**Descripción del Juego**

Power Racing, es un juego de carros, desarrollado en Python utilizando la biblioteca Pygame, el cual se basa en esquivar los carros que se encuentre el jugador al largo del camino.

**Requisitos del Sistema**

* Python 3
* Pygame (instalable mediante pip install pygame)

**Instrucciones de Instalación**

* Clona o descarga el repositorio en tu máquina local.
* Asegúrate de tener Python instalado en tu sistema.
* Instala la biblioteca Pygame ejecutando el siguiente comando:

**pip install pygame**

**Cómo Jugar?**

* Ejecuta el archivo principal del juego: Main.py.
* python Main.py
* Utiliza las teclas de ‘a’ para ir a la izquierda y ‘d’ para ir a la derecha para controlar el carro y esquivar obstáculos.

**Carpetas**

* **Buttons:** Carpeta donde se guardan las imágenes de los botones de los menús del juego
* **Cars:** Carpeta donde se guardan los sprites del juego e imágenes pequeñas de visualización para el puntaje, vida y score
* **Img:** Carpeta donde se guardan los fondos de la ventana, iconos y carretera principal
* **Power Racing**: Carpeta en la cual contiene el archivo juego.py el cual contiene toda la lógica del juego
* **Sounds:** Carpeta donde almacena los efectos de sonido y música del juego

**Archivos:**

**Juego.py**

Este código importa los módulos y clases necesarios para el juego "Power Racing".

Importa los siguientes módulos: `pygame`, `sys`, `random`, `pygame.mask` y `pygame.mixer`.

* **pygame:** La biblioteca principal que facilita el desarrollo de juegos en Python. Proporciona funciones y herramientas para la creación de gráficos, sonidos, interacción con el teclado y mucho más.
* **sys:** Este módulo proporciona acceso a algunas variables utilizadas o mantenidas por el intérprete y funciones que interactúan fuertemente con el intérprete.
* **random:** Permite la generación de números aleatorios, lo que es útil para introducir elementos de azar en el juego, como la posición inicial de los obstáculos o la velocidad de los oponentes.
* **pygame.mask:** Proporciona funcionalidades para trabajar con máscaras, que son utilizadas para detectar colisiones de manera precisa en el juego.
* **pygame.mixer:** Maneja la reproducción de sonidos. Es esencial para incorporar efectos de sonido, música de fondo u otros elementos auditivos que enriquecen la experiencia del jugador.

También importa clases de los siguientes archivos: `button.py`, `Player.py`, `enemy.py`, `settings.py`, `power.py`, y `road.py`. Además, inicializa el módulo pygame, establece el tamaño de la pantalla, el título, carga imágenes para el fondo y la pantalla de fin del juego.



Se tienen las siguientes funciones:

**regresar() :** esta funcion importa el módulo "Main" y llama a la función "menu\_principal" desde ese módulo, estar diseñada para llevar al usuario de vuelta al menú principal del juego.

**GameOver():** Esta función representa la lógica asociada con la pantalla de "Game Over" en el juego.

* Define una lista de botones llamada buttons\_GameOver, donde cada botón es una instancia de la clase Button. Estos botones tienen propiedades como posición, imágenes normales y en estado de hover, función asociada al clic y sonido de clic.
* La función entra en un bucle while que se ejecuta mientras running sea True.
* Dentro del bucle, maneja eventos de Pygame, como clics y movimientos del mouse. Dependiendo del evento, llama a los métodos apropiados de los botones.
* Rellena la pantalla con un color negro, coloca una imagen de fondo (fondo\_gameOver), dibuja los botones y actualiza la pantalla.
* El bucle continuará hasta que el usuario cierre la ventana del juego (pygame.QUIT), momento en el cual running se establecerá en False, y el juego se cerrará limpiamente.



**poder():** esta función reproduce un efecto de sonido y aumenta en 1 el valor de la variable "num\_vidas".

**crash():** esta funcion comprueba si se ha producido una colisión y actualiza el recuento de colisiones, reproduce un efecto de sonido e imprime el recuento de colisiones, y si el recuento de colisiones excede un cierto valor imprime "GAME OVER" y detiene el juego.

