Part 3 – Pygame Car Crash Tutorial

Contents

Part 3 – Pygame Car Crash Tutorial 1
config.py 1
Car Crash Code 2
Explanation 3
Player Class 6
Explanation 7
Enemy Class 8
Explanation 8
Assignment Submission 9

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.

In this section we're going to move our classes into separate files, add movement, and cover Sprite Grouping.

config.py

```
'''
    Filename: config.py
    Author:
    Date:
    Purpose: Global variables and constants for the entire program
'''
# import config module into all other modules

# Setup global constants and variables for screen size and speed
SCREEN_WIDTH = 400
SCREEN_HEIGHT = 600

# Global variable
speed = 5
```

The config file stores global constants and variables for all the program files. It must be imported into each of the program files.

Car Crash Code

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

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

# Import our classes
import player, enemy, config

# Create a Player and Enemy object
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 if needed
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)

class CarCrash:
    ''' Setup the object data fields '''
    # Setup color and screen size constants
    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()

        # 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():
                # 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)

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

            # 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()
```

Explanation

Some of the constants and variables have been moved to the separate class files. They need to referenced as shown above: `config.SCREEN_WIDTH`. The player and enemy object construction reference the player and enemy class files.

```
# Create a Player and Enemy object
# The file has to be referenced with dot notation
# creating an object from a class file
player = player.Player()
enemy = enemy.Enemy()
```

Note that the object creation references each class file.

```
# Create Sprites Groups, add Sprites to Groups
# A separate enemies group is created,
# to allow for more enemy Sprites later on if needed
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)
```

Here we've created "groups" for our sprites. A Sprite group is sort of like a classification. It's much easier to deal with 2 or 3 groups, rather than having to deal with dozens or even hundreds of sprites. Keeping them in one group allows us to easily access every sprite in that group.

In the example above, we've created two groups one called `enemy` and the other called `all_sprites`. This code doesn't have more than one enemy, but since multiple enemies could easily be added here, we've created a separate group for it. To add a Sprite to a group, use the `add()` function.

```
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)

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

        # Redraw the surface
        pygame.display.update()

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

The commands shown above are all in the game loop, so they are repeating continuously. First the update and move functions for both the Enemy and Player objects are called.

1. Refresh the screen using the `surface.fill(WHITE)` function.
2. Call the draw and move functions for all of the sprites, drawing them to the screen.
3. The `pygame.display.update()` command updates the screen with all the commands that have occurred up-till this point.
4. The `tick()` method makes sure the Game Loop repeats only 60 times per second.

Player Class

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

import pygame
from pygame.locals import *

# config.py contains global variables and constants
import config

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

    # 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 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
        pressedKeys = 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 at a time
            if pressedKeys[K_LEFT]:
                self.rect.move_ip(-5, 0)

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

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

Explanation

Let's look at the new items in the player class.

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

With this line, the Player sprite appears at the bottom of the screen as he should. Previously, he would appear in the top left corner.

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

    # Read the keyboard to see if any keys pressed
    pressedKeys = 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 at a time
        if pressedKeys[K_LEFT]:
            self.rect.move_ip(-5, 0)

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

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

We define a `move` method for the Player class that controls the movement of the player. When this function is called, it checks to see if any keys are pressed down or not.

The if statements check for 2 keys, LEFT and RIGHT. If the if statement proves true, then the `move_ip()` method is called on `Player.rect` moving it in a certain direction. The `move_ip()` takes two parameters, the first representing the distance to be moved in the X direction and second, the distance to be moved in the Y direction.

The two if statements, `if self.rect.left > 0:` and `if self.rect.left < config.SCREEN_WIDTH:` ensure that the player isn't able to move off screen.

Enemy Class

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

import pygame
from pygame.locals import *
import random

# config.py contains global variables and constants
import config

# Define the enemy class and methods
class Enemy(pygame.sprite.Sprite):
     # Construct an Enemy object
    def __init__(self):

        # Construct an enemy object from Sprite class
        super().__init__()

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

        # Create a surface rectangle the same size as the image
        self.surf = pygame.Surface((50, 80))

        # 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, 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, config.speed)

        # When the sprite reaches the botton of the surface,
        # Move to the top, random center location on the X axis
        if (self.rect.bottom > config.SCREEN_HEIGHT):
            self.rect.top = 0
            self.rect.center = (random.randint(30, 370), 0)
```

Explanation

```
# When the sprite reaches the botton of the surface,
# Move to the top of the window, a random location on the X axis
if (self.rect.bottom > config.SCREEN_HEIGHT):
    self.rect.top = 0
    self.rect.center = (random.randint(30, 370), 0)
```

The enemy class also has a move method. It first calls the `move_ip()` function, moving the Enemy object down by **speed** pixels at a time. Next it checks to see if the top of the Enemy has reached the end of the screen. If True, it resets it back to the top of screen at a random location on the X axis.

Assignment Submission

Zip up the program files folder and submit in Blackboard.