PROGRAMACIÓN CON PYTHON

(CEFIRE CTEM)

Videojuegos con PyGame
Ejercicio obligatorio

Autora: María Paz Segura Valero (segura_marval@gva.es)

CONTENIDO

1. Introducción

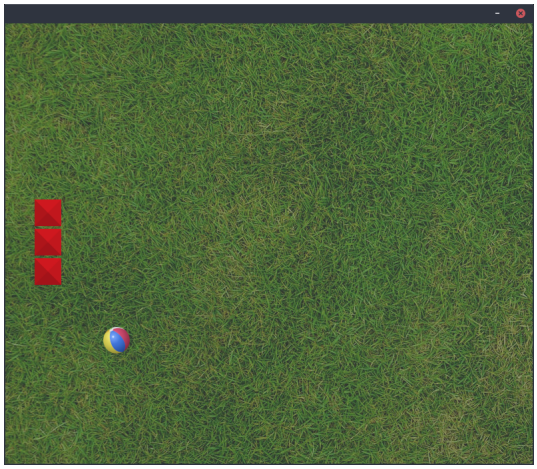
En este documento puedes encontrar el **ejercicio obligatorio** de esta unidad. Es imprescindible entregarlo en tiempo y forma para superar esta parte del curso.

Tendrás la oportunidad de realizar la entrega de varias versiones del ejercicio hasta que consigas superarlo y la profesora te indicará en cada corrección las mejoras necesarias.

En cualquier momento puedes lanzar preguntas al **foro** del curso o realizar una **entrega parcial** del ejercicio acompañada de una **lista de dudas** para que la profesora pueda orientarte en su resolución.

2. El juego de la raqueta

El objetivo de esta actividad consiste en utilizar la librería **pygame** y la estructura de programa vista en la teoría para crear "El juego de la raqueta".



Se trata de un juego en el que hay una pelota que va rebotando de forma automática por la pantalla y el jugador dispone de una raqueta (tres cuadrados rojos) que puede manejar arriba y abajo para conseguir que la pelota nunca toque el borde izquierdo de la pantalla. Es decir, la pelota nunca debe rebasar a la raqueta por su lado izquierdo ya que se acabaría el juego.

Para ello, el jugador debe mover la raqueta arriba y abajo de la pantalla con las flechas del teclado para conseguir que la pelota rebote sobre ella y salga disparada hacia el otro lado.

2.1. Elementos del programa

Para que resulte más sencilla la creación del juego, nos vamos a basar en los elementos del programa **game_snake.py** estudiado en esta sesión, así que vas a encontrar muchas cosas que te resulten familiares.

Puedes crear el programa desde cero o basarte en el esquema que tienes en el fichero **ud6_ejercicio_obligatorio (ESQUEMA).py** del aula virtual. En este caso, deberás cambiar las instrucciones *pass* que encuentres en el código.

En la siguiente tabla se indican las clases que hay que crear en el programa y cuál es su cometido:

CLASE	DESCRIPCIÓN
`Pelota`	Se encargará de actualizar la posición de la `pelota` y dibujarla en la pantalla.
`Raqueta`	Se encargará de actualizar la posición de la `raqueta`, según las instrucciones dadas por el jugador, y dibujarla en la pantalla.
`App`	Se encargará de coordinar todas las tareas del juego apoyándose en las funcionalidades de las otras dos clases y las suyas propias: • Crear los objetos de la `pelota` y la `raqueta`. • Controlar si la `pelota` ha chocado con la `raqueta` o con alguno de los bordes de la pantalla. ◦ Si ha chocado contra el borde izquierdo, acabará el juego. ◦ Si ha chocado contra la raqueta u otro borde, rebotará hacia el otro lado. • Detectar los eventos que se produzcan en el programa y responder con las acciones oportunas en cada momento. • Redibujar todos los elementos del juego: `escenario`, `raqueta` y `pelota`.

A continuación se dan algunas sugerencias para crear cada una de las clases. No obstante, si planteas una solución alternativa que funcione, el ejercicio estará superado.

2.2. La clase Pelota

En la siguiente tabla se explican los *atributos* de la clase:

ATRIBUTO	DESCRIPCIÓN
`x, y`	La combinación de estos dos atributos marcan la posición concreta de la

	pelota en la pantalla.
`pasos`	Este atributo se utiliza para calcular la posición inicial de la pelota con respecto a unas coordenadas iniciales.
`direccionX`, `direccionY`	Almacenan la dirección en la que se está moviendo la pelota. • Izquierda/oeste o derecha/este se almacenará en direccionX. Por ejemplo, con el valor 1 para el oeste y 2 para el este. • Norte/arriba y sur/abajo se almacenará en direccionY. Por ejemplo, con el valor 1 para el norte y 2 para el sur.
`avance`	Indica en cuánto avanzará la pelota cuando deba moverse en alguna dirección. Por ejemplo: 0.5, 0.75, 1, etc.
`tamanyo`	Almacena el tamaño en píxeles de la pelota. Por defecto: 40.

En la siguiente tabla se explican los *métodos* de la clase:

MÉTODO	**DESCRIPCIÓN**
`__init__(self)`	Crea una nueva pelota (sólo es necesaria una por juego) e inicializa los atributos **x** e **y** con la primera posición que debe ocupar. Podría ser la mitad de la pantalla o estar apoyada sobre la mitad de la raqueta, etc.
`update(self, dirX, dirY)`	Actualiza la posición de la pelota en pantalla teniendo en cuenta la dirección que se pasa por parámetro.
`draw(self, surface, image)`	Carga la imagen de la pelota sobre una imagen en una posición concreta de la misma.

2.3. La clase Raqueta

En la siguiente tabla se explican los *atributos* de la clase:

ATRIBUTO	**DESCRIPCIÓN**
`x`	Se trata de una lista que almacena la posición **x** de cada uno de los bloques que componen la raqueta.
`y`	Similar al anterior con la posición **y**.
`pasos`	Se utiliza para calcular la nueva posición de la raqueta.
`direccion`	Almacena la dirección hacia la que se mueve la raqueta. Sus valores pueden ser los siguientes:

	• 0: parada → valor inicial • 1: norte/arriba • 2: sur/abajo
`longitud`	Número de bloques del cuerpo de la raqueta. Por defecto, serán 3 bloques rojos los que conformen el cuerpo de la misma.

En la siguiente tabla se explican los *métodos* de la clase:

MÉTODO	**DESCRIPCIÓN**
`def __init__(self, longitud)`	Crea una raqueta de tres bloques e inicializa la posición de cada uno de ellos, almacenando los valores en las listas **x** e **y**. Tú decides qué ubicación debe ocupar la raqueta pero sería recomendable que estuviese cerca del borde izquierdo.
`def update(self, windowHeight)`	Desplaza todos los bloques del cuerpo una posición, dependiendo de la dirección en la que se deba mover la raqueta. • Si se mueve hacia *arriba*, debe mover una posición la raqueta hacia arriba y cambiar la dirección a **parado** para que no siga moviéndose eternamente. • Si se mueve hacia *abajo*, debe mover una posición la raqueta hacia abajo y cambiar la dirección a **parado**. Antes de mover la raqueta, debes controlar que existe suficiente espacio para ello. El atributo *raqueta.pasos* te puede ayudar y el parámetro *windowHeight* también.
`def moveUp(self)`	Actualiza el atributo **direccion** para indicar que la serpiente se mueve hacia el norte/arriba.
`def moveDown(self)`	Similar al anterior pero hacia el sur/abajo.
`def draw(self, surface, image)`	Carga cada uno de los bloques de la raqueta sobre una imagen en una posición concreta de la misma.

2.4. La clase App

En la siguiente tabla se explican los *atributos* de la clase:

ATRIBUTO	DESCRIPCIÓN
`windowWidth`	Ancho de la pantalla. Por defecto: 792 (= 18 x 44)
`windowHeight`	Altura de la pantalla. Por defecto: 660 (= 15 x 44)
`x, y`	Posición de la imagen de fondo. Por defecto: (0, 0)
`raqueta`	Objeto de la clase **Raqueta**
`pelota`	Objeto de la clase **Pelota**
`_running`	Indica si la aplicación debe seguir ejecutándose (True) o no (False).
`_display_surf`	Superficie (surface) que mostrará todas las imágenes.
`_image_surf`	Imagen de fondo. Por defecto: cesped.jpg
`_raqueta_surf`	Imagen de uno de los bloques que compone la raqueta. Por defecto: cuadrado.jpg
`_pelota_surf`	Imagen de la pelota. Por defecto: pelota.png

En la siguiente tabla se explican los *métodos* de la clase comparándolos con los del programa **game_snake.py**:

MÉTODO	DESCRIPCIÓN
`def __init__(self)`	Similar pero se crean los objetos *raqueta* y *pelota* en lugar de *serpiente* y *manzana*.
`def on_init(self)`	Similar excepto que cargamos las imágenes de la *raqueta* y la *pelota*.
`def on_event(self, event)`	Similar excepto que no hay que controlar la flecha izquierda ni derecha del teclado. Ojo con la llamada a las funciones.
`def isCollision(self,x1,y1, x2,y2,bsize)`	Similar pero añadiendo el control de la colisión con los bordes.
`def on_loop(self)`	Debe controlar si la pelota ha chocado con algún borde o con la raqueta. *(Ver sugerencias más abajo)* Además, actualiza la posición de la raqueta y de la pelota.
`def on_render(self)`	Similar pero hay que revisar las clases utilizadas.
`on_cleanup(self)`	Igual
`def on_execute(self)`	Similar excepto que no necesitamos utilizar la función **time.sleep()**.

2.4.1. Cómo manejar las colisiones de la pelota

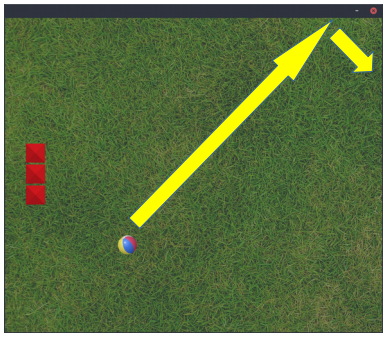
En este apartado se explican las situaciones que podemos encontrar, cuando la pelota colisiona contra la raqueta o contra un borde de la pantalla, y cómo debe reaccionar el programa en cada una de ellas para que el funcionamiento sea el esperado.

Se van a utilizar los siguientes acrónimos:

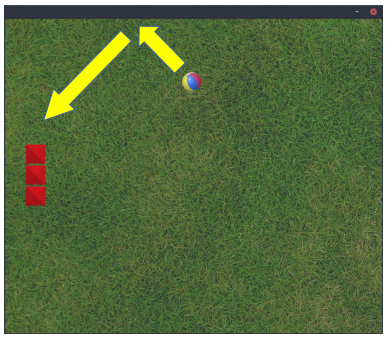
Acrónimo	Significado
NE	Noreste
NO	Noroeste
SE	Sureste
SO	Suroeste

Situación 1: la pelota choca con el borde superior

Si la pelota iba en dirección NE deberá rebotar hacia el SE.

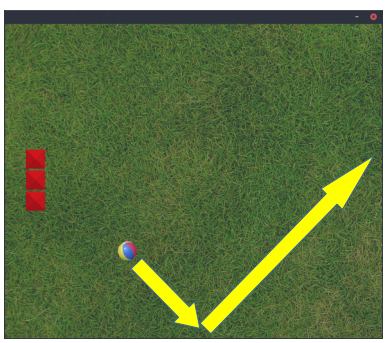


Si la pelota iba en dirección NO, deberá rebotar hacia el SO.

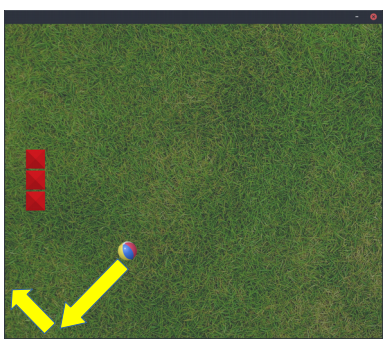


Situación 2: la pelota choca con el borde inferior

Si la pelota iba en dirección SE deberá rebotar hacia el NE.

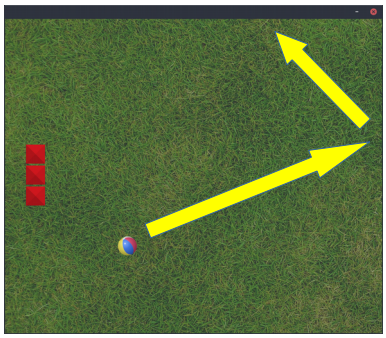


Si la pelota iba en dirección SO, deberá rebotar hacia el NO.

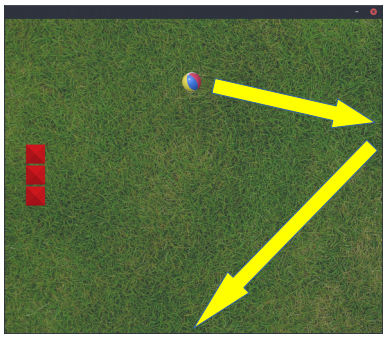


Situación 3: la pelota choca con el borde derecho

Si la pelota iba en dirección NE deberá rebotar hacia el NO.



Si la pelota iba en dirección SE, deberá rebotar hacia el SO.

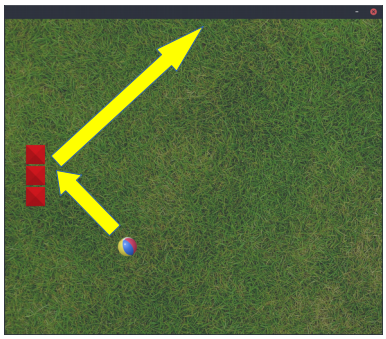


Situación 4: la pelota choca con el borde izquierdo

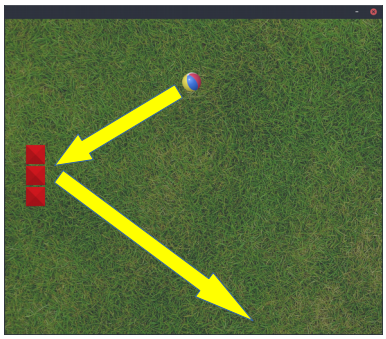
En este caso no importa la dirección porque el juego debe acabar de todas formas.

Situación 5: la pelota choca contra la raqueta

Si la pelota iba en dirección NO deberá rebotar hacia el NE.



Si la pelota iba en dirección SO, deberá rebotar hacia el SE.



Sube al aula virtual un fichero con el nombre **ud6_ejercicio_obligatorio.py**.

3. Fuentes de los recursos multimedia

	Modificación de image by Ioachim Marcu en Pixabay

	Modificación de image by PublicDomainPictures en Pixabay
	Modificación de image by OpenClipart-Vectors en Pixabay