PyGame - Game Design and Implementation with Python

CS-489 | Dr. Wang

May 2, 2022

Kyle Forker

Table of Contents

1. Abstract

This research project covers the use of PyGame, which is a set of Python modules. This project implements PyGame alongside previous knowledge of the programming language Python. This showcases rudimentary games that have been recreated from scratch based off the research during the current semester. The project ranges from basic working the way up to the use of basic AI that a player can play against. The implementation of each game has used the compounded knowledge of the previous games created. This has allowed for a progression of research throughout the term of the project.

2. Games

a. <u>The Basics</u>

This was the start of the first term. Term one was started off by learned and developing the basics of PyGame and how the modules work and interact with different elements of the programming language Python. This first term was the foundation for all the terms and work to come throughout the entire semester. The existing knowledge of Python existing allowed for more organized and readable code throughout the entire project. This project allowed for skills to be built and bolstered within the Python programming language.

The basics of PyGame involve setting up a game window, using while loops to allow the game to continually run, and integrating functions for neater and tidier code. The use of classes with Python is also present when dealing with PyGame for organization and functionality.

PyGame allowed for the knowledge of object oriented programming to be explored with python as well. PyGame uses object oriented programming to create different objects allowing the player to interact with many things in any given game. Python is not known for object oriented programming unlike Java and many other languages as Python is mainly a scripting language.

All of these basics were used to start the creation of the first game, Snake. This game will be discussed later in the paper detailing the ins and outs of how the game functions as well as how the player interacts within the game.

b. <u>Snake</u>

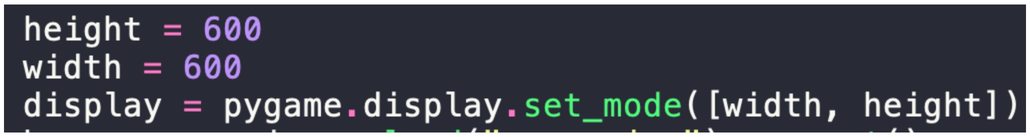
i. Overview

Snake is a popular retro game in which the player must navigate a “snake” with inputs to collect “food”. When the player collects the food the snake grows in length. This progresses and the length of the snake compounds until the player runs into themselves or attempts to go out of bounds.

ii. How it Works

1. Game Window

Snake is arguably one of the most basic example games that a person can make with PyGame. This game was used to learn and better understand the foundations and workings with PyGame and the modules it has. To start off below in Figure 1.0, is how the developer sets up the PyGame window itself. The developer can use variables or just insert the numbers for the height and width of the game window as a tuple into the function called *set_mode*. This function sets the height and width of the display window which will be used to print out models, players, and other objects that are programmed into the game. This is the display that the player will see when they are actually playing the game.

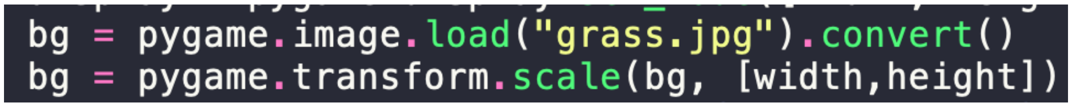


```
height = 600
width = 600
display = pygame.display.set_mode([width, height])
```

Figure 1.0 (Snake Source Code)

2. Background Images

Once the display window for the game has been set up. The developer can program in a background image or what is known as a sprite. This image can be anything the developer would like; all the developer has to do is enter the path for the destination file they are wanting to use. Below in Figure 1.1, is how an image is constructed and scaled within PyGame. First the image must be loaded and converted to what is known as a sprite. The developer may also use a rectangle object, but for the background that is stationary, the sprite is the best option. Once the image is loaded and converted the image can be transformed or stretched to the dimensions of the game window.

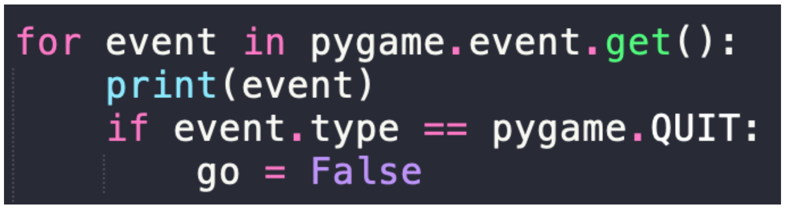


```
bg = pygame.image.load("grass.jpg").convert()
bg = pygame.transform.scale(bg, [width,height])
```

Figure 1.1 (Snake Source Code)

3. Event Triggers

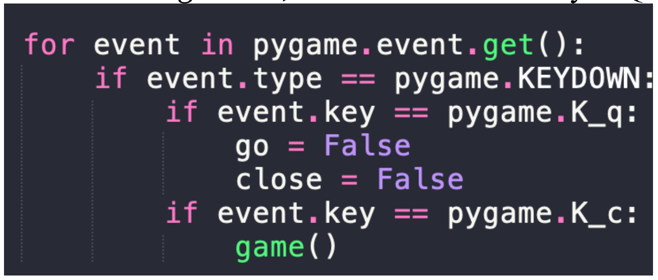
Below, in Figure 1.2, is the must use for loop with if statement in the games main function. Every PyGame script runs with a while loop. This while loop can be nested within a main function to allow for cleaner, organized, well-practiced code. In Figure 1.2, the developer needs this for loop because this allows the game to be closed or quit by the player. This for loop checks if a certain event has happened and if it has it will execute an action. This type of event loop has other functionalities including executing anything the developer would want off some sort of keyboard event. For the sake of Figure 1.2, this will close the game upon triggering.



```
for event in pygame.event.get():
    print(event)
    if event.type == pygame.QUIT:
        go = False
```

Figure 1.2 (Snake Source Code)

Continuing, this is how the event triggers can be integrated for the use of keyboard interactions. Below in Figure 1.3, the even trigger checks for a event.type and this event type it looks for is a keyboard input. In the instance of Figure 1.3, this checks for the keys “Q” or “C” to be hit.



```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_q:
            go = False
            close = False
        if event.key == pygame.K_c:
            game()
```

Figure 1.3 (Snake Source Code)

iii. Final Product

The final product of snake can be found below in Figure 1.4. There is a score counter so the player can keep track of the score they are currently at. The player is the purple-colored box and the item the player must pick up is the green colored box. These green boxes spawn randomly around the game window and the user must pick up as many as the player can before they either hit their own snake or the borders of the game window.

