pygtails Documentation

Release

Josie Thompson

Apr 12, 2018

Contents:

CHAPTER 1

Pygtails Library

A simple wrapper around pygame.

Game implements engine functionality. Subclass to build games. GameObject A simple class to provide a more intuitive approach to gamedev.

1.1 Game

class `pygtails.`**`Game`**(*resolution*, *title*, *flags=0*, *depth=0*)

A class that handles pygame events, input, and mouse-collision.

resolution is a 2-tuple of integers that specify the width and height of the screen.

title is a string used as the title of the window.

flags is an integer flag representing the different controls over the display mode that are active. For a full list of the different flags, see *Pygame Display Mode Flags*. For more information on how flags work, see *the Flags tutorial*.

Public Methods:

main, quit, on_focus, on_key_down, on_key_up, on_mouse_move,
on_mouse_up, on_mouse_down, on_resize, update, add_object,
destroy_object, key_is_pressed

Instance variables:

screen

`add_object`(*other*)

Add a GameObject `other` to the Game and return its id.

`destroy_object`(*_id*)

Destroys the object with the given id from the game.

Note: Does not "undraw" the object. This must be done manually (for now)

key_is_pressed(*key*)
Return True if a key is pressed, False if not.

key is pygame keycode. For a full list of keycodes, see *Pygame Keycodes*.

main()
The main loop. Call this to run the game.

on_focus(*event*)
This method is called whenever the window loses or gains focus.

event is a pygame `ACTIVEEVENT` event. It contains the event attributes `gain` and `state`.

event.gain is an integer. It has a value of 1 when the window comes into focus or when the mouse enters the window. It has a value of 0 when the window goes out of focus or when the mouse leaves the window.

event.state is an integer. It has a value of 1 when the mouse exits or leaves the window. It has a value of 2 when the window gains or loses focus.

This method is not predefined.

on_key_down(*event*)
This method is called whenever a key is pressed.

event is a pygame `KEYDOWN` event. It contains the event attributes `unicode`, `key`, and `mod`.

event.unicode is the unicode representation of the key being pressed.

event.key is a pygame keycode representing the key being pressed. For a full list key constants, see *Pygame Keycodes*.

event.mod is a pygame key mod flag representing the "modulating" keys (shift, ctrl, alt, etc.) being pressed when the current key was pressed. For a list of these flags, see *Pygame Key Mod Flags*.

This method is not predefined.

on_key_up(*event*)
This method is called whenever a key is released.

event is a pygame `KEYUP` event. It contains the event attributes `key` and `mod`.

event.key is a pygame keycode representing the key being released. For a full list key constants, see *Pygame Keycodes*.

event.mod is a pygame key mod flag representing the "modulating" keys (shift, ctrl, alt, etc.) pressed when the current key was released. For a full list of these flags, see *Pygame Key Mod Flags*.

This method is not predefined.

on_mouse_down(*event*)
This method is called whenever a mouse button is pressed.

event is a pygame `MOUSEBUTTONDOWN` event. It contains the event attributes `pos` and `button`.

event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse when it was released.

event.button is an integer representing the button being pressed. 1 represents the left mouse button, 2 represents the middle mouse button, and 3 represents the right mouse button.

This method is predefined to implement the GameObject.on_mouse_down method and to update internal data bout whether or not an object is clicked.

To redefine this method while keeping the implementation, call super().on_mouse_up(event) at the top of your function.

on_mouse_move (*event*)

This method is called whenever the mouse is moved.

event is a pygame `MOUSEMOTION` event. It contains the event attributes `pos`, `rel`, and `buttons`.

event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse.

event.rel is a 2-tuple of integers representing the change in x and y coordinates since the last time this function was called.

event.buttons is a 3-tuple of integers representing the amount of mouse buttons being pressed. Index 0 represents the left mouse button, 1 represents the middle mouse button, 2 represents the right mouse button. If the mouse button is down, the value is 1, 0 if it's up.

This method is predefined to implement the on_mouse_[enter, exit, drag] functions.

If you aren't satisfied with the implementation, feel free to redefine it. If you want to keep the implementation but also add additional functionality call super().on_mouse_move(event) when you're redefining the function.

on_mouse_up (*event*)

This method is called whenever a mouse button is released.

event is a pygame `MOUSEBUTTONUP` event. It contains the event attributes `pos` and `button`.

event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse when it was released.

event.button is an integer representing the button being released. 1 represents the left mouse button, 2 represents the middle mouse button, and 3 represents the right mouse button.

