() (pygamelib.board_items.BoardItemComplex(Samphalaze()) (pygamelib.board_items.BoardItemComplex(Samphalaze()) (pygamelib.board_items.BoardItemComplex(Samphalaze()) (pygamelib.board_items.BoardItemComplex(Samphalaze()) (pygamelib.board_items.BoardItemComplex(Samphalaze()) (pygamelib.board_items.Complex()) serialize() (pygamelib.board_items.Complex() serialize() (pygamelib.gfx.particles.Partic		
serialize() (pygamelib.base. Text method), 37 serialize() (pygamelib.base. Text method), 255 serialize() (pygamelib.base. Text method), 255 serialize() (pygamelib.base. Vector 2D method), 259 serialize() (pygamelib.base. Vector 2D method), 259 serialize() (pygamelib.board_items. Actionable method), 269 serialize() (pygamelib.board_items. BoardComplextem method), 310 serialize() (pygamelib.board_items. BoardItem complextom method), 310 serialize() (pygamelib.board_items. Complex Nor method), 339 serialize() (pygamelib.board_items. Complex Nor method), 339 serialize() (pygamelib.board_items. Complex Nor method), 330 serialize() (pygamelib.board_items. Complex Nor method), 331 serialize() (pygamelib.board_items. Complex Nor method), 332 serialize() (pygamelib.board_items. Complex Nor method), 333 serialize() (pygamelib.board_items. Complex Nor method), 334 serialize() (pygamelib.board_items. Complex Nor method), 335 serialize() (pygamelib.board_items. GenericStructure method), 335 serialize() (pygamelib.board_items. GenericStructure method), 435 serialize() (pygamelib.board_items.	***	
method), 37 serialize() (pygamelib.base.Text method), 259 serialize() (pygamelib.base.Vector2D method), 259 serialize() (pygamelib.base.Vector2D method), 259 serialize() (pygamelib.board_items.Actionable method), 269 serialize() (pygamelib.board_items.ActionableTile method), 290 serialize() (pygamelib.board_items.BoardComplexItem method), 310 serialize() (pygamelib.board_items.BoardItem Complex (pygamelib.board_items.BoardItem Complex (pygamelib.board_items.BoardItem Complex (pygamelib.board_items.BoardItem Complex (pygamelib.board_items.BoardItem Complex (pygamelib.board_items.Complex (pygamelib.board_items.Complex) serialize() (pygamelib.board_items.ComplexNPC method), 339 serialize() (pygamelib.board_items.ComplexNPC method), 332 serialize() (pygamelib.board_items.ComplexNPC method), 334 serialize() (pygamelib.board_items.ComplexNPC method), 335 serialize() (pygamelib.board_items.ComplexNPC method), 336 serialize() (pygamelib.board_items.ComplexNPC method), 331 serialize() (pygamelib.board_items.ComplexNPC method), 332 serialize() (pygamelib.board_items.ComplexNPC method), 333 serialize() (pygamelib.board_items.ComplexNPC method), 332 serialize() (pygamelib.board_items.ComplexNPC method), 333 serialize() (pygamelib.board_items.ComplexNPC method), 334 serialize() (pygamelib.board_items.ComplexNPC method), 335 serialize() (pygamelib.board_items.ComplexNPC method), 331 serialize() (pygamelib.board_items.ComplexNPC method), 333 serialize() (pygamelib.board_items.ComplexNPC method), 334 serialize() (pygamelib.board_items.ComplexNPC method), 334 serialize() (pygamelib.board_items.ComplexNPC method), 334 serialize() (pygamelib.board_items.ComplexNPC method), 334 seria		
serialize() (pygamelib.base.Vector2D method), 259 serialize() (pygamelib.base.Vector2D method), 269 serialize() (pygamelib.board_items.Actionable method), 269 serialize() (pygamelib.board_items.BoardComplextnem method), 279 serialize() (pygamelib.board_items.BoardItem method), 300 serialize() (pygamelib.board_items.BoardItemComplex() (pygamelib.gfx.core.Sprite method), 610 serialize() (pygamelib.board_items.BoardItemComplex() (pygamelib.gfx.core.Spritemethod), 610 serialize() (pygamelib.board_items.BoardItemComplex() (pygamelib.gfx.particles.ColorParticlemethod), 310 serialize() (pygamelib.board_items.BoardItemVoid method), 339 serialize() (pygamelib.board_items.ComplexNPC method), 360 serialize() (pygamelib.board_items.ComplexNPC method), 339 serialize() (pygamelib.board_items.ComplexTreasure method), 339 serialize() (pygamelib.board_items.ComplexTreasure method), 339 serialize() (pygamelib.board_items.ComplexXever.Animation method), 610 serialize() (pygamelib.gfx.core.Sprite method), 610 serialize() (pygamelib.gfx.particles.ColorParticlemethod), 630 serialize() (pygamelib.board_items.ComplexNPC method), 339 serialize() (pygamelib.board_items.ComplexNPC method), 360 serialize() (pygamelib.board_items.ComplexTreasure method), 360 serialize() (pygamelib.board_items.ComplexTreasure method), 370 serialize() (pygamelib.board_items.ComplexXever.Animation method), 260 serialize() (pygamelib.gfx.core.Sprite method), 610 serialize() (pygamelib.gfx.particles.ColorPartiticlemethod), 639 serialize() (pygamelib.board_items.ComplexNPC method), 330 serialize() (pygamelib.board_items.ComplexVertemethod), 340 serialize() (pygamelib.board_items.ComplexXevertemethod), 340 serialize() (pygamelib.board_items.ComplexVertemethod), 340 serialize() (pygamelib.board_items.ComplexVertemethod), 340 serialize() (pygamelib.board_items.ComplexXevertemethod), 340 serialize() (pygamelib.board_items.ComplexXevertemethod), 340 serialize() (pygamelib.board_items.ComplexXevertemethod), 340 serialize() (pygamelib.gfx.particles.ColorPartitionPartic		
serialize() (pygamelib.board_items.Actionable method), 269 serialize() (pygamelib.board_items.Actionable method), 279 serialize() (pygamelib.board_items.BoardComplexttem method), 310 serialize() (pygamelib.board_items.BoardItemComplex() (pygamelib.board_items.BoardItemComplex() (pygamelib.board_items.BoardItemComplex() (pygamelib.gfx.core.Sprite method), 610 serialize() (pygamelib.board_items.BoardItemComplex() (pygamelib.gfx.core.Sprite method), 610 serialize() (pygamelib.board_items.BoardItemComplex() (pygamelib.gfx.particles.ColorParticlemethod), 693 serialize() (pygamelib.board_items.BoardItemComplex() (pygamelib.gfx.particles.ColorParticlemethod), 693 serialize() (pygamelib.board_items.ComplexDoor method), 319 serialize() (pygamelib.board_items.ComplexNPC method), 360 serialize() (pygamelib.board_items.ComplexNPC method), 360 serialize() (pygamelib.board_items.ComplexNPare method), 360 serialize() (pygamelib.board_items.ComplexNPare method), 371 serialize() (pygamelib.board_items.ComplexNPare method), 373 serialize() (pygamelib.board_items.ComplexNeaulemethod), 382 serialize() (pygamelib.board_items.ComplexNeaulemethod), 383 serialize() (pygamelib.board_items.ComplexNeaulemethod), 383 serialize() (pygamelib.board_items.ComplexNeaulemethod), 384 serialize() (pygamelib.board_items.ComplexNeaulemethod), 385 serialize() (pygamelib.board_items.ComplexNeaulemethod), 385 serialize() (pygamelib.board_items.ComplexNeaulemethod), 403 serialize() (pygamelib.board_items.GenericAtionableSissettiatem.move() (pygamelib.board_items.Actionable method), 413 serialize() (pygamelib.board_items.GenericStructure method), 421 serialize() (pygamelib.board_items.Movable method), 440 serialize() (pygamelib.board_items.Movable method), 430 serialize() (pygamelib.board_items.Movable method), 430 serialize() (pygamelib.board_items.BoardItemComplexComponent method), 431 serialize() (pygamelib.board_items.BoardItemComplexComponent method), 331		
serialize() (pygamelib.board_items.Actionable method), 269  serialize() (pygamelib.board_items.BoardLomplextlem method), 279  serialize() (pygamelib.board_items.BoardLomplextlem method), 290  serialize() (pygamelib.board_items.BoardItem method), 310  serialize() (pygamelib.board_items.BoardItem method), 310  serialize() (pygamelib.board_items.BoardItem method), 310  serialize() (pygamelib.board_items.BoardItem method), 319  serialize() (pygamelib.board_items.Camera method), 329  serialize() (pygamelib.board_items.Camera method), 339  serialize() (pygamelib.board_items.ComplexDeor method), 360  serialize() (pygamelib.board_items.ComplexPlayer method), 393  serialize() (pygamelib.board_items.ComplexPlayer method), 393  serialize() (pygamelib.board_items.ComplexPlayer method), 393  serialize() (pygamelib.board_items.ComplexPlayer method), 341  serialize() (pygamelib.board_items.GenericStructure method), 412  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.Simmovable method), 440  serialize() (pygamelib.board_items.Simmovable method), 440  serialize() (pygamelib.board_items.Simmovable method), 440  serialize() (pygamelib.board_items.SomplexMall method), 440  serialize() (pygamelib.gfx.		
serialize() (pygamelib.board_items.ActionableTite method), 279 serialize() (pygamelib.board_items.BoardComplexItem method), 290 serialize() (pygamelib.board_items.BoardItem method), 310 serialize() (pygamelib.board_items.BoardItems.Complex.Board_items.BoardItems.Complex.Board_items.BoardItems.Board_items.BoardItems.Board_items.BoardItems.Board_items.BoardItems.Board_items.Board	serialize() (pygamelib.base.Vector2D method), 259	serialize() (pygamelib.gfx.core.Animation method),
serialize() (pygamelib.board_items.ActionableTile method), 279 serialize() (pygamelib.board_items.BoardComplexItem method), 310 serialize() (pygamelib.board_items.BoardItem method), 310 serialize() (pygamelib.board_items.BoardItem method), 310 serialize() (pygamelib.board_items.BoardItemComplex\tangle serialize() (pygamelib.gx.particles.CircleEmitter method), 310 serialize() (pygamelib.board_items.BoardItemVoid method), 319 serialize() (pygamelib.board_items.Camera method), 329 serialize() (pygamelib.board_items.Camera method), 339 serialize() (pygamelib.board_items.Camera method), 349 serialize() (pygamelib.board_items.ComplexDoor method), 340 serialize() (pygamelib.board_items.ComplexPlaver method), 360 serialize() (pygamelib.board_items.ComplexPlaver method), 371 serialize() (pygamelib.board_items.ComplexPlaver method), 382 serialize() (pygamelib.board_items.ComplexPlaver method), 383 serialize() (pygamelib.board_items.ComplexPlaver method), 383 serialize() (pygamelib.board_items.ComplexPlaver method), 381 serialize() (pygamelib.board_items.ComplexPlaver method), 382 serialize() (pygamelib.board_items.ComplexPlaver method), 383 serialize() (pygamelib.board_items.ComplexWall method), 383 serialize() (pygamelib.board_items.GenericStructure method), 384 serialize() (pygamelib.board_items.GenericStructure method), 385 serialize() (pygamelib.board_items.GenericStructure method), 385 serialize() (pygamelib.board_items.GenericStructure) method), 385 serialize()	serialize() (pygamelib.board_items.Actionable	584
method), 279 serialize() (pygamelib.board_items.BoardComplextern method), 290 serialize() (pygamelib.board_items.BoardItem method), 310 serialize() (pygamelib.board_items.BoardItem method), 310 serialize() (pygamelib.board_items.BoardItemComplexCamplexDec method), 310 serialize() (pygamelib.board_items.BoardItemComplexCamplexDec method), 310 serialize() (pygamelib.board_items.BoardItemComplexCamplexDec method), 310 serialize() (pygamelib.board_items.BoardItemComplexCamplexDec method), 310 serialize() (pygamelib.board_items.CamplexDec method), 310 serialize() (pygamelib.board_items.CamplexDec method), 310 serialize() (pygamelib.board_items.CamplexDec method), 339 serialize() (pygamelib.board_items.ComplexDec method), 339 serialize() (pygamelib.board_items.ComplexPlayer method), 311 serialize() (pygamelib.board_items.ComplexTreasure method), 331 serialize() (pygamelib.board_items.ComplexTreasure method), 332 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexVall method), 331 serialize() (pygamelib.board_items.ComplexVall method), 331 serialize() (pygamelib.board_items.ComplexWall method), 331 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexVall method), 331 serialize() (pygamelib.board_items.ComplexVall method), 331 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexVall method), 331 serialize() (pygamelib.board_items.ComplexVall method), 332 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexWall method), 331 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexWall method), 331 serialize() (pygamelib.board_items.ComplexWall method), 333 serialize() (pygamelib.board_items.ComplexWall	method), 269	serialize() (pygamelib.gfx.core.Color method), 617
serialize() (pygamelib.board_items.BoardComplexItem method), 310 serialize() (pygamelib.board_items.BoardItemComplexComplexItem method), 310 serialize() (pygamelib.board_items.BoardItemComplexComplexItem method), 319 serialize() (pygamelib.board_items.BoardItemComplexComplexItem method), 319 serialize() (pygamelib.board_items.Comera method), 329 serialize() (pygamelib.board_items.ComplexDoor method), 339 serialize() (pygamelib.board_items.ComplexNPC method), 330 serialize() (pygamelib.board_items.ComplexNPC method), 310 serialize() (pygamelib.board_items.ComplexNPC method), 330 serialize() (pygamelib.board_items.ComplexWall method), 343 serialize() (pygamelib.board_items.ComplexWall method), 360 serialize() (pygamelib.board	<pre>serialize() (pygamelib.board_items.ActionableTile</pre>	serialize() (pygamelib.gfx.core.Sprite method), 600
serialize() (pygamelib.gfx.core.Sprixel method), 610 serialize() (pygamelib.board_items.Boardltem method), 310 serialize() (pygamelib.board_items.BoardltemComplexCompethod), 599 serialize() (pygamelib.board_items.BoardltemVoid method), 319 serialize() (pygamelib.board_items.Camera method), 339 serialize() (pygamelib.board_items.ComplexNPComethod), 360 serialize() (pygamelib.board_items.ComplexPlayer method), 382 serialize() (pygamelib.board_items.ComplexPlayer method), 382 serialize() (pygamelib.board_items.ComplexWall method), 393 serialize() (pygamelib.board_items.ComplexWall method), 393 serialize() (pygamelib.board_items.ComplexWall method), 393 serialize() (pygamelib.board_items.ComplexWall method), 393 serialize() (pygamelib.board_items.ComplexWall method), 413 serialize() (pygamelib.board_items.GenericActionableSsetucan_move() (pygamelib.engine.Game method), 558 serialize() (pygamelib.board_items.GenericStructure method), 421 serialize() (pygamelib.board_items.Movable method), 451 serialize() (pygamelib.board_items.Boardltems.Movable method), 451	method), 279	serialize() (pygamelib.gfx.core.SpriteCollection
serialize() (pygamelib.board_items.BoardItem method), 310	<pre>serialize() (pygamelib.board_items.BoardComplexItem</pre>	method), 592
serialize() (pygamelib.board_items.BoardItem method), 310		
method), 310  serialize() (pygamelib.board_items.BoardItemComplex(sun)aidate() (pygamelib.gfx.particles.ColorParticle method), 599  serialize() (pygamelib.board_items.BoardItemVoid method), 319  serialize() (pygamelib.board_items.Camera method), 329  serialize() (pygamelib.board_items.Character method), 339  serialize() (pygamelib.board_items.ComplexDoor method), 349  serialize() (pygamelib.board_items.ComplexNPC method), 360  serialize() (pygamelib.board_items.ComplexPlayer method), 371  serialize() (pygamelib.board_items.ComplexTreasure method), 382  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.ComplexWall method), 413  serialize() (pygamelib.board_items.GenericActionableStruture method), 421  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.Immovable method), 451  serialize() (pygamelib.board_items.BoardItems.Movable method), 300  serialize() (pygamelib.board_items.BoardItems.Movable method), 300	serialize() (pygamelib.board items.BoardItem	
serialize() (pygamelib.board_items.BoardItemVoid method), 299 serialize() (pygamelib.board_items.BoardItemVoid method), 319 serialize() (pygamelib.board_items.Camera method), 705 serialize() (pygamelib.board_items.Camera method), 705 serialize() (pygamelib.board_items.Camera method), 705 serialize() (pygamelib.board_items.Character method), 339 serialize() (pygamelib.board_items.ComplexDoor method), 349 serialize() (pygamelib.board_items.ComplexNPC method), 360 serialize() (pygamelib.board_items.ComplexPlayer method), 371 serialize() (pygamelib.board_items.ComplexPlayer method), 382 serialize() (pygamelib.board_items.ComplexWall serialize() (pygamelib.gfx.particles.RandomColorParticle method), 393 serialize() (pygamelib.board_items.ComplexWall serialize() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 393 serialize() (pygamelib.board_items.ComplexWall serialize() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 349 serialize() (pygamelib.board_items.GenericActionableStaturen method), 413 serialize() (pygamelib.board_items.GenericStructure method), 431 serialize() (pygamelib.board_items.GenericStructure set_can_move() (pygamelib.board_items.ActionableTile method), 240 serialize() (pygamelib.board_items.GenericStructureComplexCamponer() (pygamelib.board_items.BoardItem method), 410 set_can_move() (pygamelib.board_items.BoardItem method), 310 serialize() (pygamelib.board_items.GenericStructureComplexCamponer() (pygamelib.board_items.BoardItem method), 310 serialize() (pygamelib.board_items.GenericStructureComplexCamponer() (pygamelib.board_items.BoardItem method), 311 set_can_move() (pygamelib.board_items.BoardItemComplexComponent method), 300	***	
serialize() (pygamelib.board_items.BoardItemVoid method), 319 serialize() (pygamelib.board_items.Camera method), 329 serialize() (pygamelib.board_items.Character method), 339 serialize() (pygamelib.board_items.ComplexDoor method), 349 serialize() (pygamelib.board_items.ComplexPlayer method), 371 serialize() (pygamelib.board_items.ComplexPlayer method), 381 serialize() (pygamelib.board_items.ComplexPlayer method), 382 serialize() (pygamelib.board_items.ComplexPlayer method), 383 serialize() (pygamelib.board_items.ComplexPlayer method), 382 serialize() (pygamelib.board_items.ComplexPlayer method), 383 serialize() (pygamelib.board_items.ComplexPlayer method), 383 serialize() (pygamelib.board_items.ComplexWall serialize() (pygamelib.gfx.particles.PartitionParticle method), 743 serialize() (pygamelib.board_items.ComplexWall serialize() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 743 serialize() (pygamelib.board_items.GenericActionableStacturan_move() (pygamelib.gfx.particles.RandomColorPartitionParticle method), 743 serialize() (pygamelib.board_items.GenericStructure method), 269 serialize() (pygamelib.board_items.GenericStructure method), 413 serialize() (pygamelib.board_items.GenericStructure method), 421 serialize() (pygamelib.board_items.GenericStructureComplexComponent method), 441 serialize() (pygamelib.board_items.GenericStructureComplexComponent method), 451 serialize() (pygamelib.board_items.BoardItemComplexComponent method), 451	serialize()(pygamelib.board_items.BoardItemComplex	(csanpadare() (pygamelib.gfx.particles.ColorParticle
serialize() (pygamelib.board_items.Camera method), 329 serialize() (pygamelib.board_items.Character method), 339 serialize() (pygamelib.board_items.ComplexDoor method), 349 serialize() (pygamelib.board_items.ComplexNPC method), 360 serialize() (pygamelib.board_items.ComplexPlayer method), 371 serialize() (pygamelib.board_items.ComplexTreasure method), 382 serialize() (pygamelib.board_items.ComplexWall method), 393 serialize() (pygamelib.board_items.ComplexWall method), 393 serialize() (pygamelib.board_items.ComplexWall method), 413 serialize() (pygamelib.board_items.GenericActionableStructure method), 413 serialize() (pygamelib.board_items.GenericStructure method), 421 serialize() (pygamelib.board_items.GenericStructure method), 421 serialize() (pygamelib.board_items.Movable method), 451 serialize() (pygamelib.board_items.Movable method), 451 serialize() (pygamelib.board_items.BoardItemComplexComponent method), 451 serialize() (pygamelib.board_items.BoardItemComplexComponent method), 300	serialize() (pygamelib.board_items.BoardItemVoid	${\tt serialize()} \ (pygame lib. gfx. particles. Color Partition Particle$
serialize() (pygamelib.board_items.Character method), 339  serialize() (pygamelib.board_items.ComplexDoor method), 349  serialize() (pygamelib.board_items.ComplexNPC method), 360  serialize() (pygamelib.board_items.ComplexPlayer method), 371  serialize() (pygamelib.board_items.ComplexPlayer method), 382  serialize() (pygamelib.board_items.ComplexTreasure method), 382  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.Door method), 557  serialize() (pygamelib.board_items.GenericActionable/Statuture method), 413  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.GenericStructure tems.det), 421  serialize() (pygamelib.board_items.Immovable method), 451  serialize() (pygamelib.board_items.Movable method), 451  serialize() (pygamelib.board_items.Movable method), 451  serialize() (pygamelib.board_items.BoardItemComplexComponent method), 451	<pre>serialize() (pygamelib.board_items.Camera method),</pre>	serialize() (pygamelib.gfx.particles.EmitterProperties
method), 349  serialize() (pygamelib.board_items.ComplexNPC method), 360  serialize() (pygamelib.board_items.ComplexPlayer method), 371  serialize() (pygamelib.board_items.ComplexPlayer method), 371  serialize() (pygamelib.board_items.ComplexTreasure method), 382  serialize() (pygamelib.board_items.ComplexTreasure method), 393  serialize() (pygamelib.board_items.ComplexWall method), 749  serialize() (pygamelib.board_items.Door method), 403  serialize() (pygamelib.board_items.GenericActionableSsettucan_move() (pygamelib.engine.Game method), 557  serialize() (pygamelib.board_items.GenericStructure method), 431  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.Immovable method), 440  serialize() (pygamelib.board_items.Immovable method), 451  serialize() (pygamelib.board_items.Movable method), 451  serialize() (pygamelib.board_items.BoardItemComplexComponent method), 300		serialize() (pygamelib.gfx.particles.Particle method),
method), 360  serialize() (pygamelib.board_items.ComplexPlayer method), 371  serialize() (pygamelib.board_items.ComplexPlayer method), 371  serialize() (pygamelib.board_items.ComplexTreasure method), 382  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.Door method), 403  serialize() (pygamelib.board_items.GenericStructure method), 413  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.GenericStructure method), 420  serialize() (pygamelib.board_items.Immovable method), 440  serialize() (pygamelib.board_items.Movable method), 451	***	
method), 371  serialize() (pygamelib.board_items.ComplexTreasure method), 382  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.ComplexWall method), 393  serialize() (pygamelib.board_items.Door method), 403  serialize() (pygamelib.board_items.GenericActionableStrecture method), 413  serialize() (pygamelib.board_items.GenericStructure method), 431  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.Immovable method), 440  serialize() (pygamelib.board_items.Movable method), 451  serialize() (pygamelib.board_items.Movable method), 300	***	
method), 382 serialize() (pygamelib.board_items.ComplexWall method), 393 serialize() (pygamelib.board_items.Door method), 403 serialize() (pygamelib.board_items.GenericActionableStructure method), 413 serialize() (pygamelib.board_items.GenericStructure method), 431 serialize() (pygamelib.board_items.GenericStructureComplexCannmove() (pygamelib.board_items.ActionableTile method), 421 serialize() (pygamelib.board_items.Immovable method), 440 serialize() (pygamelib.board_items.Movable method), 451 serialize() (pygamelib.board_items.Movable method), 451 serialize() (pygamelib.board_items.Movable method), 300		
method), 393  serialize() (pygamelib.board_items.Door method), 403  serialize() (pygamelib.board_items.GenericActionableStructure method), 413  serialize() (pygamelib.board_items.GenericStructure method), 431  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.GenericStructure method), 421  serialize() (pygamelib.board_items.Immovable method), 440  serialize() (pygamelib.board_items.Movable method), 451  serialize() (pygamelib.board_items.Movable method), 451  serialize() (pygamelib.board_items.Movable method), 451  serialize() (pygamelib.board_items.Movable method), 300		
session_logs() (pygamelib.engine.Game method), 558  serialize() (pygamelib.board_items.GenericActionableStructuren_move() (pygamelib.board_items.Actionable method), 413	1	
serialize() (pygamelib.board_items.GenericActionableStracturen_move() (pygamelib.board_items.Actionable method), 413		
serialize() (pygamelib.board_items.GenericStructure method), 431 serialize() (pygamelib.board_items.GenericStructureComptexCampmoner(t) (pygamelib.board_items.BoardComplexItem method), 421 method), 290 serialize() (pygamelib.board_items.Immovable method), 440 serialize() (pygamelib.board_items.Movable method), 311 serialize() (pygamelib.board_items.Movable method), 451 set_can_move() (pygamelib.board_items.BoardItemComplexComponent method), 300	serialize() (pygamelib.board_items.GenericActionable	Structucen_move() (pygamelib.board_items.Actionable
serialize() (pygamelib.board_items.GenericStructureComptexComptone(f) (pygamelib.board_items.BoardComplexItem method), 421 method), 290  serialize() (pygamelib.board_items.Immovable method), 440 set_can_move() (pygamelib.board_items.BoardItem method), 311  serialize() (pygamelib.board_items.Movable method), 451 set_can_move() (pygamelib.board_items.BoardItemComplexComponent method), 300	${\tt serialize()}\ (pygame lib. board\_items. Generic Structure$	set_can_move() (pygamelib.board_items.ActionableTile
serialize() (pygamelib.board_items.Immovable set_can_move() (pygamelib.board_items.BoardItem method), 311 serialize() (pygamelib.board_items.Movable set_can_move() (pygamelib.board_items.BoardItemComplexComponent method), 451 serialize() (pygamelib.board_items.BoardItemComplexComponent method), 300	${\tt serialize()}\ (pygame lib.board\_items. Generic Structure College Structure Coll$	o <b>nptexConnponer(</b> t) (pygamelib.board_items.BoardComplexItem
serialize() (pygamelib.board_items.Movable set_can_move() (pygamelib.board_items.BoardItemComplexComponent method), 451 set_can_move() (pygamelib.board_items.BoardItemComplexComponent method), 300	serialize() (pygamelib.board_items.Immovable	set_can_move() (pygamelib.board_items.BoardItem
	serialize() (pygamelib.board_items.Movable	$\verb set_can_move()  (pygame lib.board\_items. Board Item Complex Component) $

