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 a lo largo del camino y sobrevivir el mayor tiempo posible.

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` y `power.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.

```
import pygame
import sys
import random
import pygame.mask
import pygame.mixer
from button import *
pygame.init()

screen_size = pygame.display.set_mode([800, 500])
pygame.display.set_caption("Power Racing")
clock = pygame.time.Clock()

background = pygame.image.load("./Img/Carretera.png").convert()
background_width = background.get_width()
fondo_gameOver = pygame.image.load("./Img/game_over.png")

from Player import *
from enemy import *
from settings import *
from power import *
from road import *
```

Se tienen las siguientes funciones:

capturar_nombre(name)

- Descripción: Esta función toma el nombre del usuario como argumento y lo asigna a la variable global nombreJugador.
- Uso en el Código: Se llama cuando se captura el nombre del usuario en el juego.

regresar()

- Descripción: Esta función importa el módulo Main (presumiblemente contiene el código principal del juego) y llama a la función menu_principal(), regresando así al menú principal del juego.
- Uso en el Código: Se utiliza para regresar al menú principal desde otras partes del juego.

agregar_nombre()

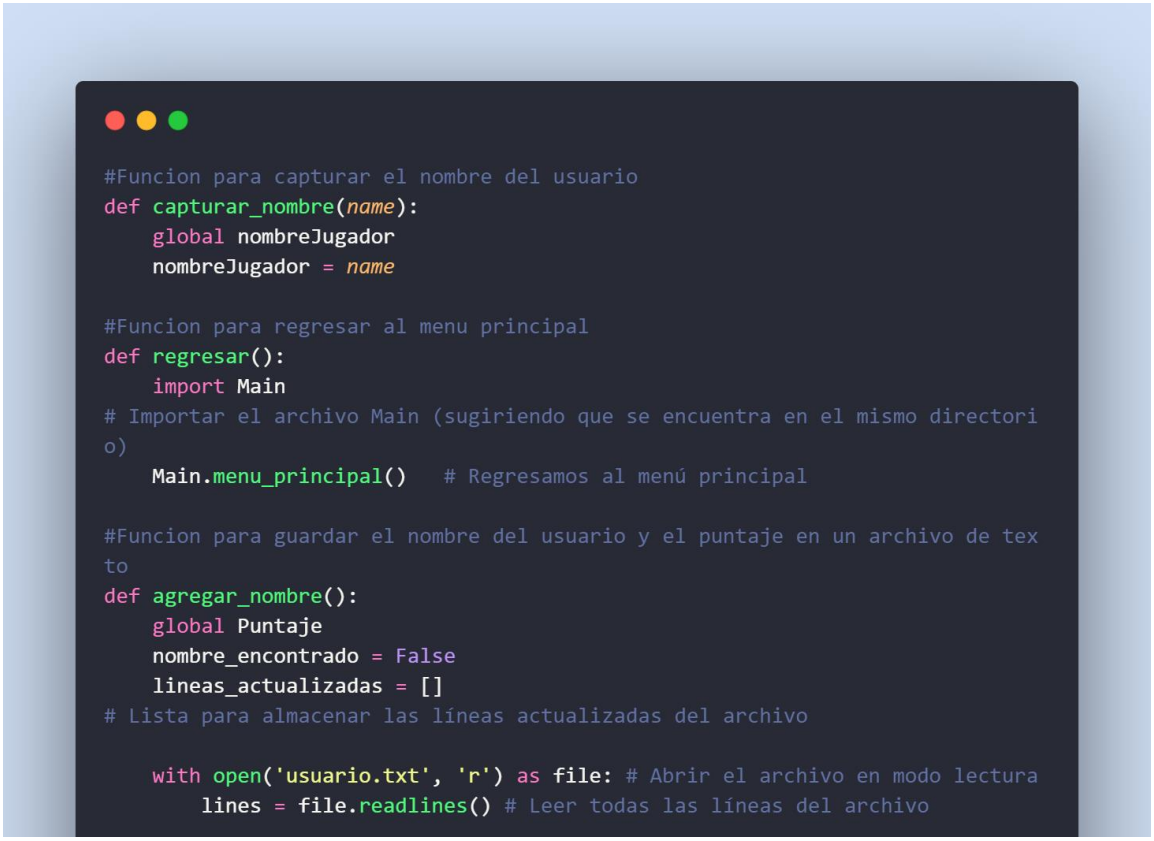
- Descripción: Esta función se encarga de actualizar el archivo de texto 'usuario.txt' con el nombre del jugador y su puntaje. Si el nombre ya existe en el archivo, compara el puntaje actual con el puntaje almacenado y actualiza si es mayor. Si el nombre no existe, agrega un nuevo registro al archivo.
- Uso en el Código: Se llama cuando se desea guardar el nombre del jugador y su puntaje al finalizar el juego.

Notas Adicionales:

nombreJugador: Variable global utilizada para almacenar el nombre del jugador.

Puntaje: Variable global que debe estar definida en algún lugar del código principal del juego antes de llamar a agregar_nombre(). Representa el puntaje del jugador.

