BAHRIA UNIVERSITY

DSA LAB PROJECT REPORT

NAME: KASHAN MAQSOOD(022)

NAME: KHUBAIB MUHAMMAD(023) LEAD

NAME: MAHAD AJMAL(024)

LAB JOURNAL 11

FACULTY NAME: SAMIA KIRAN

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY ISLAMABAD CAMPUS

Interactive Menu-Based Sudoku Application with Advanced UI Elements

2. Abstract

Overview:
The project aims to create a modular and interactive Sudoku game with a menu-driven design using C++ and SFML. Key features include user-friendly navigation, modular components, and advanced graphical representation.

Key Objectives:

1. Build an engaging and intuitive interface.
2. Apply modular programming principles for scalability and maintainability.
3. Ensure seamless transitions and robust event handling between different game components.

Outcomes:
The result is a fully functional Sudoku game with visually appealing graphics, smooth controls, and efficient performance.

---

3. Introduction

Problem Statement:
Explains the shortcomings of existing Sudoku applications, such as a lack of modular design or poor visual interfaces, which can lead to limited usability and maintenance challenges.

Objectives:

1. Develop a **main menu** for seamless navigation.
2. Design modular components to ensure structured and maintainable code.
3. Use advanced graphical elements and textures to enhance user experience.

4. Methodology

Data Structures

1. **Dynamic Memory Allocation:**
   - Used to manage window objects, textures, and graphical elements dynamically.
   - For example, game states and windows are dynamically allocated at runtime, enabling efficient resource management.

2. **Arrays/Grids:**
   - The Sudoku grid is represented as a 2D array (or grid), where each cell holds either a fixed puzzle value or a player input.
   - Example:
   - int sudokuGrid[9][9] = { /* Predefined puzzle values */ };
3. **Vectors:**
   - Used to store player inputs, settings, and other dynamic lists.
   - Example: A std::vector<int> stores all the hints used by the player for efficient access and modification.
   - std::vector<int> hintsUsed;
4. **Maps (Associative Arrays):**
   - Used to associate game settings or options with their values.
   - Example:

     map<std::string, int> maps setting names to their values (e.g., difficulty level or number of hints allowed).

```
  map<std::string, int> gameSettings = {
{"Difficulty", 1},  // 1: Easy, 2: Medium, 3: Hard
{"HintsAllowed", 3}
  };
```

5. **Stacks:**
   - Used to implement an Undo feature, allowing players to revert their previous moves.
   - Example:

     std::stack<std::pair<int, int>> undoMoves; // Stores grid coordinates of recent moves
6. **Queues:**
   - Used for animations or transitions between windows.
   - Example: A queue could manage animation frames for smooth transitions.
7. **Priority Queues:**
   - Could be used for advanced features, such as prioritizing hints or AI solving steps.

Implementation

1. **Tools and Libraries:**
   - SFML: For graphical rendering and event handling.
   - C++: Provides flexibility and performance for handling complex logic.
2. **Classes and Objects:**
   - MainMenu: Manages navigation options and event handling for the main menu.
   - PlayGameWindow: Handles Sudoku game logic, player inputs, and board rendering.
   - OptionsWindow: Manages settings customization.
   - HowToPlayWindow: Displays instructions and game rules.
3. **Event Handling:**
   - Uses keyboard events for navigation (e.g., arrow keys and Enter key).
   - Mouse events handle clicks for selecting options or interacting with the Sudoku grid.
4. **Graphics and Textures:**
   - Textures are loaded dynamically for the background, buttons, and grid elements to create a polished visual design.
5. **Sudoku Grid Implementation:**
   - A 2D vector of structs could represent each cell with attributes like fixed value, current value, and whether it's editable.
     Example:

```
struct Cell {
   int fixedValue;
   int currentValue;
   bool isEditable;
      };


vector<std::vector<Cell>> sudokuGrid(9, std::vector<Cell>(9));
```

