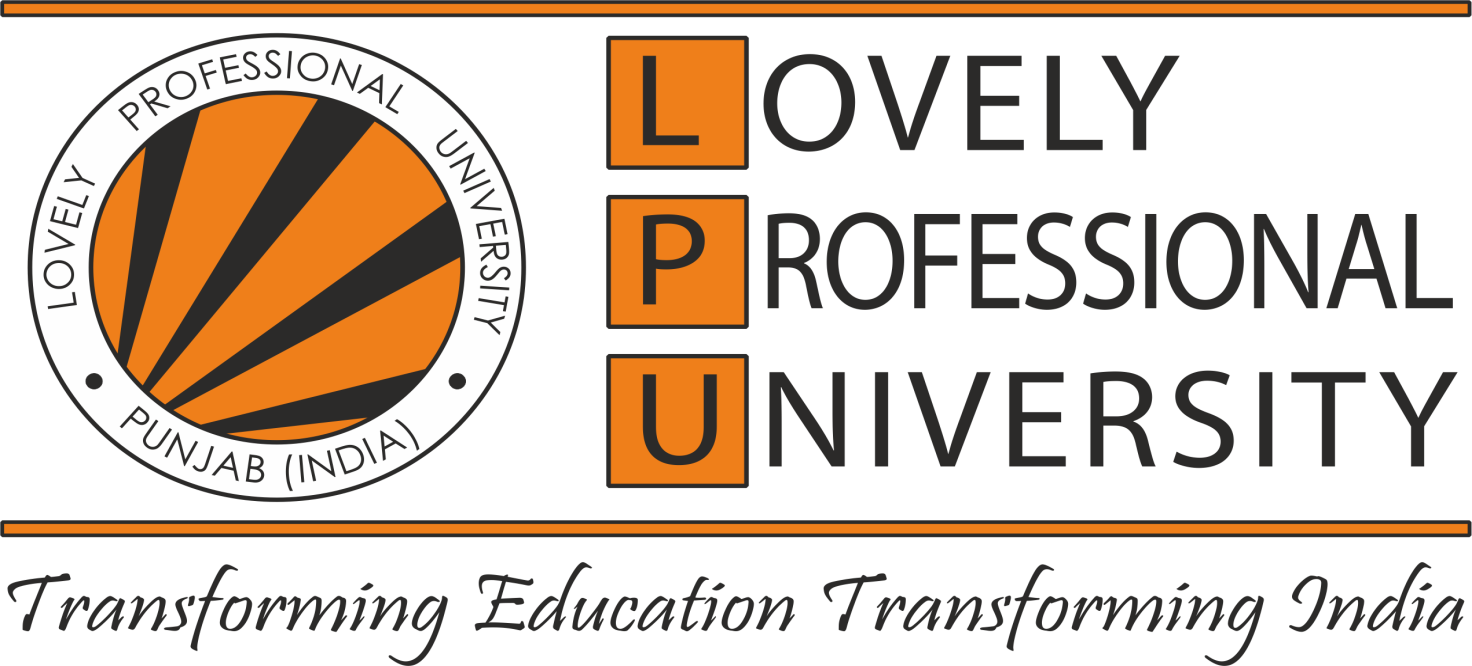
**PROJECT REPORT**



**Project Title - Sudoku Solver**

Name - Abhishek Kumar Maurya

Registration No. – 12223228

Roll No. – 46

Class – 9SK02

Submitted to – Rahul Singh Rajput

Table of contents

1. **Introduction**
   1. Key Features
   2. Purpose
   3. Technology Used
2. **Code Structure**
3. **Code Analysis**
   1. Main Method
      1. Key Features
      2. Purpose
   2. solveSudoku Method
      1. Key Features
      2. Purpose
   3. isValid Method
      1. Key Features
      2. Purpose
4. **Conclusion**
5. **GitHub Repository Link**
6. **Screenshots**
7. **Code**
8. **Introduction**

The Sudoku Solver Visualization project is a Java-based application that provides a graphical interface for solving Sudoku puzzles. The project utilizes the backtracking algorithm to solve the puzzle and visualizes the solving process in real-time. This report provides an overview of the project, its features, and the implementation details.

