TEAM: C GRADES

GitHub link: https://github.com/Chiraagkv/Snakes-and-Ladders/

Members	Work Contribution
Naksh Sharma	File I/O (Save & Load system)
Utkarsh Gupta	Major game logic, movement, dice system, animation logic
Chiraag KV	Graphics, UI drawing, rendering and remaining functionalities

Introduction

This project implements the classic board game **Snakes and Ladders** using the **C programming language** along with the **Raylib graphics library**.The focus was to recreate the game with interactive graphics, smooth animations, multiplayer support, and save game persistence.

The game supports **2–6 players**, name input, random placement of snakes and ladders, real-time dice rolling, animated movement, and a rematch option after a winner is declared.

Project Structure

Snakes-Ladders

- include

- s_and_l.h
- src
  - main.c
  - game.c
  - drawing.c
- README.md
- Makefile

Explanation of Major Files

s_and_l.h

- Contains all global constants and macro definitions such as board size and maximum players.
- Defines data structures: `Player`, `Snake`, `Ladder`, and vector coordinate utility type.
- Declares external global variables shared across files.
- Declares function prototypes used in gameplay and drawing modules, ensuring modular programming.

game.c

- Implements core gameplay logic: dice rolling, updating player movements, and applying snake/ladder rules.
- Generates valid random positions for snakes and ladders.
- Contains save and load functionality (File I/O system).
- Handles animation path creation and interpolated movement between cells.

drawing.c

- Responsible for all graphical rendering using Raylib.
- Draws the board grid, numbering style, snakes, ladders, and circular player tokens.
- Renders UI panel showing player names, colors, and turn highlighting.
- Converts board cell numbers into pixel coordinates for accurate placement.

main.c

- Contains the main game loop and controls application flow.
- Implements state machine logic for screen transitions:
  - Load/Resume game
  - Number of players input
  - Name entry interface
  - Gameplay mode
  - Game Over / Rematch
- Handles mouse and keyboard interaction.
- Calls Save, Load, Restart, animation, and drawing functions.

Makefile

- Automates the compilation and build process for the project.
- Compiles all `.c` source files and links required libraries (Raylib, math, threading, system libs).
- Creates an executable without manually typing complex `gcc` commands.
- Supports commands:
  - **make** – builds the executable program
  - **make run** – builds and launches the game
  - **make clean** – removes compiled output and temporary files

- Ensures portability and simplified development workflow.

README.md

- Contains instructions on how to compile, run, and clean the project.
- Describes required dependencies and features of the game.
- Provides overview of project structure.

Gameplay Functionality

- Players roll a dice using the button or space key
- Animated move from current cell to new cell
- If landed on a ladder base → move up
- If landed on snake head → move down
- First player who reaches **cell 100** wins
- Game-over screen gives **rematch** option
- Automatic save on exit, and resume if chosen at startup

Major Functions Defined in the C Files

Functions in game.c

- **roll()**
  - Returns a random number between 1 and 6 (dice simulation).
- **snakes_and_ladders(snakes[], ladders[])**
  - Randomly generates snake mouth/tail and ladder top/bottom positions with validation to avoid conflicts.
- **InitNewGame(players[], np, snakes[], ladders[], &currentTurn)**

- Resets all player positions to 0, assigns colors and IDs, initializes new snake and ladder placement.

- **CheckSaveFileExists(fileName)**
  - Checks if a saved game exists before offering resume option.
- **SaveGame(players, np, currentTurn, snakes, ladders, statusMsg)**
  - Saves current gameplay information into `s_and_l_save.txt` including names, positions, and board structure.
- **LoadGame(players, &np, &currentTurn, snakes, ladders, statusMsg)**
  - Loads saved game state from file to restore previous progress.
- **start_animation(player, oldPos, newPos)**
  - Computes movement path for smooth tile-by-tile animation visually.

Functions in drawing.c

- **cell_to_pixel(cell, cellSize)**
  - Converts a board cell number (1–100) into actual screen coordinates.
- **draw_board(size, cellSize)**
  - Draws the 10x10 numbered serpentine board grid.
- **draw_snakes_and_ladders(snakes[], ladders[], cellSize)**
  - Renders snakes (curved lines) and ladders (step lines) visually on the board.
- **draw_players(players[], np, cellSize)**
  - Draws round colored tokens for players; handles overlapping and animated movement.
- **draw_ui_panel(players[], np, currentTurn)**
  - Draws right side panel displaying list of players, colors, and whose turn it is.

Functions / Logic in main.c

- **main()**
  - Initializes window, loads assets, manages game loop, and handles cleanup.
- **State machine switch cases**
  - Controls transitions between screens:
    - Load or New
    - Player count selection
    - Name input
    - Gameplay
    - Game Over
- **Input and event handling**
  - Manages button interactions such as Roll, Save, Load, Restart
  - Oversees keyboard shortcuts (SPACE, ENTER, S, L, R)
- **Winner detection**
  - Checks if any player reaches tile 100 and triggers game-over mode.

Conclusion

This project successfully recreates a playable **Snakes & Ladders game** with:

- Modular multi-file structure in C
- Real-time graphical rendering and movement animation

- Persistent save & load system
- Multiplayer support and polished UI