Manual Técnico: Space Invaders EPN - Enfoque en Spritesheet Este manual describe los componentes clave del proyecto Space Invaders EPN, un juego desarrollado en C++ con la biblioteca SFML. El enfoque principal está en cómo se creó y seleccionó la pantalla de sprites (spritesheet) para los elementos gráficos del juego, junto con un resumen de las clases y funciones más importantes. El juego incluye un menú, un jugador, enemigos, balas, muros protectores, un HUD y pantallas de victoria/derrota.

1. Spritesheet: Creación y Selección El juego utiliza un archivo de spritesheet (spritesheet.png) como la fuente principal para los gráficos de los elementos del juego (jugador, enemigos, balas y muros). A continuación, se detalla cómo se creó y se selecciona en el código:

Creación del Spritesheet:

El archivo spritesheet.png es una imagen única que contiene todos los sprites del juego organizados en una cuadrícula. Cada sprite tiene dimensiones específicas (por ejemplo, 32x32 píxeles para enemigos, 8x8 píxeles para balas). Los sprites están dispuestos en filas y columnas, donde cada fila representa un tipo de elemento o estado (por ejemplo, enemigos de diferentes niveles, animaciones, jugador, balas, muros). La selección de sprites se realiza mediante coordenadas (IntRect) que especifican la región del spritesheet a usar para cada elemento.

Selección de Sprites:

La clase sf::Sprite de SFML se utiliza para cargar una porción específica del spritesheet mediante setTextureRect(IntRect(x, y, ancho, alto)). Las coordenadas (x, y) y dimensiones (ancho, alto) se calculan según la posición del sprite en el spritesheet. Los sprites se escalan (usando setScale) para ajustar su tamaño en la pantalla del juego.

Uso General:

El spritesheet se carga una vez en main2.cpp o versionDemo.cpp usando spritesheet.loadFromFile("spritesheet.png"). Cada clase (Player, Bullet, Enemie, Muro) recibe una referencia a esta textura y selecciona la región adecuada para su sprite.

2. Archivo: main2.cpp Propósito Archivo principal que inicializa el juego, maneja el menú y controla el bucle principal. Gestiona la carga del spritesheet y la inicialización de los elementos gráficos. Componentes Clave

Carga del Spritesheet: Texture spritesheet; if (!spritesheet.loadFromFile("spritesheet.png")) { cout << "Error al cargar la textura\n"; }

El spritesheet se carga al inicio y se pasa como referencia a las clases Player, Bullet, Enemie y Muro.

Inicialización de Enemigos: Los enemigos se organizan en una matriz de 7 filas y 12 columnas. Cada enemigo selecciona una región del spritesheet según su fila:Vector2f sectionSpritesheet; if (i == 0) { sectionSpritesheet = Vector2f(0, 0); // Primera fila del spritesheet } else if (i < 3) { sectionSpritesheet = Vector2f(0, 9 + (8 * 4 + 4)); // Segunda/tercera fila } else if (i < 5) { sectionSpritesheet = Vector2f(0, 18 + (8 * 4 + 4) * 2); // Cuarta/quinta fila } else if (i < 7) { sectionSpritesheet = Vector2f(0, 27 + (8 * 4 + 4) * 3); // Sexta/séptima fila } enemies[i][j] = Enemie(j * 30 + 24, i * 30 + 24, spritesheet, sectionSpritesheet);

Cada tipo de enemigo usa una región diferente del spritesheet, con coordenadas y calculadas para seleccionar la fila adecuada.

Funciones Principales: UpdatePlayer: Actualiza el movimiento y disparo del jugador. UpdateBulletPlayer: Maneja la lógica de colisión de la bala del jugador con enemigos. UpdateEnemies: Actualiza el movimiento de los enemigos. UpdateBulletsEnemies: Gestiona las balas enemigas y colisiones con el jugador. UpdateMuro: Maneja colisiones de balas con los muros.