* 1. **Key Features**
* Graphical Interface: The application features a graphical interface that displays the Sudoku board as a 9x9 grid of text fields.
* Sudoku Solver: The application includes a Sudoku solver that uses the backtracking algorithm to solve the puzzle.
* Visualization: The solving process is visualized in real-time, with each step of the algorithm highlighted on the board.
* Reset Button: A reset button allows the user to reset the board to its initial state.
* Start Button: A start button initiates the solving process.
  1. **Purpose**

The purpose of this project is to create a Java-based application that:

* Solves Sudoku Puzzles: Utilizes the backtracking algorithm to solve Sudoku puzzles.
* Visualizes the Solving Process: Displays the step-by-step solving process in real-time, providing a clear understanding of how the algorithm works.
* Provides a Graphical Interface: Offers a user-friendly graphical interface for users to interact with the application.
  1. **Technologies Used**

The following technologies were used to develop the Sudoku Solver Visualization project:

* Programming Language: Java
* Graphical User Interface (GUI) Toolkit: Java Swing
* Algorithm: Backtracking algorithm for solving Sudoku puzzles
* Multithreading: Java threads for running the solving process in the background
* UI Components: JTextField, JButton, JPanel, JFrame
* Layout Managers: BorderLayout, GridLayout
* Event Handling: ActionListener for button clicks
* Threading: Thread.sleep() for adding delay between steps
* Graphics: repaint() method for updating the UI

1. **Code Structure**

The code is organized into the following classes and methods:

1. SudokuSolverVisualizer: The main class that contains the GUI components and the solving logic.
2. solveSudoku: The method that implements the backtracking algorithm to solve the puzzle.
3. isValid: The method that checks if a number can be placed at a given position without violating Sudoku rules.
4. usedInRow, usedInCol, usedInBox: Methods that check if a number is already used in a row, column, or box.
5. resetBoard: The method that resets the board to its initial state.
6. **Code Analysis**
   1. **Main Method**

The main method is the entry point of the Sudoku Solver Visualization project. It is the starting point of the program, and it is responsible for launching the application.

* + 1. **Key Points:**
* public static void main(String[] args): This is the signature of the main method, which is the entry point of the program.
* SwingUtilities.invokeLater(): This method is used to run the GUI-related code on the Event-Dispatching Thread (EDT), which is responsible for handling GUI events.
* SudokuSolverVisualizer sudokuSolverVisualizer = new SudokuSolverVisualizer(): This line creates a new instance of the SudokuSolverVisualizer class, which is the main class of the application.
* sudokuSolverVisualizer.createAndShowGUI(): This method is called on the newly created instance, which creates and configures the GUI components, such as the Sudoku board, buttons, and text fields.
  + 1. A screen shot of a computer program

       Description automatically generated **Purpose:**

The main method serves the following purposes:

* It launches the application.
* It creates an instance of the SudokuSolverVisualizer class.
* It initializes the GUI components and makes them visible.

By using SwingUtilities.invokeLater(), the main method ensures that the GUI-related code is run on the EDT, which is essential for ensuring the thread safety of the application.

* 1. **SolveSudoku Method**

The solveSudoku method is a recursive algorithm that solves the Sudoku puzzle using the backtracking technique. It is a crucial part of the Sudoku Solver Visualization project.