method), 319	method), 311
set_can_move() (pygamelib.board_items.Camera method), 329	<pre>set_overlappable() (pygamelib.board_items.BoardItemComplexCompo method), 300</pre>
	set_overlappable()(pygamelib.board_items.BoardItemVoid
method), 339	method), 320
	set_overlappable() (pygamelib.board_items.Camera
method), 349	method), 329
<i>"</i>	set_overlappable()(pygamelib.board_items.Character
method), 360	method), 339
	set_overlappable()(pygamelib.board_items.ComplexDoor
method), 371	method), 349
	urset_overlappable()(pygamelib.board_items.ComplexNPC
method), 382	method), 360
<pre>set_can_move() (pygamelib.board_items.ComplexWall</pre>	<pre>set_overlappable() (pygamelib.board_items.ComplexPlayer</pre>
method), 393	method), 371
set_can_move() (pygamelib.board_items.Door	<pre>set_overlappable() (pygamelib.board_items.ComplexTreasure</pre>
method), 404	method), 382
set_can_move() (pygamelib.board_items.GenericActiona	ub <b>sextrovere</b> appable() (pygamelib.board_items.ComplexWall
method), 413	method), 393
set_can_move() (pygamelib.board_items.GenericStructu	rset_overlappable() (pygamelib.board_items.Door
method), 432	method), 404
set_can_move() (pygamelib.board_items.GenericStructu	resetunphenclappabelet() (pygamelib.board_items.GenericActionableStructur
method), 422	method), 413
set_can_move() (pygamelib.board_items.Immovable	<pre>set_overlappable() (pygamelib.board_items.GenericStructure</pre>
method), 441	method), 432
set_can_move() (pygamelib.board_items.Movable	<pre>set_overlappable() (pygamelib.board_items.GenericStructureComplex)</pre>
method), 451	method), 422
set_can_move() (pygamelib.board_items.NPC	<pre>set_overlappable() (pygamelib.board_items.Immovable</pre>
method), 461	method), 441
set_can_move() (pygamelib.board_items.Player	set_overlappable()(pygamelib.board_items.Movable
method), 471	method), 451
set_can_move() (pygamelib.board_items.Projectile	set_overlappable() (pygamelib.board_items.NPC
method), 483	method), 461
set_can_move() (pygamelib.board_items.TextItem	set_overlappable() (pygamelib.board_items.Player
method), 494	method), 471
set_can_move() (pygamelib.board_items.Tile method),	<pre>set_overlappable() (pygamelib.board_items.Projectile</pre>
505	method), 484
	set_overlappable() (pygamelib.board_items.TextItem
method), 516	method), 494
set_can_move() (pygamelib.board_items.Wall method),	set_overlappable() (pygamelib.board_items.Tile
526	method), 505
set_color() (pygamelib.gfx.ui.ColorPickerDialog	set_overlappable() (pygamelib.board_items.Treasure
method), 625	method), 516
set_destination() (pygamelib.actuators.PathFinder	set_overlappable() (pygamelib.board_items.Wall
method), 29	method), 526
set_direction() (pygamelib.board_items.Projectile	set_path() (pygamelib.actuators.PathActuator
method), 483	method), 17
set_overlappable() (pygamelib.board_items.Actionable	
method), 269	method), 22
set_overlappable() (pygamelib.board_items.Actionable	
method), 279	method), 269
	mpslex[tpinckable() (pygamelib.board_items.ActionableTile
method), 290	method), 279 a set_pickable() (pygamelih hogrd_items RogrdComplexItem
SEL OVECTADDADTELLIOVO(MØIIN NO/IYO 110MS KOOYAITON	

```
method), 290
                                                                                                 method), 311
                              (pygamelib.board_items.BoardItem set_restorable() (pygamelib.board_items.BoardItemComplexComponent
set_pickable()
             method), 311
                                                                                                 method), 300
set_pickable() (pygamelib.board_items.BoardItemComptexCareparanthle() (pygamelib.board_items.BoardItemVoid
             method), 300
                                                                                                 method), 320
set_pickable() (pygamelib.board items.BoardItemVoid set_restorable()
                                                                                                                     (pygamelib.board items.Camera
             method), 320
                                                                                                 method), 329
set_pickable()
                                  (pygamelib.board_items.Camera set_restorable() (pygamelib.board_items.Character
             method), 329
                                                                                                 method), 339
set_pickable()
                               (pygamelib.board_items.Character
                                                                                   set_restorable() (pygamelib.board_items.ComplexDoor
             method), 339
                                                                                                 method), 349
set_pickable() (pygamelib.board_items.ComplexDoor set_restorable() (pygamelib.board_items.ComplexNPC
             method), 349
                                                                                                 method), 361
                                                                                   set_restorable() (pygamelib.board_items.ComplexPlayer
set_pickable() (pygamelib.board_items.ComplexNPC
             method), 361
                                                                                                 method), 371
set_pickable() (pygamelib.board_items.ComplexPlayer set_restorable() (pygamelib.board_items.ComplexTreasure
             method), 371
                                                                                                 method), 383
set_pickable() (pygamelib.board_items.ComplexTreasurset_restorable() (pygamelib.board_items.ComplexWall
             method), 382
                                                                                                 method), 393
set_pickable() (pygamelib.board_items.ComplexWall
                                                                                   set_restorable()
                                                                                                                         (pygamelib.board items.Door
             method), 393
                                                                                                 method), 404
set_pickable()
                                      (pygamelib.board_items.Door
                                                                                  set_restorable() (pygamelib.board_items.GenericActionableStructure
             method), 404
                                                                                                 method), 413
set_pickable() (pygamelib.board items.GenericActionable() (pygamelib.board items.GenericStructure
             method), 413
                                                                                                 method), 432
set_pickable() (pygamelib.board_items.GenericStructureset_restorable() (pygamelib.board_items.GenericStructureComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComplexComp
             method), 432
                                                                                                 method), 422
set_pickable() (pygamelib.board_items.GenericStructures@mphestCamphae()t (pygamelib.board_items.Immovable
                                                                                                 method), 441
             method), 422
                                                                                   set_restorable()
set_pickable()
                              (pygamelib.board_items.Immovable
                                                                                                                    (pygamelib.board_items.Movable
              method), 441
                                                                                                 method), 451
set_pickable()
                                 (pygamelib.board_items.Movable
                                                                                   set_restorable()
                                                                                                                          (pygamelib.board_items.NPC
             method), 451
                                                                                                 method), 462
set_pickable()
                                      (pygamelib.board_items.NPC
                                                                                   set_restorable()
                                                                                                                       (pygamelib.board_items.Player
             method), 461
                                                                                                 method), 471
                                    (pygamelib.board items.Player
set_pickable()
                                                                                   set_restorable() (pygamelib.board_items.Projectile
             method), 471
                                                                                                 method), 484
set_pickable()
                               (pygamelib.board_items.Projectile
                                                                                   set_restorable()
                                                                                                                    (pygamelib.board_items.TextItem
             method), 484
                                                                                                 method), 494
set_pickable()
                                 (pygamelib.board_items.TextItem
                                                                                   set_restorable()
                                                                                                                           (pygamelib.board_items.Tile
             method), 494
                                                                                                 method), 505
set_pickable() (pygamelib.board items.Tile method),
                                                                                   set_restorable()
                                                                                                                    (pygamelib.board_items.Treasure
              505
                                                                                                 method), 516
set_pickable()
                                 (pygamelib.board_items.Treasure
                                                                                   set_restorable()
                                                                                                                          (pygamelib.board_items.Wall
             method), 516
                                                                                                 method), 526
set_pickable() (pygamelib.board_items.Wall method),
                                                                                   set_selection() (pygamelib.gfx.ui.ColorPickerDialog
                                                                                                 method), 625
                                                                                   set_sprixel() (pygamelib.gfx.core.Sprite method), 600
set_restorable() (pygamelib.board_items.Actionable
             method), 269
                                                                                   set_transparency()
                                                                                                                              (pygamelib.gfx.core.Sprite
set_restorable() (pygamelib.board_items.ActionableTile
                                                                                                 method), 600
             method), 279
                                                                                   setdefault()
                                                                                                                (pygamelib.gfx.core.SpriteCollection
set_restorable() (pygamelib.board_items.BoardComplexItem
                                                                                                 method), 592
             method), 290
                                                                                   SEVEN_OCLOCK
                                                                                                                  (pygamelib.assets.graphics.Models
set_restorable() (pygamelib.board items.BoardItem
                                                                                                 attribute), 214
```

