**Informe del Juego: Retro Super Snake**

**Índice**

1. Introducción

- Descripción general del proyecto "Retro Super Snake".

- Uso de Pygame para el desarrollo de videojuegos en Python.

- Características adicionales del juego.

2. Objetivo del juego

3. Configuración Inicial

- Definición de dimensiones de pantalla, tamaño de celda, y colores.

- Inicialización de Pygame y configuración de la pantalla.

- Función para cargar y escalar imágenes.

4. Clases Principales

a. GameObject

- Clase base abstracta para todos los objetos del juego.

b. Food

- Representa la comida que la serpiente debe comer para crecer.

c. Snake

- Representa la serpiente del juego con métodos para moverse, crecer, y cambiar de dirección.

d. Obstacle

- Representa obstáculos estáticos en el juego que la serpiente debe evitar.

e. SpecialItem

- Representa un ítem especial que otorga un impulso temporal en velocidad a la serpiente.

5. Conceptos de Programación Orientada a Objetos (POO)

- Clases y Objetos

- Métodos y Atributos

- Herencia

- Polimorfismo

- Encapsulamiento

- Abstracción

6. Mecánicas del Juego

- Movimiento y Crecimiento de la Serpiente

- Colisiones

- Ítems Especiales

7. Pantallas del Juego

a. Pantalla de Inicio

b. Pantalla de Juego

c. Pantalla de Game Over

8. Control de Juego

- Movimiento y pausa del juego.

9. Bucle Principal

- Manejo de eventos, actualizaciones de estado y dibujado en pantalla.

- Uso de un reloj para controlar la velocidad de actualización.

10. Conclusión

- Resumen del proyecto y su implementación.

- Destacar las características y capacidades del juego.

**Introducción**

El proyecto "Retro Super Snake" es una implementación moderna del clásico juego de la serpiente, utilizando Pygame, una biblioteca para desarrollo de videojuegos en Python. El juego incluye características adicionales como alimentos especiales, obstáculos, y un sistema de puntuación.

**Objetivo del Juego**

En Retro Super Snake, tu objetivo es guiar a una serpiente hambrienta a través de un campo lleno de deliciosos manjares mientras evitas chocar con las paredes, su propio cuerpo o cualquier obstáculo que se interponga en su camino. Controla la dirección de la serpiente y asegúrate de que pueda comer tanto como sea posible. ¡Mientras más coma, más grande y más puntaje obtendrás! Este desafío simple pero adictivo te mantendrá entretenido mientras tratas de superar tus propias marcas y llegar más lejos en cada partida.

**Configuración Inicial**

El juego se inicializa con una serie de configuraciones y constantes:

- Dimensiones de pantalla: 800x600 píxeles.

- Tamaño de celda: 20 píxeles.

- Colores: Definidos en RGB (blanco, negro, rojo, verde, azul, púrpura, café oscuro y claro).

- Velocidad de actualización: 10 FPS (frames por segundo).

Código:

# Inicializar Pygame

pygame.init()

# Definición de constantes

WIDTH, HEIGHT = 800, 600

CELL\_SIZE = 20

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 0, 255)

PURPLE = (255, 0, 255)

CAFEOS = (75, 54, 33)

CAFE = (139, 69, 19)

FPS = 10

# Configuración de la pantalla

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption('Retro Super Snake')

# Fuente para el puntaje

font = pygame.font.Font(None, 36)

**Carga de Imágenes**

Las imágenes para los objetos del juego se cargan y escalan mediante la función `load\_and\_scale\_image`, mejorando la presentación visual del juego.

Código:

# Función para cargar y escalar imágenes

def load\_and\_scale\_image(filename, size):

image = pygame.image.load(filename)

return pygame.transform.scale(image, size)

**Clases Principales**

El juego se compone de varias clases que representan los diferentes elementos y mecánicas del juego:

**a**. `**GameObject**`

Clase base abstracta para todos los objetos del juego. Define los métodos básicos y atributos que las clases derivadas deben implementar.

Código:

class GameObject:

def \_\_init\_\_(self, position):

self.position = position

def draw(self, screen):

pass

**b**. `**Food**`

Hereda de `GameObject`. Representa la comida que la serpiente debe comer para crecer. La imagen se carga y se escala a 25x25 píxeles.

Código:

class Food(GameObject):

def \_\_init\_\_(self, position):

super().\_\_init\_\_(position)

self.image = load\_and\_scale\_image('Red-Apple-icon.png', (25, 25))

def draw(self, screen):

screen.blit(self.image, self.position)

**c**. `**Snake**`

Hereda de `GameObject`. Representa la serpiente del juego con atributos para su dirección, cuerpo, crecimiento, y estado de "boost". Incluye métodos para moverse, crecer, cambiar de dirección y dibujar la serpiente en pantalla.