* + 1. **Key Points:**
* The method takes a 2D array board representing the Sudoku puzzle as input.
* It iterates through each cell of the board, searching for empty cells (represented by 0).
* For each empty cell, it tries numbers from 1 to 9, checking if the number is valid using the isValid method.
* If a valid number is found, it assigns the number to the cell and recursively calls solveSudoku to solve the rest of the puzzle.
* If the recursive call returns true, it means the puzzle is solved, and the method returns true.
* A screen shot of a computer program

  Description automatically generatedIf no valid number is found for a cell, it backtracks by setting the cell to 0 and highlighting it in red.
  + 1. **Purpose:**

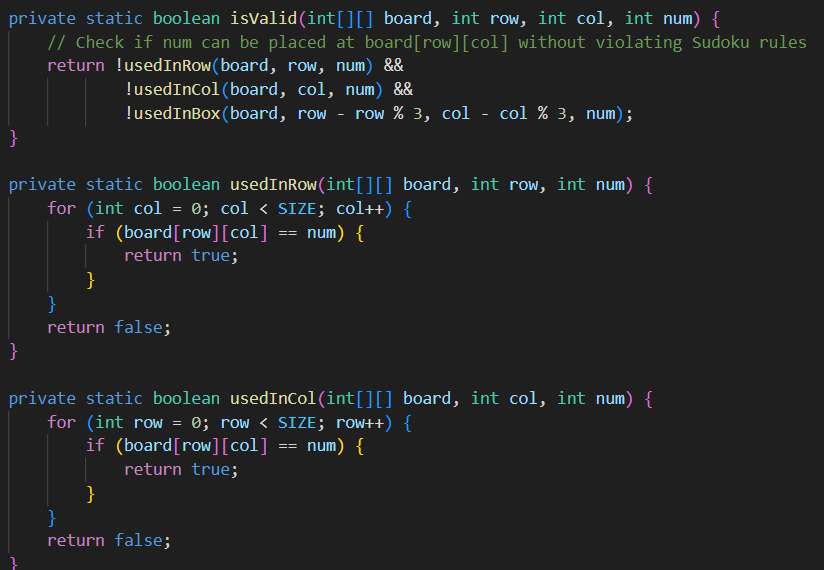
The solveSudoku method is designed to:

* Solve the Sudoku puzzle using the backtracking algorithm.
* Visualize the solving process by updating the GUI in real-time.
* Highlight the backtracking process by changing the cell color to red.
* By using recursion and the backtracking technique, the solveSudoku method efficiently solves the Sudoku puzzle and provides a visual representation of the solving process.
  1. **isValid Method**

The isValid method is a helper function used in the Sudoku Solver Visualization project to check if a given number can be placed in a specific cell without violating the Sudoku rules.

* + 1. **Key Points:**

The method takes four parameters:

* board: the Sudoku board
* row: the row index of the cell
* col: the column index of the cell
* num: the number to be checked
* It calls three helper methods:
* usedInRow: checks if num is already used in the same row
* usedInCol: checks if num is already used in the same column
* usedInBox: checks if num is already used in the same 3x3 box
* It returns true if num is not used in the row, column, or box, and false otherwise.
  + 1. **Purpose:**

The isValid method is designed to:

* Check if a given number can be placed in a specific cell without violating the Sudoku rules.
* Ensure that each row, column, and 3x3 box contains each number only once.

By using this method, the Sudoku Solver Visualization project can efficiently check the validity of a given number in a specific cell and ensure that the Sudoku puzzle is solved correctly.

1. **Conclusion**

The Sudoku Solver Visualization project is a Java-based application that provides a graphical interface for solving Sudoku puzzles. The application utilizes the backtracking algorithm to solve the puzzle and visualizes the solving process in real-time. The project demonstrates a clear understanding of the Sudoku solving algorithm and Java programming concepts.

