Manual Técnico de Space Invaders EPN

1. Introducción

Space Invaders EPN es una implementación del clásico juego arcade *Space Invaders*, desarrollado en C++ utilizando la biblioteca SFML (Simple and Fast Multimedia Library). El juego incluye un menú principal, mecánicas de juego (jugador, enemigos, balas, muros), un HUD para mostrar vidas y puntaje, y pantallas de victoria y derrota. El proyecto está alojado en el repositorio `git@github.com:Nicolax-b/SpaceInvaders.git`.

Este manual técnico describe la arquitectura del juego, sus componentes principales, la estructura de los archivos, las dependencias, y las instrucciones para compilar y ejecutar el proyecto. Está dirigido a desarrolladores que deseen entender el funcionamiento del juego, realizar modificaciones o extender sus funcionalidades.

2. Arquitectura del Sistema

El juego sigue una arquitectura orientada a objetos, con clases que representan las entidades principales del juego (jugador, enemigos, balas, muros, HUD, menús). La lógica principal se encuentra en `main2.cpp`, que maneja el flujo del juego, incluyendo el menú inicial, la partida, y las pantallas de victoria/derrota. SFML se utiliza para la gestión de gráficos, audio, entrada de usuario (teclado y joystick), y la ventana de renderizado.

2.1 Componentes Principales

- **Jugador (`Player`)**: Representa la nave del jugador, que se mueve horizontalmente y dispara balas. Gestiona vidas y colisiones con balas enemigas.
- **Enemigos (`Enemie`)**: Una matriz de enemigos que se mueven en patrones predefinidos, disparan balas y cambian de dirección al alcanzar los bordes de la pantalla.
- **Balas (`Bullet`)**: Proyectiles disparados por el jugador y los enemigos. Incluyen colisiones con enemigos, muros y el jugador.
- **Muros (`Muro`)**: Barreras protectoras que se degradan al recibir impactos de balas.
- **HUD (`HUD`)**: Muestra el puntaje y las vidas del jugador en pantalla.
- **Menú (`Menu`)**: Pantalla inicial con opciones "Play" y "Exit".
- **Pantallas de Victoria/Derrota (`Win`, `Lose`)**: Mostradas al ganar (eliminar todos los enemigos) o perder (sin vidas o enemigos alcanzan el límite inferior).
- **Entrada de Usuario (`InputHandler`)**: Gestiona entradas de teclado y joystick para navegar por menús y controlar al jugador.

2.2 Flujo del Juego

1. **Menú Principal**: El juego inicia con un menú (`Menu`) que permite seleccionar "Play" o "Exit" usando teclado o joystick.
2. **Partida**: Al seleccionar "Play", se inicia una ventana en pantalla completa donde el jugador controla una nave, dispara a enemigos, y evita sus balas. Los muros protegen al jugador, pero se degradan con los impactos.
3. **Condiciones de Fin**:
    - **Victoria**: Se eliminan todos los enemigos (`cantEnemies == 0`).

    - **Derrota**: El jugador pierde todas sus vidas (`!player.Vivo()`) o un enemigo alcanza el límite inferior de la pantalla (`y >= screenHeight * 0.9f`).
4. **Pantallas Finales**: Se muestra la pantalla de victoria (`Win`) o derrota (`Lose`), con opciones para reiniciar o salir.

3. Estructura de Archivos

El proyecto está organizado en archivos de cabecera (`.hpp`) y de implementación (`.cpp`). A continuación, se detalla cada archivo relevante:

- **Bala.hpp, Bala.cpp**: Define la clase `Bullet` para las balas del jugador y enemigos, con métodos para actualizar posición (`Update`), obtener posición (`Pos`), y dibujar (`draw`).
- **Enemie.hpp, Enemie.cpp**: Define la clase `Enemie` para los enemigos, con lógica para movimiento, cambio de dirección (`ChangeDir`), disparo (`ActivarDisparo`, `UpdateDisparo`), y sonidos.
- **HUD.hpp, HUD.cpp**: Define la clase `HUD` para mostrar vidas y puntaje en pantalla.
- **InputHandler.hpp**: Define la enumeración `InputResult` y la función `manejarInputMenu` para procesar entradas de teclado y joystick en los menús.
- **Jugador.hpp, Jugador.cpp**: Define la clase `Player` para la nave del jugador, con métodos para movimiento (`Update`), disparo (`Shoot`), y gestión de vidas (`QuitarVida`, `Vivo`).
- **Lose.hpp, Lose.cpp**: Define la clase `Lose` para la pantalla de derrota, con opciones "Play Again" y "Exit".
- **Menu.hpp, Menu.cpp**: Define la clase `Menu` para el menú principal, con navegación y música de fondo.
- **Muro.hpp, Muro.cpp**: Define la clase `Muro` para las barreras protectoras, que se degradan con colisiones (`Colision`).
- **Win.hpp, Win.cpp**: Define la clase `Win` para la pantalla de victoria, con opciones "Play Again" y "Exit".
- **main2.cpp**: Archivo principal que integra todos los componentes, gestiona el flujo del juego, y actualiza/renderiza las entidades.
- **versionDemo.cpp, Demostracion.cpp**: Versiones alternativas del archivo principal, probablemente para pruebas o versiones simplificadas.

3.1 Recursos

- **Texturas**: `textures/spritesheetnuevo.png` (sprites del jugador, enemigos, balas) y `textures/spritesheet.png` (muros).
- **Fuentes**: `fonts/ARCADE_N.TTF` para textos del HUD y menús.
- **Sonidos**: Archivos `.wav` (`sounds/Bala.wav`, `sounds/shootenemy.wav`, `sounds/explosionenemy.wav`) para disparos y explosiones.
- **Música**: Archivos `.ogg` (`Music/menu.ogg`, `Music/Play1.ogg`, `Music/Play2.ogg`, `Music/Win.ogg`, `Music/game-over-39-199830.ogg`) para fondo y eventos.

4. Dependencias

El juego utiliza las siguientes bibliotecas externas:

- **SFML**: Para gráficos (`sf::Sprite`, `sf::Texture`, `sf::RenderWindow`), audio (`sf::Sound`, `sf::Music`), y entrada (`sf::Keyboard`, `sf::Joystick`).
- **C++ Standard Library**: Para estructuras de datos (`std::vector`, `std::string`), entrada/salida (`std::cout`, `std::cerr`), y funciones de tiempo (`std::time`).

4.1 Instalación de Dependencias

Para compilar el juego, asegúrate de tener SFML instalado:

- **Ubuntu/Debian**:

```
sudo apt-get install libsfml-dev
```

- **Windows**: Descarga SFML desde sfml-dev.org y configura tu entorno (por ejemplo, con Visual Studio o MinGW).
- **macOS**:

```
brew install sfml
```

Además, necesitas un compilador C++ compatible (por ejemplo, `g++`) y un sistema de construcción como `make` o `CMake`.

5. Compilación y Ejecución

5.1 Clonar el Repositorio

Clona el repositorio desde GitHub:

```
git clone git@github.com:Nicolax-b/SpaceInvaders.git
cd SpaceInvaders
```

5.2 Compilación

El proyecto no incluye un archivo `Makefile` o `CMakeLists.txt` en los archivos proporcionados, por lo que se debe compilar manualmente. Un comando de ejemplo para compilar con `g++` en Linux:

```
g++ -c main2.cpp Jugador.cpp Bala.cpp Enemie.cpp Muro.cpp HUD.cpp Menu.cpp Win.cpp
Lose.cpp -I. -I/usr/include/SFML
g++ *.o -o SpaceInvaders -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
```

Asegúrate de que los directorios `textures/`, `fonts/`, y `Music/` estén en el mismo directorio que el ejecutable, o ajusta las rutas en el código.

5.3 Ejecución

Ejecuta el juego:

```
./SpaceInvaders
```

El juego se inicia en modo pantalla completa, mostrando el menú principal. Usa las teclas de flecha o el joystick para navegar y la tecla Enter o el botón A del joystick para seleccionar.

6. Mecánicas del Juego

6.1 Controles

- **Teclado**:
  - Flecha Izquierda/Derecha: Mover la nave del jugador.
  - Espacio: Disparar una bala.
  - Flecha Arriba/Abajo: Navegar por los menús.
  - Enter: Seleccionar opción en el menú.
- **Joystick**:
  - Eje X: Mover la nave horizontalmente.
  - Botón 0 (A): Disparar.
  - Eje Y: Navegar por los menús.
  - Botón 0 (A): Seleccionar opción en el menú.

6.2 Dinámica del Juego

- **Jugador**: Comienza con 3 vidas, se mueve horizontalmente, y dispara una bala a la vez (velocidad: -15 píxeles por frame).
- **Enemigos**: Organizados en una matriz de 7 filas y 12 columnas. Se mueven lateralmente y descienden al alcanzar los bordes de la pantalla. Disparan balas aleatoriamente (velocidad: 15 píxeles por frame).
- **Muros**: Cuatro barreras protectoras, cada una con 5 segmentos que se degradan al recibir impactos (hasta 5 por segmento).
- **Puntuación**: Los enemigos otorgan puntos según su fila (100 a 700 puntos, más puntos por filas superiores).
- **Colisiones**:
  - Balas del jugador contra enemigos: Elimina al enemigo y desactiva la bala.
  - Balas enemigas contra jugador: Reduce una vida.
  - Balas (jugador o enemigas) contra muros: Degrada el segmento impactado.
- **Condiciones de Fin**:
  - Victoria: Eliminar todos los enemigos.
  - Derrota: Perder todas las vidas o un enemigo alcanza el 90% de la altura de la pantalla.

6.3 Audio

- **Sonidos**: Disparos del jugador (`Bala.wav`), disparos de enemigos (`shootenemy.wav`), explosión de enemigos (`explosionenemy.wav`).
- **Música**: Reproduce aleatoriamente `Play1.ogg` o `Play2.ogg` durante el juego, `menu.ogg` en el menú, `Win.ogg` en la pantalla de victoria, y `game-over-39-199830.ogg` en la pantalla de derrota.

7. Detalles Técnicos

7.1 Resolución y Escalado

- El juego se ejecuta en modo pantalla completa, adaptándose a la resolución del escritorio (`VideoMode::getDesktopMode()`).

- Las posiciones de los elementos (jugador, enemigos, muros) se calculan dinámicamente según `screenWidth` y `screenHeight`.
- Los sprites se escalan (por ejemplo, `sprite.setScale(3, 3)` para el jugador) para ajustarse a diferentes resoluciones.

7.2 Gestión de Sprites

- Se utiliza un solo *spritesheet* (`spritesheetnuevo.png`) para el jugador, enemigos y balas, con diferentes regiones (`IntRect`) para cada entidad.
- Los muros usan `spritesheet.png`, con regiones específicas para sus estados de degradación.
- Ejemplo de configuración de sprite para el jugador:

```
sprite.setTextureRect(IntRect(0, 118, 32, 32));
sprite.setScale(3, 3);
```

7.3 Colisiones

- Las colisiones se verifican usando `sf::IntRect` para rectángulos de colisión aproximados.
- Para balas del jugador, se considera el trayecto vertical (`getPreviousY` y `Pos().y`) para una detección más precisa.
- Ejemplo de colisión entre bala y enemigo:

```
bulletRect = IntRect(bulletPlayer.Pos().x, bulletPlayer.Pos().y, 9, 32);
enemieRect = IntRect(enemies[i][j].Pos().x, enemies[i][j].Pos().y, 32, 32);
if (enemieRect.intersects(bulletRect)) {
    enemies[i].erase(enemies[i].begin() + j);
    bulletActive = false;
}
```

7.4 Cadencia de Disparo

- Los enemigos disparan aleatoriamente cada `cadencia` frames (inicialmente 125, disminuye al eliminar enemigos).
- La función `AumentarCadencia` reduce la cadencia de los enemigos, aumentando la dificultad.

8. Problemas Conocidos y Posibles Mejoras

- **Errores Potenciales**:
  - En `main2.cpp`, el bucle anidado en `UpdateMuro` usa el mismo índice `i` para dos bucles, lo que puede causar accesos incorrectos a `posicionMuro`.
  - La inicialización de `srand(time(NULL))` se llama repetidamente en `UpdateEnemies`, lo que puede afectar la aleatoriedad. Debería llamarse una sola vez en `main`.
- **Mejoras Sugeridas**:
  - Agregar un archivo `CMakeLists.txt` o `Makefile` para facilitar la compilación.
  - Implementar una pantalla de pausa.
  - Añadir niveles de dificultad progresiva (por ejemplo, aumentando la velocidad de los enemigos).

- Optimizar la detección de colisiones usando estructuras espaciales (como un quadtree) para grandes cantidades de enemigos.
- Agregar animaciones para explosiones de enemigos.

9. Contribuciones

Para contribuir al proyecto, clona el repositorio, crea una rama para tus cambios, y envía un *pull request*:

```
git checkout -b mi-nueva-funcionalidad
git commit -am "Añadir nueva funcionalidad"
git push origin mi-nueva-funcionalidad
```

Consulta las guías de contribución en el repositorio para más detalles.

10. Referencias

- Documentación de SFML: www.sfml-dev.org
- Repositorio del proyecto: git@github.com:Nicolax-b/SpaceInvaders.git
- Inspiración en implementaciones de *Space Invaders* en C++ y otras tecnologías.

11. Conclusión

Space Invaders EPN es una implementación sólida del clásico juego arcade, con soporte para teclado y joystick, gráficos escalables, y audio inmersivo. Su diseño modular permite extensiones y mejoras, como la adición de nuevos niveles, animaciones, o integración con sistemas de puntuación en línea. Este manual proporciona una base para que los desarrolladores comprendan, mantengan y amplíen el proyecto.