LOVELY PROFESSIONAL UNIVERSITY

PROJECT REPORT

SUDOKU SOLVER VISUALIZER

SUBMITTED TO : Mr. RAHUL SINGH

NAME	REG NO	SECTION	ROLL NO
MOHAMED IJAS	12221580	9SK02	R9SK02A39

ABSTRACT

The Sudoku Solver Visualizer is an interactive Java application designed to solve Sudoku puzzles while providing a real-time visual representation of the solving process. Utilizing a backtracking algorithm, this project serves as both an educational tool and a practical application for understanding and solving Sudoku puzzles. The visualizer not only demonstrates the algorithm's step-by-step process but also highlights the power of visualization in grasping complex computational methods.

INTRODUCTION

The primary objective of the Sudoku Solver Visualizer project is to create a Java-based application that solves Sudoku puzzles and provides a dynamic visualization of the solving process. This project aims to:

- Demonstrate the working of a backtracking algorithm in solving constraint satisfaction problems.
- Offer an interactive and engaging learning tool for students and Sudoku enthusiasts.
- Showcase the use of Java Swing for developing graphical user interfaces (GUIs) .

DESIGN

The design of the Sudoku Solver Visualizer encompasses several key components:

1. **Graphical User Interface (GUI):**

- **Java Swing:** Used for creating a responsive and platform-independent GUI. The main window is a **JFrame** containing a grid of **JLabels** representing the Sudoku cells.
- **Color Coding:** Different colors indicate the state of each cell during the solving process (cyan for placed numbers, red for backtracked attempts, light gray for initial and final states) .

2. **Algorithm Implementation:**

- **Backtracking Algorithm:** A systematic trial-and-error method to solve the puzzle, placing numbers in empty cells and backtracking when a placement leads to an invalid state.
- **Safety Checks:** The **isSafe** method ensures a number can be placed in a cell without violating Sudoku rules by checking the row, column, and 3x3 subgrid .

