**Project Report**

**On**

**Sudoku Solver Visualizer**



**SUBMITTED TO : Rahul sir**

**SUBMITTED BY:**

NAME :- Ananya Gupta

Registration No :- 12205994R

Roll No :- 5

Section :- 9SK01

**Table of Contents**

**1. Introduction**

**2. Project Objectives**

**3. System Requirements**

**4. Project Design**

4.1. SudokuSolver Class

4.2. SudokuVisualizer Class

**5. Implementation Details**

5.1. SudokuSolver Class Methods

5.2. SudokuVisualizer Class Methods

**6. User Guide**

**7. Code Snippet & Output**

**8. Future Enhancements**

**9. Conclusion**

**1. Introduction**

The Sudoku Solver Visualizer is a Java-based application designed to solve Sudoku puzzles and visualize the solving process. It employs a backtracking algorithm to find the solution and displays each step in real-time, making it a valuable tool for understanding the problem-solving approach.

**2. Project Objectives**

* Implement a backtracking algorithm to solve Sudoku puzzles.
* Visualize the solving process step-by-step.
* Provide an interactive user interface for users to input and solve Sudoku puzzles.
* Enhance understanding of backtracking algorithms through visualization.

**3. System Requirements**

* **Hardware**: Any modern computer capable of running Java applications.
* **Software**:
  + Java Development Kit (JDK) 8 or higher
  + An Integrated Development Environment (IDE) like IntelliJ IDEA, Eclipse, or NetBeans
  + Swing library for GUI components (included in standard Java library)

**4. Project Design**

The project consists of two main components:

* **SudokuSolver**: Handles the logic for solving the Sudoku puzzle using a backtracking algorithm.
* **SudokuVisualizer**: Provides a graphical user interface (GUI) to display the Sudoku grid and visualize the solving process.

**4.1) SudokuSolver Class**

This class is responsible for solving the Sudoku puzzle. It utilizes a recursive backtracking algorithm to fill the empty cells in the grid.

**4.2 SudokuVisualizer Class**

This class provides the GUI for the Sudoku Solver. It displays the Sudoku grid, allows user interaction, and visualizes the step-by-step solving process.

**5. Implementation Details**

**5.1 SudokuSolver Class Methods**

* **Constructor**:
  + public SudokuSolver(int[][] board, SudokuVisualizer visualizer, int delay)
    - Initializes the board, visualizer, and delay parameters.
* **solve**:
  + public boolean solve()
    - Uses the backtracking algorithm to solve the Sudoku puzzle. It iteratively tries numbers in empty cells and recurses to solve subsequent cells.
    - Updates the visualizer at each step and introduces a delay to visualize the solving process.
* **isSafe**:
  + private boolean isSafe(int row, int col, int num)
    - Checks if placing a number in a specific cell is valid according to Sudoku rules (no duplicates in the row, column, or 3x3 subgrid).
* **getBoard**:
  + public int[][] getBoard()
    - Returns the current state of the board.

**5.2) SudokuVisualizer Class Methods**

* **Constructor**:
  + public SudokuVisualizer(int[][] board, int delay)
    - Initializes the Sudoku grid, sets up the GUI components, and starts the visualizer.
* **initBoard**:
  + private void initBoard(JPanel gridPanel)
    - Initializes the Sudoku grid cells, sets their properties (font, alignment, background color), and adds them to the grid panel.
* **addControls**:
  + private void addControls()
    - Adds control buttons (e.g., "Solve" button) to the GUI and sets up their action listeners.
* **solveSudoku**:
  + private void solveSudoku()
    - Calls the solver's solve method to start solving the puzzle. Runs in a separate thread to avoid freezing the GUI.
* **updateCell**:
  + public void updateCell(int row, int col, int value)
    - Updates the text of a specific cell in the grid to visualize the solving process.
* **updateBoard**:
  + private void updateBoard()
    - Updates the entire board after solving to reflect the final solution.