* valor del parámetro: El parámetro "valor" es un valor booleano que indica si se ha producido una colisión o no.



**main\_juego():** esta función contiene la lógica principal del juego, incluyendo la inicialización de variables, la creación de grupos de sprites, el manejo de eventos, la actualización de posiciones y colisiones, y la representación visual en pantalla.

* El juego se ejecuta en un bucle principal hasta que se cumple la condición game\_over.
* Se utilizan clases como Settings, Player, Road, enemy\_car, y Power para representar diferentes elementos del juego, y estos elementos se manejan mediante grupos de sprites.
* La función controla la creación y movimiento de enemigos y poderes, maneja colisiones, actualiza la pantalla con información relevante y ajusta la velocidad del juego a 60 cuadros por segundo.
* Al cerrar el juego, se llama a pygame.quit() y sys.exit() para salir limpiamente del programa.





Texto

Descripción generada automáticamente

**Button.py**

Este código define la clase Button, la cual representa un botón interactivo en una aplicación Pygame con funcionalidad de hover.

* La clase Button se utiliza para crear botones interactivos en una interfaz de usuario Pygame.
* Se inicializa con las propiedades necesarias para posicionar y gestionar el botón, incluyendo las imágenes normales y de hover, así como las funciones asociadas al clic y al hover.
* El método draw dibuja el botón en la pantalla utilizando la imagen correspondiente según el estado de hover.
* Los métodos handle\_click y handle\_hover manejan los eventos de clic y hover, respectivamente. Al hacer clic en el botón, se ejecuta la acción asociada, y al pasar el mouse sobre el botón, se activa la funcionalidad de hover.
* El atributo self.hovered se utiliza para rastrear el estado de hover del botón. Se actualiza cuando el mouse está sobre el área del botón.
* Esta clase proporciona una manera fácil y reutilizable de crear botones interactivos en juegos o aplicaciones Pygame.



**enemy.py**

Este código define la clase enemy\_car, que representa un tipo de enemigo (auto)

* La clase enemy\_car hereda de pygame.sprite.Sprite, lo que la hace compatible con la funcionalidad de sprites en Pygame.
* Se inicializan atributos como el tipo de enemigo (bad), el carril en el que aparece (lane), y se carga la imagen correspondiente según el tipo de enemigo.
* Se ajusta el tamaño de la imagen del enemigo utilizando pygame.transform.scale.
* La función aumentar\_velocidad aumenta la velocidad del auto en función del tiempo transcurrido en el juego. La velocidad se incrementa si el tiempo es un múltiplo de 700.
* La función move mueve el auto hacia abajo en la pantalla, aumentando su posición en el eje Y según la velocidad actual del juego (settings.car\_speed). Si la posición en el eje X supera 650, el auto es eliminado.
* La máscara (self.mask) se utiliza para detectar colisiones precisas en el juego.
* Este código es parte del uego y asume la existencia de otras clases, como Settings, y recursos, como las imágenes de los enemigos, definidos en módulos externos.

****

**load\_sprites.py**

Este código carga diversas imágenes que se utilizan en un juego desarrollado

* El código utiliza la biblioteca Pygame para cargar varias imágenes que se utilizarán en un juego.

Las imágenes cargadas incluyen:

* Player\_image: La imagen del jugador principal, representado por un carro de color rojo.
* Imágenes de enemigos (enemy\_modern\_blue, enemy\_super\_cyan, enemy\_kar\_pink, enemy\_modern\_green, enemy\_modern\_pink): Representan diferentes tipos de enemigos en el juego, cada uno con su propio diseño y color.
* power: Imagen que representa un poder en el juego.
* corazon: Imagen de un corazón, se usa para representar vidas del jugador.
* reloj: Imagen de un reloj, que se usa para representar el tiempo en el juego.
* score: Imagen que podría representar el puntaje en el juego.
* Todas las imágenes se cargan con el método pygame.image.load() y se convierten al formato adecuado con convert\_alpha(), lo que optimiza su rendimiento en Pygame.



**Menu.py**

Este código configura la ventana del juego, carga recursos como imágenes y sonidos, define la apariencia del menú principal y configura la música.

