Martinez Moreo Juan Pablo

Lopez Franco Michelle

Tolerancia a fallas

Checkpoint

Este proyecto implementa un juego interactivo de laberinto utilizando la biblioteca Pygame en Python. El juego permite al usuario controlar un jugador que debe navegar a través de un laberinto, evitando colisiones con las paredes y alcanzando checkpoints.

Desarrollo

Configuración del entorno Para la ejecución del programa, se requiere la instalación de Pygame, una biblioteca que permite desarrollar videojuegos en Python.

```
import pygame
import sys

pygame.init()

ANCHO, ALTO = 600, 600
pantalla = pygame.display.set_mode((ANCHO, ALTO))
pygame.display.set_caption("Laberinto con Checkpoints y Colisiones")

BLANCO = (255, 255, 255)
NEGRO = (0, 0, 0)
AZUL = (0, 0, 255)
VERDE = (0, 255, 0)
ROJO = (255, 0, 0)


jugador_tam = 20
jugador = pygame.Rect(40, 40, jugador_tam, jugador_tam)


checkpoint = pygame.Rect(250, 300, 30, 30)
punto_final = pygame.Rect(550, 550, 30, 30)
```

Elementos del programa

Se utiliza un rectángulo de 20 pixeles para representar el jugador.

El punto de control se representa de 30 pixeles para que el jugador los reconozca se pinta de un color verde.

El punto final esta representado por un rectángulo rojo que indica el final del laberinto.

```
paredes = [
    pygame.Rect(0, 0, 600, 20), pygame.Rect(0, 0, 20, 600), pygame.Rect(580, 0, 20, 600), pygame.Rect(0, 580, 600, 20),
    pygame.Rect(60, 60, 120, 20), pygame.Rect(180, 60, 20, 100), pygame.Rect(100, 140, 100, 20),
    pygame.Rect(100, 140, 20, 100), pygame.Rect(100, 240, 120, 20), pygame.Rect(220, 140, 20, 100),
    pygame.Rect(220, 140, 100, 20), pygame.Rect(300, 60, 20, 100), pygame.Rect(320, 140, 120, 20),
    pygame.Rect(420, 140, 20, 100), pygame.Rect(340, 240, 100, 20), pygame.Rect(340, 240, 20, 100),
    pygame.Rect(340, 340, 120, 20), pygame.Rect(460, 240, 20, 100), pygame.Rect(460, 240, 100, 20),
    pygame.Rect(540, 340, 20, 100), pygame.Rect(440, 440, 120, 20), pygame.Rect(440, 440, 20, 100),
    pygame.Rect(340, 540, 100, 20), pygame.Rect(340, 540, 20, 40), pygame.Rect(220, 540, 100, 20),
    pygame.Rect(220, 540, 20, 40), pygame.Rect(100, 540, 100, 20), pygame.Rect(100, 440, 20, 100),
    pygame.Rect(100, 440, 100, 20), pygame.Rect(200, 340, 20, 100), pygame.Rect(200, 340, 100, 20),
]
```

Esta parte del código representa la estructura del juego que en este caso son las paredes del laberinto.

```
    velocidad = 5
    checkpoint_alcanzado = False
    posicion_inicial = (40, 40)
    posicion_checkpoint = (250, 300)
```

Aquí se definen las variables inciales que son elementos principales del programa.

```
corriendo = True
while corriendo:
    pantalla.fill(BLANCO)
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            corriendo = False
```

Este bucle while permite que el juego siga ejecutándose hasta que el jugador decida salir.

```
            break

    if colision:
        if checkpoint_alcanzado:
            jugador.topleft = posicion_checkpoint
        else:
            jugador.topleft = posicion_inicial
    else:
        jugador = nuevo_jugador
```

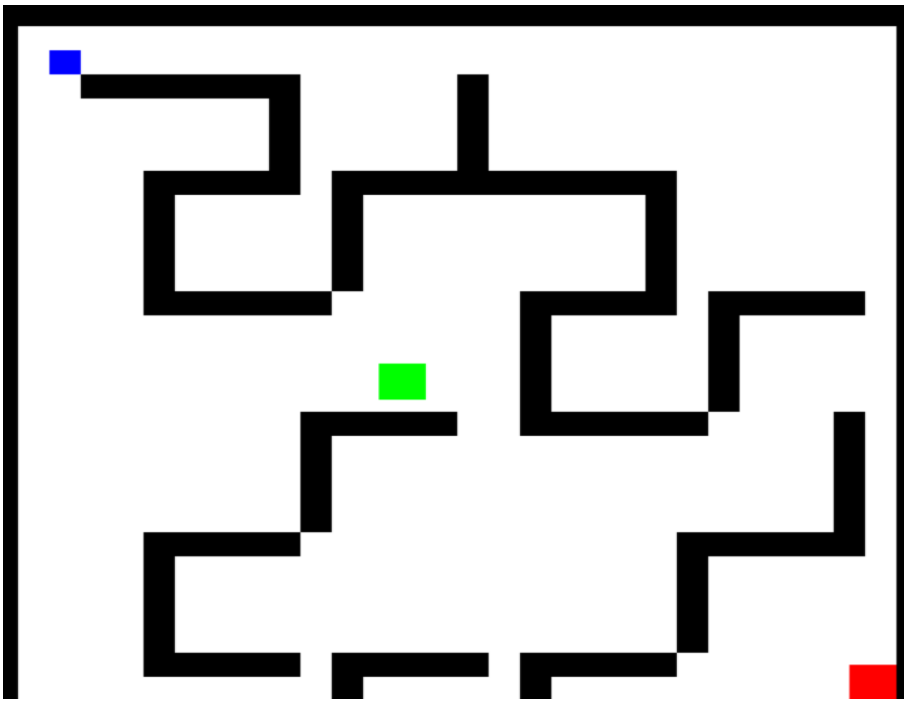
Si se detecta alguna colisión regresa al jugador a su posición inicial.

```python
if jugador.colliderect(checkpoint):
    checkpoint_alcanzado = True

if jugador.colliderect(punto_final):
    print("¡Has completado el laberinto!")
    corriendo = False
```

Por ultimo si el jugador a alanzo el checkpoint lo reajnudara directo desde este punto en caso de que no será retornado un false y volverá al punto de partida.

Al finalizar se agrega un punto final que marca el final del juego con un rectángulo en rojo



Al ejecutar el código se mira de esta forma, señalizando que el checkpoint es el cuadro verde, mientras que el rojo es el final del nivel y el azul es la representación del jugador

Conclusión

El sistema de checkpoint es muy útil para gestionar estados y manejar fallos en cualquier tipo de aplicación o proceso, no solo en videojuegos. Básicamente, un checkpoint funciona como un punto en el que el sistema guarda su estado en un momento específico. Si por alguna razón ocurre un error o se interrumpe el proceso, el sistema puede regresar rápidamente a ese punto guardado, lo que hace que la restauración sea más eficiente y que no se pierda todo el progreso realizado hasta ese momento.