Assignment1: 101 Bouncing Images

20181200 권혁채
MAS2011-01
Prof. Yongduek Seo
2022-12-25

A. Introduction

The purpose of this assignment is to understand how to detect collision between objects and screen boundaries, and how to create multiple objects of similar characteristics using python classes. We will be using the 'pygame' package, and our goal is to create 101 class objects with at least 10 different images, and one separate object which will be controlled by the player.

B. Programming

1. Importing Packages, Setting General Variables

```python
import pygame
import os
import numpy as np

# assets 경로 설정
current_path = os.path.dirname(__file__)
assets_path = os.path.join(current_path, 'assets') #specify path to image file

# 게임 윈도우 크기
WINDOW_WIDTH = 1600
WINDOW_HEIGHT = 900

# Pygame 초기화
pygame.init()

# 윈도우 제목
pygame.display.set_caption("Mushrooms and Lumas")

# 윈도우 생성
screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))

# 게임 화면 업데이트 속도
clock = pygame.time.Clock()

# sound
sound = pygame.mixer.Sound(os.path.join(assets_path, 'boing.wav'))
sound.set_volume(0.03)
pygame.mixer.set_num_channels(101)

# 색 정의
SKY = (209, 237, 242)
```

First we must import all python packages necessary for this project. In addition to pygame, we must import os to gain information and access to image files stored in our computer, and numpy to generate random integers. Next, we define general variables that will be used throughout the program. Said variables include:

current_path: holds the location in which this python code is located

assets_path: goes into the 'assets' folder within the current directory
WINDOW_WIDTH: width of screen on which the objects will be created
WINDOW_HEIGHT: height of screen on which objects will be created
pygame.init(): initialize pygame to start manipulation
pygame.display.set_caption() : set caption of screen
screen: screen used to display objects, size is defined
clock: keeps track of update speed of game
sound: reads sound file that will be used during the game. We can change volume using sound.set_volume() and play the sound with sound.play()
pygame.mixer.set_num_channels(101): since we will be making 101 bouncing objects, 101 sounds must be played at the same time. Therefore we should create 101 separate channels
SKY: RGB values of color 'skyblue'

2. Create Class

```
class Fren:
    def __init__(self,name,index):
        self.img = pygame.image.load(os.path.join(assets_path, name + '.png')) #join connects two strings
        self.x = np.random.randint(200,900)
        self.y = np.random.randint(200,500)
        self.dx = np.random.randint(-11,12)
        self.idx = index
        while self.dx == 0:
            self.dx = np.random.randint(-11,12)
        self.dy = self.dx

    def update(self):
        self.x+=self.dx
        self.y+=self.dy

        if (self.x + self.img.get_width()) > WINDOW_WIDTH or (self.x) < 0:
            pygame.mixer.Channel(self.idx).play(sound)
            self.dx *= -1
        if (self.y + self.img.get_height()) > WINDOW_HEIGHT or (self.y) < 0:
            pygame.mixer.Channel(self.idx).play(sound)
            self.dy *= -1

    def draw(self, screen):
        screen.blit(self.img, [self.x, self.y])
```

Since we will be creating 101 different objects, it is more efficient to create a class function instead of defining, initializing, and manipulating each object separately. Notice that the init function takes a name and index as arguments. Each object will have the following components:

self.img: loads image
self.x, self.y: x,y coordinate of object generation point
self.dx, self.dy: speed of object along x,y axis
self.idx = index of current object. This will be used when we decide which channel to play our sound effect from

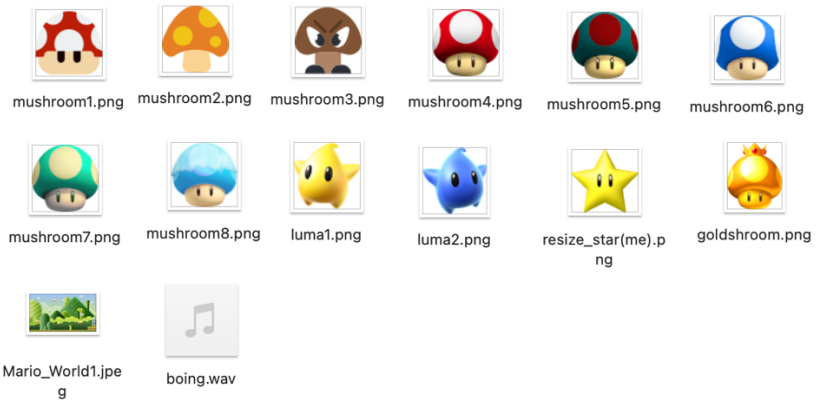
x,y coordinates and speed will be set to a random value. The y speed of each

object will be the same as its x speed, and in order to make sure there are no static objects, they will be reallocated if they are 0.

