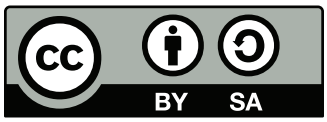
Pygame tutorial

Paweł Widera



This work is licensed under a
Creative Commons Attribution-ShareAlike 4.0 International License.

2014-02-12

Contents

Introduction

This is a tutorial to pygame, a Python library for game development that is a wrapper over the SDL library written in C. We will start from a simple code that displays an empty window, and incrementally introduce more functionality to our example game. Sometimes we will do a bit of refactoring and rewrite older code in a better way or fix a bug. The entire process should be quite similar to a development of a real game (but simpler).

Beware that this is not be just a copy & paste exercise, and although the code fragments will be provided to save the typing time and effort, the complete code will not be shown. You will have to understand what each piece of code does to put them all together.

After each new piece is integrated, the code should by **tested** by running the game. This way you will be able to understand the effect of each change, as well as verify that no errors have been introduced.

At the end of the tutorial (which should last no longer than 1.5h), we will continue working on our game in a dojo fashion. The task would be to add a computer controlled player to the game. The AI should be challenging enough but at the same time not cheating too much to make a win possible.

PONG

The game we are going to write together will be PONG, one of the first computer games ever made. It is a simulation of a table tennis game. It was released 40 years ago in a form of a coin-operated arcade game machine by Atari. A home version of the game was released a few years later and was an instant hit. It was plugged directly to a TV, had analogue controls and was the grand grand mother of the today's game consoles :)

The basics

Before we start, make sure that you have the pygame module installed. Follow the instructions on pygame website or if you use GNU/Linux, just install it from the package repository. We will be using version 1.9.1. After the installation, test if the module works by typing `import pygame` in the Python console.

When you run the game, use the command line to make sure that you will not miss the error messages.

Game code template

Of course before we run our game we need to write some code first. We will start from creating a game window:

```
#!/usr/bin/env python

# import the pygame module, so you can use it
import pygame

# define a main function
def main():
  # create a screen surface of a given size
  screen = pygame.display.set_mode((320,240))
```

```
  # set window caption
  pygame.display.set_caption("PONG")

  # control variable for the main loop
  running = True

  # main game loop
  while running:
    # read events from the event queue
    for event in pygame.event.get():
      # on QUIT event, exit the main loop
      if event.type == pygame.QUIT:
        running = False

# if this module is executed as a script, run the main function
if __name__ == "__main__":
  main()
```

Be careful with the leading whitespaces. Either use a tab or the same amount of spaces for every indent. You have to be consistent because Python will not forgive you and will throw syntax errors. To make sure it works **test** it now.

Drawing objects

Empty window is not that very exciting, so let's draw some objects. We will start with a background. We aim for the classic tennis table look of PONG, so we are going to split the screen with a dotted line. Add the following code to the `main` function (below the window caption setter) and **run** your code:

```
# create background surface
background = pygame.Surface([320, 240])
background.fill(pygame.Color("black"))
for y in range(0, 240, 10):
  pygame.draw.line(background, pygame.Color("white"), (160,y), (160,y+3))

# draw background on screen
screen.blit(background, (0, 0))

# display screen surface
pygame.display.flip()
```

For the PONG game we will also need a ball and two rackets. We draw them after the background is copied to the screen (the `blit` function) but before the screen is displayed (the `flip` function):

```
# draw a ball (color, position, radius)
pygame.draw.circle(screen, pygame.Color("white"), (160,120), 3)

# draw racket on the left
rectangle = pygame.Rect(10, 120-10, 4, 20) # (x, y, width, height)
pygame.draw.rect(screen, pygame.Color("white"), rectangle)
```

```
# draw racket on the right
rectangle = pygame.Rect(310-4, 120-10, 4, 20) # (x, y, width, height)
pygame.draw.rect(screen, pygame.Color("white"), rectangle)
```