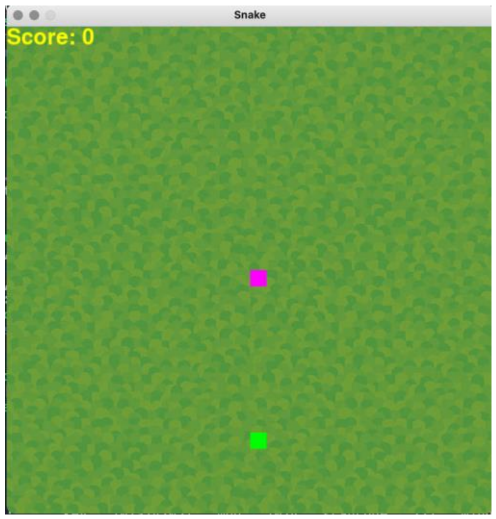


Figure 1.4 (Snake Game Window)

c. Flappy Bird

i. Overview

Flappy bird was a popular mobile game back around 2015. This game required the player to tap the screen to get through a series of obstacles. The goal of the game was to reach as far as the player could without colliding with the obstacles. This game was incorporated into this research project for the purpose of learning “infinite” game length. As well, this was incorporated to the research project to allowing for the learning and development of collisions with moving objects as well as creating and managing moving objects and their coordinates on the game.

ii. How it Works

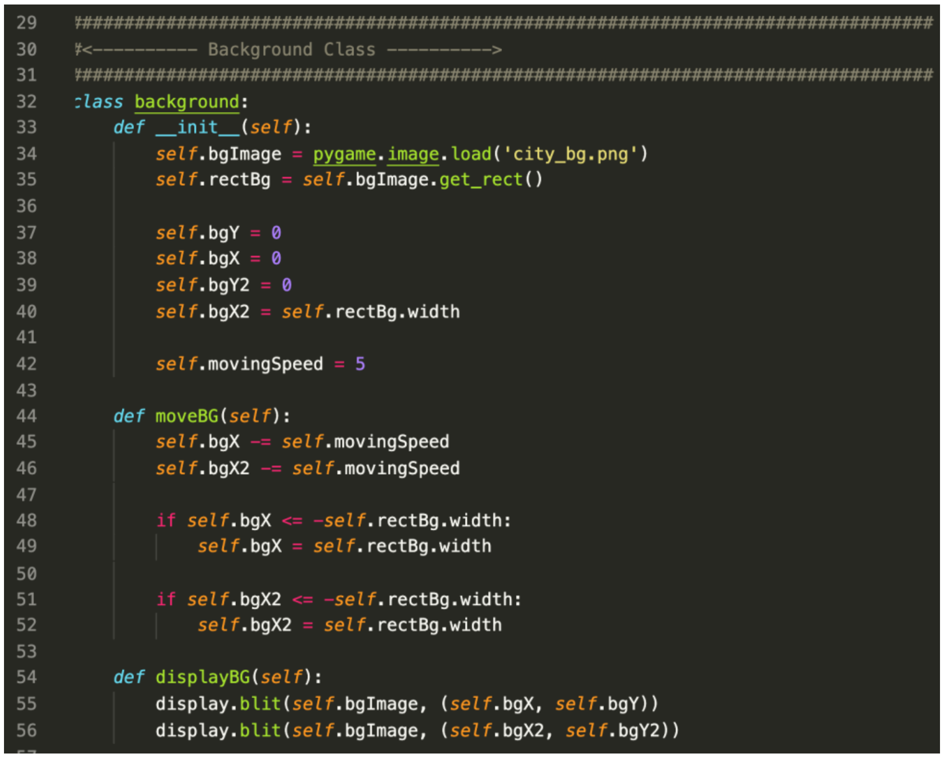
1. Scrolling Background Class

Instead of just making a game window that is infinitely long the developer can use a trick to make the game seem infinitely long. The way to accomplish this is to move the background image across the screen at a constant rate. Once the background image gets to a point where it would be going off the screen leaving the game window blank, it must teleport or be readjusted back to the starting position. This sounds complicated, but in reality, the use of if statements can check the position of the background just using its image size compared to the size of the game window itself.

In this game this was accomplished by creating a class for the background in the code. This class allows for variables to be contained and modified with every function inside the class. This also allows the code to call the class and run all of the functions at once. Below in Figure 2.0 the class starts off with an initialization function. This is used to initialize the variables that the class will use. The use of self gives the variables precedence over all the functions in the entire class. This allows all the functions to edit and use these variables without having to constant return values and feed all sorts of values throughout the functions that are being used.

The moveBG function is what allows the background to have a scrolling or infinite look to the game. This function checks the x-coordinate of the left and right of the background. If the right coordinate is about to push onto the game window leaving a blank space, the background then gets “teleported” or moved back to the original position. This allows for a seamless transition of the background to add an “infinite” look to the game.

The final function, displayBG, renders the image that was previously loaded, the background image that the developer is looking to use, to the game window so the player is able to see it.



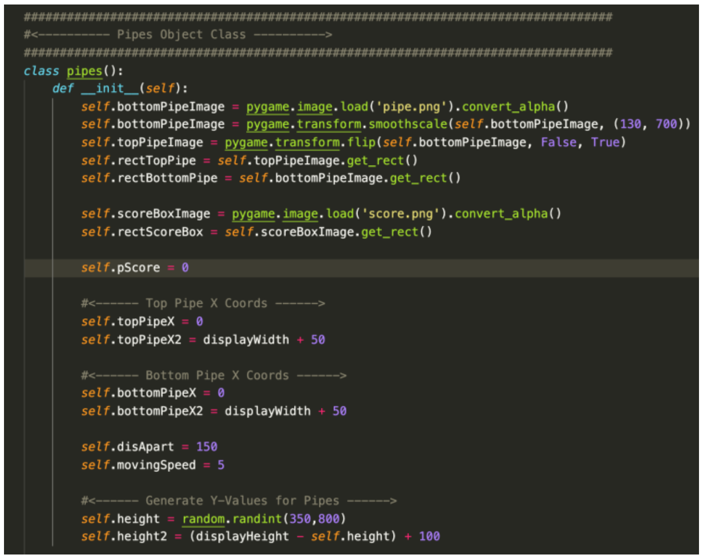
```
##############################################################################
#<---------- Background Class ---------->
##############################################################################
class background:
    def __init__(self):
        self.bgImage = pygame.image.load('city_bg.png')
        self.rectBg = self.bgImage.get_rect()

        self.bgY = 0
        self.bgX = 0
        self.bgY2 = 0
        self.bgX2 = self.rectBg.width

        self.movingSpeed = 5

    def moveBG(self):
        self.bgX -= self.movingSpeed
        self.bgX2 -= self.movingSpeed

        if self.bgX <= -self.rectBg.width:
            self.bgX = self.rectBg.width

        if self.bgX2 <= -self.rectBg.width:
            self.bgX2 = self.rectBg.width

    def displayBG(self):
        display.blit(self.bgImage, (self.bgX, self.bgY))
        display.blit(self.bgImage, (self.bgX2, self.bgY2))
```

