Especificaciones del Programa: Untoter

Autor del proyecto:

Santiago Castañeda Pérez

22 de mayo de 2025

1. Requerimientos del Sistema

- El sistema debe estar implementado en Python.
- Debe utilizar la librería **pygame** para gráficos y sonido.
- El sistema debe contener: clases, métodos, objetos, herencia, modularidad, polimorfismo y encapsulamiento.
- El juego debe ser ejecutable en Windows.
- El usuario debe poder controlar un personaje, disparar, y enfrentarse a enemigos.
- Debe haber un sistema de mejoras y niveles.
- Debe existir una pantalla de menú, controles y game over.

2. Temática o Contexto del Desarrollo

Untoter es un videojuego de acción en 2D donde el jugador controla a un personaje que debe sobrevivir a oleadas de enemigos y jefes, mejorando sus habilidades a medida que avanza. El objetivo es alcanzar la mayor puntuación posible antes de ser derrotado.

3. Integrantes del Grupo

- Santiago Castañeda Pérez – santiago.castaneda1@utp.edu.co

4. Flujo de Trabajo

1. El usuario inicia el juego desde el menú principal.
2. Puede consultar los controles o iniciar la partida.
3. Durante la partida, el jugador se mueve y dispara a los enemigos.
4. Al derrotar enemigos, aumenta el puntaje y puede recibir mejoras.

5. Cada ciertos niveles aparece un jefe.
6. Si la vida del jugador llega a cero, aparece la pantalla de Game Over.
7. El usuario puede reiniciar.

5. Librerías Implementadas

- **pygame**: Para gráficos, sonido y manejo de eventos.
- **random**: Para generación de enemigos y mejoras aleatorias.

6. Tarjetas CRC

Clase	**Responsabilidades**	**Colaboradores**
Game	Controlar el flujo principal del juego, gestionar niveles, puntaje, mejoras, enemigos y jugador.	Player, Enemy, Boss, Upgrade
Player	Controlar el movimiento, disparo, salud y animaciones del jugador.	Projectile, Game
Enemy	Controlar el comportamiento, movimiento y ataque de los enemigos.	Player, Game
Boss	Comportamiento especial de jefe, mayor vida y daño.	Player, Game
Upgrade	Definir y aplicar mejoras al jugador.	Player, Game
Projectile	Representar los disparos del jugador.	Player, Enemy

Clase	**Responsabilidades (métodos)**	**Colaboradores**
Game	`__init__()`, `get_instance()`, `run()`, `show_upgrade_screen()`, `spawn_enemies()`, `game_over_screen()`, `draw_text()`, `draw_health_bar()`, `pause_screen()`, `increase_difficulty()`, `spawn_boss()`, `apply_upgrade()`, `reset_game()`, `draw_enemy_health_bar()`	Player, Enemy, Boss, Upgrade

Player	`__init__()`, `shoot()`, `update()`, `take_damage()`, `load_images()`, `load_shoot_images()`, `animate()`	Projectile, Game
Enemy	`__init__()`, `load_images()`, `load_attack_images()`, `update()`, `animate()`, `attack()`, `take_damage()`	Player, Game
Boss	`__init__()`, `load_images()`, `load_attack_images()`	Player, Game
Upgrade	`__init__()`, `get_random_upgrades()`	Player, Game
Projectile	`__init__()`, `update()`	Player, Enemy

7. Diagrama de Clases

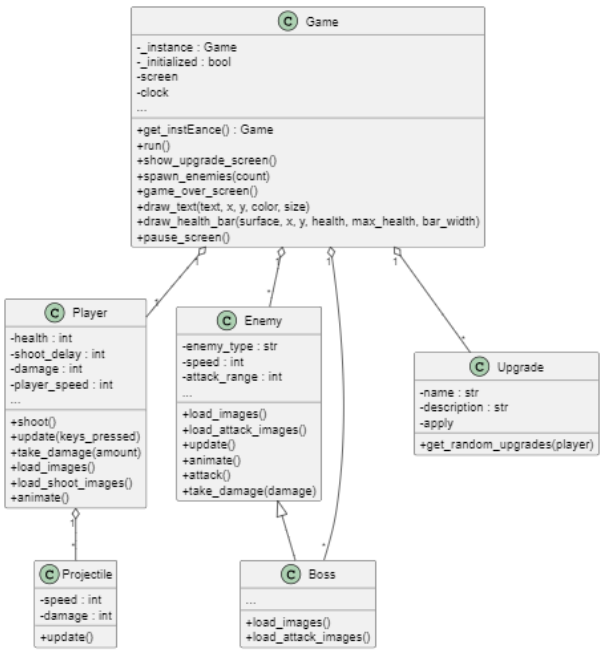

Figura 1: Diagrama de clases del videojuego UML

8. Casos de Uso

Nombre	Iniciar Juego
Creado por	Santiago Castañeda Pérez
Fecha de creación	21/05/2025
Actores	Jugador
Descripción	El usuario selecciona "Jugar."[en] el menú principal para comenzar una nueva partida.
Disparador	El usuario selecciona "Jugar."[en] el menú principal.
Pre-condiciones	El juego debe estar en la pantalla del menú principal.
Post-condiciones	Se inicia una nueva partida y el jugador controla al personaje en el escenario inicial.
Flujo normal	1. El jugador inicia el juego y se muestra el menú principal. 2. El jugador selecciona la opción "Jugar". 3. El sistema inicializa los parámetros del juego y muestra la pantalla de juego. 4. El jugador comienza a jugar.
Flujo alternativo	1a. El jugador decide no seleccionar "Jugarz elige otra opción del menú. 2a. El sistema muestra la pantalla correspondiente a la opción seleccionada.

Cuadro 3: Caso de Uso: Iniciar Juego

Nombre	Moverse y Disparar
Creado por	Santiago Castañeda Pérez
Fecha de creación	21/05/2025
Actores	Jugador
Descripción	El usuario controla al personaje con las teclas WASD y dispara con el mouse.
Disparador	El usuario presiona las teclas de movimiento o el botón del mouse.
Pre-condiciones	El juego debe estar en curso y el jugador debe tener control del personaje.
Post-condiciones	El personaje se mueve y dispara según las acciones del usuario.
Flujo normal	1. El jugador presiona WASD para moverse. 2. El jugador hace clic con el mouse para disparar. 3. El sistema actualiza la posición y dispara proyectiles.
Flujo alternativo	1a. El jugador no realiza ninguna acción y el personaje permanece quieto.

Cuadro 4: Caso de Uso: Moverse y Disparar

Nombre	Seleccionar Mejora
Creado por	Santiago Castañeda Pérez
Fecha de creación	21/05/2025
Actores	Jugador
Descripción	Al subir de nivel, el usuario elige una mejora entre varias opciones que afectan las habilidades del personaje.
Disparador	El jugador sube de nivel.
Pre-condiciones	El jugador debe haber alcanzado la experiencia suficiente para subir de nivel.
Post-condiciones	Se aplica la mejora seleccionada al personaje.
Flujo normal	1. El jugador sube de nivel. 2. El sistema muestra la pantalla de selección de mejoras. 3. El jugador selecciona una mejora. 4. El sistema aplica la mejora al personaje.
Flujo alternativo	1a. El jugador no selecciona ninguna mejora y el juego permanece en pausa hasta que lo haga.

Cuadro 5: Caso de Uso: Seleccionar Mejora

Nombre	Pausar Juego
Creado por	Santiago Castañeda Pérez
Fecha de creación	21/05/2025
Actores	Jugador
Descripción	El usuario puede pausar la partida presionando la tecla P y acceder a opciones de reinicio o salida.
Disparador	El usuario presiona la tecla P durante la partida.
Pre-condiciones	El juego debe estar en curso.
Post-condiciones	El juego se detiene temporalmente y se muestran las opciones de pausa.
Flujo normal	1. El jugador presiona P. 2. El sistema detiene el juego y muestra el menú de pausa. 3. El jugador puede reanudar, reiniciar o salir.
Flujo alternativo	1a. El jugador no selecciona ninguna opción y el juego permanece en pausa.

Cuadro 6: Caso de Uso: Pausar Juego

Nombre	Game Over
Creado por	Santiago Castañeda Pérez
Fecha de creación	21/05/2025
Actores	Jugador
Descripción	Al perder toda la vida, el usuario puede reiniciar la partida o volver al menú principal.
Disparador	El jugador pierde toda la vida.
Pre-condiciones	El jugador debe estar en una partida activa.
Post-condiciones	Se muestra la pantalla de Game Over y el jugador puede elegir reiniciar o volver al menú.
Flujo normal	1. El jugador pierde toda la vida. 2. El sistema muestra la pantalla de Game Over. 3. El jugador selecciona reiniciar o volver al menú.
Flujo alternativo	1a. El jugador no selecciona ninguna opción y permanece en la pantalla de Game Over.

Cuadro 7: Caso de Uso: Game Over

Nombre	Consultar Controles
Creado por	Santiago Castañeda Pérez
Fecha de creación	21/05/2025
Actores	Jugador
Descripción	El usuario puede ver los controles del juego desde el menú principal.
Disparador	El usuario selecciona la opción Çontroles.[en] el menú principal.
Pre-condiciones	El juego debe estar en la pantalla del menú principal.
Post-condiciones	Se muestra la pantalla de controles al usuario.
Flujo normal	1. El jugador selecciona Çontroles.[en] el menú principal. 2. El sistema muestra la pantalla de controles. 3. El jugador puede volver al menú principal.
Flujo alternativo	1a. El jugador no selecciona ninguna opción y permanece en la pantalla de controles.

Cuadro 8: Caso de Uso: Consultar Controles

Nombre	Salir del Juego
Creado por	Santiago Castañeda Pérez
Fecha de creación	21/05/2025
Actores	Jugador
Descripción	El usuario puede salir del juego desde el menú principal o durante la partida.
Disparador	El usuario selecciona la opción "Salir.[en] el menú principal o durante la partida.
Pre-condiciones	El juego debe estar en ejecución.
Post-condiciones	El juego se cierra correctamente.
Flujo normal	1. El jugador selecciona "Salir.[en] el menú principal o durante la partida. 2. El sistema cierra el juego.
Flujo alternativo	1a. El jugador cancela la acción y el juego continúa en ejecución.

Cuadro 9: Caso de Uso: Salir del Juego

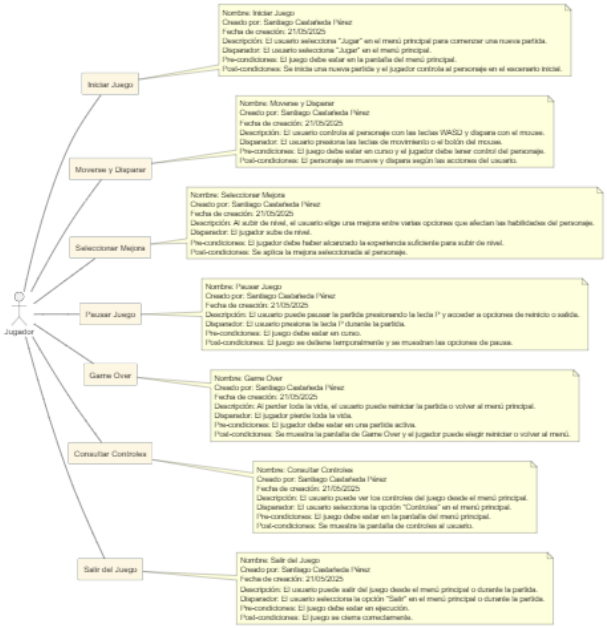

Figura 2: Diagrama casos de uso UML

9. Patrón de Diseño: Singleton

En el desarrollo de este videojuego se implementó el **patrón de diseño Singleton** en la clase `Game`. Este patrón garantiza que sólo exista una única instancia de la clase principal del juego durante toda la ejecución del programa, evitando así problemas de duplicidad y facilitando el acceso global a la instancia del juego desde cualquier parte del código.

¿Cómo se implementó?

La clase `Game` define un atributo de clase privado `_instance` que almacena la única instancia permitida. El método especial `__new__` se sobreescribe para controlar la creación de la instancia, y un método estático `get_instance()` permite obtener siempre la misma instancia:

```
class Game:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(Game, cls).__new__(cls)
        return cls._instance

    @staticmethod
    def get_instance():
        if Game._instance is None:
            Game()
        return Game._instance
```

¿Cómo se utiliza?

En el resto del código, cada vez que se necesita acceder a la instancia del juego, se utiliza el método `Game.get_instance()` en lugar de crear una nueva instancia con `Game()`. Por ejemplo, en el menú principal:

```
if options[selected_index] == "Jugar":
    game = Game.get_instance()
    game.run()
```

De esta forma, se asegura que toda la lógica y el estado del juego se gestionen desde una única instancia centralizada.

10. Anexos

- Código fuente disponible en la carpeta del proyecto.
- Recursos gráficos y de sonido en las carpetas `assets/` y `sounds/`.