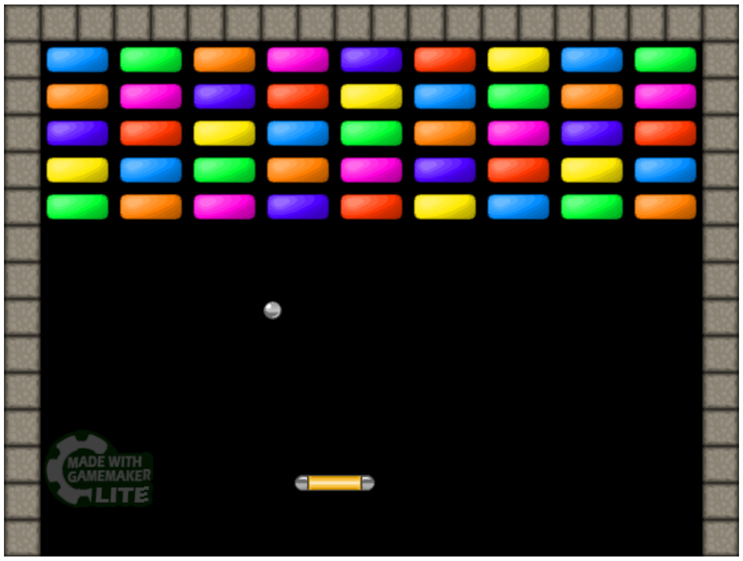
University of Surrey Widening Participation and Outreach Coding Hub

William T Scott-Jackson

Iwona Sobieraj

April 13, 2016

Welcome back after the break! This time we have a new task for you. During the following 6 meetings you will write your first game in Python. Step by step, starting form just a white screen with a moving paddle we will try to finish with a game like this:



Writing computer games seems a very difficult and complicated task. Luckily, we have some tools written by other programmers that make coding much easier. One of them is **pygame**. Pygame is a set of Python modules designed especially for writing games, making programming much easier!

Step 1: First window

We will start by just creating a new window with a white screen and the title of our game.

Create a new file, in which you will write your game. Name it as you wish, for example *myGame.py* and save it in your folder, where you will keep all the files needed for the game. Now, let's start coding!

Firstly, we need to tell our programme that we will use **pygame** and all the tools that are in this module. So before you start any code, we have to import pygame and some other modules, like this:

```
import pygame
import sys
import random
```

Ok, our programme knows what module to use! Once we have it imported, let's start it:

```
pygame.init()
```

During your long and prosperous careers as programmers, you'll often see the word "init" in some code. This is not derived from a classless colloquialism by people wearing baseball caps and shell suits. It is a short hand version for the word "initialise". In short, this line is telling Pygame to start up!

Now it's time to open a window for the game. Firstly, we have to define the size of our window in pixels. Then, we show it and give it a title.

```
# Define the size of the screen
width=500
height=500
size = width, height
# Initialise the game screen
screen = pygame.display.set_mode(size)
pygame.display.set_caption("The Title of Your Game!")
```

Time to run and see if it works!

We can see something but it just opens and closes! Let's try to keep it open as long as we want.

```
# Run the programme indefinitely!
while True:
    # Get every event that has happened
    for event in pygame.event.get():
        # If the player has pressed exit button, quit the game
        if event.type == pygame.QUIT:
            sys.exit()
```

This looks a bit complicated. But it is not at all! Firstly, we want to run our game forever, until the player does not decide to close it. That is why we put the *while* loop with an argument *True*. Then, we tell the programme to get all the so called EVENTS that happen in the game. One of such events is for example when the player presses the keyboard or the mouse. So, we can tell the programme, that if the player presses button $QUIT$, the game should close.

Let's see if the code works. Can you make the screen bigger or really small?

Step 2: Game Background

Now we have Pygame set up and running nicely, it's time to start putting your game together. It's at this moment that we should emphasise that this is YOUR game! Outside of writing code for the main systems, it's up to you to decide how your game looks, feels and plays. But we'll be here to help make that happen for you! So let's start with creating your background.

One thing you will need to understand is how computers put colour pixels on the screen. Pixels are made up of three different values (called channels) Red, Green and Blue. These three colours are mixed together to create different colour pixels. Using the code snippet below to create your own custom background colour. You may need to experiment a little bit to find the colour that is just right.

```
redVal = 0
greenVal = 0
blueVal = 0
BG_Colour = redVal, greenVal, blueVal
```

When you think you have a background color you like, add the following code to your game's loop:

```
# Run the programme indefinitely!
```

```
while True:
    # Get every event that has happened
    for event in pygame.event.get():
        # If the player has pressed exit button, quit the game
        if event.type == pygame.QUIT:
            sys.exit()
    screen.fill(BG_Colour)
    pygame.display.update()
```

Try running your game to see if the background colour is to your liking. If you aren't quite happy with the background colour, spend some more time experimenting with different colour values. You can always open up an image editing program and look at the colour picker there which will tell you the numbers you need (ask a demonstrator if you aren't sure).

Step 3: Player Character

Here comes the tricky part, making your playable paddle and controlling them with keyboard inputs. There are several aspects of the design of the paddle that you are in charge of and it's up to you to get make it how you want it. To make the process of writing this code easier, we're going to delve into the world of Object Oriented Programming (OOP). For now, consider this as a way of writing code where the user can define their own "objects" and functions to control this objects.