* El código importa las bibliotecas necesarias para el juego, como Pygame, sys y otras personalizadas (Juego y button).
* Inicia Pygame con pygame.init().
* Establece las dimensiones y el título de la ventana del juego mediante pygame.display.set\_mode() y pygame.display.set\_caption().
* Configura el ícono de la ventana con pygame.display.set\_icon().
* Carga imágenes y sonidos que se utilizarán en diferentes partes del juego, como el fondo del menú principal y la música de fondo.
* Se define una fuente para el texto con pygame.font.Font(None, 40).
* Se configura un reloj (clock) para controlar la velocidad de actualización de la pantalla durante el juego.



**StartMusic():** esta función detiene la música que se está reproduciendo actualmente y comienza a reproducir una nueva canción.

Parametro cancion: El parámetro "cancion" es una variable que representa el archivo de música o sonido que quieres escuchar. Se utiliza la biblioteca de Pygame para reproducir música.

**menu\_user():** esta función representa la ventana de introducción del nombre de usuario en el juego.

* La función comienza reproduciendo la música asociada al menú de usuario mediante la llamada a la función StartMusic.
* Configura la fuente de texto utilizando la variable getFont, que probablemente se haya definido anteriormente en el código.
* Utiliza variables globales para almacenar el nombre de usuario introducido.
* Define un objeto Button que representa el botón "OK" en el menú de usuario. Este botón tiene coordenadas, dimensiones y acciones asociadas.
* Entra en un bucle principal que maneja eventos de Pygame, como clics de mouse y pulsaciones de teclas.
* Permite al jugador ingresar texto y utilizar teclas como retroceso y retorno de carro (Enter).
* Actualiza la pantalla continuamente, mostrando el fondo del menú de usuario, el texto ingresado y el botón "OK".
* Al salir del bucle, cierra Pygame y sale del programa utilizando pygame.quit() y sys.exit().