- SEVEN\_THIRTY (pygamelib.assets.graphics.Models attribute), 214
- SEWING\_NEEDLE (pygamelib.assets.graphics.Models attribute), 214
- SHALLOW\_PAN\_OF\_FOOD
  - (pygamelib.assets.graphics.Models attribute), 214
- SHAMROCK (pygamelib.assets.graphics.Models attribute), 214
- SHARK (pygamelib.assets.graphics.Models attribute), 214 SHAVED\_ICE (pygamelib.assets.graphics.Models attribute), 214 tribute), 214
- SHEAF\_OF\_RICE (pygamelib.assets.graphics.Models attribute), 214
- SHIELD (pygamelib.assets.graphics.Models attribute), 214
- SHINTO\_SHRINE (pygamelib.assets.graphics.Models attribute), 214
- SHIP (pygamelib.assets.graphics.Models attribute), 214
- SHOOTING\_STAR (pygamelib.assets.graphics.Models attribute), 214
- SHOPPING\_BAGS (pygamelib.assets.graphics.Models attribute), 214
- SHOPPING\_CART (pygamelib.assets.graphics.Models attribute), 214
- SHORTCAKE (pygamelib.assets.graphics.Models attribute), 214
- SHORTS (pygamelib.assets.graphics.Models attribute), 214
- SHOULDERED\_OPEN\_BOX
  - (pygamelib.assets.graphics.MiscTechnicals attribute), 91
- show() (pygamelib.gfx.ui.ColorPickerDialog method),
  625
- show() (pygamelib.gfx.ui.Dialog method), 632
- show() (pygamelib.gfx.ui.FileDialog method), 635
- show() (pygamelib.gfx.ui.GridSelectorDialog method), 646
- show() (pygamelib.gfx.ui.LineInputDialog method), 662
- show() (pygamelib.gfx.ui.MessageDialog method), 675
- show() (pygamelib.gfx.ui.MultiLineInputDialog method), 677
- show () (pygamelib.gfx.ui.ProgressDialog method), 682
- show\_hidden\_files (pygamelib.gfx.ui.FileDialog property), 635
- SHOWER (pygamelib.assets.graphics.Models attribute), 215
- SHRIMP (pygamelib.assets.graphics.Models attribute), 215
- SHUFFLE\_TRACKS\_BUTTON
  - (pygamelib.assets.graphics.Models attribute), 215
- SHUSHING\_FACE (pygamelib.assets.graphics.Models attribute), 215