Código:

class Snake(GameObject):

def \_\_init\_\_(self, position):

super().\_\_init\_\_(position)

self.directions = {

'UP': pygame.Vector2(0, -1),

'DOWN': pygame.Vector2(0, 1),

'LEFT': pygame.Vector2(-1, 0),

'RIGHT': pygame.Vector2(1, 0)

}

self.body = [position, position - pygame.Vector2(CELL\_SIZE, 0)]

self.direction = pygame.Vector2(1, 0)

self.growing = False

self.speed\_boost = False

self.eye\_offset = [(7, 7), (CELL\_SIZE - 7 \* 2, 7)]

self.head\_image = load\_and\_scale\_image("cabeza.png", (23, 24))

self.body\_image = load\_and\_scale\_image("e9ojfc2s.png", (22, 23))

self.tail\_image = load\_and\_scale\_image("Sin título.png", (22, 24))

def move(self):

head\_position = self.body[0] + self.direction \* CELL\_SIZE

if self.growing:

self.body = [head\_position] + self.body

self.growing = False

else:

self.body = [head\_position] + self.body[:-1]

def grow(self):

self.growing = True

def draw(self, screen):

for segment in self.body[1:-1]:

screen.blit(self.body\_image, segment)

head\_pos = self.body[0]

rotated\_head\_image = pygame.transform.rotate(self.head\_image, self.directions\_to\_angle())

screen.blit(rotated\_head\_image, head\_pos)

tail\_pos = self.body[-1]

opposite\_direction = -self.direction

rotated\_tail\_image = pygame.transform.rotate(self.tail\_image, self.directions\_to\_angle(opposite\_direction))

screen.blit(rotated\_tail\_image, tail\_pos)

eye\_offset = [(15, 8), (10, 8)]

for offset in eye\_offset:

eye\_pos = (head\_pos[0] + offset[0], head\_pos[1] + offset[1])

pygame.draw.circle(screen, WHITE, eye\_pos, 2)

def directions\_to\_angle(self, direction=None):

if direction is None:

direction = self.direction

if direction == pygame.Vector2(1, 0):

return 270

elif direction == pygame.Vector2(-1, 0):

return 90

elif direction == pygame.Vector2(0, -1):

return 0

elif direction == pygame.Vector2(0, 1):

return 180

def change\_direction(self, direction):

if direction in self.directions:

new\_direction = self.directions[direction]

opposite\_direction = -self.direction

if new\_direction != opposite\_direction:

self.direction = new\_direction

**d**. `**Obstacle**`

Hereda de `GameObject`. Representa obstáculos estáticos en el juego que la serpiente debe evitar. La imagen del obstáculo es un contenedor de madera.

Código:

class Obstacle(GameObject):

def \_\_init\_\_(self, position):

super().\_\_init\_\_(position)

self.image = load\_and\_scale\_image('depositphotos\_54217117-stock-photo-wood-crate-generated-hires-texture.jpg', (23, 24))

def draw(self, screen):

screen.blit(self.image, self.position)

**e**. `**SpecialItem**`

Hereda de `GameObject`. Representa un ítem especial que, al ser consumido por la serpiente, le otorga un impulso temporal en velocidad. La imagen del ítem es una manzana amarilla.

Código:

class SpecialItem(GameObject):

def \_\_init\_\_(self, position):

super().\_\_init\_\_(position)

self.timer = 200

self.image = load\_and\_scale\_image('Yellow-Apple-icon.png', (25, 25))

def draw(self, screen):

screen.blit(self.image, self.position)

**Conceptos de Programación Orientada a Objetos (POO)**

**a**. Clases y Objetos: Se han definido varias clases como `GameObject`, `Food`, `Snake`, `Obstacle`, y `SpecialItem`, las cuales representan los elementos del juego. Estas clases se utilizan para crear objetos con atributos y comportamientos específicos.

**b.** Métodos y Atributos: Las clases contienen tanto métodos (funciones) como atributos (variables) que definen el comportamiento y el estado de los objetos del juego, respectivamente. Estos métodos y atributos están organizados de manera lógica dentro de cada clase.

**c. Herencia**: Utilizas la herencia en las clases Food, Snake, Obstacle, y SpecialItem, las cuales heredan de la clase base GameObject. Esto te permite reutilizar código y definir comportamientos comunes en la clase base.