Animation

So now that we have some objects on the screen what about making them move? Let's start from the ball. We will move it randomly, choosing new direction every frame. To do that we need to remember the current ball position:

```
# draw a ball (color, position, radius)
x, y = 160, 120
pygame.draw.circle(screen, pygame.Color("white"), (x,y), 3)
```

On each animation frame we have to erase the ball from the screen, update its position and draw it in a new location. To do that, add the following code at the beginning of the `while` loop:

```
# animate ball
pygame.draw.circle(screen, pygame.Color("black"), (x,y), 3)
x += random.randint(-3,3)
y += random.randint(-3,3)
pygame.draw.circle(screen, pygame.Color("white"), (x,y), 3)
pygame.display.flip()
```

Did you got an error? Don't forget to import the `random` module at the beginning of the file:

```
import random
```

Our animation works but has a major drawback. We don't control its speed. On fast CPUs the game will be unplayable. Everything will happen so fast, that the player will have no chance to react. To solve this problem we will limit the animation speed to 10 frames per second.

To do that we need to create a clock object (before the game loop):

```
# clock to control the game frame rate
clock = pygame.time.Clock()
```

and use the clock to delay the animation by waiting for a tick inside the game loop:

```
# set game frame rate
clock.tick(10)
pygame.display.set_caption("PONG - {0:.2f} fps".format(clock.get_fps()))
```

The pygame way

Altough our animation procedure worked, it was far from perfect. Putting aside the problem of erased background, for some games we might need to animate not one but tens or hundreds of objects. And then our code will very quickly get messy. To reduce the complexity of objects handling in our game loop, we are going to use the pygame `Sprite` object.

Sprites

Sprites are small 2D images that are overlayed on the screen. We can easily control their position, rotation and scale. We can also define a behaviour of a sprite by implementing the `Sprite.update` method (it will be called at every frame). Finally, we can draw or clear entire groups of sprites with no more than a single function call, which greatly simplifies the animation process.

Let's start refactoring our code. Instead of drawing a disk directly on the screen, we will use the `Sprite` object to handle the ball behaviour:

```
from pygame.sprite import Sprite

class Ball(Sprite):
  """
  Handles behaviour of the ball.

  """
  def __init__(self, color, position):
    # call parent class constructor
    Sprite.__init__(self)

    # create surface
    self.image = pygame.Surface([6, 6])
    # draw filled circle
    pygame.draw.circle(self.image, pygame.Color(color), (3,3), 3)

    # get sprite bounding box
    self.rect = self.image.get_rect()
    # set sprite initial position
    self.rect.center = position

  def update(self):
    x = random.randint(-3,3)
    y = random.randint(-3,3)
    self.rect.move_ip(x, y)
```

To use our newly defined sprite, we need to create the `Ball` object. Then we add it to a special sprites group, to be able to easily draw or clear all our sprites later:

```
# create the ball sprite
ball = Ball("white", (160,120))

# list of sprites to render
sprites = pygame.sprite.RenderClear([ball])
```

Now we can replace the ball animation code in the main game loop with more general methods operating on the entire group of sprites, which will make our code much cleaner:

```
# animate sprites
sprites.update()
sprites.draw(screen)
```

```
# display screen
pygame.display.flip()

# draw background over sprites
sprites.clear(screen, background)
```

Let's continue the refactoring and define another `Sprite` object, this time to represent a racket:

```
class Racket(Sprite):
  """
  Handles behaviour of the players racket.

  """
  def __init__(self, color, position):
    Sprite.__init__(self)
    self.image = pygame.Surface([4, 20])
    self.rect = pygame.Rect(0, 0, 4, 20) # (x, y, width, height)
    pygame.draw.rect(self.image, pygame.Color(color), self.rect)
    self.rect.center = position

  def update(self):
    y = random.randint(-10,10)
    self.rect.move_ip(0, y)
```

Have you noticed that rackets have their own behaviour now?

