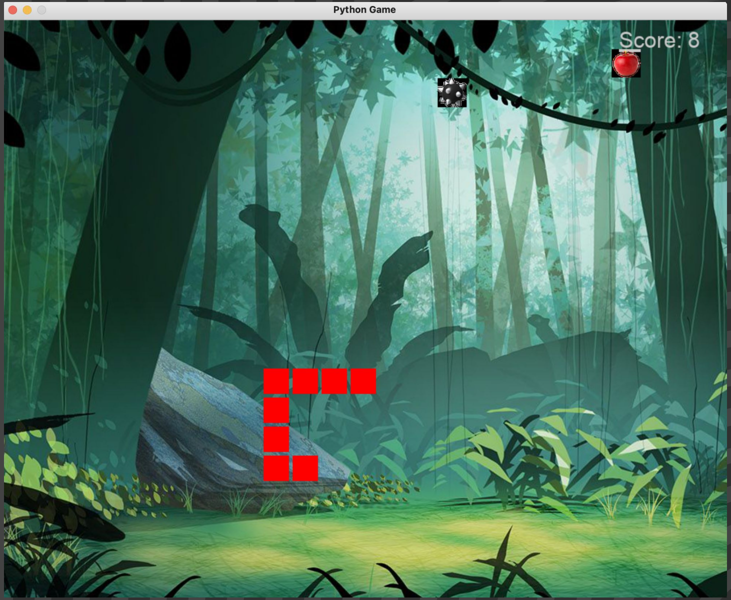
Snake Game

Teagan Walters
CSI 1310

Introduction

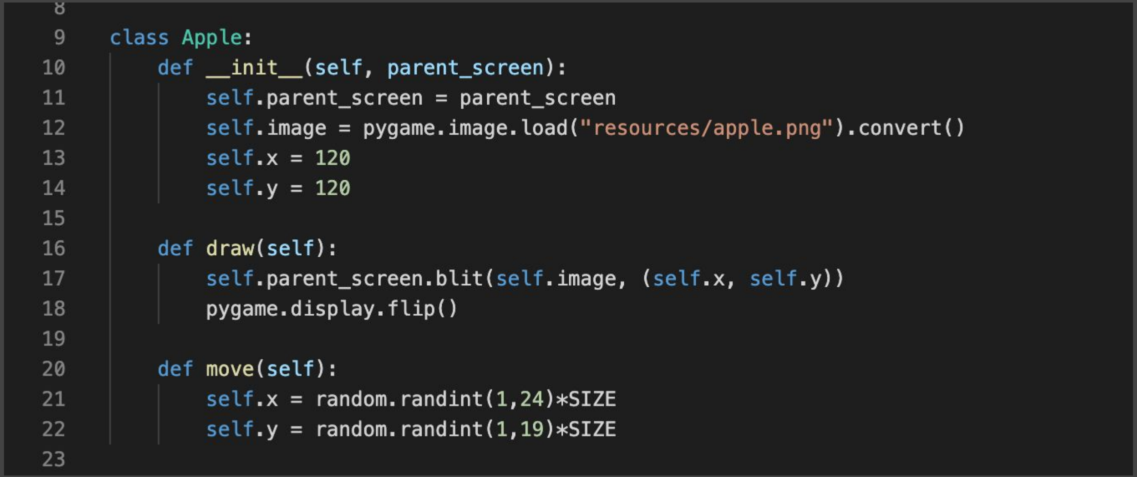
I made a version of the classic game Snake, where the player has to collect apples and increase their size without hitting anything. I used pygame for my visual component.