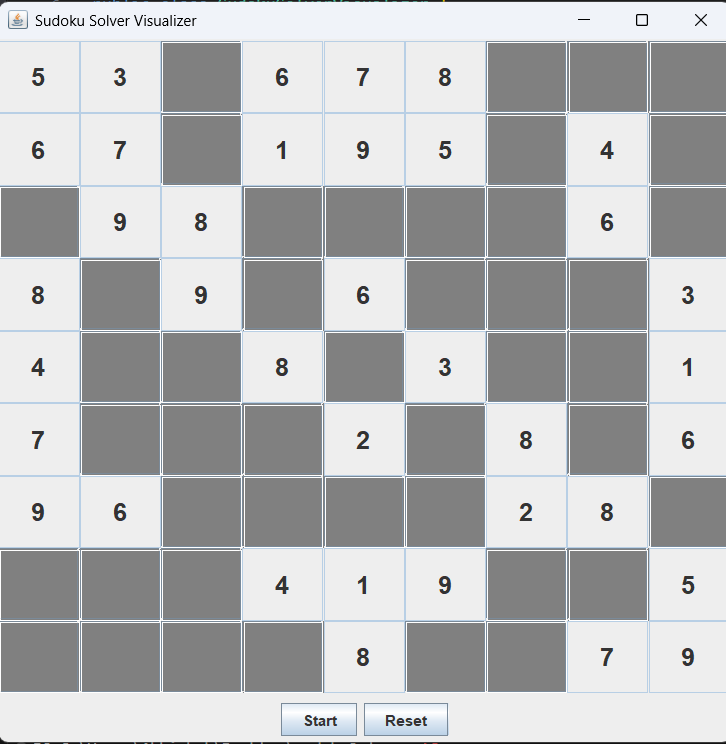
Future Enhancements

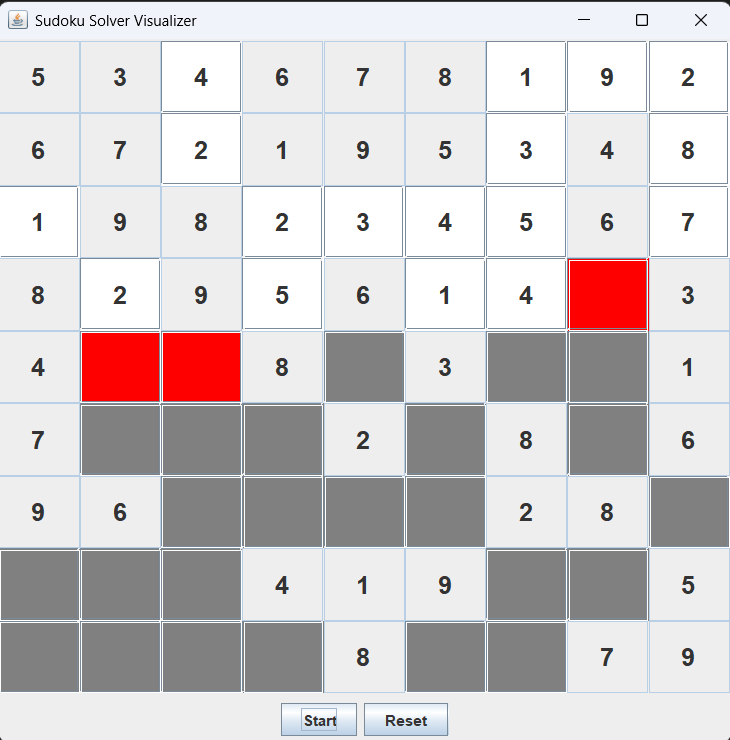
* Improving Performance: Optimizing the solving algorithm to reduce the solving time.
* Adding Difficulty Levels: Adding different difficulty levels for the user to choose from.
* Saving and Loading: Adding the ability to save and load Sudoku puzzles.

