PyGame Tutorial

Clarissa Littler

September 26, 2017

Contents

1 What is PyGame?

PyGame is a simple and relatively easy to use library for writing games in Python.

It's based on an old and very successful system for graphics, media, and games writing called SDL (`https://www.libsdl.org/`).

2 Our First PyGame example

Here's a small, minimal example of using PyGame that actually loads up a 600 x 480 window and makes the entire window white

```
import pygame
pygame.init()

size = [600,480]
screen = pygame.display.set_mode(size)

screen.fill((255,255,255))
pygame.display.flip()
```

That's not a **ton** of code but that's also not trivial. Let's go through it line by line!

The first thing you have to do is load the `pygame` library in the first place! There can be a **ton** of libraries installed on your system that you might want to be use so not all of them can be loaded at once. So, instead, you need to tell the Python interpreter that it needs to look for the library and load all of *its* code before continuing on.

```
import pygame
```

The next line is something that's required by PyGame itself to do all the initial setup and communication with the hardware in order to be able to do things like display images or play sound.

```
pygame.init()
```

After that, we have actually to setup a screen by using `pygame.display.set_mode` to set the size of the window and create it

```
size = [600,480]
screen = pygame.display.set_mode(size)
```

Here we're using a type of Python data called a *list* that allows us to group data together.

Finally, we have these last two lines where we say the screen should be white and then actually display.

```
screen.fill((255,255,255))
pygame.display.flip()
```

Y'see, up to the point of `pygame.display.flip()` nothing is actually displayed on the screen. Most computer graphics today is done via *buffering*, where all the changes to the screen are calculated in advance and then applied all at once rather than going through all the work of changing the actual display over and over again.

Now, you might notice this code only creates a window for a few moments before it then closes. That's no good!

How do we actually get our window to stay open? That involves introducing a few new concepts. The first one of them is the idea of a *loop* or *iteration*. The basic concept is that if you want to do something *over* and *over* again until something changes, you use a "while loop". In this case, we're going to introduce a loop that is going to keep running and displaying to the screen.

```
import pygame
pygame.init()

size = [600,480]
screen = pygame.display.set_mode(size)

while True:
    screen.fill((255,255,255))
    pygame.display.flip()
```

Uhhh there's a problem here though. You can't actually close this program properly. It's done a good job of staying open, but now it's just staying open **forever**!

The final piece is that we need to listen for an *event*. We need to check to see if the user has attempted to close the window. This is going to be an event called `pygame.QUIT`.

So, now, we're going to create a variable called `done` that we'll use to keep track of whether a `QUIT` even has happened. We are going to use a different kind of loop, a "for loop", to check all the events that have happened since the loop last ran. We're going to use an "if statement" to check if an event was a `QUIT` event, and if it was we'll change `done` to `True`.

```
import pygame
pygame.init()

size = [600,480]
screen = pygame.display.set_mode(size)

done = False

while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
    screen.fill((255,255,255))
    pygame.display.flip()
```

3 The main concepts of PyGame

So we've managed to make a really simple program that

- opens a window
- displays something on it
- responds to events, in this case a `QUIT` event

All things considered, that's a good bit of what you need to know! A lot of what we're going to do in the rest of this tutorial is going to expand on the ideas of displaying things to the screen and responding to events.

There's a couple of important concepts though: surfaces & rectangles.

A surface is the basic way that PyGame (and its underlying library, SDL) handle data that's ready to display. There's always at least one surface in a game, which is the one created when you call `pygame.display.set_mode`. In our example above, we called this surface `screen`. This is the surface that gets rendered every time we call `pygame.display.flip`.

There'll be *other* surfaces though, like the ones made when you load an image or render some text onto the screen.

