Taller de Objetos y Eventos con Pygame

2013
VídeoJuegos con Python
PyGroupUD
Proyecto Curricular de Ingeniería de Sistemas

Taller de Introducción a Pygame

Índice de contenido

Introducción

En esta entrega no introduciremos nuevos conceptos de programación, veremos como usar algunas características de pygame. Los recursos usados para este ejercicio son 4 imágenes una de un fondo, una nave y un ufo, se sugieren los siguientes:

- nave: https://www.dropbox.com/s/fg9ou6xal0i2a7a/ship.png
- fondo: https://www.dropbox.com/s/3klqyg45q1v9oqj/space.png
- ufo: https://www.dropbox.com/s/7u9daz7ylvnx745/ufo.png
- bala: https://www.dropbox.com/s/i9pubh9k7t86nbm/bullet.png

Creando los enemigos

Iniciemos creando la clase “Ufo” la cual representará los enemigos de la nave; al igual que la clase “Ship”, “Ufo” tiene dos métodos, el método __init__ que establecerá los valores iniciales del Ufo y el método update que actualizara estos valores en cada ciclo del juego. El script esta clase queda de la siguiente forma:

```
import pygame
from pygame.sprite import Sprite
from pygame.locals import *

class Ufo(Sprite):
    def __init__(self, cont_size, starfleet_pos, starfleet_size):
        Sprite.__init__(self)
        self.vel = [2,0]
        self.starfleet_pos = starfleet_pos
        self.starfleet_size = starfleet_size
        self.cont_size = cont_size
        self.image = pygame.image.load("imagenes/ufo.png")
        self.rect = self.image.get_rect()
        pos_x = (self.cont_size[0] / starfleet_size) + (self.starfleet_pos * 65)
```

```
        self.rect.move_ip(pos_x, 10)

    def update(self):
        self.rect = self.rect.move(self.vel)
        left_max = (self.starfleet_pos * 65)
         right_max = self.cont_size[0] - (((self.starfleet_size - 1) - self.starfleet_pos) *
65)
        if self.rect.left < left_max or self.rect.right > right_max:
            self.vel[0] = -self.vel[0]
            self.rect.top += 5
```

En esta clase a diferencia de la clase “Ship” la actualización de la posición de los valores no depende de un evento del teclado si no de los valores que representan la posición del objeto, en el método update se establecen dos valores que controlan los cambios de dirección de cada uno de los enemigos:

```
left_max = (self.starfleet_pos * 65)
right_max = self.cont_size[0] - (((self.starfleet_size - 1) - self.starfleet_pos) * 65)
```

left_max será el máximo valor que alcance un enemigo a la izquierda y *right_max* el máximo a la derecha, ambos son determinados en relación *starfleet_pos* que establece cual es la posición del enemigo en la flota y *starfleet_size* que es el tamaño de la flota de enemigos. Cuando un enemigo alcanza estos topes cambia de dirección cambiando el signo de la velocidad en el eje x y al mismo tiempo incrementa el valor que representa su posición en el eje y.

```
if self.rect.left < left_max or self.rect.right > right_max:
            self.vel[0] = -self.vel[0]
            self.rect.top += 5
```

Agregando Ufos en el juego

Veamos como cambia el script principal a agregar los ufos al juego:

```
import sys, pygame
from pygame.locals import *
from ship import Ship
from ufo import Ufo

size = width, height = 800, 600
screen = pygame.display.set_mode(size)

def main():
    pygame.init()

    background_image = pygame.image.load("imagenes/space.png")
    background_rect = background_image.get_rect()

    pygame.display.set_caption( "asteroids" )

    ship = Ship(size)
    ufos = []
    for i in range(0,10):
        ufos.append(Ufo(size, i, 10))

    while 1:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        ship.update()
        screen.blit(background_image, background_rect)
        screen.blit(ship.image, ship.rect)

        for ufo in ufos:
            ufo.update()
            screen.blit(ufo.image, ufo.rect)

        pygame.display.update()
        pygame.time.delay(10)

if __name__ == '__main__':
    main()
```

Al igual que para poder usar la clase Ship debemos importar la clase Ufo para poder usarla en ele script:

```
from ufo import Ufo
```

Ahora agreguemos una lista de ufos a nuestro script, creando primero una lista vacia y agregando uno a uno mediante un ciclo la cantidad deseada a la lista:

```
ufos = []
    for i in range(0,10):
        ufos.append(Ufo(size, i, 10))
```

Los parámetros pasados al constructor del Ufo son según se definió en el método __init__ del mismo:

- cont_size,: tamaño del contenedor (800 x 600).
- starfleet_pos: posición en la lista de ufos.
- starfleet_size: tamaño de la lista de Ufos.

Estos valores se usan para determinar en donde es dibujado el ufo en la pantalla.

