Computer Animation Lab: 06

Contents

Creating a Shooter Game in Pygame

In this task, we will create a shoot game. The game consists of a player at the bottom of the screen that shoot bullets. Enemies are spawned from the top of the screen. When a bullet collides with an enemy, both are deleted from the screen.

Shooter.py

```
import pygame as pg
import sys
import random

pg.init()

window_width = 800
window_height = 600

window = pg.display.set_mode((window_width, window_height))
pg.display.set_caption("Shooter Game")

clock = pg.time.Clock()


class Player:
    def __init__(self):
        self.speed = 5
        self.x = window_width // 2
        self.y = window_height - 100
        self.img = pg.image.load('player.png').convert_alpha()
        self.rect = self.img.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y
        self.x_change = 0

    def update(self):
        self.x += self.x_change
        self.rect.x = self.x

    def draw(self, window):
        window.blit(self.img, self.rect)
```

```
class Enemy:
    def __init__(self):
        self.x = random.randint(0, window_width - 20)
        self.y = -30
        self.img = pg.image.load("enemy.png").convert_alpha()
        self.rect = self.img.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y
        self.speed = 5

    def update(self):
        self.y += self.speed
        self.rect.y = self.y

    def draw(self, window):
        window.blit(self.img, self.rect)

    def reset(self):
        self.y = -30
        self.rect.y = self.y
        self.x = random.randint(0, window_width - 20)
        self.rect.x = self.x


class Bullet:
    def __init__(self, x, y):
        self.width = 5
        self.height = 10
        self.speed = 7
        self.x = x
        self.y = y
        self.img = pg.image.load("bullet.png").convert_alpha()
        self.rect = self.img.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y

    def update(self):
        self.y -= self.speed
        self.rect.y = self.y

    def draw(self, window):
        window.blit(self.img, self.rect)
```

```
enemy_lst = []
bullet_lst = []
enemy_timer = 0
spawn_enemy_timer = 2000

def check_collisions(obj1, obj2):
    return obj1.rect.colliderect(obj2.rect)

player = Player()

while True:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            pg.quit()
            sys.exit()

        if event.type == pg.KEYDOWN:
            if event.key == pg.K_SPACE:
                bullet_x = player.rect.centerx
                bullet_y = player.rect.top
                bullet_lst.append(Bullet(bullet_x, bullet_y))

    keys = pg.key.get_pressed()
    if keys[pg.K_LEFT] and player.rect.left > 50:
        player.x_change = -player.speed
    elif keys[pg.K_RIGHT] and player.rect.right < window_width - 50:
        player.x_change = player.speed
    else:
        player.x_change = 0

    player.update()

    for bullet in bullet_lst:
        bullet.update()

bullet_lst = [bullet for bullet in bullet_lst if bullet.rect.bottom > 0]

current_time = pg.time.get_ticks()
if current_time - enemy_timer > spawn_enemy_timer:
    new_enemy = Enemy()
    enemy_lst.append(new_enemy)
    enemy_timer = current_time
```

```
for enemy in enemy_lst:
    enemy.update()

for bullet in bullet_lst[:]:
    for enemy in enemy_lst[:]:
        if check_collisions(bullet, enemy):
            bullet_lst.remove(bullet)
            enemy_lst.remove(enemy)
            break

for enemy in enemy_lst[:]:
    if enemy.rect.top > window_height:
        enemy_lst.remove(enemy)

window.fill((0, 0, 0))

player.draw(window)

for bullet in bullet_lst:
    bullet.draw(window)

for enemy in enemy_lst:
    enemy.draw(window)

pg.display.update()

clock.tick(60)
```

Step-by-step code

- Import $pygame$ and $random$ modules and initialize $pygame$

```
import pygame as pg
import random

# Initialize Pygame modules
pg.init()
```

- Define the width and the height of the window and define a margin constant for the objects to be off from the borders of the window. Also, initialize the clock variable to track the time.

```
# Set the window width and height
window_width = 800
window_height = 600
window = pg.display.set_mode((window_width, window_height))

# Offsets from the left and right borders of the window
MARGIN = 20

# Window title
pg.display.set_caption("Shooter Game")

# An object to help to track the time
clock = pg.time.Clock()
```

- Create the player class with the following attributes and functions:
  - $speed$: speed of moving left and right.
  - x and y: the x-y positions on the window.
  - img: the image file of the player.

- $rect$: the rectangular object that embodies the image. This is important for detecting collisions.
- x_change: the change in the x value to move left or right.
- $update()$: to reflect the movement along the x-axis.
- $draw()$: to draw the image on pygame's window.

```
class Player:
    def __init__(self):
        """Initialize player attributes."""
        # The speed of moving left and right
        self.speed = 5
        # x-y positions on the window
        self.x = window_width // 2
        self.y = window_height - 100
        # Read the image and convert the pixels to alpha format for better performance
        self.img = pg.image.load('player.png').convert_alpha()
        # Get the rectangle object that embodies the image.
        # This is useful for detecting collisions
        self.rect = self.img.get_rect()
        # Set the rectangle's positions
        self.rect.x = self.x
        self.rect.y = self.y
        # A variable to track the left and right positions
        self.x_change = 0

    def update(self):
        """Move the player."""
        self.x += self.x_change
        # Always update the rect, because it's
        # needed for the collision detection.
        self.rect.x = self.x

    def draw(self, window):
        """Draw the player."""
        window.blit(self.img, self.rect)
```

- Create the player class with the following attributes and functions:
  - $speed$: speed of moving left and right.
  - x and y: the x-y positions on the window.
  - img: the image file of the enemy.
  - $rect$: the rectangular object that embodies the image. This is important for detecting collisions.
  - x_change: the change in the x value to move left or right.
  - $update()$: to reflect the movement along the x-axis.
  - $draw()$: to draw the image on pygame's window.
  - $reset()$: to redraw the enemy again from the top of the window

```
class Enemy:
    def __init__(self):
        self.x = random.randint(0, window_width - MARGIN-40)
         self.y = -30
        self.img = pg.image.load("enemy.png").convert_alpha()
        self.rect = self.img.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y
        self.speed = 2

    def update(self):
        self.y += self.speed
        self.rect.y = self.y

    def draw(self, window):
        window.blit(self.img, self.rect)

    def reset(self):
        self.y = -30
        self.rect.y = self.y
        self.x = random.randint(0, window_width - MARGIN-40)
        self.rect.x = self.x
```

- Create the bullet class with attributes and functions as before.

```
class Bullet:
    def __init__(self, x, y):
        """Initialize the bullet attributes."""
        self.width = 5
        self.height = 10
        self.speed = 7
        self.x = x
        self.y = y
        self.img = pg.image.load("bullet.png").convert_alpha()
        self.rect = self.img.get_rect()
        self.rect.x = self.x
        self.rect.y = self.y

    def update(self):
        """Move the bullet upwards."""
        self.y -= self.speed
        self.rect.y = self.y

    def draw(self, window):
        window.blit(self.img, self.rect)
```

- Create lists to store the enemies and bullets objects.

```
# The list of enemies
enemy_lst = []
# The list of bullets
bullet_lst = []
```

- Define a timer to spawn enemies regularly.

```
# A timer to spawn new enemies
enemy_timer = 0
spawn_enemy_timer = 1000
```

- A function to detect collisions between two objects.

```
# Function to detect it the bullet collide with the enemy
def check_collisions(obj1, obj2):
```

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

# The game loop
while True:
    … code continues here…
```

- Create a player object and start the game main loop

- Listen for two events: the first when the user quits the game and the second when the user is pressing on the space button to shoot bullets. When a user press on the space button, we define the x-y positions of the start of the bullet with respect to the player's position. Then, we append the bullet object with its x and y values.

```
# Listen for events
for event in pg.event.get():
    # If the user closes the window
    if event.type == pg.QUIT:
        pg.quit()  # uninitialize all pygame modules

    # Check if the user is pressing on the space key
    if event.type == pg.KEYDOWN:
        if event.key == pg.K_SPACE:
            # Create a new bullet object
            bullet_x = player.rect.centerx  # x = center of the player
            bullet_y = player.rect.top  # y = top part of the player
            bullet_lst.append(Bullet(bullet_x, bullet_y))  # add the bullet to the list
```

- To allow the user to move the player using the left and right buttons, we get a dictionary object of the keys pressed on the keyboard. If the left button is pressed and the player's rectangle are not touching the borders (MARGIN), then move the object to the left by decrementing x_change value. If the right button is pressed and the player's rectangle are not touching the borders, then move the object to the right by incrementing its x_change value. Otherwise, leave the x_change to 0. Finally, call $player.update()$ to update its position on the window.

```
keys = pg.key.get_pressed()
if keys[pg.K_LEFT] and player.rect.left > MARGIN:
    player.x_change = -player.speed
elif keys[pg.K_RIGHT] and player.rect.right < window_width - MARGIN:
    player.x_change = player.speed
# Otherwise, do not change the player position
else:
    player.x_change = 0

# Update player when the left or right keys are pressed
player.update()
```

- Update the bullets' position by iterating over the objects in the list.

```
# Update bullets to move them down
for bullet in bullet_lst:
    bullet.update()
```

- Get the current time in millisecond. If the time difference is more than 1000 millisecond, then create a new enemy. Then, iterate over the list of enemies to update them.

```
# Spawn a new enemy according to current time
current_time = pg.time.get_ticks() # return the time in ms
if current_time - enemy_timer > spawn_enemy_timer:
    # If a specified amount of time is passed, create a new enemy
    new_enemy = Enemy()
    # Add it to the list
    enemy_lst.append(new_enemy)
    # Update the time
    enemy_timer = current_time

# Update enemies
for enemy in enemy_lst:
    enemy.update()
```

- Check for a collision between the bullet and the enemy. If collided, remove the bullet and the enemy from the list.

```
# Check for bullet-enemy collisions
for bullet in bullet_lst:
    for enemy in enemy_lst:
        if check_collisions(bullet, enemy):
            bullet_lst.remove(bullet)
            enemy_lst.remove(enemy)
            break
```

- Remove the enemies and the bullets that go off the screen.

```
# Remove enemies that are off the screen
for enemy in enemy_lst:
    if enemy.rect.top > window_height:
        enemy_lst.remove(enemy)

# Remove bullets that are off-screen
bullet_lst = [bullet for bullet in bullet_lst if bullet.rect.bottom > 0]
```

- Finally, set the window's background to black, draw the player, draw the bullets, draw the enemies, update the display, and set the frames per second to 60.

```
# Fill the background with black
window.fill((0, 0, 0))

# Draw the player
player.draw(window)

# Draw the bullets
for bullet in bullet_lst:
    bullet.draw(window)

# Draw the enemies
for enemy in enemy_lst:
    enemy.draw(window)

# Update the display
pg.display.update()

# Set the FPS
clock.tick(60)
```