**d.** Polimorfismo: Se ha implementado el método `draw()` en las clases hijas (`Food`, `Snake`, `Obstacle`, y `SpecialItem`) de manera que cada una dibuja su respectivo elemento de manera diferente, pero todas utilizando el mismo nombre de método. Este es un ejemplo de polimorfismo, donde objetos de diferentes clases pueden ser tratados de manera uniforme.

**e.** Encapsulamiento: En la implementación, los atributos y métodos de las clases están encapsulados dentro de las clases respectivas, lo que significa que el estado interno de cada objeto se mantiene oculto y solo puede ser modificado mediante métodos específicos.

**f. Abstracción**: La clase GameObject sirve como una abstracción para todos los elementos del juego, definiendo un conjunto común de métodos y atributos que son compartidos por las clases hijas. Esto ayuda a simplificar el diseño y a hacerlo más modular.

**g.** GameObject: ` sirve como una abstracción para todos los elementos del juego, definiendo un conjunto común de métodos y atributos que son compartidos por las clases hijas. Esto ayuda a simplificar el diseño y a hacerlo más modular. - Clases y Objetos: `GameObject` sirve como una abstracción para todos los elementos del juego, definiendo un conjunto común de métodos y atributos que son compartidos por las clases hijas. Esto ayuda a simplificar el diseño y hacerlo más modular. - Polimorfismo: Aunque `GameObject` no implementa directamente un comportamiento específico, proporciona un punto de partida para el polimorfismo al definir métodos como `draw()` que se espera que las clases hijas implementen de manera única. - Encapsulamiento: Los atributos y métodos de `GameObject` están encapsulados dentro de la clase, asegurando que el estado interno y el comportamiento de los objetos derivados se mantengan controlados y ocultos. - Métodos y Atributos: Aunque `GameObject` en sí mismo no tiene implementaciones concretas de métodos, define atributos como `position` que son compartidos por todas las clases hijas y esperados por ciertos métodos como `draw()`.

**h.** Food: - Clases y Objetos: La clase `Food` representa la comida que la serpiente debe comer para crecer. Utiliza la herencia para extender las funcionalidades de la clase base `GameObject`. - Polimorfismo: Implementa el método `draw()` de manera única para dibujar la comida en la pantalla. - Encapsulamiento: Todos los atributos y métodos de la clase están encapsulados, asegurando que el estado interno de la comida se mantenga oculto. - Métodos y Atributos: La clase `Food` contiene atributos como `position` e `image`, así como el método `draw()` para su representación visual en el juego.

**i.** Snake: - Herencia: La clase `Snake` hereda de `GameObject`, aprovechando la herencia para reutilizar código y definir comportamientos comunes. - Polimorfismo: Implementa el método `draw()` para dibujar la serpiente en pantalla de manera única, permitiendo que objetos de diferentes clases sean tratados uniformemente. - Encapsulamiento: Los atributos y métodos de la serpiente están encapsulados, asegurando un acceso controlado a su estado interno. - Abstracción: La clase `Snake` abstrae los conceptos de dirección, cuerpo, crecimiento, y estado de "boost", proporcionando una interfaz clara para interactuar con la serpiente. - Métodos y Atributos: La serpiente tiene atributos como `body`, `direction`, y `growing`, junto con métodos para moverse, crecer y cambiar de dirección.

**j.** Obstacle: - Herencia: Al igual que las clases anteriores, `Obstacle` hereda de `GameObject`, lo que le permite compartir funcionalidades comunes con otros elementos del juego. - Encapsulamiento: Todos los atributos y métodos de la clase están encapsulados, asegurando un acceso controlado al estado interno del obstáculo. - Abstracción: La clase `Obstacle` proporciona una abstracción para representar obstáculos estáticos en el juego, simplificando su manejo y colocación en el entorno de juego. - Métodos y Atributos: La clase `Obstacle` tiene atributos como `position` e `image`, junto con el método `draw()` para su representación visual.

**k.** SpecialItem:- Herencia: `SpecialItem` hereda de `GameObject`, lo que le permite compartir funcionalidades comunes con otros elementos del juego. - Polimorfismo: Implementa el método `draw()` de manera única para representar visualmente el elemento especial en la pantalla. - Encapsulamiento: Todos los atributos y métodos de la clase están encapsulados, garantizando un acceso controlado a su estado interno. - Abstracción: `SpecialItem` abstrae la noción de un elemento especial con un temporizador activo en el juego, proporcionando una interfaz clara para su manipulación. - Métodos y Atributos: La clase `SpecialItem` tiene atributos como `position`, `timer` e `image`, junto con el método `draw()` para su representación visual y el método `place\_special\_item()` para ubicar el elemento especial en el juego.

