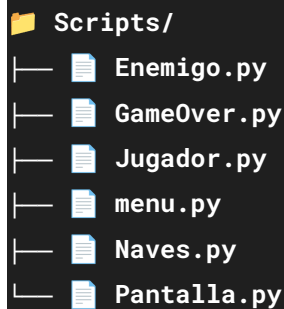


Space Truchaders

Este archivo contiene una documentación técnica de los códigos utilizados y la estructura más a detalle, hecha para entender el proyecto y para facilitar la creación de mods y mejoras

Scripts

Esta carpeta contiene los códigos que hacen la jugabilidad y logica del mismo del juego

A screenshot of a file explorer window with a dark background. It shows a folder named 'Scripts/' with a yellow folder icon. Below it, there are six Python files listed with their icons: 'Enemyo.py', 'GameOver.py', 'Jugador.py', 'menu.py', 'Naves.py', and 'Pantalla.py'. Each file has a small blue icon representing a document.

```
Scripts/  
├── Enemyo.py  
├── GameOver.py  
├── Jugador.py  
├── menu.py  
├── Naves.py  
└── Pantalla.py
```

Enemyo.py

Descripción general

Este módulo define a los enemigos del juego y su lógica de comportamiento, incluyendo:

- Movimiento y animación
- Disparo en ráfagas
- Recibir daño y explosión animada
- Drop aleatorio de power-ups al morir
- Sistema de puntaje (Marcador)

También incluye la clase Marcador, que gestiona el puntaje en pantalla.

Clases y responsabilidades

Class **Enemigo(pygame.sprite.Sprite)**

Representa un enemigo individual en el juego. Hereda de pygame.sprite.Sprite y maneja múltiples aspectos:

Atributos principales:

- **self.nave:** sprite animado aleatorio, cargado desde enemigos_T1 o enemigos_T2.
- **self.vida:** puntos de vida (400 para T1, 600 para T2).
- **self.armamento:** diccionario con tipos de disparos.
- **self.vel_x / vel_y:** dirección del movimiento (con rebote lateral).
- **self.disparo_actual:** clase de proyectil asociada al enemigo.
- **self.explosion_sprites:** animación de explosión al morir.

Métodos clave:

- **update(grupo_powerups):** mueve al enemigo, detecta límites, reproduce animación, y gestiona su muerte.
- **disparar(grupo_disparos):** dispara en ráfagas con pausas aleatorias.
- **recibir_dano(cantidad):** descuenta vida; si llega a cero, inicia la animación de explosión.
- **iniciar_explosion():** cambia el estado del enemigo a "muriendo" y reproduce sonido.
- **animacion_direccionada(direccion):** maneja la animación direccional izquierda/derecha del sprite.

Notas técnicas:

- Cada enemigo tiene una hitbox reducida (60%) respecto a su tamaño visual para mejorar la jugabilidad.
- El disparo en ráfagas se alterna entre períodos activos e inactivos.
- Puede soltar un power-up aleatorio tras morir, con probabilidad variable según el tipo.

Class **Marcador**

Clase auxiliar para mostrar el puntaje del jugador en pantalla.

Atributos:

- **puntaje:** número acumulado de puntos.
- **fuentes:** fuente utilizada para renderizar el texto.
- **color:** color del texto (blanco por defecto).

Métodos:

- **sumar_puntos(cantidad):** agrega puntos y actualiza el texto.
- **actualizar_texto():** regenera la superficie de texto con el nuevo puntaje.
- **imprimir(ventana):** dibuja el puntaje en la esquina superior izquierda.

Relación con otros módulos

- **Naves.py:** Provee diccionarios con los sprites (enemigos_T1, enemigos_T2), armamento y animaciones.
 - **Jugador.py:** Se importa `crear_powerup_aleatorio` para generar power-ups al morir un enemigo.
 - **pantalla.py:** Es donde se instancian, actualizan y dibujan los enemigos, se detectan colisiones y se llama a sus métodos.
-

Jugador . py

Descripción general

Este módulo define al jugador, sus controles, animaciones, disparos, y el sistema de power-ups. Contiene también la lógica visual para mostrar la vida y el escudo del jugador en pantalla.