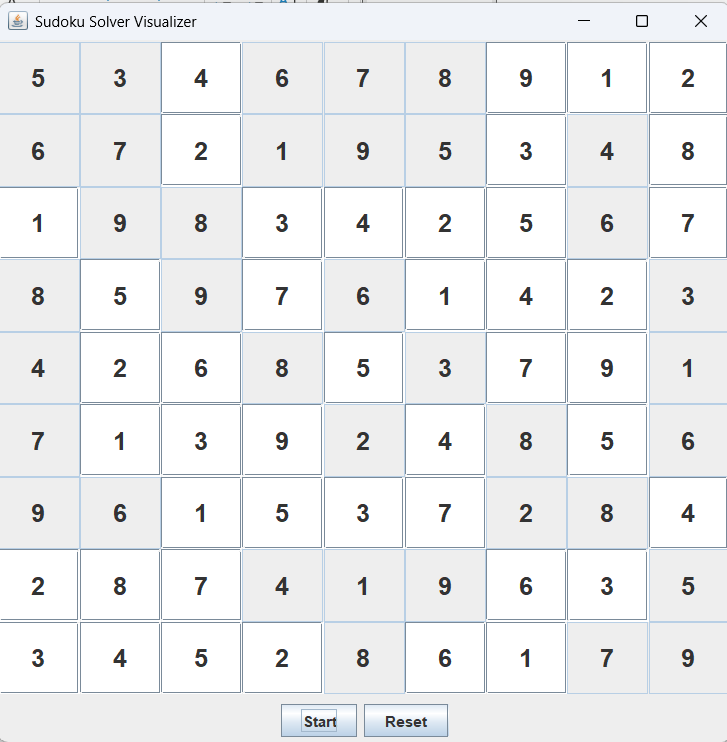
1. **GitHub Repository**

[Abhishek-Maurya2/sudokuSolver (github.com)](https://github.com/Abhishek-Maurya2/sudokuSolver)

