Coding One: Pong Game for two players

Table of Content:

Identifying the Problem; Basic Setup for the Game

P.S. just in case there is a problem with my .py file I also uploaded it to GitHub: https://github.com/AndriiArtemenko3/PongGame-Andrii_Artemenko-

For the first unit of Coding One we had a task to create a two-player game, with the limitation that the game has to be written in Python and pre-made game engines cannot be used. I use PyGame library for my game which is a library that includes computer graphics and sound libraries for Python. I created the code on MacOS and used Visual Studio Code as a text editor for the game.

The game I made is a classic Pong game with two players; I designed the game specifically so that two people can play on one PC or laptop, the player one being on the left side of the screen and using key “1” to move his paddle down and key “2” to move his paddle up, the player number two is sitting on the right and using key “9” to move the right paddle down and key “0” to move up. Therefore the game is balanced and no players get an unfair advantage.

To create a basic setup for my game I have to write a couple lines of code that will import the library, set up dimensions of the window, set up the “speed” of the game or so-called FPS (frames per second), enable the players to close the window etc.

```
1   # First I have to import PyGame lybrary. I first had to install it via terminal because it wasn't working with the Visual Studio
2   import pygame
3   from sys import exit # To fix the error -> pygame.error video system not initialized
4   print(pygame.ver)
5
6   # That is the command which basically "starts" the pygame
7   pygame.init()
8   # Setting dimensions for the game
9   screen_width = 1200
10  screen_height = 900
11  screen = pygame.display.set_mode((screen_width,screen_height))
12  pygame.display.set_caption('GameO1') #Set the caption of the window e.g. name the game in the system
13  clock = pygame.time.Clock()
```

Line 2 -> imports pygame library; order of the lines is important so if we want to import anything for our project it always has to be at the beginning before we actually start to write the code. If the import is successful we will see a following message in the terminal

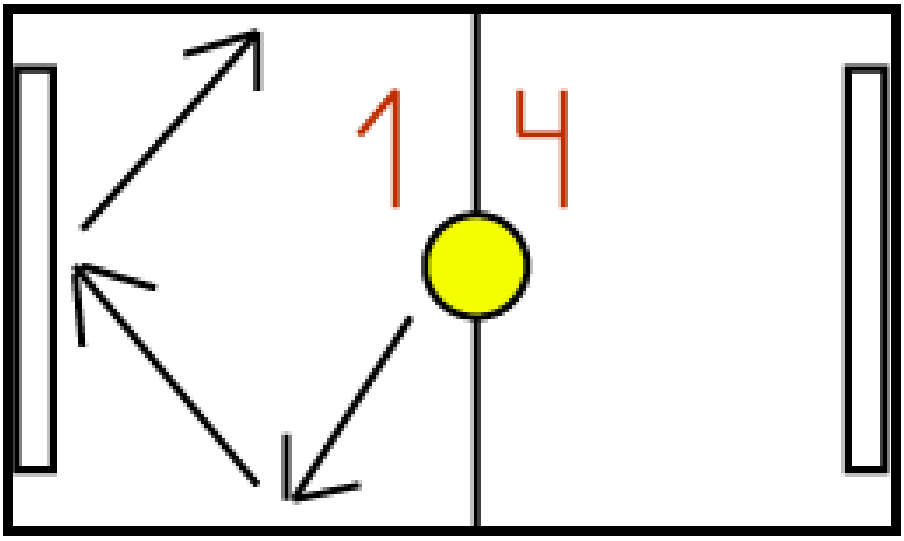
```
2024-01-20 18:45:22.347 Python[1988:39593] WARNING: Secure coding is not enabled for restorable state! Enable secure coding by implementing
 NSApplicationDelegate.applicationSupportsSecureRestorableState: and returning YES.
andriiartemenko@Andriis-MBP V01.GamePython %  cd /Users/andriiartemenko/Desktop/V01.GamePython ; /usr/bin/env /usr/local/bin/python3 /Users
/andriiartemenko/.vscode/extensions/ms-python.python-2023.22.1/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 50288 -- /User
s/andriiartemenko/Desktop/V01.GamePython/new.py
pygame 2.5.2 (SDL 2.28.3, Python 3.12.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
andriiartemenko@Andriis-MBP V01.GamePython %
```