Classes

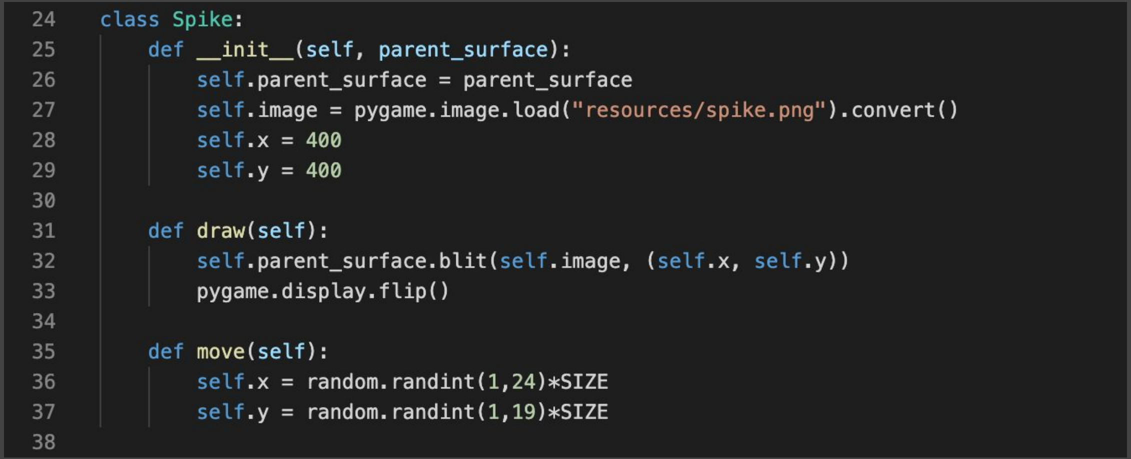
1. class Apple
2. class Snake
3. class Spike
4. class Game

class Apple



```
class Apple:
    def __init__(self, parent_screen):
        self.parent_screen = parent_screen
        self.image = pygame.image.load("resources/apple.png").convert()
        self.x = 120
        self.y = 120

    def draw(self):
        self.parent_screen.blit(self.image, (self.x, self.y))
        pygame.display.flip()

    def move(self):
        self.x = random.randint(1,24)*SIZE
        self.y = random.randint(1,19)*SIZE
```

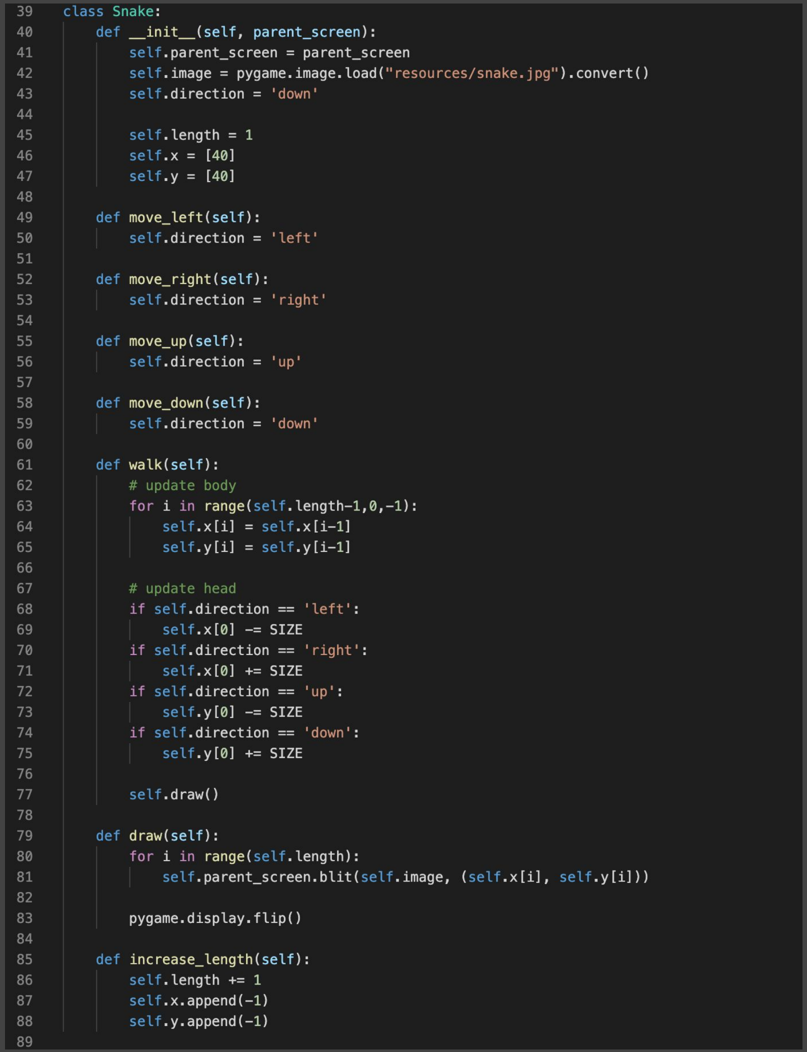
This class represents the apple that the snake eats. The __init__() function loads the image of the apple and sets the starting location for the apple. The draw() function uses blit() to draw the apple and flip() to update the screen to show the apple. Finally, the move() function moves the apple to a random location.



```python
class Spike:
    def __init__(self, parent_surface):
        self.parent_surface = parent_surface
        self.image = pygame.image.load("resources/spike.png").convert()
        self.x = 400
        self.y = 400

    def draw(self):
        self.parent_surface.blit(self.image, (self.x, self.y))
        pygame.display.flip()

    def move(self):
        self.x = random.randint(1,24)*SIZE
        self.y = random.randint(1,19)*SIZE

