Project Report

Sudoku Solver Visualizer Project

Submitted by:

Name: Shashi Chawla

Reg.no. 12216715

Sudoku Solver Visualizer

Introduction

Sudoku is a popular logic-based number placement puzzle. The objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 sub grids contain all digits from 1 to 9. This report describes the implementation of a Sudoku solver visualizer using Java and Swing. The visualizer not only solves the puzzle but also provides a step-by-step graphical representation of the solving process.

Concept Behind Sudoku

Sudoku is a puzzle that involves filling a grid with numbers under specific constraints:

- Each row must contain all digits from 1 to 9 without repetition.
- Each column must contain all digits from 1 to 9 without repetition.
- Each 3x3 subgrid must contain all digits from 1 to 9 without repetition.

The puzzle is usually partially completed when presented to the solver, who must fill in the remaining cells.

Approach to Solve Sudoku:

The Sudoku solving algorithm typically used is backtracking, a recursive technique that tries to build a solution incrementally. The main steps of the approach are:

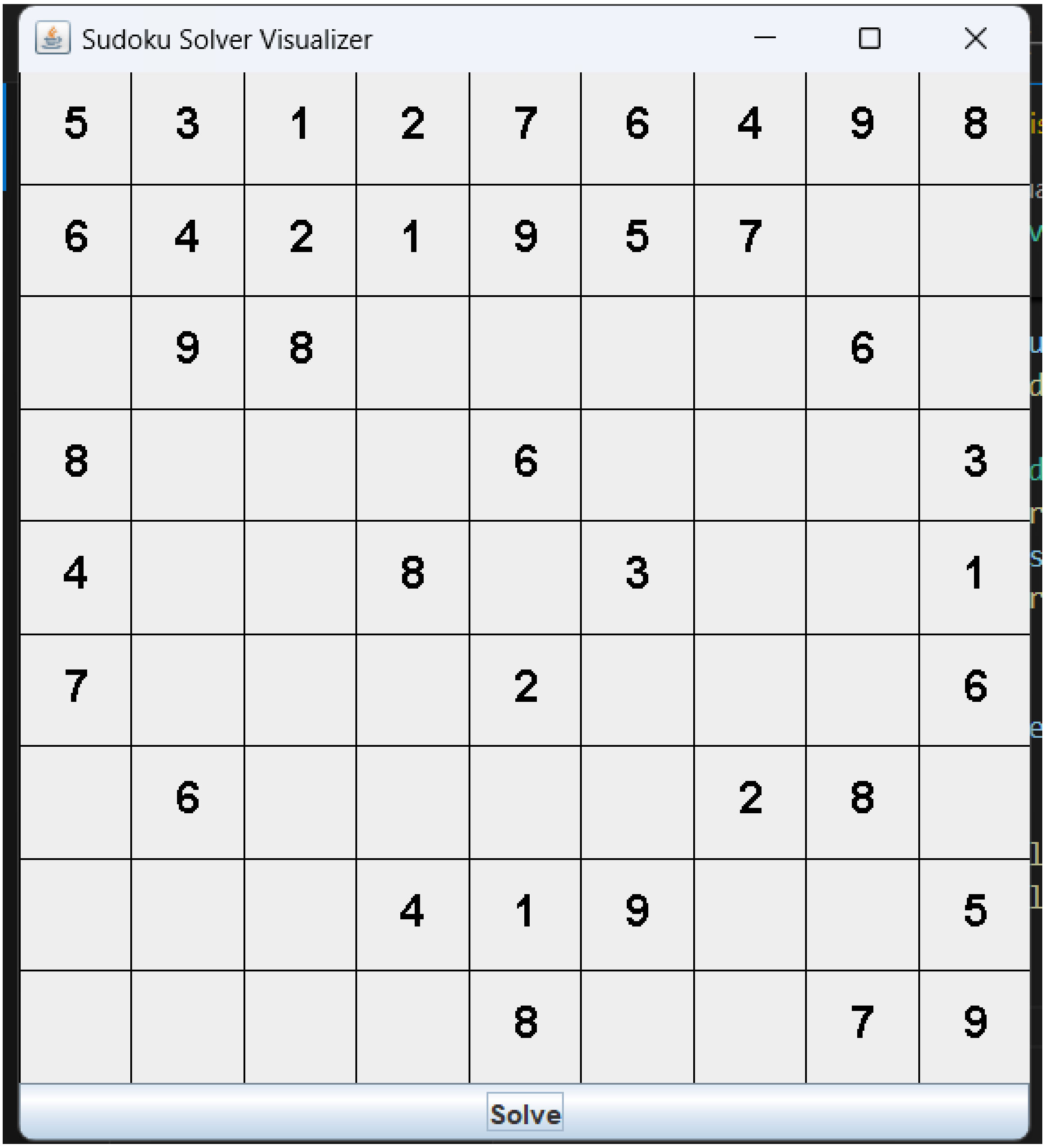
1. **Find the Next Empty Cell:**
    - Traverse the grid to find the next cell that is empty (contains 0).

2. **Try Possible Numbers:**
    - For each empty cell, try placing each number from 1 to 9.
    - Check if placing the number violates any Sudoku constraints (row, column, or subgrid).
3. **Check Constraints:**
    - Ensure that the number does not already appear in the current row, column, or 3x3 subgrid.
4. **Recursive Call:**
    - If a valid number is placed, recursively attempt to solve the rest of the grid.
    - If placing a number leads to a conflict later, backtrack by resetting the cell and trying the next number.
5. **Visualization:**
    - To visualize the solving process, update the grid and repaint the GUI after each placement.
    - Introduce a delay to observe the changes step-by-step.
6. **Completion:**
    - If the entire grid is filled without conflicts, the puzzle is solved.
    - If no valid number can be placed in an empty cell, backtrack to the previous cell.

Source Code:

```
package visualizer;


import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class SudokuSolverVisualizer extends JPanel {
  private static final int SIZE = 9;
  private int[][] board;
  private boolean solved;
  private Timer timer;


  public SudokuSolverVisualizer(int[][] board) {
    this.board = board;
    this.solved = false;
    setPreferredSize(new Dimension(450, 450));
  }


  private boolean solveSudoku() {
    for (int row = 0; row < SIZE; row++) {
      for (int col = 0; col < SIZE; col++) {
        if (board[row][col] == 0) {
          for (int num = 1; num <= SIZE; num++) {
            if (isSafe(row, col, num)) {
```

```java
                            }
                            board[row][col] = num;
                            repaint();
                            try {
                                Thread.sleep(100); // To visualize the process
                            } catch (InterruptedException ex) {
                                Thread.currentThread().interrupt();
                            }
                            if (solveSudoku()) {
                                return true;
                            }
                            board[row][col] = 0;
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }

    private boolean isSafe(int row, int col, int num) {
        for (int x = 0; x < SIZE; x++) {
            if (board[row][x] == num || board[x][col] == num || board[row - row % 3 + x / 3][col - col % 3 + x % 3] == num) {
                return false;
            }
        }
        return true;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLACK);

        for (int i = 0; i <= SIZE; i++) {
            g.drawLine(i * 50, 0, i * 50, 450);
            g.drawLine(0, i * 50, 450, i * 50);
        }

        for (int row = 0; row < SIZE; row++) {
            for (int col = 0; col < SIZE; col++) {
                if (board[row][col] != 0) {
                    g.setFont(new Font("Arial", Font.BOLD, 20));
                    g.drawString(String.valueOf(board[row][col]), col * 50 + 20, row * 50 + 30);
                }
            }
        }
    }

    private void addButton(JFrame frame) {
        JButton solveButton = new JButton("Solve");
        solveButton.setBounds(200, 460, 100, 30);
        solveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                new Thread(() -> solveSudoku()).start();
            }
        });
        frame.add(solveButton);
    }

    public static void main(String[] args) {
        int[][] board = {
            {5, 3, 0, 0, 7, 0, 0, 0, 0},
            {6, 0, 0, 1, 9, 5, 0, 0, 0},
            {0, 9, 8, 0, 0, 0, 0, 6, 0},
            {8, 0, 0, 0, 6, 0, 0, 0, 3},
            {4, 0, 0, 8, 0, 3, 0, 0, 1},
            {7, 0, 0, 0, 2, 0, 0, 0, 6},
            {0, 6, 0, 0, 0, 0, 2, 8, 0},
            {0, 0, 0, 4, 1, 9, 0, 0, 5},
            {0, 0, 0, 0, 8, 0, 0, 7, 9}
        };

        JFrame frame = new JFrame("Sudoku Solver Visualizer");
        SudokuSolverVisualizer visualizer = new SudokuSolverVisualizer(board);
        frame.add(visualizer);
        frame.setSize(500, 550);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(null);
    visualizer.setBounds(0, 0, 450, 450);
    visualizer.addButton(frame);
    frame.setVisible(true);
  }
}
```



5	3	1	2	7	6	4	9	8
6	4	2	1	9	5	7		
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sudoku Solver Visualizer

References:

- ChatGPT: Used for generating and refining the source code and explanations.
- GitHub: Utilized for examples and inspiration for solving and visualizing the Sudoku problem.