Incluye además la clase **PowerUp**, la función **crear_powerup_aleatorio**, y un diccionario con todos los tipos de power-ups del juego.

Clases y responsabilidades

Class **Jugador(pygame.sprite.Sprite)**

Representa al jugador controlado por el usuario.

Atributos clave:

- **vida, escudo, vida_maxima:** atributos protegidos con getters/setters para encapsulamiento.
- **nave:** contiene la animación de la nave seleccionada.
- **imagen:** imagen actual a mostrar del jugador (cambia con la animación y dirección).
- **disparo_actual:** objeto de disparo (instancia de clase desde armamento).
- **hitbox:** rectángulo de colisión más pequeño que el sprite visual.
- **muriendo:** controla la animación de explosión al morir.

- **tiempo_powerup_aplicado, duracion_powerup**: gestión de duración de power-ups temporales.

Métodos principales:

- **actualizar()**: actualiza posición, animaciones y duración de power-ups.
- **gestionar_teclas(...)**: controla movimiento y disparo según teclas presionadas.
- **dibujar_jugador(ventana)**: dibuja la nave en pantalla.
- **dibujar_vida(ventana)**: muestra la barra de vida y escudo.
- **cambiar_disparo(nuevo_tipo)**: cambia el tipo de disparo del jugador.
- **disparar(grupo_disparos)**: dispara si pasó suficiente tiempo desde el último disparo.
- **aplicar_powerup(powerup)**: activa el efecto del power-up (vida, escudo o tipo de disparo).
- **recibir_dano(cantidad)**: descuenta daño primero del escudo y luego de la vida.
- **iniciar_explosion()**: inicia animación de explosión.
- **animacion_direccionada(direccion)**: recorre los frames de animación de la nave según dirección.
- **establecer_nave()**: establece el sprite inicial de la nave.

Notas adicionales:

- Usa animación de sprites bidireccional (flip horizontal)
- Controla duración de power-ups con `pygame.time.get_ticks()`.

PowerUp(`pygame.sprite.Sprite`)

Representa un ítem que cae del cielo y que el jugador puede recoger.

Atributos:

- **nombre**: tipo de power-up (ej. "escudo", "plasma canon")
- **image**: imagen escalada (50x30 px)
- **rect**: posición en pantalla
- **velocidad**: caída constante
- **duracion**: si aplica, duración en milisegundos (solo para power-ups de tipo disparo)

Método:

- **update()**: mueve el power-up hacia abajo; si sale de la pantalla, se elimina.

funcion **crear_powerup_aleatorio(x, y)**

Genera un **PowerUp** aleatorio según dos categorías:

- "Disparo" → minigun, plasma canon
- "Bufo" → escudo, botiquin

El power-up se crea con su imagen correspondiente y se le asigna duración si es de tipo disparo.

Relación con otros módulos

- **Naves.py:** Provee diccionario armamento, jugador_naves y las imágenes de power-ups.
 - **pantalla.py:** Controla el jugador en tiempo real, aplica daño, muestra HUD y detecta colisiones.
 - **Enemigo.py:** Usa crear_powerup_aleatorio para dropear ítems tras la muerte del enemigo.
-

menu.py (Menú principal y flujo del juego)

Descripción general

Este módulo define la interfaz principal del juego, incluyendo:

- Menú principal con botones
- Submenús: Opciones y Selección de nave
- Flujo de ejecución general (menu → juego → game over → repetir o salir)
- Fondo animado con estrellas
- Reproducción y control de música de fondo

Clases y responsabilidades

Class Button

Clase personalizada para representar botones en pantalla. Incluye propiedades encapsuladas (@property) y cambio dinámico de texto.

Atributos:

- **rect:** área interactiva del botón.
- **text:** texto mostrado.
- **font:** fuente utilizada.
- **action:** acción a realizar al presionar.

Método:

- **draw(surface):** renderiza el botón y gestiona el efecto hover.

