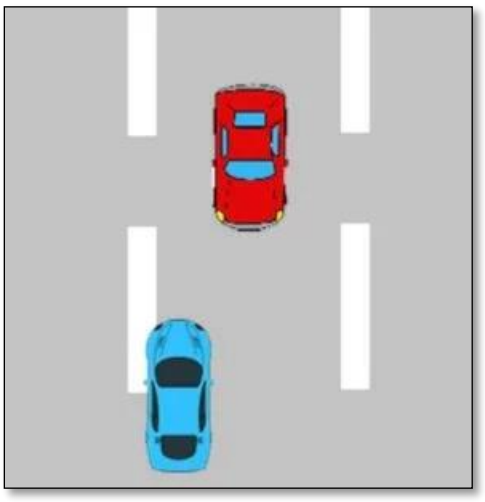
Part 1 – Pygame Car Crash Tutorial

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

Content

There is a **Content.zip** file attached to the assignment. This has all the images and sounds we will use in the game.

1. Extract the files and copy them to the same folder you are creating your game in.

CarCrash Class

The first step is to draw the game without any motion. We will start with displaying the player's car.

1. Create a new Python file. save it as **car_crash_1.py** Add the following code.

```
'''
    Name: car_crash.py
    Author:
    Date:
    Purpose: Draw the player's car
'''

# Import pygame and sys modules
import pygame, sys
from pygame.locals import *
import player

# Create a Player object
player = player.Player()

class CarCrash:
    ''' Setup the object data fields '''
    # Setup color and screen size constants
    WHITE = (255, 255, 255)
    WIDTH = 400
    HEIGHT = 600
    # 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()

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

```

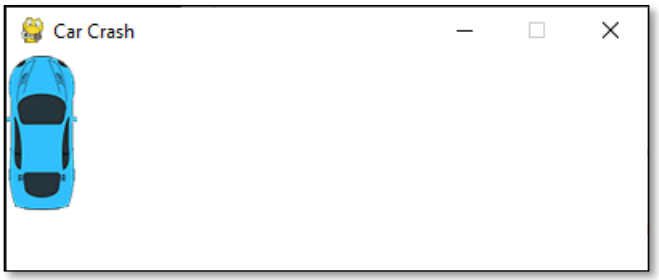
```
    def run_game(self):
        ''' Start the infinite Game Loop '''
        while True:
            # Closing the program by clicking the X
            # causes the QUIT event to be fired
            for event in pygame.event.get():

                # 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)

            # Draw the player sprite on the surface
            player.draw(self.surface)

            # 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()
```

Player Class

Create a new python file named: **player.py** Edit and add the following code.

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

# Import modules
import pygame
from pygame.locals import *

class Player(pygame.sprite.Sprite):
    ''' Define the player class and methods '''
    # Construct a Player object

    def __init__(self):

        # Construct a player object from Sprite class
        super().__init__()

        # Load an image from file
        self.image = pygame.image.load("player.png")

        # Create a surface rectangle the same size as the image
        self.surf = pygame.Surface((50, 100))

        # Gets the rectangle area of the Surface
        # Starts at 0, 0 which is the upper left corner of the surface
        self.rect = self.surf.get_rect()

    # Draw the Player object on the surface
    def draw(self, surface):
        surface.blit(self.image, self.rect)
```

Example run:



This is how the game will look at this stage. The blue player car (sprite) is drawn to the screen.

Classes and Objects

We'll be using Classes and Objects for our program. Whether it's GUI, Pygame or any other large application the Object-Oriented Programming approach is almost always the best idea.

The approach we used earlier for the drawing program was fine because our program was small, and there wasn't anything that required repetition or needed to be reused. Using Classes, we'll be using methods to store blocks of code that are to be repeated several times throughout the game.

Modules, Variables, and the Surface

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

At the top of the code the standard Pygame modules `pygame`, `sys`, and `from pygame.locals import *` are imported.

```
# Initialize pygame
pygame.init()

# Setup color constants
WHITE = (255, 255, 255)

# Setup a computer clock object for Frames per Second
FramePerSec = pygame.time.Clock()
```

We first initialize pygame.

Color constants are setup using standard RGB (Red, Green, Blue) values for use later in the program.

Then `FramePerSec` is setup as a computer clock object to ensure we get 60 frames per second according to the computer clock.

```
# Create the game window, color and caption
surface = pygame.display.set_mode((400 , 600))
surface.fill(WHITE)
pygame.display.set_caption("Car Crash")
```

This code sets up the display surface, clears the window by painting it with `WHITE`, and sets the caption.

Player Class

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

    # Initialize or construct a Player object
    def __init__(self):

        # Initialize or construct a player object from Sprite class
        super().__init__()

        # Load an image from file
        self.image = pygame.image.load("player.png")

        # Create a surface rectangle the same size as the image
        self.surf = pygame.Surface((50, 100))

        # Gets the rectangle area of the Surface
        # Starts at 0,0 which is the upper left corner of the surface
        self.rect = self.surf.get_rect()
```

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 the file path of our image. 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.

Draw the Player Class

In the Player class, we create a rectangle of width 50 and length 100 as the body of our Sprite. The Image and rectangle are the same size. If they aren't, there will be issues during collision detection. The `get_rect()` function has an optional (X, Y) location

argument. Without an argument, the rectangle starts at (0,0), which is the upper left corner of the surface.

```
# Draw the Player object on the surface
def draw(self, surface):
    surface.blit(self.image, self.rect)
```

`blit()` takes two inputs, the first the surface to be drawn to and second, the object. Normally we would write `surface.blit(self.surf, self.rect)` since we're drawing the rectangle to the surface we've defined. But since we're using an image, we pass `self.image` instead of `self.surf`.

Objects and the Game Loop

```
# Create a Player object
player = Player()
```

This code creates a Player object from our class. We could create as many unique objects as we wish. For this game, we only need one Player.

```
# Game Loop
while True:

    # Exit the game
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

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

    # Draw the player sprite on the surface
    player.draw(surface)

    # Update the screen
    pygame.display.update()

    # Game Loop 60 times a second
    FramePerSec.tick(60)
```

The commands shown above are all in the game loop, they repeat continuously.

1. Test for any Pygame events. If we close the program window, the program exits.
2. Refresh and clear the surface using the `surface.fill(WHITE)` function.
3. Call the player draw functions.

4. The `pygame.display.update()` command updates the screen with all the commands that have occurred up-till this point.
5. The `tick()` method makes sure the Game Loop repeats only 60 times per second based on the computer clock.

Assignment Submission

Zip up the program files folder and submit in Blackboard.