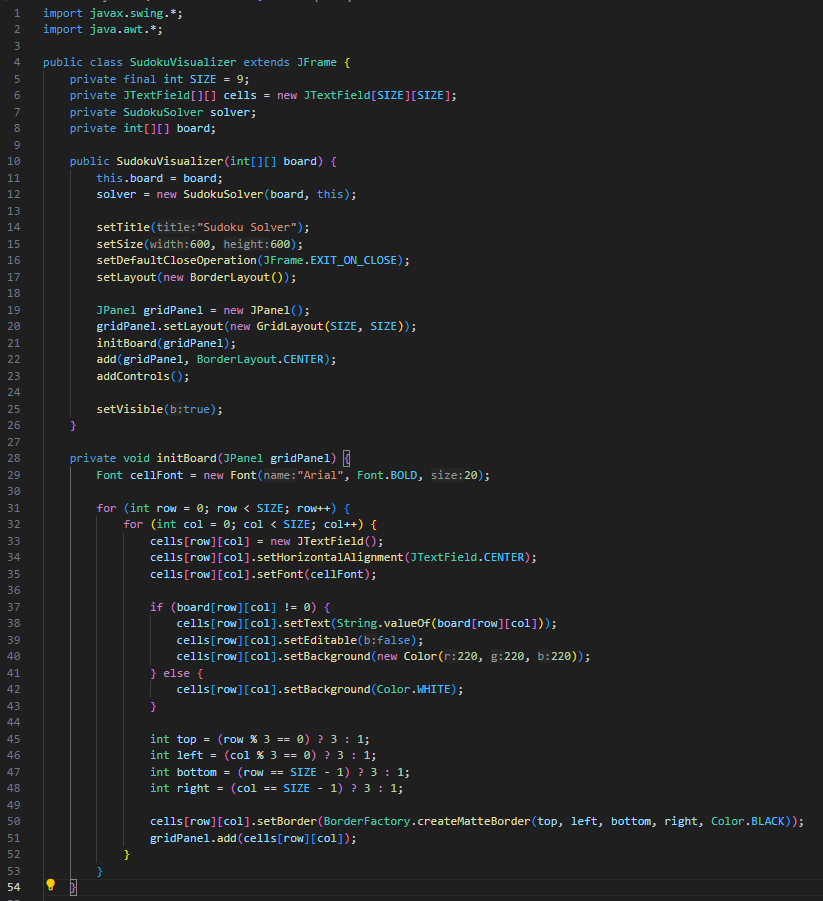
**6. User Guide**

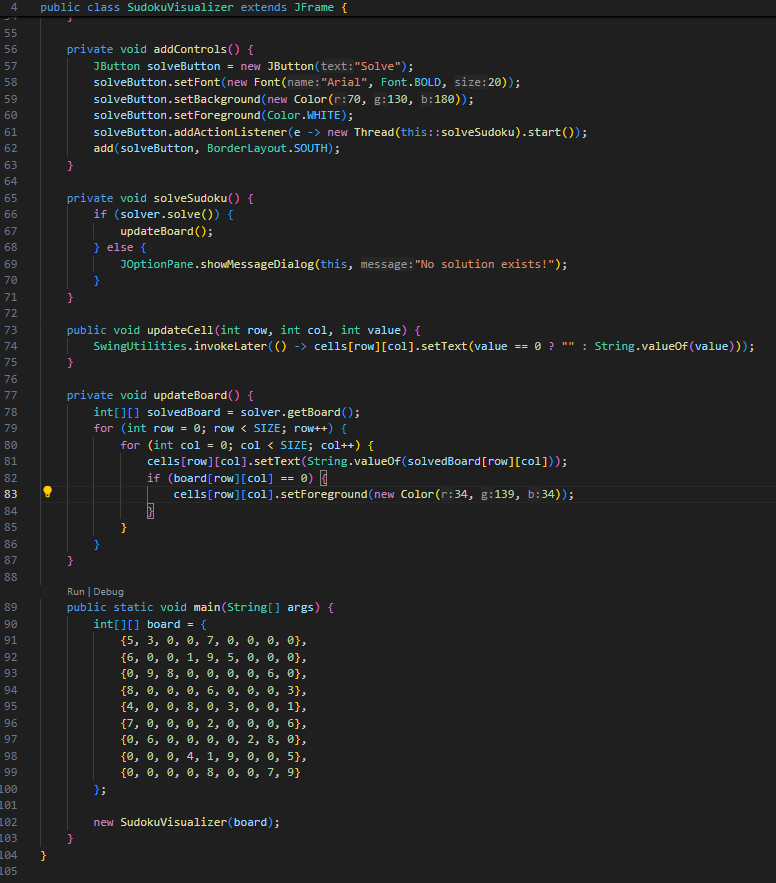
1. **Running the Application**:
   * Compile and run the SudokuVisualizer class.
   * The application window will open, displaying the Sudoku grid with a "Solve" button.
2. **Using the Application**:
   * Initially, a default Sudoku puzzle is loaded.
   * Click the "Solve" button to start solving the puzzle.
   * The solving process will be visualized in real-time, with each step displayed on the grid.
3. **Adjusting the Speed**:
   * Modify the delay parameter in the SudokuVisualizer main method to adjust the speed of the visualization.