Estas funciones están diseñadas para gestionar la información del jugador, permitiendo la captura y actualización de su nombre y puntaje en un archivo de texto.



```python
#Funcion para capturar el nombre del usuario
def capturar_nombre(name):
    global nombreJugador
    nombreJugador = name

#Funcion para regresar al menu principal
def regresar():
    import Main
# Importar el archivo Main (sugiriendo que se encuentra en el mismo directori
o)
    Main.menu_principal()   # Regresamos al menú principal

#Funcion para guardar el nombre del usuario y el puntaje en un archivo de tex
to
def agregar_nombre():
    global Puntaje
    nombre_encontrado = False
    lineas_actualizadas = []
# Lista para almacenar las líneas actualizadas del archivo

    with open('usuario.txt', 'r') as file: # Abrir el archivo en modo lectura
        lines = file.readlines() # Leer todas las líneas del archivo
```

```
        for line in lines: # Iterar sobre cada línea
            parts = line.split(', ') # Dividir la línea en nombre y puntaje
            nombre = parts[0] # Obtener el nombre
            puntaje = int(parts[1]) # Obtener el puntaje
            if nombre == nombreJugador:
# Si el nombre coincide con el nombre del jugador
                nombre_encontrado = True # Establecer la bandera en True
                if Puntaje > puntaje:
# Si el puntaje actual es mayor que el puntaje almacenado
                    line = f"{nombreJugador}, {Puntaje}\n"
# Actualizar la línea con el nuevo puntaje
            lineas_actualizadas.append(line.rstrip('\n'))
# Elimina el salto de línea existente

    if not nombre_encontrado: # Si el nombre no se encontró en el archivo
        lineas_actualizadas.append(f"{nombreJugador}, {Puntaje}")

    with open('usuario.txt', 'w') as file:
# Abrir el archivo en modo escritura
        file.write('\n'.join(lineas_actualizadas))
# Une todas las líneas con saltos de línea

    if nombre_encontrado: # Si el nombre se encontró en el archivo
        print("Puntaje actualizado correctamente.")
    else:
        print("Nuevo registro agregado.")
```

GameOver(): Descripción: Esta función se encarga de mostrar el menú de Game Over y proporciona la funcionalidad de dos botones: uno para reiniciar el juego y otro para regresar al menú principal

- La función crea dos botones, cada uno representado por un objeto de la clase Button. Estos botones se almacenan en la lista buttons_GameOver.
- El bucle principal (while running:) se ejecuta hasta que la variable running se establece en False. Esto permite mantener la ventana del menú de Game Over abierta hasta que el jugador realice una acción.
- Dentro del bucle, se detectan los eventos del usuario, como clics del mouse y movimientos, para interactuar con los botones.
- Cuando se detecta un clic en uno de los botones, se ejecuta la función asociada al botón, ya sea reiniciar el juego o regresar al menú principal.
- La pantalla se actualiza con el fondo del menú de Game Over y los botones dibujados en ella.

Esta función proporciona una interfaz gráfica para que el jugador pueda decidir si desea reiniciar el juego o regresar al menú principal después de perder.

```python
#Funcion para mostrar el menu de game over y su funcionalidad
def GameOver():
    buttons_GameOver = [
    Button(200,440,200,53, pygame.image.load("Buttons/play.png"
), pygame.image.load("Buttons/play_on.png"), main_juego, music_click.play),
    Button(405,440,200,53, pygame.image.load("Buttons/back.png"
),pygame.image.load("Buttons/back_on.png"),regresar, music_click.play)
    ]
    running = True # Variable para controlar el bucle principal
    while running: # Bucle principal
        for event in pygame.event.get(): # Iterar sobre los eventos
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEMOTION:
                    for button in buttons_GameOver:
                        button.handle_hover()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                    for button in buttons_GameOver:
                        button.handle_click()

        screen_size.fill((0, 0, 0))
        screen_size.blit(fondo_gameOver, (0, 0))
        for button in buttons_GameOver:
            button.draw(screen_size)
        pygame.display.flip()

    pygame.quit()
    sys.exit()
```

crash(value): Esta función se encarga de detectar colisiones en el juego. Controla el aumento del contador de colisiones, reproduce un sonido de choque y, si el número de colisiones alcanza el límite de vidas, imprime "GAME OVER", detiene la música, agrega el nombre del usuario y el puntaje al archivo de texto y muestra el menú de Game Over.

Parámetros:

- value: Un valor booleano que indica si se ha producido una colisión (True) o no (False).

Variables Globales Utilizadas:

- aux: Variable que controla el estado de la colisión (si ya se ha registrado o no).

collision_count: Contador de colisiones, se incrementa cada vez que se detecta una colisión.

La función utiliza variables globales (aux y collision_count) para controlar el estado de las colisiones y llevar la cuenta de cuántas colisiones ha experimentado el jugador.

Esta función es esencial para gestionar el estado del juego cuando se producen colisiones y decide cuándo mostrar el menú de Game Over.

```python
#Función para detectar colisiones
def crash(value):
    global aux,collision_count #Variables globales para controlar el choque
    if value == True and aux == False: #Si el valor es True y aux es False
        aux = True  #Cambiamos el valor de aux a True
        collision_count += 1 #Aumentamos el contador de colisiones
        sound_shok_channel.play(sound_shok) #Reproducimos el sonido de choque
        print("Choque:", collision_count) #Imprimimos el numero de choques

    if value == False and aux == True: #Si el valor es False y aux es True
        aux = False #Cambiamos el valor de aux a False

    if collision_count >= settings.num_vidas:
#Si el numero de choques es mayor o igual al numero de vidas
        print("GAME OVER") #Imprimimos Game Over
        pygame.mixer.stop() #Detenemos la musica
        agregar_nombre()
#Agregamos el nombre del usuario y el puntaje al archivo de texto
        GameOver()  #Mostramos el menu de Game Over
```

main_juego(): Esta función contiene la lógica principal del juego. Gobierna la generación de enemigos y poderes, el movimiento de sprites, la detección de colisiones, la actualización de la pantalla y la gestión de eventos del jugador.