3. Archivo: versionDemo.cpp Propósito Versión simplificada del juego sin menú ni música, pero con la misma lógica de spritesheet. Componentes Clave

Carga del Spritesheet: Igual que en main2.cpp, carga spritesheet.png y lo pasa a las clases. Selección de Sprites para Enemigos: Usa el mismo código que main2.cpp para inicializar enemigos con diferentes regiones del spritesheet según su fila. Diferencias: No incluye HUD ni pantallas de victoria/derrota, pero la lógica de sprites es idéntica.

4. Archivo: Jugador.hpp y Jugador.cpp Propósito Define la clase Player, que representa al jugador y su sprite en el juego. Componentes Clave

Clase Player: Selección de Sprite:sprite.setTexture(texture); sprite.setTextureRect(IntRect(0, 118, 32, 32)); sprite.setPosition(x, y); sprite.setScale(3, 3);

Selecciona un sprite de 32x32 píxeles en las coordenadas (0, 118) del spritesheet, correspondiente al gráfico del jugador. Escala el sprite por 3 para hacerlo más grande en pantalla.

Métodos: Update(): Mueve el sprite del jugador usando teclado o joystick. Shoot(): Dispara una bala (creada con su propio sprite) al presionar Espacio o el botón del joystick. Pos(): Devuelve la posición del sprite. draw(): Dibuja el sprite en la ventana.

5. Archivo: Bala.hpp y Bala.cpp Propósito Define la clase Bullet, que representa las balas del jugador y enemigos. Componentes Clave

Clase Bullet: Selección de Sprite:Bullet::Bullet(int x, int y, Texture &texture, IntRect intRect, int v) : sprite(texture, intRect) { sprite.setPosition(sf::Vector2f(static_cast(x), static_cast(y))); sprite.setScale(sf::Vector2f(3.0f, 3.0f)); vel = v; ubi = static_cast(y); }

Recibe un IntRect que especifica la región del spritesheet (por ejemplo, (13 * 8 + 16, 6 * 8 + 6, 8, 8) para la bala del jugador en main2.cpp). Escala el sprite por 3 para ajustar su tamaño.

Métodos: Update(): Mueve la bala verticalmente según su velocidad. Pos(): Devuelve la posición actual. draw(): Dibuja el sprite de la bala.

6. Archivo: Enemie.hpp y Enemie.cpp Propósito Define la clase Enemie, que representa a los enemigos y sus animaciones. Componentes Clave

Clase Enemie: Selección de Sprite:Enemie::Enemie(int x, int y, Texture &texture, Vector2f p) { point = p; sprite.setTexture(texture); sprite.setTextureRect(IntRect(point.x, point.y, 32, 32)); sprite.setPosition(x, y); sprite.setScale(2.5, 2.5); }

Usa las coordenadas point (recibidas desde main2.cpp) para seleccionar una región de 32x32 píxeles del spritesheet. Escala el sprite por 2.5.

Animación:void Enemie::Update() { if (timer >= cadencia) { sprite.move(vel, 0); state++; state %= 2; sprite.setTextureRect(IntRect(point.x + state * 33, point.y, 30, 30)); timer = 0; } timer++; }

Cambia el sprite cada cadencia frames, alternando entre dos estados de animación (desplazando point.x por 33 píxeles). Durante el disparo, usa un sprite especial:sprite.setTextureRect(IntRect(96, point.y, 30, 30));

Métodos: ChangeDir(): Invierte la dirección y baja al enemigo. ActivarDisparo(): Cambia al sprite de disparo. UpdateDisparo(): Restaura el sprite normal tras el disparo. draw(): Dibuja el sprite.

7. Archivo: Muro.hpp y Muro.cpp Propósito Define la clase Muro, que representa los muros protectores formados por múltiples sprites. Componentes Clave

Clase Muro: Selección de Sprite:Muro::Muro(int x, int y, Texture &texture) { sprites.resize(5); state.resize(5, {0, 0}); for (int i = 0; i < 5; i++) { sprites[i].setTexture(texture); sprites[i].setTextureRect(IntRect(16*8+9+state[i].first*9, 14*8+14+state[i].second*9, 8, 8)); sprites[i].setScale(4, 4); } sprites[0].setPosition(x, y); sprites[1].setPosition(x+24, y); sprites[2].setPosition(x+48, y); sprites[3].setPosition(x, y+24); sprites[4].setPosition(x+48, y+24); }

Cada muro está compuesto por 5 sprites de 8x8 píxeles, seleccionados desde las coordenadas (16*8+9, 14*8+14) del spritesheet. Los sprites cambian según el daño recibido (state[i].first o state[i].second incrementan la coordenada x o y). Escala cada sprite por 4.

Métodos: Update(): Actualiza los sprites según el estado de daño. Pos(): Devuelve las posiciones de las partes no destruidas. Colision(): Incrementa el daño de una parte del muro. draw(): Dibuja solo las partes no destruidas.

8. Archivo: HUD.hpp y HUD.cpp Propósito Define la clase HUD, que muestra las vidas y el puntaje en pantalla. Componentes Clave

Clase HUD: Sin Sprites: No usa el spritesheet, sino textos (sf::Text) con una fuente externa. Métodos: HUD(Font &font): Inicializa los textos de vidas y puntaje. updateVidas(), updateScore(): Actualizan los valores mostrados. draw(): Dibuja los textos.

9. Archivo: Menu.hpp y Menu.cpp Propósito Define la clase Menu, que muestra el menú principal con las opciones "Play" y "Exit". Componentes Clave

Clase Menu: Sin Sprites: Usa textos (sf::Text) con la fuente ARCADE_N.TTF. Métodos: Menu(float width, float height): Inicializa las opciones del menú. draw(): Dibuja los textos. MoveUp(), MoveDown(): Cambian la selección. PlayMusic(), StopMusic(): Controlan la música del menú.

10. Archivo: Lose.hpp y Lose.cpp Propósito Define la clase Lose, que muestra la pantalla de derrota. Componentes Clave

Clase Lose: Sin Sprites: Usa textos (sf::Text) con la fuente ARCADE_N.TTF para "GAME OVER" y las opciones "Play Again" y "Exit". Métodos: Lose(float width, float height): Inicializa los textos y la música. draw(): Dibuja los textos. MoveUp(), MoveDown(): Cambian la selección. PlayMusic(), StopMusic(): Controlan la música.

11. Archivo: Win.hpp y Win.cpp Propósito Define la clase Win, que muestra la pantalla de victoria. Componentes Clave

Clase Win: Sin Sprites: Usa textos (sf::Text) con la fuente ARCADE_N.TTF para "WELL DONE HERO!!!" y las opciones "Play Again" y "Exit". Métodos: Win(float width, float height): Inicializa los textos y la música. draw(): Dibuja los textos. MoveUp(), MoveDown(): Cambian la selección. PlayMusic(), StopMusic(): Controlan la música.

12. Archivo: InputHandler.hpp Propósito Maneja la entrada del usuario para los menús. Componentes Clave

Enumeración InputResult: Define acciones (Nada, Subir, Bajar, Seleccionar). Función manejarInputMenu: Procesa entrada de teclado y joystick. Sin Sprites: No interactúa con el spritesheet.

13. Archivo: Joystick.cpp Propósito Maneja la entrada del joystick y muestra su estado. Componentes Clave

Funciones: updateIdentification, updateAxes, updateButtons, updateValues para procesar y mostrar el estado del joystick. Sin Sprites: Usa textos para mostrar información, no interactúa con el spritesheet.

Notas Adicionales

Spritesheet en Detalle: El spritesheet organiza los sprites en una cuadrícula, con regiones específicas para cada elemento: Jugador: (0, 118, 32, 32). Bala del Jugador: (13*8+16, 6*8+6, 8, 8). Enemigos: Diferentes filas (por ejemplo, (0, 0), (0, 45), (0, 90), (0, 135)) según el tipo. Muros: (16*8+9, 14*8+14, 8, 8) con desplazamientos para estados de daño.

La selección precisa de sprites asegura que cada elemento tenga el gráfico correcto.

Mejoras Potenciales: Documentar las coordenadas exactas del spritesheet en un archivo de configuración para facilitar cambios. Optimizar la carga del spritesheet para reducir el uso de memoria. Corregir inconsistencias en el número de muros (3 en versionDemo.cpp, 4 en main2.cpp).