El siguiente cambio en el script es la actualización de cada ufo en la lista y el pintado del mismo de acuerdo a su actualización:

```
for ufo in ufos:
     ufo.update()
     screen.blit(ufo.image, ufo.rect)
```

Al ejecutar el script el resultado se observa en las siguientes imágenes:

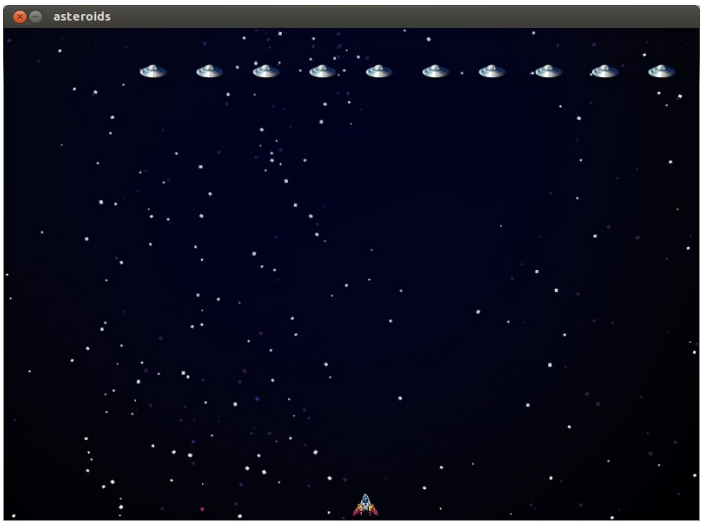

Ilustración 1: Ufos agregados al script

Ilustración 2: Ufos en descenso

Ahora agreguemos las balas a nuestro juego, para esto creamos una clase “Bullet” de la que serán agregados varios objetos a nuestra nave. Veamos como queda la clase “Bullet”:

```
class Bullet(Sprite):
    def __init__(self, pos):
        Sprite.__init__(self)
        self.vel = [0,-2]
        self.image = pygame.image.load("imagenes/bullet.png")
        self.rect = self.image.get_rect()
        self.rect.move_ip(pos[0], pos[1])

    def update(self):
        self.rect = self.rect.move(self.vel)
```

En esta clase el método update es automático y no depende de ningún evento ni controla ningún aspecto del juego, y en el __init__ la velocidad en el segundo componente es negativo para hacer que la bala suba por la pantalla.

Ahora modifiquemos la clase Ship para asociar las balas a la nave, para esto agregaremos un atributo a nuestra clase que contendrá una lista de objetos de tipo Bullet.

```
import pygame
from pygame.sprite import Sprite
from pygame.locals import *
from bullet import Bullet

class Ship(Sprite):
    def __init__(self, cont_size):
        Sprite.__init__(self)
        self.puntos = 0
        self.vida = 100
        self.cont_size = cont_size
        self.image = pygame.image.load("imagenes/ship.png")
        self.rect = self.image.get_rect()
        self.rect.move_ip(cont_size[0]/2, cont_size[1]-35)
        self.bullets = []
```

```
    def update(self):
        teclas = pygame.key.get_pressed()
        if teclas[K_LEFT] and self.rect.left > 0:
            self.rect.x -= 10
        elif teclas[K_RIGHT] and self.rect.right < self.cont_size[0]:
            self.rect.x += 10
        elif teclas[K_UP]:
            pos = [self.rect.centerx-10, self.rect.centery-20]
            self.bullets.append(Bullet(pos))
        elif teclas[K_DOWN]:
            pass
```

el atributo lo asignamos con la siguiente línea:

```
self.bullets = []
```

y la asociación de la nueva bala con el atributo y el evento de K_UP en las siguientes líneas del método update:

```
elif teclas[K_UP]:
    pos = [self.rect.centerx-10, self.rect.centery-20]
    self.bullets.append(Bullet(pos))
```

y en el script principal en el ciclo del juego trabajamos con las balas de la nave para su actualización y pintura, quedando el script como sigue:

```
import sys, pygame
from pygame.locals import *
from ship import Ship
from ufo import Ufo

size = width, height = 800, 600
screen = pygame.display.set_mode(size)

def main():
    pygame.init()

    background_image = pygame.image.load("imagenes/space.png")
    background_rect = background_image.get_rect()
```

```
    pygame.display.set_caption( "asteroids" )

    ship = Ship(size)
    ufos = []
    for i in range(0,10):
        ufos.append(Ufo(size, i, 10))

    while 1:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        ship.update()
        screen.blit(background_image, background_rect)
        screen.blit(ship.image, ship.rect)

        for bullet in ship.bullets:
            bullet.update()
            if bullet.rect.top <= 0:
                ship.bullets.remove(bullet)
            screen.blit(bullet.image, bullet.rect)

        for ufo in ufos:
            ufo.update()
            screen.blit(ufo.image, ufo.rect)

        pygame.display.update()
        pygame.time.delay(10)

if __name__ == '__main__':
    main()
```

donde el trabajo con las balas se hace en las siguientes líneas:

```
for bullet in ship.bullets:
            bullet.update()
            if bullet.rect.top <= 0:
                ship.bullets.remove(bullet)
            screen.blit(bullet.image, bullet.rect)
```

El resultado se ven en la siguiente imagen:

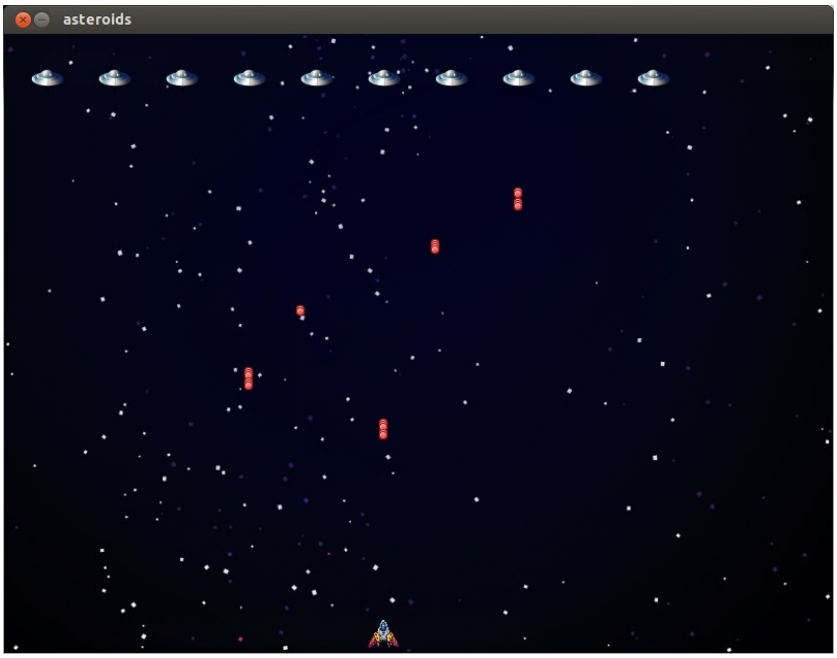

Ilustración 3: juego con balas y ufos

Trabajando con Colisiones

Vamos a trabajar ahora con las colisiones entre las balas y los ufos, para esta tarea trabajaremos con la operación colliderect del rect asociando al rect de las balas con el rect de los ufos de la siguiente forma:

```
for bullet in ship.bullets:
            bullet.update()
            for ufo in ufos:
                if bullet.rect.colliderect(ufo.rect):
                    ship.bullets.remove(bullet)
                    ufos.remove(ufo)
            if bullet.rect.top <= 0:
                ship.bullets.remove(bullet)
            screen.blit(bullet.image, bullet.rect)
```

quedando el script principal así:

```
import sys, pygame
from pygame.locals import *
from ship import Ship
from ufo import Ufo

size = width, height = 800, 600
screen = pygame.display.set_mode(size)

def main():
    pygame.init()

    background_image = pygame.image.load("imagenes/space.png")
    background_rect = background_image.get_rect()

    pygame.display.set_caption( "asteroids" )

    ship = Ship(size)
    ufos = []
    for i in range(0,10):
        ufos.append(Ufo(size, i, 10))

    while 1:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        ship.update()
        screen.blit(background_image, background_rect)
        screen.blit(ship.image, ship.rect)

        for bullet in ship.bullets:
            bullet.update()
            for ufo in ufos:
                if bullet.rect.colliderect(ufo.rect):
                    ship.bullets.remove(bullet)
                    ufos.remove(ufo)
            if bullet.rect.top <= 0:
                ship.bullets.remove(bullet)
            screen.blit(bullet.image, bullet.rect)
```

```
        for ufo in ufos:
            ufo.update()
            screen.blit(ufo.image, ufo.rect)

        pygame.display.update()
        pygame.time.delay(10)

if __name__ == '__main__':
    main()
```

el resultado del manejo de colisiones queda de la siguiente manera:

Ilustración 4: trabajo con colisiones

Trabajo a realizar

Sigue las instrucciones de modificación del ejercicio indicadas en cada uno de los puntos siguientes y genera un zip que contenga los archivos solicitados (scripts de python) y los recursos (imágenes, sonidos, archivos de texto, etc) necesarios para su ejecución.

1. Agrega 10 enemigos más de manera que los ufos se presenten en dos filas de 10 guarda el script como taller03_01.py
2. Agrega un enemigo con un comportamiento diferente de tal manera que descienda más rápido que los demás y que para eliminarlo requiera más de un impacto, guarda el script como taller03_02.py

Bibliografía consultada:

“Pygame documentatión”.
<http://www.pygame.org/docs/index.html>
(01 Sep 2013)

“Python documentation”.
<http://www.python.org/doc/>
(01 Sep 2013)