UT 07	Actividad	Tiempo 3 h	**Tipo:** enseñanza –aprendizaje **Taller:** 2 alumnos grupo
Título	**POO Creación de videojuego con pygame**		
Nombres y Apellidos			

Objetivo

El objetivo de esta práctica es entender y poner en práctica los conocimientos sobre programación orientada a objetos mediante la creación de un videojuego utilizando para ello la librería pygame.

Criterios de evaluación

- 3.1 Elabora programas informáticos, utilizando lenguajes de programación orientados a objetos y aplicando las técnicas generales de la programación.
- 3.5 Se han implementado programas informáticos utilizando alguno de los lenguajes de programación orientados a objetos, verificando su correcto funcionamiento y utilizando las técnicas adecuadas de programación.

Procedimiento

Cada ejercicio se deberá editar, probar y corregir en la herramienta visual studio code o similar.

Se deberán guardar todos los ejercicios por separado en ficheros .py dentro de la carpeta GIT, en un nuevo repositorio llamado “practica0702_usuario01_usuario02”.

Al concluir los ejercicios, se deberán subir a la plataforma GitHub mediante la herramienta GitHub Desktop en indicar aquí la URL pública del repositorio:

URL:

Además se deberá insertar el código de cada ejercicio en este documento, dentro del recuadro dispuesto a tal efecto.

En este recurso vamos a introducir el desarrollo de un pequeño juego, usando la librería pyGame de Python. Vamos a crear un juego donde tenemos una pelota que va rebotando en los bordes de la ventana y controlamos una plataforma en la parte inferior de la pantalla que hace rebotar la pelota. El jugador tiene que evitar que la pelota llegue al borde inferior de la ventana.

Y ahora que sé para qué sirve, ¿cómo lo pongo en práctica?

Vamos a crear distintos programas que nos vayan acercando a la versión final del juego. Cada programa lo podemos construir en una sesión de clase para que el alumno le de tiempo de ir asimilando el funcionamiento. Empecemos:

Ejercicio 1: Construir la ventana del juego

En esta primera sesión, vamos a crear y configurar la ventana que vamos a usar posteriormente para dibujar los elementos de nuestro juego.
Lo primero que tenemos que entender es la estructura que va a tener los juegos que desarrollemos con pyGame:

1. Se crea y configura la ventana del juego, con los elementos que va a tener nuestro juego.
2. Se comprueban los eventos: Dentro de un bucle, comprobamos los posibles eventos que se han producido, por ejemplo, hemos pulsado el botón de cierre de la ventana y el juego concluye, o hemos pulsado una determinada tecla,…
3. Se actualiza la pantalla: Según la lógica del juego o de algún evento que haya sucedido, se modifican los elementos (se mueve la pelota, se mueve la plataforma,…) y se vuelve a dibujar en la pantalla.

En esta primera sesión vamos a crear un programa, que nos muestra una ventana sin ningún elemento. Este programa puede servir a los alumnos como plantilla para que desarrollen sus propios juegos. El primer ejemplo lo tenemos en el fichero ejercicio1.py y sería el siguiente:

```
import pygame

# Inicialización de Pygame
pygame.init()

# Inicialización de la superficie de dibujo
resolution = (640,480)
ventana = pygame.display.set_mode(resolution)
pygame.display.set_caption("Haxball")

# Bucle principal del juego
jugando = True
while jugando:
    # Comprobamos los eventos
    #Comprobamos si se ha pulsado el botón de cierre de la ventana
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            jugando = False
```

This work is licensed under CC BY NC SA 4.0

```
    # Se pinta la ventana con un color
    # Esto borra los posibles elementos que teníamos anteriormente
    ventana.fill((255, 255, 255))

    # Todos los elementos del juego se vuelven a dibujar
    pygame.display.flip()

    # Controlamos la frecuencia de refresco (FPS)
    pygame.time.Clock().tick(60)

pygame.quit()
```

Veamos con detalle el programa:

