pygamelib Documentation

Release 1.2.3

Arnaud Dupuis

Mar 01, 2021

Contents (API reference):

CHAPTER 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 inheritate from one of these base class.

This module contains the simple actuators classes. Simple actuators are movement related one. They allow for predetermined movements patterns.

`Actuator`(parent)	Actuator is the base class for all Actuators.
`Behavioral`(parent)	The behavioral actuator is inheriting from Actuator and is adding a next_action() method.
`RandomActuator`([moveset, parent])	A class that implements a random choice of movement.
`PathActuator`([path, parent])	The path actuator is a subclass of `Actuator`.
`PatrolActuator`([path, parent])	The patrol actuator is a subclass of `PathActuator`.
`UnidirectionalActuator`([direction, parent])	A class that implements a single movement.
`PathFinder`([game, actuated_object, . . .])	**Important:** This module assume a one step movement.

1.1 pygamelib.actuators.Actuator

class `pygamelib.actuators.`**`Actuator`**(*parent*)

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*)

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.
`next_move`()	That method needs to be implemented by all actuators or a NotImplementedError exception will be raised.
`pause`()	Set the actuator state to PAUSED.
`start`()	Set the actuator state to RUNNING.
`stop`()	Set the actuator state to STOPPED.

1.2 pygamelib.actuators.Behavioral

class pygamelib.actuators.**Behavioral**(*parent*)

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*)

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

Methods

`__init__`(parent)	The constructor simply construct an Actuator.
`next_action`()	That method needs to be implemented by all behavioral 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.
`pause`()	Set the actuator state to PAUSED.
`start`()	Set the actuator state to RUNNING.
`stop`()	Set the actuator state to STOPPED.

1.3 pygamelib.actuators.RandomActuator

class pygamelib.actuators.**RandomActuator**(*moveset=None*, *parent=None*)

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

- **moveset** (`list`) – A list of movements.

- **parent** (pygamelib.board_items.BoardItem) – The parent object to actuate.

__init__(*moveset=None*, *parent=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.
next_move()	Return a randomly selected movement
pause()	Set the actuator state to PAUSED.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.

1.4 pygamelib.actuators.PathActuator

class pygamelib.actuators.**PathActuator**(*path=None*, *parent=None*)

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=None*, *parent=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.
next_move()	Return the movement based on current index
pause()	Set the actuator state to PAUSED.
set_path(path)	Defines a new path
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.

1.5 pygamelib.actuators.PatrolActuator

class pygamelib.actuators.**PatrolActuator**(*path=None*, *parent=None*)

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 beggining of the list. Once it reaches the beggining 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=None*, *parent=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.
next_move()	Return the movement based on current index
pause()	Set the actuator state to PAUSED.
set_path(path)	Defines a new path
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.

1.6 pygamelib.actuators.UnidirectionalActuator

class pygamelib.actuators.**UnidirectionalActuator**(*direction=10000100*, *parent=None*)

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=10000100*, *parent=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.
next_move()	Return the direction.
pause()	Set the actuator state to PAUSED.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.

1.7 pygamelib.actuators.PathFinder

class pygamelib.actuators.**PathFinder**(*game=None*, *actuated_object=None*, *circle_waypoints=True*, *parent=None*)

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 instanciated 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)

__init__(*game=None*, *actuated_object=None*, *circle_waypoints=True*, *parent=None*)

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

Methods

__init__([game, actuated_object, . . .])	The constructor simply construct an Actuator.
add_waypoint(row, column)	Add a waypoint to the list of waypoints.
clear_waypoints()	Empty the waypoints stack.
current_path()	This method simply return a copy of the current path of the actuator.
current_waypoint()	Return the currently active waypoint.
find_path()	Find a path to the destination.
next_action()	That method needs to be implemented by all behavioral actuators or a NotImplementedError exception will be raised.
next_move()	This method return the next move calculated by this actuator.
next_waypoint()	Return the next active waypoint.
pause()	Set the actuator state to PAUSED.

Continued on next page

Table 8 – continued from previous page

remove_waypoint(row, column)	Remove a waypoint from the stack.
set_destination([row, column])	Set the targeted destination.
start()	Set the actuator state to RUNNING.
stop()	Set the actuator state to STOPPED.

class pygamelib.actuators.**Actuator**(*parent*)

Bases: `object`

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.

next_move()

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

Raises NotImplementedError

pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

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()
```

class pygamelib.actuators.**Behavioral**(*parent*)

Bases: *pygamelib.actuators.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.

next_action()

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

Raises NotImplementedError

next_move()

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

Raises NotImplementedError

pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

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()
```

class pygamelib.actuators.**PathActuator**(*path=None*, *parent=None*)

Bases: *pygamelib.actuators.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.

next_move()

Return the movement based on current index

The movement is selected from path if state is RUNNING, otherwise it should return None. 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 | None

Example:

```
pathactuator.next_move()
```

pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

set_path (*path*)

Defines a new path

This will also reset the index back to 0.

Parameters **path** (*list*) – A list of movements.

Example:

```
pathactuator.set_path([constants.UP,constants.DOWN,constants.LEFT,constants.
↪RIGHT])
```

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()
```

class pygamelib.actuators.**PathFinder** (*game=None*, *actuated_object=None*, *circle_waypoints=True*, *parent=None*)

Bases: *pygamelib.actuators.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 instanciated 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)

add_waypoint (*row*, *column*)

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)
```

clear_waypoints()

Empty the waypoints stack.

Example:

```
pf.clear_waypoints()
```

current_path()

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:

```
mykillernpc.actuator = PathFinder(
                            game=mygame,
                            actuated_object=mykillernpc
                        )
mykillernpc.actuator.set_destination(
                            mygame.player.pos[0],
                            mygame.player.pos[1]
                        )
mykillernpc.actuator.find_path()
for i in mykillernpc.actuator.current_path():
    print(i)
```

current_waypoint()

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)
```

find_path()

Find a path to the destination.

Destination (PathFinder.destination) has to be set beforehand. This method implements a Breadth First Search algorithm (Wikipedia) to find the shortest path to destination.

Example:

```
mykillernpc.actuator = PathFinder(
        game=mygame, actuated_object=mykillernpc
    )
mykillernpc.actuator.set_destination(
        mygame.player.pos[0], mygame.player.pos[1]
    )
mykillernpc.actuator.find_path()
```

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

next_action()

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

Raises NotImplementedError

next_move()

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 betwen 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 *find_path()*.
- In any case, if there is no more waypoints in the path this method returns NO_DIR (see *constants*)

Example:

```
seeker = NPC(model=Sprites.SKULL)
seeker.actuator = PathFinder(game=mygame,actuated_object=seeker)
while True:
    seeker.actuator.set_destination(mygame.player.pos[0],mygame.player.pos[1])
    # next_move() will call find_path() for us.
    next_move = seeker.actuator.next_move()
    if next_move == constants.NO_DIR:
        seeker.actuator.set_destination(mygame.player.pos[0],mygame.player.
↪pos[1])
    else:
        mygame.current_board().move(seeker,next_move,1)
```

next_waypoint()

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)
```

pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

remove_waypoint(*row*, *column*)

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:

```
method()
```

set_destination(*row=0*, *column=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()
```

class pygamelib.actuators.**PatrolActuator**(*path=None*, *parent=None*)

Bases: *pygamelib.actuators.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 beggining of the list. Once it reaches the beggining 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.

next_move()

Return the movement based on current index

The movement is selected from path if state is RUNNING, otherwise it should return None. 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 | None

Example:

```
patrolactuator.next_move()
```

pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

set_path(*path*)

Defines a new path

This will also reset the index back to 0.

Parameters **path** (*list*) – A list of movements.

Example:

```
pathactuator.set_path([constants.UP,constants.DOWN,constants.LEFT,constants.
↪RIGHT])
```

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()
```

class pygamelib.actuators.**RandomActuator**(*moveset=None*, *parent=None*)

Bases: *pygamelib.actuators.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

- **moveset** (*list*) – A list of movements.
- **parent** (pygamelib.board_items.BoardItem) – The parent object to actuate.

next_move()

Return a randomly selected movement

The movement is randomly selected from moveset if state is RUNNING, otherwise it should return None.

Returns The next movement

Return type int | None

Example:

```
randomactuator.next_move()
```

pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

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()
```

class pygamelib.actuators.**UnidirectionalActuator**(*direction=10000100*, *parent=None*)

Bases: *pygamelib.actuators.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.

next_move()

Return the direction.

The movement is always direction if state is RUNNING, otherwise it returns None.

Returns The next movement

Return type int | None

Example:

```
unidirectional_actuator.next_move()
```

pause()

Set the actuator state to PAUSED.

Example:

```
mygame.pause()
```

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()
```

CHAPTER 2

assets

2.1 graphics

Important: The Graphics module was introduced in version 1.1.0.

The Graphics module contains the following classes:

Models	List of models (emojis by unicode denomination)
Blocks	Block elements (unicode)
BoxDrawings	Box drawing elements (unicode)
GeometricShapes	Geometric shapes elements (unicode)

2.1.1 pygamelib.assets.graphics.Models

```
class pygamelib.assets.graphics.Models
```

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 =
- PIñATA =
- 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 =
- 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 = ™
- 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__``()`

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`	Initialize self.

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
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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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
COUCH_AND_LAMP

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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
NAIL_POLISH
NAME_BADGE
NATIONAL_PARK
NAUSEATED_FACE
NAZAR_AMULET
NECKTIE
NERD_FACE
NESTING_DOLLS
NEUTRAL_FACE

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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
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

Continued on next page

Table 3 – continued from previous page

TOOTH
TOOTHBRUSH
TOP_ARROW
TOP_HAT
TORNADO
TRACKBALL
TRACTOR
TRADE_MARK
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
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

Continued on next page

Table 3 – continued from previous page

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

Continued on next page

Table 3 – continued from previous page

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

2.1.2 pygamelib.assets.graphics.Blocks

class pygamelib.assets.graphics.**Blocks**

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__``()`

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`	Initialize self.

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`

Continued on next page

Table 5 – continued from previous page

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

2.1.3 pygamelib.assets.graphics.BoxDrawings

class pygamelib.assets.graphics.**BoxDrawings**

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 = └
- 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__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.

