Kyle Forker

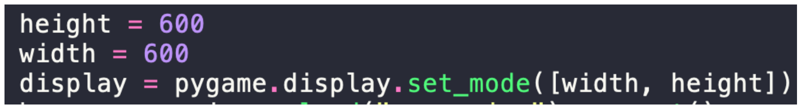
Dr. Wang

CS-489

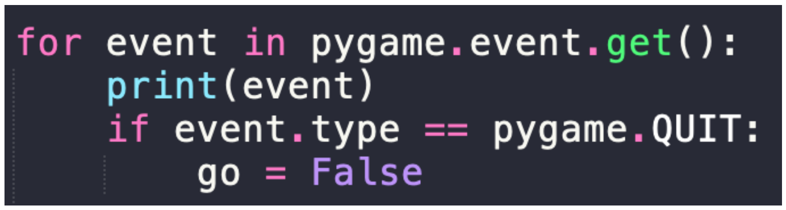
March 20, 2022

Mid-Term Summary

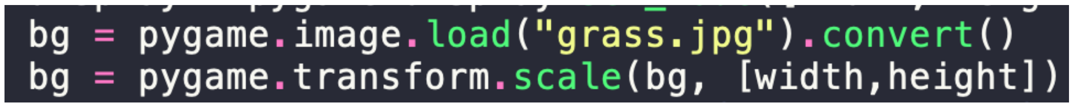
The first term was used as a foundation for the coming terms. In this term the topic was chosen and research on PyGame had begun. This term was essential to the rest of the projects because it was used for learning the ins and outs of PyGame and how to set up basic games with it. This research includes how to set up the game:



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

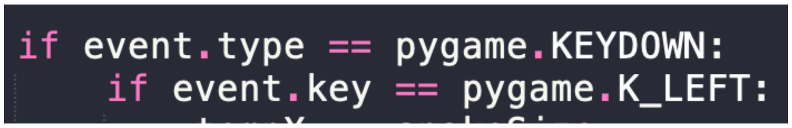


```
for event in pygame.event.get():
    print(event)
    if event.type == pygame.QUIT:
        go = False
```

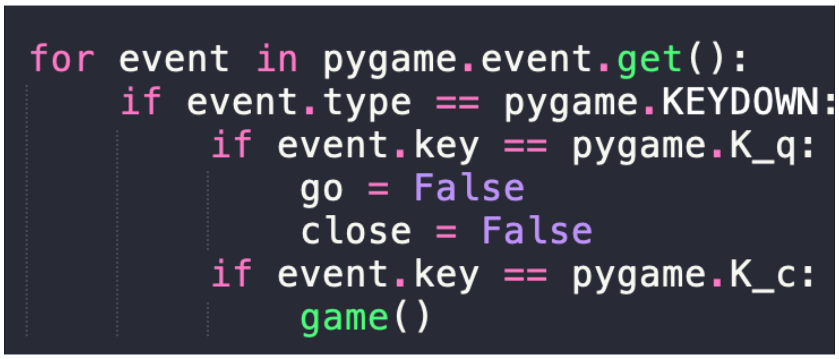


```
bg = pygame.image.load("grass.jpg").convert()
bg = pygame.transform.scale(bg, [width,height])
```

This also includes how to deal with user interaction:



```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_LEFT:
```