Figure 2.0 (Flappy Bird Source Code)

2. “Pipes” or Obstacles Class

This class is the obstacles class. This class handles the generation, locations, and collisions of the pipes or better known as the obstacles within the game. These pipes have a randomly generated y-value, so the player has a challenge rather than just having to go straight for the entire game. The pipes also must move at the same speed and time as the background, so the game looks and feels fluid to the player. This is done almost identical to how the background was moved for the background class. The only trick for these is that the developer needs to manage the top obstacle and the bottom obstacle simultaneously.

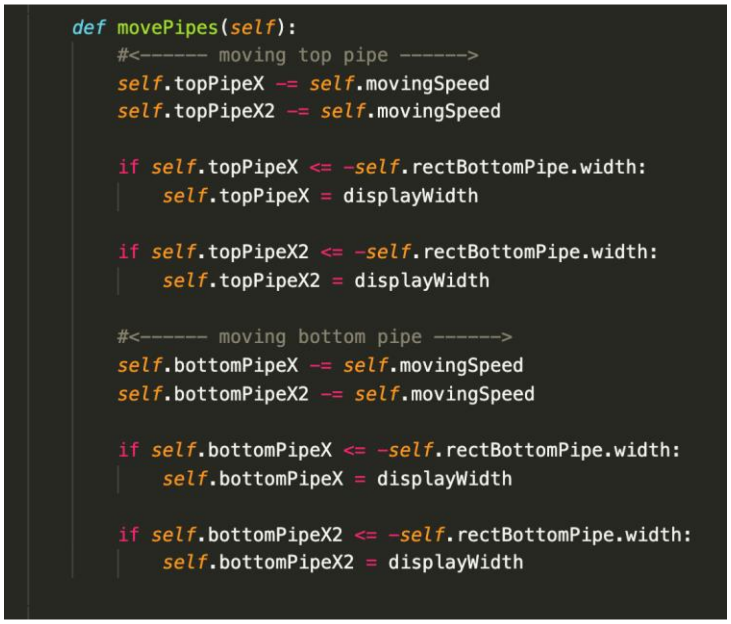
Below in Figure 2.1 is the initialization function in the obstacles class. This is used to once again initialize all the variables that the class will use and interact with throughout all the function incapsulated in it. The very important variables that are being defined and initialized are the “self.height” and “self.height2” variables. These variables generate the random y-value for each of the obstacles while also keeping a set distance between the top obstacle and the bottom obstacle.



```
##################################################################################
#<---------- Pipes Object Class ---------->
##################################################################################
class pipes():
    def __init__(self):
        self.bottomPipeImage = pygame.image.load('pipe.png').convert_alpha()
        self.bottomPipeImage = pygame.transform.smoothscale(self.bottomPipeImage, (130, 700))
        self.topPipeImage = pygame.transform.flip(self.bottomPipeImage, False, True)
        self.rectTopPipe = self.topPipeImage.get_rect()
        self.rectBottomPipe = self.bottomPipeImage.get_rect()

        self.scoreBoxImage = pygame.image.load('score.png').convert_alpha()
        self.rectScoreBox = self.scoreBoxImage.get_rect()

        self.pScore = 0

        #<------ Top Pipe X Coords ------>
        self.topPipeX = 0
        self.topPipeX2 = displayWidth + 50

        #<------ Bottom Pipe X Coords ------>
        self.bottomPipeX = 0
        self.bottomPipeX2 = displayWidth + 50

        self.disApart = 150
        self.movingSpeed = 5

        #<------ Generate Y-Values for Pipes ------>
        self.height = random.randint(350,800)
        self.height2 = (displayHeight - self.height) + 100
```

Figure 2.1 (Flappy Bird Source Code)

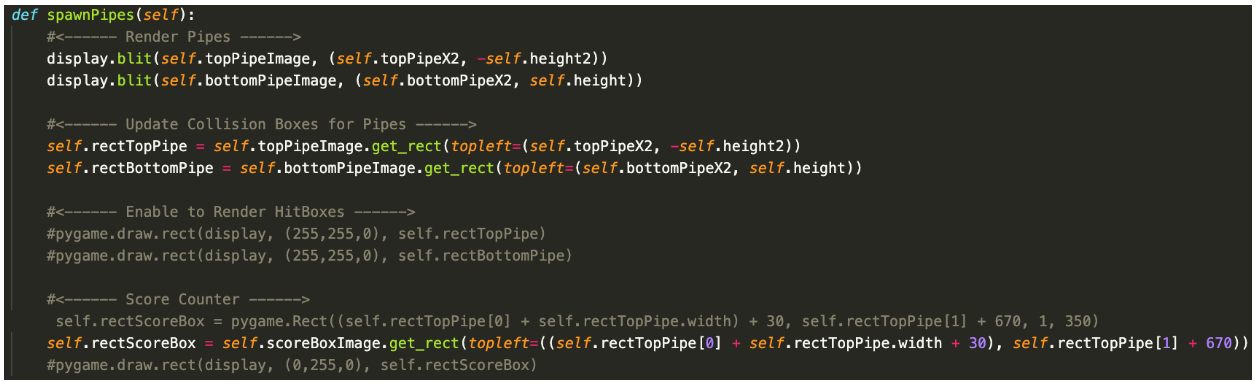
The movePipes function, which is picture in Figure 2.2 below, allows both the top and bottom obstacle to move at the same speed and same area. This keeps the top and bottom obstacle in sync and allows for the gameplay to be fluid for the player. Earlier in this report in the background class section, this exact strategy was used to move the background across the screen. Once again, this is used here.



```
    def movePipes(self):
        #<------ moving top pipe ------>
        self.topPipeX -= self.movingSpeed
        self.topPipeX2 -= self.movingSpeed

        if self.topPipeX <= -self.rectBottomPipe.width:
            self.topPipeX = displayWidth

        if self.topPipeX2 <= -self.rectBottomPipe.width:
            self.topPipeX2 = displayWidth

        #<------ moving bottom pipe ------>
        self.bottomPipeX -= self.movingSpeed
        self.bottomPipeX2 -= self.movingSpeed

        if self.bottomPipeX <= -self.rectBottomPipe.width:
            self.bottomPipeX = displayWidth

        if self.bottomPipeX2 <= -self.rectBottomPipe.width:
            self.bottomPipeX2 = displayWidth
```