To start using rackets in our game, we need to create the `Racket` objects and add them to the `sprites` group. We will use this as an opportunity to set more exciting colours:

```
# create two racket sprites
player1 = Racket("green", (10, 120))
player2 = Racket("orange", (310, 120))

# list of sprites to render
sprites = pygame.sprite.RenderClear([ball, player1, player2])
```

Keyboard

Random sprite movements were useful in testing the animation but what we really want is to be able to control the sprites with keyboard. Let's start with replacing the update method of the `Racket` object with two new methods controlling its movement:

```
def up(self):
  self.rect.move_ip(0, -5)

def down(self):
  self.rect.move_ip(0, 5)
```

We will use a dictionary to assign keys to the newly created methods. Let's assume that `player1` will use WASD and `player2` will use the arrow keys. Add the following code before the game loop:

```
key_map = {
  pygame.K_w: player1.up,
  pygame.K_s: player1.down,
  pygame.K_UP: player2.up,
  pygame.K_DOWN: player2.down
}
```

Now, we need to extend the event handling loop to check for key press events. Using the key map we will decide which function to call:

```
# on QUIT event, exit the main loop
if event.type == pygame.QUIT:
  running = False
# on key press
elif event.type == pygame.KEYDOWN:
  key_map[event.key]()
```

Does it work? Not really... The rackets move on a key press but not while the key is hold. Also, they can go outside the screen boundary.

To fix the first issue we will define a `velocity` field in the `Racket` object constructor:

```
# one dimensional velocity vector (vertical axis only)
self.velocity = 0
```

The `up` and `down` methods will change the racket velocity which then will be used to update the position of the racket at every frame. To prevent the racket from moving off the screen, we need to add an extra post-move position correction:

```
def up(self):
  self.velocity -= 5

def down(self):
  self.velocity += 5

def update(self):
  self.rect.move_ip(0, self.velocity)
  # move only within the screen border
  self.rect.top = max(0, self.rect.top)
  self.rect.bottom = min(240, self.rect.bottom)
```

To improve the keyboard control we need to respond to two key events: key press and key release. On key press, we change the velocity. On key release, we restore it to previous state. This way the velocity will remain non-zero while the key is hold. To define which function to call on each event we will redefine our `key_map` dictionary as follows:

```
key_map = {
  pygame.K_w: [player1.up, player1.down],
  pygame.K_s: [player1.down, player1.up],
  pygame.K_UP: [player2.up, player2.down],
  pygame.K_DOWN: [player2.down, player2.up]
}
```

We also need to adjust the event handling loop to make use of the new `key_map` and call the first function in the list on key press and the second on key release:

```
# on key press
elif event.type == pygame.KEYDOWN:
  key_map[event.key][0]()
# on key release
elif event.type == pygame.KEYUP:
  key_map[event.key][1]()
```

To really appreciate the above improvements in the key handling, let's increase the fps:

```
clock.tick(30)
```

Have you found a small bug in our code? What happens when a key that is not assigned to any function is pressed? Let's fix that:

```
# on key press
elif event.type == pygame.KEYDOWN and event.key in key_map:
  key_map[event.key][0]()
# on key release
elif event.type == pygame.KEYUP and event.key in key_map:
  key_map[event.key][1]()
```

Gameplay

As we have the main game design elements covered, let's start implementing the PONG gameplay. We want to serve the ball on key press, and make it bounce of the rackets and the top and bottom edges of the screen. We also would like to keep the score.

Ball behaviour

Before we implement the serve, we need to recall some basic vector math. Imagine we have a unit velocity vector $\vec{v} = [1, 0]$ pointing to the right. We want to rotate this vector by a random angle α in range $(0, \frac{\pi}{2})$. The coordinates of the rotated vector $\vec{w}$ can be found using trigonometric functions: $\vec{w} = [cos(\alpha), sin(\alpha)]$.