**7. Code Snippet & Output**

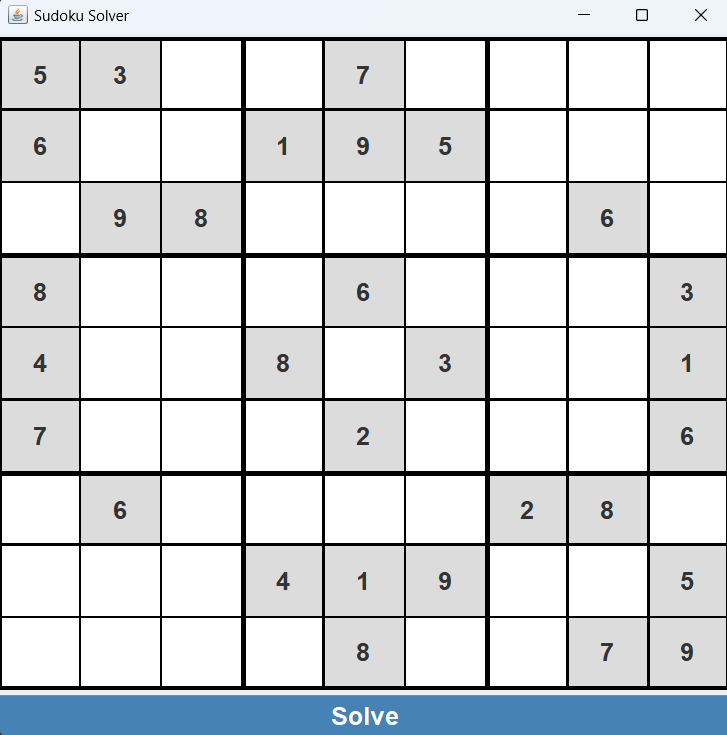
**7.1) Sudoku solver code**

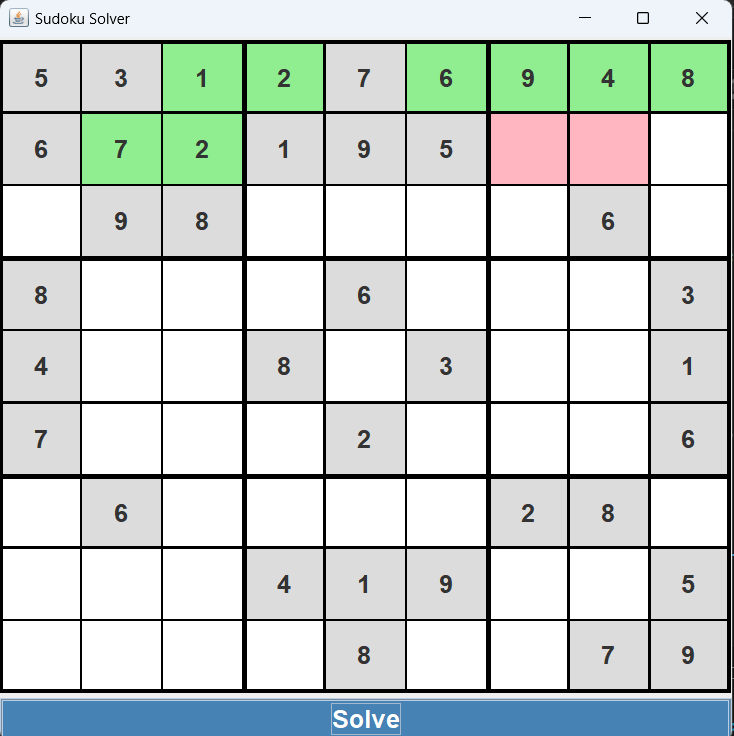
**7.2) Sudoku Visualizer Code**

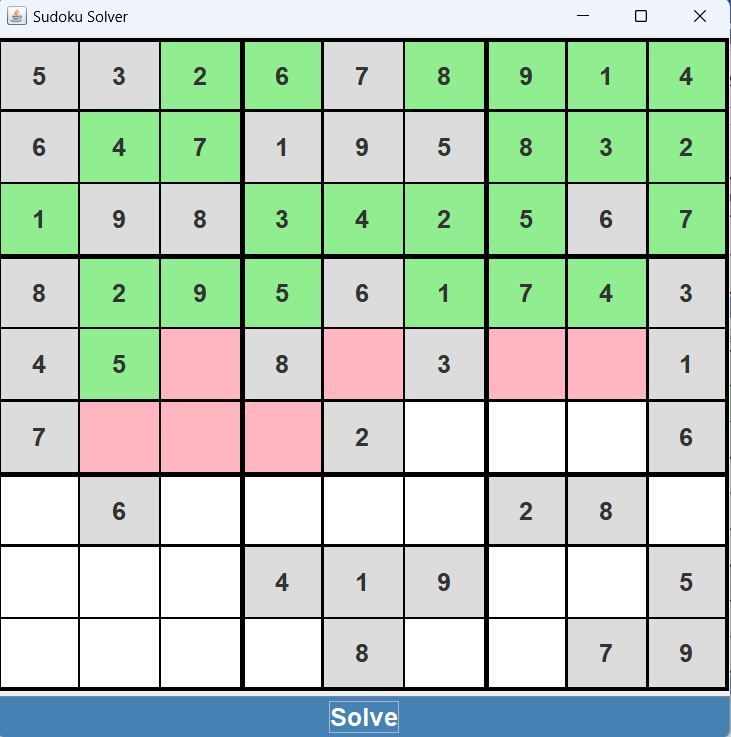
****

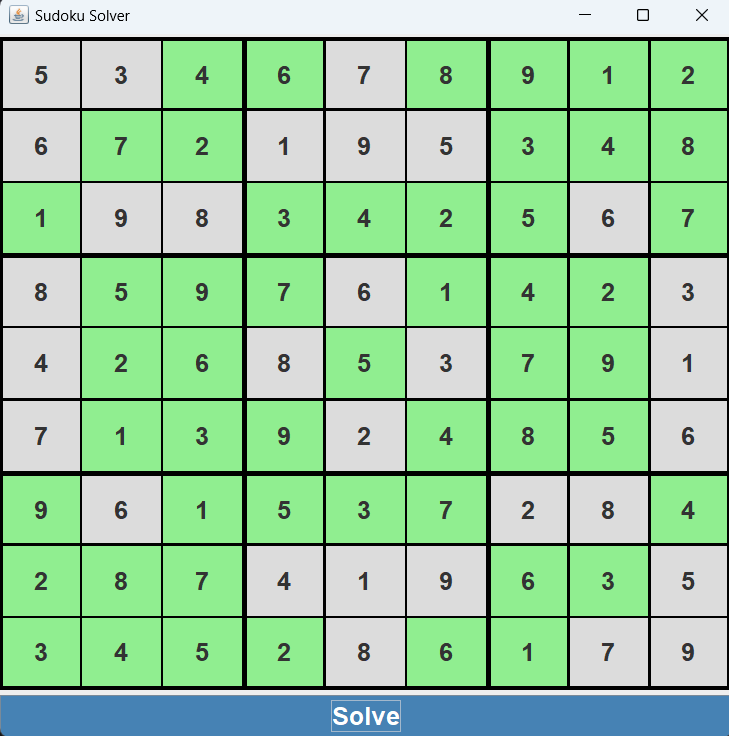
****

**7.3) Code Outputs**

****

****

****

****

**8. Future Enhancements**

* **User Input**: Allow users to input their own Sudoku puzzles directly into the grid.
* **Customization**: Add options for users to customize the delay and colors used in the visualization.
* **Performance Optimization**: Improve the efficiency of the solving algorithm for faster visualization.
* **Mobile Compatibility**: Develop a mobile version of the application.

**9. Conclusion**

The Sudoku Solver Visualizer provides an interactive way to understand the backtracking algorithm used to solve Sudoku puzzles. By visualizing each step of the process, users can gain deeper insights into how the algorithm works and how it approaches solving complex problems.