Class **Star**

Clase auxiliar para generar el fondo animado de estrellas en movimiento.

Atributos:

- **_x, _y**: coordenadas actuales.
- **_speed_x, _speed_y**: velocidad diagonal.
- **_length**: longitud del trazo.
- **_angle_rad**: ángulo de movimiento.

Métodos:

- **move()**: actualiza la posición.
- **draw(surface)**: dibuja una estrella como una línea diagonal.

Funciones clave

show_menu(ventana)

Muestra y controla el menú principal del juego. Maneja los siguientes elementos:

- Botones: iniciar juego, opciones, seleccionar nave, cerrar juego
- Submenús: opciones (activar/desactivar sonido) y selección de nave (roja o azul)
- Fondo con estrellas animadas
- Sonido de música de fondo
- Cambio de estados (**MENU_MAIN**, **MENU_OPTIONS**, **MENU_SHIP**)
- Cambia la variable global **SHIP_SELECTION** para pasar al juego

Se ejecuta un bucle while running que mantiene el menú visible y responde a interacciones del usuario.

load_and_run_game(ventana)

Carga y ejecuta la función de gameplay desde **Pantalla.bucle_partida(...)**, deteniendo temporalmente la música del menú.

También gestiona errores con trazas (traceback) e intenta reiniciar pygame si algo falla.

Flujo principal del juego

Al ejecutar el script directamente (**if __name__ == "__main__"**):

1. Se inicializa pygame y se crea la ventana.
2. Se ejecuta **show_menu()** hasta que el usuario elija "Comenzar Juego".
3. Llama a **load_and_run_game()** para iniciar la partida (usa **Pantalla.bucle_partida()**)
4. Al finalizar la partida, muestra el menú de Game Over (**GameOver.show_game_over_menu()**).

5. Según la elección del jugador:
- "Reintentar": vuelve al gameplay
 - "Menú Principal": regresa al menú
 - "Salir": cierra el juego

Relación con otros módulos

- **Pantalla.py**: contiene el gameplay (se invoca con **Pantalla.bucle_partida(...)**).
- **GameOver.py**: contiene la pantalla de "Game Over".
- **Naves.py**: usado indirectamente para obtener naves según la variable **SHIP_SELECTION**.

pantalla.py (Gameplay principal)

Descripción general

Este módulo controla la partida completa:

- Movimiento y control del jugador
- Generación y gestión de enemigos
- Disparos (jugador/enemigos)
- Power-ups y efectos
- Sistema de puntuación
- Explosiones y animaciones
- Secuencia Game Over con créditos

Función principal

bucle_partida(ventana, nave_seleccionada)

Encapsula toda la lógica de gameplay en un bucle que:

- Inicia todos los recursos necesarios (sprites, sonidos, imágenes).
- Instancia al jugador con la nave seleccionada (**jugador_naves[nave_seleccionada]**).
- Controla eventos de teclado para el movimiento y disparo.
- Genera enemigos cada cierto intervalo aleatorio.
- Gestiona colisiones entre disparos y enemigos, así como power-ups.
- Desencadena la secuencia de Game Over al morir el jugador.

Al terminar, devuelve el control al menú principal.

Elementos destacados

Jugador

- Controlado por **WASD** y **ESPACIO**.
- Tiene vida, escudo, animación de explosión, y distintos tipos de disparo (power-up).
- Puede recibir daño o recoger ítems.

Enemigos

- Se generan en oleadas aleatorias.
- Tienen **vida**, **disparo**, **explosión**, y al morir pueden soltar un **power-up**.

Disparos

- Disparos del jugador y enemigos se agrupan en **grupo_disparos** y **grupo_disparos_enemigos**.
- Se detectan colisiones con **collide_hitboxes()** usando hitboxes personalizadas.

Power-ups

- Tipos: "**botiquin**" (**vida**), "**escudo**", o mejoras de disparo ("**minigun**", "**plasma canon**").
- Se generan al morir enemigos y pueden ser recogidos por el jugador.

Marcador

- Se actualiza con cada enemigo derrotado usando **marcador.sumar_puntos(...)**.
- Se muestra en pantalla con **marcador.imprimir(...)**.