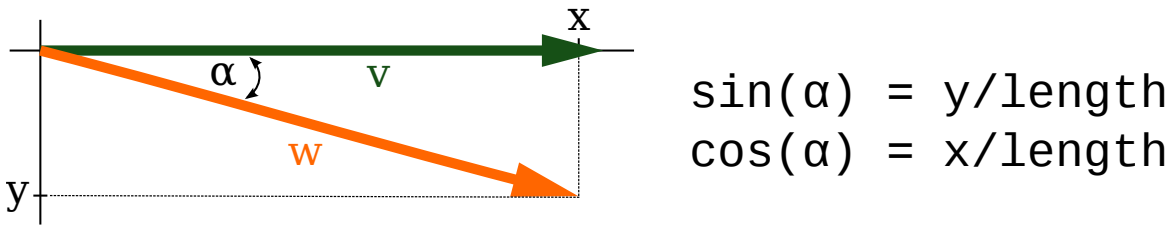


Figure 1: Rotation of the ball velocity vector.

OK, so first we need to add a `velocity` field to the `Ball` object constructor. We will also remember the starting position:

```
# two dimensional velocity vector
self.velocity = [0,0]
# remember starting point
self.start = position
```

Now we can implement a `Ball.serve` method that selects a random direction and moves the ball to the starting position (do not forget to import the `math` module!):

```
def serve(self):
  # random angle in radians (between 0 and 90 degrees)
  angle = random.uniform(0, math.pi/2)
  angle *= random.choice([-1,1])

  # choose serving side randomly
  side = random.choice([-1,1])
  # rotate the velocity vector [5, 0], flip horizontally if side < 0
  self.velocity = [side * 5 * math.cos(angle), 5 * math.sin(angle)]
  # restore the ball starting position
  self.rect.center = self.start
```

We also need a `Ball.update` method that will modify the ball position at every frame (according to the current ball's velocity).

```
def update(self):
  self.rect.move_ip(*self.velocity)
```

To be able to use the `Ball.serve` method, we need to add a key assignment to the `key_map`. We will use the space key to serve:

```
pygame.K_SPACE: [ball.serve, nop]
```

But we have not defined the `nop` function yet! As we do not need to perform any action on key release this time, this will be just a dummy "do nothing" function:

```
# no operation, dummy function
def nop():
  pass
```

Our ball is flying! That's great, but what about bouncing? Let's add the following checks and direction reversals at the end of the `Ball.update` method:

```
# bounce the ball of the top screen border
if self.rect.top < 0:
  self.velocity[1] *= -1
  self.rect.top = 1
# bounce the ball of the bottom screen border
elif self.rect.bottom > 240:
  self.velocity[1] *= -1
  self.rect.bottom = 239
```

That was easy. Can we do the same with the rackets? We will use pygame collision detection to check if the sprite bounding boxes overlap. Let's add more code to the `Ball.update` method:

```
# detect collision with the rackets
for racket in self.rackets:
  if self.rect.colliderect(racket.rect):
    # bounce the ball of the racket
    self.velocity[0] *= -1
    self.rect.x += self.velocity[0]
    # don't check both rackets
    break
```

But wait, what is `self.rackets`? That is a list of racket sprites. It must be passed to the `Ball` object constructor:

```
def __init__(self, color, position, rackets):
  (...)
  # remember racket sprites for collision check
  self.rackets = rackets
```

Also the constructor call have to be modified:

```
# create the ball sprite
ball = Ball("white", (160,120), [player1, player2])
```

At this point we could almost play the game of PONG. It would be nice, however, if we wouldn't have to count the score ourselves...