The following is an example of some code where the text on the screen updates with every letter you type. Every time the loop runs, the program is going to render a surface consisting of all the text in the variable `text`. This surface is stored in the variable `textSurface`. It then combines the text-surface with the main surface for the screen using the `blit` function. The `blit` function takes a surface as an argument and the upper-left corner to render the surface too. In this case, it'll be at the x/y coordinates of (100,100)

```
import pygame
pygame.init()

size = [600,480]
screen = pygame.display.set_mode(size)

done = False
text = ""

alphabet = "abcdefghijklmnopqrstuvwxzy"

while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
```

```
        elif event.type == pygame.KEYDOWN:
            key = pygame.key.name(event.key).lower()
            if key in alphabet:
                text = text + key
            elif event.key == pygame.K_BACKSPACE:
                text = text[0:len(text)-1]
    screen.fill((255,255,255))
    font = pygame.font.SysFont('Arial', 25)
    textSurface = font.render(text,True,(0,0,0))
    screen.blit(textSurface,[100,100])
    pygame.display.flip()
```

Similarly, we can render *images* to the screen from files like this example

```
import pygame
pygame.init()

size = [600,480]
screen = pygame.display.set_mode(size)

done = False

while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
    screen.fill((255,255,255))
    image = pygame.image.load("2000px-Thumbs-up-icon-left.svg.png")
    image = pygame.transform.scale(image,(100,100))

    screen.blit(image,[200,200])
    pygame.display.flip()
```

The other concept that we've been implicitly using this whole time is the *rect*, short for "rectangle". In PyGame, a `Rect` is a kind of primitive data that lays out a rectangular area for a few different uses:

- drawing onto a surface
- determining if two sprites are intersecting

(TO FINISH)

4 Making Pong

Pong is kind of the "hello world" of games writing simply because, well, it was one of the first video games.

If you haven't seen it before it's `https://upload.wikimedia.org/wikipedia/commons/f/f8/Pong.png`

and it's a simple table-tennis type of game.

So we're going to create a really simple Pong clone

```
import pygame
import random
pygame.init()
pygame.mixer.init()

#helper function for choosing random ball velocities to start
def randomVel():
    v = random.randrange(-5,5)
    while v == 0:
        v = random.randrange(-5,5)
    return v

screenHeight = 480
screenWidth = 600
screen = pygame.display.set_mode((screenWidth,screenHeight))

done = False

playerYPos = screenHeight / 2
enemyYPos = screenHeight / 2

playerRect = pygame.Rect(50,screenHeight / 2 - 30, 20, 60)
enemyRect = pygame.Rect(500,screenHeight / 2 - 30, 20, 60)

ballRect = pygame.Rect(screenWidth/2,screenHeight/2,10,10)
ballVel = [randomVel(),randomVel()]

playerScore = 0
enemyScore = 0

backgroundColor = (0,0,0)
```

```
blockColor = (255,255,255)

pygame.key.set_repeat(50,50)

clock = pygame.time.Clock()

font = pygame.font.SysFont('Arial', 50)

pygame.mixer.music.load("bgm.mp3")
pygame.mixer.music.play(-1)

while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
        elif event.type == pygame.KEYDOWN:
            key = event.key
            if key == pygame.K_UP:
                playerRect = playerRect.move(0,-15)
            elif key == pygame.K_DOWN:
                playerRect = playerRect.move(0,15)

    screen.fill(backgroundColor)
    #displayScore
    pscore = font.render(str(playerScore),True,blockColor)
    escore = font.render(str(enemyScore),True,blockColor)
    screen.blit(pscore,[50,50])
    screen.blit(escore,[500,50])
    #move ball and check collisions
    ballRect = ballRect.move(ballVel[0],ballVel[1])
    if ballRect.bottom > screenHeight or ballRect.top < 0:
        ballVel[1] = -ballVel[1]
    elif ballRect.left < 0:
        enemyScore = enemyScore + 1
        ballRect.center = (screenWidth / 2, screenHeight / 2)
        ballVel = [randomVel(),randomVel()]
    elif ballRect.right > screenWidth:
        playerScore = playerScore + 1
        ballRect.center = (screenWidth / 2, screenHeight / 2)
        ballVel = [randomVel(),randomVel()]
```

```
elif ballRect.colliderect(playerRect):
    ballVel[0] = -ballVel[0]
elif ballRect.colliderect(enemyRect):
    ballVel[0] = -ballVel[0]

pygame.draw.rect(screen,blockColor, ballRect)
# draw player
pygame.draw.rect(screen,blockColor, playerRect)
# draw enemy
pygame.draw.rect(screen,blockColor, enemyRect)

pygame.display.flip()

clock.tick(30)
```