EDGES

OUR EDGE IS YOUR SUCCESS

Project – Build Sudoku Game in C++

Project Overview:

The **Sudoku Game** project is a console-based implementation of the popular Sudoku puzzle, developed in C++. It demonstrates a modular, object-oriented design (OOP) and leverages the C++ Standard Template Library (STL). The game supports:

- **Console interaction**: Users can load puzzles from files, place values on the board, check puzzle validity, solve automatically, and save current puzzles.
- **Optional puzzle generation**: Automatically create Sudoku boards of varying difficulty.
- **Optional advanced checks**: More sophisticated Sudoku-solving techniques (e.g., locked candidates, naked pairs).

It's built in such a way that the **core logic** (board representation, solver, generator, advanced checks) is **decoupled** from the **user interface** (console), enabling future upgrades (like a Qt GUI) with minimal changes to the core modules.

---

2. Project Objectives

1. Reinforce OOP and C++ STL practices by developing a multi-class console application.
2. Implement Sudoku functionality:
   - A 9×9 grid.
   - Board operations (validity checks, get/set cells).
   - Puzzle solving using backtracking.
3. Showcase separation of concerns: Keep board logic, solver, and user interaction (console I/O) in distinct modules.
4. Optional: Introduce advanced features (puzzle generation, advanced solving checks, saving/loading from files).

---

3. Deliverables

Students must submit:

1. Source Code: All .cpp and .hpp files implementing each module.
2. Build Instructions: A simple CMakeLists.txt (or another build system) so the project can be compiled easily.
3. Recorded Demo

---

4. Functional Requirements

1. **Console Application**
   - Upon running, display the Sudoku puzzle (9×9), using a dot . for empty cells.

    - Offer a menu with options (e.g., enter a move, solve automatically, load/save puzzle, exit).
2. **Board Operations**
    - **Load** an initial puzzle into a 9×9 matrix (e.g., from a hard-coded array or a file).
    - **Check validity** of user moves (row/column/3×3 constraints).
    - **Set** a value in the puzzle if valid.
    - **Print** the board in a neat ASCII format.
3. **Solving**
    - **Backtracking Solver**: Attempt to fill remaining cells and complete the puzzle.
4. **Menu & Error Handling**
    - **Enter a Move**:
        1. Ask for row, column, and value (1–9).
        2. Validate integer input and range.
        3. If invalid, print error message. Otherwise, place the value in the board.
    - **Solve Automatically**:
        1. Trigger the solver and display the completed board or an error if unsolvable.
    - **Load From File** *(Optional)*:
        1. Reads a file containing a 9×9 puzzle.
    - **Save To File** *(Optional)*:
        1. Writes the current board to a file.
    - **Exit**: End the application safely.
5. **Optional Features**:
    - **Puzzle Generation**: Create new Sudoku puzzles of varying difficulty.
    - **Advanced Techniques** (Locked Candidates, Naked Pairs, etc.): Provide deeper logic checks.

---

5. Technical Details

5.1 Class Responsibilities

- **SudokuBoard**
    - Stores and manipulates the 9×9 grid.
    - Validates row/column/box constraints.
    - Prints the board.
    - (Optional) Loads and saves puzzle data to a file.
- **SudokuSolver**
    - Backtracking method to find a solution or detect unsolvability.
- **SudokuGame**
    - Presents console menu, reads/writes user input.
    - Coordinates SudokuBoard and SudokuSolver.
    - Handles error-checking for moves and file operations.
- **(Optional) SudokuGenerator**
    - Creates a fully solved board, then removes cells while ensuring uniqueness.
- **(Optional) SudokuAdvancedChecks**
    - Houses advanced Sudoku logic (locked candidates, naked pairs, etc.).

5.2 Data Structures

- **Board**: std::vector<std::vector<int>> board holding integers in [0..9], where 0 denotes an empty cell.

5.3 Error Handling

- **Input Validation**:
  - Only accept integers for menu choices.
  - Reject out-of-range row/col/value.
  - Provide user feedback if input is invalid.

---

6. Sample Console Flow

1. **Startup**:

```
-------------------------------------
5 3 .    | . 7 .   | . . .
6 . .    | 1 9 5  | . . .
. 9 8    | . . .   | . 6 .
-------------------------------------
8 . .    | . 6 .   | . . 3
4 . .    | 8 . 3  | . . 1
7 . .    | . 2 .   | . . 6
-------------------------------------
. 6 .    | . . .   | 2 8 .
. . .    | 4 1 9  | . . 5
. . .    | . 8 .   | . 7 9
-------------------------------------

1) Enter a move
2) Solve automatically
3) Load puzzle from file
4) Save current puzzle to file
5) Exit
Choice:
```

2. **User Enters Move**:

```
Choice: 1
Enter row (1-9), column (1-9), and value (1-9): 1 3 4
Move accepted!
(Board displays again with accepted move)
```

3. **Invalid Move**:

```
Choice: 1
Enter row (1-9), column (1-9), and value (1-9): 10 5 2
Row must be between 1 and 9.
(Board displays redraw unchanged)
```

4. **Solve Automatically**:

```
Choice: 2
Puzzle solved!
(Board displays fully solved)
```

5. **Exit**:

```
Choice: 5
(Program ends)
```

Hints:

1. **Suggested Project Structure:**

```
.
├── SudokuBoard.hpp        // Board representation & basic I/O
├── SudokuBoard.cpp
├── SudokuSolver.hpp       // Backtracking solver
├── SudokuSolver.cpp
├── SudokuGame.hpp         // Console interface / game loop
├── SudokuGame.cpp
├── (Optional) SudokuGenerator.hpp   // Puzzle generation
├── (Optional) SudokuGenerator.cpp
├── (Optional) SudokuAdvancedChecks.hpp   // Advanced logic
├── (Optional) SudokuAdvancedChecks.cpp
└── main.cpp               // Entry point
```

2. **Please Refer to attached document for more Input/Output Samples, and CMakelists.txt Implementation**

Thank You
Edges For Training Team