Pygame Tutorials

By Saurav Singh

Pygame

- It is a module in python used to create video games.
- This library includes several modules for playing sound, drawing graphics, handling mouse inputs, etc.
- Pygame.init() – It is used to initialize the pygame library.
- pygame.display.set_mode((width, height)) – use to set dimensions of the screen.

```
import pygame as py
#initializing the pygame
py.init()

#creating the screen
screen = py.display.set_mode((800,600))
```

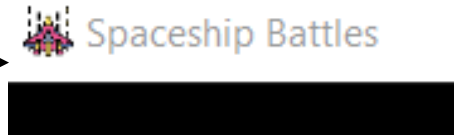
Game loop

- The screen of the game window is getting disappeared after running the code.
- So we have to use while loop which keeps the code running and the display of the screen will remain forever until we close it.

```
#Game loop
running = True
while running:
    for event in py.event.get():
        if event.type == py.QUIT: #when we press the cross button
            running = False       #while loop becomes False and gets terminated
                                  #until then the screen remains visible
```

Changing title, logo and background color

```
# Changing title and logo
py.display.set_caption("Spaceship Battles")
icon = py.image.load('spaceship.png')
py.display.set_icon(icon)
```



- For background color we have to add that in the **gaming loop** because we want that color forever. So, anything you want persistent just put it into the gaming loop.
- And after every change in the loop you have to use pygame.display.update() to update the changes.

```
#Game loop
running = True
while running:
    for event in py.event.get():
        if event.type == py.QUIT: #when we press the cross button
            running = False       #while loop becomes False and gets terminated
                                  #until then the screen remains visible

    #setting background color
    screen.fill((255,0,0))
    py.display.update()           #to update the changes
```

Adding Image

- Screen.blit(image,(x,y)) – blit means that we are drawing something on the screen.

```
#adding player image
playerImg = py.image.load('player.png')
x_axis = 370
y_axis = 480
#creating a function to draw the player
def player():
    screen.blit(playerImg,(x_axis,y_axis))
```

- Calling this player function in the game loop because we want this image for ever.

Movement Mechanism

- To move the image we just have to add/subtract the values of x/y coordinate.
- If you want to move left/right then subtract/add the x values.
- If you want to move up/down then subtract/add the y values.

Key Pressed events

- First we have to check whether any key is pressed or not by event.type==pygame.KEYDOWN
- If key is pressed then specify conditions –
  - event.key==pygame.K_LEFT – for left key.
  - event.key==pygame.K_RIGHT – for right key.
- The release of key is determined by - event.type==pygame.KEYUP

```python
# Game loop
if __name__ == '__main__':
    running = True
    while running:
        for event in py.event.get():
            if event.type == py.QUIT:  # when we press the cross button
                running = False  # while loop becomes False and gets terminated
                # until then the screen remains visible

            if event.type == py.KEYDOWN:  # checking if any key is pressed or not
                if event.key == py.K_LEFT:  # if left key is pressed
                    x_change = -0.3
                if event.key == py.K_RIGHT:  # if right key is pressed
                    x_change = 0.3
            if event.type == py.KEYUP:  # when key is released
                if event.key == py.K_LEFT or event.key == py.K_RIGHT:
                    x_change = 0
        x_axis += x_change

        # setting background color
        screen.fill((0, 0, 0))
        player(x_axis, y_axis)
        py.display.update()  # to update the changes
```

Adding Boundaries

- So the approach for boundaries is that if the x coordinate go less than 0 make it 0 again AND if it goes greater than 736(because the size of spaceship is 64px i.e. 800-64), make it 736 again.

```
if x_axis <= 0:
    x_axis = 0
elif x_axis >= 736:
    x_axis = 736
player(x_axis, y_axis)
py.display.update()  # to update the changes
```

```
# adding bounudaries
if player_x <= 0:
    player_x = 0
elif player_x >= 736:
    player_x = 736
```

Game loop

Creating enemies

- We will use random module because we want the enemy to appear at different places every time it dies.
- So we will set the dimensions of enemy by random.

```
# Enemy
enemyImg = py.image.load('enemy.png')
enemy_x = rd.randint(0,800) #because we want the enemy in different places every time
enemy_y = rd.randint(50,150)
# creating a function to draw the enemy
def enemy(x, y):
    screen.blit(enemyImg, (x, y))
```

Movement of enemy

- We have used random number for enemy dimensions. So for its movement we have to initialize the change variable with some value.
- If we give 0 then 0 will be added to the dimension and it will not move but when we give 0.3 and add it to the dimension then the enemy will move and then we will put the if else.

```
# Enemy
enemyImg = py.image.load('enemy.png')
enemy_x = rd.randint(0,780) #because we want the enemy in different places every time
enemy_y = rd.randint(50,150)
# creating a function to draw the enemy
def enemy(x, y):
    screen.blit(enemyImg, (x, y))
enemy_x_change = 0.4 # to keep moving the enemy even if the value is not 0
enemy_y_change = 30
```

```
# enemy movement
if enemy_x <= 0:
    enemy_x_change = 0.4
    enemy_y += enemy_y_change
elif enemy_x >= 736:
    enemy_x_change = -0.4
    enemy_y += enemy_y_change
```

Game loop

Adding Background Image

- Load the image by pygame.image.load().
- Then use screen.blit(image,(x,y)) in game loop.

