Part 4 – Pygame Car Crash Tutorial

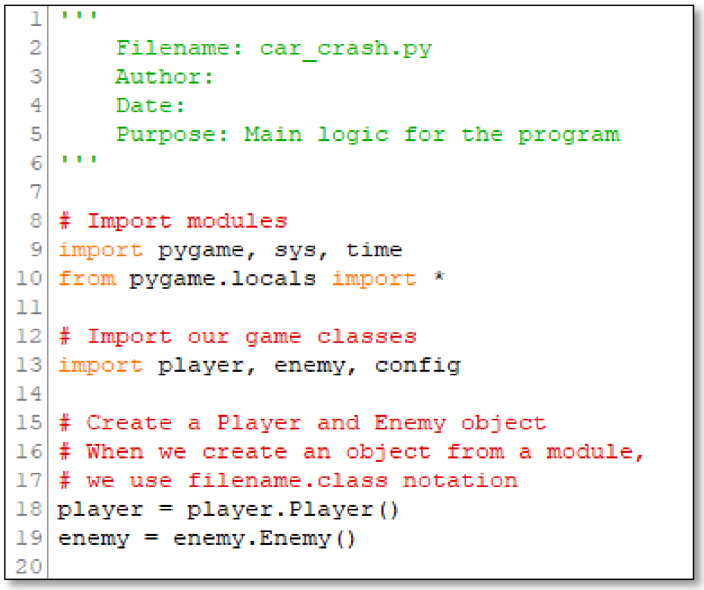
Contents

Time required: 30 minutes

Our game is still incomplete. There's no fun in playing a game with the same thing happening over and over again. There is no end point, no variation in the game difficulty and most importantly, there are no consequences of colliding with the enemy.

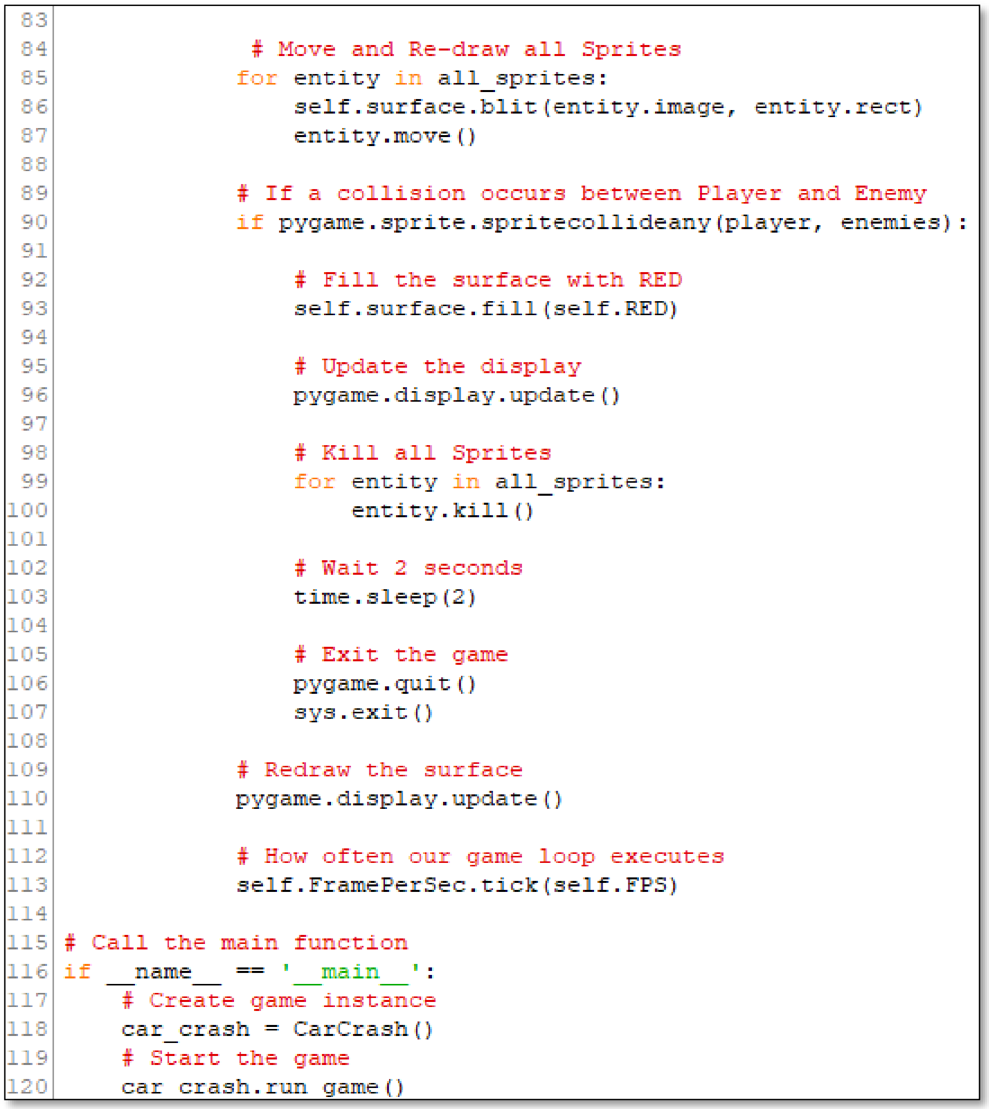
In this section we're going to cover Sprite Grouping, Collision Detection, User events and some other minor features. The player, config, and enemy classes have not changed.