CODE

```
import javax.swing.*;

import java.awt.*;



public class SudokuVisualizer extends JFrame {

        private static final int SIZE = 9;

        private JTextField[][] cells = new JTextField[SIZE][SIZE];

        private int[][] board = {

                        { 1, 0, 0, 4, 8, 9, 0, 0, 6 },

                        { 7, 3, 0, 0, 0, 0, 0, 4, 0 },

                        { 0, 0, 0, 0, 0, 1, 2, 9, 5 },

                        { 0, 0, 7, 1, 2, 0, 6, 0, 0 },

                        { 5, 0, 0, 7, 0, 3, 0, 0, 8 },

                        { 0, 0, 6, 0, 9, 5, 7, 0, 0 },

                        { 9, 1, 4, 6, 0, 0, 0, 0, 0 },

                        { 0, 2, 0, 0, 0, 0, 0, 3, 7 },

                        { 8, 0, 0, 5, 1, 2, 0, 0, 4 }

        };

        public SudokuVisualizer() {

                setTitle("Sudoku Solver");

                setSize(600, 600);

                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                setLayout(new BorderLayout());
```

```
JPanel sudokuPanel = new JPanel();
sudokuPanel.setLayout(new GridLayout(3, 3, 5, 5));
for (int blockRow = 0; blockRow < 3; blockRow++) {
    for (int blockCol = 0; blockCol < 3; blockCol++) {
        JPanel blockPanel = new JPanel();
        blockPanel.setLayout(new GridLayout(3, 3));
        blockPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        for (int row = blockRow * 3; row < blockRow * 3 + 3; row++) {
            for (int col = blockCol * 3; col < blockCol * 3 + 3; col++) {
                cells[row][col] = new JTextField();
                cells[row][col].setHorizontalAlignment(JTextField.CENTER);
                if (board[row][col] != 0) {
                    cells[row][col].setText(String.valueOf(board[row][col]));
                    cells[row][col].setEditable(false);
                    cells[row][col].setBackground(Color.LIGHT_GRAY);
                }
                blockPanel.add(cells[row][col]);
            }
        }
        sudokuPanel.add(blockPanel);
    }
}
JButton solveButton = new JButton("Solve");
solveButton.addActionListener(e -> solveSudoku());
```

```
        add(sudokuPanel, BorderLayout.CENTER);

        add(solveButton, BorderLayout.SOUTH);

    }

    private void solveSudoku() {

        SudokuSolver solver = new SudokuSolver(cells);

        new Thread(() -> {

            if (solver.solveSudoku(board)) {

                JOptionPane.showMessageDialog(this, "Sudoku Solved!");

            } else {

                JOptionPane.showMessageDialog(this, "No solution exists");

            }

        }).start();

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> {

            SudokuVisualizer frame = new SudokuVisualizer();

            frame.setVisible(true);

        });

    }

}

class SudokuSolver {

    private static final int SIZE = 9;

    private static final int EMPTY = 0;

    private JTextField[][] cells;
```

```
public SudokuSolver(JTextField[][] cells) {
    this.cells = cells;
}
public boolean solveSudoku(int[][] board) {
    for (int row = 0; row < SIZE; row++) {
        for (int col = 0; col < SIZE; col++) {
            if (board[row][col] == EMPTY) {
                for (int num = 1; num <= SIZE; num++) {