This method is predefined to implement the GameObject.on_mouse_up method and to update internal data about whether or not an object is clicked.

To redefine this method while keeping the implementation call super().on_mouse_up(event) at the top of your function.

on_resize (*event*)

This method is called whenever the window is resized.

event is a pygame `VIDEORESIZE` event. it contains the event attributes `size`, `w`, and `h`.

event.size is a 2-tuple of integers representing the width and height of the screen.

event.w is an integer representing the width of the screen.

event.h is an integer representing the height of the screen.

This method is not predefined.

quit (*event*)

The method called when the exit button is pressed.

event is a pygame `QUIT` event. It has no event attributes.

This method is predefined as:

```
pygame.quit()
sys.exit()
```

Redefine it if you need more control.

screen

The pygame Surface used to draw and blit images to the screen.

update()
: This method is called every frame.

  This method is not predefined.

1.2 GameObject

class pygtails.**GameObject**(*game*)
: A simple class to (hopefully) make pygame more intuitive.

  game is the pygame.Game that this GameObject will be added to.

  Intializing a GameObject modifies internal data in the Game it's instantiated by.

  Public Methods:

  > update, on_mouse_enter, on_mouse_exit, on_mouse_stay, on_mouse_down, on_mouse_up, on_mouse_drag, move

  Instance Variables:

  > game, ID

  ID
  : An integer that represents this object's id.

  destroy()
  : Deletes this object from the game world.

  game
  : The pygtails.Game object that this object is a part of.

  on_mouse_down(*event*)
  : This method is called when the mouse is pressed inside this object.

    event is a pygame `MOUSEBUTTONDOWN` event. It contains the event attributes `pos` and `button`.

    event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse when it was released.

    event.button is an integer representing the button being pressed. 1 represents the left mouse button, 2 represents the middle mouse button, and 3 represents the right mouse button.

    This method is not predefined.

  on_mouse_drag(*event*)
  : This method is called each frame this object is dragged by the mouse.

    event is a pygame `MOUSEMOTION` event. It contains the event attributes `pos`, `rel`, and `buttons`.

    event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse.

    event.rel is a 2-tuple of integers representing the change in x and y coordinates since the last time this function was called.

    event.buttons is a 3-tuple of integers representing the amount of mouse buttons being pressed. Index 0 represents the left mouse button, 1 represents the middle mouse button, 2 represents the right mouse button. If the mouse button is down, the value is 1, 0 if it's up.

    This method is not predefined.

  on_mouse_enter(*event*)
  : This method is called whenever the mouse enters this object.

    event is a pygame `MOUSEMOTION` event. It contains the event attributes `pos`, `rel`, and `buttons`.

event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse.

event.rel is a 2-tuple of integers representing the change in x and y coordinates since the last time this function was called.

event.buttons is a 3-tuple of integers representing the amount of mouse buttons being pressed. Index 0 represents the left mouse button, 1 represents the middle mouse button, 2 represents the right mouse button. If the mouse button is down, the value is 1, 0 if it's up.

This method is not predefined.

`on_mouse_exit``(`*event*`)`

This method is called whenever the mouse exits this object.

event is a pygame `MOUSEMOTION` event. It contains the event attributes `pos`, `rel`, and `buttons`.

event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse.

event.rel is a 2-tuple of integers representing the change in x and y coordinates since the last time this function was called.

event.buttons is a 3-tuple of integers representing the amount of mouse buttons being pressed. Index 0 represents the left mouse button, 1 represents the middle mouse button, 2 represents the right mouse button. If the mouse button is down, the value is 1, 0 if it's up.

This method is not predefined.

`on_mouse_stay``(`*event*`)`

This method is called each frame the mouse is within this object.

event is a pygame `MOUSEMOTION` event. It contains the event attributes `pos`, `rel`, and `buttons`.

event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse.

event.rel is a 2-tuple of integers representing the change in x and y coordinates since the last time this function was called.

event.buttons is a 3-tuple of integers representing the amount of mouse buttons being pressed. Index 0 represents the left mouse button, 1 represents the middle mouse button, 2 represents the right mouse button. If the mouse button is down, the value is 1, 0 if it's up.

This method is not predefined.

`on_mouse_up``(`*event*`)`

This method is called on mouse up if this object is clicked.

event is a pygame `MOUSEBUTTONUP` event. It contains the event attributes `pos` and `button`.

event.pos is a 2-tuple of integers representing the x and y coordinates of the mouse when it was released.

event.button is an integer representing the button being released. 1 represents the left mouse button, 2 represents the middle mouse button, and 3 represents the right mouse button.