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
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

Continued on next page

Table 7 – continued from previous page

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

Continued on next page

Table 7 – continued from previous page

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
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

2.1.4 pygamelib.assets.graphics.GeometricShapes

class pygamelib.assets.graphics.**GeometricShapes**

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 = ◯
- 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__()`

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.

Attributes

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
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

Continued on next page

Table 9 – continued from previous page

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
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

Continued on next page

Table 9 – continued from previous page

WHITE_UP_POINTING_TRIANGLE
WHITE_UP_POINTING_TRIANGLE_WITH_DOT
WHITE_VERTICAL_RECTANGLE

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 *Sprixel* 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

class pygamelib.assets.graphics.**Blocks**

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 =

class pygamelib.assets.graphics.**BoxDrawings**

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 = └
- 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 =

class `pygamelib.assets.graphics.`**`GeometricShapes`**

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 = ◯
- 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 =

class pygamelib.assets.graphics.**Models**

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 =
- PIñATA =
- 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 =
- 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 = ™
- 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 =

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.

CHAPTER 3

base

The Game.py module has only one class: Game. It is what could be called the game engine. It holds a lot of methods that helps taking care of some complex mechanics behind the curtain.

This module contains the Inventory class.

This module regroup all the specific exceptions of the library. The idea behind most exceptions is to provide more context and info that the standard exceptions.

This module contains the Board class. It is the base class for all levels.

`Math`()	The math class regroup math functions required for game development.
`PglException`(error, message)	Exception raised for non specific errors in the pygamelib.
`PglInvalidLevelException`(message)	Exception raised if a level is not associated to a board in Game().
`PglInvalidTypeException`(message)	Exception raised for invalid types.
`PglObjectIsNotMovableException`(message)	Exception raised if the object that is being moved is not a subclass of Movable.
`PglOutOfBoardBoundException`(message)	Exception for out of the board's boundaries operations.
`Vector2D`([row, column])	A 2D vector class.
`Text`([text, fg_color, bg_color, style])	An object to manipulate and display text in multiple contexts.

3.1 pygamelib.base.Math

class `pygamelib.base.`**`Math`**

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__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
distance(row1, column1, row2, column2)	Return the euclidian distance between to points.
intersect(row1, column1, width1, height1, . . .)	This function check if 2 rectangles intersect.

3.2 pygamelib.base.PglException

exception pygamelib.base.**PglException**(*error*, *message*)
Exception raised for non specific errors in the pygamelib.

3.3 pygamelib.base.PglInvalidLevelException

exception pygamelib.base.**PglInvalidLevelException**(*message*)
Exception raised if a level is not associated to a board in Game().

3.4 pygamelib.base.PglInvalidTypeException

exception pygamelib.base.**PglInvalidTypeException**(*message*)
Exception raised for invalid types.

3.5 pygamelib.base.PglObjectIsNotMovableException

exception pygamelib.base.**PglObjectIsNotMovableException**(*message*)
Exception raised if the object that is being moved is not a subclass of Movable.

3.6 pygamelib.base.PglOutOfBoardBoundException

exception pygamelib.base.**PglOutOfBoardBoundException**(*message*)
Exception for out of the board’s boundaries operations.

3.7 pygamelib.base.Vector2D

class pygamelib.base.**Vector2D**(*row=0.0*, *column=0.0*)
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 (if you want integer for example).

Vector2D use the row/column internal naming convention as it is easier to visualize For learning developers. 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

- **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*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([row, column])	Initialize self.

Continued on next page

Table 3 – continued from previous page

from_direction(direction, step)	Build and return a Vector2D from a direction.
length()	Returns the length of a 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.
y	y is an alias for row.

3.8 pygamelib.base.Text

class pygamelib.base.**Text**(*text=”*, *fg_color=”*, *bg_color=”*, *style=”*)

An object to manipulate and display text in multiple contexts.

New in version 1.2.0.

The Text class is a collection of text formating 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 generate a Sprite to represent itself. This is particularly useful to the place text on the game Board.

Parameters

- **text** (*str*) – The text to manipulate
- **fg_color** (*str*) – The foreground color for the text.
- **bg_color** (*str*) – The background color for the text.
- **style** (*str*) – The style for the text.

__init__(*text=”*, *fg_color=”*, *bg_color=”*, *style=”*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([text, fg_color, bg_color, style])	Initialize self.
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.
black_dim(message)	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.

Continued on next page

Table 5 – continued from previous page

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.
cyan_bright(message)	This method works exactly the way green_bright() work with different color.
cyan_dim(message)	This method works exactly the way green_bright() work with different color.
debug(message)	Print a debug message.
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
green_dim(message)	This method works exactly the way green_bright() work with different color.
info(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.
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 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.

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

Bases: *pygamelib.base.PglException*

A simple forward to PglException

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `pygamelib.base.`**`HacInvalidLevelException`**(*message*)

Bases: *`pygamelib.base.PglInvalidLevelException`*

Forward to PglInvalidLevelException

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `pygamelib.base.`**`HacInvalidTypeException`**(*message*)

Bases: *`pygamelib.base.PglInvalidTypeException`*

A simple forward to PglInvalidTypeException

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `pygamelib.base.`**`HacInventoryException`**(*error*, *message*)

Bases: *`pygamelib.base.PglInventoryException`*

Forward to PglInventoryException.

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `pygamelib.base.`**`HacObjectIsNotMovableException`**(*message*)

Bases: *`pygamelib.base.PglObjectIsNotMovableException`*

Simple forward to PglObjectIsNotMovableException

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `pygamelib.base.`**`HacOutOfBoardBoundException`**(*message*)

Bases: *`pygamelib.base.PglOutOfBoardBoundException`*

Simple forward to PglOutOfBoardBoundException

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

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.

static distance(*row1*, *column1*, *row2*, *column2*)

Return the euclidian 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:

```
distance = round(base.Math.distance(player.row,
                                    player.column,
                                    npc.row,
                                    npc.column)
```

static intersect(*row1*, *column1*, *width1*, *height1*, *row2*, *column2*, *width2*, *height2*)

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:

```
if intersect(projectile.row, projectile.column, projectile.width,
             projectile.height, bady.row, bady.column, bady.width,
             bady.height):
    projectile.hit([bady])
```

exception pygamelib.base.**PglException**(*error*, *message*)

Bases: Exception

Exception raised for non specific errors in the pygamelib.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pygamelib.base.**PglInvalidLevelException**(*message*)

Bases: Exception

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

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pygamelib.base.**PglInvalidTypeException**(*message*)

Bases: Exception

Exception raised for invalid types.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pygamelib.base.**PglInventoryException**(*error*, *message*)

Bases: Exception

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

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pygamelib.base.**PglObjectIsNotMovableException**(*message*)

Bases: Exception

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

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pygamelib.base.**PglOutOfBoardBoundException**(*message*)

Bases: Exception

Exception for out of the board's boundaries operations.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pygamelib.base.**PglOutOfItemBoundException**(*message*)

Bases: Exception

Exception for out of the item's boundaries operations.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pygamelib.base.**Text**(*text=''*, *fg_color=''*, *bg_color=''*, *style=''*)

Bases: object

An object to manipulate and display text in multiple contexts.

New in version 1.2.0.

The Text class is a collection of text formating 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 generate a *Sprite* to represent itself. This is particularly useful to the place text on the game *Board*.

Parameters

- **text** (*str*) – The text to manipulate
- **fg_color** (*str*) – The foreground color for the text.
- **bg_color** (*str*) – The background color for the text.
- **style** (*str*) – The style for the text.

bg_color = None
The bg_color attribute sets the background color. It needs to be a str.

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.

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...")
```

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|")
```

fg_color = None
The fg_color attribute sets the foreground color. It needs to be a str.

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.

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.")
```

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.

parent = None
This object's parent. It needs to be a *BoardItem*.

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.

style = None

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

text = None

The text attribute. It needs to be a str.

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.

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 (if you want integer for example).

Vector2D use the row/column internal naming convention as it is easier to visualize For learning developers. 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

- **`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
```

`column`

The column component of the vector.

`classmethod from_direction`(*direction*, *step*)

Build and return a Vector2D from a direction.

Directions are from the constants module.

Parameters

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

Example:

```
v2d_up = Vector2D.from_direction(constants.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...')
```

rounding_precision = None

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.

row

The row component of the vector.

unit()

Returns a normalized unit vector.

Returns A unit vector

Return type *Vector2D*

Example:

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

x

x is an alias for column.

y

y is an alias for row.

base.**Text = <class 'pygamelib.base.Text'>**

CHAPTER 4

board_items

This module contains the basic board items classes.

BoardItem(**kwargs)	Base class for any item that will be placed on a Board.
BoardItemVoid(**kwargs)	A class that represent a void cell.
BoardComplexItem(**kwargs)	New in version 1.2.0.
BoardItemComplexComponent(**kwargs)	The default component of a complex item.
Movable(**kwargs)	A class representing BoardItem capable of movements.
Projectile([name, direction, step, range, ...])	A class representing a projectile type board item.
Immovable(**kwargs)	This class derive BoardItem and describe an object that cannot move or be moved (like a wall).
Actionable(**kwargs)	This class derives Immovable.
Character(**kwargs)	A base class for a character (playable or not)
Player(**kwargs)	A class that represent a player controlled by a human.
ComplexPlayer(**kwargs)	New in version 1.2.0.
NPC(**kwargs)	A class that represent a non playable character controlled by the computer.
ComplexNPC(**kwargs)	New in version 1.2.0.
TextItem([text])	New in version 1.2.0.
Wall(**kwargs)	A Wall is a specialized Immovable object that as unmodifiable characteristics:
ComplexWall(**kwargs)	New in version 1.2.0.
Treasure(**kwargs)	A Treasure is an Immovable that is pickable and with a non zero value.
ComplexTreasure(**kwargs)	New in version 1.2.0.

Continued on next page

Table 1 – continued from previous page

Door(**kwargs)	A Door is a *GenericStructure* that is not pickable, overlappable and restorable.
ComplexDoor(**kwargs)	New in version 1.2.0.
GenericStructure(**kwargs)	A GenericStructure is as the name suggest, a generic object to create all kind of structures.
GenericActionableStructure(**kwargs)	A GenericActionableStructure is the combination of a *GenericStructure* and an *Actionable*.
Tile(**kwargs)	New in version 1.2.0.