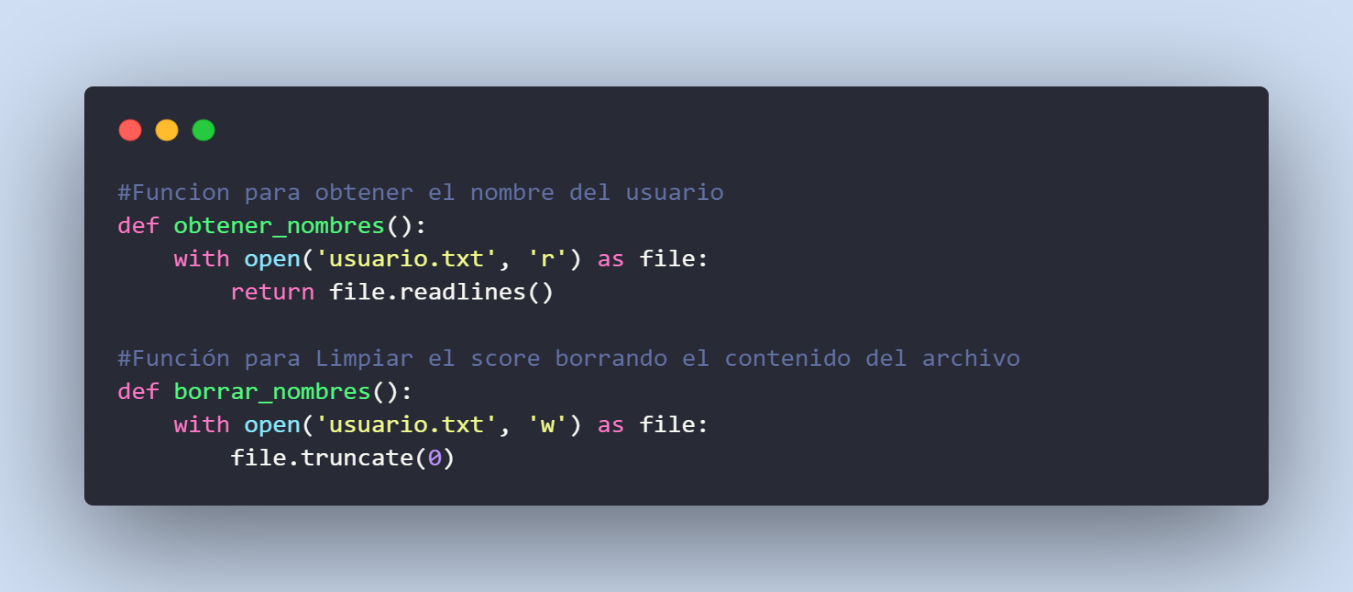


**obtener\_nombres():**

* Esta función lee el contenido del archivo 'usuario.txt' en modo lectura ('r').
* Utiliza el contexto de un bloque with para garantizar que el archivo se cierre correctamente después de leerlo.
* Retorna una lista de líneas, donde cada línea contiene un nombre de usuario.

**borrar\_nombres():**

* Esta función abre el archivo 'usuario.txt' en modo escritura ('w').
* Utiliza el contexto de un bloque with para garantizar que el archivo se cierre correctamente después de realizar operaciones de escritura.
* Utiliza el método truncate(0) para borrar el contenido del archivo, dejándolo vacío.

****

**menu\_score():** esta función, llamada menu\_score(), representa la ventana de puntuaciones del juego.

* La función comienza reproduciendo la música asociada a la pantalla de puntuaciones mediante la llamada a la función StartMusic.
* Crea objetos Button que representan botones para volver al menú principal y borrar las puntuaciones. Estos botones tienen coordenadas, dimensiones y acciones asociadas.
* Entra en un bucle principal que maneja eventos de Pygame, como clics de mouse y pulsaciones de teclas.
* Muestra en pantalla el fondo de la ventana de puntuaciones, títulos ("Users" y "Points"), y los nombres y puntajes almacenados.
* Permite al usuario interactuar con los botones, como hacer clic para volver al menú principal o borrar las puntuaciones.
* Actualiza la pantalla continuamente mientras el bucle está en ejecución.
* Al salir del bucle, cierra Pygame y sale del programa utilizando pygame.quit() y sys.exit().
* Esta función proporciona a los jugadores una interfaz para ver las puntuaciones almacenadas y realizar acciones como volver al menú principal o borrar las puntuaciones.

****

****

**menu\_help():** esta función representa la ventana de ayuda del juego

* La función comienza reproduciendo la música asociada a la pantalla de ayuda mediante la llamada a la función StartMusic.
* Crea un objeto Button que representa un botón para volver al menú principal. Este botón tiene coordenadas, dimensiones y una acción asociada.
* Entra en un bucle principal que maneja eventos de Pygame, como clics de mouse y pulsaciones de teclas.
* Muestra en pantalla el fondo de la ventana de ayuda y el botón para volver al menú principal.
* Permite al usuario interactuar con el botón, como hacer clic para volver al menú principal.
* Actualiza la pantalla continuamente mientras el bucle está en ejecución.
* Al salir del bucle, cierra Pygame y sale del programa utilizando pygame.quit() y sys.exit().

Esta función proporciona a los jugadores información adicional o tutoriales sobre el juego mediante una interfaz de usuario simple, y les permite regresar al menú principal cuando lo deseen.



**menu\_principal():** esta función representa la ventana del menú principal del juego.

* La función comienza reproduciendo la música de fondo del menú mediante la llamada a la función StartMusic.
* Define una lista de objetos Button, cada uno representando un botón en el menú principal. Cada botón tiene coordenadas, dimensiones, imágenes y acciones asociadas.
* Entra en un bucle principal que maneja eventos de Pygame, como clics de mouse y pulsaciones de teclas.
* Muestra en pantalla el fondo del menú principal y los botones disponibles.
* Permite al usuario interactuar con los botones, como hacer clic para ir a otras secciones del juego o salir del juego.
* Actualiza la pantalla continuamente mientras el bucle está en ejecución.
* Al salir del bucle, cierra Pygame y sale del programa utilizando pygame.quit() y sys.exit().

Esta función crea la interfaz principal del juego, permitiendo a los jugadores acceder a diferentes secciones del juego, como jugar, ver puntajes, obtener ayuda o salir del juego.Principio del formulario