At the top of your source code (below the import statements):

```
# Class that defines the attributes of the in game paddle
class Paddle:
    def __init__(self, paddleWidth, paddleHeight, paddleSpeed, paddleColour)
    :
        self.width = paddleWidth
        self.height = paddleHeight
        self.colour = paddleColour
        self.speed = paddleSpeed
        self.centreX = 0
        self.centreY = 0
        self.rect = pygame.Rect(self.centreX, self.centreY, self.width, self
    .height)
```

Once you have written this block of code, add the following section where you have written the code to set up the screen size etc.

```
# Define the colour of the paddle
paddleRed = 0
paddleGreen = 0
paddleBlue = 0
paddleColour = paddleRed, paddleGreen, paddleBlue

# Define the size and the speed of the paddle
paddleSpeed = 4
paddleWidth = 100
paddleHeight = 15

# Create a Paddle object
paddle = Paddle(paddleWidth, paddleHeight, paddleSpeed, paddleColour)

# Define starting positions of the paddle
paddle.centreX = width * 0.5
paddle.centreY = height * 0.8

# Initialise the "player character" as a rectangle of specified position and
     dimensions
paddle.rect = pygame.Rect(paddle.centreX, paddle.centreY, paddle.width,
    paddle.height)

```

```
# Define in which direction is the Paddle going to move now
paddleDirection = 0
```

Let's try to run the code and see the Paddle on the screen. Nothing there, right? That is because we have not told our programme to print the Paddle yet. To do this we need to add the following line just before updating the display.

```
# Draw the Paddle on the screen
pygame.draw.rect(screen, paddle.colour, paddle.rect, 0)
```

Do you see it? Now, try to experiment with the colour of the Paddle, its position on the screen and its size. Can you draw it outside the screen? And how about changing the shape of our Paddle? Try using a function: *pygame.draw.ellipse()* instead of *pygame.draw.rect()*.

Step 3: Controlling the Player Character

Now that we have the Paddle on the screen we need to be able to control it with the keyboard. To do that, firstly we want to be able to detect that the player pressed the keyboard, that is detect an EVENT (remember? if not, look at Step 1 again!). We were already detecting, when the player has clicked on the QUIT button. Just below, let's add the detection of a pressed key:

```
# Was there a key pressed down on the keyboard?
    if event.type == pygame.KEYDOWN:
        # Get the details of the keys that were pressed
        keys = pygame.key.get_pressed()
```

Fine, now under the variable *keys* we have a list of keys that were pressed. We want to check if any of these keys was a right or left arrow, and if so, we want to tell the paddle to move in the right direction. If we press the left arrow key, the Paddle should move some pixels left. That is, we need a negative value of the direction. To move it right we need to add some pixels to its position - the value of the direction is positive. To do so, we add the following lines:

```
            # Which key was pressed? Set the direction accordingly
            if keys[pygame.K_LEFT]:
                direction = -paddle.speed
            elif keys[pygame.K_RIGHT]:
                direction = paddle.speed
```

Now that we know in which direction we want to move the Paddle, we just need a function that will actually do that. How about using a function *move()* just before drawing the screen and the Paddle?

```
paddle.rect = paddle.rect.move(paddleDirection, 0)
```

Let's try if it works!

Hmmm... Seems that the Paddle moves not only one we press the left or right key. Do you know why? Think what is the value of *direction* every time we press a button. Does it change? If so, how? How can we fix it?

We need to tell our programme that if a different key than left or right arrows has benn pressed, the direction is zero again and we do not want to add any pixels to the position of our Paddle! So, the complete code for controling the Paddle should look like this:

```
        # Was there a key pressed down on the keyboard?
        if event.type == pygame.KEYDOWN:

            # Get the details of the keys that were pressed
            keys = pygame.key.get_pressed()

            # Which key was pressed? Set the direction accordingly
            if keys[pygame.K_LEFT]:
                paddleDirection = -paddle.speed
            elif keys[pygame.K_RIGHT]:
                paddleDirection = paddle.speed
        else:
            paddleDirection = 0

        paddle.rect = paddle.rect.move(paddleDirection, 0)
```

Uff, it seems to be working. Now we can play a bit with our Player Character. Can you make the Paddle go up and down? Or perhaps move in diagonal? And what if the player presses *Esc* key? Write a line of code, that will read this event and quit the game. Tip: *Esc* key is denoted as *ESCAPE* in pygame.

To keep things smooth, we need to lock the "refresh rate" of the game. The refresh rate is defined by how many times per second the computer repaints the graphics of the game back onto the screen. You sometimes hear this referred to as the "frame rate" of the game and a hotly debated topic among gamers. We need to set up a "clock" which keeps track of how much time has passed and only let the lock tick a certain amount of times per second. Add this line of code **BEFORE** the while loop:

```
clock = pygame.time.Clock();
```

And then add this code at the start of the while loop:

```
clock.tick(60);
```

With this we are telling the game to refresh 60 times a second. This equates to a frame rate of 60 frames per second (FPS). We chose this value as it is a nice smooth frame rate which doesn't appear too stuttered to the human eye.

Why not prove it? Change the value from 60 to 30, and then from 30 to 15. What happens? Have the person next to you run the code with a different frame rate and see if you can notice the difference. Also what happens to the speed of your paddle? Why do you think this is happening?

Step 4: Where is my Paddle?

Now that we taought our Paddle to move around, we are actually able to get it out of the screen!If we just move it right enough times it will finally disappear. That is not what we really want, is it? That is the task for you! Try to think how can we detect when the Paddle has reached the end of the screen and wher should we move it to stop it from leaving the screen. Good luck!