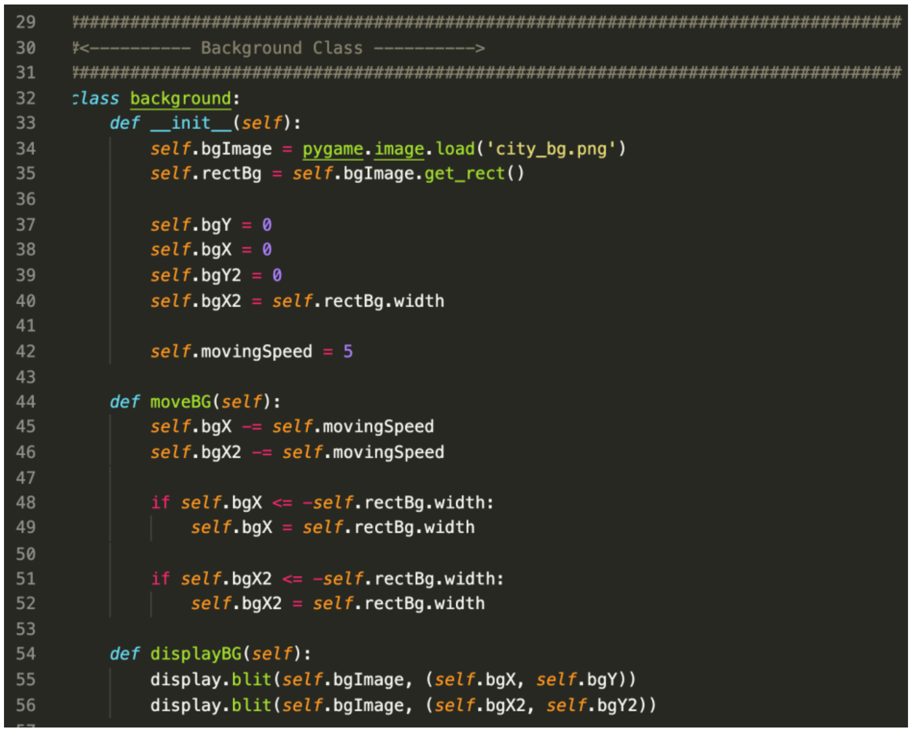


```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_q:
            go = False
            close = False
        if event.key == pygame.K_c:
            game()
```

This led to the first game being made called Snake. This is a classic arcade game that is very basic and allowed for learning the skills needed to produce more complex games later.

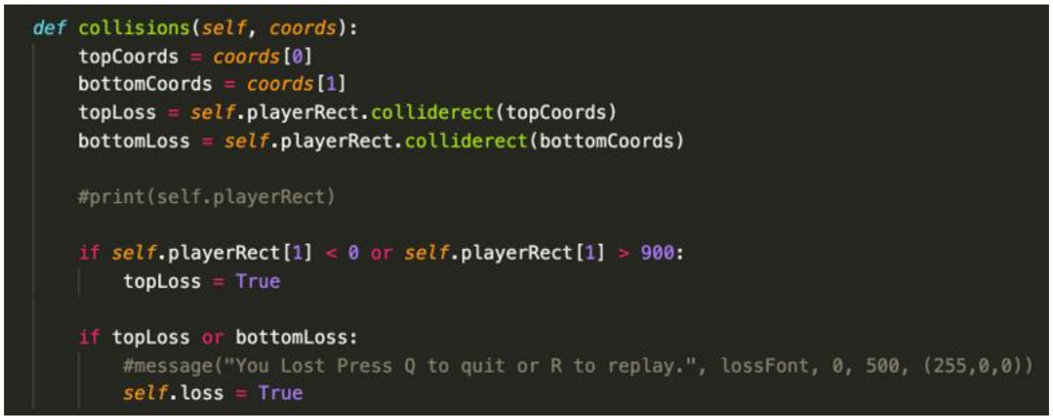
The second term was used to create the first real game with PyGame. This term was also used to refine the skills learned in term one and find ways to optimize and make them better and cleaner. This game that was made was Flappy Bird which used to be a very popular mobile game. Flappy Bird allowed for the showcase of all the materials learned in term one and for new research to be done in this term as well.

This term research involved moving backgrounds:



```
################################################################################
#<---------- 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))
```

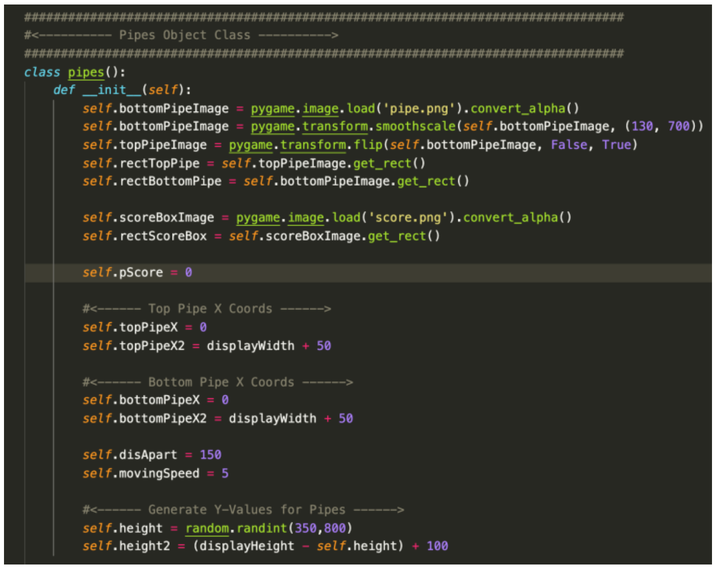
This term research also involved collisions with other objects:



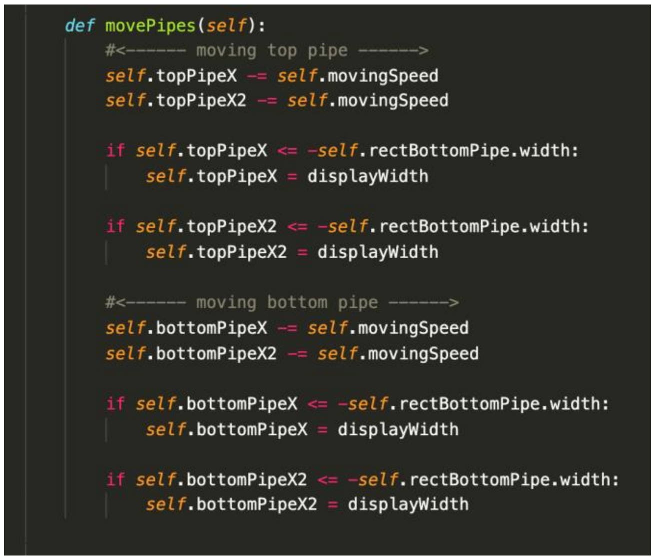
```python
def collisions(self, coords):
    topCoords = coords[0]
    bottomCoords = coords[1]
    topLoss = self.playerRect.colliderect(topCoords)
    bottomLoss = self.playerRect.colliderect(bottomCoords)

    #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
```

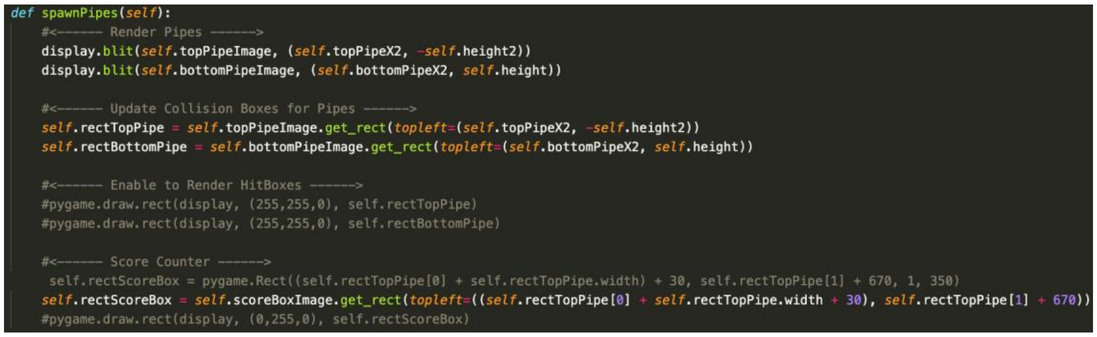
Collisions are very important in PyGame because this is how different objects in the game interact with each other.