Shooting bullets

Things to keep in mind –

- The dimensions of the bullet should be equal to the dimensions of the player(before shooting only) because we want the bullet to be shotted from the player only.
- After shooting, the y coordinate of the bullet should decrease and the x coordinate should remain same from where it was shotted.
- The bullet should not follow the spaceship after firing.

```python
# bullet
# ready - bullet is not visible on the screen
# fired - bullet is visible
bulletImg = py.image.load('bullet.png')
bullet_x = 0
bullet_y = 480  # bullet position = player position
bullet_x_change = 0
bullet_y_change = 6
bullet_state = "ready"
```

```python
def fire_bullet(x, y):
    global bullet_state
    bullet_state = "fired"
    # because we want the bullets to be shooted from the nose of spaceship
    screen.blit(bulletImg, (x+16, y+10))
```

Game loop

```python
if event.key == py.K_SPACE:
    if bullet_state is "ready":  # if bullet is not on screen then only shoot another bullet.
        # when we press space then the currrent value of player_x is stored in bullet_x
        # so after firing it will not follow the player
        bullet_x = player_x
        fire_bullet(bullet_x, bullet_y)
```

```python
# bullet movement
if bullet_y <= 0:
    bullet_y = 480
    bullet_state = "ready"
if bullet_state is "fired":
    fire_bullet(bullet_x, bullet_y)
    bullet_y -= bullet_y_change
```

Collision detection

- The approach is - we will find the distance between bullet and enemy by distance formula and if it is very less then we will reset –
  - bullet_y value i.e 480
  - bullet_state
  - enemy_x
  - enemy_y

```
def is_collision(enemy_x, enemy_y, bullet_x, bullet_y):
    distance = math.sqrt((math.pow(enemy_x-bullet_x, 2)) +
                         (math.pow(enemy_y-bullet_y, 2)))  # distance formula
    if distance < 27:
        return True
    else:
        return False

score = 0
def collision_true():
    global bullet_y, bullet_state, enemy_x, enemy_y, score
    bullet_y = 480
    bullet_state = "ready"
    score += 1
    print(score)
    enemy_x = rd.randint(0, 780)
    enemy_y = rd.randint(50, 150)
```

Game loop

```
# collision detection
collision = is_collision(enemy_x, enemy_y, bullet_x, bullet_y)
if collision is True:
    collision_true()
```

Multiple enemies

```
# Enemy
enemyImg = []
enemy_x = []
enemy_y = []
enemy_x_change =[]
enemy_y_change = []
no_of_enemy = 7
for i in range(no_of_enemy):
    enemyImg.append(py.image.load('enemy.png'))
    # because we want the enemy in different places every time
    enemy_x.append(rd.randint(0, 780))
    enemy_y.append(rd.randint(50, 150))
    enemy_x_change.append(2.5)  # to keep moving the enemy even if the value is not 0
    enemy_y_change.append(30)
```

```
# enemy movement
for i in range(no_of_enemy):
    enemy_x[i] += enemy_x_change[i]
    if enemy_x[i] <= 0:
        enemy_x_change[i] = 2.5
        enemy_y[i] += enemy_y_change[i]
    elif enemy_x[i] >= 736:
        enemy_x_change[i] = -2.5
        enemy_y[i] += enemy_y_change[i]

    # collision detection
    collision = is_collision(enemy_x[i], enemy_y[i], bullet_x, bullet_y)
    if collision is True:
        collision_true()
    enemy(enemy_x[i], enemy_y[i],i)
```

Displaying scores on the screen

- Pygame.font.Font('fontstyle.ttf',size) – to set font.
- Pygame.font.Font.render("text",True,(RGB)) – first render text then blit.

```
# score
score_value = 0
font = py.font.Font('freesansbold.ttf', 32)
score_x = 10
score_y = 10


def show_score(x, y):
    score = font.render("Score:" + str(score_value), True, (255, 255, 255))
    screen.blit(score, (x, y))
```

Adding sound

```
mixer.music.load('background.wav')
mixer.music.play(-1)  # -1 so that it plays forever
```

Background sound

```
bullet_sound = mixer.Sound('bullet.wav')
bullet_sound.play()
```

Bullet sound

```
explosion = mixer.Sound('explosion.wav')
explosion.play()
```

Collision sound

```
over_sound = mixer.Sound('gameover.wav')
over_sound.play()
```

Gameover sound

GameOver

```
# enemy movement
for i in range(no_of_enemy):
    #game over
    if enemy_y[i]>410:
        for j in range(no_of_enemy):
            enemy_y[j] = 2000
        game_over()
        over_sound = mixer.Sound('gameover.wav')
        over_sound.play()
        break
```