Line 3 -> When I was working on the exit button of the window I got an error and my game was crashing so I had to import exit from sys library as well to fix the error.
Line 4 -> This line is not necessary for the functionality of the game but it just prints out the current version of the Pygame used in the terminal;
Line 7 -> This is important command which basically activates the Pygame functionality
Lines 9 to 11 -> Setting up our dimensions for the game, in pixels. I made them into variables so that it is easier to manipulate them later on in the code
Line 12 -> It is optional but it just sets the name of the game at the top of the window
Line 13 -> gets the clock command which will set the frames per second at which our game will update

Game Design

What I learned from this project is that PyGame has its own Game Design system.
The fundament of the game is the Display Surface which is the window that player/s see. All the elements will therefore be placed on the display surface. Also, it is possible to add multiple surfaces on the display surface, but I do not need it for my current project.

What I needed for the visual part of my game is essentially a background color; a division line in the middle; two paddles for each player; a ball; score count for each player; It all has to be animated and logic should be implemented to the game but I will talk about it later.



So I learned that the game components like ball and paddles should be first constructed as rectangles (pygame.Rect(x, y, width, height).

```
16  #Rectangles
17  #pygame.Rect(x,y,width,height)
18  ball = pygame.Rect(screen_width/2-10, screen_height/2-10, 20, 20)
19  player1 = pygame.Rect(screen_width-100, screen_height/2-50, 10, 100)
20  player2 = pygame.Rect(100, screen_height/2-50, 10, 100)
```

And then under the while True loop where our game is running and updating I set the visual appearance of my rectangles.

```
79      pygame.draw.rect(screen, "white", player1)
80      pygame.draw.rect(screen, "white", player2)
81      pygame.draw.circle(screen, "yellow", ball.center, 10) #important to specify the radius here if we making a circle
```

Note: it is important to set the radius parameter if I want to make a circle, otherwise the game won't work.
For all the colors, fonts, etc. I used official documentation on pygame website (pygame.org, n/a)
For my game I kept it simple and specified the colors I want to assign to my rectangles, the surface which is my screen and radius for the ball.

I also made a line at the center which separates two sides of the Pong field.
There is no need to create a pygame.Rect for the line because it is a purely visual element that has no interaction with the other elements.

```
82      pygame.draw.aaline(screen, 'azure', (screen_width/2, 0), (screen_width/2, screen_height))
```

Building Logic and Data Structures; Creating “Physics” of the Game

So I learned that essentially adding movement to the game objects is possible through manipulating their positions on the grid of coordinates; and for player movement we can assign certain keys that will enable players to move.

I did not make my own dictionary for the keys because there is a pre-installed dictionary in PyGame that allowed me to assign certain keys and create a logical structure with if statements that basically states: if I press key “1” the left paddle moves down, if I press key “2” the left paddle moves up, and so on.
I used the official documentation for this as well: https://www.pygame.org/docs/ref/key.html
I also had to set up a limit of movement with if condition so that the paddles don’t go off the screen and only move within the visible area for both players.

```
33  # I made a loop to make the game running and all the visual content of the game should be in this loop
34  while True:
35      key_pressed = pygame.key.get_pressed() # I created a variable called
36  # Creating the "movement" logic of both players using the key pressed to shift the position of rectangles accordingly
37      if key_pressed[pygame.K_0]:
38          if player1.top > 0: # I have to set up a limit for players like this so that the rectangle doesn't go off the screen
39              player1.top -=3
40      if key_pressed[pygame.K_9]:
41          if player1.bottom < screen_height:
42              player1.bottom +=3
43      if key_pressed[pygame.K_2]:
44          if player2.top > 0:
45              player2.top -=3
46      if key_pressed[pygame.K_1]:
47          if player2.bottom < screen_height:
48              player2.bottom +=3
```

By adjusting the += x, I can change the move speed of both players. I found experimentally that 3 works the best - not too slow but not too fast as well. If I had a smaller or bigger game screen it would be worth it to check that move speed again and adjust accordingly.
The logic behind the chosen keys was previously explained in the Basic Setup section.

That I had to make the ball move. First I created variables that I will later use to create physics and for collision detection as well:

```
26  ball_x_speed = 2.75
27  ball_y_speed = 2.75
```

And in my while loop I “link” them to the coordinates of the ball rect, so that ball_x_speed is linked to ball.x axis and ball_y_speed is linked to ball.y axis. As I understand it every frame ball rect is moving by its coefficient set in ball_x_speed and y speed, so by changing those coefficients I can make the ball move faster or slower. I figured out experimentally that 2.75 is optimal speed because 2 seems too slow but 3 is too fast, so the optimal speed lies within that range.

```
78      ball.x += ball_x_speed
79      ball.y += ball_y_speed
```

Then I had to prevent the ball from going outside of the visible window and in a way programmed another collision -> when ball reaches the top or bottom limit of height it then changes the direction (by setting the speed to -) at the same rate as its speed:

```
    if ball.y >= screen_height:
        ball_y_speed -= 2.75
    if ball.y <= 0:
        ball_y_speed = 2.75
    if ball.x <= 0:
        player1_score += 1
        ball.center = (screen_width/2, screen_height/2)
        ball_x_speed = random.choice([2.75,-2.75]) #why do we need randomisation here? Basically if not introduced, ba
        ball_y_speed = random.choice([2.75,-2.75])
    if ball.x >= screen_width:
        player2_score += 1
        ball.center = (screen_width/2, screen_height/2)
        ball_x_speed = random.choice([2.75,-2.75])
        ball_y_speed = random.choice([2.75,-2.75])
        # problem - ball is too slow after it gets back to the center
        # problem solved - in random.choice([]) the values should ideally match the speed of the ball, so in my case i
```

And when the ball reaches the “limits” of the window by the x axis, so if it bounces to one of the sides of the players 3 things happen 1) the player on the opposite side of which the ball hit scores 1 point 2) ball gets “respawned” to the center of the surface screen 3) the ball’s direction is randomized

The third parameter actually was what I consider to be a bug, because the ball typically would just go to the right side all the time which gives the left player an advantage, so I had to fix it. I found a good solution in this video (Baraltech, 2021) where the author is using randomization to make the ball’s movement less predictable. So I had to import random for my python game as well in order for this thing to work. I also realized that random.choice([x1, x2]) should ideally match or be close to the ball speed because if you set it to a different parameters the ball may be too slow/too fast, it was set to 1 for me at first and I couldn’t realize what is the problem until I understood how the randomization works.

For the score system I referred to two video tutorials (Clear Code, 2020) and (Baraltech, 2021) and created my score counter for the game. So first I specified the font variable by assigning a pre-installed Arial font in pygame.font.SysFont

```
    font = pygame.font.SysFont("arial", 30)
```

I set the variables player1_score and player_2 zero and set their initial values to 0 as there should be 0 points for each player at the beginning

```
    player1_score = 0
    player2_score = 0
```

Then again in my code portion where I coded the ball logics and movements, I set up an algorithm that when a ball hits the right or the left wall the player opposite to the wall which the

ball hits will score 1 point. So now what is only left is to visually represent it. I go back to my while loop and visualize the scores using my font variable as well

```
78      player1_score_text = font.render(str(player1_score), True, "azure")
79      player2_score_text = font.render(str(player2_score), True, "azure")
```

Note: player1 and player 2 score variables has to be put in string str(), otherwise we have a bug and the game is crushing

Collision Detection

So to assign collisions I will use colliderect(), which as I assume is not the only way to detect collisions but I find it to be the easiest way for my purposes.

```
71      if ball.colliderect(player1) or ball.colliderect(player2):
72          ball_x_speed *= -1.05
73          # problem -> -2.75 reverse speed is way too high, try make it 50% less . upd -> 1.32 is also too much, try -1 now. upd: -1
```

So here with my if statement colliderect() I specify that if the ball collides with player1 or player2 its speed should change to -1.05 coefficient which basically works as if a ball would bounce off from the physical object. I had a bit of experimentation with the move speed here again and initially i did not realize that it will increase the speed by too much if i set a high coefficient, so i set -2,75 and it was unplayable, i had to change it to lower number and experimentally found -1.05 to be the best value because then on -1 it can get “stuck” sometimes from certain angles but with -1.05 it seems to happen less often and the ball bounces off smoother.

Areas to improve

I also want to talk about things that I would improve in the future.

1) Game speed and playing sustainability
   The game is playable now but I see that it glitches sometimes and I had it crash once which I assume is because a) PyGame is not ideal for let say commercial games, it is more of a great tool for students like myself to improve and train my python skills but the productivity is not optimal b) I probably have areas in my code that I could make shorter/optimize so that the game runs smoother and faster but as it works right now I’d rather keep it like it is, given that it is my first ever game made on python, I am quite happy about it, but future me would find a way to reduce the noise/shorten the lines of code
2) Visual appearance
   Needless to say my visual game design is rather simple and that should be like that, but maybe adding some textures/different colors, like accent colors, outlines etc., would be a good idea. Once again I am afraid that it can overcomplicate the code and game will become quite slow, so I am not sure about what is the limit of visual improvement here
3) Adding special effects and sounds
   I think it always makes the game more engaging when there is sound added to it, as well as visual effects for example when the ball collides with the walls or player paddles I

could add sparks etc., looking forward to the sound and image processing unit as it might give me some insights on how to do it

Study Resourced used

For my project I didn't copy-paste any of the code, instead I watched some videos that gave me a) a general idea of how pygame works and what are its possibilities b) examples of how Pong game can be built. I made sure that I fully understand what every line of code does and I can explain it with my own words, which I believe I show quite well in this README documents or in the comments in the code itself. In some situations like with the score counter I used a "pre-made" solution from the video tutorial but almost everything else is built on my own logic and experiments. However it is obvious that I had to first study pygame specifics and see examples of how a similar game can be done, so I post a full list of resources that I used for my prior education on pygame and videos or resources I was looking up to or referring to while working on the project. :

1. Clear Code. (2021) 'The ultimate introduction to Pygame' Youtube, Available at: https://www.youtube.com/watch?v=AY9MnQ4x3zk&t=1473s (Accessed 20 January 2024)
2. Clear Code (2019) 'Learning Pygame by making Pong' Youtube, Available at: https://www.youtube.com/watch?v=Qf3-aDXG8q4&t=1311s (Accessed 20 January 2024)
3. Clear Code (2020) 'Learning Pygame by making Pong part 2: Adding the score and countdown timer' Youtube, Available at: https://www.youtube.com/watch?v=E4Ih9mpn5tk&t=30s (Accessed 20 January 2024)
4. Baraltech (2021) 'How to make Pong in Python and Pygame' Youtube, Available at: https://www.youtube.com/watch?v=iSZXroL4apY&t=934s (Accessed 20 January 2024)
5. Tech with Tim (2017) 'Pygame Tutorial #1 - Basic Movement and Key presses' Youtube, available at: https://www.youtube.com/watch?v=i6xMBig-pP4&list=PLzMcBGfZo4-lp3jAExUCewBfMx3UZFkh5 (Accessed 20 January 2024)
6. Tech with Tim (2018) 'Pygame Tutorial #7 - Collision and Hit Boxes' Youtube, Available at: https://www.youtube.com/watch?v=1aGuhUFwvXA&list=PLzMcBGfZo4-lp3jAExUCewBfMx3UZFkh5&index=7 (Accessed 20 January 2024)
7. Pygame.org (n/a) Color List, Available at: https://www.pygame.org/docs/ref/color_list.html (Accessed 20 January 2024)
8. Pygame.org (n/a) pygame.key, Available at: https://www.pygame.org/docs/ref/key.html (Accessed 20 January 2024)
9. Pygame.org (n/a) pygame.font, Available at: https://www.pygame.org/docs/ref/font.html (Accessed 20 January 2024)