4.1 pygamelib.board_items.BoardItem

class pygamelib.board_items.**BoardItem**(***kwargs*)

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** (*array*) – 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 '*'.
- **parent** – The parent object of the board item. Usually a Board or Game object.

Important: 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. Python does not really provide a way to prevent changing that member but if you do, you'll break rendering. You have been warned.

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	This is a virtual method that must be implemented in deriving classes.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.

Continued on next page

Table 2 – continued from previous page

display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.
inventory_space()	This is a virtual method that must be implemented in deriving class.
overlappable()	This is a virtual method that must be implemented in deriving class.
pickable()	This is a virtual method that must be implemented in deriving class.
position_as_vector()	Returns the current item position as a Vector2D
row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.2 pygamelib.board_items.BoardItemVoid

class `pygamelib.board_items.`**`BoardItemVoid`**`(**kwargs)`

A class that represent a void cell.

`__init__``(**kwargs)`

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	This is a virtual method that must be implemented in deriving classes.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.
inventory_space()	This is a virtual method that must be implemented in deriving class.
overlappable()	A BoardItemVoid is obviously overlappable (so player and NPC can walk over).
pickable()	A BoardItemVoid is not pickable, therefor this method return false.
position_as_vector()	Returns the current item position as a Vector2D
row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.3 pygamelib.board_items.BoardComplexItem

class pygamelib.board_items.**BoardComplexItem**(***kwargs*)

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. 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 *Sprixel*

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	This is a virtual method that must be implemented in deriving classes.
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`height`	Convenience method to get the height of the item.
`inventory_space`()	This is a virtual method that must be implemented in deriving class.
`item`(row, column)	Return the item at the row, column position if it is within the item's boundaries.
`overlappable`()	This is a virtual method that must be implemented in deriving class.

Continued on next page

Table 4 – continued from previous page

pickable()	This is a virtual method that must be implemented in deriving class.
position_as_vector()	Returns the current item position as a Vector2D
row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
update_sprite()	Update the complex item with the current sprite.
width	Convenience method to get the width of the item.

4.4 pygamelib.board_items.BoardItemComplexComponent

class pygamelib.board_items.**BoardItemComplexComponent**(***kwargs*)

The default component of a complex item.

It is literrally 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*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	Returns True if the item can move, False otherwise.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.
inventory_space()	This is a virtual method that must be implemented in deriving class.
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
restorable()	Returns True if the item is restorable, False otherwise.
row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.5 pygamelib.board_items.Movable

class pygamelib.board_items.**Movable**(***kwargs*)

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__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	Movable implements can_move().
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
has_inventory()	This is a virtual method that must be implemented in deriving class.
height	Convenience method to get the height of the item.
inventory_space()	This is a virtual method that must be implemented in deriving class.
overlappable()	This is a virtual method that must be implemented in deriving class.
pickable()	This is a virtual method that must be implemented in deriving class.
position_as_vector()	Returns the current item position as a Vector2D

Continued on next page

Table 6 – continued from previous page

`row`	Convenience method to get the current stored row of the item.
`store_position`(row, column)	Store the BoardItem position for self access.
`width`	Convenience method to get the width of the item.

4.6 pygamelib.board_items.Projectile

class pygamelib.board_items.**Projectile**(*name='projectile'*, *direction=10000100*, *step=1*, *range=5*, *model=''*, *movement_animation=None*, *hit_animation=None*, *hit_model=None*, *hit_callback=None*, *is_aoe=False*, *aoe_radius=0*, *parent=None*, *callback_parameters=[]*, *movement_speed=0.15*)

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** (*int*) – 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
- **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:

```
fireball = Projectile(
                        name="fireball",
                        model=Utils.red_bright(black_circle),
                        hit_model=Sprites.EXPLOSION,
                    )
fireball.set_direction(constants.RIGHT)
my_game.add_projectile(1, fireball,
                        my_game.player.pos[0], my_game.player.pos[1] + 1)
```

__init__(*name='projectile'*, *direction=10000100*, *step=1*, *range=5*, *model=''*, *movement_animation=None*, *hit_animation=None*, *hit_model=None*, *hit_callback=None*, *is_aoe=False*, *aoe_radius=0*, *parent=None*, *callback_parameters=[]*, *movement_speed=0.15*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([name, direction, step, range, . . .])	Initialize self.
add_directional_animation(direction, animation)	Add an animation for a specific direction.
add_directional_model(direction, model)	Add an model for a specific direction.
can_move()	Movable implements can_move().
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
directional_animation(direction)	Return the animation for a specific direction.
directional_model(direction)	Return the model for a specific direction.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
has_inventory()	Projectile cannot have inventory by default.
height	Convenience method to get the height of the item.

Continued on next page

Table 7 – continued from previous page

hit(objects)	A method that is called when the projectile hit something.
inventory_space()	This is a virtual method that must be implemented in deriving class.
overlappable()	Projectile are overlappable by default.
pickable()	This is a virtual method that must be implemented in deriving class.
position_as_vector()	Returns the current item position as a Vector2D
remove_directional_animation(direction)	Remove an animation for a specific direction.
remove_directional_model(direction)	Remove the model for a specific direction.
restorable()	We assume that by default, Projectiles are restorable.
row	Convenience method to get the current stored row of the item.
set_direction(direction)	Set the direction of a projectile
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.7 pygamelib.board_items.Immovable

class pygamelib.board_items.**Immovable**(***kwargs*)

This class derive BoardItem and describe an object that cannot move or be moved (like a wall). Thus this class implements BoardItem.can_move(). However it does not implement BoardItem.pickable() or BoardItem.overlappable()

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	Return the capability of moving of an item.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.
inventory_space()	Return the size of the Immovable Item for the Inventory.
overlappable()	This is a virtual method that must be implemented in deriving class.
pickable()	This is a virtual method that must be implemented in deriving class.
position_as_vector()	Returns the current item position as a Vector2D
restorable()	This is a virtual method that must be implemented in deriving class.

Continued on next page

Table 8 – continued from previous page

row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.8 pygamelib.board_items.Actionable

class pygamelib.board_items.**Actionable**(***kwargs*)

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

Parameters

- **action** (*function*) – the reference to a function (Attention: no parentheses at the end of the function name).
- **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*

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.

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
activate()	This function is calling the action function with the action_parameters.
can_move()	Return the capability of moving of an item.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.
inventory_space()	Return the size of the Immovable Item for the *Inventory*.
overlappable()	This is a virtual method that must be implemented in deriving class.
pickable()	This is a virtual method that must be implemented in deriving class.

Continued on next page

Table 9 – continued from previous page

position_as_vector()	Returns the current item position as a Vector2D
restorable()	This is a virtual method that must be implemented in deriving class.
row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.9 pygamelib.board_items.Character

class pygamelib.board_items.**Character**(***kwargs*)

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__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.

4.10 pygamelib.board_items.Player

class pygamelib.board_items.**Player**(***kwargs*)

A class that represent a player controlled by a human. It accepts all the parameters from *Character* and is a *Movable*.

This class sets a couple of variables to default values:

- 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.

Note: If no inventory is passed as parameter a default one is created.

`__init__`(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	Movable implements can_move().
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`has_inventory`()	This method returns True (a player has an inventory).
`height`	Convenience method to get the height of the item.
`inventory_space`()	This is a virtual method that must be implemented in deriving class.
`overlappable`()	This method returns false (a player cannot be overlapped).
`pickable`()	This method returns False (a player is obviously not pickable).
`position_as_vector`()	Returns the current item position as a Vector2D
`row`	Convenience method to get the current stored row of the item.
`store_position`(row, column)	Store the BoardItem position for self access.
`width`	Convenience method to get the width of the item.

4.11 pygamelib.board_items.ComplexPlayer

class pygamelib.board_items.**ComplexPlayer**(***kwargs*)
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:

```
player = ComplexPlayer(
        name='Mighty Wizard',
```

(continues on next page)

(continued from previous page)

```
        sprite=sprite_collection['wizard_idle']
    )
```

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	Movable implements can_move().
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`has_inventory`()	This method returns True (a player has an inventory).
`height`	Convenience method to get the height of the item.
`inventory_space`()	This is a virtual method that must be implemented in deriving class.
`item`(row, column)	Return the item at the row, column position if it is within the item's boundaries.
`overlappable`()	This method returns false (a player cannot be overlapped).
`pickable`()	This method returns False (a player is obviously not pickable).
`position_as_vector`()	Returns the current item position as a Vector2D
`row`	Convenience method to get the current stored row of the item.
`store_position`(row, column)	Store the BoardItem position for self access.
`update_sprite`()	Update the complex item with the current sprite.
`width`	Convenience method to get the width of the item.

4.12 pygamelib.board_items.NPC

class pygamelib.board_items.**NPC**(***kwargs*)

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__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	Movable implements can_move().
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
has_inventory()	Define if the NPC has an inventory.
height	Convenience method to get the height of the item.
inventory_space()	This is a virtual method that must be implemented in deriving class.
overlappable()	Define if the NPC is overlappable.
pickable()	Define if the NPC is pickable.
position_as_vector()	Returns the current item position as a Vector2D
row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.13 pygamelib.board_items.ComplexNPC

class pygamelib.board_items.**ComplexNPC**(***kwargs*)
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:

```
player = ComplexNPC(
        name='Idiot McComplexStupid',
        sprite=npc_sprite_collection['troll_licking_stones']
    )
```

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	Movable implements can_move().
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`has_inventory`()	Define if the NPC has an inventory.
`height`	Convenience method to get the height of the item.
`inventory_space`()	This is a virtual method that must be implemented in deriving class.
`item`(row, column)	Return the item at the row, column position if it is within the item's boundaries.
`overlappable`()	Define if the NPC is overlappable.
`pickable`()	Define if the NPC is pickable.
`position_as_vector`()	Returns the current item position as a Vector2D
`row`	Convenience method to get the current stored row of the item.
`store_position`(row, column)	Store the BoardItem position for self access.
`update_sprite`()	Update the complex item with the current sprite.
`width`	Convenience method to get the width of the item.