- SIGN\_OF\_THE\_HORNS (pygamelib.assets.graphics.Models attribute), 215
- SIX\_OCLOCK (pygamelib.assets.graphics.Models attribute), 215
- SIX\_THIRTY (pygamelib.assets.graphics.Models attribute), 215
- ${\tt size}\ (pygamelib.board\_items. Actionable\ property),\ 269$
- $\begin{array}{c} {\tt size} \ (pygamelib.board\_items. Actionable Tile \ property), \\ 280 \end{array}$
- size (pygamelib.board\_items.BoardComplexItem property), 291
- size (pygamelib.board\_items.BoardItem property), 311
- $\verb|size| (pygamelib.board\_items.BoardItemComplexComponent| \\ property), 300$
- size (pygamelib.board\_items.Camera property), 329
- size (pygamelib.board\_items.Character property), 339
- size (pygamelib.board\_items.ComplexNPC property),
  361
- size (pygamelib.board\_items.ComplexPlayer property), 371
- size (pygamelib.board\_items.ComplexTreasure property), 383
- size (pygamelib.board\_items.ComplexWall property), 394
- size (pygamelib.board\_items.Door property), 404
- size (pygamelib.board\_items.GenericActionableStructure property), 413
- size (pygamelib.board\_items.GenericStructure property), 432
- size (pygamelib.board\_items.GenericStructureComplexComponent property), 422
- size (pygamelib.board\_items.Immovable property), 441
- size (pygamelib.board\_items.Movable property), 451
- ${\tt size}\ (pygamelib.board\_items.NPC\ property), 462$
- size (pygamelib.board\_items.Player property), 471
- size (pygamelib.board\_items.Projectile property), 484
- size (pygamelib.board items. TextItem property), 495
- size (pygamelib.board\_items.Tile property), 505
- $\verb|size| (pygamelib.board\_items. Treasure| property), 516$
- size (pygamelib.board\_items.Wall property), 526
- size() (pygamelib.engine.Inventory method), 567
- size\_constraint (pygamelib.gfx.ui.BoxLayout property), 623
- size\_constraint (pygamelib.gfx.ui.LineInput property), 660
- size\_constraint (pygamelib.gfx.ui.Widget property),
  688
- SKATEBOARD (pygamelib.assets.graphics.Models attribute), 215
- SKIER (pygamelib.assets.graphics.Models attribute), 215

SKIS (pygamelib.assets.graphics.Models attribute), 215 SKULL (pygamelib.assets.graphics.Models attribute), 215 SKULL_AND_CROSSBONES	SMILING_FACE_WITH_TEAR (pygamelib.assets.graphics.Models attribute), 216
(pygamelib.assets.graphics.Models attribute), 215	SMIRKING_FACE (pygamelib.assets.graphics.Models attribute), 216
SKUNK (pygamelib.assets.graphics.Models attribute), 215 SLED (pygamelib.assets.graphics.Models attribute), 215 SLEEPING_FACE (pygamelib.assets.graphics.Models attribute), 215	SNAIL (pygamelib.assets.graphics.Models attribute), 216 SNAKE (pygamelib.assets.graphics.Models attribute), 216 SNEEZING_FACE (pygamelib.assets.graphics.Models attribute), 216
SLEEPY_FACE (pygamelib.assets.graphics.Models attribute), 215	SNOW_CAPPED_MOUNTAIN  (pygamelib.assets.graphics.Models attribute),
SLIGHTLY_FROWNING_FACE	216
(pygamelib.assets.graphics.Models attribute), 215	SNOWBOARDER (pygamelib.assets.graphics.Models attribute), 216
SLIGHTLY_SMILING_FACE (pygamelib.assets.graphics.Models attribute),	SNOWFLAKE (pygamelib.assets.graphics.Models attribute), 216
215 SLOPE (pygamelib.assets.graphics.MiscTechnicals at-	SNOWMAN (pygamelib.assets.graphics.Models attribute), 216
tribute), 91	SNOWMAN_WITHOUT_SNOW
SLOT_MACHINE (pygamelib.assets.graphics.Models attribute), 215	(pygamelib.assets.graphics.Models attribute), 216
SLOTH (pygamelib.assets.graphics.Models attribute), 215 SMALL_AIRPLANE (pygamelib.assets.graphics.Models attribute), 215	SOAP (pygamelib.assets.graphics.Models attribute), 216 SOCCER_BALL (pygamelib.assets.graphics.Models at- tribute), 216
${\tt SMALL\_BLUE\_DIAMOND}\ (pygamelib. assets. graphics. Models$	SOCKS (pygamelib.assets.graphics.Models attribute), 216
attribute), 215 SMALL_ORANGE_DIAMOND	SOFT_ICE_CREAM (pygamelib.assets.graphics.Models attribute), 216
(pygamelib.assets.graphics.Models attribute), 215	${\tt SOFTBALL}\ (pygamelib. assets. graphics. Models\ attribute),\\ 216$
SMILE (pygamelib.assets.graphics.MiscTechnicals attribute), 91	SOFTWARE_FUNCTION_SYMBOL (pygamelib.assets.graphics.MiscTechnicals
SMILING_CAT_WITH_HEART_EYES	attribute), 91
(pygamelib.assets.graphics.Models attribute), 215	SOON_ARROW (pygamelib.assets.graphics.Models attribute), 216
SMILING_FACE (pygamelib.assets.graphics.Models attribute), 215	SOS_BUTTON (pygamelib.assets.graphics.Models attribute), 216
SMILING_FACE_WITH_HALO	spacing (pygamelib.gfx.ui.BoxLayout property), 623
(pygamelib.assets.graphics.Models attribute), 215	spacing (pygamelib.gfx.ui.FormLayout property), 639 spacing (pygamelib.gfx.ui.GridLayout property), 644
SMILING_FACE_WITH_HEART_EYES	spacing (pygamelib.gfx.ui.Layout property), 653
(pygamelib.assets.graphics.Models attribute), 215	spacing (pygamelib.gfx.ui.MenuBar property), 672 SPADE_SUIT (pygamelib.assets.graphics.Models at-
SMILING_FACE_WITH_HEARTS	tribute), 216
(pygamelib.assets.graphics.Models attribute), 215	SPAGHETTI (pygamelib.assets.graphics.Models attribute), 216
SMILING_FACE_WITH_HORNS	${\tt SPARKLE}\ (pygamelib. assets. graphics. Models\ attribute),$
(pygamelib.assets.graphics.Models attribute), 215	216 SPARKLER (pygamelib.assets.graphics.Models attribute),
SMILING_FACE_WITH_SMILING_EYES	216
(pygamelib.assets.graphics.Models attribute), 215	SPARKLES (pygamelib.assets.graphics.Models attribute), 216
SMILING_FACE_WITH_SUNGLASSES	SPARKLING_HEART (pygamelib.assets.graphics.Models
(pygamelib.assets.graphics.Models attribute), 215	attribute), 216 SPEAK_NO_EVIL_MONKEY

- (pygamelib.assets.graphics.Models attribute), 216
- SPEAKER\_HIGH\_VOLUME

(pygamelib.assets.graphics.Models attribute),

- attribute), 216
- SPEAKER\_MEDIUM\_VOLUME

(pygamelib.assets.graphics.Models attribute), 216

- SPEAKING\_HEAD (pygamelib.assets.graphics.Models attribute), 216
- SPEECH\_BALLOON (pygamelib.assets.graphics.Models attribute), 216
- **SPEEDBOAT** (pygamelib.assets.graphics.Models attribute), 216
- SPIDER (pygamelib.assets.graphics.Models attribute), 216
- (pygamelib.assets.graphics.Models SPIDER\_WEB tribute), 217
- ${\tt SPIRAL\_CALENDAR} \quad (pygamelib. assets. graphics. Models$ attribute), 217
- SPIRAL\_NOTEPAD (pygamelib.assets.graphics.Models attribute), 217
- SPIRAL\_SHELL (pygamelib.assets.graphics.Models attribute), 217
- SPONGE (pygamelib.assets.graphics.Models attribute),
- SPOON (pygamelib.assets.graphics.Models attribute), 217 SPORT\_UTILITY\_VEHICLE

(pygamelib.assets.graphics.Models attribute), 217

- SPORTS\_MEDAL (pygamelib.assets.graphics.Models attribute), 217
- SPOUTING\_WHALE (pygamelib.assets.graphics.Models attribute), 217
- Sprite (class in pygamelib.gfx.core), 592
- sprite (pygamelib.board\_items.ActionableTile property), 280
- (pygamelib.board\_items.BoardComplexItem sprite property), 291
- (pygamelib.board\_items.ComplexDoor propsprite erty), 350
- sprite (pygamelib.board\_items.ComplexNPC property),
- sprite (pygamelib.board\_items.ComplexPlayer property), 372
- sprite (pygamelib.board\_items.ComplexTreasure property), 383
- sprite (pygamelib.board\_items.ComplexWall property),
- sprite (pygamelib.board\_items.TextItem property), 495 sprite (pygamelib.board items. Tile property), 506
- SpriteCollection (class in pygamelib.gfx.core), 588

- Sprixel (class in pygamelib.gfx.core), 602
- sprixel (pygamelib.gfx.ui.Cursor property), 631
- sprixel() (pygamelib.gfx.core.Sprite method), 601
- ${\tt SQUARE\_FOOT}\ (pygamelib. assets. graphics. Misc Technicals$ attribute), 91
- SPEAKER\_LOW\_VOLUME (pygamelib.assets.graphics.Models SQUARE\_LOZENGE (pygamelib.assets.graphics.MiscTechnicals attribute), 91
  - SQUARE\_WITH\_DIAGONAL\_CROSSHATCH\_FILL

(pygamelib.assets.graphics.GeometricShapes attribute), 65

SQUARE\_WITH\_HORIZONTAL\_FILL

(pygamelib.assets.graphics.GeometricShapes attribute), 65

- SQUARE\_WITH\_LEFT\_HALF\_BLACK (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUARE\_WITH\_LOWER\_RIGHT\_DIAGONAL\_HALF\_BLACK (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUARE\_WITH\_ORTHOGONAL\_CROSSHATCH\_FILL (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUARE\_WITH\_RIGHT\_HALF\_BLACK (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUARE\_WITH\_UPPER\_LEFT\_DIAGONAL\_HALF\_BLACK (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUARE\_WITH\_UPPER\_LEFT\_TO\_LOWER\_RIGHT\_FILL (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUARE\_WITH\_UPPER\_RIGHT\_TO\_LOWER\_LEFT\_FILL (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUARE\_WITH\_VERTICAL\_FILL (pygamelib.assets.graphics.GeometricShapes attribute), 65
- SQUID (pygamelib.assets.graphics.Models attribute), 217 SQUINTING\_FACE\_WITH\_TONGUE

(pygamelib.assets.graphics.Models attribute),

- STADIUM (pygamelib.assets.graphics.Models attribute),
- STAR (pygamelib.assets.graphics.Models attribute), 217 STAR\_AND\_CRESCENT (pygamelib.assets.graphics.Models
- attribute), 217 STAR\_OF\_DAVID (pygamelib.assets.graphics.Models at-
- tribute), 217
- STAR\_STRUCK (pygamelib.assets.graphics.Models attribute), 217
- start() (pygamelib.actuators.Actuator method), 10
- start() (pygamelib.actuators.Behavioral method), 14
- start() (pygamelib.actuators.PathActuator method), 18
- start() (pygamelib.actuators.PathFinder method), 30

start() (pygamelib.actuators.PatrolActuator method),	method), 350
22	store_position() (pygamelib.board_items.ComplexNPC
start() (pygamelib.actuators.RandomActuator	method), 362
method), 34	store_position() (pygamelib.board_items.ComplexPlayer
start() (pygamelib.actuators.UnidirectionalActuator	method), 372
method), 38	store_position() (pygamelib.board_items.ComplexTreasure
start() (pygamelib.engine.Game method), 558	method), 383
start() (pygamelib.gfx.core.Animation method), 584	store_position() (pygamelib.board_items.ComplexWall
state (pygamelib.engine.Game property), 558	method), 394
STATION (pygamelib.assets.graphics.Models attribute), 217	store_position() (pygamelib.board_items.Door method), 405
STATUE_OF_LIBERTY (pygamelib.assets.graphics.Models attribute), 217	store_position() (pygamelib.board_items.GenericActionableStructure method), 414
STEAMING_BOWL (pygamelib.assets.graphics.Models at-	<pre>store_position() (pygamelib.board_items.GenericStructure</pre>
tribute), 217	method), 433
STETHOSCOPE (pygamelib.assets.graphics.Models at-	<pre>store_position() (pygamelib.board_items.GenericStructureComplexCo.</pre>
tribute), 217	method), 423
stop() (pygamelib.actuators.Actuator method), 10	<pre>store_position() (pygamelib.board_items.Immovable</pre>
stop() (pygamelib.actuators.Behavioral method), 14	method), 442
stop() (pygamelib.actuators.PathActuator method), 18	<pre>store_position() (pygamelib.board_items.Movable</pre>
stop() (pygamelib.actuators.PathFinder method), 30	method), 452
stop() (pygamelib.actuators.PatrolActuator method), 22	<pre>store_position() (pygamelib.board_items.NPC</pre>
<pre>stop() (pygamelib.actuators.RandomActuator method),</pre>	method), 462
34	store_position() (pygamelib.board_items.Player
stop() (pygamelib.actuators.UnidirectionalActuator	method), 472
method), 38	<pre>store_position() (pygamelib.board_items.Projectile</pre>
stop() (pygamelib.engine.Game method), 558	method), 485
stop() (pygamelib.gfx.core.Animation method), 585	<pre>store_position() (pygamelib.board_items.TextItem</pre>
STOP_BUTTON (pygamelib.assets.graphics.Models at-	method), 495
tribute), 217	<pre>store_position() (pygamelib.board_items.Tile</pre>
STOP_SIGN (pygamelib.assets.graphics.Models at-	method), 506
tribute), 217	<pre>store_position() (pygamelib.board_items.Treasure</pre>
STOPPED (pygamelib.constants.State attribute), 530	method), 517
STOPWATCH (pygamelib.assets.graphics.MiscTechnicals	store_position() (pygamelib.board_items.Wall
attribute), 91	method), 527
STOPWATCH (pygamelib.assets.graphics.Models at-	<pre>store_screen_position()</pre>
tribute), 217	(pygamelib.actuators.Actuator method), 10
<pre>store_position() (pygamelib.board_items.Actionable</pre>	store_screen_position()
method), 270	(pygamelib.actuators.Behavioral method),
$store\_position()$ (pygamelib.board_items.ActionableTi	$e^{ile}$ 14
method), 280	store_screen_position()
${\tt store\_position()}\ (pygamelib.board\_items.BoardComplete )$	
method), 291	store_screen_position()
store_position() (pygamelib.board_items.BoardItem	(pygamelib.actuators.PathFinder method),
method), 312	30
$store\_position()$ (pygamelib.board_items.BoardItemCo	
method), 301	(pygamelib.actuators.PatrolActuator method),
$store\_position()$ (pygamelib.board_items.BoardItemValue)	
method), 320	store_screen_position()
store_position() (pygamelib.board_items.Camera	(pygamelib.actuators.RandomActuator
method), 330	method), 34
store_position() (pygamelib.board_items.Character	-
method), 340	(pygamelib.actuators.UnidirectionalActuator
store position() (pygamelih hoard items ComplexDoc	or method), 38

<pre>store_screen_position()</pre>	method), 423
	store_screen_position()
247	(pygamelib.board_items.Immovable method),
<pre>store_screen_position() (pygamelib.base.Text</pre>	442
method), 256	store_screen_position()
<pre>store_screen_position()</pre>	(pygamelib.board_items.Movable method),
(pygamelib.board_items.Actionable method),	452
270	store_screen_position()
<pre>store_screen_position()</pre>	(pygamelib.board_items.NPC method), 462
$(pygame lib.board\_items. Actionable Tile$	<pre>store_screen_position()</pre>
method), 281	(pygamelib.board_items.Player method),
store_screen_position()	472
(pygamelib.board_items.BoardComplexItem	store_screen_position()
method), 292	(pygamelib.board_items.Projectile method),
store_screen_position()	485
(pygamelib.board_items.BoardItem method),	store_screen_position()
312	(pygamelib.board_items.TextItem method),
<pre>store_screen_position()</pre>	496
(pygamelib.board_items.BoardItemComplexCom	ponome_screen_position()
method), 301	(pygamelib.board_items.Tile method), 506
<pre>store_screen_position()</pre>	<pre>store_screen_position()</pre>
$(pygamelib.board\_items.BoardItemVoid$	(pygamelib.board_items.Treasure method),
method), 321	517
<pre>store_screen_position()</pre>	store_screen_position()
(pygamelib.board_items.Camera method),	(pygamelib.board_items.Wall method), 527
330	store_screen_position() (pygamelib.engine.Board
<pre>store_screen_position()</pre>	method), 543
(pygamelib.board_items.Character method), 340	store_screen_position() (pygamelib.engine.Game method), 558
<pre>store_screen_position()</pre>	store_screen_position()
(pygamelib.board_items.ComplexDoor	(pygamelib.engine.Inventory method), 567
method), 351	store_screen_position() (pygamelib.engine.Screen
store_screen_position()	method), 579
(pygamelib.board_items.ComplexNPC	store_screen_position() (pygamelib.gfx.core.Color
method), 362	method), 617
store_screen_position()	store_screen_position() (pygamelib.gfx.core.Sprite
(pygamelib.board_items.ComplexPlayer	method), 601
method), 373	store_screen_position()
store_screen_position()	(pygamelib.gfx.core.Sprixel method), 610
(pygamelib.board_items.ComplexTreasure	store_screen_position()
method), 384	(pygamelib.gfx.particles.CircleEmitter
store_screen_position()	method), 694
(pygamelib.board_items.ComplexWall	store_screen_position()
method), 395	(pygamelib.gfx.particles.ColorParticle
store_screen_position()	method), 700
(pygamelib.board_items.Door method), 405	store_screen_position()
store_screen_position()	(pygamelib.gfx.particles.ColorPartitionParticle
(pygamelib.board_items.GenericActionableStruc	110
method), 414	store_screen_position()
store_screen_position()	(pygamelib.gfx.particles.Particle method),
(pygamelib.board_items.GenericStructure	722
method), 433	store_screen_position()
store_screen_position()	(pygamelib.gfx.particles.ParticleEmitter
(pygamelib.board items.GenericStructureComple	

<pre>store_screen_position()</pre>	tribute), 218
(pygamelib.gfx.particles.ParticleSprixel method), 730	SUNFLOWER (pygamelib.assets.graphics.Models attribute), 217
<pre>store_screen_position()</pre>	SUNGLASSES (pygamelib.assets.graphics.Models at-
(pygame lib. gfx. particles. Partition Particle	tribute), 217
method), 737	SUNRISE (pygamelib.assets.graphics.Models attribute),
store_screen_position()	217
(pygamelib.gfx.particles.RandomColorParticle	SUNRISE_OVER_MOUNTAINS
method), 743	(pygamelib.assets.graphics.Models attribute),
store_screen_position()  (prograph of particles PandomColorPartitionP	218  SHINGET (magnetic agents are marking Models, attribute)
(pygamelib.gfx.particles.RandomColorPartitionP method), 749	ASTINGET (pygamelib.assets.graphics.Models attribute), 218
store_screen_position()	SUPERHERO (pygamelib.assets.graphics.Models at-
(pygamelib.gfx.ui.BoxLayout method), 623	tribute), 218
store_screen_position() (pygamelib.gfx.ui.Cursor	SUPERVILLAIN (pygamelib.assets.graphics.Models
method), 631	attribute), 218
store_screen_position()	SUSHI (pygamelib.assets.graphics.Models attribute), 218
(pygamelib.gfx.ui.FormLayout method), 639	SUSPENSION_RAILWAY (pygamelib.assets.graphics.Models
store_screen_position()	attribute), 218
(pygamelib.gfx.ui.GridLayout method), 644	SWAN (pygamelib.assets.graphics.Models attribute), 218
<pre>store_screen_position() (pygamelib.gfx.ui.Layout</pre>	SWEAT_DROPLETS (pygamelib.assets.graphics.Models at-
method), 653	tribute), 218
store_screen_position()	SYMMETRY (pygamelib.assets.graphics.MiscTechnicals at-
(pygamelib.gfx.ui.LineInput method), 660	tribute), 91
store_screen_position() (pygamelib.gfx.ui.Widget	SYNAGOGUE (pygamelib.assets.graphics.Models at-
method), 688	tribute), 218
STRAIGHT_RULER (pygamelib.assets.graphics.Models at-	SYRINGE (pygamelib.assets.graphics.Models attribute),
tribute), 217	218
STRAIGHTNESS (pygamelib.assets.graphics.MiscTechnicals attribute), 91	T
STRAWBERRY (pygamelib.assets.graphics.Models at-	T_REX (pygamelib.assets.graphics.Models attribute), 220
tribute), 217	T_SHIRT (pygamelib.assets.graphics.Models attribute),
STUDIO_MICROPHONE (pygamelib.assets.graphics.Models	220
attribute), 217	TACO (pygamelib.assets.graphics.Models attribute), 218
STUFFED_FLATBREAD (pygamelib.assets.graphics.Models	TAKEOUT_BOX (pygamelib.assets.graphics.Models at-
attribute), 217	tribute), 218
style (pygamelib.base.Text attribute), 256	TAMALE (pygamelib.assets.graphics.Models attribute),
${\tt SUMMATION\_BOTTOM}\ (pygame lib. assets. graphics. Misc Technology and the property of the$	icals 218
attribute), 91	${\tt TANABATA\_TREE} \ (pygamelib. assets. graphics. Models \ at-$
${\tt SUMMATION\_TOP}\ (pygame lib. assets. graphics. Misc Technical Summation of the property o$	<i>"</i>
attribute), 91	TANGERINE (pygamelib.assets.graphics.Models at-
SUN (pygamelib.assets.graphics.Models attribute), 217	tribute), 218
SUN_BEHIND_CLOUD (pygamelib.assets.graphics.Models	TAURUS (pygamelib.assets.graphics.Models attribute),
attribute), 218 SUN_BEHIND_LARGE_CLOUD	218
(pygamelib.assets.graphics.Models attribute),	TAXI (pygamelib.assets.graphics.Models attribute), 218 TEACUP_WITHOUT_HANDLE
218	(pygamelib.assets.graphics.Models attribute),
SUN_BEHIND_RAIN_CLOUD	218
(pygamelib.assets.graphics.Models attribute),	TEAPOT (pygamelib.assets.graphics.Models attribute),
218	218
SUN_BEHIND_SMALL_CLOUD	TEAR_OFF_CALENDAR (pygamelib.assets.graphics.Models
(pygamelib.assets.graphics.Models attribute),	attribute), 218
218	TEDDY_BEAR (pygamelib.assets.graphics.Models at-
SUN WITH FACE (pygamelib assets graphics Models at-	tribute) 218

- TELEPHONE (pygamelib.assets.graphics.Models THUMBS\_UP (pygamelib.assets.graphics.Models tribute), 218
- TELEPHONE\_RECEIVER (pygamelib.assets.graphics.Models TICKET (pygamelib.assets.graphics.Models attribute), attribute), 218 219
- TELEPHONE\_RECORDER (pygamelib.assets.graphics.MiscTechtiGERs (pygamelib.assets.graphics.Models attribute), 219 attribute), 91 TIGER\_FACE (pygamelib.assets.graphics.Models
- **TELESCOPE** (pygamelib.assets.graphics.Models attribute), 218
- **TELEVISION** (pygamelib.assets.graphics.Models attribute), 218
- (pygamelib.assets.graphics.Models TEN\_OCLOCK attribute), 219
- TEN\_THIRTY (pygamelib.assets.graphics.Models attribute), 219
- TENNIS (pygamelib.assets.graphics.Models attribute), 218
- TENT (pygamelib.assets.graphics.Models attribute), 219
- terminate() (pygamelib.gfx.particles.ColorParticle *method*), 700
- terminate() (pygamelib.gfx.particles.ColorPartitionParticle method), 706
- terminate() (pygamelib.gfx.particles.Particle method), 722
- terminate() (pygamelib.gfx.particles.PartitionParticle method), 738
- terminate() (pygamelib.gfx.particles.RandomColorParticle *method*), 743
- TEST\_TUBE (pygamelib.assets.graphics.Models tribute), 219
- Text (class in pygamelib.base), 249
- text (pygamelib.base.Text property), 256
- text (pygamelib.board\_items.TextItem property), 496
- text (pygamelib.gfx.ui.LineInput property), 660
- TextItem (class in pygamelib.board items), 486
- THERMOMETER (pygamelib.assets.graphics.Models tribute), 219
- THINKING\_FACE (pygamelib.assets.graphics.Models attribute), 219
- THIRD\_PLACE\_MEDAL (pygamelib.assets.graphics.Models attribute), 219
- THONG\_SANDAL (pygamelib.assets.graphics.Models attribute), 219
- THOUGHT\_BALLOON (pygamelib.assets.graphics.Models attribute), 219
- THREAD (pygamelib.assets.graphics.Models attribute), 219
- THREE\_OCLOCK (pygamelib.assets.graphics.Models attribute), 219
- THREE\_THIRTY (pygamelib.assets.graphics.Models attribute), 219
- THUMBS\_DOWN (pygamelib.assets.graphics.Models attribute), 219

- tribute), 219
- tribute), 219
- Tile (class in pygamelib.board\_items), 497
- TIMER\_CLOCK (pygamelib.assets.graphics.MiscTechnicals attribute), 91
- TIMER\_CLOCK (pygamelib.assets.graphics.Models tribute), 219
- tint() (pygamelib.gfx.core.Sprite method), 602
- TIRED\_FACE (pygamelib.assets.graphics.Models attribute), 219
- title (pygamelib.gfx.ui.Box property), 619
- title (pygamelib.gfx.ui.ColorPickerDialog property),
- title (pygamelib.gfx.ui.GridSelectorDialog property),
- title (pygamelib.gfx.ui.LineInputDialog property), 662
- title (pygamelib.gfx.ui.Menu property), 666
- title (pygamelib.gfx.ui.MenuAction property), 669
- title (pygamelib.gfx.ui.MessageDialog property), 675
- title (pygamelib.gfx.ui.MultiLineInputDialog property), 678
- title\_width() (pygamelib.gfx.ui.Menu method), 666
- terminate() (pygamelib.gfx.particles.RandomColorPartitivinPhetistivdth() (pygamelib.gfx.ui.MenuAction method), *method*), 749 669
  - to\_json\_file() (pygamelib.gfx.core.SpriteCollection *method*), 592
    - toggle\_active() (pygamelib.gfx.particles.CircleEmitter method), 694
    - toggle\_active() (pygamelib.gfx.particles.ParticleEmitter method), 714
    - TOILET (pygamelib.assets.graphics.Models attribute), 219
    - TOKYO\_TOWER (pygamelib.assets.graphics.Models attribute), 219
    - OTAMOT (pygamelib.assets.graphics.Models attribute),
    - TONGUE (pygamelib.assets.graphics.Models attribute),
    - TOOLBOX (pygamelib.assets.graphics.Models attribute),
    - TOOTH (pygamelib.assets.graphics.Models attribute), 219
    - TOOTHBRUSH (pygamelib.assets.graphics.Models tribute), 219
    - TOP (pygamelib.constants.Alignment attribute), 528
    - TOP\_ARROW (pygamelib.assets.graphics.Models attribute), 219
    - TOP\_CURLY\_BRACKET (pygamelib.assets.graphics.MiscTechnicals attribute), 91
    - TOP\_HALF\_INTEGRAL (pygamelib.assets.graphics.MiscTechnicals

attribute), 91	TUMBLER_GLASS (pygamelib.assets.graphics.Models at-
TOP_HAT (pygamelib.assets.graphics.Models attribute), 219	tribute), 220 TURKEY (pygamelib.assets.graphics.Models attribute),
TOP_LEFT_CORNER (pygamelib.assets.graphics.MiscTechnia	
attribute), 91	TURNED_NOT_SIGN (pygamelib.assets.graphics.MiscTechnicals
TOP_LEFT_CROP (pygamelib.assets.graphics.MiscTechnical	
attribute), 91	TURTLE (pygamelib.assets.graphics.Models attribute),
TOP_PARENTHESIS (pygamelib.assets.graphics.MiscTechnic attribute), 91	
auribuie), 91 TOP_RIGHT_CORNER (pygamelib.assets.graphics.MiscTechn	TWELVE_OCLOCK (pygamelib.assets.graphics.Models at- tribute), 220
attribute), 91	TWELVE_THIRTY (pygamelib.assets.graphics.Models at-
TOP_RIGHT_CROP (pygamelib.assets.graphics.MiscTechnical	
attribute), 91	TWO_HEARTS (pygamelib.assets.graphics.Models at-
TOP_SQUARE_BRACKET (pygamelib.assets.graphics.MiscTed	
attribute), 91	TWO_HUMP_CAMEL (pygamelib.assets.graphics.Models at-
TOP_TORTOISE_SHELL_BRACKET (pygamelib.assets.graphics.MiscTechnicals	tribute), 220 TWO_OCLOCK (pygamelib.assets.graphics.Models at-
attribute), 91	tribute), 220
TORNADO (pygamelib.assets.graphics.Models attribute),	TWO_THIRTY (pygamelib.assets.graphics.Models at-
219	tribute), 220
TOTAL_RUNOUT (pygamelib.assets.graphics.MiscTechnicals	11
attribute), 92	U
TRACKBALL (pygamelib.assets.graphics.Models at- tribute), 219	UiConfig (class in pygamelib.gfx.ui), 682
TRACTOR (pygamelib.assets.graphics.Models attribute),	UMBRELLA (pygamelib.assets.graphics.Models attribute), 220
219	UMBRELLA_ON_GROUND (pygamelib.assets.graphics.Models
TRADE_MARK (pygamelib.assets.graphics.Models at-	attribute), 220
tribute), 220	UMBRELLA_WITH_RAIN_DROPS
TRAIN (pygamelib.assets.graphics.Models attribute), 220	(pygamelib.assets.graphics.Models attribute),
TRAM (pygamelib.assets.graphics.Models attribute), 220	220
TRAM_CAR (pygamelib.assets.graphics.Models attribute), 220	UNAMUSED_FACE (pygamelib.assets.graphics.Models at- tribute), 220
${\tt TRANSGENDER\_SYMBOL}\ (pygamelib. assets. graphics. Models$	
attribute), 220	530
Treasure (class in pygamelib.board_items), 507	undo() (pygamelib.base.History method), 243
TRIANGULAR_FLAG (pygamelib.assets.graphics.Models attribute), 220	undo() (pygamelib.gfx.ui.LineInput method), 660
TRIANGULAR_RULER (pygamelib.assets.graphics.Models	UNDO_SYMBOL (pygamelib.assets.graphics.MiscTechnicals attribute), 92
attribute), 220	UNICORN (pygamelib.assets.graphics.Models attribute),
TRIDENT_EMBLEM (pygamelib.assets.graphics.Models at-	220
tribute), 220	UnidirectionalActuator (class in
trigger_rendering() (pygamelib.engine.Screen	pygamelib.actuators), 34
method), 580 TROLLEYBUS (pygamelib.assets.graphics.Models at-	unit() (pygamelib.base.Vector2D method), 259
tribute), 220	unlock_position() (pygamelib.gfx.ui.Cursor method), 631
TROPHY (pygamelib.assets.graphics.Models attribute),	UNLOCKED (pygamelib.assets.graphics.Models attribute),
220	220
TROPICAL_DRINK (pygamelib.assets.graphics.Models at-	UP (pygamelib.constants.Direction attribute), 528
tribute), 220	UP_ARROW (pygamelib.assets.graphics.Models attribute),
TROPICAL_FISH (pygamelib.assets.graphics.Models attribute), 220	221
TRUMPET (pygamelib.assets.graphics.Models attribute),	UP_ARROWHEAD (pygamelib.assets.graphics.MiscTechnicals attribute), 92
220	UP_ARROWHEAD_BETWEEN_TWO_HORIZONTAL_BARS
TULIP (pygamelib.assets.graphics.Models attribute), 220	(pygamelib.assets.graphics.MiscTechnicals

- attribute), 92
- UP\_BUTTON (pygamelib.assets.graphics.Models tribute), 221
- UP\_DOUBLE\_AND\_HORIZONTAL\_SINGLE

(pygamelib.assets.graphics.BoxDrawings attribute), 55

- UP\_DOUBLE\_AND\_LEFT\_SINGLE
  - (pygamelib.assets.graphics.BoxDrawings attribute), 55
- UP\_DOUBLE\_AND\_RIGHT\_SINGLE

(pygamelib.assets.graphics.BoxDrawings attribute), 55

- UP\_DOWN\_ARROW (pygamelib.assets.graphics.Models attribute), 221
- UP\_HEAVY\_AND\_DOWN\_HORIZONTAL\_LIGHT (pygamelib.assets.graphics.BoxDrawings attribute), 55
- UP\_HEAVY\_AND\_HORIZONTAL\_LIGHT (pygamelib.assets.graphics.BoxDrawings attribute), 55
- UP\_HEAVY\_AND\_LEFT\_DOWN\_LIGHT (pygamelib.assets.graphics.BoxDrawings attribute), 55
- UP\_HEAVY\_AND\_LEFT\_LIGHT
  (pygamelib.assets.graphics.BoxDrawings
  attribute), 55
- UP\_HEAVY\_AND\_RIGHT\_DOWN\_LIGHT (pygamelib.assets.graphics.BoxDrawings attribute), 55
- UP\_HEAVY\_AND\_RIGHT\_LIGHT (pygamelib.assets.graphics.BoxDrawings attribute), 55
- UP\_LEFT\_ARROW (pygamelib.assets.graphics.Models attribute), 221
- UP\_LIGHT\_AND\_DOWN\_HORIZONTAL\_HEAVY (pygamelib.assets.graphics.BoxDrawings attribute), 55
- UP\_LIGHT\_AND\_HORIZONTAL\_HEAVY (pygamelib.assets.graphics.BoxDrawings attribute), 56
- UP\_LIGHT\_AND\_LEFT\_DOWN\_HEAVY (pygamelib.assets.graphics.BoxDrawings attribute), 56
- UP\_LIGHT\_AND\_LEFT\_HEAVY

  (pygamelib.assets.graphics.BoxDrawings
  attribute), 56
- UP\_LIGHT\_AND\_RIGHT\_DOWN\_HEAVY
  (pygamelib.assets.graphics.BoxDrawings
  attribute), 56
- UP\_LIGHT\_AND\_RIGHT\_HEAVY

  (pygamelib.assets.graphics.BoxDrawings
  attribute), 56
- UP\_POINTING\_TRIANGLE\_WITH\_LEFT\_HALF\_BLACK (pygamelib.assets.graphics.GeometricShapes

- attribute), 65
- UP\_POINTING\_TRIANGLE\_WITH\_RIGHT\_HALF\_BLACK (pygamelib.assets.graphics.GeometricShapes attribute), 65
  - UP\_RIGHT\_ARROW (pygamelib.assets.graphics.Models attribute), 221
  - UP\_SINGLE\_AND\_HORIZONTAL\_DOUBLE (pygamelib.assets.graphics.BoxDrawings attribute), 56
  - UP\_SINGLE\_AND\_LEFT\_DOUBLE (pygamelib.assets.graphics.BoxDrawings attribute), 56
  - UP\_SINGLE\_AND\_RIGHT\_DOUBLE (pygamelib.assets.graphics.BoxDrawings attribute), 56

  - update() (pygamelib.gfx.particles.CircleEmitter method), 694
  - update() (pygamelib.gfx.particles.ColorParticle method), 700
  - update() (pygamelib.gfx.particles.ColorPartitionParticle method), 706
  - update() (pygamelib.gfx.particles.Particle method), 722 update() (pygamelib.gfx.particles.ParticleEmitter

method), 714

- update() (pygamelib.gfx.particles.PartitionParticle method), 738

- update\_sprite() (pygamelib.board\_items.TextItem method), 496
- UPPER\_HALF\_BLOCK (pygamelib.assets.graphics.Blocks attribute), 43
- UPPER\_HALF\_CIRCLE (pygamelib.assets.graphics.GeometricShapes

221

attribute), 65 UPPER_HALF_INVERSE_WHITE_CIRCLE	vcenter (pygamelib.engine.Screen property), 580
(pygamelib.assets.graphics.GeometricShapes	Vector2D (class in pygamelib.base), 257 VERTICAL (pygamelib.constants.Orientation attribute),
attribute), 65	529
UPPER_LEFT_OR_LOWER_RIGHT_CURLY_BRACKET_SECTI	OWNERTICAL DOUBLE AND HORIZONTAL SINGLE
(pygamelib.assets.graphics.MiscTechnicals at-	(pygamelib.assets.graphics.BoxDrawings
tribute), 92	attribute), 56
UPPER_LEFT_QUADRANT_CIRCULAR_ARC	VERTICAL_DOUBLE_AND_LEFT_SINGLE
(pygame lib. assets. graphics. Geometric Shapes	(pygamelib.assets.graphics.BoxDrawings
attribute), 65	attribute), 56
UPPER_LEFT_TRIANGLE	VERTICAL_DOUBLE_AND_RIGHT_SINGLE
(pygamelib.assets.graphics.GeometricShapes	(pygamelib.assets.graphics.BoxDrawings attribute), 56
attribute), 65 UPPER_ONE_EIGHTH_BLOCK	VERTICAL_HEAVY_AND_HORIZONTAL_LIGHT
(pygamelib.assets.graphics.Blocks attribute),	(pygamelib.assets.graphics.BoxDrawings
43	attribute), 56
UPPER_RIGHT_OR_LOWER_LEFT_CURLY_BRACKET_SECTI	
(pygamelib.assets.graphics.MiscTechnicals at-	(pygamelib.assets.graphics.BoxDrawings
tribute), 92	attribute), 56
UPPER_RIGHT_QUADRANT_CIRCULAR_ARC	VERTICAL_HEAVY_AND_RIGHT_LIGHT
(pygame lib. assets. graphics. Geometric Shapes	(pygamelib. assets. graphics. BoxDrawings
attribute), 65	attribute), 56
UPPER_RIGHT_TRIANGLE	VERTICAL_LIGHT_AND_HORIZONTAL_HEAVY
(pygamelib.assets.graphics.GeometricShapes	(pygamelib.assets.graphics.BoxDrawings
attribute), 65	attribute), 56 VERTICAL_LIGHT_AND_LEFT_HEAVY
UPSIDE_DOWN_FACE (pygamelib.assets.graphics.Models attribute), 221	(pygamelib.assets.graphics.BoxDrawings
UPWARDS_BUTTON (pygamelib.assets.graphics.Models at-	attribute), 56
tribute), 221	VERTICAL_LIGHT_AND_RIGHT_HEAVY
user_input (pygamelib.gfx.ui.ColorPickerDialog prop-	(pygamelib.assets.graphics.BoxDrawings
erty), 625	attribute), 56
user_input (pygamelib.gfx.ui.Dialog property), 632	VERTICAL_LINE_EXTENSION
<pre>user_input (pygamelib.gfx.ui.FileDialog property), 635</pre>	(pygame lib. assets. graphics. Misc Technicals
${\tt user\_input}~(\textit{pygamelib.gfx.ui.GridSelectorDialog~prop-}$	attribute), 92
erty), 647	VERTICAL_LINE_WITH_MIDDLE_DOT
user_input (pygamelib.gfx.ui.LineInputDialog prop-	(pygamelib.assets.graphics.MiscTechnicals
erty), 662	attribute), 92
user_input (pygamelib.gfx.ui.MessageDialog prop- erty), 675	VERTICAL_SINGLE_AND_HORIZONTAL_DOUBLE (pygamelib.assets.graphics.BoxDrawings
user_input (pygamelib.gfx.ui.MultiLineInputDialog	attribute), 56
property), 678	VERTICAL_SINGLE_AND_LEFT_DOUBLE
user_input (pygamelib.gfx.ui.ProgressDialog prop-	(pygamelib.assets.graphics.BoxDrawings
erty), 682	attribute), 56
•	VERTICAL_SINGLE_AND_RIGHT_DOUBLE
V	(pygamelib.assets.graphics.BoxDrawings
V_CENTER (pygamelib.constants.Alignment attribute),	attribute), 56
528	<pre>vertical_spacing (pygamelib.gfx.core.Font property),</pre>
value (pygamelib.gfx.ui.ProgressBar property), 680	588
value (pygamelib.gfx.ui.ProgressDialog property), 682	vertical_spacing (pygamelib.gfx.ui.FormLayout
value() (pygamelib.engine.Inventory method), 567	property), 640
values() (pygamelib.gfx.core.SpriteCollection method),	vertical_spacing (pygamelib.gfx.ui.GridLayout prop- erty), 644
VANDEDE (magazelih assets ananhias Madala attributa)	VERTICAL_TRAFFIC_LIGHT
VAMPIRE (pygamelib.assets.graphics.Models attribute),	·

828 Index

(pygamelib.assets.graphics.Models attribute),

221 WAVY\_DASH (pygamelib.assets.graphics.Models tribute), 222 VIBRATION\_MODE (pygamelib.assets.graphics.Models at-WAVY\_LINE (pygamelib.assets.graphics.MiscTechnicals tribute), 221 VICTORY\_HAND (pygamelib.assets.graphics.Models attribute), 92 attribute), 221 WAXING\_CRESCENT\_MOON VIDEO\_CAMERA (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Models attribute), attribute), 221 222 VIDEO\_GAME (pygamelib.assets.graphics.Models WAXING\_GIBBOUS\_MOON (pygamelib.assets.graphics.Models attribute), tribute), 221 VIDEOCASSETTE (pygamelib.assets.graphics.Models at-222 tribute), 221 WEARY\_CAT (pygamelib.assets.graphics.Models at-VIEWDATA\_SQUARE (pygamelib.assets.graphics.MiscTechnicals tribute), 222 attribute), 92 WEARY\_FACE (pygamelib.assets.graphics.Models attribute), 222 VIOLIN (pygamelib.assets.graphics.Models attribute), WEDDING (pygamelib.assets.graphics.Models attribute), VIRGO (pygamelib.assets.graphics.Models attribute), 221 222 VOLCANO (pygamelib.assets.graphics.Models attribute), WHALE (pygamelib.assets.graphics.Models attribute), 222 221 WHEEL\_OF\_DHARMA (pygamelib.assets.graphics.Models VOLLEYBALL (pygamelib.assets.graphics.Models atattribute), 222 WHEELCHAIR\_SYMBOL (pygamelib.assets.graphics.Models tribute), 221 VS\_BUTTON (pygamelib.assets.graphics.Models atattribute), 222 tribute), 221 white() (pygamelib.base.Text static method), 256 white\_bright() (pygamelib.base.Text static method), VULCAN\_SALUTE (pygamelib.assets.graphics.Models attribute), 221 WHITE\_BULLET (pygamelib.assets.graphics.GeometricShapes W attribute), 65 WHITE\_CANE (pygamelib.assets.graphics.Models WAFFLE (pygamelib.assets.graphics.Models attribute), tribute), 222 221 WHITE\_CIRCLE (pygamelib.assets.graphics.GeometricShapes Wall (class in pygamelib.board\_items), 518 attribute), 65 WANING\_CRESCENT\_MOON WHITE\_CIRCLE (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.Models attribute), attribute), 222 WHITE\_CIRCLE\_WITH\_LOWER\_LEFT\_QUADRANT WANING\_GIBBOUS\_MOON (pygamelib.assets.graphics.GeometricShapes (pygamelib.assets.graphics.Models attribute), attribute), 65 WHITE\_CIRCLE\_WITH\_LOWER\_RIGHT\_QUADRANT warn() (pygamelib.base.Text static method), 256 (pygamelib.assets.graphics.GeometricShapes WARNING (pygamelib.assets.graphics.Models attribute), attribute), 65 221 WHITE\_CIRCLE\_WITH\_UPPER\_LEFT\_QUADRANT WASTEBASKET (pygamelib.assets.graphics.Models (pygamelib.assets.graphics.GeometricShapes tribute), 221 attribute), 65 WATCH (pygamelib.assets.graphics.MiscTechnicals WHITE\_CIRCLE\_WITH\_UPPER\_RIGHT\_QUADRANT tribute), 92 (pygamelib.assets.graphics.GeometricShapes WATCH (pygamelib.assets.graphics.Models attribute), 221 attribute), 65 WATER\_BUFFALO (pygamelib.assets.graphics.Models at- ${\tt WHITE\_DIAMOND}$  (pygamelib.assets.graphics.GeometricShapes tribute), 221 attribute), 65 WATER\_CLOSET (pygamelib.assets.graphics.Models WHITE\_DIAMOND\_CONTAINING\_BLACK\_SMALL\_DIAMOND attribute), 221 (pygamelib.assets.graphics.GeometricShapes (pygame lib. assets. graphics. ModelsWATER\_WAVE atattribute), 66 tribute), 221 white\_dim() (pygamelib.base.Text static method), 256 WATERMELON (pygamelib.assets.graphics.Models at-WHITE\_DOWN\_POINTING\_SMALL\_TRIANGLE tribute), 221 (pygamelib.assets.graphics.GeometricShapes WAVING\_HAND (pygamelib.assets.graphics.Models at-

