PyGame Car Crash Tutorial - Part 3

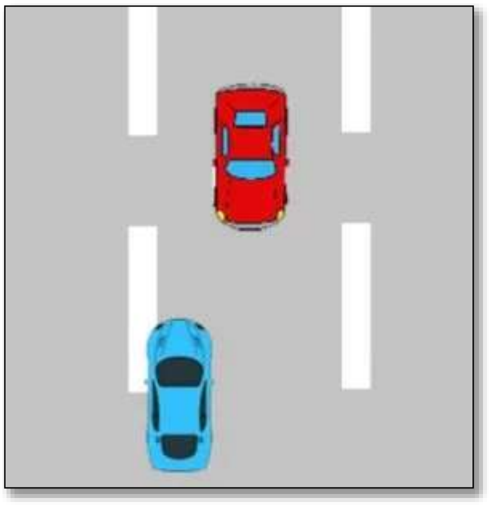
Contents

Time required: 30 minutes

Preview of the Game

Here's a sneak peak of the game that we are going to work on.

Car Crash Demo Video



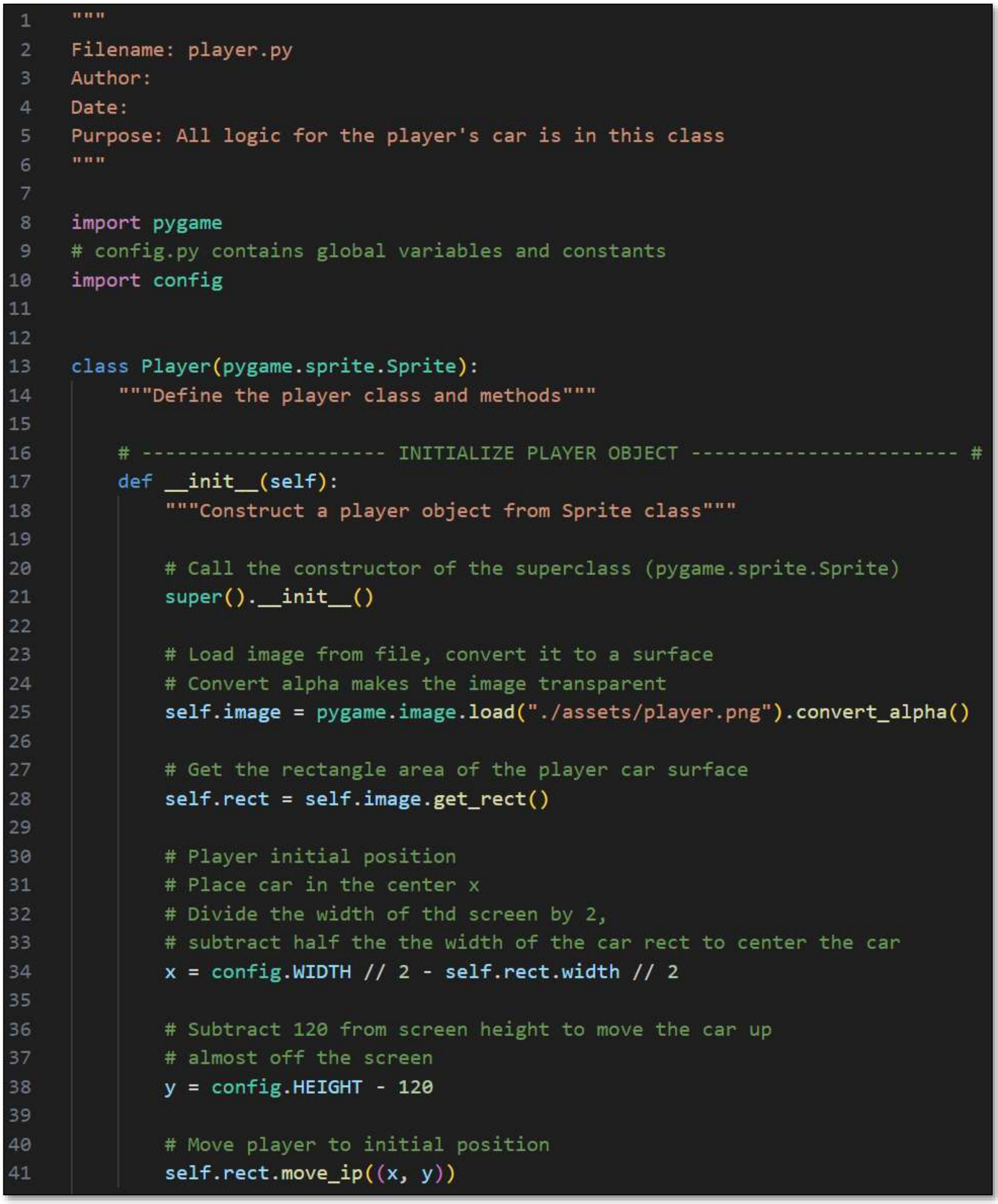
Car Crash is simple arcade type game. The object is to move your blue car back and forth to avoid the oncoming red cars.

Player Class

Let's display the player's car.

1. Create a new Python file. save it as **player.py**

2. Add the following code.



```python
"""
Filename: player.py
Author:
Date:
Purpose: All logic for the player's car is in this class
"""

import pygame
# config.py contains global variables and constants
import config


class Player(pygame.sprite.Sprite):
    """Define the player class and methods"""

    # --------------------- INITIALIZE PLAYER OBJECT ----------------------- #
    def __init__(self):
        """Construct a player object from Sprite class"""

        # Call the constructor of the superclass (pygame.sprite.Sprite)
        super().__init__()

        # Load image from file, convert it to a surface
        # Convert alpha makes the image transparent
        self.image = pygame.image.load("./assets/player.png").convert_alpha()

        # Get the rectangle area of the player car surface
        self.rect = self.image.get_rect()

        # Player initial position
        # Place car in the center x
        # Divide the width of thd screen by 2,
        # subtract half the the width of the car rect to center the car
        x = config.WIDTH // 2 - self.rect.width // 2

        # Subtract 120 from screen height to move the car up
        # almost off the screen
        y = config.HEIGHT - 120

        # Move player to initial position
        self.rect.move_ip((x, y))
```

Above is the code for the Player Class. Classes are like templates. They are used to create objects. From one cookie cutter (class) you can make multiple cookies (objects).

One of the benefit of using classes is that we can spawn multiple entities/objects from the same block of code. This doesn’t really apply to the Player Class; most games will only have one player. It does apply to the Enemy Class as most games will have multiple enemies.

Passing **pygame.sprite.Sprite** into the parameters makes the Player Class it’s child class. This allows the Player class to create Sprite objects which inherit all the properties and methods of the Sprite class.

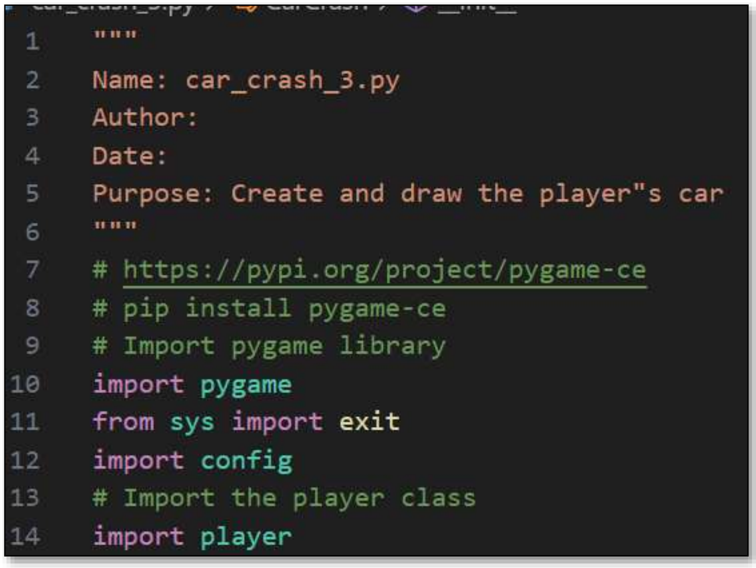
The **init()** function initializes or constructs an object from a class. **super().init()** calls the **init()** function of the Sprite class. This gives the Player object the properties and methods of the Sprite class.

The **image.load()** function loads our image into a variable. This does not define the borders for our Player Sprite. This is done using the **Surface()** and **get_rect()** functions that create a rectangle of the specified size. It is much easier to manipulate a rectangle than an image.

CarCrash Class

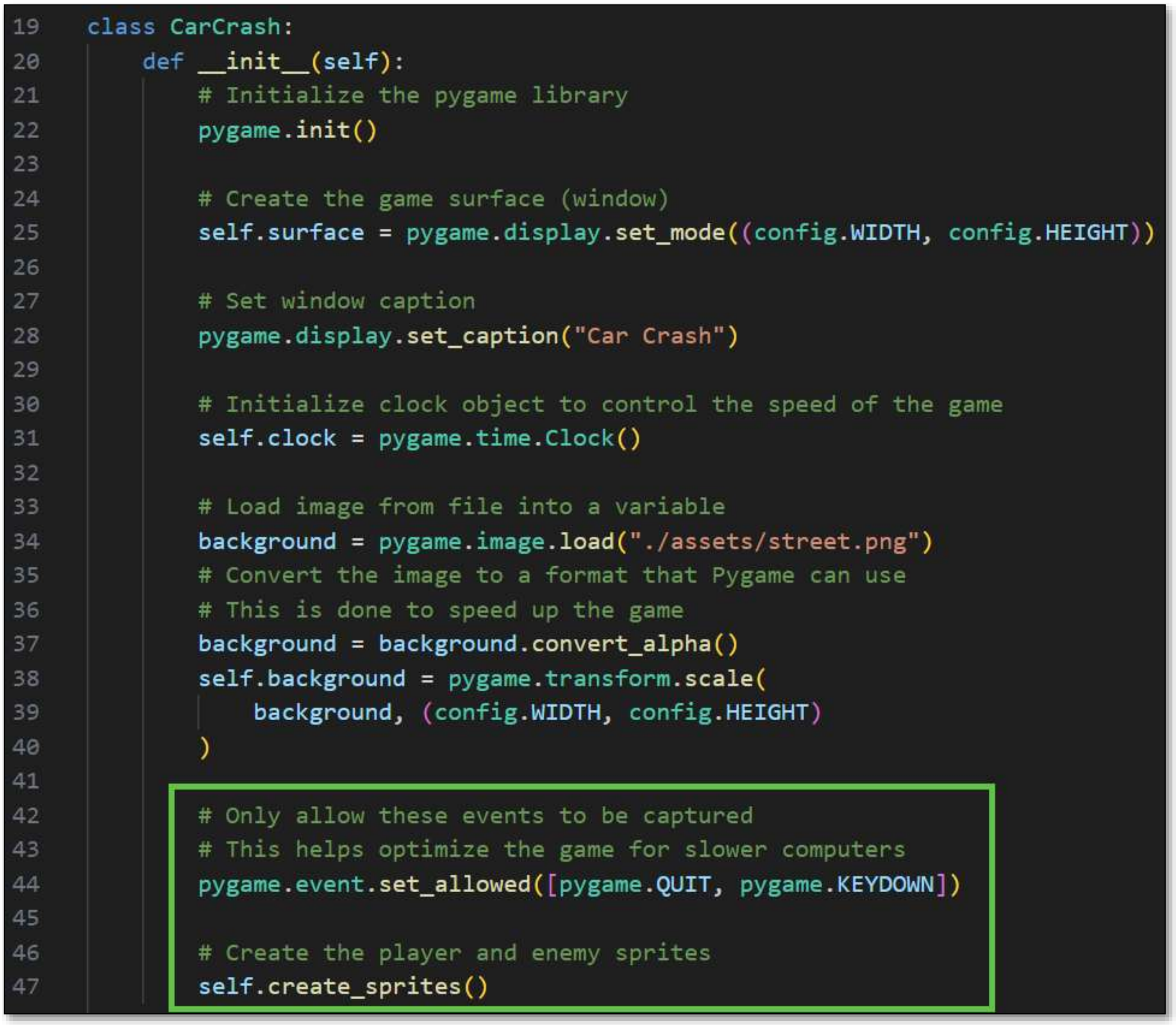
It is a good idea to save versions of complex as you work through them. You can go back to a working version to see what went wrong.

1. Save your current **car_crash_2.py** as **car_crash_3.py**
2. Add an import for the **player** class.



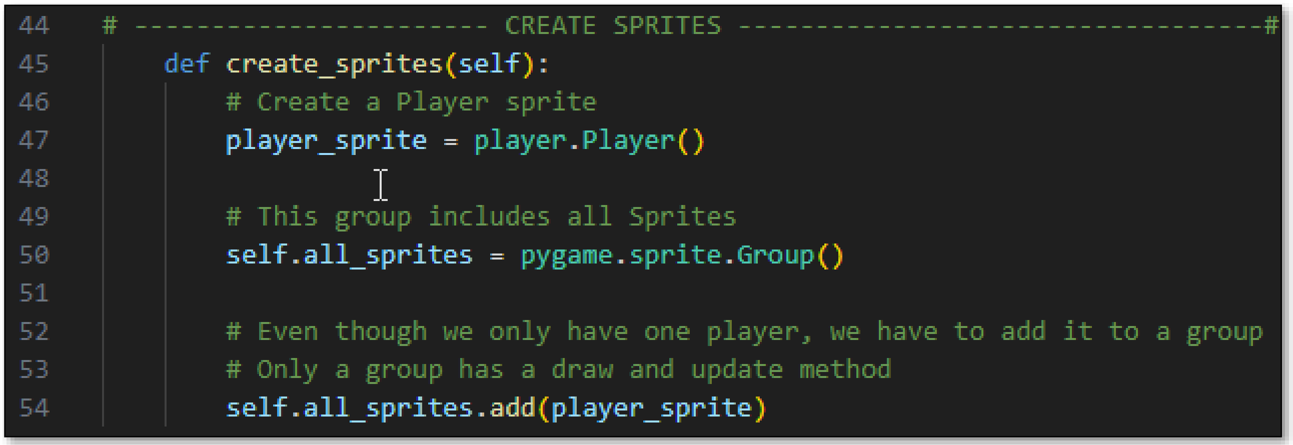
```python
"""
Name: car_crash_3.py
Author:
Date:
Purpose: Create and draw the player"s car
"""
# https://pypi.org/project/pygame-ce
# pip install pygame-ce
# Import pygame library
import pygame
from sys import exit
import config
# Import the player class
import player
```

3. We only need the events listed. All other events can be ignored. That saves processing events that don’t need to be processed.



```
class CarCrash:
    def __init__(self):
        # Initialize the pygame library
        pygame.init()

        # Create the game surface (window)
        self.surface = pygame.display.set_mode((config.WIDTH, config.HEIGHT))

        # Set window caption
        pygame.display.set_caption("Car Crash")

        # Initialize clock object to control the speed of the game
        self.clock = pygame.time.Clock()

        # Load image from file into a variable
        background = pygame.image.load("./assets/street.png")
        # Convert the image to a format that Pygame can use
        # This is done to speed up the game
        background = background.convert_alpha()
        self.background = pygame.transform.scale(
            background, (config.WIDTH, config.HEIGHT)
        )

        # Only allow these events to be captured
        # This helps optimize the game for slower computers
        pygame.event.set_allowed([pygame.QUIT, pygame.KEYDOWN])

        # Create the player and enemy sprites
        self.create_sprites()
```

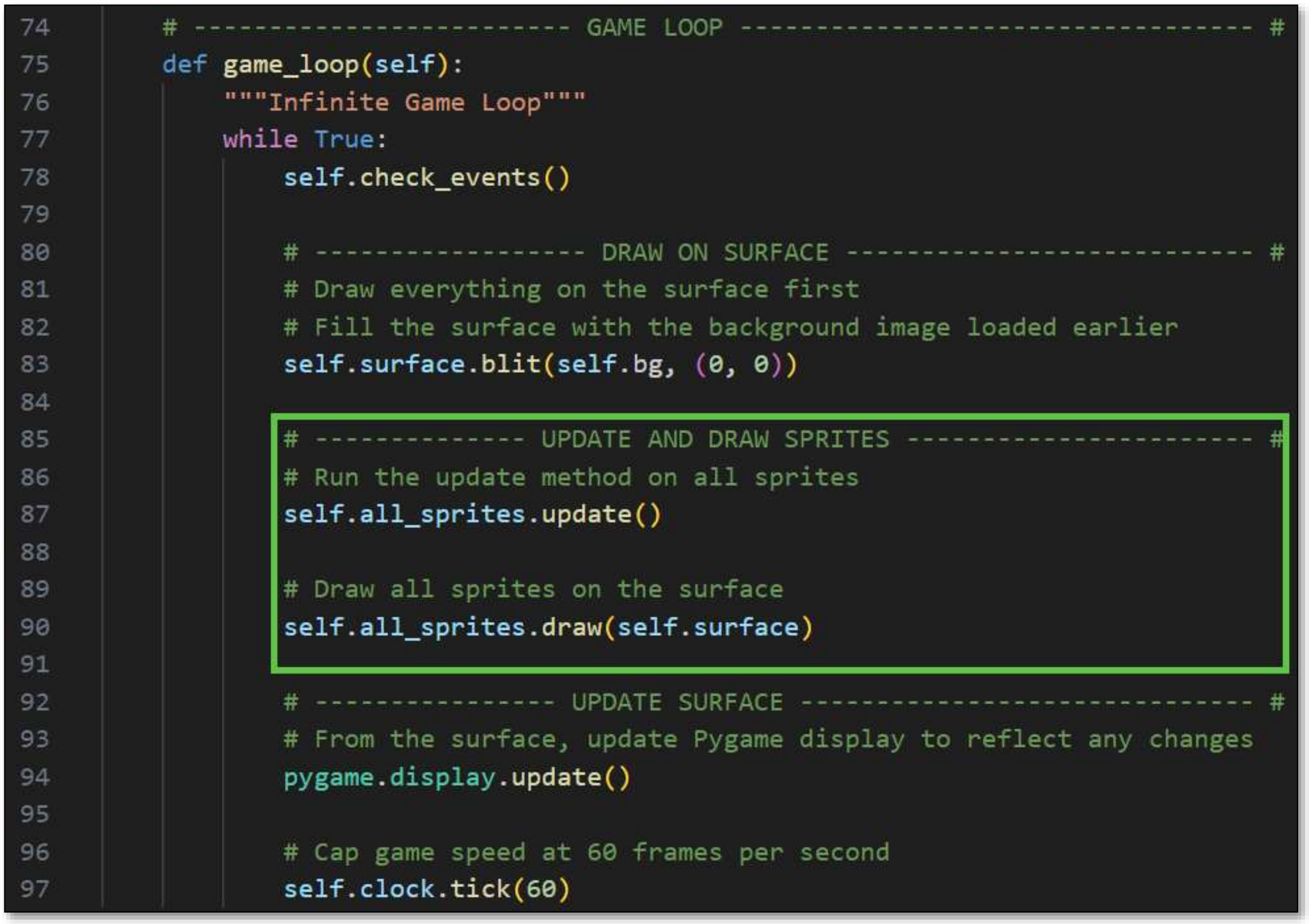
4. This method creates the player sprite, the group, then adds the player to the group. A Sprite group has built in methods that make them easy to use in a game.



```python
# ----------------------- CREATE SPRITES ----------------------------------#
    def create_sprites(self):
        # Create a Player sprite
        player_sprite = player.Player()

        # This group includes all Sprites
        self.all_sprites = pygame.sprite.Group()

        # Even though we only have one player, we have to add it to a group
        # Only a group has a draw and update method
        self.all_sprites.add(player_sprite)
```

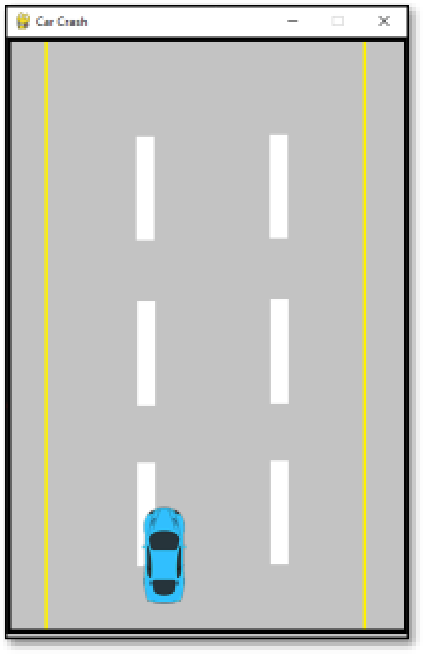
5. The **check_events()** method does not change.

6. We update and draw the sprite group. There isn't anything in the update method, yet. We are drawing the player to the screen as a stationary image. We will move it later.



```python
    # ------------------------ GAME LOOP --------------------------------- #
    def game_loop(self):
        """Infinite Game Loop"""
        while True:
            self.check_events()

            # ------------------ DRAW ON SURFACE --------------------------- #
            # Draw everything on the surface first
            # Fill the surface with the background image loaded earlier
            self.surface.blit(self.bg, (0, 0))

            # ------------- UPDATE AND DRAW SPRITES ---------------------- #
            # Run the update method on all sprites
            self.all_sprites.update()

            # Draw all sprites on the surface
            self.all_sprites.draw(self.surface)

            # ---------------- UPDATE SURFACE ------------------------------ #
            # From the surface, update Pygame display to reflect any changes
            pygame.display.update()

            # Cap game speed at 60 frames per second
            self.clock.tick(60)
```

Example run:



Assignment Submission

1. Attach a screenshot showing the operation of the program.
2. Zip up the program files folder and submit in Blackboard.