PROGRAMACIÓN AVANZADA CON PYTHON

(CEFIRE CTEM)

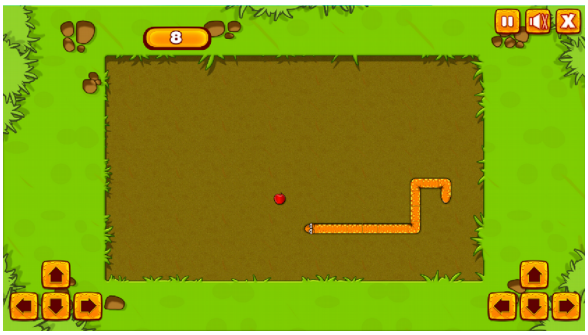
El juego de la serpiente

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

CONTENIDO

1. Juguemos a la serpiente

Hasta ahora hemos visto cómo cargar imágenes de escenario y montar la *estructura básica* de un juego pero, realmente, no hemos podido jugar a nada. Vamos a dar un paso más allá y completaremos nuestra *estructura básica* con las instrucciones necesarias para programar el "*Juego de la Serpiente*".



Estoy segura de que has debido jugar más de una vez pero, por si acaso, te recuerdo las reglas básicas.

2.1. Reglas básicas del juego

El juego consiste en conseguir que una serpiente, que se mueve por dentro de un recinto, se coma las manzanas que van a apareciendo en él sin chocar con su propio cuerpo. El jugador puede usar las flechas del teclado para cambiar la dirección de movimiento de la serpiente.

Hasta aquí parece algo sencillo pero la cosa se va complicando a medida que la serpiente va comiendo manzanas porque cambia de tamaño. Así que, cuando ha comido muchas, el movimiento evitando su propio cuerpo comienza a ser casi imposible.

Como ya habrás podido deducir, el objetivo es conseguir que la serpiente sobreviva el mayor tiempo posible.

Si quieres jugar un poco con este juego, descarga del aula virtual el siguiente fichero y descomprímelo: **game_snake.zip**

También puedes ver otra versión en la siguiente página web: https://www.paisdelosjuegos.es/juego/serpiente.html

Y ahora, manos a la obra.

2.2. La explicación de la falla

Como aparecen elementos nuevos en el juego con unas características y funcionamiento diferenciado los unos de los otros, vamos a definir las siguientes clases en el programa:

CLASE	DESCRIPCIÓN
`Apple`	Se encargará de la `manzana` que debe comerse la serpiente y dibujarla en una posición concreta de la pantalla.
`Snake`	Se encargará de actualizar la posición de `serpiente` (personaje principal del juego) 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 `serpiente` y la `manzana`. • Controlar si la `serpiente` se ha comido la `manzana`. • Controlar si la `serpiente` ha colisionado con su propio cuerpo. • 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`, `serpiente` y `manzana`.

Recordemos que las *clases* son recursos de un programa que permiten distribuir la funcionalidad del mismo atendiendo a criterios de agrupación de objetos.

En este caso, podemos diferenciar claramente el cometido de varios objetos involucrados en el juego: la `manzana`, la `serpiente` y la `aplicación`. Así que, crearemos clases diferenciadas para cada tipo de objetos y, en cada una de ellas, definiremos los atributos (variables) y métodos (funciones) que le son comunes.

En todas las clases existe un método especial llamado **__init__()** que se utiliza para crear nuevos objetos de la clase. En este método se suelen inicializar los atributos del nuevo objeto.

Dependiendo del tipo de programa, puede que necesitemos crear varios objetos distintos de la misma clase o sólo uno. Cuando se definen varios objetos distintos de la misma clase, los atributos de cada objeto pueden contener valores diferentes.

Según el cometido de cada clase se podrán crear más métodos específicos que podrán usar o no los atributos almacenados en el objeto.

En todos los métodos de una clase suele aparecer un parámetro inicial llamado *self* que hace referencia al objeto mismo que ejecuta dicho método. Es útil, por ejemplo, cuando

queremos indicar de manera explícita dentro del cuerpo de un método que nos estamos refiriendo a un atributo del objeto y no a una variable normal.

Este parámetro no hace falta especificarlo explícitamente en la llamada (ejecución) al método ya que Python lo incorpora automáticamente.

Vamos a ir conociendo cada una de estas clases poco a poco.

2.3. La clase Apple

```
class Apple:
    x = 0
    y = 0
    pasos = 44

    def __init__(self,param_x,param_y):
        self.x = param_x * self.pasos
        self.y = param_y * self.pasos

    def draw(self, surface, image):
        surface.blit(image,(self.x, self.y))
```

Esta clase almacena la posición de la `manzana` (atributos **x** e **y**) y la va actualizando conforme se va modificando durante el juego. El atributo **pasos** sirve para calcular su posición inicial.