Here's the Code Version 4 for the main CarCrash program. There are several additions, some lines have changed and been removed. Take a good look at the code before moving on to the explanation.



```python
'''
    Filename: car_crash.py
    Author:
    Date:
    Purpose: Main logic for the program
'''

# Import modules
import pygame, sys, time
from pygame.locals import *

# Import our game classes
import player, enemy, config

# Create a Player and Enemy object
# When we create an object from a module,
# we use filename.class notation
player = player.Player()
enemy = enemy.Enemy()

```

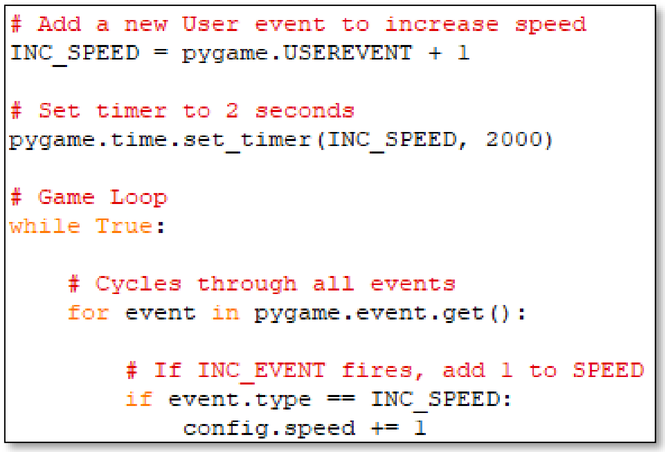
```python
# Create Sprites Groups, add Sprites to Groups
# A separate enemies group is created,
# to allow for more enemy Sprites later on
enemies = pygame.sprite.Group()
enemies.add(enemy)

# This group includes all Sprites
all_sprites = pygame.sprite.Group()
all_sprites.add(player)
all_sprites.add(enemy)

# Add a new User event to increase speed
# + 1 ensures that the USERENENT is unique
INC_SPEED = pygame.USEREVENT + 1

class CarCrash:
    ''' Setup the object data fields '''
    # Setup color constants
    BLUE =  (0, 0, 255)
    RED =   (255, 0, 0)
    GREEN = (0, 255, 0)
    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    # Constant for Frames Per Second (FPS)
    FPS = 60
    # Setup a computer clock object
    FramePerSec = pygame.time.Clock()

    def __init__(self):
        ''' Initialize the object '''
        # Initialize pygame for action
        pygame.init()

        # Set timer for INC_SPEED to increase speed every 2 seconds
        pygame.time.set_timer(INC_SPEED, 2000)

        # Create the game window, color and caption
        self.surface = pygame.display.set_mode((config.SCREEN_WIDTH,
                                                config.SCREEN_HEIGHT))
        self.surface.fill(self.WHITE)
        pygame.display.set_caption("Car Crash")

    def run_game(self):
        ''' Start the infinite Game Loop '''
        while True:
            # Closing the program by clicking the X
            # causes the QUIT event to be fired
            for event in pygame.event.get():

                # If INC_EVENT fires, add .5 to SPEED
                if event.type == INC_SPEED:
                    config.speed += .5

                # Exit game if window is closed
                if event.type == QUIT:
                    # Quit Pygame
                    pygame.quit()
                    # Exit Python
                    sys.exit()

            # Fill the surface with white to clear the screen
            self.surface.fill(self.WHITE)
```



```python

             # Move and Re-draw all Sprites
            for entity in all_sprites:
                self.surface.blit(entity.image, entity.rect)
                entity.move()

            # If a collision occurs between Player and Enemy
            if pygame.sprite.spritecollideany(player, enemies):

                # Fill the surface with RED
                self.surface.fill(self.RED)

                # Update the display
                pygame.display.update()

                # Kill all Sprites
                for entity in all_sprites:
                    entity.kill()

                # Wait 2 seconds
                time.sleep(2)

                # Exit the game
                pygame.quit()
                sys.exit()

            # Redraw the surface
            pygame.display.update()

            # How often our game loop executes
            self.FramePerSec.tick(self.FPS)

# Call the main function
if __name__ == '__main__':
    # Create game instance
    car_crash = CarCrash()
    # Start the game
    car_crash.run_game()
```

That’s a lot of code. Considering the size of the average game created in Python pygame, it’s still very small.

Event Objects


```
# Add a new User event to increase speed
INC_SPEED = pygame.USEREVENT + 1

# Set timer to 2 seconds
pygame.time.set_timer(INC_SPEED, 2000)

# Game Loop
while True:

    # Cycles through all events
    for event in pygame.event.get():

        # If INC_EVENT fires, add 1 to SPEED
        if event.type == INC_SPEED:
            config.speed += 1
```

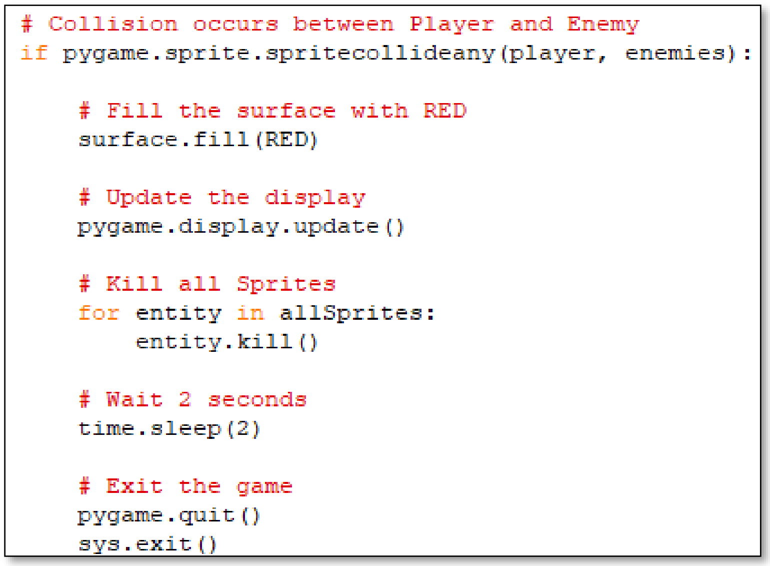
We talked about event objects earlier, such as `QUIT`. Pygame, gives us the option to create custom events called "User events". Here we've created an event called `INC_SPEED`. To do so, we called the `pygame.USEREVENT` and added one into it (to ensure that it will have a unique ID).

We use the Pygame time module's `set_timer()` function to call the `INC_SPEED` event object every 2000 milliseconds, or 2 seconds.

The next piece of code if about the Game loop. In the for loop where we iterate over every event that occurs, we insert an if statement to check for the `INC_SPEED` event occurring. If it does, we increase the `speed` variable by 1. This variable is used by the Enemy class to determine its speed.

All in all, the purpose of this code is to make the game more challenging as time passes.

Collision Detection



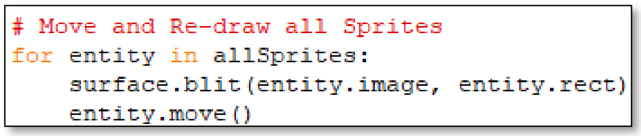
```python
# Collision occurs between Player and Enemy
if pygame.sprite.spritecollideany(player, enemies):

    # Fill the surface with RED
    surface.fill(RED)

    # Update the display
    pygame.display.update()

    # Kill all Sprites
    for entity in allSprites:
        entity.kill()

    # Wait 2 seconds
    time.sleep(2)

    # Exit the game
    pygame.quit()
    sys.exit()
```

This section of code is related to collision detection in Python pygame. Remember how we created groups earlier? You're about to see a massive benefit that we get from having meaningful groups.

The `spritecollideany()` function takes two parameters, the first must be a regular Sprite, like `player` or `enemy`. The second must be a Sprite group, such as `enemies` or `allSprites`. This function compares the sprite passed in the first parameter, to see if it's touching any of the sprites in the group passed in parameter two.

In our case, it checks to see whether our Player has collided with any of the sprites in the `enemies` group. The benefit of this function is that even if there are 1000 enemy sprites, we don't have to check collisions with them individually, and instead just use this function.

When the collision holds True, we kill all the sprites using the `kill()` function, fill the screen with red, wait two seconds and close the entire program.



```python
# Move and Re-draw all Sprites
for entity in allSprites:
    surface.blit(entity.image, entity.rect)
    entity.move()
```

Another benefit of the grouping system, we can now call the "move" functions for all the sprites and redraw them in just 3 lines of code.

Assignment Submission

Zip up the program files folder and submit in Blackboard.