```

class Spike

This class is almost identical to the Apple class. The only difference is the image that is loaded and the starting location.



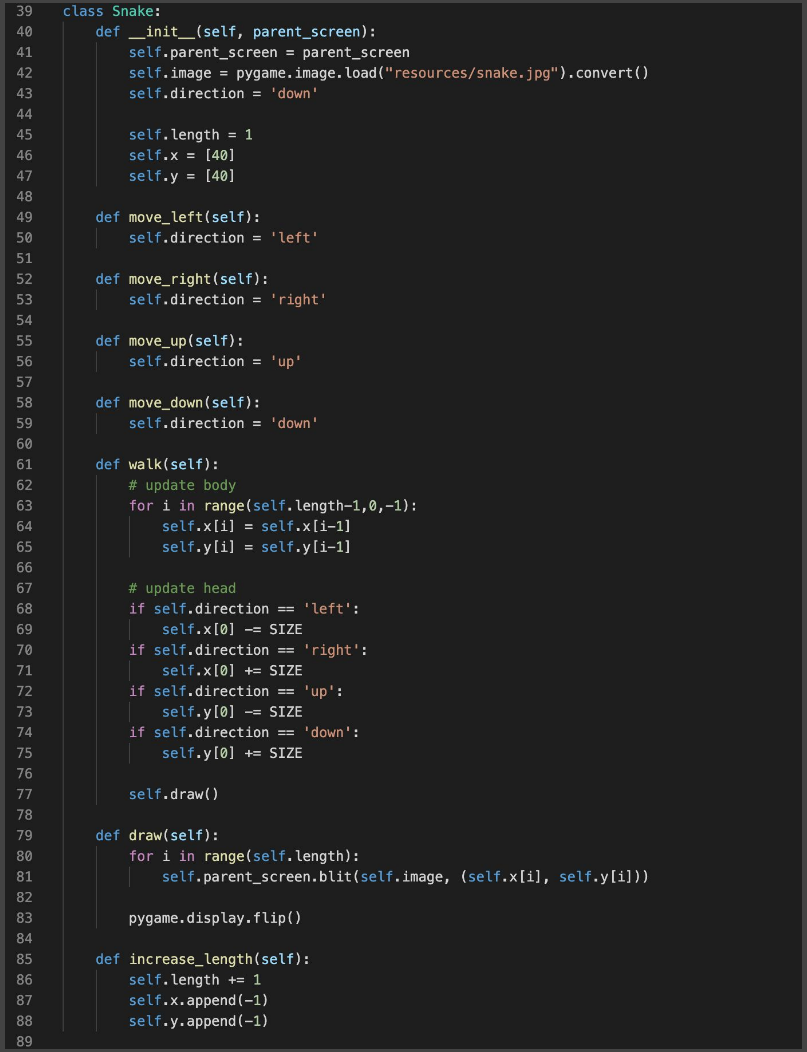
```python
class Snake:
    def __init__(self, parent_screen):
        self.parent_screen = parent_screen
        self.image = pygame.image.load("resources/snake.jpg").convert()
        self.direction = 'down'

        self.length = 1
        self.x = [40]
        self.y = [40]

    def move_left(self):
        self.direction = 'left'

    def move_right(self):
        self.direction = 'right'

    def move_up(self):
        self.direction = 'up'

    def move_down(self):
        self.direction = 'down'

    def walk(self):
        # update body
        for i in range(self.length-1,0,-1):
            self.x[i] = self.x[i-1]
            self.y[i] = self.y[i-1]

        # update head
        if self.direction == 'left':
            self.x[0] -= SIZE
        if self.direction == 'right':
            self.x[0] += SIZE
        if self.direction == 'up':
            self.y[0] -= SIZE
        if self.direction == 'down':
            self.y[0] += SIZE

        self.draw()

    def draw(self):
        for i in range(self.length):
            self.parent_screen.blit(self.image, (self.x[i], self.y[i]))

        pygame.display.flip()

    def increase_length(self):
        self.length += 1
        self.x.append(-1)
        self.y.append(-1)

