PROJECT REPORT
ON

SUDOKU SOLVER VISUALIZER

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

LOVELY PROFESSIONAL UNIVERSITY

SUBMITTED BY:

Name- Dharani Sree.M.C
Registration Number - 12208381
Roll. No – 12
Section-9SK01

1. Introduction

1.1 Background

Sudoku is a logic-based number placement puzzle that has become a favourite pastime for many puzzle enthusiasts worldwide. The objective is to fill a 9x9 grid with digits from 1 to 9 so that each row, each column, and each of the nine 3x3 sub grids contains all the digits from 1 to 9 exactly once.

1.2 Project Scope

The Sudoku Solver project aims to develop a software application with the following capabilities:

- An interactive graphical interface for solving Sudoku puzzles.
- A robust algorithm to solve Sudoku puzzles programmatically.
- Functionality for generating random Sudoku grids for users to solve.

2. Features

2.1 Graphical User Interface (GUI)

2.1.1Grid Display

The main component of the GUI is a 9x9 grid of text fields that users can interact with to input or view Sudoku puzzles. The grid is designed for easy readability and usability.

2.1.2 Control Buttons

The interface includes four main buttons for user interaction:

- Solve: Initiates the puzzle-solving process.
- Stop: Halts the solving process.
- Reset: Clears the grid for a new puzzle.
- Random Fill: Generates a random Sudoku puzzle.

2.2 Functionality

2.2.1 Solver Algorithm

The application uses a recursive backtracking algorithm to solve Sudoku puzzles. This method systematically fills the grid and backtracks when an invalid number placement is encountered.

2.2.2 Random Grid Generator

The application can generate random Sudoku puzzles by filling the grid completely and then removing a specified number of cells to create a puzzle with a unique solution.

2.3 Visual Feedback

The interface provides visual feedback using color coding:

- User Input: White text for numbers entered by the user.
- Random Fill: Green text for numbers generated by the random fill function.
- Solver Process: Yellow text for numbers placed by the solver algorithm.

3. Implementation Details

3.1 Technologies Used

- Java: The primary programming language used for developing the application.
- Swing: The GUI toolkit used for creating the graphical interface.
- Threading: Used to run the solving algorithm in a separate thread, keeping the interface responsive.
- Random: Used for generating random numbers and creating Sudoku puzzles.

3.2 Algorithms

3.2.1 Backtracking Algorithm

The backtracking algorithm is a recursive approach that fills each cell with numbers from 1 to 9, checking for validity at each step. If an invalid placement is detected, it backtracks and tries a different number.

3.2.2 Cell Removal Algorithm

After generating a complete Sudoku solution, the algorithm randomly removes a specified number of cells to create a puzzle with a unique solution.

3.3 GUI Design

The GUI is designed using a 'BorderLayout' for the main components, with the Sudoku grid in the center and the control panel at the bottom. Custom fonts, colors, and borders are used to enhance the visual appeal and usability of the interface.

4. Code Explanation

4.1 Main Class and GUI Setup

The main class extends 'JFrame' and sets up the GUI components, including the grid and control buttons.

SudokuSolver.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

public class SudokuSolver extends JFrame {
    25 usages
    private static final int SIZE = 9;
    17 usages
    private JTextField[][] grid;
    3 usages
    private JButton solveButton, stopButton, resetButton, randomFillButton;
    3 usages
    private volatile boolean solving;

    1 usage
    public SudokuSolver() {
        setTitle("Sudoku Solver");
        setSize( width: 600,  height: 700);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JPanel board = new JPanel(new GridLayout(SIZE, SIZE));
        board.setBorder(BorderFactory.createLineBorder(Color.WHITE,  thickness: 2));
        board.setBackground(Color.BLACK);
        grid = new JTextField[SIZE][SIZE];
        for (int row = 0; row < SIZE; row++) {
            for (int col = 0; col < SIZE; col++) {
                grid[row][col] = new JTextField();
                grid[row][col].setHorizontalAlignment(JTextField.CENTER);
                grid[row][col].setFont(new Font( name: "Arial", Font.BOLD,  size: 20));
```

SudokuSolver.java

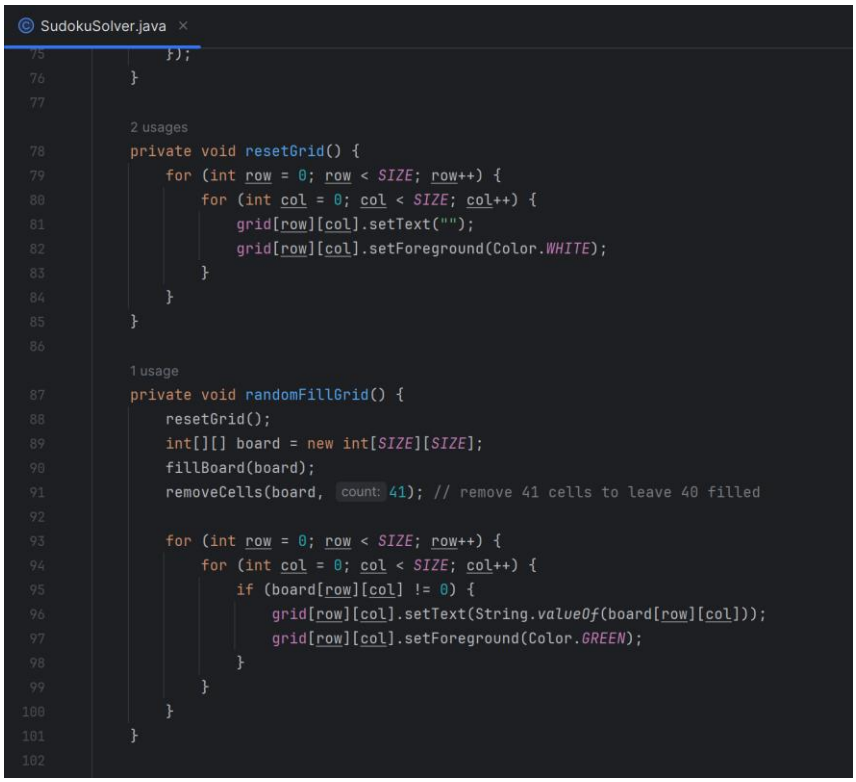
```
        for (int row = 0; row < SIZE; row++) {
            for (int col = 0; col < SIZE; col++) {
                grid[row][col] = new JTextField();
                grid[row][col].setHorizontalAlignment(JTextField.CENTER);
                grid[row][col].setFont(new Font( name: "Arial", Font.BOLD,  size: 20));
                grid[row][col].setForeground(Color.WHITE);
                grid[row][col].setBackground(Color.BLACK);
                grid[row][col].setBorder(BorderFactory.createLineBorder(Color.GRAY));
                board.add(grid[row][col]);
            }
        }
        add(board, BorderLayout.CENTER);

        JPanel controlPanel = new JPanel();
        solveButton = new JButton( text: "Solve");
        stopButton = new JButton( text: "Stop");
        resetButton = new JButton( text: "Reset");
        randomFillButton = new JButton( text: "Random Fill");

        controlPanel.add(solveButton);
        controlPanel.add(stopButton);
        controlPanel.add(resetButton);
        controlPanel.add(randomFillButton);
        add(controlPanel, BorderLayout.SOUTH);

        solveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                solving = true;
                new Thread(new Solver()).start();
            }
        });
```

SudokuSolver.java

```
        controlPanel.add(randomFillButton);
        add(controlPanel, BorderLayout.SOUTH);

        solveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                solving = true;
                new Thread(new Solver()).start();
            }
        });

        stopButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                solving = false;
            }
        });

        resetButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                resetGrid();
            }
        });

        randomFillButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                randomFillGrid();
            }
        });
    }
```

4.2 Grid Reset and Random Fill Methods

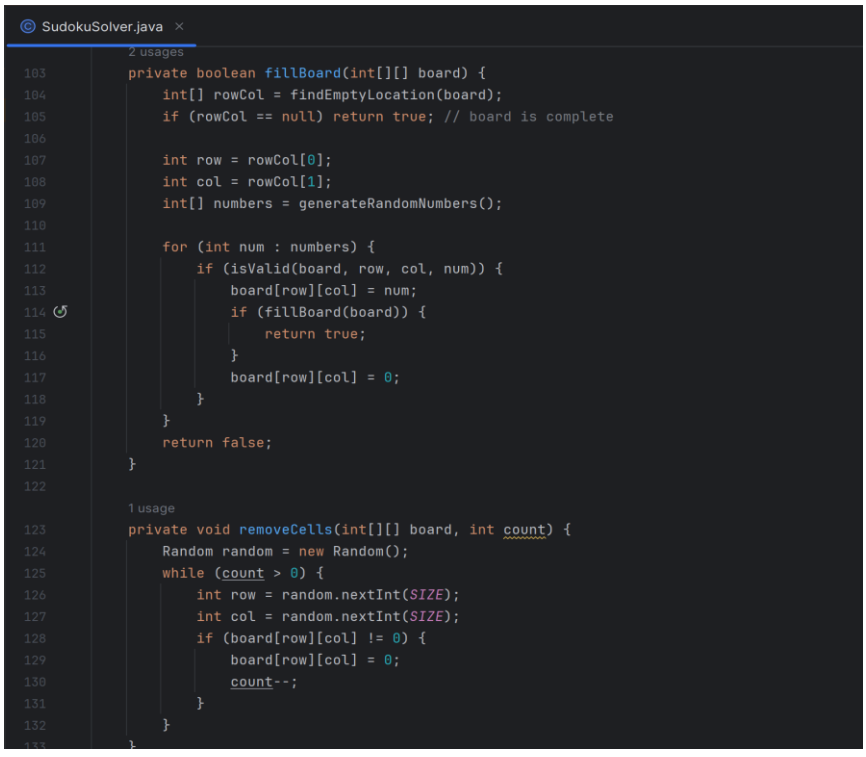
The methods for resetting the grid and filling it with a random puzzle are defined.



```
SudokuSolver.java
        });
    }

    2 usages
    private void resetGrid() {
        for (int row = 0; row < SIZE; row++) {
            for (int col = 0; col < SIZE; col++) {
                grid[row][col].setText("");
                grid[row][col].setForeground(Color.WHITE);
            }
        }
    }

    1 usage
    private void randomFillGrid() {
        resetGrid();
        int[][] board = new int[SIZE][SIZE];
        fillBoard(board);
        removeCells(board, count: 41); // remove 41 cells to leave 40 filled

        for (int row = 0; row < SIZE; row++) {
            for (int col = 0; col < SIZE; col++) {
                if (board[row][col] != 0) {
                    grid[row][col].setText(String.valueOf(board[row][col]));
                    grid[row][col].setForeground(Color.GREEN);
                }
            }
        }
    }
```

4.3 Sudoku Solver Algorithm

The solving algorithm uses backtracking to fill the grid.



```
SudokuSolver.java
    2 usages
    private boolean fillBoard(int[][] board) {
        int[] rowCol = findEmptyLocation(board);
        if (rowCol == null) return true; // board is complete

        int row = rowCol[0];
        int col = rowCol[1];
        int[] numbers = generateRandomNumbers();

        for (int num : numbers) {
            if (isValid(board, row, col, num)) {
                board[row][col] = num;
                if (fillBoard(board)) {
                    return true;
                }
                board[row][col] = 0;
            }
        }
        return false;
    }

    1 usage
    private void removeCells(int[][] board, int count) {
        Random random = new Random();
        while (count > 0) {
            int row = random.nextInt(SIZE);
            int col = random.nextInt(SIZE);
            if (board[row][col] != 0) {
                board[row][col] = 0;
                count--;
            }
        }
    }
```

SudokuSolver.java

```
1 usage
private int[] findEmptyLocation(int[][] board) {
    for (int row = 0; row < SIZE; row++) {
        for (int col = 0; col < SIZE; col++) {
            if (board[row][col] == 0) {
                return new int[]{row, col};
            }
        }
    }
    return null;
}