Game Over y Créditos

- Al morir el jugador, se muestra una secuencia:
 1. Imagen de "Game Over"
 2. Imagen de Sega + sonido
 3. Imagen de "Demanda" + sonido
 4. Créditos animados en distintos idiomas
 5. Sonido final de créditos
- Se termina automáticamente después de 12 segundos.

Función auxiliar:

- **generar_credits(...)**: construye los créditos con traducciones y nombres humorísticos en varios idiomas.

Diccionarios auxiliares

- **roles**: lista de roles involucrados en el desarrollo.
- **idiomas**: traducciones de los roles (español, inglés, portugués, francés, italiano).
- **nombres_divertidos**: nombres ficticios para los créditos, muchos con humor y juegos de palabras.

Relación con otros módulos

- **Jugador.py**: contiene la clase Jugador y la lógica de power-ups.
 - **Enemigo.py**: clase Enemigo, Marcador, y sus interacciones.
 - **Naves.py**: provee los datos de animación y tipos de naves disponibles.
 - **menu.py**: invoca **bucle_partida()** con la nave seleccionada.
-

GameOver.py (Pantalla de Game Over)

Descripción general

Este módulo define la pantalla de Game Over, que se muestra tras perder una partida.

Incluye:

- Fondo gráfico personalizado
- Música de Game Over
- Título animado ("GAME OVER") con efecto 3D simulado
- Botones interactivos para:
 - Reintentar el juego
 - Volver al menú principal
 - Salir del juego

Clases y responsabilidades

class **Button**

Clase reutilizable para representar botones visuales con interacciones y texto dinámico. Se encapsulan atributos usando @property y setters.

Atributos:

- **rect**: rectángulo de colisión e interacción.
- **text**: texto mostrado en el botón.
- **font**: fuente utilizada para renderizar el texto.
- **action**: string que identifica la acción asociada al botón.

Métodos:

- **draw(surface, mouse_pos):** dibuja el botón, cambiando de color si el mouse está encima (efecto hover).

Funciones clave

show_game_over_menu(ventana)

Función principal del módulo. Muestra una interfaz de Game Over al finalizar la partida.

Incluye:

- Reproducción de música (**GameOverMusic.mp3**)
- Carga de fondo gráfico (**Fondo_GameOver.jpeg**)
- Título con efecto de sombra (profundidad 3D simulada)
- Renderizado de tres botones con acciones asociadas

Devuelve un string indicando la acción elegida:

- "retry": volver a jugar
- "main_menu": regresar al menú principal
- "quit": salir del juego

Se ejecuta un bucle while running que mantiene visible la pantalla hasta que se elija una opción.

Relación con otros módulos

- **menu.py:** este módulo invoca show_game_over_menu() después de cada partida
- **Pantalla.py:** módulo que ejecuta la lógica de la partida (gameplay). Al finalizar la vida del jugador, se vuelve aquí.

Naves.py (Definición de naves, disparos, explosiones y recursos visuales)

Descripción general

Este módulo contiene todos los recursos gráficos y sonoros relacionados con:

- Naves del jugador y enemigos (sprites animados)
- Armas y disparos (sprites + sonidos)
- Explosiones
- Power-ups visuales

También proporciona las clases Disparo y Nave, así como los diccionarios necesarios para instanciar naves y proyectiles en el juego.

Clases y responsabilidades

`class Disparo`

Clase que representa un disparo en pantalla. Hereda de `pygame.sprite.Sprite` y tiene soporte para animación, movimiento y colisiones mediante hitboxes personalizadas.

Atributos:

- **daño**: daño que inflige el disparo.
- **velocidad**: velocidad de desplazamiento (negativa para el jugador, positiva para enemigos).
- **sprites**: lista de sprites que componen la animación del disparo.
- **image, rect**: imagen actual y su posición.
- **hitbox**: rectángulo de colisión ajustado.
- **tiempo_entre_disparos**: usado por quien dispara para controlar cadencia.

Métodos:

- **get_hitbox()**: devuelve el rectángulo de colisión ajustado.
- **update()**: mueve el disparo, gestiona animación y elimina si sale de pantalla.

`class Nave`

Contenedor simple para animaciones de naves. Se usa tanto para el jugador como para enemigos.

Atributos:

- **animacion**: lista de sprites de animación.
- **tiempo_cambio**: tiempo entre frames (100 ms por defecto).
- **frame_actual, last_update**: usados para controlar la animación.

Funciones clave

CrearSetDeSprites(ruta_nave: str)

Carga todos los archivos de imagen desde una carpeta y los devuelve como lista de sprites (Surface).

Se utiliza para cargar animaciones de naves, disparos y explosiones.

Recursos definidos

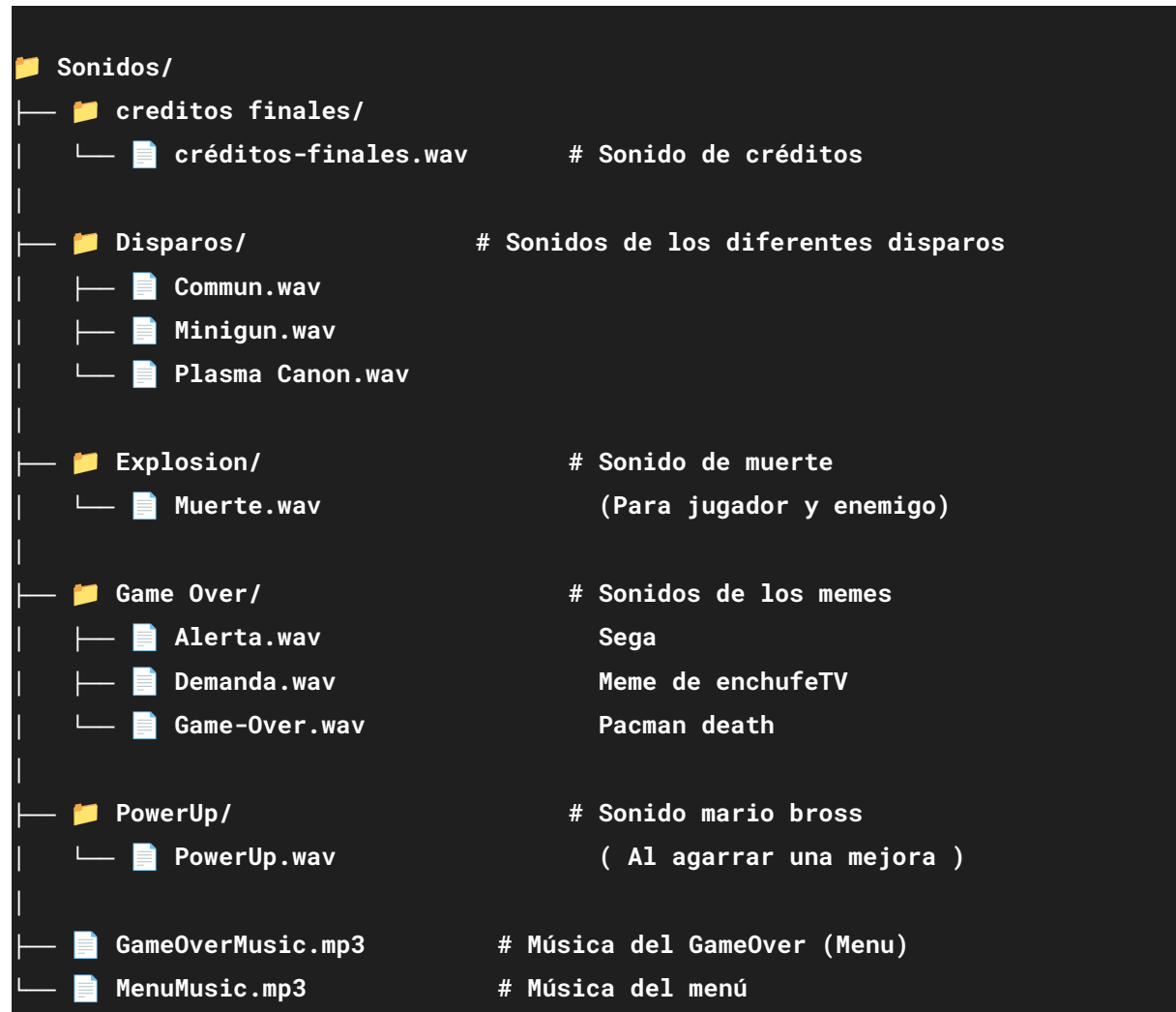
- **Fuente:**
 - **fuelle:** Fuente personalizada (**Fuelle.ttf**) para textos en pantalla.
- **Sprites de naves del jugador:**
 - **jugador_naves:** Diccionario con las variantes "**azul**" y "**roja**" (instancias de Nave)
- **Sprites de enemigos:**
 - **enemigos_T1:** Enemigos Tier I por color (azul, roja, verde)
 - **enemigos_T2:** Enemigos Tier II por color (azul, roja, verde)
- **Explosiones:**
 - **explosion_sprites:** lista de sprites de animación de explosión.
 - **explosion_sonido:** sonido reproducido al morir una nave.
- **Disparos y armamento del jugador:**
 - **armamento:** diccionario de funciones lambda que crean instancias de Disparo según tipo:
 - "**comun**"
 - "**minigun**"
 - "**plasma canon**"
- **Disparos enemigos:**
 - **armamento_enemigo:** mismo formato que armamento, pero con velocidad positiva y disparo invertido verticalmente.
- **Power-ups visuales:**
 - **power_ups:** imágenes de power-ups disponibles:
 - "**minigun**"
 - "**plasma canon**"
 - "**escudo**"
 - "**botiquin**"

Relación con otros módulos

- **Jugador.py:** usa **jugador_naves**, **armamento** y **power_ups** para configurar al jugador.
 - **Enemigo.py:** usa **enemigos_T1**, **enemigos_T2**, y **armamento_enemigo** para crear enemigos y sus disparos.
 - **Pantalla.py:** accede a **explosion_sprites**, **explosion_sonido** y **fuentes** para dibujar y animar objetos.
-

Sonidos

Esta carpeta contiene los efectos de sonido utilizados en el juego



Sprites(Assets / imagenes)

```
└─ Sprints/
└─ └─ Backgrounds/
    └─ └─ Fondo_GameOver.jpeg    # Fondo del GameOver (Menu)
    └─ Credits finales/          # Frames de la animación de los créditos
    └─ Disparos/                  # Animaciones de los disparos
    └─ └─ Comun/                  # 3 frames de animación
    └─ └─ Minigun/                # 3 frames de animación
    └─ └─ Plasma_Canon/           # 3 frames de animación
    └─ Enemigo/
    └─ └─ Tier_I/
    └─ └─ └─ Azul/                 # 3 animaciones (static, left, right)
    └─ └─ └─ Rojo/                 # 3 animaciones
    └─ └─ └─ Verde/                # 3 animaciones
    └─ └─ Tier_II/
    └─ └─ └─ Azul/                 # 3 animaciones
    └─ └─ └─ Rojo/                 # 3 animaciones
    └─ └─ └─ Verde/                # 3 animaciones
    └─ Explosion/                 # Animación de muerte(9 Frames)
    └─ Fondos/
    └─ └─ PixelBackgroundSeamless.png    # (Fondo del gameplay)
    └─ Fonts/                     # Fuentes
    └─ Fuente/
    └─ Jugador/                   # Animaciones del jugador
    └─ └─ Nave roja/              # 3 animaciones
    └─ └─ Nave azul/             # 3 animaciones
    └─ Pantalla/                  # Memes de GameOver
    └─ └─ Demanda.png
    └─ └─ Game-Over.png
    └─ └─ Sega.png
    └─ PixelSpaceRage/            # Para futuras implementaciones de
    └─                               asteroides
    └─ PowerUps/                  # Imágenes de las mejoras
```