This method is not predefined.

`update``()`

This method is called every frame.

This method is not predefined.

1.3 Circle

class `pygtails.`**`Circle`**(*game*, *corner*, *radius*)

A GameObject with a circular “hitmask”.

game is the Game this object is a part of.

corner is a 2-tuple of integers representing the x and y coordinates of the upper-left corner of the bounding square of circle.

radius is a numeric value representing the radius of the circle.

Initializing a Circle will modify internal data in the Game it’s instantiated with.

Public Methods:

> update, on_mouse_enter, on_mouse_exit, on_mouse_stay, on_mouse_down,
> on_mouse_up, on_mouse_drag, move

Instance Variables:

> game, ID, center, corner, radius

`center`

A 2-tuple of integers representing the center of the circle.

Setting this will change the `center` and `corner` attributes.

`corner`

A 2-tuple of integers representing the center of the circle.

Setting this will change the `corner` and `center` attributes.

`radius`

An integer representing the radius of the circle.

Setting this will change the `radius` and `center` attributes.

1.4 Rectangle

class `pygtails.`**`Rectangle`**(*game*, *corner*, *width*, *height*)

A GameObject with a rectangular “hitmask”.

game is the Game this object is a part of.

corner is a 2-tuple of integers representing the x and y coordinates of the upper-left corner of the rectangle.

width is an integer representing the width of the rectangle.

height is an integer representing the height of the rectangle.

Initializing a Rectangle will modify internal data in the Game it’s instantiated with.

Public Methods:

> update, on_mouse_enter, on_mouse_exit, on_mouse_stay, on_mouse_down,
> on_mouse_up, on_mouse_drag, move

Instance Variables:

> game, ID, corner, width, height

`corner`

The upper left corner of the rectangle.

A 2-tuple of integers that represent the x and y coordinates of the upper-left corner of the rectangle.

This attribute is mutable.

`corners`

A tuple of all of the corners of the rectangle.

A 2-dimensional tuple, where the inner tuples are 2-tuples of integers representing the x and y coordinates of the different corners of the rectangle.

The order that the points appear are top-left, top-right, bottom-right, bottom-left.

This attribute is immutable.

`height`

An integer that represents the height of the rectangle.

This attribute is mutable.

`width`

An integer that represents the width of the rectangle.

This attribute is mutable.

CHAPTER 2

Hello, Pygtails!

This page provides a simple hello world program written with pygtails and a step-by-step breakdown of what's happening.

```python
import pygame
from pygtails import Game

class Hello(Game):
    def __init__(self):
        super().__init__((400, 300), "Hello, world!")
        self.screen.fill((255, 255, 255))
        pygame.display.flip()

game = Hello()
game.main()
```

Let's start from the beginning

```python
import pygame
from pygtails import Game
```

Pygtails is merely an extension of pygame and having pygame installed is a requirement. `import pygame` will import the pygame module.

`pygtails.Game` is the main class to use to run your pygtails games. `from pygtails import Game` imports the Game class directly into the current namespace so that you don't have to preface `Game` with `pygtails.` everytime you reference it.

`pygtails.Game` also saves an instance of a `pygame.Surface` object to an attribute named `screen` that acts as the main display window for your game. The `Game` constructor takes the same arguments as `pygame.display.set_mode` and is documented here.

So `super().__init__((400, 300), "Hello, world!")` uses the `pygtails.Game` constructor to create a screen with dimensions 400x300 pixels and with aa title "Hello, world!"

`self.screen.fill((255, 255, 255))` accesses the Surface object that was created in the previous line and fills it with the color white. The most basic specification of `Surface.fill` takes a 3-tuple of integers as the RGB values for a color.

Whenever anything is drawn to the screen, it's actually drawn to a sort of buffer screen and changes don't immediately show up. In order to see the changes, we need flush the buffer screen onto the actual screen. To do this we call `pygame.display.flip()`

```
game = Hello()
game.main()
```

Finaly we create an instance of the class we just created and call its main method to run the game. And we're finished! This is the most basic program you can create with pygame and pygtails. To practice using the API, try starting from scratch and creating programs with different sizes, names and background colors without looking at the already written code for reference.

CHAPTER 3

Pygame Constants

One of the reasons I decided to create this little library is because I feel that pygame isn't very well documented. Part of this is attributed to poor organization (in my opinion) of documentation of related data types and functions. This documentation aims to remedy that by putting the documentation in a visible spot to be easy to reference.