6. **Hint System with Maps and Grids:**
   - Hints are stored in a map, associating grid coordinates with correct values:

```
map<std::pair<int, int>, int> hints = {
   {{0, 0}, 5},  // Row 0, Column 0 has a hint of 5
   {{1, 2}, 3}
```

};

7. Undo and Redo Feature with Stacks:
    - Tracks moves using stacks for undo and redo operations:
    - stack<std::pair<int, int>> undoStack; // Stores previous moves
    - stack<std::pair<int, int>> redoStack; // Stores undone moves

Code Explanation

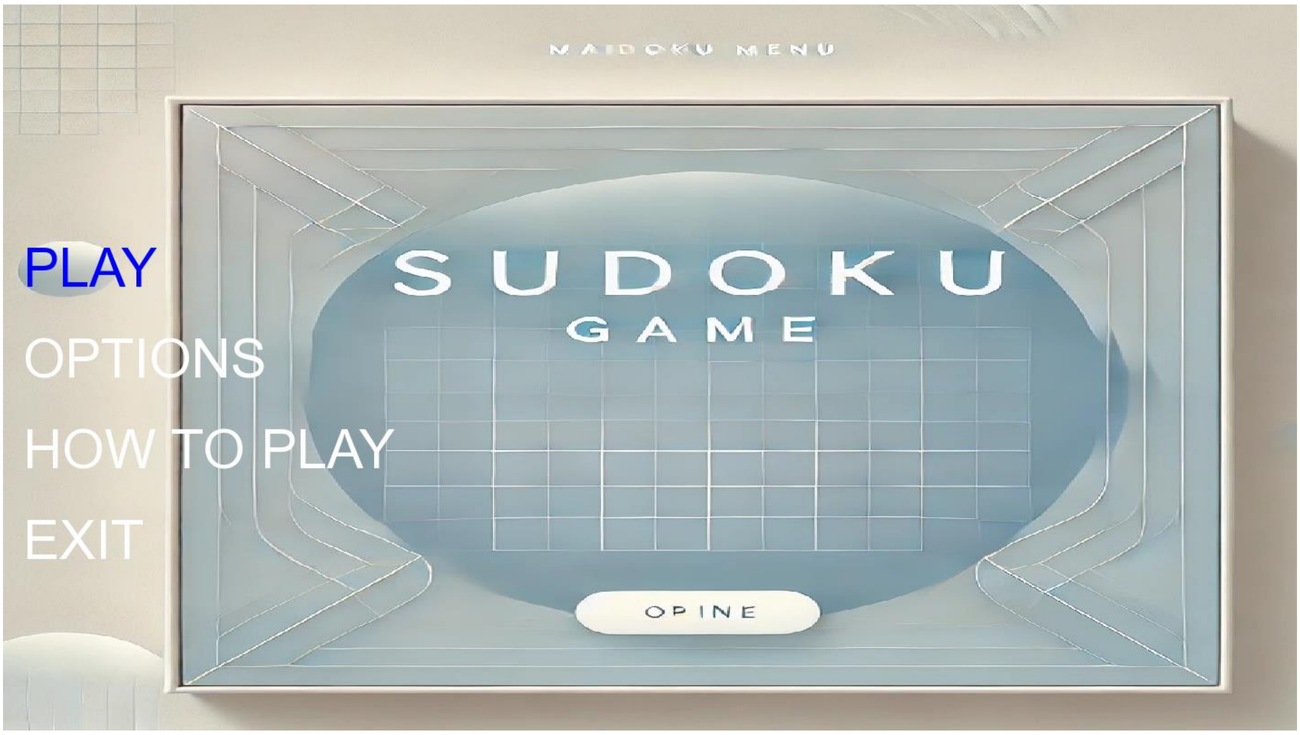
1. **Main Menu:**
    - Displays navigation options such as Play Game, Options, How-to-Play, and Exit.
    - Arrow keys are used to navigate, and Enter selects an option.
2. **Play Window:**
    - Initializes the Sudoku board and uses background textures for visual clarity.
    - Incorporates logic for user interactions and event-driven programming.
3. **Options Window:**
    - Configures gameplay settings such as hints and error limits.
4. **How-to-Play Window:**
    - Provides a guide for understanding Sudoku rules and gameplay.