**Mecánicas del Juego**

**a.** **Movimiento y Crecimiento de la Serpiente**

La serpiente se mueve en la dirección indicada por el usuario (W, A, S, D). Cada vez que la serpiente come una comida, crece y aumenta su puntuación.

**b. Colisiones**

El juego finaliza si la serpiente choca con los bordes de la pantalla, con su propio cuerpo, o con un obstáculo. La detección de colisiones se maneja en el método `check\_collision`.

Código:

def check\_collision(self):

if self.snake.body[0] == self.food.position:

self.snake.grow()

self.food = self.place\_food()

self.score += 10

if random.randint(0, 100) < 20 and self.special\_item is None:

self.special\_item = self.place\_special\_item()

if self.special\_item and self.snake.body[0] == self.special\_item.position:

self.special\_item

self.special\_item = None

self.snake.speed\_boost = True

self.special\_item\_timer = 200

if self.snake.body[0].x < 0 or self.snake.body[0].x >= WIDTH or self.snake.body[0].y < 0 or self.snake.body[0].y >= HEIGHT:

return True

if len(self.snake.body) != len(set(tuple(segment) for segment in self.snake.body)):

return True

for obstacle in self.obstacles:

if self.snake.body[0] == obstacle.position:

return True

return False

**c. Ítems Especiales**

Al azar, aparece un ítem especial que, si es consumido, otorga un impulso temporal a la velocidad de la serpiente.

**Pantallas del Juego**

1. **Pantalla de Inicio**

Muestra un mensaje para comenzar el juego. El juego espera a que el usuario presione cualquier tecla para empezar.

Código:

def show\_start\_screen():

background\_img = pygame.image.load("snake1.png")

screen.blit(background\_img, (-280, -80))

title = font.render('Presione cualquier tecla para comenzar', True, WHITE)

screen.blit(title, [WIDTH // 4, HEIGHT // 2])

pygame.display.flip()

wait\_for\_key()

1. **Pantalla de Juego**

Dibuja la serpiente, comida, obstáculos, y, si está presente, el ítem especial. Muestra la puntuación actual en la parte superior izquierda.

Código:

def draw(self, screen):

background\_image = pygame.image.load("R.jpeg")

screen.blit(background\_image, (0, 0))

self.snake.draw(screen)

self.food.draw(screen)

for obstacle in self.obstacles:

obstacle.draw(screen)

if self.special\_item:

self.special\_item.draw(screen)

self.draw\_score(screen)

pygame.display.flip()

1. **Pantalla de Game Over**

Cuando el juego termina, se muestra una pantalla con el mensaje "Game Over", la puntuación obtenida y la instrucción para reiniciar el juego.

Código:

def show\_game\_over\_screen(score):

screen.fill(BLACK)

game\_over\_text = font.render('Game Over', True, RED)

score\_text = font.render(f'Tu puntaje: {score}', True, WHITE)

restart\_text = font.render('Presiona cualquier tecla para reiniciar el juego', True, WHITE)

screen.blit(game\_over\_text, [WIDTH // 3, HEIGHT // 3])

screen.blit(score\_text, [WIDTH // 3, HEIGHT // 2])

screen.blit(restart\_text, [WIDTH // 4, HEIGHT // 1.5])

pygame.display.flip()

wait\_for\_key()

**Control de Juego**

- Movimiento: La serpiente se controla con las teclas W (arriba), A (izquierda), S (abajo) y D (derecha).

- Pausa: Presionando la barra espaciadora se pausa y se reanuda el juego.

**Bucle Principal**

El bucle principal del juego maneja la lógica de eventos, actualizaciones de estado del juego, y dibujado en pantalla. Utiliza un reloj (`clock`) para controlar la velocidad de actualización y asegurar un juego fluido.

Código:

def main():

clock = pygame.time.Clock()

game = Game()

show\_start\_screen()

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

game.paused = not game.paused

elif not game.paused:

if event.key == pygame.K\_w:

game.snake.change\_direction('UP')

elif event.key == pygame.K\_s:

game.snake.change\_direction('DOWN')

elif event.key == pygame.K\_a:

game.snake.change\_direction('LEFT')

elif event.key == pygame.K\_d:

game.snake.change\_direction('RIGHT')

if not game.paused:

if game.update():

show\_game\_over\_screen(game.score)

game = Game()

show\_start\_screen()

game.draw(screen)

clock.tick(FPS + game.score // 50)

else:

paused\_text = font.render('Juego Pausado', True, WHITE)

screen.blit(paused\_text, [WIDTH // 3, HEIGHT // 3])