4.14 pygamelib.board_items.TextItem

class pygamelib.board_items.**TextItem**(*text=None*, ***kwargs*)
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*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`([text])	Initialize self.
`can_move`()	This is a virtual method that must be implemented in deriving classes.
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`height`	Convenience method to get the height of the item.
`inventory_space`()	This is a virtual method that must be implemented in deriving class.
`item`(row, column)	Return the item at the row, column position if it is within the item's boundaries.
`overlappable`()	This is a virtual method that must be implemented in deriving class.
`pickable`()	This is a virtual method that must be implemented in deriving class.
`position_as_vector`()	Returns the current item position as a Vector2D
`row`	Convenience method to get the current stored row of the item.
`store_position`(row, column)	Store the BoardItem position for self access.
`update_sprite`()	Update the complex item with the current sprite.
`width`	Convenience method to get the width of the item.

Attributes

`text`	The text within the item.

4.15 pygamelib.board_items.Wall

class `pygamelib.board_items.`**`Wall`**(***kwargs*)
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*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	Return the capability of moving of an item.
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`height`	Convenience method to get the height of the item.
`inventory_space`()	Return the size of the Immovable Item for the `Inventory`.
`overlappable`()	This represent the capacity for a `BoardItem` to be overlapped by player or NPC.
`pickable`()	This represent the capacity for a `BoardItem` to be pick-up by player or NPC.
`position_as_vector`()	Returns the current item position as a Vector2D
`restorable`()	This represent the capacity for an `Immovable` `Movable` item.
`row`	Convenience method to get the current stored row of the item.
`store_position`(row, column)	Store the BoardItem position for self access.
`width`	Convenience method to get the width of the item.

4.16 pygamelib.board_items.ComplexWall

class pygamelib.board_items.**ComplexWall**(***kwargs*)

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:

```
wall = ComplexWall(
        sprite=sprite_brick_wall
    )
```

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	Return the capability of moving of an item.
`collides_with`(other)	Tells if this item collides with another item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`inventory_space`()	Return the size of the Immovable Item for the `Inventory`.
`item`(row, column)	Return the item at the row, column position if it is within the item's boundaries.
`overlappable`()	This represent the capacity for a `BoardItem` to be overlapped by player or NPC.
`pickable`()	This represent the capacity for a `BoardItem` to be pick-up by player or NPC.
`position_as_vector`()	Returns the current item position as a Vector2D
`restorable`()	This represent the capacity for an `Immovable` `Movable` item.
`store_position`(row, column)	Store the BoardItem position for self access.
`update_sprite`()	Update the complex item with the current sprite.

Attributes

`column`	Convenience method to get the current stored column of the item.
`height`	Convenience method to get the height of the item.
`row`	Convenience method to get the current stored row of the item.
`width`	Convenience method to get the width of the item.

4.17 pygamelib.board_items.Treasure

class pygamelib.board_items.**Treasure**(***kwargs*)

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=Sprites.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__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	Return the capability of moving of an item.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.
inventory_space()	Return the size of the Immovable Item for the *Inventory*.
overlappable()	This represent the capacity for a Treasure to be overlapped by player or NPC.
pickable()	This represent the capacity for a Treasure to be picked-up by player or NPC.
position_as_vector()	Returns the current item position as a Vector2D
restorable()	This represent the capacity for a Treasure to be restored after being overlapped.
row	Convenience method to get the current stored row of the item.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.18 pygamelib.board_items.ComplexTreasure

class pygamelib.board_items.**ComplexTreasure**(***kwargs*)
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*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	Return the capability of moving of an item.
`collides_with`(other)	Tells if this item collides with another item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`inventory_space`()	Return the size of the Immovable Item for the `Inventory`.
`item`(row, column)	Return the item at the row, column position if it is within the item's boundaries.
`overlappable`()	This represent the capacity for a Treasure to be overlapped by player or NPC.
`pickable`()	This represent the capacity for a Treasure to be picked-up by player or NPC.
`position_as_vector`()	Returns the current item position as a Vector2D
`restorable`()	This represent the capacity for a Treasure to be restored after being overlapped.
`store_position`(row, column)	Store the BoardItem position for self access.
`update_sprite`()	Update the complex item with the current sprite.

Attributes

`column`	Convenience method to get the current stored column of the item.
`height`	Convenience method to get the height of the item.
`row`	Convenience method to get the current stored row of the item.
`width`	Convenience method to get the width of the item.

4.19 pygamelib.board_items.Door

class `pygamelib.board_items.`**`Door`**(***kwargs*)

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=Sprites.DOOR,type='locked_door')
```

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	Return the capability of moving of an item.
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`height`	Convenience method to get the height of the item.
`inventory_space`()	Return the size of the Immovable Item for the `Inventory`.
`overlappable`()	This represent the capacity for a `BoardItem` to be overlapped by player or NPC.
`pickable`()	This represent the capacity for a BoardItem to be picked-up by player or NPC.
`position_as_vector`()	Returns the current item position as a Vector2D
`restorable`()	This represent the capacity for an `Immovable BoardItem` (in this case a GenericStructure item) to be restored by the board if the item is overlappable and has been overlapped by another `Movable` item.
`row`	Convenience method to get the current stored row of the item.

Continued on next page

Table 23 – continued from previous page

set_overlappable(val)	Make the structure overlappable or not.
set_pickable(val)	Make the structure pickable or not.
set_restorable(val)	Make the structure restorable or not.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.20 pygamelib.board_items.ComplexDoor

class pygamelib.board_items.**ComplexDoor**(***kwargs*)

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*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	Return the capability of moving of an item.
collides_with(other)	Tells if this item collides with another item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
inventory_space()	Return the size of the Immovable Item for the *Inventory*.
item(row, column)	Return the item at the row, column position if it is within the item's boundaries.
overlappable()	This represent the capacity for a *BoardItem* to be overlapped by player or NPC.
pickable()	This represent the capacity for a BoardItem to be picked-up by player or NPC.
position_as_vector()	Returns the current item position as a Vector2D
restorable()	This represent the capacity for an *Immovable BoardItem* (in this case a GenericStructure item) to be restored by the board if the item is overlappable and has been overlapped by another *Movable* item.
set_overlappable(val)	Make the structure overlappable or not.
set_pickable(val)	Make the structure pickable or not.
set_restorable(val)	Make the structure restorable or not.

Continued on next page

Table 24 – continued from previous page

store_position(row, column)	Store the BoardItem position for self access.
update_sprite()	Update the complex item with the current sprite.

Attributes

column	Convenience method to get the current stored column of the item.
height	Convenience method to get the height of the item.
row	Convenience method to get the current stored row of the item.
width	Convenience method to get the width of the item.

4.21 pygamelib.board_items.GenericStructure

class pygamelib.board_items.**GenericStructure**(***kwargs*)

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 GenericActionableStructure 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.

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__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
can_move()	Return the capability of moving of an item.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.

Continued on next page

Table 26 – continued from previous page

debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.
inventory_space()	Return the size of the Immovable Item for the *Inventory*.
overlappable()	This represent the capacity for a *BoardItem* to be overlapped by player or NPC.
pickable()	This represent the capacity for a BoardItem to be picked-up by player or NPC.
position_as_vector()	Returns the current item position as a Vector2D
restorable()	This represent the capacity for an *Immovable BoardItem* (in this case a GenericStructure item) to be restored by the board if the item is overlappable and has been overlapped by another *Movable* item.
row	Convenience method to get the current stored row of the item.
set_overlappable(val)	Make the structure overlappable or not.
set_pickable(val)	Make the structure pickable or not.
set_restorable(val)	Make the structure restorable or not.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.22 pygamelib.board_items.GenericActionableStructure

class pygamelib.board_items.**GenericActionableStructure**(***kwargs*)

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.

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
activate()	This function is calling the action function with the action_parameters.
can_move()	Return the capability of moving of an item.
collides_with(other)	Tells if this item collides with another item.
column	Convenience method to get the current stored column of the item.
debug_info()	Return a string with the list of the attributes and their current value.
display()	Print the model WITHOUT carriage return.
distance_to(other)	Calculates the distance with an item.
height	Convenience method to get the height of the item.

Continued on next page

Table 27 – continued from previous page

inventory_space()	Return the size of the Immovable Item for the *Inventory*.
overlappable()	This represent the capacity for a *BoardItem* to be overlapped by player or NPC.
pickable()	This represent the capacity for a BoardItem to be picked-up by player or NPC.
position_as_vector()	Returns the current item position as a Vector2D
restorable()	This represent the capacity for an *Immovable BoardItem* (in this case a GenericStructure item) to be restored by the board if the item is overlappable and has been overlapped by another *Movable* item.
row	Convenience method to get the current stored row of the item.
set_overlappable(val)	Make the structure overlappable or not.
set_pickable(val)	Make the structure pickable or not.
set_restorable(val)	Make the structure restorable or not.
store_position(row, column)	Store the BoardItem position for self access.
width	Convenience method to get the width of the item.

4.23 pygamelib.board_items.Tile

class pygamelib.board_items.**Tile**(***kwargs*)

New in version 1.2.0.

A Tile is a standard *BoardComplexItem* configured by default to:

- be overlappable
- 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.

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.

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*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(**kwargs)	Initialize self.
`can_move`()	A Tile cannot move.
`collides_with`(other)	Tells if this item collides with another item.
`column`	Convenience method to get the current stored column of the item.
`debug_info`()	Return a string with the list of the attributes and their current value.
`display`()	Print the model WITHOUT carriage return.
`distance_to`(other)	Calculates the distance with an item.
`height`	Convenience method to get the height of the item.
`inventory_space`()	Return the size of the Immovable Item for the `Inventory`.
`item`(row, column)	Return the item at the row, column position if it is within the item's boundaries.
`overlappable`()	This represent the capacity for a `BoardItem` to be overlapped by player or NPC.
`pickable`()	This represent the capacity for a BoardItem to be picked-up by player or NPC.
`position_as_vector`()	Returns the current item position as a Vector2D
`row`	Convenience method to get the current stored row of the item.
`store_position`(row, column)	Store the BoardItem position for self access.
`update_sprite`()	Update the complex item with the current sprite.
`width`	Convenience method to get the width of the item.

class `pygamelib.board_items.`**`Actionable`**(***kwargs*)

Bases: *`pygamelib.board_items.Immovable`*

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

Parameters

- **`action`** (*`function`*) – the reference to a function (Attention: no parentheses at the end of the function name).
- **`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`*

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.

`activate`()

This function is calling the action function with the action_parameters.

Usually it's automatically called by *`move()`* when a Player or NPC (see *`board_items`*)

class pygamelib.board_items.**BoardComplexItem**(***kwargs*)

Bases: *pygamelib.board_items.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. 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 *Sprixel*

item(*row*, *column*)

Return the item at the row, column position if it is within the item's boundaries.

Return type *pygamelib.board_items.BoardItem*

Raises ***PglOutOfBoardBoundException*** – if row or column are out of bound.

update_sprite()

Update the complex item with the current sprite. This method needs to be called everytime the sprite is changed.

Example:

```
item = BoardComplexItem(sprite=position_idle)
for s in [walk_1, walk_2, walk_3, walk_4]:
    item.sprite = s
    item.update_sprite()
    board.move(item, constants.RIGHT, 1)
    time.sleep(0.2)
```

class pygamelib.board_items.**BoardItem**(***kwargs*)

Bases: object

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** (*array*) – 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 '*'.
- **parent** – The parent object of the board item. Usually a Board or Game object.

Important: 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. Python does not really provide a way to prevent changing that member but if you do, you'll break rendering. You have been warned.

can_move()

This is a virtual method that must be implemented in deriving classes. This method has to return True or False. This represent the capacity for a BoardItem to be moved by the Board.

collides_with(*other*)

Tells if this item collides with another item.

Parameters **other** (*BoardItem*) – The item you want to check for collision.

Return type bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

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

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
```

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.')
```

inventory_space()

This is a virtual method that must be implemented in deriving class. This method has to return an integer. This represent the size of the BoardItem for the *Inventory*. It is used for example to evaluate the space taken in the inventory.

Important: That abstract function was called size() before version 1.2.0. As it was exclusively used for inventory space management, it as been renamed. Particularly because now items do have a need for a size.

overlappable()

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 BoardItem to be overlapped by another BoardItem.

pickable()

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 BoardItem to be pick-up by player or NPC.

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()
```

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...')
```

store_position (*row*, *column*)

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self postion. 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*.

Example:

```
item.store_position(3,4)
```

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.')
```

class pygamelib.board_items.**BoardItemComplexComponent** (***kwargs*)

Bases: *pygamelib.board_items.BoardItem*

The default component of a complex item.

It is literrally 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.

can_move ()

Returns True if the item can move, False otherwise.

Example:

```
if item.item(4,5).can_move():
    print('The item can move')
```

overlappable ()

Returns True if the item is overlappable, False otherwise.

Example:

```
if item.item(4,5).overlappable():
    print('The item is overlappable')
```

pickable ()

Returns True if the item is pickable, False otherwise.

Example:

```
if item.item(4,5).pickable():
    print('The item is pickable')
```

restorable()

Returns True if the item is restorable, False otherwise.

Example:

```
if item.item(4,5).restorable():
    print('The item is restorable')
```

class pygamelib.board_items.**BoardItemVoid**(***kwargs*)

Bases: *pygamelib.board_items.BoardItem*

A class that represent a void cell.

overlappable()

A BoardItemVoid is obviously overlappable (so player and NPC can walk over).

Returns True

pickable()

A BoardItemVoid is not pickable, therefor this method return false.

Returns False

class pygamelib.board_items.**Character**(***kwargs*)

Bases: object

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.

class pygamelib.board_items.**ComplexDoor**(***kwargs*)

Bases: *pygamelib.board_items.Door*, *pygamelib.board_items.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
    )
```

class pygamelib.board_items.**ComplexNPC**(***kwargs*)

Bases: *pygamelib.board_items.NPC*, *pygamelib.board_items.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:

```
player = ComplexNPC(
        name='Idiot McComplexStupid',
        sprite=npc_sprite_collection['troll_licking_stones']
    )
```

class pygamelib.board_items.**ComplexPlayer**(***kwargs*)

Bases: *pygamelib.board_items.Player*, *pygamelib.board_items.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:

```
player = ComplexPlayer(
        name='Mighty Wizard',
        sprite=sprite_collection['wizard_idle']
    )
```

class pygamelib.board_items.**ComplexTreasure**(***kwargs*)

Bases: *pygamelib.board_items.Treasure*, *pygamelib.board_items.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
    )
```

class pygamelib.board_items.**ComplexWall**(***kwargs*)

Bases: *pygamelib.board_items.Wall*, *pygamelib.board_items.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:

```
wall = ComplexWall(
        sprite=sprite_brick_wall
    )
```

class pygamelib.board_items.**Door**(***kwargs*)

Bases: *pygamelib.board_items.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=Sprites.DOOR,type='locked_door')
```

class pygamelib.board_items.**GenericActionableStructure**(***kwargs*)

Bases: *pygamelib.board_items.GenericStructure*, *pygamelib.board_items.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.

class pygamelib.board_items.**GenericStructure**(***kwargs*)

Bases: *pygamelib.board_items.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 GenericActionableStructure 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.

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.

overlappable()

This represent the capacity for a *BoardItem* to be overlapped by player or NPC.

To set this value please use *set_overlappable()*

Returns False

Return type bool

See also:

set_overlappable()

pickable()

This represent the capacity for a BoardItem to be picked-up by player or NPC.

To set this value please use *set_pickable()*

Returns True or False

Return type bool

See also:

set_pickable()

restorable()

This represent the capacity for an *Immovable BoardItem* (in this case a GenericStructure item) to be restored by the board if the item is overlappable and has been overlapped by another *Movable* item.

The value of this property is set with *set_restorable()*

Returns False

Return type bool

See also:

set_restorable()

set_overlappable(*val*)

Make the structure overlappable or not.

Parameters **val** (*bool*) – True or False depending on the fact that the structure can be overlapped (i.e that a Player or NPC can step on it) or not.

Example:

```
myneatstructure.set_overlappable(True)
```

set_pickable(*val*)

Make the structure pickable or not.

Parameters **val** (*bool*) – True or False depending on the pickability of the structure.

Example:

```
myneatstructure.set_pickable(True)
```

set_restorable(*val*)

Make the structure restorable or not.

Parameters **val** (*bool*) – True or False depending on the restorability of the structure.

Example:

```
myneatstructure.set_restorable(True)
```

class pygamelib.board_items.**GenericStructureComplexComponent**(***kwargs*)

Bases: *pygamelib.board_items.GenericStructure*, *pygamelib.board_items.BoardItemComplexComponent*

A ComplexComponent specifically for generic structures.

class pygamelib.board_items.**Immovable**(***kwargs*)

Bases: *pygamelib.board_items.BoardItem*

This class derive BoardItem and describe an object that cannot move or be moved (like a wall). Thus this class implements BoardItem.can_move(). However it does not implement BoardItem.pickable() or BoardItem.overlappable()

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

inventory_space()

Return the size of the Immovable Item for the *Inventory*.

Returns The size of the item.

Return type int

restorable()

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 an Immovable BoardItem to be restored by the board if the item is overlappable and has been overlapped by another Movable (*Movable*) item.

class pygamelib.board_items.**Movable**(***kwargs*)

Bases: *pygamelib.board_items.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()

can_move()

Movable implements can_move().

Returns True

Return type Boolean

dtmove

has_inventory()

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.

class pygamelib.board_items.**NPC**(***kwargs*)

Bases: *pygamelib.board_items.Movable*, *pygamelib.board_items.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()
```

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 :(")
```

overlappable()

Define if the NPC is overlappable.

Obviously this method also always return False.

Returns False

Return type Boolean

Example:

```
if mynpc.overlappable():
    Utils.warn("Something is fishy, that NPC is overlappable but"
        "is not a Ghost...")
```

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...")
```

class pygamelib.board_items.**Player**(***kwargs*)

Bases: *pygamelib.board_items.Movable*, *pygamelib.board_items.Character*

A class that represent a player controlled by a human. It accepts all the parameters from *Character* and is a *Movable*.

This class sets a couple of variables to default values:

- 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.

Note: If no inventory is passed as parameter a default one is created.

`has_inventory``()`
This method returns True (a player has an inventory).

`overlappable``()`
This method returns false (a player cannot be overlapped).

Note: If you wish your player to be overlappable, you need to inherit from that class and re-implement overlappable().

`pickable``()`
This method returns False (a player is obviously not pickable).

class `pygamelib.board_items.`**`Projectile`**(*name='projectile', direction=10000100, step=1, range=5, model='', movement_animation=None, hit_animation=None, hit_model=None, hit_callback=None, is_aoe=False, aoe_radius=0, parent=None, callback_parameters=[], movement_speed=0.15*)

Bases: *`pygamelib.board_items.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`** (*`int`*) – 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
- **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:

```
fireball = Projectile(
                        name="fireball",
                        model=Utils.red_bright(black_circle),
                        hit_model=Sprites.EXPLOSION,
                    )
fireball.set_direction(constants.RIGHT)
my_game.add_projectile(1, fireball,
                       my_game.player.pos[0], my_game.player.pos[1] + 1)
```

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(constants.UP, constants.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(constants.UP, updward_animation)
```

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(constants.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(constants.UP)
```

has_inventory()

Projectile cannot have inventory by default.

Returns False

Return type Boolean

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])
```

overlappable()

Projectile are overlappable by default.

Returns True

Return type Boolean

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(constants.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(constants.UP)
```

restorable()

We assume that by default, Projectiles are restorable.

Returns True

Return type bool

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(constants.UP)
```

class pygamelib.board_items.**TextItem**(*text=None*, ***kwargs*)

Bases: *pygamelib.board_items.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)
```

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.

class pygamelib.board_items.**Tile**(***kwargs*)

Bases: *pygamelib.board_items.GenericStructure*, *pygamelib.board_items.BoardComplexItem*

New in version 1.2.0.

A Tile is a standard *BoardComplexItem* configured by default to:

- be overlappable
- 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.

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.

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] )
```

can_move()

A Tile cannot move.

Returns False

Return type bool

class pygamelib.board_items.**Treasure**(***kwargs*)

Bases: *pygamelib.board_items.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=Sprites.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}")
```

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

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

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

class pygamelib.board_items.**Wall**(***kwargs*)

Bases: *pygamelib.board_items.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.

overlappable()

This represent the capacity for a *BoardItem* to be overlapped by player or NPC.

Returns False

Return type bool

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?')
```

restorable()

This represent the capacity for an *Immovable* Movable item. A wall is not overlappable.

Returns False

Return type bool

CHAPTER 5

constants

Accessible constants are the following:

General purpose:

- PYGAMELIB_VERSION

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:

- POS_TOP
- POS_BOTTOM
- ORIENTATION_HORIZONTAL
- ORIENTATION_VERTICAL

Actions states (for Actuators for example):

- RUNNING
- PAUSED
- STOPPED

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.

CHAPTER 6

engine

The game module contains the core classes for a game:

- The Game object itself.
- The Board object.
- The Inventory object.

The Game object is what could be called the game engine. It holds a lot of methods that helps taking care of some complex mechanics behind the curtain.

The Board class is the base class for all levels.

`Board`(**kwargs)	A class that represent a game board.
`Game`([name, boards, menu, current_level, . . .])	A class that serve as a game engine.
`Inventory`([max_size, parent])	A class that represent the Player (or NPC) inventory.
`Screen`([terminal])	The screen object is pretty straightforward: it is an object that allow manipulation of the screen.

6.1 pygamelib.engine.Board

class `pygamelib.engine.`**`Board`**(***kwargs*)

A class that represent a game board.

The board is being represented by a square matrix. For the moment a board only support one player.

The Board object is the base object to build a level : you create a Board and then you add BoardItems (or objects derived from BoardItem).

Parameters

- **`name`** (*`str`*) – the name of the Board
- **`size`** (*`list`*) – array [width,height] with width and height being int. The size of the board.

- **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** (*bool*) – A boolean to show or hide the warning about boards bigger than 80 rows and columns.

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(**kwargs)	Initialize self.
check_sanity()	Check the board sanity.
clear_cell(row, column)	Clear cell (row, column)
display()	Display the entire board.
display_around(item, row_radius, column_radius)	Display only a part of the board.
generate_void_cell()	This method return a void cell.
get_immovables(**kwargs)	Return a list of all the Immovable objects in the Board.
get_movables(**kwargs)	Return a list of all the Movable objects in the Board.
height	A convenience read only property to get the height of the Board.
init_board()	Initialize the board with BoardItemVoid that uses ui_board_void_cell as model.
init_cell(row, column)	Initialize a specific cell of the board with BoardItemVoid that uses ui_board_void_cell as model.
item(row, column)	Return the item at the row, column position if within board's boundaries.
move(item, direction[, step])	Board.move() is a routing function.
place_item(item, row, column)	Place an item at coordinates row and column.
width	A convenience read only property to get the width of the Board.

6.2 pygamelib.engine.Game

class pygamelib.engine.**Game**(*name='Game'*, *boards={}*, *menu={}*, *current_level=None*, *enable_partial_display=False*, *partial_display_viewport=None*, *mode=90000003*, *user_update=None*, *input_lag=0.01*, *enable_physic=False*)

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.

Parameters

- **name** (*str*) – The Game name.
- **boards** (*dict*) – A dictionnary of boards with the level number as key and a board reference as value.
- **menu** (*dict*) – A dictionnary of menus with a category (str) as key and another dictionnary (key: a shortcut, value: a description) 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()`.
- **mode** (*int*) – The mode parameter configures the way the run() method is going to behave. The default value is constants.MODE_TBT. TBT is short for "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 constants.MODE_RT. 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.
- **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.

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='Game'*, *boards={}*, *menu={}*, *current_level=None*, *enable_partial_display=False*, *partial_display_viewport=None*, *mode=90000003*, *user_update=None*, *input_lag=0.01*, *enable_physic=False*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`([name, boards, menu, ...])	Initialize self.
`actuate_npcs`(level_number[, elapsed_time])	Actuate all NPCs on a given level
`actuate_projectiles`(level_number[, elapsed_time])	Actuate all Projectiles on a given level
`add_board`(level_number, board)	Add a board for the level number.
`add_menu_entry`(category, shortcut, message)	Add a new entry to the menu.
`add_npc`(level_number, npc[, row, column])	Add a NPC to the game.
`add_projectile`(level_number, projectile[, ...])	Add a Projectile to the game.
`animate_items`(level_number[, elapsed_time])	That method goes through all the BoardItems of a given map and call Animation.next_frame().
`change_level`(level_number)	Change the current level, load the board and place the player to the right place.
`clear_screen`()	Clear the whole screen (i.e: remove everything written in terminal)
`config`([section])	Get the content of a previously loaded configuration section.
`create_config`(section)	Initialize a new config section.
`current_board`()	This method return the board object corresponding to the current_level.
`delete_menu_category`([category])	Delete an entire category from the menu.
`display_board`()	Display the current board.
`display_menu`(category[, orientation, paginate])	Display the menu.
`display_player_stats`([life_model, void_model])	Display the player name and health.
`get_board`(level_number)	This method returns the board associated with a level number.
`get_key`()	Reads the next key-stroke returning it as a string.
`get_menu_entry`(category, shortcut)	Get an entry of the menu.
`load_board`(filename[, lvl_number])	Load a saved board
`load_config`(filename[, section])	Load a configuration file from the disk.
`move_player`(direction[, step])	Easy wrapper for Board.move().
`neighbors`([radius, obj])	Get a list of neighbors (non void item) around an object.
`pause`()	Set the game engine state to PAUSE.
`remove_npc`(level_number, npc)	This methods remove the NPC from the level in parameter.
`save_board`(lvl_number, filename)	Save a board to a JSON file
`save_config`([section, filename, append])	Save a configuration section.
`start`()	Set the game engine state to RUNNING.
`stop`()	Set the game engine state to STOPPED.
`update_menu_entry`(category, shortcut, message)	Update an entry of the menu.

6.3 pygamelib.engine.Inventory

class pygamelib.engine.**Inventory**(*max_size=10*, *parent=None*)

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.

The constructor takes only one parameter: the maximum size of the inventory. 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. Deafult value: 10.
- **parent** – The parent object (usually a BoardItem).

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*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([max_size, parent])	Initialize self.
add_item(item)	Add an item to the inventory.
delete_item(name)	Delete the item corresponding to the name given in argument.
empty()	Empty the inventory Example.
get_item(name)	Return the item corresponding to the name given in argument.
items_name()	Return the list of all items names in the inventory.
search(query)	Search for objects in the inventory.
size()	Return the cumulated size of the inventory.
value()	Return the cumulated value of the inventory.

6.4 pygamelib.engine.Screen

class pygamelib.engine.**Screen**(*terminal=None*)

The screen object is pretty straightforward: it is an object that allow manipulation of the screen. At the moment it relies heavily on the blessed module, but it wraps a lot of its methods and provide easy calls to actions.

Parameters **terminal** (Terminal) – A Terminal reference.

Example:

```
screen = Screen(terminal=Terminal())
screen.display_at('This is centered', int(screen.height/2), int(screen.width/2))
```

__init__(*terminal=None*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([terminal])	Initialize self.
clear()	This methods clear the screen

Attributes

height	This method wraps Terminal.height and return the height of the terminal window in number of characters.
width	This method wraps Terminal.width and return the width of the terminal window in number of characters.

class pygamelib.engine.**Board**(***kwargs*)

Bases: object

A class that represent a game board.

The board is being represented by a square matrix. For the moment a board only support one player.

The Board object is the base object to build a level : you create a Board and then you add BoardItems (or objects derived from BoardItem).

Parameters

- **name** (*str*) – the name of the Board
- **size** (*list*) – array [width,height] with width and height being int. The size of the board.
- **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** (*bool*) – A boolean to show or hide the warning about boards bigger than 80 rows and columns.

check_sanity()

Check the board sanity.

This is essentially an internal method called by the constructor.

clear_cell(*row*, *column*)

Clear cell (row, column)

This method clears a cell, meaning it position a void_cell BoardItemVoid at these coordinates.

Parameters

- **row** (*int*) – The row of the item to remove
- **column** (*int*) – The column of the item to remove

Example:

```
myboard.clear_cell(3,4)
```

Warning: This method does not check the content before, it *will* overwrite the content.

Important: This method test if something is left on the overlapped layer. If so, it restore what was overlapped instead of creating a new void item. It also removes the items from the the list of movables and immovables. 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.

display()

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*)

Display only a part of the board.

This method behaves like display() but only display a part of the board around an object (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

- **object** (*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")
```

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 as model.

Example:

```
myboard.init_board()
```

init_cell(*row*, *column*)

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)
```

item(*row*, *column*)

Return the item at the row, column position if within board's boundaries.

Return type *pygamelib.board_items.BoardItem*

Raises ***PglOutOfBoardBoundException*** – if row or column are out of bound.

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** (pygamelib.constants 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,constants.UP,1)
```

Important: if the move is successfull, 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.

place_item(*item*, *row*, *column*)

Place an item at coordinates row and column.

If row or column are our of the board boundaries, an PglOutOfBoardBoundException is raised.

If the item is not a subclass of BoardItem, an PglInvalidTypeException

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 `clear_cell()`. 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.

Parameters **item** (`BoardItem`) – The item to remove.

Example:

```
game.current_board().remove_item(game.player)
```

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...')
```

class pygamelib.engine.**Game**(*name='Game'*, *boards={}*, *menu={}*, *current_level=None*, *enable_partial_display=False*, *partial_display_viewport=None*, *mode=90000003*, *user_update=None*, *input_lag=0.01*, *enable_physic=False*)

Bases: `object`

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.

Parameters

- **name** (*str*) – The Game name.
- **boards** (*dict*) – A dictionnary of boards with the level number as key and a board reference as value.

- **menu** (*dict*) – A dictionnary of menus with a category (str) as key and another dictionnary (key: a shortcut, value: a description) 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()`.
- **mode** (*int*) – The mode parameter configures the way the run() method is going to behave. The default value is constants.MODE_TBT. TBT is short for "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 constants.MODE_RT. 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.
- **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.

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.

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.)

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:

```
mygame.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.

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.

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.

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*, *board*)

Add a board for the level number.

This method associate a Board (*pygamelib.engine.Board*) to a level number.

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_menu_entry(*category*, *shortcut*, *message*, *data=None*)

Add a new entry to the menu.

Add another shortcut and message to the specified category.

Categories help organize the different sections of a menu or dialogues.

Parameters

- **category** (*str*) – The category to which the entry should be added.
- **shortcut** (*str*) – A shortcut (usually one key) to display.
- **message** (*various*) – a message that explains what the shortcut does.
- **data** – a data that you can get from the menu object.

The shortcut and data is optional.

Example:

```
game.add_menu_entry('main_menu','d','Go right',constants.RIGHT)
game.add_menu_entry('main_menu',None,'-----------------')
game.add_menu_entry('main_menu','v','Change game speed')
```

add_npc(*level_number*, *npc*, *row=None*, *column=None*)

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().

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)
```

change_level (*level_number*)

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()

config (*section='main'*)

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()
```

create_config(*section*)

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()

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_menu_category(*category=None*)

Delete an entire category from the menu.

That function removes the entire list of messages that are attached to the category.

Parameters **category** (*str*) – The category to delete.

Raises ***PglInvalidTypeException*** – If the category is not a string

---

Important: If the entry have no shortcut it's advised not to try to update unless you have only one NoneType as a shortcut.

Example:

```
game.add_menu_entry('main_menu','d','Go right')
game.update_menu_entry('main_menu','d','Go LEFT',constants.LEFT)
```

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 constants.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]
# This will call Game.current_board().display_around()
mygame.display()
mygame.enable_partial_display = False
```

(continues on next page)

(continued from previous page)

```
# This will call Game.current_board().display()
mygame.display()
```

display_menu(*category*, *orientation=30000100*, *paginate=10*)

Display the menu.

This method display the whole menu for a given category.

Parameters

- **category** (*str*) – The category to display. **Mandatory** parameter.
- **orientation** (*pygamelib.constants*) – The shortcut of the entry you want to get.
- **paginate** (*int*) – pagination parameter (how many items to display before changing line or page).

Example:

```
game.display_menu('main_menu')
game.display_menu('main_menu', constants.ORIENTATION_HORIZONTAL, 5)
```

display_player_stats(*life_model=’\x1b[41m \x1b[0m’*, *void_model=’\x1b[40m \x1b[0m’*)

Display the player name and health.

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*)

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.

get_menu_entry(*category*, *shortcut*)

Get an entry of the menu.

This method return a dictionnary with 3 entries :

- shortcut
- message
- data

Parameters

- **category** (*str*) – The category in which the entry is located.
- **shortcut** (*str*) – The shortcut of the entry you want to get.

Returns The menu entry or None if none was found

Return type dict

Example:

```
ent = game.get_menu_entry('main_menu','d')
game.move_player(int(ent['data']),1)
```

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*, *section='main'*)

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(constants.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]})')
```

`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.
8. Actuates projectiles.
9. Animates items.
10. Actuates particles (WIP).

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.

save_config (*section=None*, *filename=None*, *append=False*)

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')
```

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()
```

stop ()

Set the game engine state to STOPPED.

Example:

```
mygame.stop()
```

update_menu_entry (*category*, *shortcut*, *message*, *data=None*)

Update an entry of the menu.

Update the message associated to a category and a shortcut.

Parameters

- **category** (*str*) – The category in which the entry is located.
- **shortcut** (*str*) – The shortcut of the entry you want to update.
- **message** (*various*) – a message that explains what the shortcut does.
- **data** – a data that you can get from the menu object.

Important: If the entry have no shortcut it's advised not to try to update unless you have only one NoneType as a shortcut.

Example:

```
game.add_menu_entry('main_menu','d','Go right')
game.update_menu_entry('main_menu','d','Go LEFT',constants.LEFT)
```

class pygamelib.engine.**Inventory** (*max_size=10*, *parent=None*)

Bases: object

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.

The constructor takes only one parameter: the maximum size of the inventory. 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. Deafult value: 10.
- **parent** – The parent object (usually a BoardItem).

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.

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.size is greater than the inventory max_size)

Parameters item (*BoardItem*) – the item you want to add

Raises PglInventoryException, PglInvalidTypeException

Example:

```
item = Treasure(model=Sprites.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)
```

Warning: if you try to add more than one item with the same name (or if the name is empty), this function will automatically change the name of the item by adding a UUID to it.

delete_item(*name*)

Delete the item corresponding to the name given in argument.

Parameters name (*str*) – the name of the item you want to delete.

Note: in case an execpetion is raised, the error will be 'no_item_by_that_name' and the message is giving the specifics.

See also:

pygamelib.base.PglInventoryException.

Example:

```
life_container = mygame.player.inventory.get_item('heart_1')
if isinstance(life_container,GenericActionableStructure):
    life_container.action(life_container.action_parameters)
    mygame.player.inventory.delete_item('heart_1')
```

empty()

Empty the inventory Example:

```
if inventory.size() > 0:
    inventory.empty()
```

get_item(*name*)

Return the 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*

Raises PglInventoryException

Note: in case an execpetion is raised, the error will be 'no_item_by_that_name' and the message is giving the specifics.

See also:

pygamelib.base.PglInventoryException.

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.

items_name()

Return the list of all items names in the inventory.

Returns a list of string representing the items names.

Return type list

search(*query*)

Search for objects in the inventory.

All objects that matches the query are going to be returned. :param query: the query that items in the inventory have to match to be returned :type name: str :returns: a table of BoardItems. :rtype: list

Example:

```
for item in game.player.inventory.search('mighty'):
    print(f"This is a mighty item: {item.name}")
```

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}")
```

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
```

class pygamelib.engine.**Screen**(*terminal=None*)

Bases: `object`

The screen object is pretty straightforward: it is an object that allow manipulation of the screen. At the moment it relies heavily on the blessed module, but it wraps a lot of its methods and provide easy calls to actions.

Parameters **terminal** (`Terminal`) – A Terminal reference.

Example:

```
screen = Screen(terminal=Terminal())
screen.display_at('This is centered', int(screen.height/2), int(screen.width/2))
```

clear()

This methods clear the screen

display_at(*text, row=0, column=0, clear_eol=False, end='\n', file=<colorama.ansitowin32.StreamWrapper object>, flush=False*)

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 accross the library. It is reversed compared to the blessed module.

Example:

```
screen.display_at('This is centered',
                  int(screen.height/2),
                  int(screen.width/2),
                  clear_eol=True,
                  end=''
                )
```

display_line (**text*, *end='\n'*, *file=<colorama.ansitowin32.StreamWrapper object>*, *flush=False*)

New in version 1.2.0.

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 the text.
- **end** (*str*) – end sub string added to the printed text. Usually a carriage return.
- **file** (*stream*) –
- **flush** (*bool*) –

Example:

```
game.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}')
```

height

This method wraps Terminal.height and return the height of the terminal window in number of characters.

width

This method wraps Terminal.width and return the width of the terminal window in number of characters.

CHAPTER 7

gfx

The gfx (for graphics) sub-module holds all the classes related to the graphics system.

7.1 core

This module contains the core classes for the "graphic" system.

`Sprixel`([model, bg_color, fg_color, . . .])	A sprixel is the representation of 1 cell of the sprite or one cell on the Board.
`Sprite`([sprixels, default_sprixel, parent, . . .])	The Sprite object represent a 2D "image" that can be used to represent any complex item.
`SpriteCollection`([data])	SpriteCollection is a dictionnary class that derives collections.UserDict.
`Animation`([display_time, auto_replay, . . .])	The Animation class is used to give the ability to have more than one model for a BoardItem.

7.1.1 pygamelib.gfx.core.Sprixel

class `pygamelib.gfx.core.`**`Sprixel`**(*model=", bg_color=", fg_color=", is_bg_transparent=None*)

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 have use Sprixel instead of model.

If the background color and the is_bg_transparent are None or empty strings, the sprixel will be automatically configured with transparent background. In that case, as we can really achieve transparency in the console, the sprixel will take the background color of whatever it is overlapping.

Parameters

- **`model`** (*`str`*) – The model, it can be any string. Preferrably a single character.

- **bg_color** (*str*) – An ANSI escape sequence to configure the background color.
- **fg_color** (*str*) – An ANSI escape sequence to configure the foreground color.
- **is_bg_transparent** –

Example:

```
player = Player(sprixel=Sprixel(
                                '#',
                                screen.terminal.on_color_rgb(128,56,32),
                                screen.terminal.color_rgb(255,255,0),
                                ))
```

__init__(*model=”*, *bg_color=”*, *fg_color=”*, *is_bg_transparent=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([model, bg_color, fg_color, . . .])	Initialize self.
black_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLACK_RECT.
black_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLACK_SQUARE.
blue_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE_RECT.
blue_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.BLUE_SQUARE.
cyan_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.CYAN_RECT.
cyan_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.CYAN_SQUARE.
from_ansi(string)	Takes an ANSI string, parse it and return a Sprixel.
green_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.GREEN_RECT.
green_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.GREEN_SQUARE.
magenta_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.MAGENTA_RECT.
magenta_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.MAGENTA_SQUARE.
red_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.RED_RECT.
red_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.RED_SQUARE.
serialize()	Serialize a Sprixel into a dictionary.
white_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.WHITE_RECT.

Continued on next page

Table 2 – continued from previous page

white_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.WHITE_SQUARE.
yellow_rect()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.YELLOW_RECT.
yellow_square()	This classmethod returns a sprixel that is the equivalent of pygamelib.assets.graphics.YELLOW_SQUARE.

Attributes

bg_color
fg_color
model

7.1.2 pygamelib.gfx.core.Sprite

class pygamelib.gfx.core.**Sprite**(*sprixels=None, default_sprixel=[0m, parent=None, size=[2, 2], name=None*)

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 *Sprixel*.