Variables Locales Utilizadas:

- **scroll:** Variable que controla el desplazamiento de la carretera.
- **nombreJugador:** Variable que almacena el nombre del jugador.
- **collision_count:** Contador de colisiones.
- **tiempo:** Contador de tiempo transcurrido.
- **Puntaje:** Puntuación del jugador.
- **aumento_vel:** Velocidad de desplazamiento de la carretera.
- **fuente:** Objeto de fuente utilizado para renderizar texto.
- **settings:** Objeto de la clase Settings que contiene la configuración del juego.
- **player:** Objeto de la clase Player que representa al jugador.
- **sound_car, sound_shok, sound_power:** Objetos de sonido para el carro, choque y poder.

- **sound_car_channel, sound_shok_channel, sound_power_channel:** Canales de sonido asociados a los sonidos correspondientes.
- **music_click:** Objeto de sonido para el clic de botón.
- **all_sprites, enemy_sprites, power_sprites:** Grupos de sprites para todos los sprites, enemigos y poderes, respectivamente.
- **enemy_timer:** Temporizador para controlar la generación de enemigos.
- **aux:** Variable que controla el estado de colisiones.
- **invulnerable:** Variable que indica si el jugador está en estado invulnerable.
- **invulnerable_timer:** Temporizador para controlar la duración del estado invulnerable.
- **INVULNERABLE_DURATION:** Duración en frames del estado invulnerable.

Detalles Adicionales:

La función inicializa todas las variables y objetos necesarios para el juego.

Configura canales de audio y carga sonidos.

Utiliza grupos de sprites (all_sprites, enemy_sprites, power_sprites) para facilitar la manipulación y actualización de los sprites en pantalla.

La lógica del juego se ejecuta en un bucle principal, donde se gestionan eventos, se actualizan las posiciones de los sprites, se detectan colisiones, se actualiza la pantalla y se incrementan el tiempo y el puntaje.

Los enemigos y los poderes se generan y mueven en la pantalla de acuerdo con ciertos intervalos de tiempo especificados en la configuración del juego.

Se utiliza la detección de colisiones mediante spritecollide y máscaras de colisión para gestionar las interacciones entre el jugador, los enemigos y los poderes.

Se implementa un sistema de invulnerabilidad después de una colisión para evitar colisiones consecutivas en un corto período.

La función muestra en pantalla información importante como el número de vidas, el tiempo transcurrido y el puntaje actual del jugador.