```

class Snake

This class is used to represent the snake. __init__() loads the image and sets the initial direction to down and sets the length and location of the snake.

The move_'direction'() functions just sets the value of the direction.

Walk() is the function the moves the snake. First it moves all the blocks by using a for loop to make the blocks replace the block in front if it. Second it checks the value of direction and moves the first block in that direction by subtracting or adding the size of the block to the x and y coordinates.

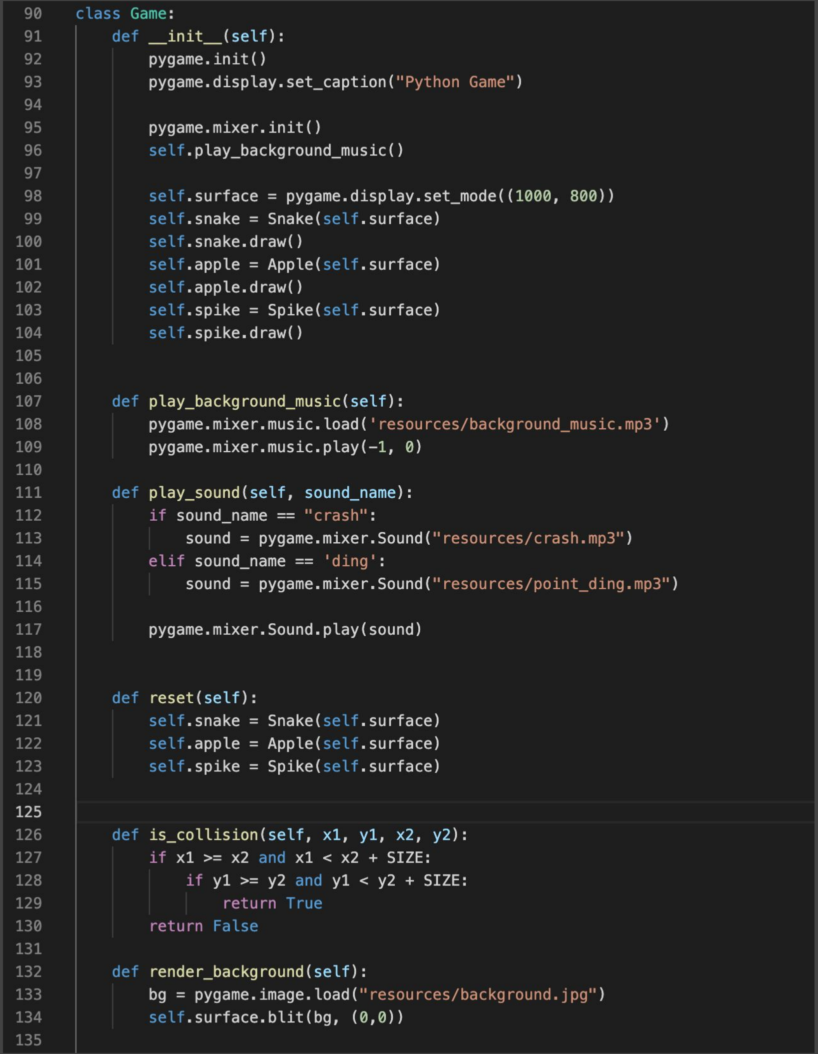


```python
class Snake:
    def __init__(self, parent_screen):
        self.parent_screen = parent_screen
        self.image = pygame.image.load("resources/snake.jpg").convert()
        self.direction = 'down'

        self.length = 1
        self.x = [40]
        self.y = [40]

    def move_left(self):
        self.direction = 'left'

    def move_right(self):
        self.direction = 'right'

    def move_up(self):
        self.direction = 'up'

    def move_down(self):
        self.direction = 'down'

    def walk(self):
        # update body
        for i in range(self.length-1,0,-1):
            self.x[i] = self.x[i-1]
            self.y[i] = self.y[i-1]

        # update head
        if self.direction == 'left':
            self.x[0] -= SIZE
        if self.direction == 'right':
            self.x[0] += SIZE
        if self.direction == 'up':
            self.y[0] -= SIZE
        if self.direction == 'down':
            self.y[0] += SIZE

        self.draw()

    def draw(self):
        for i in range(self.length):
            self.parent_screen.blit(self.image, (self.x[i], self.y[i]))

        pygame.display.flip()

    def increase_length(self):
        self.length += 1
        self.x.append(-1)
        self.y.append(-1)