3.1 Pygame Keycodes

The following table is almost directly taken from the pygame.key documentation.

Keycode Name	Ascii	Description
K_BACKSPACE	\b	backspace
K_TAB	\t	tab
K_CLEAR		clear
K_RETURN	\r	return
K_PAUSE		pause
K_ESCAPE	^[	escape
K_SPACE		space
K_EXCLAIM	!	exlclamation point
K_QUOTEDBL	“	double quote
K_HASH	#	hashtag
K_DOLLAR	$	dollar sign
K_AMPERSAND	&	ampersand
K_QUOTE	‘	single quote
K_LEFTPAREN	(	opening parenthesis
K_RIGHTPAREN	)	closing parenthesis
K_ASTERISK	*	asterisk
K_PLUS	+	plus
K_COMMA	,	comma
K_MINUS	-	hyphen
K_PERIOD	.	period

Continued on next page

Table 3.1 – continued from previous page

Keycode Name	Ascii	Description
K_SLASH	/	forward slash
K_0	0	0
K_1	1	1
K_2	2	2
K_3	3	3
K_4	4	4
K_5	5	5
K_6	6	6
K_7	7	7
K_8	8	8
K_9	9	9
K_COLON	:	colon
K_SEMICOLON	;	semicolon
K_LESS	<	less-than
K_EQUALS	=	equals
K_GREATER	>	greater-than
K_QUESTION	?	question mark
K_AT	@	at sign
K_LEFTBRACKET	[	opening sqaure bracket
K_BACKSLASH	\	backslash
K_RIGHTBRACKET	]	closing right bracket
K_CARET	^	caret
K_UNDERSCORE	_	underscore
K_BACKQUOTE	‘	backtick
K_a	a	a
K_b	b	b
K_c	c	c
K_d	d	d
K_e	e	e
K_f	f	f
K_g	g	g
K_h	h	h
K_i	i	i
K_j	j	j
K_k	k	k
K_l	l	l
K_m	m	m
K_n	n	n
K_o	o	o
K_p	p	p
K_q	q	q
K_r	r	r
K_s	s	s
K_t	t	t
K_u	u	u
K_v	v	v
K_w	w	w
K_x	x	x
K_y	y	y

Continued on next page

Table 3.1 – continued from previous page

Keycode Name	Ascii	Description
K_z	z	z
K_DELETE		delete
K_KP0		numpad 0
K_KP1		numpad 1
K_KP2		numpad 2
K_KP3		numpad 3
K_KP4		numpad 4
K_KP5		numpad 5
K_KP6		numpad 6
K_KP7		numpad 7
K_KP8		numpad 8
K_KP9		numpad 9
K_KP_PERIOD	.	numpad period
K_KP_DIVIDE	/	numpad divide
K_KP_MULTIPLY	*	numpad multiply
K_KP_MINUS	-	numpad minus
K_KP_PLUS	+	numpad plus
K_KP_ENTER	\r	numpad enter
K_KP_EQUALS	=	numpad equals
K_UP		up arrow
K_DOWN		down arrow
K_RIGHT		right arrow
K_LEFT		left arrow
K_INSERT		insert
K_HOME		home
K_END		end
K_PAGEUP		page up
K_PAGEDOWN		page down
K_F1		F1
K_F3		F3
K_F4		F4
K_F5		F5
K_F6		F6
K_F7		F8
K_F9		F9
K_F10		F10
K_F11		F11
K_F12		F12
K_F13		F13
K_F14		F14
K_F15		F15
K_NUMLOCK		num lock
K_CAPSLOCK		caps lock
K_SCROLLOCK		scroll lock
K_RSHIFT		right shift
K_LSHIFT		left shift
K_RCTRL		right control
K_LCTRL		left control
K_RALT		right alt

Continued on next page

Table 3.1 – continued from previous page

Keycode Name	Ascii	Description
K_LALT		left alt
K_RMETA		right meta
K_LMETA		left meta
K_LSUPER		left "windows" key
K_RSUPER		right "windows" key
K_MODE		mode shift
K_HELP		help
K_PRINT		print screen
K_SYSREQ		sysrq
K_BREAK		break
K_MENU		menu
K_POWER		power
K_EURO		euro

3.2 Pygame Key Mod Flags

The following table is interpreted from the pygame.key documentation. Descriptions left blank are Key Mod Flags that are unclear, and I haven't been able to determine what they do.

