



Taller de sonidos 2 y transformaciones con Pygame

2013

VídeoJuegos con Python

PyGroupUD

Proyecto Curricular de Ingeniería de Sistemas

Taller de sonidos 2 y transformaciones con Pygame

Índice de contenido

Introducción.....	3
Creando la base del juego.....	3
Rotando y moviendo la nave.....	6
La rotación.....	7
Ahora el desplazamiento.....	9
Agregando sonido al acelerar.....	11
Agregando disparos.....	13
Agregando los asteroides.....	18
Explotando los asteroides.....	22
Terminando el juego.....	26
Trabajo a realizar.....	30
Bibliografía consultada:.....	31

Introducción

En esta entrega agregaremos a un video juego un sonido de fondo, también trabajaremos con sonidos que dependan de un evento y añadiremos transformaciones a nuestras imágenes del juego, en esta ocasión rotaciones. Para el manejo de los recursos de imágenes y sonidos, se sugieren los siguientes links:

- nave: <https://www.dropbox.com/s/fg9ou6xal0i2a7a/ship.png>
- fondo: <https://www.dropbox.com/s/3klqyg45q1v9oqi/space.png>
- asteroide: <https://www.dropbox.com/s/k2ecprbtwbaqqym/asteroid.png>
- bala: <https://www.dropbox.com/s/i9pubh9k7t86nbm/bullet.png>
- explosión: <https://www.dropbox.com/s/8skxhixplfnuic9/explosion.png>
- sonido de disparo: <https://www.dropbox.com/s/lgex3te2cylqcdp/disparo.wav>
- sonido de explosión:
<https://www.dropbox.com/s/n7fuw5cxtgn6itm/explosion.wav>
- sonido de impulso:
<https://www.dropbox.com/s/tttipudm25buzqi/impulso.wav>
- musica de fondo: <https://www.dropbox.com/s/fk7i3hee9f8hp0f/viento.mp3>

Creando la base del juego

El video juego crear es el famoso “asteroids” en el que una nave debe dispararle a unos asteroides que aparecen aleatoriamente en el juego. Para la creación de la base del juego crearemos la pantalla del mismo con un fondo que ambiente el juego en el espacio y la nave que aparecerá en el centro de la pantalla, para el juego se ejecutará en todo momento un sonido de fondo que en los recursos se llama “viento.mp3”. Veamos como queda la pantalla del juego:

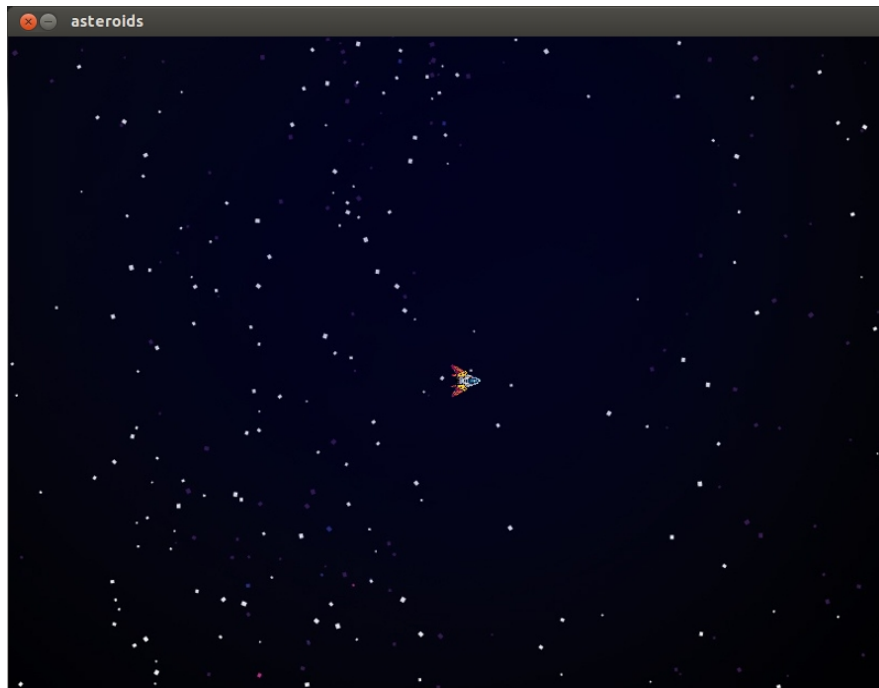


Ilustración 1: ventana inicial del juego

Para obtener este resultado crearemos dos scripts, el script principal del juego y el script de la clase ship, en el script principal definiremos la ventana, cargamos la imagen de fondo, instanciamos la nave y cargamos y reproducimos el sonido.