```

class Snake (cont.)

The draw() function uses a for() loop and blit() to draw each block of the snake. Then it uses flip() to update the screen.

Increase_length() adds 1 to the self.length value.



```python
class Game:
    def __init__(self):
        pygame.init()
        pygame.display.set_caption("Python Game")

        pygame.mixer.init()
        self.play_background_music()

        self.surface = pygame.display.set_mode((1000, 800))
        self.snake = Snake(self.surface)
        self.snake.draw()
        self.apple = Apple(self.surface)
        self.apple.draw()
        self.spike = Spike(self.surface)
        self.spike.draw()


    def play_background_music(self):
        pygame.mixer.music.load('resources/background_music.mp3')
        pygame.mixer.music.play(-1, 0)

    def play_sound(self, sound_name):
        if sound_name == "crash":
            sound = pygame.mixer.Sound("resources/crash.mp3")
        elif sound_name == 'ding':
            sound = pygame.mixer.Sound("resources/point_ding.mp3")

        pygame.mixer.Sound.play(sound)


    def reset(self):
        self.snake = Snake(self.surface)
        self.apple = Apple(self.surface)
        self.spike = Spike(self.surface)


    def is_collision(self, x1, y1, x2, y2):
        if x1 >= x2 and x1 < x2 + SIZE:
            if y1 >= y2 and y1 < y2 + SIZE:
                return True
        return False

    def render_background(self):
        bg = pygame.image.load("resources/background.jpg")
        self.surface.blit(bg, (0,0))

```

class Game

This class contains all the code needed to make the game actually work. The first thing it does is play the background music, create the game window, and draw all of the components on the game window in __init__.

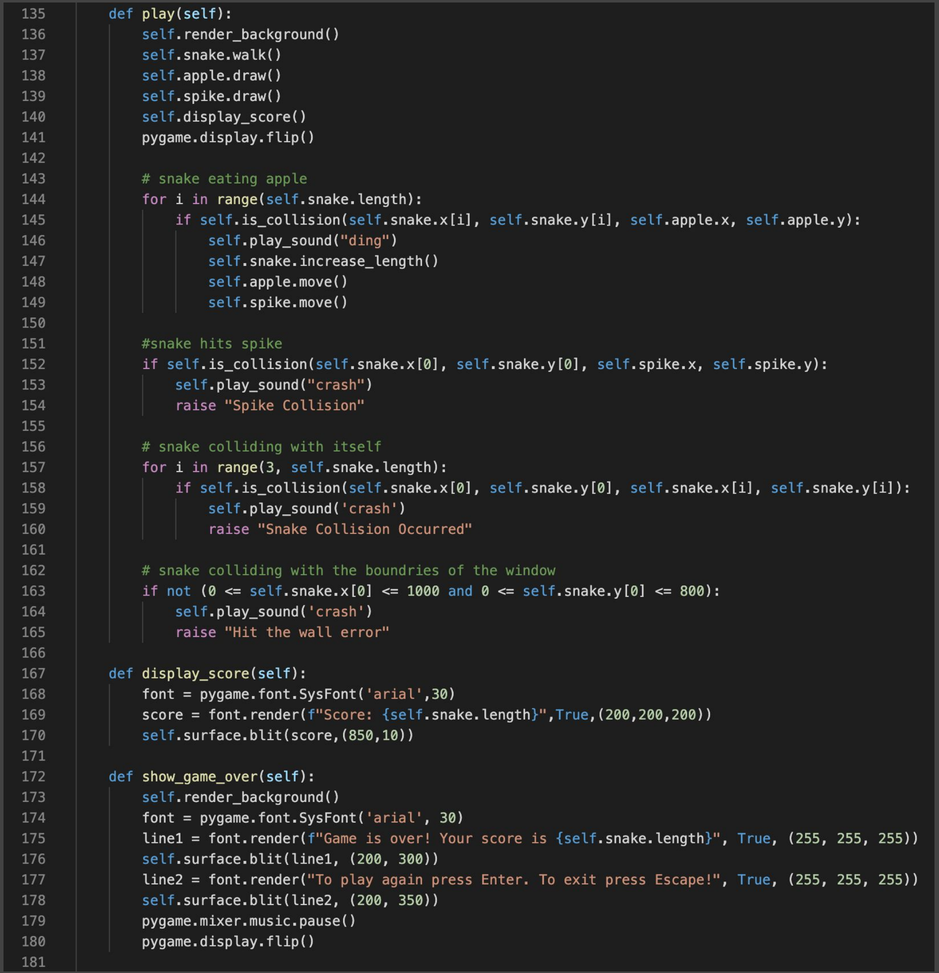
Play_background_music loads the music and plays it.

Play_sound uses an if() statement to check the value of sound_name and plays the correct sound.

Reset() moves the apple, snake, and spike back to the starting positions.

Is_collision() checks for a collision and returns a boolean.

Render_background() loads the background image and updates the screen to display it.



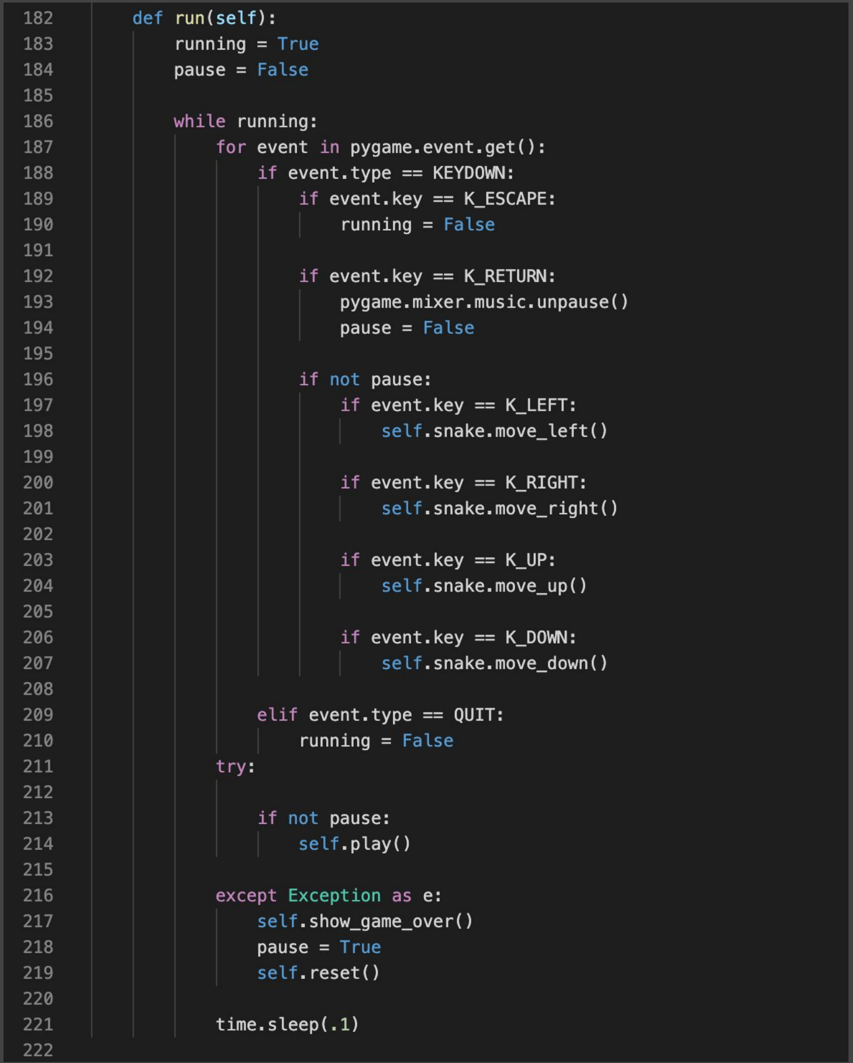
```python
    def play(self):
        self.render_background()
        self.snake.walk()
        self.apple.draw()
        self.spike.draw()
        self.display_score()
        pygame.display.flip()

        # snake eating apple
        for i in range(self.snake.length):
            if self.is_collision(self.snake.x[i], self.snake.y[i], self.apple.x, self.apple.y):
                self.play_sound("ding")
                self.snake.increase_length()
                self.apple.move()
                self.spike.move()

        #snake hits spike
        if self.is_collision(self.snake.x[0], self.snake.y[0], self.spike.x, self.spike.y):
            self.play_sound("crash")
            raise "Spike Collision"

        # snake colliding with itself
        for i in range(3, self.snake.length):
            if self.is_collision(self.snake.x[0], self.snake.y[0], self.snake.x[i], self.snake.y[i]):
                self.play_sound('crash')
                raise "Snake Collision Occurred"

        # snake colliding with the boundries of the window
        if not (0 <= self.snake.x[0] <= 1000 and 0 <= self.snake.y[0] <= 800):
            self.play_sound('crash')
            raise "Hit the wall error"

    def display_score(self):
        font = pygame.font.SysFont('arial',30)
        score = font.render(f"Score: {self.snake.length}",True,(200,200,200))
        self.surface.blit(score,(850,10))

    def show_game_over(self):
        self.render_background()
        font = pygame.font.SysFont('arial', 30)
        line1 = font.render(f"Game is over! Your score is {self.snake.length}", True, (255, 255, 255))
        self.surface.blit(line1, (200, 300))
        line2 = font.render("To play again press Enter. To exit press Escape!", True, (255, 255, 255))
        self.surface.blit(line2, (200, 350))
        pygame.mixer.music.pause()
        pygame.display.flip()
```