1. Es necesario importar el módulo pygame (import pygame) y a continuación se inicializa el módulo (pygame.init()).
2. Creamos y configuramos la ventana del juego:
    - Con pygame.display.set_mode se crea una ventana con el tamaño señalado. Se genera un objeto ventana que representa nuestra ventana de juego.
    - Con pygame.display.set_caption se configura el título de la ventana.
3. La parte central del programa es un bucle que repite las siguientes instrucciones:
    - El método pygame.event.get() nos devuelve una lista con los posibles eventos que han sucedido en el juego. Usamos una instrucción for para comprobar esta lista de eventos.
    - En este ejemplo, comprobamos el evento de pulsar el botón de cierre de la ventana. Esto ocurre cuando el tipo de evento (event.type) es igual al valor pygame.QUIT. Si esta condición ocurre se modificará la variable jugando que hará que el bucle principal del juego termine.
    - En el bucle principal se actúa sobre los elementos de la ventana. En nuestro caso no tenemos ninguno.
    - Se borran los posibles elementos que tengamos, pintando la pantalla de un color: ventana.fill((252, 243, 207)). en este caso usando la notación RGB lo pintamos de amarillo claro.
    - Volvemos a pintar los elementos en su nueva posición: pygame.display.flip(), controlando en todo momento que la frecuencia de refresco de la imagen sea de 60 fps.
4. Si salimos del bucle principal se ha terminado el programa: pygame.quit().

Si ejecutamos el programa: python3 ejemplo1.py, nos debe aparecer una ventana.

This work is licensed under CC BY NC SA 4.0

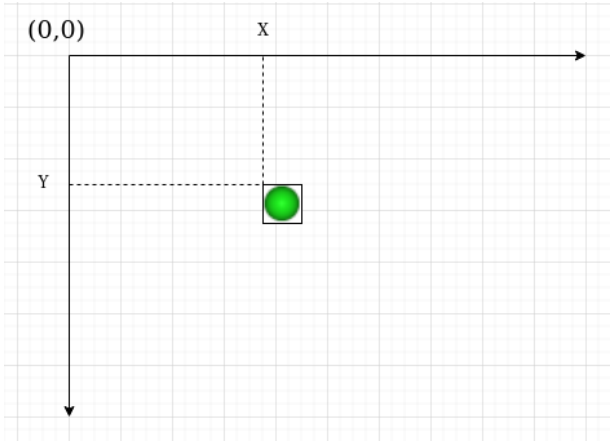
Ejercicio 2: Añadimos la pelota a nuestro juego

En esta sesión vamos a modificar el ejemplo anterior, para incluir el primer objeto a nuestro juego: una pelota que se moverá e irá rebotando por los bordes de la ventana. La pelota va a ser una imagen que tenemos en nuestro directorio: ball.png. El ejercicio2.py quedaría de la siguiente forma:

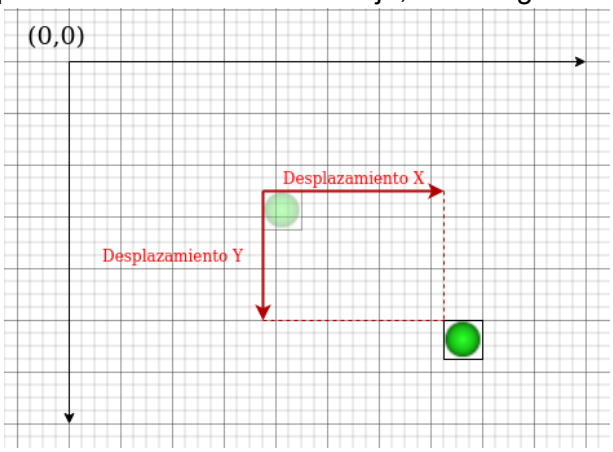
```
import pygame


pygame.init()
ventana = pygame.display.set_mode((640,480))
pygame.display.set_caption("Haxball2")

# Crea el objeto pelota
ball = pygame.image.load("ball.png")

# Obtengo el rectángulo del objeto anterior
ballrect = ball.get_rect()

# Inicializo los valores con los que se van a mover la pelota
speedball = [0,0]

# Pongo la pelota en el centro
ballrect.move_ip(resolution[0]/2, resolution[1]/2)

jugando = True
while jugando:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            jugando = False

    # Muevo la pelota
    ballrect = ballrect.move(speed)

    # Compruebo si la pelota llega a los límites de la ventana
    if ballrect.left < 0 or ballrect.right > ventana.get_width():
        speedball[0] = -speedball[0]

    if ballrect.top < 0 or ballrect.bottom > ventana.get_height():
        speedball[1] = -speedball[1]

    ventana.fill((252, 243, 207))

    # Dibujo la pelota
    ventana.blit(ball, ballrect)
    pygame.display.flip()
    pygame.time.Clock().tick(60)

pygame.quit()
```

Antes de explicar las nuevas instrucciones que hemos introducido en este ejemplo, tenemos que aprender como posicionamos y movemos los objetos dentro de la ventana. Cada objeto se representa por el rectángulo que ocupa, y se posiciona en la ventana indicando la ordenada X (posición horizontal) y la ordenada Y (posición vertical). Tenemos que saber que el origen de coordenadas (0,0) se encuentra en la esquina superior izquierda.

This work is licensed under CC BY NC SA 4.0



Para mover un objeto necesitaremos dos valores: uno para indicar el desplazamiento lateral (ordenada X), si es positivo desplazaremos el objeto a la derecha, si es negativo a la izquierda; y otro para indicar el desplazamiento vertical (ordenada Y), si es positivo se moverá hacia abajo, si es negativo hacía arriba.



Ya podemos explicar las nuevas instrucciones que hemos incluido para mover la pelota:

1. A partir de una imagen png, creamos un objeto imagen (ball = pygame.image.load("ball.png")), pero como hemos comentado vamos a posicionar el rectángulo que ocupa la imagen. Para obtener dicho rectángulo hemos ejecutado ballrect = ball.get_rect().
2. Inicializamos una lista con dos valores, que llamamos speed. El primer valor representa el desplazamiento horizontal, y el segundo el desplazamiento vertical. Lo utilizaremos para mover la pelota.
3. Posicionamos la pelota en el origen de coordenadas: ballrect.move_ip(0,0).
4. Dentro del bucle del juego: movemos la pelota con los datos guardados en la lista speed: ballrect = ballrect.move(speed).
5. Y comprobamos si ha llegado a algún borde:
    - Podemos obtener la posición del rectángulo que representa la pelota con ballrect.left (posición izquierda), ballrect.rigth (posición derecha), ballrect.top (posición superior) y ballrect.bottom (posición inferior).
    - Si la posición izquierda es menor que 0 o la posición derecha es mayor que la anchura de la ventana (ventana.get_width()) habremos tocado los bordes laterales. En esta situación cambiamos el signo del primer dato guardado en speed, es decir, si se movía a la derecha ahora se moverá a la izquierda, y al contrario.

This work is licensed under CC BY NC SA 4.0

- Si la posición superior es menor que 0 o la posición inferior es mayor que la altura de la ventana (ventana.get_height()) habremos tocado los bordes superior o inferior. En esta situación cambiamos el signo del segundo dato guardado en speed, es decir, si se movía hacía abajo ahora se moverá hacía arriba, y al contrario.

6. Finalmente volvemos a pintar la pelota en la ventana (ventana.blit(ball, ballrect)).

Y ya podemos ejecutar el programa (python3 ejercicio2.py).

Otra forma de añadir la pelota es mediante un objeto:

```
#Creamos la clase Ball
class Ball:

#Definimos sus parámetros iniciales
   def __init__(self, xPos =  resolution[0] / 2, yPos = resolution[1] / 2,
xVel = 1, yVel = 1, rad = 15):
       self.x = xPos
       self.y = yPos
       self.dx = xVel
       self.dy = yVel
       self.radius = rad
       self.type = "ball"

#Dibujamos el objeto
   def draw(self, surface):
       pygame.draw.circle(surface, black, (int(self.x), int(self.y)),
self.radius)

#Definimos como se actualizan sus parámetros
   def update(self, gameObjects):
       self.x += self.dx
       self.y += self.dy
       if (self.x <= 0 or self.x >= resolution[0]):
           self.dx *= -1
       if (self.y <= 0 or self.y >= resolution[1]):
           self.dy *= -1
```

Ejercicio 3: Añadimos el jugador 1

En este ejercicio, partiendo de lo que habíamos realizado en la anterior, vamos a añadir otro objeto a nuestro juego: un bate, que controlaremos con el cursor derecho e izquierdo. La pelota al tocar el bate rebotará. El fichero ejercicio3.py quedaría de la siguiente manera:

```
import pygame
```

This work is licensed under CC BY NC SA 4.0

```
pygame.init()
resolution = (640, 480)
ventana = pygame.display.set_mode(resolution)
pygame.display.set_caption("Haxball3")

ball = pygame.image.load("ball.png")
ballrect = ball.get_rect()
speedball = [0,0]
speed1 = [0,0]
ballrect.move_ip(0,0)

# Crea el objeto bate, y obtengo su rectángulo
player1 = pygame.image.load("player1.png")
player1rect = player1.get_rect()

# Pongo el bate en la parte inferior de la pantalla
player1rect.move_ip(0,resolution[1]/2)

jugando = True
while jugando:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            jugando = False

    # Compruebo si se ha pulsado alguna tecla
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        player1rect = player1.move(-2,0)
        speed1[0] = -3
    if keys[pygame.K_RIGHT]:
        player1rect = player1.move(2,0)
        speed1[0] = 3
    if keys[pygame.K_UP]:
        player1rect = player1.move(0,-2)
        speed1[1] = -3
    if keys[pygame.K_DOWN]:
        player1rect = player1.move(0,2)
        speed1[1] = 3

    # Compruebo si hay colisión
    if ballrect.left < 0 or ballrect.right > ventana.get_width():
        speedball[0] = -speedball[0]
    if ballrect.top < 0 or ballrect.bottom > ventana.get_height():
        speedball[1] = -speedball[1]
    ventana.fill((252, 243, 207))
    ventana.blit(ball, ballrect)

    # Dibujo el player1
    ventana.blit(player1, player1rect)

    pygame.display.flip()
    pygame.time.Clock().tick(60)

pygame.quit()
```

Veamos las nuevas instrucciones que hemos añadido:

1. Ahora el objeto bate se crea a partir de otra imagen (player1 = pygame.image.load("player1.png")), obtenemos el rectángulo que ocupa (player1rect = player1.get_rect()) y lo colocamos en su posición inicial (player1rect.move_ip(0,resolution[1]/2)).

This work is licensed under CC BY NC SA 4.0

2. Dentro del bucle, comprobamos si hemos pulsado alguna tecla. Con keys = pygame.key.get_pressed() obtenemos una lista con las teclas que se han pulsado.
3. Si hemos pulsado el cursor izquierdo (if keys[pygame.K_LEFT]:) movemos el bate tres posiciones a la izquierda (player1rect = player1rect.move(-2,0)).
4. Si hemos pulsado el cursor derecho (if keys[pygame.K_RIGHT]:) movemos el bate tres posiciones a la derecha (player1rect = player1rect.move(2,0)).
5. Si hemos pulsado el cursor izquierdo (if keys[pygame.K_UP]:) movemos el bate tres posiciones a la izquierda (player1rect = player1rect.move(0,-2)).
6. Si hemos pulsado el cursor derecho (if keys[pygame.K_DOWN]:) movemos el bate tres posiciones a la derecha (player1rect = player1rect.move(0,2)).
7. Finalmente, volvemos a pintar el bate en la ventana (ventana.blit(player1, player1rect)).

Y ya podemos ejecutar el programa (python3 ejercicio2.py).

También podemos hacerlo creando una clase:

```
#Definimos la clase Player

class Player:

#Definimos sus parámetros iniciales

   def __init__(self, rad = 20):

       self.x = resolution[0]/2

       self.y = resolution[1]/2

       self.dx = 0

       self.dy = 0

       self.radius = rad

       self.type = "player"

#Dibujamos el objeto

   def draw(self, surface):

       pygame.draw.circle(surface, red, (self.x, self.y), self.radius)

#Definimos como se actualiza el objeto

   def update(self, gameObjects):

       keys = pygame.key.get_pressed()

       if keys[pygame.K_LEFT]:

           self.dx = -2

           self.x += self.dx

       if keys[pygame.K_RIGHT]:
```