                    if (isValid(board, row, col, num)) {
                        board[row][col] = num;
                        updateUI(row, col, num);
                        if (solveSudoku(board)) {
                            return true;
                        } else {
                            board[row][col] = EMPTY;
                            updateUI(row, col, EMPTY);
                        }
                    }
                }
                return false;
            }
        }
    }
```

```
        return true;
    }
    private boolean isValid(int[][] board, int row, int col, int num) {
        for (int i = 0; i < SIZE; i++) {
            if (board[row][i] == num || board[i][col] == num
                    || board[row - row % 3 + i / 3][col - col % 3 + i % 3] == num) {
                return false;
            }
        }
        return true;
    }
    private void updateUI(int row, int col, int num) {
        SwingUtilities.invokeLater(() -> cells[row][col].setText(num == EMPTY ? "" :
String.valueOf(num)));
        try {
            Thread.sleep(50);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

IMPLEMENTATION

The technical implementation is divided into several parts:

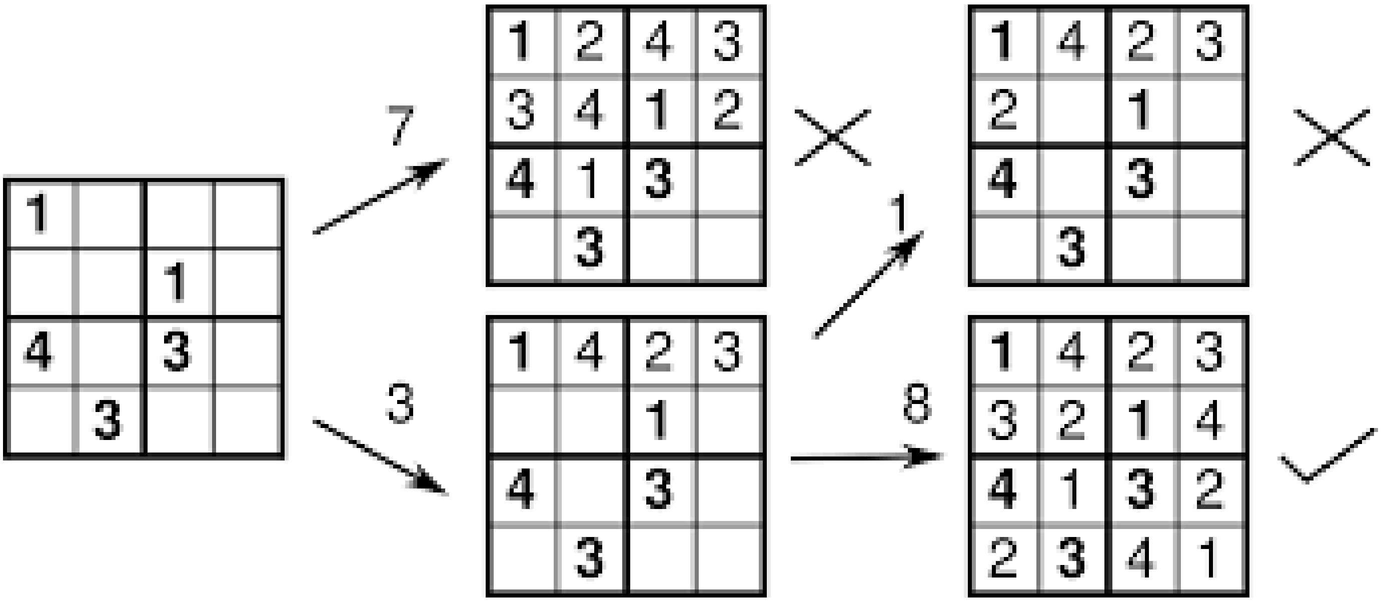
1. **Programming Language and Framework:**

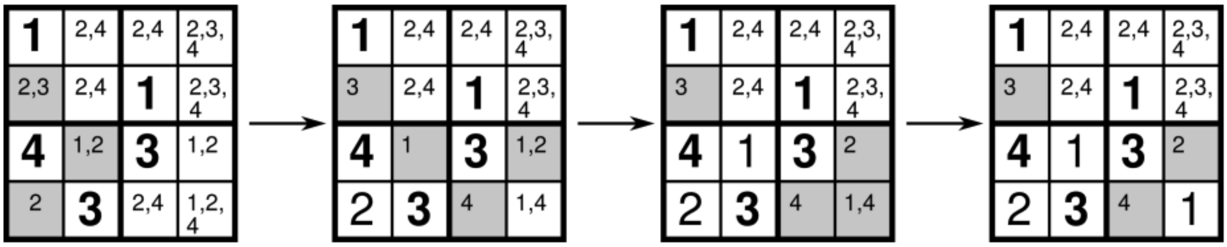
- **Java:** The application is compatible with Java 8 and above, utilizing standard libraries (**java.awt**, **javax.swing**) for GUI components.
- **Java Swing:** The GUI is constructed using **JFrame** and **JLabel** components, arranged in a **GridLayout** for the 9x9 Sudoku grid .

2. **Core Components:**

- **SudokuSolver Class:** Manages both the solving algorithm and the GUI, with methods like **findSolution** (for the backtracking process) and **isSafe** (for safety checks).
- **GUI Elements:** Includes initialization of the **JFrame**, configuration of **JLabel** components, and real-time updates during the solving process .

DEMONTRATION





CONCLUSION

The Sudoku Solver Visualizer successfully achieves its goals by providing an effective Sudoku-solving tool and an educational platform for understanding backtracking algorithms. Key accomplishments include:

- **Functional Sudoku Solver:** Efficiently solves 9x9 Sudoku puzzles using a backtracking algorithm.

- **Effective Visualization:** The real-time visual representation helps users understand the algorithm's decision-making process.
- **Educational Value:** Serves as a practical demonstration of constraint satisfaction problems and backtracking algorithms .

FUTURE WORK

Potential areas for future development include:

- **User Interaction:** Adding capabilities for user input of custom puzzles and control over the solving process.
- **Algorithm Diversity:** Incorporating additional solving techniques for comparison and educational purposes.
- **Scalability:** Extending support for different grid sizes and non-square grids .

BIBLOGRAPHY

https://www.fi.muni.cz/~xpelanek/publications/sudoku-arxiv.pdf

https://chatgpt.com/g/g-1YVVeimiK-sudoku-solver-visualizer/c/f335bee1-b112-4a88-9d45-5aad7ec037

https://codeforces.com/blog/entry/98543?locale=en