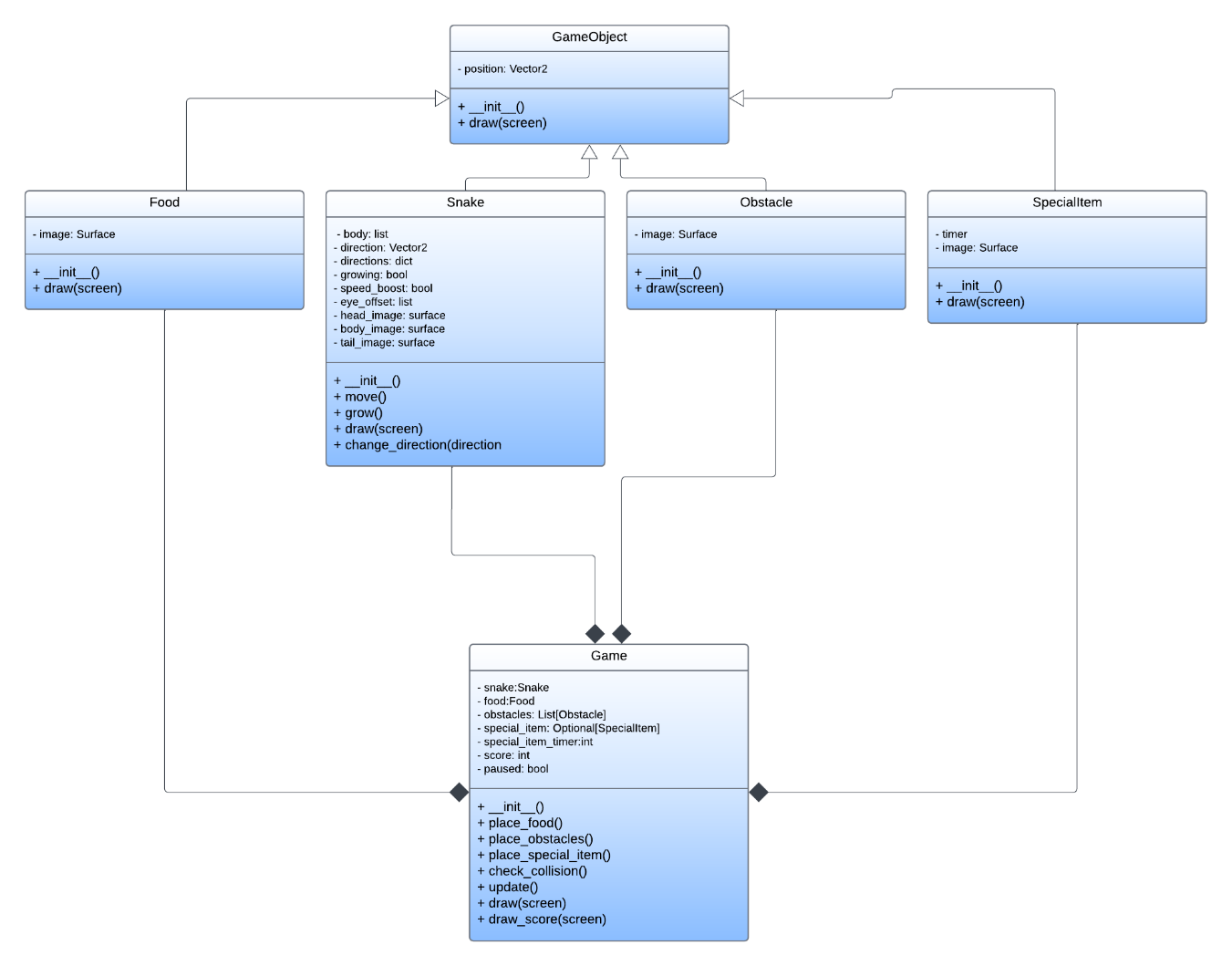
pygame.display.flip()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Conclusión**

El proyecto "Retro Super Snake" es un juego bien estructurado que emplea principios de programación orientada a objetos como la herencia y el polimorfismo. La implementación incluye características visuales atractivas y mecánicas adicionales que enriquecen la experiencia del clásico juego de la serpiente. Este juego es una excelente demostración de las capacidades de Pygame y del uso de Python en el desarrollo de videojuegos.



Video:

Nombre: Juego NombreVideojuego con POO en Python - Univalle Tuluá”, ejemplo: “Juego Snake con POO en Python - Univalle Tuluá

Caratula o miniatura del video:

Enlace de visualización del código en Github: