Snake Game - Final Project

Project Overview

This is a classic Snake Game implemented in C++ using SDL2. The player controls a snake that collects food and grows longer while avoiding collisions with the walls or itself. If the snake collides, the game is over, and the player's score is displayed.

Features

- **Pause functionality** to pause and resume the game.
- **High score tracking** that saves and loads the highest score across sessions.
- **Game over screen** to display the player's final score when the game ends.

Requirements

Dependencies

- **SDL2**: The game requires the SDL2 library for graphics and input handling.

Installation Instructions

SDL2 Installation

- **Windows**:
    1. Download SDL2 from the SDL2 website.
    2. Follow the installation instructions specific to your IDE or build system.
- **Linux**:

Install SDL2 using the package manager:

```
sudo apt-get install libsdl2-dev
```

- **macOS**:

Install SDL2 using Homebrew:

```
brew install sdl2
```

Build Instructions
Clone the repository:

```
git clone <repo_url>
cd <repo_name>
```

CMake: To build the project using CMake, run the following commands:

```
mkdir build
cd build
cmake ..

make
```

3- After building, run the game executable:

```
./snake_game
```

New Features Added

1. Pause Functionality

- The game can be paused and resumed by pressing the **'P'** key.
- When the game is paused, updates stop, and the game waits for user input to resume.

2. High Score

- The game tracks the **highest score** achieved. It is saved to a file (high_score.txt) and updated whenever a higher score is achieved.
- On game start, the high score is loaded from the file.
- The score is displayed in the window title during the game, and the high score is updated accordingly.

3. Game Over Screen

- When the snake collides with itself or the walls, the game ends.
- The game displays the **final score** on the screen.
- The game over message appears for 2 seconds before the game restarts.

Rubric Points Addressed

1. Loops, Functions, I/O

- **Loops**: The game utilizes loops to handle the game cycle, including input handling, updating the game state, and rendering the screen.
- **Functions**: Key game features like Render(), Update(), and input handling (HandleInput()) are implemented as separate functions.
- **I/O**: The game reads and writes the high score to/from a file (high_score.txt).
- **Code Locations**:
    - game.cpp: Main game loop and control flow.

2. Object-Oriented Programming

- The game is structured with **classes** like Game, Renderer, and Snake to organize the code logically.

- **Member functions** encapsulate behaviors such as rendering the game state (Render()), handling user input (HandleInput()), and updating the game state (Update()).
- **Code Locations**:
    - game.cpp: Main game logic (loops and state updates).
    - renderer.cpp: Rendering logic.
    - snake.cpp: Snake's movement and behavior.

3. Memory Management

- The project uses **references** and **pointers** where appropriate, especially when handling the game objects.
- **Code Locations**:
    - game.cpp: Handling the game objects (e.g., snake, food).

4. Concurrency

- The game includes basic **multithreading**: Pausing the game uses std::this_thread::sleep_for() to allow a delay when the game is paused, which simulates concurrency by halting the game's updates temporarily.
- **Code Locations**:
    - game.cpp: Handling the game loop with multithreading (std::this_thread::sleep_for).

5. Control Structures

- The game logic is structured using **if-else** statements, loops (for, while), and **switch** statements to handle user input, game states, and score updates.
- **Code Locations**:
    - game.cpp: Main game loop with input handling and score checks.

    - renderer.cpp: Rendering control structures for different game states (e.g., game over).

Known Issues

- No significant issues reported.

Conclusion

This project implements a functional Snake Game with added features such as pausing, high score tracking, and a game over screen. The game is implemented using object-oriented principles and demonstrates proper handling of loops, functions, memory management, and concurrency.