



**pygame**



# "La Amenaza Fantasma"

En el marco del trayecto formativo de la **Tecnicatura Superior en Programación de la Universidad Tecnológica Nacional, Regional San Rafael, Mendoza**, nace este proyecto que busca ir más allá de la mera escritura de código: "**Amenaza Fantasma**". Este videojuego, un tributo interactivo a la icónica saga de "Star Wars", fue concebido no solo como un simple **ejercicio de programación**, este proyecto fue creado como un **recurso de aprendizaje integral inspirar para inspirar a los estudiantes a sumergirse en la creación de videojuegos**.

El proyecto "**Amenaza Fantasma**" utiliza un enfoque de **programación orientada a objetos (POO)**. Este modelo de programación es fundamental para el desarrollo de proyectos complejos, ya que organiza el código alrededor de **objetos** en lugar de solo **funciones y lógica secuencial**.

La **POO** se basa en el principio de que los **programas son una colección de objetos** que **interactúan** entre sí. Estos objetos son creados a partir de **clases**, que funcionan como plantillas o planos. Cada **clase define un tipo de objeto**, especificando sus **atributos** (*los datos que lo describen*) y sus **métodos**\*(*las acciones que puede realizar*).

En este proyecto aplicamos **Modularidad** para dividir el código en **unidades lógicas y reutilizables**, lo que facilita el desarrollo y el mantenimiento.

**Encapsulamiento** para agrupar los **datos** y las **funciones** que operan sobre esos datos dentro de un único objeto, lo que mejora la organización y la seguridad del código.

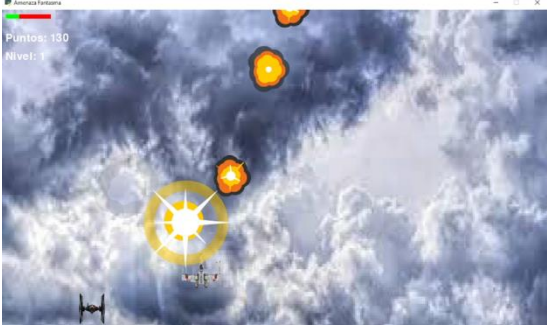

**Los invito a explorar el código, a modificarlo y a extenderlo**. Este proyecto es la semilla; les propongo que la hagan crecer. Que los conceptos y las habilidades adquiridas aquí sirvan como cimiento para construir nuevas aventuras, explorar géneros distintos y, en definitiva, continuar el viaje apasionante del desarrollo de videojuegos.

Que la Fuerza los acompañe en sus futuros proyectos.

## ESTRUCTURA DEL PROYECTO

### 📁 Organización del Código

El proyecto está estructurado y organizado en módulos (*archivos*), lo que facilita el desarrollo y el mantenimiento.

- ✓ **main.py**: El punto de entrada y el principal del juego. Aquí se inicializan todos los componentes, se gestiona la lógica central, se manejan los eventos y se actualiza la pantalla.  bucle
- ✓ **personaje.py**: Contiene las definiciones de las clases clave del juego: **Personaje** (la nave del jugador), **Enemigo** (las naves enemigas) y **Laser**. Esto sigue el principio de la programación orientada a objetos (**POO**), donde cada clase es una plantilla para un tipo de objeto.
- ✓ **explosion.py**: Define la clase **Explosion**, que maneja la animación de la explosión cuando un enemigo es destruido. 
- ✓ **constantes.py**: Almacena valores constantes como las dimensiones de la pantalla y la ruta a los activos. Usar un archivo de constantes como una buena práctica para que el código sea más legible y fácil de modificar.

### ⚙️ Uso de Clases, Funciones y Constructores

#### Clases y Objetos

Las clases son la base de la **POO** en tu juego. Cada clase es una plantilla que define un tipo de objeto, con sus propios **atributos** (*datos*) y **métodos** (*comportamientos*).

- ✓ **Personaje**:

❖ **Constructor (\_\_init\_\_):** Inicializa un objeto **Personaje**. Carga la imagen de la nave (Speeder.png), la escala y crea un objeto **Rect** para manejar su posición y detectar colisiones. También inicializa una lista vacía de láseres (self.lasers) y la energía del personaje.

❖ **Métodos:**

- **Mover(dx, dy):** Actualiza la posición de la nave según el movimiento del jugador.
- **lanzar\_laser():** Crea una nueva instancia de la clase **Laser** y la agrega a la lista self.lasers.
- **recibir\_dano ():** Reduce la energía de la nave.
- **dibujar():** Se encarga de dibujar la nave, su barra de energía y todos los láseres activos en la pantalla.

✓ **Enemigo:**

❖ **Constructor (\_\_init\_\_):** Carga la imagen del enemigo (enemigo1.png) y define su posición inicial.

❖ **Métodos:** mover() (movimiento vertical) y dibujar() (dibujar en la pantalla).

✓ **Laser:**

❖ **Constructor (\_\_init\_\_):** Carga la imagen del láser y define su posición.

❖ **Métodos:** mover() (movimiento vertical) y dibujar().

✓ **Explosion:**

❖ **Constructor (\_\_init\_\_):** Carga una secuencia de imágenes para la animación.

❖ **Métodos:** actualizar() (avanza la animación fotograma a fotograma) y dibujar().

## Funciones

Las funciones son bloques de código reutilizable. En el juego, la función `mostrar_imagen_inicial()` en `main.py` cumple este propósito.

- ✓ **mostrar\_imagen\_inicial(screen, imagen\_path, duracion):** Muestra la pantalla de introducción con un efecto de desvanecimiento gradual. Calcula la transparencia (alpha) en función del tiempo transcurrido, creando una transición suave.

## Bucle Principal y Bucle de Eventos

El juego se ejecuta dentro de un bucle infinito en la función `main()`.

- ❖ **Bucle de Eventos** (`for event in pygame.event.get()`): Es el mecanismo de Pygame para manejar la interactividad. Captura eventos como clics del ratón, pulsaciones de teclas y el cierre de la ventana.
- ❖ **Bucle Principal** (`while running`):
  - Maneja las **entradas del usuario**: Detecta si se presionan las teclas de flechas o la barra espaciadora y llama a los métodos correspondientes del objeto **Personaje**.
  - Actualiza el **estado del juego**: Mueve los enemigos y los láseres, verifica las colisiones entre ellos y actualiza las animaciones de explosión.
  - **Renderiza (dibuja) la pantalla**: Dibuja el fondo, el personaje, los enemigos y las explosiones en su nueva posición. Finalmente, `pygame.display.flip()` actualiza la pantalla para que el jugador vea los cambios.

## Interacción entre Módulos

Los diferentes archivos se comunican entre sí mediante la **importación**, `main.py` importa las clases **Personaje**, **Enemigo** y **Explosion** de `personaje.py` y `explosion.py`. Esto permite crear instancias de estas clases y usarlas para construir el juego. Las **tuplas** se usan para representar las coordenadas, como `(x, y)` para la posición de los objetos y los colores, como `(255, 0, 0)` para el color rojo de la barra de energía.