La ejecución del juego continúa hasta que se detecta una colisión que supera el límite de vidas, momento en el cual se imprime "GAME OVER", se detiene la música, se agrega el nombre del jugador y el puntaje al archivo de texto, y se muestra el menú de Game Over.

```python
#Lógica del juego
def main_juego():
    pygame.mixer.stop()
    global scroll, nombreJugador, collision_count, tiempo, Puntaje,
aumento_vel, fuente, settings, player, sound_car, sound_shok,
sound_shok_channel, music_click, all_sprites, enemy_sprites, hueco_sprites,
power_sprites, enemy_timer, aux
    fuente = pygame.font.SysFont("Pixel Operator", 30)
    # Configura los canales de audio
    pygame.mixer.set_num_channels(3)  # Establece dos canales de audio
    sound_car = pygame.mixer.Sound("./Sounds/move.mp3")
    sound_car_channel = pygame.mixer.Channel(0)
# Establece el canal 0 para el sonido del carro
    sound_car.set_volume(0.5)
    sound_shok = pygame.mixer.Sound("./Sounds/choque.mp3")
    sound_shok_channel = pygame.mixer.Channel(1)
# Establece el canal 1 para el sonido del choque
    sound_power = pygame.mixer.Sound("./Sounds/power.mp3")
    sound_power_channel = pygame.mixer.Channel(2)
# Establece el canal 2 para el sonido del poder
    music_click = pygame.mixer.Sound("./Sounds/buttonClick.mp3")
    all_sprites = pygame.sprite.Group()
# Crea un grupo para todos los sprites
    enemy_sprites = pygame.sprite.Group() # Crea un grupo para los enemigos
    power_sprites = pygame.sprite.Group() # Crea un grupo para los poderes
    settings = Settings()
    player = Player()
    all_sprites.add(player)
    enemy_timer = 0
    aux = False
    collision_count = 0
    tiempo = 0
    Puntaje = 0
    aumento_vel = 5
    game_over = False
    scroll = 0
    power_timer = 0
    invulnerable = False
# Variable para rastrear si el jugador es invulnerable
    invulnerable_timer = 0
# Temporizador para controlar la duración de la invulnerabilidad
    INVULNERABLE_DURATION = 30  # Duración en frames (0.5 segundos a 60 FPS)
    sound_car_channel.play(sound_car, loops=-1)
# Reproduce el sonido del carro en un bucle infinito
    while not game_over:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()

            player.process_event_car(event)

        player.move_car()
```

```
#sound_car_channel.play(sound_car, loops=-1)  # Reproduce el sonido del carro
en un bucle infinito
        # Desplazamiento de la carretera
        screen_size.blit(background, (0, scroll))
        screen_size.blit(background, (0, scroll - background.get_height()))
# Copia desplazada
        scroll += aumento_vel  # Velocidad de desplazamiento

        if scroll >= background.get_height():
            scroll = 0
# Restablece la posición cuando alcanza el tamaño de la imagen de fondo
        if aumento_vel <= 10:
            if tiempo % 700 == 0:
                aumento_vel += 0.5
                print("Velocidad de la carretera: ", aumento_vel)

        # Crea los enemigos
        enemy_timer += 1
        if enemy_timer >= settings.time_enemy:
            enemy_timer = 0

# Generar una lista de carriles disponibles sin incluir el carril actual del
jugador
            available_lanes = [settings.carril_1, settings.carril_2, settings
.carril_3, settings.carril_4]
            if player.rect.x in available_lanes:
                available_lanes.remove(player.rect.x)
            if available_lanes:

# Si hay más de un carril disponible, se selecciona aleatoriamente uno
                if len(available_lanes) > 1:
                    lane = random.choice(available_lanes)
                else:
                    # Si solo queda un carril disponible, se toma ese
                    lane = available_lanes[0]
                enemy = enemy_car(random.randint(1, 5), lane)
                all_sprites.add(enemy)
                enemy_sprites.add(enemy)
        # Crea los poderes
        power_timer += 1
        if power_timer >= settings.time_power:
            power_timer = 0
            lane = random.choice([settings.carril_1, settings.carril_2,
settings.carril_3, settings.carril_4])
            power = Power(lane)
            all_sprites.add(power)
            power_sprites.add(power)

        # Eliminar enemigos fuera de la pantalla
        for enemy in enemy_sprites.copy():
            if enemy.rect.y > screen_size.get_height():
                enemy.kill()

        # Eliminar poderes fuera de la pantalla
        for power in power_sprites.copy():
            if power.rect.y > screen_size.get_height():
                power.kill()
```

```
        # Mueve los enemigos
        for enemy in enemy_sprites:
            enemy.move()
        # Mueve el poder
        for power in power_sprites:
            power.move()

        power_collision_list = pygame.sprite.spritecollide(player,
power_sprites, True, pygame.sprite.collide_mask)
        for power in power_collision_list:
            #detenemos la musica del movimiento del carro
            sound_power_channel.play(sound_power)
            settings.num_vidas += 1

        # Colisiones
        for enemy in enemy_sprites:
        # Si el jugador no está en un estado invulnerable
            if not invulnerable:
                car_collision_list = pygame.sprite.spritecollide(player,
enemy_sprites, False, pygame.sprite.collide_mask)
                if car_collision_list:
                    for colliding_enemy in car_collision_list:
                        colliding_enemy.colision_move(1)
                    crash(True)
                    invulnerable = True
# Activar el estado de invulnerabilidad
                    invulnerable_timer = 0  # Reiniciar el temporizador
                else:
                    crash(False)
        # Actualizar el temporizador de invulnerabilidad
        if invulnerable:
            invulnerable_timer += 1
            if invulnerable_timer >= INVULNERABLE_DURATION:
                invulnerable = False
# Desactivar la invulnerabilidad después de la duración especificada


        # Dibuja los sprites
        all_sprites.draw(screen_size)
        texto = str(settings.num_vidas-collision_count)
        Texto1 = fuente.render(texto,False,white)
        screen_size.blit(corazon,(30,10))
        screen_size.blit(reloj,(360,10))
        screen_size.blit(Texto1,(45,22))
        screen_size.blit(score,(650,10))
        tiempo += 1 #Aumentamos enl tiempo con cada iteración
        texto_tiempo = fuente.render( str(tiempo), False, white)
#creamos el texto del tiempo
        screen_size.blit(texto_tiempo, (410, 20))
#Mostramos el tiempo en pantalla
        if tiempo % 50 == 0:
            Puntaje += 1 #Aumentamos el puntaje cada 2 segundos
        texto_puntaje = fuente.render(str(Puntaje), False, white)
#creamos el texto del puntaje
        screen_size.blit(texto_puntaje, (700, 20))
#Mostramos el puntaje en pantalla
        pygame.display.flip()
        clock.tick(60)

    pygame.quit()
    sys.exit()
```