Index 829

tribute), 221

attribute), 66

WHITE\_DOWN\_POINTING\_TRIANGLE

(pygamelib.assets.graphics.GeometricShapes attribute), 66 WHITE_EXCLAMATION_MARK	WHITE_RIGHT_POINTING_TRIANGLE  (pygamelib.assets.graphics.GeometricShapes attribute), 66
(pygamelib.assets.graphics.Models attribute), 222	WHITE_SMALL_SQUARE (pygamelib.assets.graphics.GeometricShapes attribute), 66
WHITE_FLAG (pygamelib.assets.graphics.Models at- tribute), 222	WHITE_SMALL_SQUARE (pygamelib.assets.graphics.Models attribute), 222
WHITE_FLOWER (pygamelib.assets.graphics.Models attribute), 222	WHITE_SQUARE (pygamelib.assets.graphics.GeometricShapes attribute), 66
WHITE_HAIR (pygamelib.assets.graphics.Models at- tribute), 222	<pre>white_square() (pygamelib.gfx.core.Sprixel class     method), 611</pre>
WHITE_HEART (pygamelib.assets.graphics.Models attribute), 222	<pre>white_square() (pygamelib.gfx.particles.ParticleSprixel</pre>
$\verb WHITE_LARGE_SQUARE  (pygame lib. assets. graphics. Models $	WHITE_SQUARE_BUTTON
attribute), 222 WHITE_LEFT_POINTING_POINTER	(pygamelib.assets.graphics.Models attribute), 222
(pygame lib. assets. graphics. Geometric Shapes	WHITE_SQUARE_CONTAINING_BLACK_SMALL_SQUARE
attribute), 66	(pygame lib. assets. graphics. Geometric Shapes
WHITE_LEFT_POINTING_SMALL_TRIANGLE	attribute), 66
(pygamelib.assets.graphics.GeometricShapes	WHITE_SQUARE_WITH_CENTRE_VERTICAL_LINE
attribute), 66	(pygamelib.assets.graphics.MiscTechnicals
WHITE_LEFT_POINTING_TRIANGLE	attribute), 92
(pygamelib.assets.graphics.GeometricShapes	WHITE_SQUARE_WITH_LOWER_LEFT_QUADRANT
attribute), 66	(pygamelib.assets.graphics.GeometricShapes
WHITE_MEDIUM_SMALL_SQUARE	attribute), 66
(pygamelib.assets.graphics.GeometricShapes attribute), 66	WHITE_SQUARE_WITH_LOWER_RIGHT_QUADRANT (pygamelib.assets.graphics.GeometricShapes
WHITE_MEDIUM_SMALL_SQUARE	attribute), 66
(pygamelib.assets.graphics.Models attribute),	WHITE_SQUARE_WITH_ROUNDED_CORNERS
222	(pygamelib.assets.graphics.GeometricShapes
WHITE_MEDIUM_SQUARE	attribute), 66
(pygamelib.assets.graphics.GeometricShapes	WHITE_SQUARE_WITH_UPPER_LEFT_QUADRANT
attribute), 66	(pygamelib.assets.graphics.GeometricShapes
WHITE_MEDIUM_SQUARE	attribute), 66
(pygamelib.assets.graphics.Models attribute), 222	WHITE_SQUARE_WITH_UPPER_RIGHT_QUADRANT (pygamelib.assets.graphics.GeometricShapes
WHITE_PARALLELOGRAM	attribute), 66
(pygame lib. assets. graphics. Geometric Shapes	WHITE_SQUARE_WITH_VERTICAL_BISECTING_LINE
attribute), 66	(pygame lib. assets. graphics. Geometric Shapes
WHITE_QUESTION_MARK	attribute), 66
(pygamelib.assets.graphics.Models attribute), 222	WHITE_TRAPEZIUM (pygamelib.assets.graphics.MiscTechnicals attribute), 92
	WHITE_UP_POINTING_SMALL_TRIANGLE
method), 611	(pygamelib.assets.graphics.GeometricShapes
white_rect() (pygamelib.gfx.particles.ParticleSprixel	attribute), 66
class method), 731	WHITE_UP_POINTING_TRIANGLE
WHITE_RECTANGLE (pygamelib.assets.graphics.GeometricS	
attribute), 66	attribute), 66
WHITE_RIGHT_POINTING_POINTER  (prographic Geometric Shapes	WHITE_UP_POINTING_TRIANGLE_WITH_DOT  (programalib assets araphies GeometricShapes)
(pygamelib.assets.graphics.GeometricShapes attribute), 66	(pygamelib.assets.graphics.GeometricShapes attribute), 66
WHITE_RIGHT_POINTING_SMALL_TRIANGLE	WHITE_VERTICAL_RECTANGLE
(pygamelib.assets.graphics.GeometricShapes	(pygamelib.assets.graphics.GeometricShapes
attribute), 66	attribute), 66

Widget (class in pygamelib.gfx.ui), 684	WILTED_FLOWER (pygamelib.assets.graphics.Models at-
widgets() (pygamelib.gfx.ui.BoxLayout method), 623	tribute), 222
widgets() (pygamelib.gfx.ui.FormLayout method), 640	WIND_CHIME (pygamelib.assets.graphics.Models at-
widgets() (pygamelib.gfx.ui.GridLayout method), 644	tribute), 222
widgets() (pygamelib.gfx.ui.Layout method), 653	WIND_FACE (pygamelib.assets.graphics.Models at-
width (pygamelib.board_items.Actionable property), 270	tribute), 222
width (pygamelib.board_items.ActionableTile property), 281	WINDOW (pygamelib.assets.graphics.Models attribute), 222
width (pygamelib.board_items.BoardComplexItem property), 292	WINE_GLASS (pygamelib.assets.graphics.Models attribute), 222
width (pygamelib.board_items.BoardItem property), 312	WINKING_FACE (pygamelib.assets.graphics.Models
${\tt width} ({\it pygamelib.board\_items.BoardItemComplexComport}) \\$	nent attribute), 222
property), 301	WINKING_FACE_WITH_TONGUE
${\tt width}(pygame lib. board\_items. Board Item Voidproperty),$	(pygamelib.assets.graphics.Models attribute),
321	222
width (pygamelib.board_items.Camera property), 330	WOLF (pygamelib.assets.graphics.Models attribute), 222
width (pygamelib.board_items.Character property), 340	WOMAN (pygamelib.assets.graphics.Models attribute), 222
width (pygamelib.board_items.ComplexDoor property),	WOMAN_AND_MAN_HOLDING_HANDS
351	(pygamelib.assets.graphics.Models attribute),
width (pygamelib.board_items.ComplexNPC property),	223
363	WOMAN_DANCING (pygamelib.assets.graphics.Models at-
width (pygamelib.board_items.ComplexPlayer prop-	tribute), 223
erty), 373	WOMAN_WITH_HEADSCARF
width (pygamelib.board_items.ComplexTreasure property), 384	(pygamelib.assets.graphics.Models attribute), 223
width (pygamelib.board_items.ComplexWall property), 395	WOMANS_BOOT (pygamelib.assets.graphics.Models attribute), 223
width (pygamelib.board_items.Door property), 405	WOMANS_CLOTHES (pygamelib.assets.graphics.Models at-
width (pygamelib.board_items.GenericActionableStructure	
property), 414	WOMANS_HAT (pygamelib.assets.graphics.Models at-
width (pygamelib.board_items.GenericStructure prop-	tribute), 223
erty), 433	WOMANS_SANDAL (pygamelib.assets.graphics.Models at-
width(pygamelib.board_items.GenericStructureComplexC	
property), 423	WOMEN_HOLDING_HANDS
width (pygamelib.board_items.Immovable property), 442	(pygamelib.assets.graphics.Models attribute),
width (pygamelib.board_items.Movable property), 452	223
width (pygamelib.board_items.NPC property), 463	WOMENS_ROOM (pygamelib.assets.graphics.Models at-
width (pygamelib.board_items.Player property), 472	tribute), 223
width (pygamelib.board_items.Projectile property), 485	WOOD (pygamelib.assets.graphics.Models attribute), 223
width (pygamelib.board_items.TextItem property), 496	WOOZY_FACE (pygamelib.assets.graphics.Models at-
width (pygamelib.board_items.Tile property), 507	tribute), 223
width (pygamelib.board_items.Treasure property), 517	WORLD_MAP (pygamelib.assets.graphics.Models at-
width (pygamelib.board_items.Wall property), 527	tribute), 223
width (pygamelib.engine.Board property), 543	WORM (pygamelib.assets.graphics.Models attribute), 223
width (pygamelib.engine.Screen property), 580	WORRIED_FACE (pygamelib.assets.graphics.Models
width (pygamelib.gfx.core.Sprite property), 602	attribute), 223
width (pygamelib.gfx.ui.Box property), 619	WRAPPED_GIFT (pygamelib.assets.graphics.Models
width (pygamelib.gfx.ui.BoxLayout property), 623	attribute), 223
width (pygamelib.gfx.ui.FormLayout property), 640	${\tt WRENCH}  (pygamelib. assets. graphics. Models  attribute),$
width (pygamelib.gfx.ui.GridLayout property), 644	223
width (pygamelib.gfx.ui.Layout property), 653	WRITING_HAND (pygamelib.assets.graphics.Models
width (pygamelib.gfx.ui.LineInput property), 660	attribute), 223
width (pygamelib.gfx.ui.Widget property), 688	

#### X yellow\_square() (pygamelib.gfx.core.Sprixel class method), 611 x (pygamelib.base.Vector2D property), 260 yellow\_square() (pygamelib.gfx.particles.ParticleSprixel x (pygamelib.gfx.particles.CircleEmitter property), 694 class method), 731 **x** (pygamelib.gfx.particles.ColorParticle property), 700 YEN\_BANKNOTE (pygamelib.assets.graphics.Models **x** (pygamelib.gfx.particles.ColorPartitionParticle propattribute), 223 erty), 706 YIN\_YANG (pygamelib.assets.graphics.Models attribute), x (pygamelib.gfx.particles.Particle property), 722 x (pygamelib.gfx.particles.ParticleEmitter property), 714 YO\_YO (pygamelib.assets.graphics.Models attribute), 223 **x** (pygamelib.gfx.particles.PartitionParticle property), Ζ (pygamelib.gfx.particles.RandomColorParticle prop-ZANY\_FACE erty), 744 (pygamelib.assets.graphics.Models $\mathbf{x}$ (pygamelib.gfx.particles.RandomColorPartitionParticle tribute), 223 ZEBRA (pygamelib.assets.graphics.Models attribute), 223 property), 749 ZIPPER\_MOUTH\_FACE (pygamelib.assets.graphics.Models x (pygamelib.gfx.ui.LineInput property), 660 attribute), 223 x (pygamelib.gfx.ui.Widget property), 688 X\_IN\_A\_RECTANGLE\_BOX ZOMBIE (pygamelib.assets.graphics.Models attribute), (pygamelib.assets.graphics.MiscTechnicals 223 ZZZ (pygamelib.assets.graphics.Models attribute), 223 attribute), 92 Υ y (pygamelib.base.Vector2D property), 260 y (pygamelib.gfx.particles.CircleEmitter property), 694 y (pygamelib.gfx.particles.ColorParticle property), 700 y (pygamelib.gfx.particles.ColorPartitionParticle property), 706 y (pygamelib.gfx.particles.Particle property), 722 y (pygamelib.gfx.particles.ParticleEmitter property), 714 (pygamelib.gfx.particles.PartitionParticle property), (pygamelib.gfx.particles.RandomColorParticle property), 744 y (pygamelib.gfx.particles.RandomColorPartitionParticle property), 749 y (pygamelib.gfx.ui.LineInput property), 660 y (pygamelib.gfx.ui.Widget property), 689 YARN (pygamelib.assets.graphics.Models attribute), 223 YAWNING\_FACE (pygamelib.assets.graphics.Models attribute), 223 yellow() (pygamelib.base.Text static method), 256 yellow\_bright() (pygamelib.base.Text static method), 256 YELLOW\_CIRCLE (pygamelib.assets.graphics.Models attribute), 223 yellow\_dim() (pygamelib.base.Text static method), 257 YELLOW\_HEART (pygamelib.assets.graphics.Models attribute), 223 yellow\_rect() (pygamelib.gfx.core.Sprixel method), 611 yellow\_rect() (pygamelib.gfx.particles.ParticleSprixel class method), 731

YELLOW\_SQUARE (pygamelib.assets.graphics.Models at-

tribute), 223

at-