class Game (cont.)

Play() sets up the game and updates the screen. It also checks for if the snake eats an apple or collides with something and executes the necessary code. When an apple is collected the snake length increases and the spike and apple are moved. When the snake collides with something an exception is raised.

Display_score() sets the score in the corner of the game screen. It uses the length of the snake as the score.

Show_game_over() just updates the screen with the game over message and pauses the music.



```python
def run(self):
    running = True
    pause = False

    while running:
        for event in pygame.event.get():
            if event.type == KEYDOWN:
                if event.key == K_ESCAPE:
                    running = False

                if event.key == K_RETURN:
                    pygame.mixer.music.unpause()
                    pause = False

                if not pause:
                    if event.key == K_LEFT:
                        self.snake.move_left()

                    if event.key == K_RIGHT:
                        self.snake.move_right()

                    if event.key == K_UP:
                        self.snake.move_up()

                    if event.key == K_DOWN:
                        self.snake.move_down()

            elif event.type == QUIT:
                running = False
        try:

            if not pause:
                self.play()

        except Exception as e:
            self.show_game_over()
            pause = True
            self.reset()

        time.sleep(.1)

```

class Game (cont.)

This is the last function in class Game. While the variable running is true, this function collects keyboard input to quit the game, restart the game, and change the direction of the snake.

In the try() function if the game is not paused the function play() is executed. Except() notices if one of the exceptions in play has been raised and stops the game and shows the game over screen.

The last line sets the speed of the snake by setting the time between when snake.move() is executed.

My Experience With Pygame

I was having a lot of problems downloading pygame. I tried to install it using the instructions on the official pygame website. However, I kept getting errors. I tried looking up videos on how to install pygame and tried using Homebrew, per the instructions on the pygame website, to download pygame. I also tried downloading the latest version of python and pip and nothing worked. After about two hours of troubleshooting and looking for answers. I finally just searched up “why can't I download pygame” and the answer was so frustratingly simple. Apparently pygame just didn't like python 3.8, which was the version I was using, and all I had to do is download python 3.7