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.

```python
import pygame

class Button:
    def __init__(self, x, y, width, height,image,hover_image,action,
hover_action):
        self.rect = pygame.Rect(x, y, width, height)
#rectángulo que representa el botón
        self.action = action
#función que se ejecuta cuando se hace click en el botón
        self.hover_action = hover_action
#función que se ejecuta cuando el mouse está sobre el botón
        self.image = image
#imagen que se muestra cuando el mouse no está sobre el botón
        self.hover_image = hover_image
#imagen que se muestra cuando el mouse está sobre el botón
        self.hovered = False  #atributo para rastrear el estado de hover

    # Dibuja el botón en la pantalla
    def draw(self, screen):
        if self.hovered:
            screen.blit(self.hover_image, self.rect.topleft)
        else:
            screen.blit(self.image, self.rect.topleft)
    # Maneja el evento click
    def handle_click(self):
        if self.rect.collidepoint(pygame.mouse.get_pos()):
            self.action()
    # Maneja el evento hover
    def handle_hover(self):
        if self.rect.collidepoint(pygame.mouse.get_pos()):
            if not self.hovered:
                self.hovered = True
# Cambia a True solo si no estaba previamente en ese estado


# Llama a hover_action si self.hovered es True y hover_action está definida
                if self.hover_action is not None and callable(self
.hover_action):
                    self.hover_action()
        else:
            self.hovered = False
```

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.

```python
import pygame
import pygame.mask

pygame.init()

from load_sprites import *
from pygame.sprite import *
from settings import *
# Se instancia la clase Settings
settings = Settings()
#Se crea la clase enemy_car
class enemy_car(pygame.sprite.Sprite):
    def __init__(self, bad,lane):
        super().__init__()
        self.bad = bad # Tipo de enemigo
        self.lane = lane # Carril en el que aparece
        # Carga la imagen del enemigo dependiendo del tipo
        if self.bad == 1:
            self.image = enemy_modern_blue
        elif self.bad == 2:
            self.image = enemy_super_cyan
        elif self.bad == 3:
            self.image = enemy_kar_pink
        elif self.bad == 4:
            self.image = enemy_modern_green
        else:
            self.image = enemy_modern_pink

        #se inician los demas atributos de la clase `enemy_car`.
        self.rect = self.image.get_rect()
        self.image = pygame.transform.scale(self
.image, (settings.car_width, settings.car_height))
        self.rect.x = self.lane
        self.rect.y = settings.enemy_pos_y
        self.mask = pygame.mask.from_surface(self.image)
    # Función que aumenta la velocidad de los autos
    def aumentar_velocidad(self):
        from Juego import tiempo
        if settings.car_speed <= 10:
            if tiempo % 700 == 0:
                settings.car_speed += 0.5
                print("Velocidad de los autos: ", settings.car_speed)
    # Función que mueve los autos
    def move(self):
        if self.rect.x < 650:
            self.rect.y += settings.car_speed
            self.aumentar_velocidad()
        else:
            self.kill()
```

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.

```python
import pygame

#JUGADOR PRINCIPAL
Player_image = pygame.image.load("cars/modern_red.png").convert_alpha()

#ENEMIGOS
enemy_modern_blue = pygame.image.load("cars/modern_blue.png").convert_alpha()
enemy_super_cyan = pygame.image.load("cars/super_cyan.png").convert_alpha()
enemy_kar_pink = pygame.image.load("cars/kar_pink.png").convert_alpha()
enemy_modern_green = pygame.image.load("cars/modern_green.png"
).convert_alpha()
enemy_modern_pink = pygame.image.load("cars/modern_pink.png").convert_alpha()
hueco = pygame.image.load("cars/hueco.png").convert_alpha()

#PODER
power = pygame.image.load("cars/power.png").convert_alpha()

#corazon
corazon = pygame.image.load("cars/corazon.png").convert_alpha()

#reloj
reloj = pygame.image.load("cars/reloj.png").convert_alpha()

#score
score = pygame.image.load("cars/score.png").convert_alpha()
```

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.

```python
import pygame
import sys
from Juego import *
from button import *
import pygame.mixer

pygame.init()
Ancho = 800
Alto = 500
size = (Ancho, Alto)
pygame.display.set_caption("Power Racing")
screen = pygame.display.set_mode(size)
# Tamaño del ícono deseado
icon_size = (64, 64)  # Puedes ajustar este tamaño a tu preferencia
icono = pygame.image.load("./Img/icono.png")
icon = pygame.transform.scale(icono, icon_size)
pygame.display.set_icon(icon)
clock = pygame.time.Clock()

# El código carga imágenes y sonidos para el menú del juego.
fondo_menu = pygame.image.load("./Img/Power Racing 2.png")
fondo_user = pygame.image.load("./Img/user_main.png")
fondo_score = pygame.image.load("./Img/score_main.png")
fondo_help = pygame.image.load("./Img/help_main.png")

music_menu = pygame.mixer.Sound("./Sounds/music_menu.mp3")
music_menu.set_volume(0.2)
music_user = pygame.mixer.Sound("./Sounds/music_user.mp3")
music_score = pygame.mixer.Sound("./Sounds/music_score.mp3")
music_help = pygame.mixer.Sound("./Sounds/music_help.mp3")
music_car = pygame.mixer.Sound("./Sounds/motionCar.mp3")
music_click = pygame.mixer.Sound("./Sounds/buttonClick.mp3")
getFont = pygame.font.Font(None, 40) # Fuente para el texto
```

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().

```
def StartMusic(cancion):
    #Detener musica
    pygame.mixer.stop()
    cancion.play()

#Ventana Usuario
def menu_user():
    #Reproducir musica
    StartMusic(music_user)
    fuente = getFont
    global nombreUsuario
    nombreUsuario = ""
    texto = ""
    button_menu_user = Button(305,350,200,53, pygame.image.load("
Buttons/ok.png"),pygame.image.load("Buttons/ok_on.png"
),main_juego, music_click.play)
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEMOTION:
                    button_menu_user.handle_hover()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                    print("Texto ingresado:", texto)
                    nombreUsuario = texto
                    texto = ""
                    capturar_nombre(nombreUsuario)
                    button_menu_user.handle_click()
```

```
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_BACKSPACE:
                    texto = texto[:-1]
                elif event.key == pygame.K_RETURN:
                    print("Texto ingresado:", texto)
                    nombreUsuario = texto
                    texto = ""
                    capturar_nombre(nombreUsuario)
                    main_juego()
                else:
                    texto += event.unicode

        screen.fill((0, 0, 0))
        screen.blit(fondo_user, (0, 0))
        texto_superficie = fuente.render(texto, True, "white")
        texto_rect = texto_superficie.get_rect()
        texto_rect.center = (400, 250)
        screen.blit(texto_superficie, texto_rect)
        button_menu_user.draw(screen)
        pygame.display.flip()

    pygame.quit()
    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.

```
#Funcion para obtener el nombre del usuario
def obtener_nombres():
    with open('usuario.txt', 'r') as file:
        return file.readlines()

#Función para Limpiar el score borrando el contenido del archivo
def borrar_nombres():
    with open('usuario.txt', 'w') as file:
        file.truncate(0)
```

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.

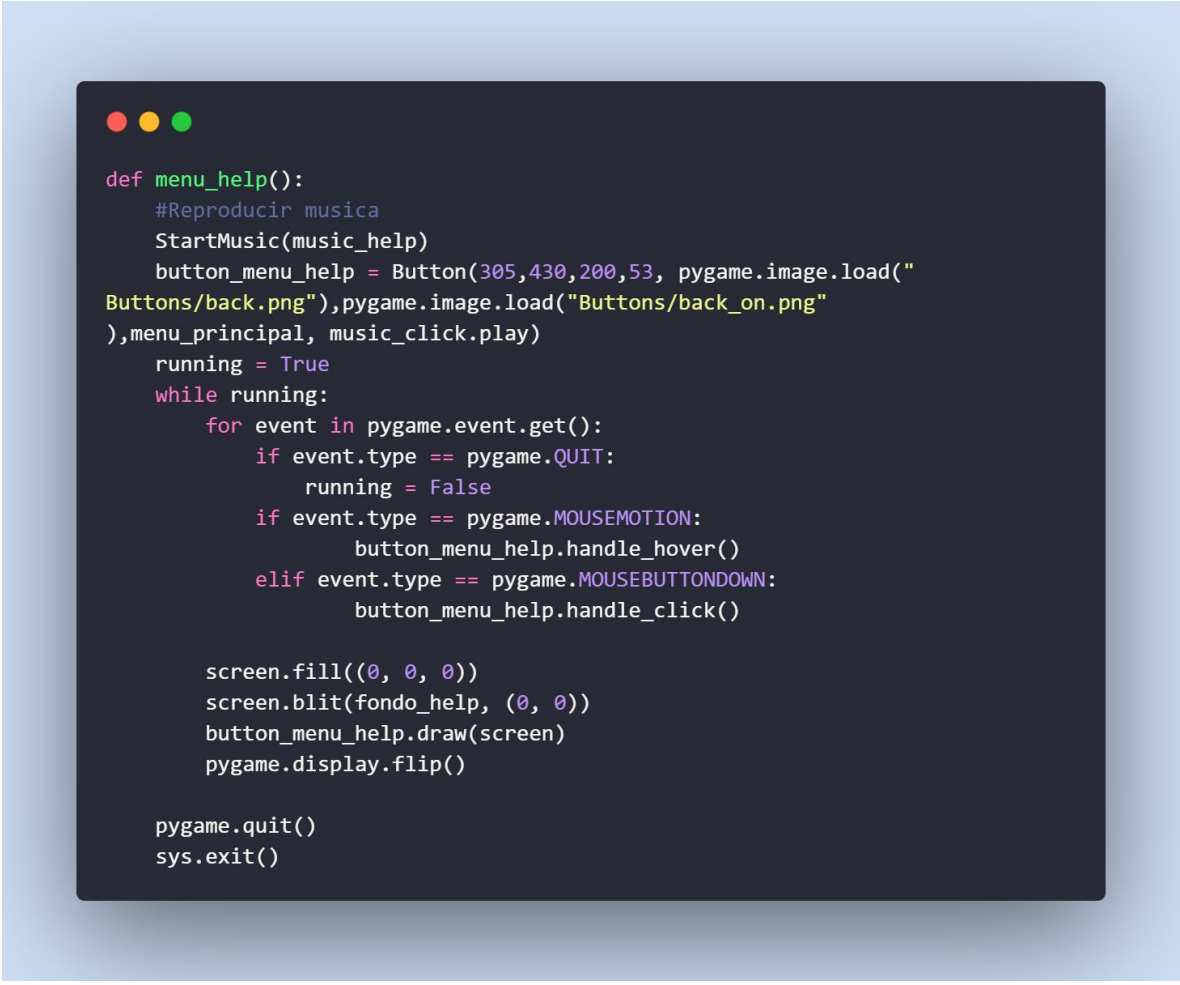
```
def menu_score():
    #Reproducir musica
    StartMusic(music_score)
    buttons_score = [
    Button(47,440,200,53, pygame.image.load("Buttons/back.png"
),pygame.image.load("Buttons/back_on.png"),menu_principal, music_click.play),
    Button(248,440,200,53, pygame.image.load("Buttons/clear.png"
),pygame.image.load("Buttons/clear_on.png"),borrar_nombres, music_click.play)
    ]
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEMOTION:
                for button in buttons_score:
                    button.handle_hover()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                for button in buttons_score:
                    button.handle_click()

        screen.fill((0, 0, 0))
        screen.blit(fondo_score, (0, 0))