The update(self) function increases the current position of the object by its speed before it is drawn in order to make it look like it has moved from its previous position. Each x,y coordinate represents the top left corner of the object, so if the current position of the x coordinate + get_widht() is the same as the screen boundary, the speed of the x coordinate will be reversed. The same goes for the y coordinate. Each time an object comes into contact with the screen boundary, a sound effect will be played from its respective channel.

The draw(self, screen) function copies the pre-loaded image using the blit() function onto the screen, placing it on the x, y coordinates stored as its class components.

3. Creating 101 Objects



These are the assets we will use to create our class objects. All assets are in the asset file within the current directory.

```
# 배경 이미지 로드
background_image = pygame.image.load(os.path.join(assets_path, 'Mario_World1.jpeg')) #path to image file

# 이미지 로드
frenList = []
for i in range(101):
    if i < 80:
        index = (i//10)+1
        Shroom = Fren('mushroom'+str(index),i)
        frenList.append(Shroom)
    elif i < 100:
        index = (i//90)+1
        Luma = Fren('luma'+str(index),i)
        frenList.append(Luma)
    else:
        star = Fren('resize_star(me)',i)
        star.dx = 5
        star.dy = 5
        frenList.append(star)
```

First we load our background image and store it in the background_image variable. Next, we create an empty frenList which will store the 101 class objects. The first 80 objects will be mushroom1 ~mushroom8, 10 each. The next 20 will be luma1 and luma2, and the last class object will be

resize_star(me).

4. Creating Player-Controllable Object

```
player_image = pygame.image.load(os.path.join(assets_path, 'goldshroom.png')) #keyboard image
player_x = int(WINDOW_WIDTH / 2)
player_y = int((WINDOW_HEIGHT*8.5) / 10)
player_dx = 0
player_dy = 0

# 게임 종료 전까지 반복
done = False

# 게임 반복 구간
while not done:
    # 이벤트 반복 구간
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

        elif event.type == pygame.KEYDOWN: #event = when pressing keys
            if event.key == pygame.K_LEFT:
                player_dx = -7.3
            elif event.key == pygame.K_RIGHT:
                player_dx = 7.3
            elif event.key == pygame.K_UP:
                player_dy = -7.3
            elif event.key == pygame.K_DOWN:
                player_dy = 7.3
        # 키가 놓일 경우
        elif event.type == pygame.KEYUP: #event = when releasing keys, so it DOESN'T MOVE when
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                player_dx = 0
            elif event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                player_dy = 0
```

The Player-Controllable Object will take on the image of ‘goldshroom.png’, and its initial location will be half of the total screen width, and 8.5 10ths of the screen height. The object will stay static at the beginning of the program, so dx, dy will be initialized to 0. This object will move 7.3 pixels per frame whenever an event occurs, or in this case when the player presses any of the arrow keys on the keyboard. When each key is released, the object will stop in place, so its respective dx, dy value will become 0.

5. Game Logic

```
# 게임 로직 구간
player_x += player_dx
player_y += player_dy

# 화면 삭제 구간

for i in range(101):
    fren = frenList[i]
    fren.update()

# 윈도우 화면 채우기
screen.fill(SKY) #clean screen first

screen.blit(background_image, background_image.get_rect()) #blit land

screen.blit(player_image, [player_x, player_y])
# 화면 그리기 구간
for i in range(101):
    fren = frenList[i]
    fren.draw(screen)
```

The position of the Player-Controllable Object will be updated depending on its current speed, and the position of all 101 objects inside the frenList will also be updated using the for loop. After that, we will first fill the screen with a the SKY color predefined at the beginning, copy the background image and Player-Controllabe Object, and then copy all updated 101 images to the screen.

6. Updated Screen, Set FPS, and Quit Game

```
    # 화면 업데이트
    pygame.display.flip()

    # 초당 60 프레임으로 업데이트
    clock.tick(60)

# 게임 종료
pygame.quit()
```

Finally we will update the screen using pygame.display.flip(), and set the framerate to 60 fps using clock.tick(60). If the done variable ever becomes true, the while loop will be terminated and the program will quit.

Result on next page


Mushrooms and Lumas
wallpapersaddicts.com