---

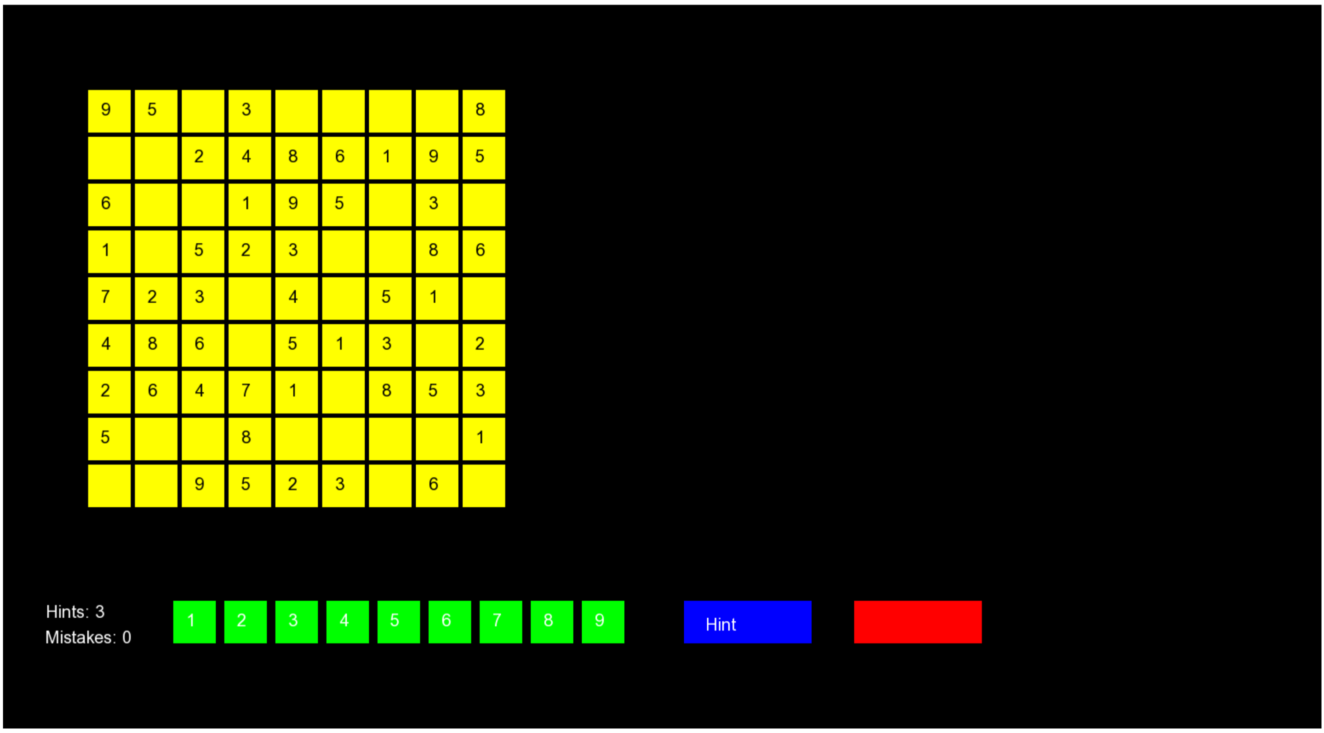
5. Results

Visual Outputs

- **Main Menu:**



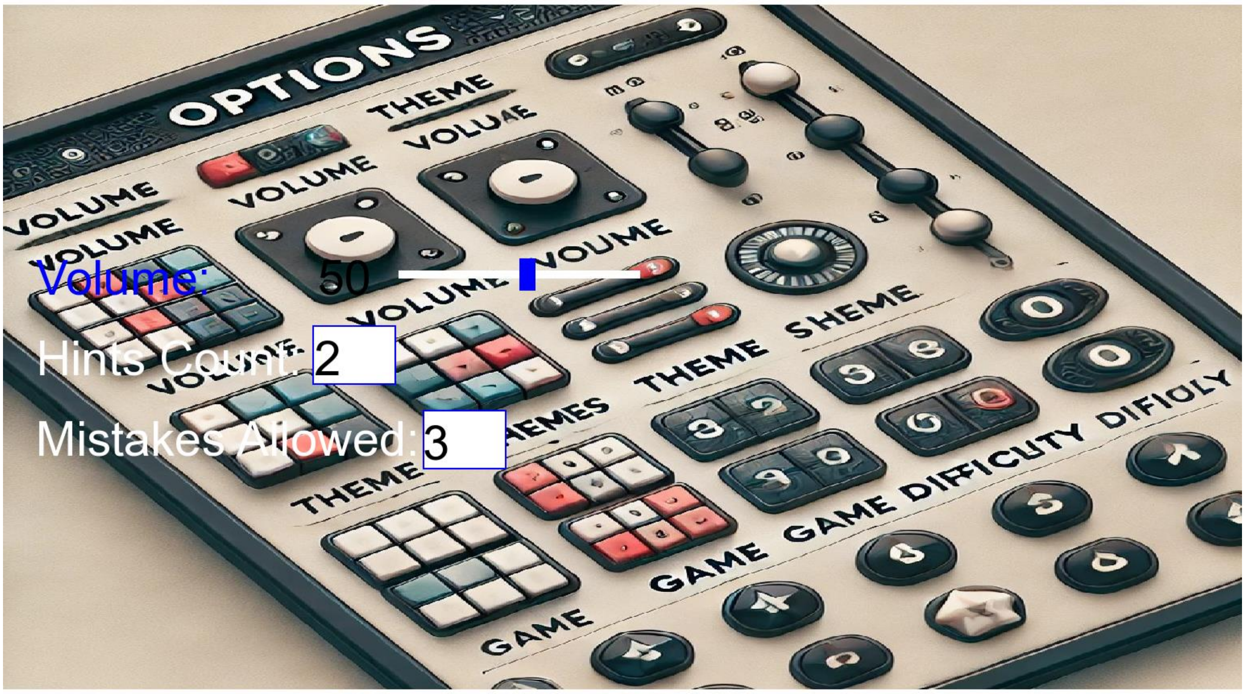
- **Play Interface:**



9	5		3					8
		2	4	8	6	1	9	5
6			1	9	5		3	
1		5	2	3			8	6
7	2	3		4		5	1	
4	8	6		5	1	3		2
2	6	4	7	1		8	5	3
5			8					1
		9	5	2	3		6	

Hints: 3
Mistakes: 0

- **Options Menu:** Allows customization of gameplay mechanics.

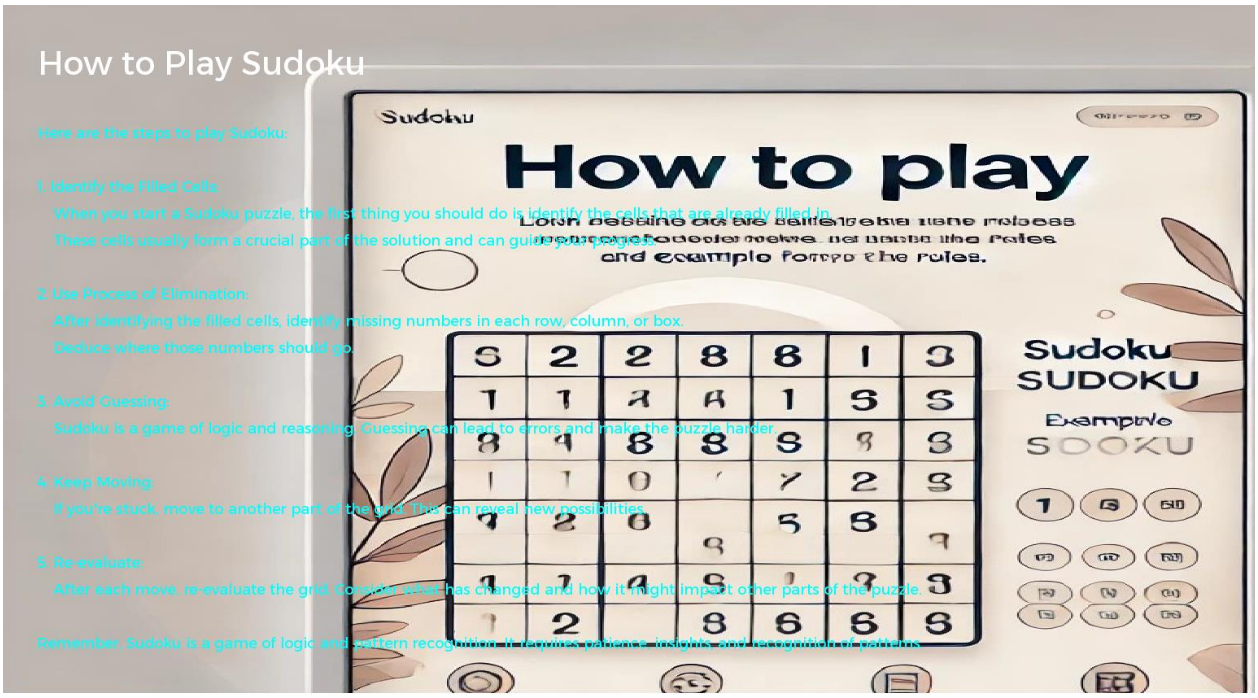


Volume: 50

Hints Count: 2

Mistakes Allowed: 3

- **How-to-Play Guide:** Offers concise and visually appealing instructions.



How to Play Sudoku

Here are the steps to play Sudoku:

1. Identify the Filled Cells:
When you start a Sudoku puzzle, the first thing you should do is identify the cells that are already filled in.
These cells usually form a crucial part of the solution and can guide your progress.

2. Use Process of Elimination:
After identifying the filled cells, identify missing numbers in each row, column, or box.
Deduce where those numbers should go.

3. Avoid Guessing:
Sudoku is a game of logic and reasoning. Guessing can lead to errors and make the puzzle harder.

4. Keep Moving:
If you're stuck, move to another part of the grid. This can reveal new possibilities.

5. Re-evaluate:
After each move, re-evaluate the grid. Consider what has changed and how it might impact other parts of the puzzle.

Remember, Sudoku is a game of logic and pattern recognition. It requires patience, insights, and recognition of patterns.

Performance Metrics

- Achieved consistent frame rates during transitions.
- Optimized memory usage for graphical elements and window management.

6. Discussion

Analysis of Results:

- Demonstrated effective modular programming to create a structured application.
- SFML proved to be a suitable library for graphics and event handling.

Challenges Faced:

1. Managing multiple event loops across different windows.
2. Optimizing texture loading for compatibility with various screen resolutions.

Limitations:

1. Does not include advanced Sudoku-solving algorithms.
2. Lacks features like user profiles and leaderboards.

Solutions and Future Work:

1. Incorporate AI-based Sudoku solving to assist or challenge players.
2. Add features such as user authentication, scoring, and difficulty customization.

---

7. Conclusion

Summary:
Successfully created a modular and interactive Sudoku application with visually appealing and user-friendly interfaces. SFML enabled smooth graphics rendering and responsive event-driven programming.

Recommendations:

1. Expand functionality by integrating AI algorithms for advanced gameplay.
2. Improve visual elements with animations and dynamic effects.

---

8. References

- **SFML Documentation:** Official documentation for the library used in the project.
- **TutorialsPoint:** Resource for learning C++ basics.
- Research papers and online tutorials about Sudoku algorithms and game design principles.