This work is licensed under CC BY NC SA 4.0

```
            self.dx = 2

            self.x += self.dx

        if keys[pygame.K_UP]:

            self.dy = -2

            self.y += self.dy

        if keys[pygame.K_DOWN]:

            self.dy = 2

            self.y += self.dy
```

Ejercicio 4. Definimos las colisiones y reducción de velocidad en base al tiempo:

```
# Compruebo si jugador 1 colisiona con pelota (esta colisión no
cumple ángulos, solo cambia todas las direcciones de movimiento)
    if ballrect.colliderect(player1rect):
        speedball[0]=-(float(speedball[0])*0.95)
        speedball[1]=-(float(speedball[1])*0.95)

# Compruebo si además "golpea la pelota con ESPACIO
        if keys[pygame.K_SPACE]:
            speedball[0] = speed1[0]
            speedball[1] = speed1[1]
    ballrect = ballrect.move(speedball)


# Reducimos la velocidad de la pelota en base al tiempo (creamos
antes la variable timer)
    timer += pygame.time.get_ticks()

    if timer >7500:
        timer = 0
        if speedball[0] > 0:
            speedball[0] = speedball[0]-0.01
        elif speedball[0] < 0:
            speedball[0] = speedball[0]+0.01
        if speedball[1] > 0:
            speedball[1] = speedball[1]-0.01
        elif speedball[1] < 0:
            speedball[1] = speedball[1]+0.01
```

En el caso de usar clases, en la clase **Player** añadimos en la función **update**:

```

#Compruebo si hay colisión entre objetos, y si se pulsa espacio
        for gameObj in gameObjects:
            if gameObj.type == "ball":
                if (gameObj.x - self.x)**2 + (gameObj.y -
self.y)**2 <= (gameObj.radius + self.radius)**2:
                    if keys[pygame.K_SPACE]:
                        gameObj.dy = self.dy
```

This work is licensed under CC BY NC SA 4.0

```
gameObj.dx = self.dx
```

Además para reducir su velocidad con el tiempo, en la clase **Ball** en la función **update** añadimos:

```
timer = pygame.time.get_ticks()

if timer > 7500:
    timer = 0
    if self.dx > 0:
        self.dx -= 0.01
    elif self.dx < 0:
        self.dx += 0.01
    if self.dy > 0:
        self.dy -= 0.01
    elif self.dy < 0:
        self.dy += 0.01
```

Fijaos que para poner las clases en funcionamiento el código deberá tener una última clase que represente la ejecución:

```
class game():

#Definimos el _init_ con las variables básicas
    def __init__(self):
        pygame.init()

        self.screen = pygame.display.set_mode(resolution)
        self.clock = pygame.time.Clock()
        self.gameObjects = []
        self.gameObjects.append(Ball())
        self.gameObjects.append(Player())

    def handleEvents(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()

#definimos el esquema de ejecución de nuestro programa
    def run(self):

        while True:
            self.handleEvents()

            for gameObj in self.gameObjects:
                gameObj.update(self.gameObjects)

            self.screen.fill(white)

            for gameObj in self.gameObjects:
                gameObj.draw(self.screen)

            self.clock.tick(60)
```

This work is licensed under CC BY NC SA 4.0

```
            pygame.display.flip()

#Ejecutamos el programa
game().run()
```

Este método está simplificado. Si queremos que la transferencia de fuerza entre objetos sea acorde al ángulo, debemos aproximar el punto de colisión por pitágoras, y además no nos serviría el rectángulo como matriz de colisión.

EJERCICIO EXTRA:

Que la transferencia de movimiento al pulsar ESPACIO entre objetos dependa de su ángulo de contacto (componente X y componente Y en base a pitágoras).

En el caso del objeto PNG con “hitbox”, podemos usar los comandos de posición: Podemos obtener la posición del rectángulo que representa la pelota y el jugador con ballrect.left (posición izquierda), ballrect.rigth (posición derecha), ballrect.top (posición superior) y ballrect.bottom (posición inferior) y con con player1rect.left (posición izquierda), player1rect.rigth (posición derecha), player1rect.top (posición superior) y player1rect.bottom (posición inferior).

Referencias:

https://code.intef.es/prop_didacticas/pygame-realizando-juegos-con-python/

This work is licensed under CC BY NC SA 4.0 Author: Alfredo Sánchez Alberca