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

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

def main():
    pygame.init()
    pygame.mixer.music.load("sonidos/viento.mp3")
    pygame.mixer.music.play(1)
    background_image = pygame.image.load("imagenes/space.png")
```

```

background_rect = background_image.get_rect()
pygame.display.set_caption( "asteroids" )
ship = Ship(size)

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)

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

if __name__ == '__main__':
    main()

```

La creación de la ventana, la carga de la imagen de fondo y la instanciación de la nave ya la hemos trabajado en entregas anteriores, la carga y la reproducción del sonido de fondo lo hacemos usando el mixer.music de pygame con las siguientes instrucciones:

```

pygame.mixer.music.load("sonidos/viento.mp3")
pygame.mixer.music.play(1)

```

En la primera línea cargamos el recurso del sonido y la segunda iniciamos la ejecución del mismo, el parámetro “loops” con valor de 1 hará que sonido se ejecute indefinidamente.

Para el script de la clase nave lo único que haremos es cargar la imagen de tal manera que se posicione en el centro de la pantalla que la instancie y prepararemos el manejo de los eventos del teclado.

```

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

```

```

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

    def update(self):
        teclas = pygame.key.get_pressed()

        if teclas[K_LEFT]:
            pass
        elif teclas[K_RIGHT]:
            pass
        elif teclas[K_UP]:
            pass
        elif teclas[K_DOWN]:
            pass

```

Rotando y moviendo la nave

Cuando se oprima la tecla de flecha a derecha la nave debe rotar hacia la derecha, cuando se oprima la tecla de flecha izquierda debe rotar a izquierda y cuando se oprima la flecha arriba la nave debe acelerar desplazándose hacia donde este el frente de la nave. Para lograr todo esto haremos uso de `pygame.transform.rotate` que nos permite generar rotación de imágenes y algo de matemáticas para que el desplazamiento se genere en el ángulo en el que rotó la nave.

Primero veamos como queda el script de la nave que permite todo esto:

```

import pygame, math
from pygame.sprite import Sprite
from pygame.locals import *

class Ship(Sprite):
    def __init__(self, contenedor):
        Sprite.__init__(self)
        self.angulo = 0

```

```

self.puntos = 0
self.vida = 100
self.vel = [0,0]
self.contenedor = contenedor
self.base_image = pygame.image.load("imagenes/ship.png")
self.image = self.base_image
self.rect = self.image.get_rect()
self.rect.move_ip(contenedor[0]/2, contenedor[1]/2)

def update(self):
    teclas = pygame.key.get_pressed()
    if teclas[K_LEFT]:
        self.rotar(2)
    elif teclas[K_RIGHT]:
        self.rotar(-2)
    elif teclas[K_UP]:
        self.acelerar()
    elif teclas[K_DOWN]:
        pass

    self.vel[0] *= 0.99
    self.vel[1] *= 0.99
    self.rect = self.rect.move(self.vel)
    self.rect.x = self.rect.x % self.contenedor[0]
    self.rect.y = self.rect.y % self.contenedor[1]


def acelerar(self):
    self.vel[0] += math.cos(math.radians((self.angulo)%360))
    self.vel[1] -= math.sin(math.radians((self.angulo)%360))

def rotar(self, angulo):
    self.angulo += angulo
    centerx = self.rect.centerx
    centery = self.rect.centery
    self.image = pygame.transform.rotate(self.base_image, self.angulo)
    self.rect = self.image.get_rect()
    self.rect.centerx = centerx
    self.rect.centery = centery

```

La rotación

Empecemos con la rotación, como ya mencionamos `pygame.transform.rotate` permite rotar



una imagen en un ángulo determinado, pero se debe tener cuidado para no deformar la imagen, para lograr esto modificaremos la clase Ship en cuanto al manejo de la imagen de la nave, ya que la rotación debe hacerse siempre en base a la imagen original y no una previamente rotada, para esto generamos dos atributos: “image” y “base_image” de la siguiente manera:

```
self.base_image = pygame.image.load("imagenes/ship.png")
self.image = self.base_image
```

Y creamos un atributo “angulo” que nos servirá como parámetro para el método de rotación de pygame, por defecto este atributo tendrá un valor cero (0).

```
self.angulo = 0
```

El método rotar de la clase Ship recibe como parámetro los grados de incremento del ángulo para la rotación de la nave, este parámetro lo sumaremos al atributo ángulo de la clase y este lo pasaremos al método “rotate” de “pygame.transform”. Parece simple pero puede haber un pequeño problema y es que la nave al rotar aparecerá en las coordenadas cero, cero de la pantalla y no en la posición que debe aparecer; para corregir este problema copiamos las propiedades “centerx” y “centery” del área rectangular de la nave antes de rotarla:

```
centerx = self.rect.centerx
centery = self.rect.centery
```

Luego realizamos la rotación:

```
self.image = pygame.transform.rotate(self.base_image, self.angulo)
```

Ahora obtenemos nuevamente el área rectangular de la nave y la desplazamos al lugar que debe aparecer sobre-escribiendo las propiedades “centerx” y “centery” por las iniciales:

```
self.rect = self.image.get_rect()
self.rect.centerx = centerx
self.rect.centery = centery
```

El resultado final del método rotar es el siguiente:


```
def rotar(self, angulo):
    self.angulo += angulo
    centerx = self.rect.centerx
    centery = self.rect.centery
    self.image = pygame.transform.rotate(self.base_image, self.angulo)
    self.rect = self.image.get_rect()
    self.rect.centerx = centerx
    self.rect.centery = centery
```

Ahora el desplazamiento

Ya resuelto el problema de la rotación crearemos el método acelerar el cual debe convertir el ángulo de rotación de la nave en un vector que modifique el atributo vel (velocidad) de la nave, para esto pasaremos a radianes el ángulo de rotación y para obtener el componente x del ángulo calculamos el coseno del mismo y el componente en y calculamos el seno del mismo; el componente x lo sumamos a la velocidad en su componente x y el componente y lo restamos del componente y de la velocidad, resultando así el método:

```
def acelerar(self):
    self.vel[0] += math.cos(math.radians((self.angulo)%360))
    self.vel[1] -= math.sin(math.radians((self.angulo)%360))
```

Como vemos el método acelerar únicamente modifica la velocidad, pero no controla el mantener la nave dentro de la pantalla ni la desaceleración de la nave cuando no se oprime la flecha arriba, esta responsabilidad se la daremos al método “update” de la clase, para mantener la nave en la ventana principal del juego cuando esta la desborde por derecha deberá aparecer por izquierda y viceversa, de la misma forma cuando la desborde por arriba aparecerá por la parte inferior y viceversa, esto lo logramos reescribiendo las propiedades x y y del área rectangular asociada a la imagen de la nave en base las dimensiones de la ventana de la siguiente forma:

```
self.rect.x = self.rect.x % self.contenedor[0]
self.rect.y = self.rect.y % self.contenedor[1]
```

Para desacelerar la nave sobre-escribimos los dos componentes de la velocidad de la misma multiplicándolas por 0,99

```
self.vel[0] *= 0.99
self.vel[1] *= 0.99
```

Ahora asociamos los métodos a los eventos de teclado correspondientes así:

```
teclas = pygame.key.get_pressed()
    if teclas[K_LEFT]:
        self.rotar(2)
    elif teclas[K_RIGHT]:
        self.rotar(-2)
    elif teclas[K_UP]:
        self.acelerar()
    elif teclas[K_DOWN]:
        pass
```

En total el método “update” queda de la siguiente manera:

```
def update(self):
    teclas = pygame.key.get_pressed()
    if teclas[K_LEFT]:
        self.rotar(2)
    elif teclas[K_RIGHT]:
        self.rotar(-2)
    elif teclas[K_UP]:
        self.acelerar()
    elif teclas[K_DOWN]:
        pass

    self.vel[0] *= 0.99
    self.vel[1] *= 0.99
    self.rect = self.rect.move(self.vel)
    self.rect.x = self.rect.x % self.contenedor[0]
    self.rect.y = self.rect.y % self.contenedor[1]
```

Y como queda el resultado en ejecución es el siguiente:

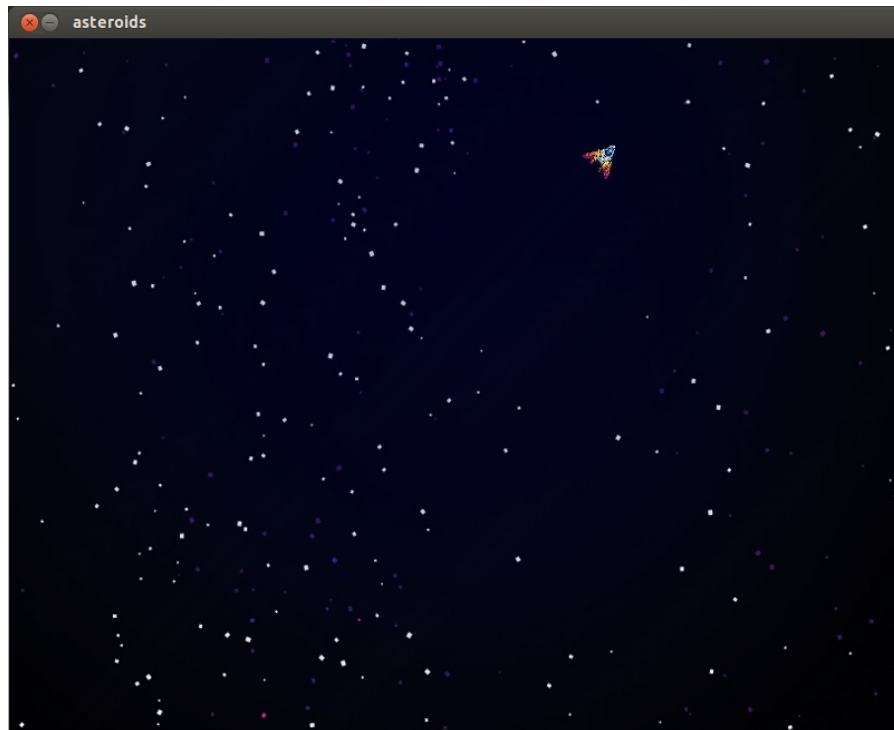


Ilustración 2: Nave rotada y desplazada

Agregando sonido al acelerar

Ahora para completar el efecto de la aceleración de la nave cuando se ejecute el método reproduciremos el sonido asociado al evento. Primero cargaremos en el método “__init__” el sonido como un parámetro de la clase Ship y modificaremos su volumen de la siguiente manera:

```
self.impulso = pygame.mixer.Sound('sonidos/impulso.wav')
self.impulso.set_volume(0.05)
```

La modificación del volumen con el método “set_volume” recibe como parámetro el valor del volumen el cual va entre 0.0 y 1.0. Para reproducir el sonido agregaremos una línea en

el método acelerar de la clase Ship

```
self.impulso.play()
```

En total las modificaciones del script de la clase Ship quedan de la siguiente manera:

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

class Ship(Sprite):
    def __init__(self, contenedor):
        Sprite.__init__(self)
        self.angulo = 0
        self.puntos = 0
        self.vida = 100
        self.vel = [0,0]
        self.contenedor = contenedor
        self.base_image = pygame.image.load("imagenes/ship.png")
        self.image = self.base_image
        self.rect = self.image.get_rect()
        self.rect.move_ip(contenedor[0]/2, contenedor[1]/2)
        self.impulso = pygame.mixer.Sound('sonidos/impulso.wav')
        self.impulso.set_volume(0.05)

    def update(self):
        teclas = pygame.key.get_pressed()
        if teclas[K_LEFT]:
            self.rotar(2)
        elif teclas[K_RIGHT]:
            self.rotar(-2)
        elif teclas[K_UP]:
            self.acelerar()
        elif teclas[K_DOWN]:
            pass

        self.vel[0] *= 0.99
        self.vel[1] *= 0.99
        self.rect = self.rect.move(self.vel)
        self.rect.x = self.rect.x % self.contenedor[0]
        self.rect.y = self.rect.y % self.contenedor[1]

    def acelerar(self):
        self.impulso.play()
```

```
self.vel[0] += math.cos(math.radians((self.angulo)%360))
self.vel[1] -= math.sin(math.radians((self.angulo)%360))

def rotar(self, angulo):
    self.angulo += angulo
    centerx = self.rect.centerx
    centery = self.rect.centery
    self.image = pygame.transform.rotate(self.base_image, self.angulo)
    self.rect = self.image.get_rect()
    self.rect.centerx = centerx
    self.rect.centery = centery
```

Agregando disparos

Vamos a darle a nuestra nave la capacidad de disparar, para esto cuando el usuario presione la tecla espaciadora la nave lanzara una ráfaga de balas que más adelante servirán para destruir los asteroides, veamos como se ve esto en pantalla:

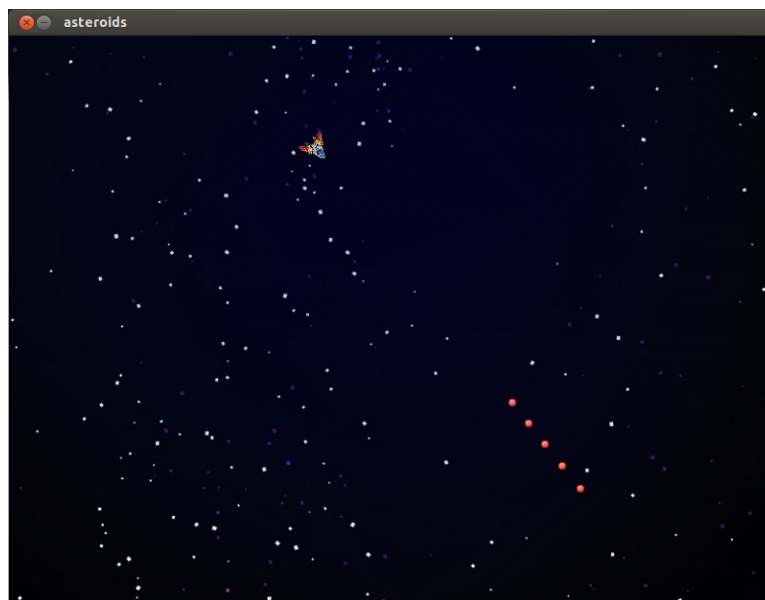


Ilustración 3: Nave disparando

Para lograr ese efecto, primero crearemos la clase Bullet que maneja las imágenes y movimiento de cada bala, en esta clase se controla el alcance que tiene cada una ya que como al igual que la nave al salir por un lado de la ventana aparece por el lado contrario, si estas no desapareciesen en algún momento tendríamos nuestra ventana llena de balas por todas partes, así que agregaremos el atributo alcance a la bala y en el método update este irá disminuyendo de manera que cuando llegue a cero el script principal la quite del juego. Veamos como queda la clase Bullet:

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

class Bullet(Sprite):
    def __init__(self, pos, angle, vel, cont):
        Sprite.__init__(self)
        self.vel = vel
        self.alcance = 25
        self.contenedor = cont
        self.image = pygame.image.load("imagenes/bullet.png")
        self.rect = self.image.get_rect()
        self.rect.move_ip(pos[0], pos[1])
        self.angulo = angle

    def update(self):
        self.alcance -= 1
        self.vel[0] += math.cos(math.radians((self.angulo)%360))
        self.vel[1] -= math.sin(math.radians((self.angulo)%360))
        self.rect = self.rect.move(self.vel)
        self.rect.x = self.rect.x % self.contenedor[0]
        self.rect.y = self.rect.y % self.contenedor[1]
```

Para asociar las balas a la nave crearemos en la clase Ship el método disparar y lo asociaremos en el método update de esta a la tecla espaciadora, en este método reproduciremos un sonido asociado al evento que al igual que acelerar se asoció a un atributo de la clase de la siguiente manera:

```
self.disparo = pygame.mixer.Sound('sonidos/disparo.wav')
self.disparo.set_volume(0.05)
```

También crearemos una lista vacía llamada bullets que contendrá todas las balas instanciadas y activas en el juego:

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

Para lanzar la bala en la misma dirección de la nave debemos calcular la posición de la bala y su velocidad en relación a la dirección y velocidad de la nave, esto lo haremos trabajando con el ángulo de la rotación de la nave al igual que lo hicimos para el desplazamiento de esta:

```
vector = [0,0]
vector[0] += math.cos(math.radians((self.angulo)%360))
vector[1] -= math.sin(math.radians((self.angulo)%360))
pos = [self.rect.x + self.radio, self.rect.y + self.radio]
vel = [self.vel[0] + 6 * vector[0], self.vel[1] + 6 * vector[1]]
```

Y así lo que nos resta es instanciar la bala y agregarla a la lista de balas de la nave:

```
self.bullets.append(Bullet(pos,self.angulo, vel, self.contenedor))
```

El método disparar completo luce de la siguiente manera:

```
def disparar(self):
    self.disparo.play()
    vector = [0,0]
    vector[0] += math.cos(math.radians((self.angulo)%360))
    vector[1] -= math.sin(math.radians((self.angulo)%360))
    pos = [self.rect.x + self.radio, self.rect.y + self.radio]
    vel = [self.vel[0] + 6 * vector[0], self.vel[1] + 6 * vector[1]]
    self.bullets.append(Bullet(pos,self.angulo, vel, self.contenedor))
```

Y nuestra clase Ship completa luce así:

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

class Ship(Sprite):
    def __init__(self, contenedor):
        Sprite.__init__(self)
```

```

self.angulo = 0
self.radio = 8
self.puntos = 0
self.vida = 100
self.vel = [0,0]
self.bullets = []
self.carga = True
self.contenedor = contenedor
self.base_image = pygame.image.load("imagenes/ship.png")
self.image = self.base_image
self.rect = self.image.get_rect()
self.rect.move_ip(contenedor[0]/2, contenedor[1]/2)
self.impulso = pygame.mixer.Sound('sonidos/impulso.wav')
self.impulso.set_volume(0.05)
self.disparo = pygame.mixer.Sound('sonidos/disparo.wav')
self.disparo.set_volume(0.05)

def update(self):
    teclas = pygame.key.get_pressed()
    if teclas[K_LEFT]:
        self.rotar(2)
    elif teclas[K_RIGHT]:
        self.rotar(-2)
    elif teclas[K_UP]:
        self.acelerar()
    elif teclas[K_SPACE]:
        self.disparar()

    self.vel[0] *= 0.99
    self.vel[1] *= 0.99
    self.rect = self.rect.move(self.vel)
    self.rect.x = self.rect.x % self.contenedor[0]
    self.rect.y = self.rect.y % self.contenedor[1]

def disparar(self):
    self.disparo.play()
    vector = [0,0]
    vector[0] += math.cos(math.radians((self.angulo)%360))
    vector[1] -= math.sin(math.radians((self.angulo)%360))
    pos = [self.rect.x + self.radio, self.rect.y + self.radio]
    vel = [self.vel[0] + 6 * vector[0], self.vel[1] + 6 * vector[1]]
    self.bullets.append(Bullet(pos, self.angulo, vel, self.contenedor))

```



```

def acelerar(self):
    self.impulso.play()
    self.vel[0] += math.cos(math.radians((self.angulo)%360))
    self.vel[1] -= math.sin(math.radians((self.angulo)%360))

def rotar(self, angulo):
    self.angulo += angulo
    centerx = self.rect.centerx
    centery = self.rect.centery
    self.image = pygame.transform.rotate(self.base_image, self.angulo)
    self.rect = self.image.get_rect()
    self.rect.centerx = centerx
    self.rect.centery = centery

```

Ahora agreguemos el renderizado de las balas al script principal, aquí en el ciclo de ejecución principal agregamos las siguientes líneas, en estas recorreremos la lista de balas de la nave controlando el alcance de cada una y si este es cero (0) las retiramos de la lista:

```

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

```

Quedando nuestro script principal de la siguiente manera:

```

import sys, pygame
from pygame.locals import *
from ship import Ship

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

def main():
    pygame.init()
    pygame.mixer.init()
    pygame.mixer.music.load("sonidos/viento.mp3")
    pygame.mixer.music.play(1)
    background_image = pygame.image.load("imagenes/space.png")
    background_rect = background_image.get_rect()

    pygame.display.set_caption( "asteroids" )

```

```

ship = Ship(size)

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

    screen.blit(background_image, background_rect)

    ship.update()

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

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

if __name__ == '__main__':
    main()

```

Agregando los asteroides

Ahora agregaremos los asteroides a nuestro juego, estos deben tener una velocidad y una rotación aleatoria, para eso crearemos la clase Asteroid que maneja el comportamiento de estos. Para definir la velocidad aleatoria lo haremos de la siguiente manera:

```
self.vel = [random.randint(-2,2), random.randint(-2,2)]
```

Y para la rotación de manera aleatoria definiremos el incremento del ángulo así:

```
self.rotacion = random.randint(-20,20)
```

Toda la lógica del movimiento del asteroide la controlará el método update de la clase, el cual maneja la rotación y desplazamiento del mismo de manera similar a como se maneja la

de la nave, el método queda de la siguiente manera:

```
def update(self):
    self.angulo += self.rotacion
    centerx = self.rect.centerx
    centery = self.rect.centery
    self.image = pygame.transform.rotate(self.base_image, self.angulo)
    self.rect = self.image.get_rect()
    self.rect.centerx = centerx
    self.rect.centery = centery
    self.rect = self.rect.move(self.vel)
    self.rect.x = self.rect.x % self.contenedor[0]
    self.rect.y = self.rect.y % self.contenedor[1]
```

Y la clase completa queda así:

```
import pygame, random, math
from pygame.sprite import Sprite
from pygame.locals import *

class Asteroid(Sprite):
    def __init__(self, cont):
        Sprite.__init__(self)
        self.vel = [random.randint(-2,2), random.randint(-2,2)]
        self.contenedor = cont
        self.angulo = 0
        self.rotacion = random.randint(-20,20)
        self.base_image = pygame.image.load("imagenes/asteroid.png")
        self.image = self.base_image
        self.rect = self.image.get_rect()
        self.rect.move_ip(random.randint(0, self.contenedor[0]),
random.randint(0, self.contenedor[1]))

    def update(self):
        self.angulo += self.rotacion
        centerx = self.rect.centerx
        centery = self.rect.centery
        self.image = pygame.transform.rotate(self.base_image, self.angulo)
        self.rect = self.image.get_rect()
        self.rect.centerx = centerx
        self.rect.centery = centery
        self.rect = self.rect.move(self.vel)
        self.rect.x = self.rect.x % self.contenedor[0]
```

```
self.rect.y = self.rect.y % self.contenedor[1]
```

Para que los asteroides aparezcan en la ventana del juego el script principal debe definir una lista inicialmente vacía para agregar los mismos:

```
asteroids = []
```

Y un condicional que los agregue aleatoriamente controlando que su número no supere los diez (10) asteroides:

```
if random.randint(0,100) % 25 == 0 and len(asteroids) < 10:  
    asteroids.append(Asteroid(size))
```

Para su renderizado en pantalla, un ciclo que los dibuje de la siguiente manera:

```
for asteroid in asteroids:  
    asteroid.update()  
    screen.blit(asteroid.image, asteroid.rect)
```

Así nuestro script principal queda así:

```
import sys, pygame, random  
from pygame.locals import *  
from ship import Ship  
from asteroid import Asteroid  
  
size = width, height = 800, 600  
screen = pygame.display.set_mode(size)  
  
def main():  
    pygame.init()  
    pygame.mixer.init()  
    pygame.mixer.music.load("sonidos/viento.mp3")  
    pygame.mixer.music.play(1)  
    background_image = pygame.image.load("imagenes/space.png")  
    background_rect = background_image.get_rect()  
    pygame.display.set_caption( "asteroids" )  
  
    ship = Ship(size)  
    asteroids = []  
  
    while 1:
```

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

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

if random.randint(0,100) % 25 == 0 and len(asteroids) < 10:
    asteroids.append(Asteroid(size))

for asteroid in asteroids:
    asteroid.update()
    screen.blit(asteroid.image, asteroid.rect)

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

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

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

Veamos ahora como queda el renderizado del juego hasta ahora:



Ilustración 4: Juego con los asteroides

Explotando los asteroides

Ahora haremos que al colisionar una bala con un asteroide el asteroide explote, reproduciendo un sonido de la explosión y eliminando la bala y el asteroide del juego. Para esto modificaremos el método “__init__” de la clase Asteroid agregando el sonido como atributo de la misma:

```
self.explosion = pygame.mixer.Sound('sonidos/explosion.wav')
self.explosion.set_volume(0.05)
```

Y agregaremos el método explotar que se encargara de cambiar la imagen del asteroide por una de una explosión y reproducir el sonido:

```
def explotar(self):
    self.image = pygame.image.load("imagenes/explosion.png")
    self.explosion.play()
```

Así queda el script modificado de la clase Asteroid:

```
import pygame, random, math
from pygame.sprite import Sprite
from pygame.locals import *

class Asteroid(Sprite):
    def __init__(self, cont):
        Sprite.__init__(self)
        self.vel = [random.randint(-2,2), random.randint(-2,2)]
        self.contenedor = cont
        self.angulo = 0
        self.rotacion = random.randint(-20,20)
        self.base_image = pygame.image.load("imagenes/asteroid.png")
        self.image = self.base_image
        self.rect = self.image.get_rect()
        self.rect.move_ip(random.randint(0,self.contenedor[0]),
                          random.randint(0,self.contenedor[1]))
        self.explosion = pygame.mixer.Sound('sonidos/explosion.wav')
        self.explosion.set_volume(0.05)

    def update(self):
        self.angulo += self.rotacion
        centerx = self.rect.centerx
        centery = self.rect.centery
        self.image = pygame.transform.rotate(self.base_image, self.angulo)
        self.rect = self.image.get_rect()
        self.rect.centerx = centerx
        self.rect.centery = centery
        self.rect = self.rect.move(self.vel)
        self.rect.x = self.rect.x % self.contenedor[0]
        self.rect.y = self.rect.y % self.contenedor[1]

    def explotar(self):
        self.image = pygame.image.load("imagenes/explosion.png")
        self.explosion.play()
```

Y en el renderizado de los asteroides en la ventana del juego se debe modificar para manejar estas colisiones:

```
for asteroid in asteroids:
```

```
asteroid.update()
screen.blit(asteroid.image, asteroid.rect)
for bullet in ship.bullets:
    if asteroid.rect.colliderect(bullet.rect):
        ship.bullets.remove(bullet)
        if asteroid in asteroids:
            asteroid.explotar()
            screen.blit(asteroid.image, asteroid.rect)
            asteroids.remove(asteroid)
```

Resultando nuestro script principal así:

```
import sys, pygame, random
from pygame.locals import *
from ship import Ship
from asteroid import Asteroid

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

def main():
    pygame.init()
    pygame.mixer.init()
    pygame.mixer.music.load("sonidos/viento.mp3")
    pygame.mixer.music.play(1)
    background_image = pygame.image.load("imagenes/space.png")
    background_rect = background_image.get_rect()
    pygame.display.set_caption( "asteroids" )

    ship = Ship(size)
    asteroids = []

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

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

        if random.randint(0,100) % 25 == 0 and len(asteroids) < 10:
            asteroids.append(Asteroid(size))
```



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

for asteroid in asteroids:
    asteroid.update()
    screen.blit(asteroid.image, asteroid.rect)
    for bullet in ship.bullets:
        if asteroid.rect.colliderect(bullet.rect):
            ship.bullets.remove(bullet)
            if asteroid in asteroids:
                asteroid.explotar()
                screen.blit(asteroid.image, asteroid.rect)
                asteroids.remove(asteroid)

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

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

Veamos un renderizado del juego hasta aquí:



Ilustración 5: Explotando asteroides

Terminando el juego

Tan solo nos resta manejar las colisiones de la nave para restar puntos de vida a la misma, esto lo haremos modificando el script principal de la siguiente manera:

```
if ship.rect.colliderect(asteroid.rect):  
    ship.vida -= 10  
    if asteroid in asteroids:  
        asteroid.explotar()  
        screen.blit(asteroid.image, asteroid.rect)  
        asteroids.remove(asteroid)
```

Aquí al detectar una colisión de la nave con los asteroides, hace explotar el asteroide y resta diez (10) puntos de vida a la nave.

Lo que nos falta para terminar es renderizar texto que nos indique los puntos obtenidos, el

nivel de vida y si esta llega a cero (0) nos indique el fin del juego; para esto crearemos en el script dos tipos de fuente, la que indica los puntos y la vida y otra que nos servira para mostrar el fin del juego, para esto usaremos `pygame.font.Font` para crear las fuentes:

```
fuentes = pygame.font.Font (None, 45)
...
fuentes_go = pygame.font.Font (None, 100)
```

La diferencia entre ambas es únicamente el tamaño de la misma. Estas instancias de la clase `Font` nos servirán para renderizar los textos requeridos:

```
texto_puntos = fuentes.render("Puntos: "+str(ship.puntos), 1, (250, 250, 250))
texto_vida = fuentes.render("Vida: "+str(ship.vida), 1, (250, 250, 250))
...
texto_fin = fuentes_go.render("FIN DEL JUEGO", 1, (250, 0, 0))
```

En este renderizado los parámetros requeridos son el texto a renderizar, el segundo parámetro el argumento `antialias` es un valor lógico, si es cierto los caracteres tendrán bordes lisos y el tercero corresponde a la tupla (red, green, blue) para el color del texto, con todo esto nuestro script principal queda así:

```
import sys, pygame, random
from pygame.locals import *
from ship import Ship
from asteroid import Asteroid

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

def main():
    pygame.init()
    pygame.mixer.init()
    pygame.mixer.music.load("sonidos/viento.mp3")
    pygame.mixer.music.play(1)
    background_image = pygame.image.load("imagenes/space.png")
    background_rect = background_image.get_rect()
    pygame.display.set_caption("asteroids")
    ship = Ship(size)

    asteroids = []
```

```

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

    screen.blit(background_image, background_rect)
    fuente = pygame.font.Font(None, 45)
    texto_puntos = fuente.render("Puntos: "+str(ship.puntos), 1, (250, 250, 250))
    texto_vida = fuente.render("Vida: "+str(ship.vida), 1, (250, 250, 250))
    fuente_go = pygame.font.Font(None, 100)
    texto_fin = fuente_go.render("FIN DEL JUEGO", 1, (250, 0, 0))

    ship.update()

    if random.randint(0, 100) % 25 == 0 and len(asteroids) < 10:
        asteroids.append(Asteroid(size))

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

    for asteroid in asteroids:
        asteroid.update()
        screen.blit(asteroid.image, asteroid.rect)
        for bullet in ship.bullets:
            if asteroid.rect.colliderect(bullet.rect):
                ship.bullets.remove(bullet)
                ship.puntos += 1
                if asteroid in asteroids:
                    asteroid.explotar()
                    screen.blit(asteroid.image, asteroid.rect)
                    asteroids.remove(asteroid)

        if ship.rect.colliderect(asteroid.rect):
            ship.vida -= 10
            if asteroid in asteroids:
                asteroid.explotar()
                screen.blit(asteroid.image, asteroid.rect)
                asteroids.remove(asteroid)

    if ship.vida > 0:
        screen.blit(texto_vida, (600, 50))

```

```
        screen.blit(texto_puntos, (100, 50))
    else:
        screen.blit(texto_fin, (150, 250))

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

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

Obteniendo el siguiente resultado:



Ilustración 6: Renderizando puntos y vida




Ilustración 7: Fin del juego

Trabajo a realizar

Para esta entrega el trabajo a realizar no corresponde a la modificación de este script como tarea ordinaria de la evaluación continua, corresponde al trabajo final del curso, para esto debes armar un juego que posea las siguientes características:

1. Debe manejar transformaciones (como mínimo rotaciones) de imágenes de manera correcta.
2. Debe reproducir musica de fondo durante la ejecución del juego.
3. Debe reproducir sonidos asociados a los eventos del juego.
4. Debe manejar al menos un héroe principal.
5. Debe manejar una lista de enemigos.
6. Debe manejar colisiones.
7. Debe manejar puntos y nivel de vida.

- 
8. Debe renderizar los puntos y nivel de vida en la ventana del juego.
 9. Cuando el juego termine debe renderizar el mensaje fin del juego e información acerca del autor o autores del mismo.
 10. Se debe generar un archivo pdf con el manual de usuario del juego.

Genera un zip que contenga los archivos solicitados (scripts de python) y los recursos (imágenes, sonidos, archivos de texto, etc) necesarios para su ejecución y el manual de usuario solicitado y sube el proyecto al aula en el enlace “Trabajo Final”.

Bibliografía consultada:

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

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