        # Mostrar título
        fuente_nombre = getFont
        title_user = fuente_nombre.render("Users", True, "white")
        title_points = fuente_nombre.render("Points", True, "white")
        title_rect = title_user.get_rect()
        title_rect.center = (250, 160)  # Posición de "Users"
        screen.blit(title_user, title_rect)

        points_rect = title_points.get_rect()
        points_rect.center = (590, 160)  # Posición de "Points"
        screen.blit(title_points, points_rect)
```

```
        # Mostrar nombres y puntajes
        nombres = obtener_nombres()  # Obtener todos los nombres del archivo
        y_offset = 200  # Ajustar la posición vertical inicial

        for line in nombres:    # Iterar sobre cada línea
            line = line.strip()
# Eliminar espacios en blanco y saltos de línea
            nombre, puntaje = line.split(",")
# Dividir la línea en nombre y puntaje

            # Mostrar el nombre centrado debajo del título "Users"
            texto_nombre = fuente_nombre.render(nombre, True, "white")
            texto_nombre_rect = texto_nombre.get_rect()
            texto_nombre_rect.center = (250, y_offset)
            screen.blit(texto_nombre, texto_nombre_rect)

            # Mostrar el puntaje centrado debajo del título "Points"
            texto_puntaje = fuente_nombre.render(puntaje.strip(), True, "
white")
            texto_puntaje_rect = texto_puntaje.get_rect()
            texto_puntaje_rect.center = (590, y_offset)
            screen.blit(texto_puntaje, texto_puntaje_rect)

            y_offset += 30
# Ajustar el espacio vertical entre nombres y puntajes

        for button in buttons_score:    # Dibujar los botones
            button.draw(screen)
        pygame.display.flip()

    pygame.quit()
    sys.exit()
```

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.



```
def menu_help():
    #Reproducir musica
    StartMusic(music_help)
    button_menu_help = Button(305,430,200,53, pygame.image.load("
Buttons/back.png"),pygame.image.load("Buttons/back_on.png"
),menu_principal, music_click.play)
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEMOTION:
                    button_menu_help.handle_hover()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                    button_menu_help.handle_click()

        screen.fill((0, 0, 0))
        screen.blit(fondo_help, (0, 0))
        button_menu_help.draw(screen)
        pygame.display.flip()

    pygame.quit()
    sys.exit()
```

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.

```python
def menu_principal():
    StartMusic(music_menu)  #Reproducir musica
    list_buttons_principal = [
    Button(361,300,400,400, pygame.image.load("Buttons/auto.png"
),pygame.image.load("Buttons/auto_on.png"),None, music_car.play),
    Button(130,15,563,70,pygame.image.load("Buttons/titulo.png"
),pygame.image.load("Buttons/titulo_on.png"),None, None),
    Button(50, 130,200,62, pygame.image.load("Buttons/play.png"
), pygame.image.load("Buttons/play_on.png"), menu_user, music_click.play),
    Button(50, 220,200,62,pygame.image.load("Buttons/score.png"
), pygame.image.load("Buttons/score_on.png"), menu_score, music_click.play),
    Button(50, 310,200,62, pygame.image.load("Buttons/help.png"
), pygame.image.load("Buttons/help_on.png"), menu_help, music_click.play),
    Button(50, 400,200,62, pygame.image.load("Buttons/exit.png"
), pygame.image.load("Buttons/exit_on.png"), sys.exit, music_click.play)
    ]
    button_main  = list_buttons_principal

    #Ciclo principal del menú
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEMOTION:
                for button in button_main:
                    button.handle_hover()

            elif event.type == pygame.MOUSEBUTTONDOWN:
                for button in button_main:
                    button.handle_click()

        screen.fill((0, 0, 0))
        screen.blit(fondo_menu, (0, 0))
        for button in button_main:
            button.draw(screen)
        pygame.display.flip()

    pygame.quit()
    sys.exit()

if __name__ == "__main__":
    menu_principal()
```

Player.py

Este código define una clase llamada Player en Pygame, que representa el objeto del jugador, es decir, el carro que se mueve en la pantalla.

- La clase Player hereda de pygame.sprite.Sprite, lo que la hace compatible con el sistema de sprites de Pygame.
- Se inicializan los atributos del jugador, como la imagen, el rectángulo, la máscara de colisión, las variables de movimiento y la velocidad.
- Hay funciones específicas para mover el carro a la derecha e izquierda, asegurándose de que no salga de ciertos límites en la pantalla.
- La función process_event_car maneja los eventos relacionados con las teclas presionadas y soltadas para iniciar y detener el movimiento del carro.
- La función move_car llama a las funciones de movimiento dependiendo de las teclas presionadas.

```python
import pygame
import pygame.mask
from load_sprites import *
from pygame.sprite import *
from settings import *
pygame.init()
# Se instancia la clase Settings
settings = Settings()

#Se crea la clase Player
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = Player_image # Imagen del jugador
        self.rect = Player_image.get_rect() # Rectángulo de la imagen
        self.image = pygame.transform.scale(self.image, (settings.car_width,
settings.car_height))
        self.rect.x = settings.car_pos_x   # Posición inicial en x
        self.rect.y = settings.car_pos_y   # Posición inicial en y
        self.mask = pygame.mask.from_surface(self.image)
# Máscara de colisión
        self.moving_right = False
# Variable que indica si se mueve a la derecha
        self.moving_left = False
# Variable que indica si se mueve a la izquierda
        self.velocidad = 4  # Velocidad de movimiento

    def move_right(self, pixels): # Función que mueve el carro a la derecha
        if self.rect.x < 550:
            self.rect.x += pixels
```

```
    def move_left(self,pixels): # Función que mueve el carro a la izquierda
        if self.rect.x > 200:
            self.rect.x -= pixels

    def process_event_car(self, event):
# Función que procesa los eventos del carro
        if event.type == pygame.KEYDOWN: # Si se presiona la tecla, se mueve
            if event.key == pygame.K_d:
                self.moving_right = True
            if event.key == pygame.K_a:
                self.moving_left = True

        if event.type == pygame.KEYUP:
# Si se suelta la tecla, se detiene el movimiento
            if event.key == pygame.K_d:
                self.moving_right = False
            if event.key == pygame.K_a:
                self.moving_left = False

    def move_car(self): # Función que mueve el carro
        if self.moving_right:
            self.move_right(self.velocidad)
        if self.moving_left:
            self.move_left(self.velocidad)
```

power.py

Este código define una clase llamada Power en Pygame, que representa un objeto de poder (en este caso, una estrella).

- La clase Power hereda de pygame.sprite.Sprite, lo que la hace compatible con el sistema de sprites de Pygame.
- Se inicializan los atributos del objeto de poder, como la imagen, el rectángulo, la máscara de colisión, el carril y la velocidad.
- La función aumentar_velocidad incrementa la velocidad del objeto de poder en intervalos de tiempo específicos.
- La función move desplaza el objeto de poder hacia abajo en la pantalla a la velocidad actual. Si el objeto sale de la pantalla, se elimina (kill).

```python
import pygame
import pygame.mask
from load_sprites import *
from pygame.sprite import *
from settings import *

pygame.init()
#Se crea una instancia de la clase Settings
settings = Settings()
#Se crea la clase Power que hereda de Sprite
class Power(pygame.sprite.Sprite):
    def __init__(self,lane):
        super().__init__()
        self.image = power
        self.lane = lane
        self.rect = self.image.get_rect()
        self.rect.x = self.lane #Posición en x
        self.rect.y = settings.power_pos_y #Posición en y
        self.mask = pygame.mask.from_surface(self.image) #Máscara de colisión
        self.power_speed = settings.car_speed #Velocidad de la estrella

    def aumentar_velocidad(self):
#Método que aumenta la velocidad de la estrella
        from Juego import tiempo, aumento_vel
#Importar variables de Juego.py
        if aumento_vel <= 10:
#Si la velocidad de la estrella es menor o igual a 10
            if tiempo % 700 == 0: #Si el tiempo es múltiplo de 700
                self.power_speed += 0.5 #Aumentar la velocidad de la estrella
                print("Velocidad de la estrella: ", self.power_speed)
#Imprimir la velocidad de la estrella

    def move(self): #Método que mueve la estrella
        if self.rect.x < 650: #Si la posición en x es menor a 650
            self.rect.y += self.power_speed #Mover la estrella
            self.aumentar_velocidad() #Llamar al método aumentar_velocidad
        else:
            self.kill() #Eliminar la estrella
```

Settings.py

Este código define una clase llamada Settings, que encapsula la configuración del juego.

- La clase Settings se utiliza para almacenar y organizar los diversos ajustes y valores del juego.
- Los atributos representan diferentes aspectos del juego, como dimensiones del carro, posiciones iniciales, velocidades, carriles, número de vidas y tiempos de aparición.
- Al organizar estos valores en una clase, se facilita la gestión y modificación de la configuración del juego desde otros módulos o clases.
- Esta clase es instanciada en otros archivos para acceder a los valores de configuración de manera consistente.

```python
class Settings:
    def __init__(self):
        self.car_width = 50 #ancho del carro
        self.car_height = 80 #alto del carro
        self.car_pos_x = 400 #posicion inicial del carro en x
        self.car_pos_y = 400 #posicion inicial del carro en y
        self.car_speed = 5 #velocidad de los autos
        self.enemy_pos_y = -100 #posicion inicial de los enemigos en y
        self.power_pos_y = -100 #posicion inicial de los poderes en y
        self.carril_1 = 235
        self.carril_2 = 345
        self.carril_3 = 450
        self.carril_4 = 550
        self.carril_5 = 390
        self.carril_6 = 290
        self.num_vidas = 5  #numero de vidas
        self.time_enemy = 40 #tiempo de aparicion de enemigos
        self.time_power = 300 #tiempo de aparicion de poderes
```