Key Mod descriptions prefaced with "Both" shouldn't be confused with "either"

Key Mod Name	Description
KMOD_NONE	No Key Mods
KMOD_LSHIFT	Left Shift
KMOD_RSHIFT	Right Shift
KMOD_SHIFT	Both Shifts
KMOD_CAPS	Caps Lock
KMOD_LCTRL	Left Control
KMOD_RCTRL	Right Control
KMOD_CTRL	Both Controls
KMOD_LALT	Left Alt
KMOD_RALT	Right Alt
KMOD_ALT	Both Alts
KMOD_LMETA	Left Meta
KMOD_RMETA	Right Meta
KMOD_META	Both Metas
KMOD_NUM	Num Lock
KMOD_MODE	

3.3 Pygame Display Mode Flags

The following table is taken almost directly from the pygame.display documentation.

Display Mode Name	Descripton
FULLSCREEN	Create a fullscreen display
DOUBLEBUF	Recommended for HWSURFACE or OPENGL
HWSURFACE	Hardware-accelerate, only in FULLSCREEEN
OPENGL	Create an OpenGL-renderable display
RESIZABLE	Create a resizable window
NOFRAME	Create window with no border or controls

CHAPTER 4

Flags and Bitwise Operations Tutorial

4.1 A Quick Primer on Flags

If you're unfamiliar with the concept of a flag in the context of computer science, hopefully this is a decent place to start. A flag, in this case, is an integer value that represents a combination of different values. Each value is represented as a power of two, or as a bit of an integer. If we look at the binary representation of an integer, it can be said that for each value in the flag, if the value is "active" it's it's relative "bit" will be represented as a 1. If the value is "inactive" it's relative bit is represented as a 0.

As an example, let's say we have a flag with two possible values; let's say "left" and "right"; let's say "left" is represented by the 2^0 place and "right" is represented by the 2^1 place. The possible flag values can then be represented by the following table:

	R inactive	R active
L inactive	00 (0)	10 (2)
L active	01 (1)	11 (3)

4.2 Bitwise Operations

Performing operations on flags is usualy done with bitwise operations. Bitwise operations deal with the binary representations of integers. This is perfect for flags because flags are defined by their binary representations.

Combining two or more flag values is usually done with bitwise-OR (|). The result of a bitwise-OR will include a 1 in every place it appeared in any of the combining values. The following are examples of the product of bitwise-OR operations on two integers::

```
8 (1000) | 1 (0001) = 9 (1001)
6 (0110) | 3 (0011) = 7 (0111)
```

Checking to see if a flag value is active is usually done with bitwise-AND (&). The result of a bitwise-AND will only have a 1 where both of the combining values are 1. The following are examples of the product of bitwise-AND operations on two integers::

```
8 (1000) & 1 (0001) = 0 (0000)
6 (0110) & 3 (0011) = 2 (0010)
```

4.3 Using Bitwise Operations with Flags

In the context of pygame, an example of how this can be used is to check if mutliple key mods are down. To check if both ctrl and alt are pressed, you might check the value of the key mod against KMOD_LCTRL | KMOD_LALT. Assuming `mod` is the flag value for the current key mods being pressed, a good way to do check if a specific combination is being pressed would look something like::

```
combo = KMOD_LCTRL | KMOD_LALT
mod & combo == combo
```

This will perform an inclusive check (other key mods can be pressed as well) to see if ctrl and alt are pressed.:

```
KMOD_LCTRL | KMOD_LALT == mod
```

will perform an exclusive check.

CHAPTER 5

About

Pygtails is a simple little extension of the pygame library.

Pygame is a cool game development library made for python, and it has a great community. It has tons of active users contributing to it's project showcase. A new game is posted there sometimes several times a week, and the majority of the submissions to pyweek seem to be working with pygame.

Despite all of this, it would be a lie to say that the engine has no shortcomings. In my own opinion, the pygame docs are poorly organized and the library itself isn't as intuitive as it can be. A quick google search will also bring up criticisms of Pygame being slower than other options. In a direct quote from their about page: "**It's not the best game library.** It's not even the second best. But we think it's sort of ok."

The Pygtails library attempts to make Pygame a slightly better game library by providing more detailed, organized, and (hopefully) intuitive documentation, as well as by providing some basic front-end functionality present in many popular game engines that Pygame for some reason lacks. One thing that Pygtails does not attempt to do is address the criticisms of Pygame's slowness.

Python Module Index

Index