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 manzana en la pantalla.
`pasos`	Este atributo se utiliza para calcular la posición inicial de la manzana con respecto a unas coordenadas iniciales.

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

MÉTODO	DESCRIPCIÓN
`__init__(self, param_x, param_y)`	Crea una nueva manzana (sólo es necesaria una por juego) e inicializa los atributos **x** e **y**.
`draw(self, surface, image)`	Carga la imagen de la manzana sobre una imagen en una posición concreta de la misma.

2.4. La clase Snake

```
class Snake:
    x = [0]     # posición x de bloques de la serpiente
    y = [0]     # posición y de bloques de la serpiente
    pasos = 44
    direccion = 0   # 0:este 1:oeste 2:norte 3:sur
    longitud = 3    # inicialmente el cuerpo son tres bloques


    def __init__(self, longitud):
       self.longitud = longitud

       # inicializa la posición de los bloques de la serpiente
       for i in range(0,2000):
           self.x.append(-100)
           self.y.append(-100)

       # posicion inicial de la serpiente sin colisión
       self.x[1] = 1 * self.pasos
       self.x[2] = 2 * self.pasos


    def update(self):
        # actualiza la posición de los bloques de la serpiente
        for i in range(self.longitud-1,0,-1):
            self.x[i] = self.x[i-1]
            self.y[i] = self.y[i-1]

        # actualiza la posición de la cabeza de la serpiente
        if self.direccion == 0: # direccion este/derecha
            self.x[0] = self.x[0] + self.pasos
        if self.direccion == 1: # direccion oeste/izquierda
            self.x[0] = self.x[0] - self.pasos
        if self.direccion == 2: # direccion norte/arriba
            self.y[0] = self.y[0] - self.pasos
        if self.direccion == 3: # direccion sur/abajo
            self.y[0] = self.y[0] + self.pasos


    def moveRight(self):
        self.direccion = 0

    def moveLeft(self):
        self.direccion = 1

    def moveUp(self):
        self.direccion = 2

    def moveDown(self):
        self.direccion = 3

    def draw(self, surface, image):
```

```
        # dibuja cada bloque de la serpiente
        for i in range(0,self.longitud):
            surface.blit(image,(self.x[i],self.y[i]))
```

Esta clase almacena la posición de cada bloque del cuerpo de la serpiente y la va desplazando de forma automática si el usuario no la mueve con los cursores.

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 cabeza y el cuerpo de la serpiente.
`y`	Similar al anterior con la posición **y**.
`pasos`	Se utiliza para calcular la nueva posición automática de la serpiente.
`direccion`[1]	Almacena la dirección en la que mira la cabeza de la serpiente. Sus valores pueden ser los siguientes: • 0: este/derecha • 1: oeste/izquierda • 2: norte/arriba • 3: sur/abajo
`longitud`	Número de bloques del cuerpo de la serpiente. Inicialmente la serpiente tiene 3 bloques pero conforme va creciendo, se irá actualizando este atributo.

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

MÉTODO	**DESCRIPCIÓN**
`def __init__(self, longitud)`	Crea una serpiente de tres bloques e inicializa la posición de cada uno de ellos, almacenando los valores en las listas **x** e **y**.
`def update(self)`	Desplaza todos los bloques del cuerpo una posición y, dependiendo de la dirección en la que se mueve la serpiente, posiciona la cabeza de la misma.
`def moveRight(self)`	Actualiza el atributo **direccion** para indicar que la serpiente se mueve hacia el este/derecha.

1 Es una buena práctica crear los nombres de las variables, funciones , etc. sin acentos porque algunos lenguajes de programación pueden tener problemas para reconocerlos.

`def moveLeft(self)`	Similar al anterior pero hacia el oeste/izquierda.
`def moveUp(self)`	Similar al anterior pero 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 serpiente sobre una imagen en una posición concreta de la misma.

2.5. La clase App

```
class App:
    windowWidth = 792
    windowHeight = 660
    x = 0
    y = 0
    snake = 0 #objeto serpiente
    apple = 0 #objeto manzana

    def __init__(self):
        self._running = True
        self._display_surf = None #Surface que mostrará las imágenes
        self._image_surf = None    #imagen de fondo, aún sin asignar

        self._snake_surf = None    #imagen de la serpiente, aún sin asignar
        self._apple_surf = None    #imagen de la manzana, aún sin asignar

        self.snake = Snake(3)      #creamos un objeto de la clase Snake
        self.apple = Apple(5,5)   # creamos un objeto de la clase Apple

    def on_init(self):
        pygame.init()
        self._running = True
        self._display_surf = pygame.display.set_mode(
           (self.windowWidth, self.windowHeight), pygame.HWSURFACE)

        #Cargamos las imágenes en pantalla
        self._image_surf =
                   pygame.image.load("cesped.jpg").convert()
        self._snake_surf =
                   pygame.image.load("cuadrado.jpg").convert()
        self._apple_surf =
                   pygame.image.load("manzana.png").convert_alpha()

    def on_event(self, event):
        if event.type == QUIT:
            self._running = False

        # comprueba si se ha pulsado una tecla
        keys = pygame.key.get_pressed()
        if (keys[K_RIGHT]):
```

```
            self.snake.moveRight()

        if (keys[K_LEFT]):
            self.snake.moveLeft()

        if (keys[K_UP]):
            self.snake.moveUp()

        if (keys[K_DOWN]):
            self.snake.moveDown()

        if (keys[K_ESCAPE]):
            self._running = False

    def isCollision(self,x1,y1,x2,y2,bsize):
        if x1 >= x2 and x1 <= x2 + bsize:
            if y1 >= y2 and y1 <= y2 + bsize:
                return True
        return False

    def on_loop(self):
        self.snake.update() #actualiza la posición de la serpiente

        # comprueba si la serpiente se ha comido una manzana
        for i in range(0,self.snake.longitud):
            if self.isCollision(self.apple.x,self.apple.y,
                self.snake.x[i],self.snake.y[i],self.snake.pasos):
                self.apple.x = randint(2,9) * self.apple.pasos
                self.apple.y = randint(2,9) * self.apple.pasos
                self.snake.longitud = self.snake.longitud + 1

        # comprueba si la serpiente ha chocado consigo misma
        for i in range(2,self.snake.longitud):
            if self.isCollision(self.snake.x[0],self.snake.y[0],
                                self.snake.x[i], self.snake.y[i],40):
                print("You lose! Collision: ")
                print("x[0] (" + str(self.snake.x[0]) + "," +
                                        str(self.snake.y[0]) + ")")
                print("x[" + str(i) + "] (" + str(self.snake.x[i])
                                + "," + str(self.snake.y[i]) + ")")
                exit(0)


    def on_render(self):
        self._display_surf.blit(self._image_surf,(self.x,self.y))
        self.snake.draw(self._display_surf, self._snake_surf)
        self.apple.draw(self._display_surf, self._apple_surf)
        pygame.display.flip()

    # cierra los recursos del programa
    def on_cleanup(self):
        pygame.quit()
```

```
    def on_execute(self):
        self.on_init()
        while( self._running ):
            for event in pygame.event.get():
                self.on_event(event)
            self.on_loop()
            self.on_render()

            time.sleep (150.0 / 1000.0);
        self.on_cleanup()
```

La clase `App` crea los objetos de las clases presentadas anteriormente y gestiona su funcionamiento.

A continuación se explican los atributos nuevos y los cambios realizados en los métodos originales de esta clase (líneas remarcadas en azul).

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

ATRIBUTO	DESCRIPCIÓN
`snake`	Objeto de la clase Snake
`apple`	Objeto de la clase Apple
`_snake_surf`	Imagen de uno de los bloques que componen el cuerpo de la serpiente
`_apple_surf`	Imagen de la manzana

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

MÉTODO	DESCRIPCIÓN
`def __init__(self)`	Inicializamos las variables que contendrán las imágenes de la serpiente y la manzana. Crea una serpiente de longitud 3 y una manzana.
`def on_init(self)`	Carga las imágenes de la serpiente y la manzana. Para cargar ficheros .png que pueden tener el fondo transparente, se usa la función `convert_alpha()` en lugar de `convert()`.
`def on_event(self, event)`	Además de controlar el cierre de la ventana con el aspa de arriba a la derecha, controla los cursores del teclado que pretenden mover la serpiente y la tecla ESC que también cierra el juego.
`def isCollision(self,x1,y1, x2,y2,bsize)`	Comprueba si dos elementos ocupan la misma posición

	en la pantalla. Por ejemplo: manzana y un bloque de la serpiente.
`def on_loop(self)`	Actualiza la posición de la serpiente en la pantalla. Comprueba si la serpiente se ha comido una manzana. Comprueba si la serpiente ha colisionado consigo misma.
`def on_render(self)`	Además de mostrar en pantalla la imagen de fondo, también muestra la serpiente y la manzana.
`def on_execute(self)`	Entre una iteración (pasada) del bucle y otra, espera unos milisegundos para que no vaya tan acelerado el movimiento de la serpiente.

Puedes ver el ejemplo completo en el siguiente fichero: ***game_snake.zip***

3. Fuentes de información

- Librería PyGame: https://github.com/pygame/pygame

4. Créditos de imágenes

	Image by janjf93 en Pixabay
	Modificación de image by Ioachim Marcu en Pixabay
	Modificación de image by PublicDomainPictures en Pixabay