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, void],
        [
            Sprixel.black_rect(),
            Sprixel.black_rect(),
            void,
```

(continues on next page)

(continued from previous page)

```
            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(),
        ],
        [
            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(),
            Sprixel.red_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
        ],
        [
            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(),
        ],
        [
```

(continues on next page)

(continued from previous page)

```
                void,
                void,
                Sprixel.black_rect(),
                Sprixel.black_rect(),
                void,
                void,
                void,
                void,
            ],
        ],
)
```

__init__(*sprixels=None, default_sprixel=[0m, parent=None, size=[2, 2], name=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`([sprixels, default_sprixel, . . .])	Initialize self.
`calculate_size`()	Calculate the size of the sprite and update the size variable.
`empty`()	Empty the sprite and fill it with default sprixels.
`flip_horizontally`()	Flip the sprite horizontally.
`flip_vertically`()	Flip the sprite vertically (i.e upside/down).
`from_text`(text_object)	Create a Sprite from a `Text` object.
`load_from_ansi_file`(filename[, default_sprixel])	Load an ANSI encoded file into a Sprite object.
`set_sprixel`(row, column, val)	Set a specific sprixel in the sprite to the given value.
`sprixel`([row, column])	Return a sprixel at a specific position within the sprite.

7.1.3 pygamelib.gfx.core.SpriteCollection

class pygamelib.gfx.core.**SpriteCollection**(*data={}*)
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]}')
```

(continues on next page)

(continued from previous page)

```
# 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={}*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`([data])	Initialize self.
`add`(sprite)	Add a Sprite to the collection.
`clear`()	
`copy`()	
`fromkeys`(iterable[, value])	
`get`(k[,d])	
`items`()	
`keys`()	
`load`(data)	Load serialized data and return a new SpriteCollection object.
`load_json_file`(filename)	Load a JSON sprite file into a new SpriteCollection object.
`pop`(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised.
`popitem`()	as a 2-tuple; but raise KeyError if D is empty.
`serialize`()	Return a serialized version of the SpriteCollection.
`setdefault`(k[,d])	
`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`()	

7.1.4 pygamelib.gfx.core.Animation

class pygamelib.gfx.core.**Animation**(*display_time=0.05*, *auto_replay=True*, *frames=None*, *animated_object=None*, *refresh_screen=None*, *initial_index=None*, *parent=None*)

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 explicitely. 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]*) – an array of "frames" (string, sprixel or sprite)
- **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.

Example

```
def redraw_screen(game_object):
    game_object.clear_screen()
    game_object.display_board()

item = BoardItem(model=Sprite.ALIEN, name='Friendly Alien')
# By default BoardItem does not have any animation, we have to
# explicitely create one
item.animation = Animation(display_time=0.1, parent=item,
                           refresh_screen=redraw_screen)
```

__init__(*display_time=0.05*, *auto_replay=True*, *frames=None*, *animated_object=None*, *refresh_screen=None*, *initial_index=None*, *parent=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([display_time, auto_replay, ...])	Initialize self.
add_frame(frame)	Add a frame to the animation.
current_frame()	Return the current frame.
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.
start()	Set the animation state to constants.RUNNING.
stop()	Set the animation state to STOPPED.

class pygamelib.gfx.core.**Animation**(*display_time=0.05*, *auto_replay=True*, *frames=None*, *animated_object=None*, *refresh_screen=None*, *initial_index=None*, *parent=None*)

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 explicitely. 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]*) – an array of "frames" (string, sprixel or sprite)
- **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.

Example

```
def redraw_screen(game_object):
    game_object.clear_screen()
    game_object.display_board()

item = BoardItem(model=Sprite.ALIEN, name='Friendly Alien')
# By default BoardItem does not have any animation, we have to
# explicitely create one
item.animation = Animation(display_time=0.1, parent=item,
                           refresh_screen=redraw_screen)
```

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*) – 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()
```

dtanimate

next_frame()

Update the parent's model, sprixel or sprite with the next frame of the animation.

That method takes care of automatically replaying the animation if the last frame is reached if the state is constants.RUNNING.

If the 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)
)
```

start()

Set the animation state to constants.RUNNING.

If the animation state is not constants.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()
```

class pygamelib.gfx.core.**Sprite**(*sprixels=None, default_sprixel=[0m, parent=None, size=[2, 2], name=None*)

Bases: `object`

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 *Sprixel*.

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, 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(),
        ],
        [
            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(),
            Sprixel.red_rect(),
            Sprixel.white_rect(),
            Sprixel.white_rect(),
        ],
        [
            void,
```

(continues on next page)

(continued from previous page)

```
            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,
        ],
    ],
)
```

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:

```
spr_size = spr.calculate_size()
if spr_size != spr.size:
    raise PglException(
                'perturbation_in_the_Force',
                'Something is very wrong with the sprite!'
            )
```

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}' )
```

classmethod load(*data*)

Create a new Sprite object based on serialized data.

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 `Sprixel` 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 `BoardComplexItem` (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')
```

serialize()

Serialize a Sprite into a dictionary.

Returns The class as a dictionary

Return type dict

Example:

```
json.dump( sprite.serialize() )
```

set_sprixel(*row*, *column*, *val*)

Set a specific sprixel in the sprite to the given value. :param name: some param :type name: str

Example:

```
method()
```

set_transparency(*state*)

This method enable transparent background to all the sprite's sprixels.

Parameters **state** – a boolean to enable or disable background transparency

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.

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!

class pygamelib.gfx.core.**SpriteCollection**(*data={}*)

Bases: `collections.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')
```

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:

```
sprites_village1 = SpriteCollection.load_json_file('gfx/village1.spr')
new_village = SpriteCollection()
new_village.add( copy.deepcopy( sprites_village1.get('bakery') ) )
print( new_village['bakery'] )
```

clear() → None. Remove all items from D.

copy()

classmethod fromkeys(*iterable*, *value=None*)

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

classmethod load(*data*)

Load serialized data and return a new SpriteCollection object.

Parameters **data** (*str*) – Serialized data that need to be expanded into objects.

Returns A new SpriteCollection object.

Return type *SpriteCollection*

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*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

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*]) → 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*) → 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() → an object providing a view on D's values

class pygamelib.gfx.core.**Sprixel**(*model=”*, *bg_color=”*, *fg_color=”*, *is_bg_transparent=None*)

Bases: `object`

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 have use Sprixel instead of model.

If the background color and the is_bg_transparent are None or empty strings, the sprixel will be automatically configured with transparent background. In that case, as we can really achieve transparency in the console, the sprixel will take the background color of whatever it is overlapping.

Parameters

- **model** (*str*) – The model, it can be any string. Preferrably a single character.
- **bg_color** (*str*) – An ANSI escape sequence to configure the background color.
- **fg_color** (*str*) – An ANSI escape sequence to configure the foreground color.
- **is_bg_transparent** –

Example:

```
player = Player(sprixel=Sprixel(
                                '#',
                                screen.terminal.on_color_rgb(128,56,32),
                                screen.terminal.color_rgb(255,255,0),
                                ))
```

bg_color

classmethod black_rect()
This classmethod 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 classmethod 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 classmethod 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 classmethod 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()
```

classmethod cyan_rect()

This classmethod 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 classmethod 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()
```

fg_color

static from_ansi(*string*)

Takes an ANSI string, parse it and return a Sprixel.

Parameters **string** (*str*) – The ANSI string to parse.

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.

classmethod green_rect()

This classmethod 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 classmethod 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:

```
sprixel = Sprixel.green_square()
```

classmethod load(*data*)

Create a new Sprixel object based on serialized data.

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 classmethod 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 classmethod 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()
```

model

classmethod red_rect()

This classmethod 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 classmethod 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()
```

serialize()

Serialize a Sprixel into a dictionary.

Returns The class as a dictionary

Return type dict

Example:

```
json.dump( sprixel.serialize() )
```

classmethod white_rect()

This classmethod 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 classmethod 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 classmethod 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 classmethod 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()
```

7.2 particles

class pygamelib.gfx.particles.**BaseParticle**(***kwargs*)

Bases: *pygamelib.board_items.Movable*

Particles are not ready. This is only an early early test. *you should not use it*. If you do, don't complain. And if you really want to help, interact on Github or Discord. Thank you ;)

can_move()

Movable implements can_move().

Returns True

Return type Boolean

collides_with(*other*)

Tells if this item collides with another item.

Parameters **other** (*BoardItem*) – The item you want to check for collision.

Return type bool

Example:

```
if projectile.collides_with(game.player):
    game.player.hp -= 5
```

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

direction()

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
```

dtmove

has_inventory()

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.

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.')
```

inventory_space()

This is a virtual method that must be implemented in deriving class. This method has to return an integer. This represent the size of the BoardItem for the `Inventory`. It is used for example to evaluate the space taken in the inventory.

Important: That abstract function was called size() before version 1.2.0. As it was exclusively used for inventory space management, it as been renamed. Particularly because now items do have a need for a size.

---

overlappable()

Overlapable always return true. As by definition a particle is overlapable.

pickable()

A particle is not pickable by default. So that method returns False.

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()
```

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...')
```

store_position(*row*, *column*)

Store the BoardItem position for self access.

The stored position is used for consistency and quick access to the self postion. 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`.

Example:

```
item.store_position(3,4)
```

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.')
```

CHAPTER 8

Credits

8.1 Development Leads

- Arnaud Dupuis (@arnauddupuis)

8.2 Top Contributors

- Kalil de Lima (@kaozdl)

8.3 Contributors

- 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)

CHAPTER 9

History

9.1 1.2.3 (2020-09-01)

Emergency release: fix a regression introduced by v1.2.2.

9.2 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.

9.3 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.
- **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!!!).

9.4 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()

9.5 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.

9.6 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).

9.7 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

9.8 2019.5

- Please see the official website.

9.9 pre-2019.5

- Please see the Github for history.

CHAPTER 10

Forewords

This python3 module is a base for the programming lessons of the Hyrule Astronomy Club. It is not meant to be a comprehensive game building library.

It is however meant (and used) to teach core programming concept to kids from age 6 to 13.

CHAPTER 11

Introduction

First of all, his module is exclusively compatible with python 3.6+ (f-string rules).

The core concept is that it revolve around the `Game` object, the `Board` object and the derivatives of *board_items*.

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

The base game makes use of:

- The main “game engine” (gamelib.Game.Game)
- **Many different types of structures (from gamelib.Structures), like:**
  - Wall (well the walls…),
  - Treasure (gems and money bag),
  - GenericStructure (trees),
  - GenericActionnableStructure (hearts and portals).
- Game()’s menu capabilities.
- Player and NPC (from gamelib.Characters)
- Inventory (from gamelib.Inventory)
- Player and Inventory stats
- **Simple actuators (gamelib.SimpleActuators) like:**
  - RandomActuator (NPCs in level 2),
  - PathActuator (NPCs in level 1).

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols