Project Report
On
Sudoku Solver Visualizer

School of Computer Science And Engineering

Lovely Professional University

Submitted to-Mr Rahul Rajput

Name-NANDINI RAJ

Registration No.-12206523

Table of Contents

Report on Sudoku Solver Visualizer

Introduction

Sudoku is a popular logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. A Sudoku Solver Visualizer is a tool that not only solves Sudoku puzzles but

also provides a visual representation of the solving process. This report discusses the importance, features, implementation, and benefits of a Sudoku Solver Visualizer.

Importance of a Sudoku Solver Visualizer

A Sudoku Solver Visualizer serves multiple purposes:

1. **Educational Tool**: It helps beginners understand the strategies and logical steps involved in solving Sudoku puzzles.
2. **Debugging Aid**: For developers working on Sudoku-solving algorithms, a visualizer can help identify and correct errors in their code.
3. **Engagement**: Visualization can make the process of solving Sudoku more engaging and interesting for users.

Features of a Sudoku Solver Visualizer

A comprehensive Sudoku Solver Visualizer typically includes the following features:

1. **Grid Display**: A clear and interactive 9x9 grid where users can input their puzzles.
2. **Step-by-Step Visualization**: The ability to display each step the algorithm takes to solve the puzzle.
3. **Algorithm Selection**: Options to choose different solving algorithms like Backtracking, Constraint Propagation, and others.
4. **Speed Control**: Controls to adjust the speed of the visualization to either slow down for detailed observation or speed up for quicker solutions.
5. **Pause and Resume**: Options to pause and resume the visualization for better understanding.
6. **Error Highlighting**: Indicators to highlight any mistakes in the current state of the puzzle.
7. **Hints and Suggestions**: Features to provide hints or suggest possible moves.

Implementation of a Sudoku Solver Visualizer

The implementation of a Sudoku Solver Visualizer can be broken down into several key steps:

1. **Grid Setup**: Creating a user-friendly interface with a 9x9 grid where users can input their Sudoku puzzles. This can be done using libraries like Tkinter for Python or JavaScript for web-based implementations.
2. **Solving Algorithms**: Implementing various solving algorithms. The most common ones are:
    - **Backtracking**: A brute-force approach that tries to place numbers in the grid and backtracks when a contradiction is found.
    - **Constraint Propagation**: Uses rules and constraints to limit the possibilities and iteratively solves the puzzle.
3. **Visualization Logic**: Incorporating a way to visualize each step. This involves updating the grid and possibly highlighting the cell being worked on. Tools like Matplotlib for Python or HTML5 Canvas for web applications can be used.
4. **Control Mechanisms**: Implementing controls for play, pause, resume, and speed adjustments. This ensures that users can follow the solving process at their own pace.

5. **Error Handling**: Adding logic to detect and highlight errors, making it easier for users to understand where mistakes occur.

Java Code Explanation

Implementing a Sudoku Solver Visualizer in Java involves several components:

1. **Grid Representation**

```
java
Copy code
public class SudokuGrid {
    private int[][] grid;

    public SudokuGrid() {
        grid = new int[9][9];
    }

    public int getValue(int row, int col) {
        return grid[row][col];
    }

    public void setValue(int row, int col, int value) {
        grid[row][col] = value;
    }

    public boolean isSafe(int row, int col, int num) {
        // Check if 'num' is not in row, column and 3x3 subgrid
        for (int x = 0; x < 9; x++) {
            if (grid[row][x] == num || grid[x][col] == num ||
                grid[row - row % 3 + x / 3][col - col % 3 + x % 3] ==
num) {
                return false;
            }
        }
        return true;
    }
}
```

2. **Backtracking Algorithm**

```
java
Copy code
public class SudokuSolver {
    private SudokuGrid grid;

    public SudokuSolver(SudokuGrid grid) {
        this.grid = grid;
    }

    public boolean solve() {
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                if (grid.getValue(row, col) == 0) {
                    for (int num = 1; num <= 9; num++) {
                        if (grid.isSafe(row, col, num)) {
                            grid.setValue(row, col, num);
                            if (solve()) {
```

```
                                return true;
                            }
                            grid.setValue(row, col, 0);
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }
}
```

3. **Visualization Logic** The visualization logic involves updating the UI to reflect the current state of the grid at each step. This can be done using a Java GUI framework like Swing.

```
java
Copy code
import javax.swing.*;
import java.awt.*;

public class SudokuVisualizer extends JFrame {
    private SudokuGrid grid;

    public SudokuVisualizer(SudokuGrid grid) {
        this.grid = grid;
        initUI();
    }

    private void initUI() {
        setTitle("Sudoku Solver Visualizer");
        setSize(400, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                g.drawString(String.valueOf(grid.getValue(row, col)),
col * 40 + 20, row * 40 + 40);
            }
        }
    }

    public void visualize() {
        SwingUtilities.invokeLater(() -> {
            setVisible(true);
            repaint();
        });
    }
}
```

4. **User Interface** The UI can be enhanced with buttons for play, pause, and speed control, and a panel to display the solving process.

```
java
Copy code
public class SudokuApp {
    public static void main(String[] args) {
        SudokuGrid grid = new SudokuGrid();
        // Set up initial grid values here

        SudokuSolver solver = new SudokuSolver(grid);
        SudokuVisualizer visualizer = new SudokuVisualizer(grid);

        JFrame frame = new JFrame();
        JButton solveButton = new JButton("Solve");
        solveButton.addActionListener(e -> {
            solver.solve();
            visualizer.visualize();
        });

        frame.add(solveButton, BorderLayout.SOUTH);
        frame.add(visualizer);
        frame.setSize(400, 450);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Benefits of Using a Sudoku Solver Visualizer

1. **Learning Tool**: It helps users learn different solving techniques and improves their problem-solving skills.
2. **Improved Debugging**: Developers can easily identify and fix issues in their algorithms by observing the step-by-step solving process.
3. **Enhanced Engagement**: Visualization makes the solving process more interactive and engaging, attracting a wider audience.
4. **Efficiency**: It allows for quick and efficient solving of complex puzzles, saving time for enthusiasts and researchers.

Conclusion

A Sudoku Solver Visualizer is a valuable tool for both Sudoku enthusiasts and developers. It provides a clear and interactive way to understand and solve puzzles while also serving as an excellent educational and debugging aid. The implementation of such a tool involves setting up a grid, implementing solving algorithms, and creating visualization and control mechanisms. The benefits it offers make it an essential addition to the toolkit of anyone interested in Sudoku.