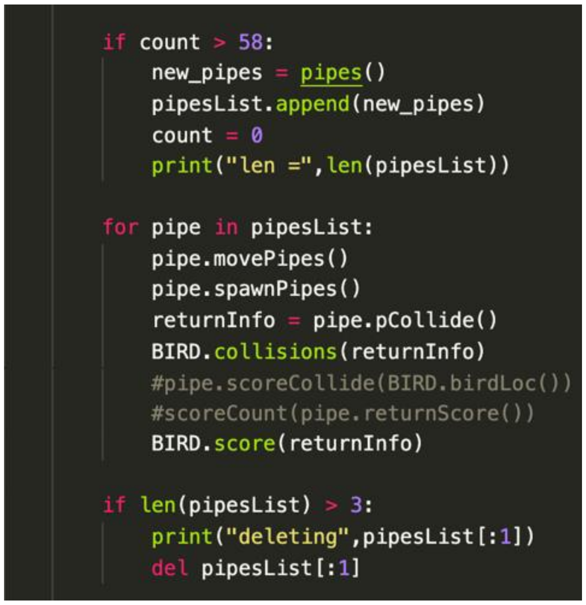
Another thing this term was used for was learning about the generation of different rectangles or objects over the course of the game's progression:



```python
################################################################################
#<---------- 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
```


```python
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
```

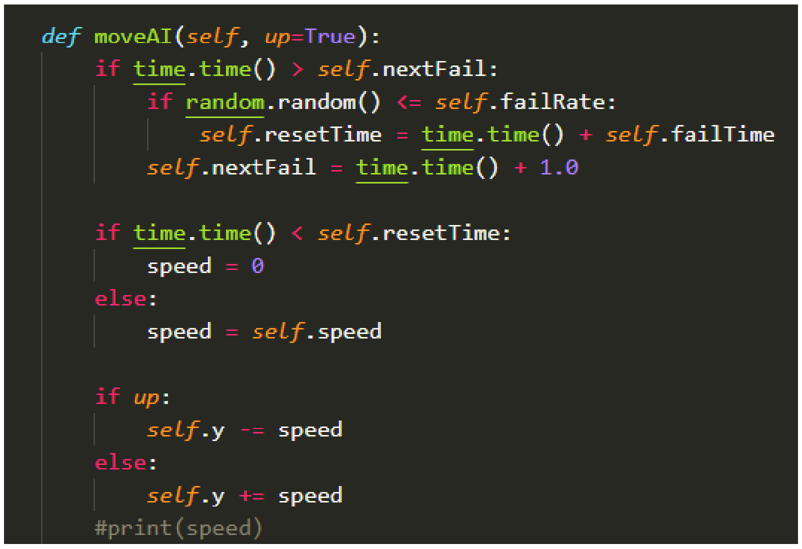

```python
def spawnPipes(self):
    #<------ Render Pipes ------>
    display.blit(self.topPipeImage, (self.topPipeX2, -self.height2))
    display.blit(self.bottomPipeImage, (self.bottomPipeX2, self.height))

    #<------ 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)
```


```python
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]
```

All these screenshots detail how the pipes are generated, kept track of, and deleted after they leave the screen. In this case the pipes were generated for the bird to fly through throughout the game's length while the player had not lost. This game also went more in depth about using player inputs and how the game can interact when a player un presses a key as well. This game also exhibits the use of passing multiple variables throughout several functions as well as using the “self” parameter in python throughout classes. This project also was used to practice the use of object-oriented programming in Python, which is usually not used with Python. This is the backbone of how PyGame works and loads of PyGame games are made through the use of object-oriented programming.

The third term was used to focus on basic AI in a game. This allows the player to be playing against the game itself. The game made was pong. This is a very basic game, and the AI allows for the groundwork to be laid for the final game to be made in the research project in term four. The AI in this game could be replaced by another player quite easily, but the AI was the main topic of research for this term. It’s easy to build an AI that would never lose, especially in a game like pong, but being able ot build one that can make mistake can be tricky.

This is the main driver behind how the AI moves and interacts in game:



```
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)
```

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 do this 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. These failTime and failRate variables are initialized very small because no one wants to just win every game every single time. If that's the case the game gets very boring very quickly. 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 it's 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 is 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.

The use of this information learned about AI in term three will go on to provide proper support for the last game in the research project. This game will be soccer and the player has to score a goal against an AI. This game will be similar to pong, but the game will have to handle the movement in the x-direction as well as the y-direction. This will create a number of different challenges for the development of the game. This final game will really show off everything learned in all of the terms throughout the semester and be a good summary to show how much was learned since the very beginning.