" **Sudoku Solver Visualizer** "

*Project report submitted in fulfilment of the requirements for the Degree of*

# BACHELOR OF TECHNOLOGY

**in**

**COMPUTER SCIENCE AND ENGINEERING**

By

**Vikash Mishra**

## ( 12203720 )



**School of Computer Science and Engineering**

Lovely Professional University Phagwara, Punjab (India) November, April 2024

**Introduction**

The Sudoku Solver Visualizer is a Java application designed to solve Sudoku puzzles interactively through a graphical user interface (GUI). Sudoku, a popular puzzle game known for its logic-based challenges, requires filling a 9x9 grid with digits from 1 to 9, ensuring that each column, each row, and each of the nine 3x3 subgrids contain all of the digits without repetition.

**Functionality and Features**

The Sudoku Solver Visualizer offers several key features to facilitate puzzle solving and interaction:

1. **Graphical User Interface (GUI)**:
   * **Grid Representation**: The main puzzle grid is displayed using Java Swing's JTextField components arranged in a 9x9 layout.
   * **Input and Display**: Users can input puzzle values manually or load predefined puzzles. Each cell's background color (CELL\_BACKGROUND) distinguishes between fixed (given) and editable cells.
   * **Button Controls**: Three control buttons (Load Puzzle, Solve, Clear) facilitate puzzle manipulation, each styled with a consistent visual theme (BUTTON\_COLOR and BUTTON\_TEXT\_COLOR).
2. **Solver Algorithms**:
   * **Backtracking Algorithm**: The core solving mechanism employs a recursive backtracking approach (solve() method). This method systematically fills each empty cell with valid numbers, backtracking when a solution path fails.
   * **Validation**: The isValid() method ensures that no number is repeated within its row, column, or 3x3 subgrid, crucial for maintaining Sudoku puzzle rules.
3. **Visual Feedback and Delay**:
   * **Update Mechanism**: During the solving process, each step updates the GUI to reflect changes (updateGUI() method). This includes filling cells with potential solutions and applying a brief delay (delay() method) to visualize the solving steps.
   * **Color Coding**: Fixed numbers (FIXED\_NUMBER\_COLOR) and solved numbers (SOLVED\_NUMBER\_COLOR) are visually distinguished to enhance clarity and user experience.
4. **User Interaction**:
   * **Load Puzzle**: Users can load predefined puzzles (loadPuzzle() method) or manually enter new puzzles.
   * **Solve and Clear**: The Solve button triggers the solving process, while the Clear button resets the grid for new inputs or puzzle loading.
5. **Error Handling and Messages**:
   * **Feedback Dialogs**: Message dialogs (JOptionPane) provide informative messages upon successful puzzle solving or errors (solvePuzzle() method).

**Technical Design and Implementation**

The Sudoku Solver GUI is implemented using object-oriented programming principles in Java, leveraging the Java Swing framework for the graphical interface. Here’s an overview of its technical components:

1. **Main Class (SudokuSolverGUI)**:
   * **Extends JFrame**: The application window (JFrame) encapsulates the entire GUI, with specified dimensions (setSize(600, 600)), title (setTitle("Sudoku Solver")), and default close operation (setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)).
   * **Layout and Components**: The GUI layout employs BorderLayout for main content (gridPanel and buttonPanel) and South-aligned controls (buttonPanel).
   * **Color Constants**: Defined constants (BACKGROUND\_COLOR, CELL\_BACKGROUND, etc.) ensure consistent visual styling across components.
2. **Grid Panel (gridPanel)**:
   * **Grid Layout**: A nested loop initializes a 9x9 grid of JTextField cells (cells[][]) within gridPanel. Each cell is configured with alignment, font, background color, and border settings.
3. **Button Panel (buttonPanel)**:
   * **Control Buttons**: Three buttons (Load Puzzle, Solve, Clear) are created using a helper method (createStyledButton()). Each button triggers corresponding actions (loadPuzzle(), solvePuzzle(), clearBoard()) upon user interaction.
4. **Data Structures**:
   * **Board Representation**: board[][] maintains the current state of the Sudoku grid, with 0 representing empty cells and 1-9 representing filled cells.
   * **Validation Sets**: HashSet<Integer>[] rows, cols, and subgrids store unique numbers in each row, column, and 3x3 subgrid, ensuring validity during puzzle solving.
5. **Event Handling**:
   * **Action Listeners**: Inline lambda expressions (e -> {...}) attach listeners to control buttons (loadButton.addActionListener(...), solveButton.addActionListener(...), clearButton.addActionListener(...)), triggering respective methods (loadPuzzle(), solvePuzzle(), clearBoard()).
6. **Solver Methods**:
   * **loadPuzzle()**: Initializes the board[][] from a predefined puzzle[][], populating cells and configuring editable and fixed values (cells[][]).
   * **solvePuzzle()**: Initiates the solving process using a new thread (solve() method), displaying success or error messages upon completion.
   * **clearBoard()**: Resets the grid (cells[][]), board[][], and validation sets (rows[], cols[], subgrids[]) for new puzzle loading or user input.
7. **Solver Algorithm (solve() Method)**:
   * **Recursive Backtracking**: Implements a backtracking approach to find valid solutions for the Sudoku puzzle. It recursively explores potential numbers (1-9) for each empty cell, validating against isValid() constraints and backtracking on failure.
8. **GUI Update (updateGUI() Method)**:
   * **SwingUtilities.invokeLater()**: Ensures thread-safe GUI updates, modifying JTextField text and color (SOLVED\_NUMBER\_COLOR) based on solver progress.
9. **Delay Mechanism (delay() Method)**:
   * **Thread.sleep()**: Introduces a short delay (50 milliseconds) between solver steps, enhancing visual feedback without affecting performance.

**Performance and Optimization**

The Sudoku Solver GUI demonstrates efficient puzzle-solving capabilities through the following performance considerations:

1. **Algorithm Efficiency**: The backtracking algorithm efficiently explores solution paths, ensuring optimal performance for typical Sudoku puzzles (SIZE = 9).
2. **Resource Management**: Minimal resource consumption (memory and CPU) during solving operations, supported by Java's efficient handling of thread management and data structures (HashSet).
3. **Thread Safety**: GUI updates (SwingUtilities.invokeLater()) and delay mechanism (Thread.sleep()) maintain responsiveness and visual clarity without blocking the main event dispatch thread (EDT).
4. **Scalability**: Designed to handle larger grid sizes (SIZE > 9) with minimal code adjustments, supporting potential future expansions or customization.

**Future Enhancements**

To further enhance the Sudoku Solver GUI, several improvements and features can be considered:

1. **Advanced Solving Techniques**: Incorporation of advanced algorithms (e.g., Constraint Propagation, Dancing Links) to optimize solving speed and efficiency for complex puzzles.
2. **User Experience Enhancements**:
   * **Interactive Hint System**: Real-time hint generation for potential moves or logical deductions.
   * **Visual Themes and Customization**: User-selectable themes (dark mode, color schemes) for personalized aesthetics.
3. **Accessibility and Internationalization**:
   * **Keyboard Navigation**: Enhanced keyboard support (tab navigation, hotkeys) for accessibility.
   * **Multi-language Support**: Localization (internationalization) for global user accessibility.
4. **Community Integration**:
   * **Online Puzzle Repository**: Integration with online Sudoku communities for puzzle sharing, competition, and collaborative solving experiences.
   * **Cloud Sync**: Synchronization capabilities (Google Drive, Dropbox) for saving and sharing puzzle progress across devices.

Github link: [vikashmishra464/sudokuSolver (github.com)](https://github.com/vikashmishra464/sudokuSolver)

**Conclusion**

In conclusion, the Sudoku Solver GUI represents a robust implementation of a Sudoku puzzle solver using Java Swing, offering intuitive user interaction, efficient solving algorithms, and visual feedback mechanisms. Its structured design, adherence to OOP principles, and utilization of Java's concurrency and GUI capabilities ensure a responsive and engaging user experience. With potential enhancements in algorithmic efficiency, user customization, and community integration, the Sudoku Solver GUI holds promise for further enriching the puzzle-solving community and catering to diverse user preferences and needs.