1 usage
private int[] generateRandomNumbers() {
    int[] numbers = new int[SIZE];
    for (int i = 0; i < SIZE; i++) {
        numbers[i] = i + 1;
    }
    Random random = new Random();
    for (int i = 0; i < SIZE; i++) {
        int j = random.nextInt(SIZE);
        int temp = numbers[i];
        numbers[i] = numbers[j];
        numbers[j] = temp;
    }
    return numbers;
}

1 usage
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) {
```

SudokuSolver.java

```
    }
    int boxRowStart = (row / 3) * 3;
    int boxColStart = (col / 3) * 3;
    for (int r = 0; r < 3; r++) {
        for (int c = 0; c < 3; c++) {
            if (board[boxRowStart + r][boxColStart + c] == num) {
                return false;
            }
        }
    }
    return true;
}
1 usage
private boolean isValid(JTextField[][] grid, int row, int col, int num) {
    String numStr = String.valueOf(num);
    for (int i = 0; i < SIZE; i++) {
        if (numStr.equals(grid[row][i].getText()) || numStr.equals(grid[i][col].getText())) {
            return false;
        }
    }
    int boxRowStart = (row / 3) * 3;
    int boxColStart = (col / 3) * 3;
    for (int r = 0; r < 3; r++) {
        for (int c = 0; c < 3; c++) {
            if (numStr.equals(grid[boxRowStart + r][boxColStart + c].getText())) {
                return false;
            }
        }
    }
    return true;
}
```

4.4 Solver Runnable Class

The ‘Solver’ class implements ‘Runnable’ to solve the Sudoku puzzle on a separate thread.

SudokuSolver.java

```
    private class Solver implements Runnable {
        @Override
        public void run() {
            solveSudoku( row: 0,  col: 0);
        }

        4 usages
        private boolean solveSudoku(int row, int col) {
            if (!solving) {
                return false;
            }
            if (row == SIZE) {
                return true;
            }
            if (col == SIZE) {
                return solveSudoku( row: row + 1,  col: 0);
            }
            if (!grid[row][col].getText().isEmpty()) {
                return solveSudoku(row,  col: col + 1);
            }
            for (int num = 1; num <= SIZE; num++) {
                if (isValid(grid, row, col, num)) {
                    grid[row][col].setText(String.valueOf(num));
                    grid[row][col].setForeground(Color.YELLOW);
                    try {
                        Thread.sleep( millis: 100); // Delay to visualize the solving process
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (solveSudoku(row,  col: col + 1)) {
                        return true;
                    }
```

SudokuSolver.java

```
                if (isValid(grid, row, col, num)) {
                    grid[row][col].setText(String.valueOf(num));
                    grid[row][col].setForeground(Color.YELLOW);
                    try {
                        Thread.sleep( millis: 100); // Delay to visualize the solving process
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (solveSudoku(row,  col: col + 1)) {
                        return true;
                    }
                    grid[row][col].setText("");
                }
            }
            return false;
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new SudokuSolver().setVisible(true);
            }
        });
    }
}
```

5. Usage

5.1 Solving Puzzles

5.1.1 Manual Input

Users can manually enter numbers into the grid using the text fields. The application checks for validity and highlights invalid entries.

5.1.2 Automatic Solving

Clicking the "Solve" button initiates the solving process. The algorithm proceeds step-by-step, filling in cells and providing visual feedback through color changes.

5.2 Interaction

5.2.1 Stopping the Solver

Users can click the "Stop" button to interrupt the solving process at any time.

5.2.2 Resetting the Grid

The "Reset" button clears all entries in the grid, allowing users to start afresh.

5.2.3 Generating Random Puzzles

The "Random Fill" button generates a new Sudoku puzzle with a predetermined number of pre-filled cells, offering a ready-to-solve challenge.

6. Conclusion

The Sudoku Solver project successfully integrates a user-friendly graphical interface with a robust solving algorithm, providing an interactive and educational experience for users. The application not only assists in solving Sudoku puzzles but also offers functionalities for puzzle generation, enhancing its utility for both casual users and puzzle enthusiasts. Future improvements could include advanced solving techniques, enhanced user feedback mechanisms, and support for different grid sizes and puzzle difficulties.

SOME OUPUT SAMPLES