Score

To display the score we will use another pygame `Sprite` object:

```
class Score(Sprite):
  """
  Displays the game score.

  """
  def __init__(self, color, position):
    pygame.sprite.Sprite.__init__(self)
    self.color = pygame.Color(color)
    self.score = [0,0]

    self.font = pygame.font.Font(None, 36)
    self.render_text()
    self.rect = self.image.get_rect()
    self.rect.center = position

  def render_text(self):
    self.image = self.font.render("{0}     {1}".format(*self.score), True, self.color)
```

```
  def increase(self, side):
    self.score[side] += 1
    self.render_text()
```

To display the sprite, we need to initialise the `pygame.font` module, create the `Score` object and add it to the sprites list:

```
# create the score sprite
pygame.font.init()
score = Score("grey", (160,20))

# list of sprites to render
sprites = pygame.sprite.RenderClear([ball, player1, player2, score])
```

That works, but the score does not change. We need to implement the goal detection. Let's expand the `Ball.update` method again:

```
# detect goal
if self.rect.centerx < 0 or self.rect.centerx > 320:
  # add a point to the score
  side = 1 if self.rect.centerx < 0 else 0
  self.score.increase(side)
  # set the ball in the starting position
  self.rect.center = self.start
```

Right, we are missing something again... `self.score` is not defined. The `Score` sprite must be passed to the `Ball` object constructor:

```
def __init__(self, color, position, rackets, score):
  (...)
  # remember racket and score sprites
  self.rackets = rackets
  self.score = score
```

We also need to update the `Ball` constructor call:

```
# create the ball sprite
ball = Ball("white", (160,120), [player1, player2], score)
```

Awesome! The scoring works, but there is a small problem with the serve. After the goal is scored the next ball is served immediately. We want to wait for the space key, to let players catch a breath.

To do that we need to reset the ball velocity when we detect a goal (in `Ball.update`):

```
self.velocity = [0,0]
```

But there is one more problem left. Our sneaky opponent hits the space every time we are about to score a point. We need to modify the `Ball.serve` method to check if the ball is in play before serving:

```
# if the ball is already in play, do nothing
if self.velocity[0] != 0:
  return
```

The final change is code clean up. As the ball position is now restored after the goal, we can also remove this code from the `Ball.serve` method:

```
# restore the ball starting position
self.rect.center = self.start
```

Final touch

To make the game more interesting we will add more control over the ball. On each hit, a racket will pass some of its own velocity to the ball. Skilful players would be able to use that to accelerate or decelerate the ball.

To implement this, we will add one line to the `Ball.update` method (in bouncing of the racket block):

```
# pass some racket velocity to the ball
self.velocity[1] += 0.5 * racket.velocity
```

There is a major bug intentionally left in the code. There is a sweet reward for each team that will show me a working fix :)

Pygame tips

For simplicity, we didn't use all the great pygame functions in our tutorial. So here are a few tips on what you might want to use in your game. Check the pygame documentation for a complete list.

Images

Instead of drawing on a surface, you can use your own images to represent a sprite:

```
self.image = pygame.image.load("ball.png")
```

Sounds

In addition to images, you can also load sound effects or music:

```
beep = pygame.mixer.Sound("beep.ogg")
beep.play()
```

Mouse

Key press is not the only event your game can respond to. You can use mouse buttons and mouse movement too:

```
if event.type == pygame.MOUSEMOTION:
  x, y = event.pos
if event.type == pygame.MOUSEBUTTONDOWN:
  button = event.button
```

Performance

On machines without hardware accelerated 2D graphics, redrawing of the entire screen on each frame with `pygame.display.flip()` would be quite slow. To improve the performance, you might track all sprite rectangles that have been changed (so called dirty ones) and redraw just them. For that you need to use the RenderUpdates sprite group:

```
sprites = pygame.sprite.RenderUpdates([sprite1, sprite2, sprite3])

dirty_rectangles = sprites.draw(screen)
pygame.display.update(dirty_rectangles)
```

Aliens game and other examples

There are several example programs distributed with pygame. You can find their source code in `/usr/share/pyshared/pygame/examples/`. For our purpose, the most interesting of them is probably an example of a shot'em up game, in the Space Invaders style, called *Aliens*.
You can run it using: `python -m pygame.examples.aliens`