Part 5 – Pygame Car Crash Tutorial

Contents

Time required: 30 minutes

Let's finish up our car game with backgrounds, sounds, fonts, and a scoring system.

Add the files attached to this assignment to the folder.

Car Crash Code

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

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

# Import the modules we created
import player
import enemy
from config import 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()

# 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 USEREVENT is unique
INC_SPEED = pygame.USEREVENT + 1


class CarCrash:

    # Setup a computer clock object
    FramePerSec = pygame.time.Clock()

    # Load background image from file
    background = pygame.image.load("animated_street.png")

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

        # Create configuration object
        self.config = Config()
```

```
        # Set up Fonts
        self.font_big = pygame.font.SysFont("Verdana", 60)
        self.font_small = pygame.font.SysFont("Verdana", 20)

        # Render this font now, it won't change and is only used once
        # Not rendering in the Game Loop increases performance
        self.game_over = self.font_big.render(
            "Game Over",
            True,
            self.config.BLACK)

        # Background music plays continuously
        pygame.mixer.music.load('background_music.wav')
        pygame.mixer.music.set_volume(0.3)
        pygame.mixer.music.play(-1)

        # 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(
            (self.config.SCREEN_WIDTH,
             self.config.SCREEN_HEIGHT))
        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:
                    self.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 the background image loaded earlier
            self.surface.blit(self.background, (0, 0))

            print(enemy.config.score)
            # Render score before drawing it on the surface
            scores = self.font_small.render(
                str(enemy.config.score),
                True,
                self.config.BLACK)


            # Draw score on the surface
            self.surface.blit(scores, (10, 10))

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

```
117
118             # If a collision occurs between Player and Enemy
119             if pygame.sprite.spritecollideany(player, enemies)
120                 # Stop background sound
121                 pygame.mixer.music.stop()
122
123                 # Play crash sound
124                 pygame.mixer.Sound('crash.wav').play()
125
126                 # Wait 1 second
127                 time.sleep(1)
128
129                 # Fill the surface with RED
130                 self.surface.fill(self.config.RED)
131
132                 # Display game over on surface
133                 self.surface.blit(self.game_over, (30, 250))
134
135                 # Update the display
136                 pygame.display.update()
137
138                 # Kill all Sprites
139                 for entity in all_sprites:
140                     entity.kill()
141
142                 # Wait 2 seconds
143                 time.sleep(2)
144
145                 # Exit the game
146                 pygame.quit()
147                 sys.exit()
148
149             # Redraw the surface
150             pygame.display.update()
151
152             # How often our game loop executes
153             self.FramePerSec.tick(self.config.FPS)
154
155
156 # Call the main function
157 if __name__ == '__main__':
158     # Create game instance
159     car_crash = CarCrash()
160     # Start the game
161     car_crash.run_game()
```

Explanation car_crash.py

```
# Set up Fonts
self.font_big = pygame.font.SysFont("Verdana", 60)
self.font_small = pygame.font.SysFont("Verdana", 20)

# Render this font now, it won't change and is only used once
# Not rendering in the Game Loop increases performance
self.game_over = self.font_big.render("Game Over", True, self.BLACK)
```

In the code above, we're setting up fonts to be used later in our program. We create two different fonts, `font_big` and `font_small` which both have the same font family, but different font sizes.

We use the `render()` function to actually create the graphics for the Font. We pass in the text we wish to be displayed and the color we want it to be in. In this case, `"Game Over"` and `BLACK` respectively.

```
# Render score before drawing it on the surface
scores = self.font_small.render(str(config.score), True, self.BLACK)

# Draw score on the surface
self.surface.blit(scores, (10, 10))
```

```
# Display game over on surface
self.surface.blit(self.game_over, (30, 250))
```

This is the second part of our fonts in the Game Loop. We render another font called `scores`. We didn't do this earlier because this font is meant to be rendered inside the Game Loop as it is continuously changing value. We moved the `game_over` font rendering out of the loop to avoid unnecessary performance loss.

We display both fonts using the `blit()` function. In it's first parameter we pass the rendered font and in the second we pass a pair of co-ordinates which mark the origin point of the font.

Background

```
# Load background image from file
background = pygame.image.load("animated_street.png")
```

```
# Fill the surface with the background image loaded earlier
self.surface.blit(self.background, (0, 0))
```

There are two steps to creating a background. We load the image (outside the game loop for performance). In the Game Loop, we draw the image using the `blit()` function. The `blit()` function must be in the game loop as it needs to re draw itself as the other objects move.

Sound

```
# Background music plays continuously
pygame.mixer.music.load('background_music.wav')
pygame.mixer.music.set_volume(0.3)
pygame.mixer.music.play(-1)
```

The steps to playing background music.

1. Load the sound into memory.
2. Set the volume. The volume range is .0 - 1.
3. Play the sound in a continuous loop. The play argument `-1` loops the music.

```
# Stop background sound
pygame.mixer.music.stop()
```

4. Stop playing. We stop playing the background sound right before the crash sound that ends the game.

```
# Play crash sound
pygame.mixer.Sound('crash.wav').play()
```

We use a one-line function from the Pygame Mixer library to play a crash sound once a collision has occurred.

Player Code

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

import pygame
from pygame.locals import *
from config import Config

# 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__()

        self.config = Config()

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

        # Create a surface rectangle a bit smaller than the image
        # This imitates crashing by overlapping the images
        self.surf = pygame.Surface((40, 75))

        # Gets the rectangle area of the Surface
        # Starts at the bottom of the screen
        self.rect = self.surf.get_rect(center=(160, 520))

    # Called each time through the Game Loop
    def move(self):

        # Read the keyboard to see if any keys pressed
        pressed_keys = pygame.key.get_pressed()

        # Keep the player on the screen
        # The sprite can't move past the left edge of the surface
        if self.rect.left > 0:

            # Left arrow key pressed, move left 5 pixels
            if pressed_keys[K_LEFT]:
                self.rect.move_ip(-5, 0)

        # The sprite can't move past the right edge of the surface
        if self.rect.right < self.config.SCREEN_WIDTH:

            # Right arrow key pressed, move right 5 pixels
            if pressed_keys[K_RIGHT]:
                self.rect.move_ip(5, 0)
```

```
52
53              # Right arrow key pressed, move right 5 pixels
54              if pressed_keys[K_RIGHT]:
55                  self.rect.move_ip(5, 0)
```

Explanation player.py

```
# Create a surface rectangle a bit smaller than the image
# This imitates crashing by overlapping the images
self.surf = pygame.Surface((40, 75))
```

We make this small change to the player file. Making the rectangle slightly smaller than the image allows for an overlap which better imitates a car crash.

Enemy Code

```
1  '''
2      Name: enemy.py
3      Author:
4      Date:
5      Purpose: All logic for the enemy's car is in this class
6  '''
7
8  import pygame
9  import random
10 from pygame.locals import *
11 from config import Config
12
13 # Define the enemy class and methods
14
15
16 class Enemy(pygame.sprite.Sprite):
17     # Construct an Enemy object
18     def __init__(self):
19
20         # Construct an enemy object from Sprite class
21         super().__init__()
22
23         self.config = Config()
24
25         # Load an image from file
26         self.image = pygame.image.load("enemy.png")
27
28         # Create a surface rectangle a bit smaller than the image
29         # This imitates crashing by overlapping the images
30         self.surf = pygame.Surface((42, 70))
31
```

```
        # Create a rectangle with a random X location
        # Stay 40 pixels from the left and right edge
        self.rect = self.surf.get_rect(center=(
            random.randint(
            40,
            self.config.SCREEN_WIDTH-40),
            0))

    # Method to move the object
    def move(self):

        # Move the sprite down SPEED pixels at a time
        self.rect.move_ip(0, self.config.speed)

        # When the sprite reaches the botton of the surface,
        # Move to the top, random center location on the X axis, increase score
        if (self.rect.bottom > self.config.SCREEN_HEIGHT):

            # Increment the score every time the player dodges an oncoming car
            self.config.score += 1

            # Move the enemy back to the top, with a random X position
            self.rect.top = 0
            self.rect.center = (random.randint(
                40, self.config.SCREEN_WIDTH - 40), 0)
```

Explanation enemy.py

```
# Create a surface rectangle a bit smaller than the image
# This imitates crashing by overlapping the images
self.surf = pygame.Surface((42, 70))
```

We do the same thing to the Enemy code. Both images overlap, which looks much more like a collision.

```
# Increment the score every time the player dodges an oncoming car
config.score += 1
```

We keep track of how many times the Enemy gets to the bottom of the screen. Each time this happens, the player gets one more point.

Config

```
'''
    Filename: config.py
    Author:
    Date:
    Purpose: Global variables and constants for the entire program
    import config module into all other modules
'''
''' Setup the object data fields '''
class Config:

    def __init__(self):
        # Setup color constants
        self.RED =   (255, 0, 0)
        self.BLACK = (0, 0, 0)
        self.WHITE = (255, 255, 255)

        # Constant for Frames Per Second (FPS)
        self.FPS = 60

        # Constants for screen width
        self.SCREEN_WIDTH = 400
        self.SCREEN_HEIGHT = 600

        # Global Variables for speed and score
        self.speed = 5
        self.score = 0
```

We added a global variable to the config file to store the score and constants to store the colors.

What's Next?

There is much more that can be done with this game. Here are some ideas for you to practice and implement on your own.

- Allow the player to restart the game.
- Keep track of the score between games.
- Multiple enemies spawning after set periods of time. (Similar to how we increased speed after a set period of time)
- Add some additional audio to the game, such as movement sounds (audio that plays when you move the character)
- Add the concept of multiple Lives or a Health bar.
- Variations in the shape and size of the "enemies".

Assignment Submission

Zip up the program files folder and submit in Blackboard.