Figure 2.2 (Flappy Bird Source Code)

Continuing, the spawnPipes() function is the function that allows for the creation of the hitbox and displaying of the pipes or obstacles within the game. Instead of just displaying the image the game must also have a collision box generated for the obstacles. Without this, the game would never be able to figure out if the player had collided with an obstacle in any efficient manner. This function also creates an invisible hitbox that the player must hit within the game. This hitbox generates at the exiting portion of any generated obstacle. This allows the game, later on, to be able to know when to give the player a point for clearing the obstacle and when to not.



```python
def spawnPipes(self):
    #<------ Render Pipes ------>
    display.blit(self.topPipeImage, (self.topPipeX2, -self.height2))
    display.blit(self.bottomPipeImage, (self.bottomPipeX2, self.height))

    #<------ Update Collision Boxes for Pipes ------>
    self.rectTopPipe = self.topPipeImage.get_rect(topleft=(self.topPipeX2, -self.height2))
    self.rectBottomPipe = self.bottomPipeImage.get_rect(topleft=(self.bottomPipeX2, self.height))

    #<------ Enable to Render HitBoxes ------>
    #pygame.draw.rect(display, (255,255,0), self.rectTopPipe)
    #pygame.draw.rect(display, (255,255,0), self.rectBottomPipe)

    #<------ Score Counter ------>
     self.rectScoreBox = pygame.Rect((self.rectTopPipe[0] + self.rectTopPipe.width) + 30, self.rectTopPipe[1] + 670, 1, 350)
    self.rectScoreBox = self.scoreBoxImage.get_rect(topleft=((self.rectTopPipe[0] + self.rectTopPipe.width + 30), self.rectTopPipe[1] + 670))
    #pygame.draw.rect(display, (0,255,0), self.rectScoreBox)
```

Figure 2.3 (Flappy Bird Source Code)

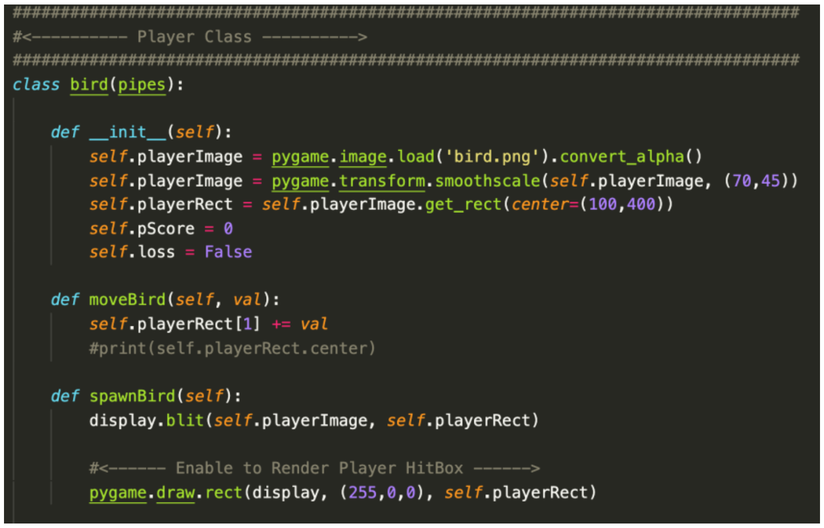
3. “Bird” or Player Class

The bird class allows the game to create a player that the user of the game can interact with and control. This class must create functions that allow the bird to be interacted with by the player, kept track of, and display the player to the screen.

Below in Figure 2.4 is the initialization function of the class. Here the hitbox and the actual image of the bird are loaded and ready to be used in the rest of the class.

The moveBird function takes in a value and then adds height to the player’s y-value based on whatever was passed into the function. This makes the player go up a certain number of pixels when the player hits a key. The functionality of the keyboard inputs will be dealt with later in the report.

The spawnBird function allows the game to spawn and draw out the image for the bird to the game window so the player can see and interpret where the bird is on the screen.



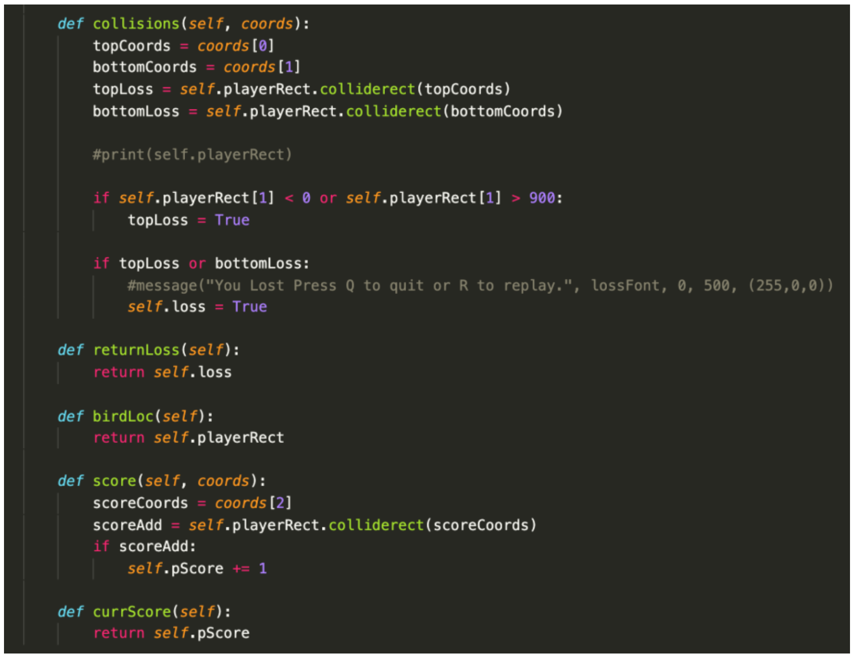
```
###############################################################################
#<---------- Player Class ---------->
###############################################################################
class bird(pipes):

    def __init__(self):
        self.playerImage = pygame.image.load('bird.png').convert_alpha()
        self.playerImage = pygame.transform.smoothscale(self.playerImage, (70,45))
        self.playerRect = self.playerImage.get_rect(center=(100,400))
        self.pScore = 0
        self.loss = False

    def moveBird(self, val):
        self.playerRect[1] += val
        #print(self.playerRect.center)

    def spawnBird(self):
        display.blit(self.playerImage, self.playerRect)

        #<------ Enable to Render Player HitBox ------>
        pygame.draw.rect(display, (255,0,0), self.playerRect)
```

Figure 2.4 (Flappy Bird Source Code)

Continuing with the bird class, the collisions function accepts a coordinate input value. This is pictured below in Figure 2.5. This function takes in the coordinates of the bottom, top, and score obstacles. This allows the game to calculate when or if the player collides with any of these three objects. There is a built in function that PyGame natively uses called .colliderect. This returns a boolean value based on if two objects' hitboxes have collided. This can be used in an if statement to then check if the player has collided with any obstacle. This does not need to be in a loop here because the entire class will be incorporated into the main while loop PyGame uses to function.

Alongside this, there is a score function within this class as well. This score function keeps track of the collisions between the player and the invisible score box. If the player successfully clears through a set of obstacles, the score is then increased by one.

Finally in Figure 2.5, there are three functions that just return basic values. These values will be used later on in the code to interact with other functions and objects outside the player class. These functions return the location of the player, if the player lost or not, and the current score of the game.

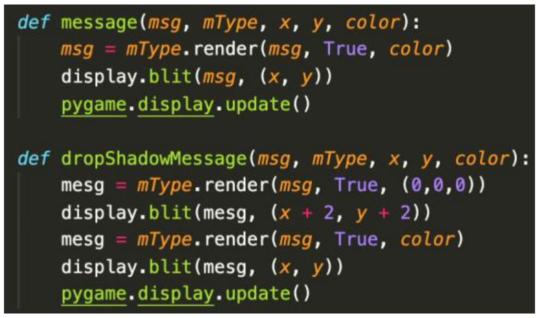


```
    def collisions(self, coords):
        topCoords = coords[0]
        bottomCoords = coords[1]
        topLoss = self.playerRect.colliderect(topCoords)
        bottomLoss = self.playerRect.colliderect(bottomCoords)

        #print(self.playerRect)

        if self.playerRect[1] < 0 or self.playerRect[1] > 900:
            topLoss = True

        if topLoss or bottomLoss:
            #message("You Lost Press Q to quit or R to replay.", lossFont, 0, 500, (255,0,0))
            self.loss = True

    def returnLoss(self):
        return self.loss

    def birdLoc(self):
        return self.playerRect

    def score(self, coords):
        scoreCoords = coords[2]
        scoreAdd = self.playerRect.colliderect(scoreCoords)
        if scoreAdd:
            self.pScore += 1

    def currScore(self):
        return self.pScore
```

Figure 2.5 (Flappy Bird Source Code)

4. Extra Necessary Functions

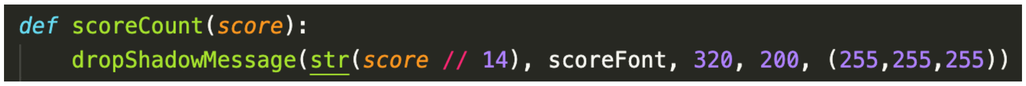
There are functions that exist outside of these main three classes, and this is due to the fact they need to be used throughout the entire script and to avoid having to call and initiate an entire class every time the developer needs one thing to be returned these are then used. The message function allows for a message, any text, to be written to the game window. To do this without hardcoding values into the function the function accepts a message type, x-value, y-value, and color of the text. The message type is dealing with the font, size, and other aspects of text decoration the developer would want. These can be stored within a variable of any name and can be initialized whenever within the entire script. PyGame has these features built in. Similarly, the dropShadowMessage function allows the script to display text with a drop shadow. This is done by creating two layers of text, each one a different color and slightly adjusting the x and y values of the text to line up off centered.



```
def message(msg, mType, x, y, color):
    msg = mType.render(msg, True, color)
    display.blit(msg, (x, y))
    pygame.display.update()

def dropShadowMessage(msg, mType, x, y, color):
    mesg = mType.render(msg, True, (0,0,0))
    display.blit(mesg, (x + 2, y + 2))
    mesg = mType.render(msg, True, color)
    display.blit(mesg, (x, y))
    pygame.display.update()
```

Figure 2.6 (Flappy Bird Source Code)

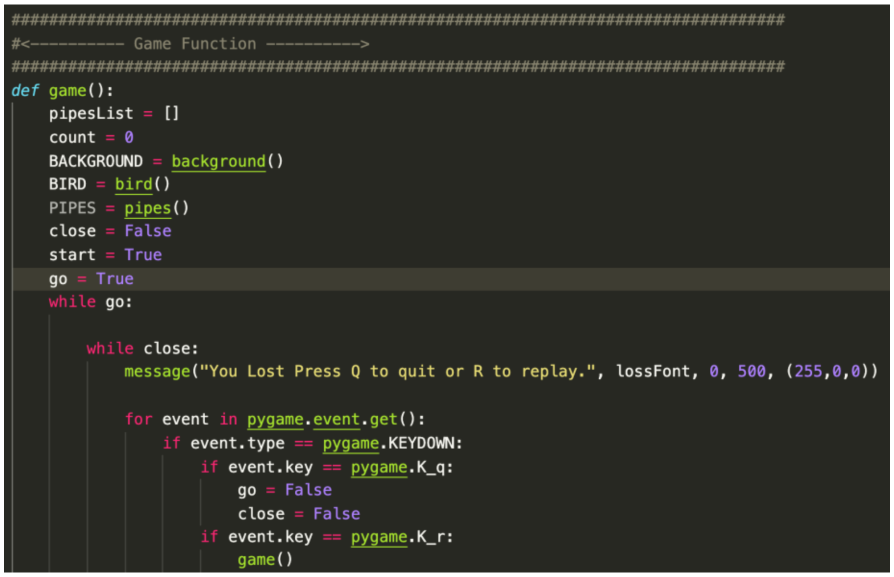
In Figure 2.7 below, the scoreCount function utilizes the dropShadowMessage function passing in a string, message type, x and y values, and a color. This displays the score to the screen in the middle of the game window so the player can see how many obstacles they have currently made it through.



```
def scoreCount(score):
    dropShadowMessage(str(score // 14), scoreFont, 320, 200, (255,255,255))
```

5. Main Game Function

In the first part of the function there is the initialization of variables that will be used in the function. This is pictured below in Figure 2.7. The use of the first while loop is for the basics of PyGame. The second while loop makes sure that the game does not just close when the player loses but displays a screen to allow the player to play again or choose the option to quit. Checking for keys being pressed are also done inside of for loops in PyGame. The code looks for an event that triggers an action and what type of even that may be; mouse click or keyboard press.

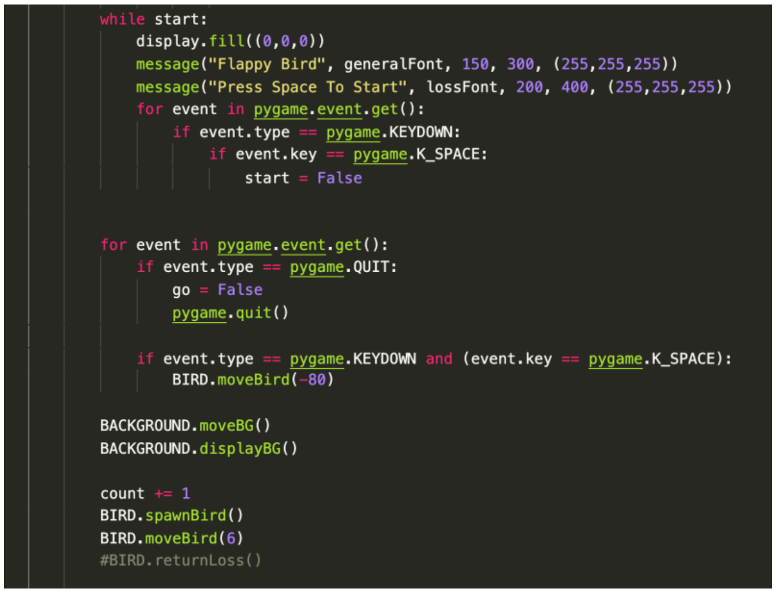


```
##################################################################################
#<---------- Game Function ---------->
##################################################################################
def game():
    pipesList = []
    count = 0
    BACKGROUND = background()
    BIRD = bird()
    PIPES = pipes()
    close = False
    start = True
    go = True
    while go:

        while close:
            message("You Lost Press Q to quit or R to replay.", lossFont, 0, 500, (255,0,0))

            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        go = False
                        close = False
                    if event.key == pygame.K_r:
                        game()
```

Figure 2.7 (Flappy Bird Source Code)

This third while loop controls the start of the game. This is shown below in Figure 2.8. This allows the user to not be instantly kicked into the game when it is started or after the player loses and chooses to retry. Once again there is the use of the for loop to check for inputs.

In the final if statement in Figure 2.8, the code checks if the player has used the space bar. This then moves the player up -80 pixels if the space bar is pressed. This then feeds into the player class and is translated to the “val” variable. The functions within the background class are called to create the background. The moveBird function has the value “6” passed into it so the player will continuously fall by 6 pixels if the player is not tapping the space bar.

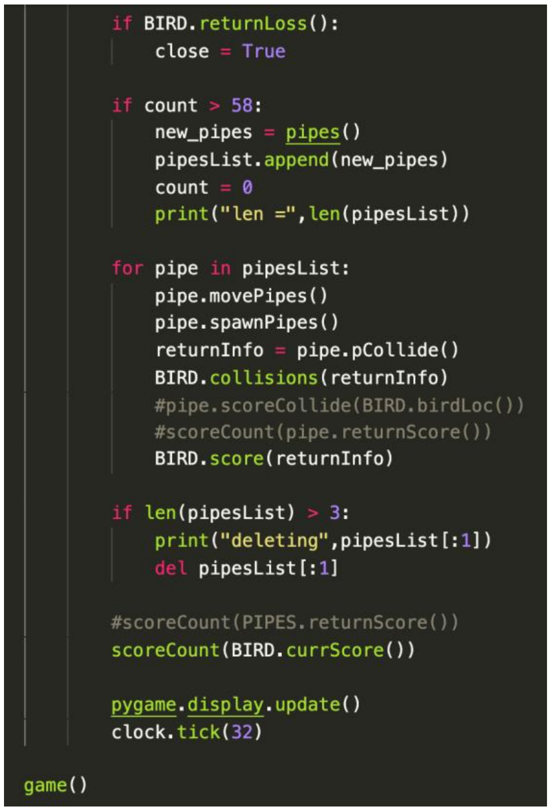


```
while start:
    display.fill((0,0,0))
    message("Flappy Bird", generalFont, 150, 300, (255,255,255))
    message("Press Space To Start", lossFont, 200, 400, (255,255,255))
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                start = False


for event in pygame.event.get():
    if event.type == pygame.QUIT:
        go = False
        pygame.quit()

    if event.type == pygame.KEYDOWN and (event.key == pygame.K_SPACE):
        BIRD.moveBird(-80)

BACKGROUND.moveBG()
BACKGROUND.displayBG()

count += 1
BIRD.spawnBird()
BIRD.moveBird(6)
#BIRD.returnLoss()
```

Figure 2.8 (Flappy Bird Source Code)

Below in Figure 2.9, the code first checks if the player has lost the game. If the player has lost the game it will break from the while loop and exit the game.

The second if statement in Figure 2.9 counts until there are 4 pipes or obstacles generated. This then adds these pies to a list as objects. This is how the game keeps track of all the pies. The pipes are then generated over and over and the list is cleared after all four pipes have gone across the screen. This prevents the memory usage of the game from getting too high and wasting memory.



```
        if BIRD.returnLoss():
            close = True

        if count > 58:
            new_pipes = pipes()
            pipesList.append(new_pipes)
            count = 0
            print("len =",len(pipesList))

        for pipe in pipesList:
            pipe.movePipes()
            pipe.spawnPipes()
            returnInfo = pipe.pCollide()
            BIRD.collisions(returnInfo)
            #pipe.scoreCollide(BIRD.birdLoc())
            #scoreCount(pipe.returnScore())
            BIRD.score(returnInfo)

        if len(pipesList) > 3:
            print("deleting",pipesList[:1])
            del pipesList[:1]

        #scoreCount(PIPES.returnScore())
        scoreCount(BIRD.currScore())

        pygame.display.update()
        clock.tick(32)

game()
```

Figure 2.9 (Flappy Bird Source Code)

Finally in Figure 2.9, if the bird loses the game will go to the close while loop. The for loop deals with creating obstacles and storing them in a list. This list then clears the obstacle that is off screen. To end, clock.tick is how fast the game operates and the display.update() allows PyGame to refresh the game window to draw all objects on the screen. The final call of the game function allows the script to run the function that holds all this code.

iii. Final Product

Below in Figure 2.10 is a picture of the final game. A video walkthrough of the game is available on the website, https://zeus.vwu.edu/~kdforker/cs489/flappybird.html. This shows off randomly generated obstacles, the score count, and the player. The video gives a good representation of the player losing as well as the pipes and background moving seamlessly together.

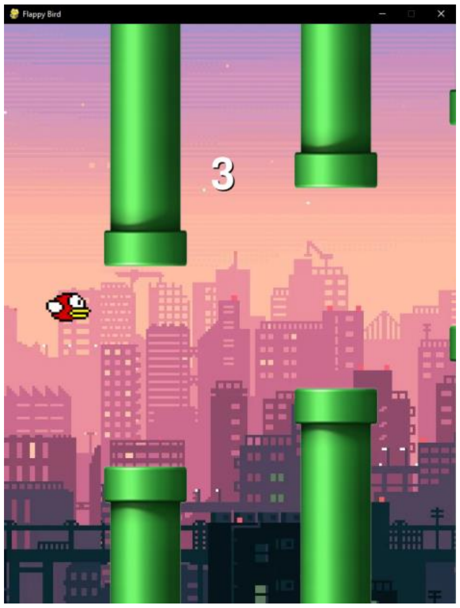


Figure 2.10 (Flappy Bird Game)

d. Pong

i. Overview

Pong is a retro arcade game that is a very simple concept. For the purpose of this research project there was a basic AI added to the game that is not perfect but plays well against the player. The main challenge in this project creation was to make the AI have the ability to lose, but in a human like way. It is easy to code an AI that never loses, but adding a human element was the main focus of research in the creation of this game. This game consists of a player and an AI facing off against each other for the first to five points. The player has a paddle to control with the up and down arrow keys. The ball bounces off the top and bottom of the game window and the player and AI paddles. Whoever gets the ball past the other person gets a point.

ii. How it Works

To start off this section of the report will focus heavily on the AI aspect of the game in pong. These PyGame scripts are all set up very similar to each other and the class and function structure of Pong relates very closely to that of the game earlier, Flappy Bird. For this reason, the main focus of this section will stay on the topic of the functionalities of the AI. Many people have made pong in PyGame and the core mechanics of these are similar, but for the sake of a research project this was used as a learning and building experience to get the hang of creating a basic AI that the player can play against. This will lead into the research that will be conducting the following semester on machine learning and neural networks.

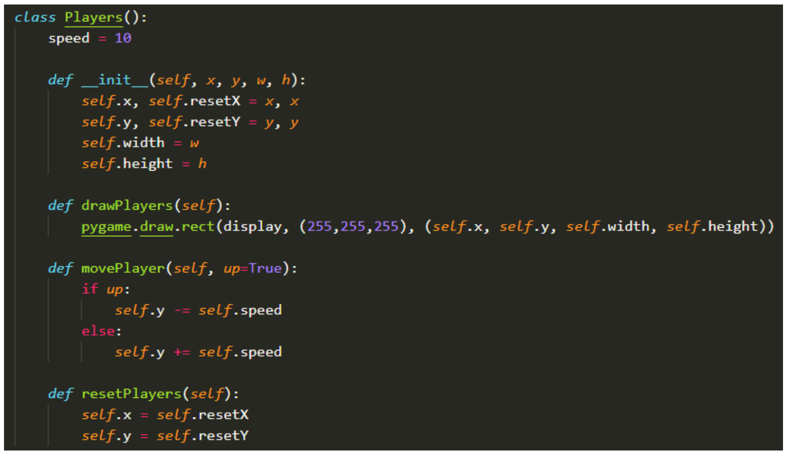
1. Player Class

A brief overview of the player class will be done here. During the creation of these functions after researching, there was the understanding that passing more values throughout the functions within a class made the code flow much better. This allows for the negation of the return functions that were previously used in the flappy bird game. This also allows the code to be cleaner and much more readable as there does not need to be a lot of extra variables.

Inside the initialization class there are variables being defined on the same line. This was also learned in the research that Python allows this to happen just as many other coding languages do. This is usually bad practice in making readable code, but for an initialization function this is perfectly fine to use.

The movePlayer function allows for "up" to be passed into the function. This allows for an if statement to be created in checking whether the player is attempting to go up or down in the game.

The resetPlayers function resets the player to the starting position. This is used later whenever a person scores the ball and the players are reset to their default initial positions.

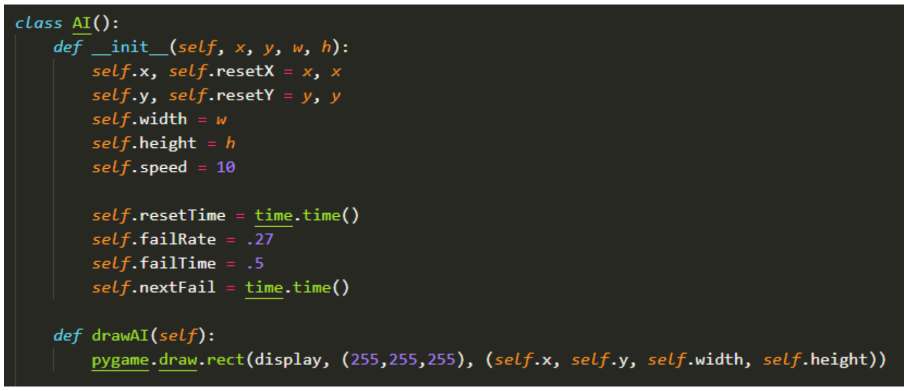


```python
class Players():
    speed = 10

    def __init__(self, x, y, w, h):
        self.x, self.resetX = x, x
        self.y, self.resetY = y, y
        self.width = w
        self.height = h

    def drawPlayers(self):
        pygame.draw.rect(display, (255,255,255), (self.x, self.y, self.width, self.height))

    def movePlayer(self, up=True):
        if up:
            self.y -= self.speed
        else:
            self.y += self.speed

    def resetPlayers(self):
        self.x = self.resetX
        self.y = self.resetY
```

Figure 3.1 (Pong Source Code)

2. AI Class

Below in Figure 3.2 is the AI class. Alongside the initialization of the x and y values are also time values. These values will be used later in the class to make the AI much more humanized. This allows the AI to "forget" where the ball is and then make a mistake. This will be explained in greater detail later on in the report.



```
class AI():
    def __init__(self, x, y, w, h):
        self.x, self.resetX = x, x
        self.y, self.resetY = y, y
        self.width = w
        self.height = h
        self.speed = 10

        self.resetTime = time.time()
        self.failRate = .27
        self.failTime = .5
        self.nextFail = time.time()

    def drawAI(self):
        pygame.draw.rect(display, (255,255,255), (self.x, self.y, self.width, self.height))
```

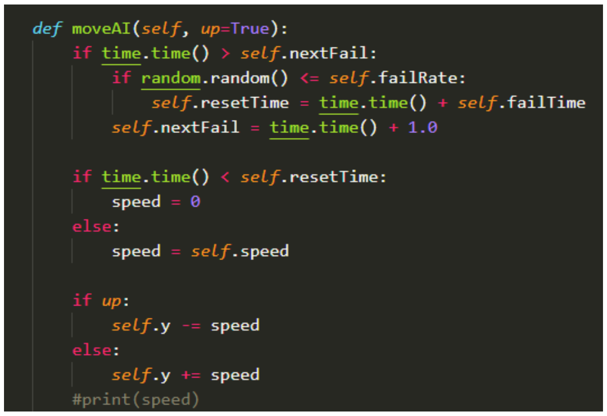
Figure 3.2 (Pong Source Code)

The moveAI function is noticeably bigger than the movePlayer function from the player class. As seen by the final if statement in this function is the core of the AI. It's very simplistic, but there is a problem; the AI never loses.

To overcome this problem the game generates random numbers and use the time module in python. The game then creates a time and keeps track of a variable called nextFail. This creates a check to see how long it has been since the AI "forgot" where it was in the game. In doing this the AI no longer matches the ball coming at it perfectly but allows the AI to "forget" where the ball is on the board. What the game then does is increase the failTime variable with the time it just took to fail and repeat the process. This allows for the AI to fail at random intervals in the game and not have a repeating pattern of failure the player can abuse.

The failTime and failRate variables are initialized very small because no person wants to just win every game every single time, that would be boring. Allowing for the randomization of the failing times the player can still have a long and drawn-out match with the AI.

Alongside this the AI can also correct for its mistakes since the time of the failure is also random. Sometimes the AI will "forget" the position of the ball for only a tenth of a second and sometimes it will "forget" for more than that. The AI is hard locked to a certain speed at which can move on the y-axis as well. This allows for the AI to make corrections from the failures, but also not allow the AI to basically teleport up and down making the game once again unwinnable for the player ever.



```
def moveAI(self, up=True):
    if time.time() > self.nextFail:
        if random.random() <= self.failRate:
            self.resetTime = time.time() + self.failTime
        self.nextFail = time.time() + 1.0

    if time.time() < self.resetTime:
        speed = 0
    else:
        speed = self.speed

    if up:
        self.y -= speed
    else:
        self.y += speed
    #print(speed)
```

Figure 3.3 (Pong Source Code)

iii. Final Product

Below in Figure 3.4 shows the game Pong in action. A demonstration video of this is also on the website, https://zeus.vwu.edu/~kdforker/cs489/pong.html. This showcases the AI and how it interacts with the player. In this screenshot, the player score, the AI score, the ball, and both the paddles for the player and AI are present.

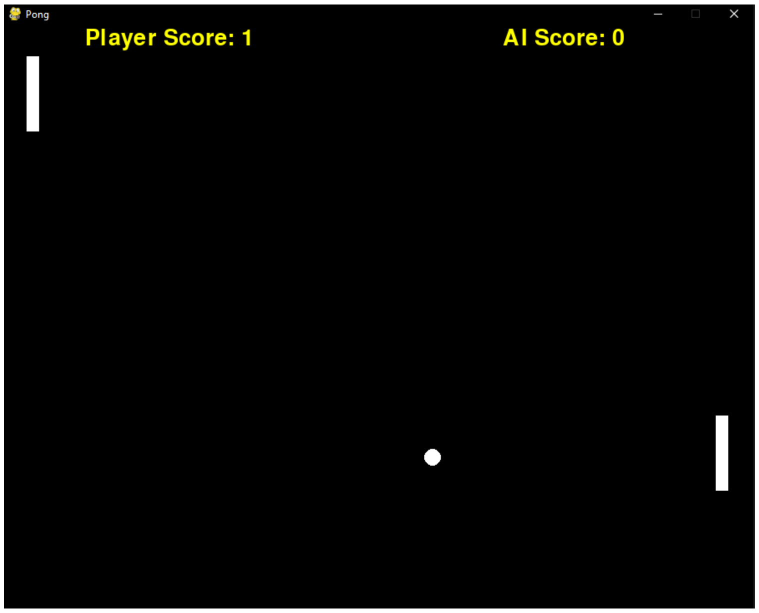


Figure 3.4 (Pong Game)

3. Conclusion

a. Summary

Within this time of this research project the knowledge of Python was used to learn about PyGame and the libraries it offers. This was incorporated into the project by creating and making simple games to showcase the knowledge learned over the course of the researching time. Each of the game builds upon what was learned in the previous one. This allowed for the basic game of Snake to be created all the way to a basic AI that plays against the player in the game Pong. The knowledge learned from this research extends past just creating simplistic games. The knowledge of object-oriented programming in the scripting language Python was deepened as well as the learning of classes and good organization within the scripting language as well. This knowledge as been beneficial to other projects since the start of this research project and the skills will also be used the coming semester to demonstrate on a deeper level of understanding.

4. Resources

"Collision Detection in PyGame." GeeksforGeeks, 18 Aug. 2021, https://www.geeksforgeeks.org/collision-detection-in-pygame/.

Dr. Zizhong John Wang, Vwu, https://zeus.vwu.edu/~zwang/.

"Pygame Front Page" Pygame Front Page - Pygame v2.1.1 Documentation, https://www.pygame.org/docs/.

"Pygame - Creating a Scrolling Background." CodersLegacy, 20 Sept. 2021, https://coderslegacy.com/python/pygame-scrolling-background/.

Techwithtim.net, https://www.techwithtim.net/.