1. **Screenshots**

****

****

****

1. **Code**
2. import javax.swing.\*;
3. import java.awt.\*;
4. import java.awt.event.ActionEvent;
5. import java.awt.event.ActionListener;
6. public class SudokuSolverVisualizer {
7. private static final int SIZE = 9;
8. private static JTextField[][] cells = new JTextField[SIZE][SIZE];
9. private static JFrame frame;
10. private static JButton startButton;
11. private static JButton resetButton;
12. private static int[][] initialBoard;
13. public static void main(String[] args) {
14. SwingUtilities.invokeLater(() -> {
15. frame = new JFrame("Sudoku Solver Visualizer");
16. frame.setLayout(new BorderLayout());
17. JPanel gridPanel = new JPanel(new GridLayout(SIZE, SIZE));
18. // Initialize Sudoku board
19. int[][] board = {
20. { 5, 3, 0, 6, 7, 8, 0, 0, 0 },
21. { 6, 7, 0, 1, 9, 5, 0, 4, 0 },
22. { 0, 9, 8, 0, 0, 0, 0, 6, 0 },
23. { 8, 0, 9, 0, 6, 0, 0, 0, 3 },
24. { 4, 0, 0, 8, 0, 3, 0, 0, 1 },
25. { 7, 0, 0, 0, 2, 0, 8, 0, 6 },
26. { 9, 6, 0, 0, 0, 0, 2, 8, 0 },
27. { 0, 0, 0, 4, 1, 9, 0, 0, 5 },
28. { 0, 0, 0, 0, 8, 0, 0, 7, 9 }
29. };
30. // Store initial board state
31. initialBoard = new int[SIZE][SIZE];
32. for (int row = 0; row < SIZE; row++) {
33. System.arraycopy(board[row], 0, initialBoard[row], 0, SIZE);
34. }
35. // Create JTextFields for each cell
36. for (int row = 0; row < SIZE; row++) {
37. for (int col = 0; col < SIZE; col++) {
38. JTextField textField = new JTextField();
39. textField.setHorizontalAlignment(JTextField.CENTER);
40. textField.setFont(new Font("Arial", Font.BOLD, 20));
41. if (board[row][col] == 0) {
42. textField.setBackground(Color.GRAY); // Color for empty cells
43. } else {
44. textField.setText(String.valueOf(board[row][col]));
45. textField.setEditable(false);
46. }
47. cells[row][col] = textField;
48. gridPanel.add(textField);
49. }
50. }
51. frame.add(gridPanel, BorderLayout.CENTER);
52. // Create and add the start button
53. startButton = new JButton("Start");
54. startButton.addActionListener(new ActionListener() {
55. @Override
56. public void actionPerformed(ActionEvent e) {
57. new Thread(() -> solveSudoku(board)).start(); // Call solver function in a new thread
58. }
59. });
60. // Create and add the reset button
61. resetButton = new JButton("Reset");
62. resetButton.addActionListener(new ActionListener() {
63. @Override
64. public void actionPerformed(ActionEvent e) {
65. resetBoard(board); // Reset the board to the initial state
66. }
67. });
68. JPanel buttonPanel = new JPanel();
69. buttonPanel.add(startButton);
70. buttonPanel.add(resetButton);
71. frame.add(buttonPanel, BorderLayout.SOUTH);
72. frame.setSize(600, 600);
73. frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
74. frame.setVisible(true);
75. });
76. }
77. private static boolean solveSudoku(int[][] board) {
78. // Backtracking algorithm to solve Sudoku
79. for (int row = 0; row < SIZE; row++) {
80. for (int col = 0; col < SIZE; col++) {
81. if (board[row][col] == 0) { // Find empty cell
82. for (int num = 1; num <= 9; num++) {
83. if (isValid(board, row, col, num)) {
84. board[row][col] = num;
85. cells[row][col].setText(String.valueOf(num));
86. cells[row][col].setBackground(Color.WHITE);
87. // Visualize solving process
88. try {
89. Thread.sleep(80); // Add delay for visualization
90. } catch (InterruptedException e) {
91. e.printStackTrace();
92. }
93. frame.repaint(); // Refresh frame to show updates
94. if (solveSudoku(board)) {
95. return true;
96. } else {
97. board[row][col] = 0; // Backtrack
98. cells[row][col].setText("");
99. cells[row][col].setBackground(Color.RED); // Highlight backtracking cells in red
100. // Visualize backtracking process
101. try {
102. Thread.sleep(80); // Add delay for visualization
103. } catch (InterruptedException e) {
104. e.printStackTrace();
105. }
106. frame.repaint(); // Refresh frame to show updates
107. }
108. }
109. }
110. return false; // No valid number found for this cell
111. }
112. }
113. }
114. return true; // Sudoku solved
115. }
116. private static boolean isValid(int[][] board, int row, int col, int num) {
117. // Check if num can be placed at board[row][col] without violating Sudoku rules
118. return !usedInRow(board, row, num) &&
119. !usedInCol(board, col, num) &&
120. !usedInBox(board, row - row % 3, col - col % 3, num);
121. }
122. private static boolean usedInRow(int[][] board, int row, int num) {
123. for (int col = 0; col < SIZE; col++) {
124. if (board[row][col] == num) {
125. return true;
126. }
127. }
128. return false;
129. }
130. private static boolean usedInCol(int[][] board, int col, int num) {
131. for (int row = 0; row < SIZE; row++) {
132. if (board[row][col] == num) {
133. return true;
134. }
135. }
136. return false;
137. }
138. private static boolean usedInBox(int[][] board, int boxStartRow, int boxStartCol, int num) {
139. for (int row = 0; row < 3; row++) {
140. for (int col = 0; col < 3; col++) {
141. if (board[row + boxStartRow][col + boxStartCol] == num) {
142. return true;
143. }
144. }
145. }
146. return false;
147. }
148. private static void resetBoard(int[][] board) {
149. for (int row = 0; row < SIZE; row++) {
150. for (int col = 0; col < SIZE; col++) {
151. board[row][col] = initialBoard[row][col];
152. if (initialBoard[row][col] == 0) {
153. cells[row][col].setText("");
154. cells[row][col].setBackground(Color.GRAY);
155. cells[row][col].setEditable(true);
156. } else {
157. cells[row][col].setText(String.valueOf(initialBoard[row][col]));
158. cells[row][col].setBackground(Color.WHITE);
159. cells[row][col].setEditable(false);
160. }
161. }
162. }
163. frame.repaint();
164. }
165. }