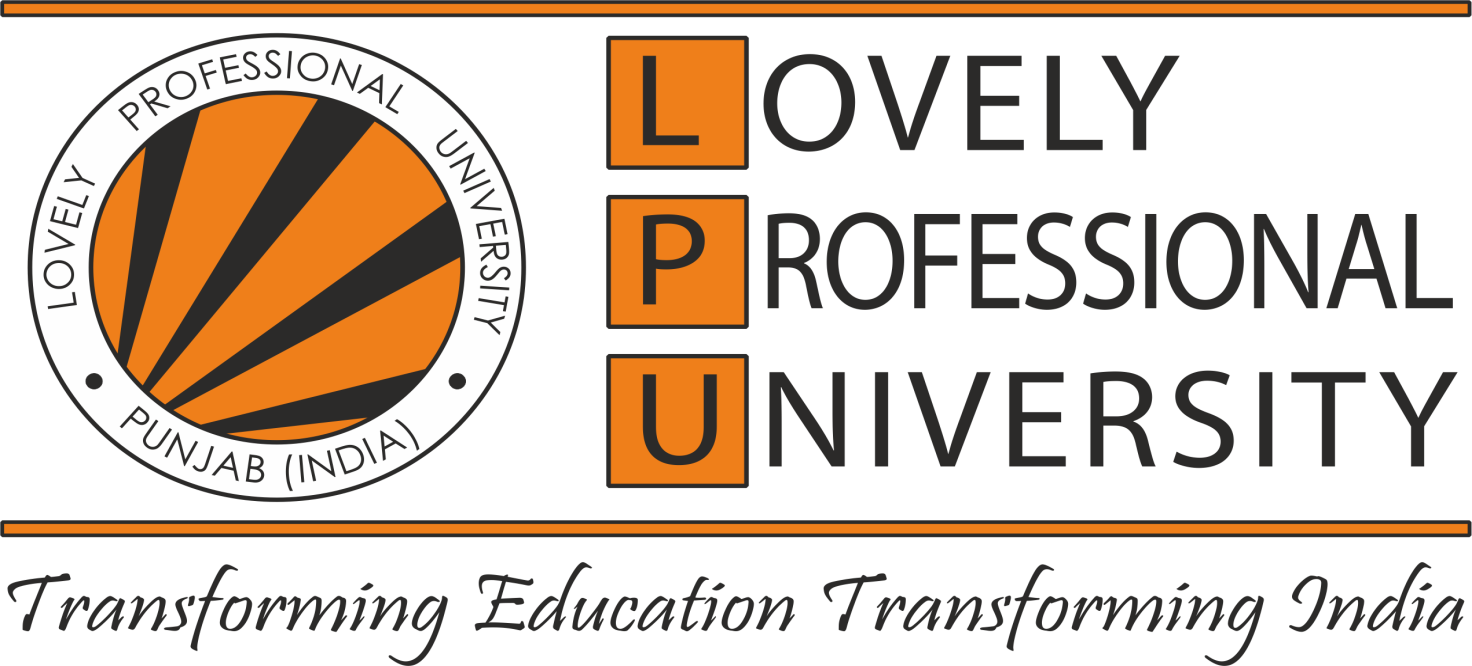
**PROJECT REPORT**



**Project Title - Sudoku Solver**

Name - Abhishek Kumar Maurya

Registration No. – 12223228

Roll No. – 46

Class – 9SK02

Submitted to - Rahul Rajput

Table of contents

1. **Introduction**
   1. Purpose
   2. Key Features
2. **SudokuSolver Class**
   1. isSafe
   2. solveSudokuWithGUI
3. **SudokuGUI Class**
   1. SudokuGUI Constructor Implementation
   2. PaintComponent Implementation
   3. updateBoard Implementation
   4. startSolver Button
4. **Conclusion**
5. **GitHub Repository Link**
6. **Code**

1.**Introduction**

This report covers the implementation and functionality of a Sudoku solver with a graphical user interface (GUI) written in Java.

The project consists of two main classes: SudokuGUI and SudokuSolver.

**1.1. Purpose**

The purpose of this project is to visualize the process of solving Sudoku using Backtracking algorithm and use java gui library like javax, jpanel, etc.

**1.2. Key Features**

* This project key feature is to visualize the process of solving sudoku using backtracking algorithms.
* A start button which initializes the process of visualization.

**2. SudokuSolver Class**

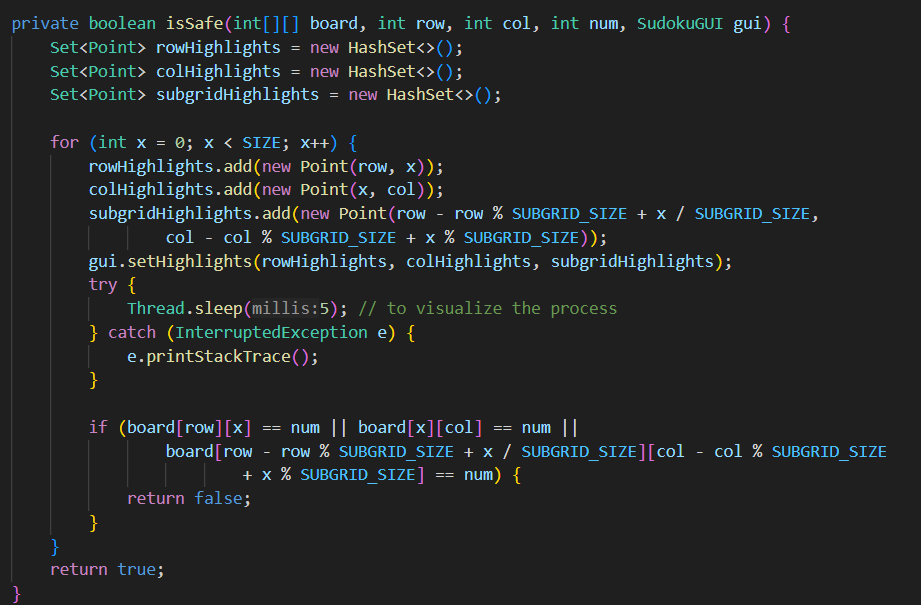
The SudokuSolver class contains the logic for solving the Sudoku puzzle. It uses a backtracking algorithm to find the solution and integrates with the GUI to visualize the solving process.

Key Methods:

* isSafe: Checks if placing a number in a specific cell is valid.
* solveSudokuWithGUI: Solves the puzzle with GUI updates to visualize the solving process.

**2.1 isSafe**

isSafe method checks whether it is safe to place a number from 1 to 9 on that cell by checking entire row, column and 3x3 subgrid.



**2.2 solveSudokuWithGUI**

This method solves the board by recursively checking every number from 1 to 9 on the board and calling isSafe function. This method implements Backtracking Algorithm to solve the board and implement gui part.



**3. SudokuGUI Class**

The SudokuGUI class extends JPanel and handles the graphical representation of the Sudoku board. It creates a window using JFrame and renders the Sudoku grid and numbers.

Key Methods:

* Constructor: Initializes the GUI with the provided Sudoku board.
* updateBoard: Updates the board and repaints the GUI.
* paintComponent: Draws the grid, numbers, and highlighted empty cells.

**3.1. Constructor Implementation**

The SudokuGUI class initializes the JFrame and sets the size and default close operation. The updateBoard method allows the board to be updated and the GUI to be repainted to reflect changes.

**3.2. PaintComponent Implementation**

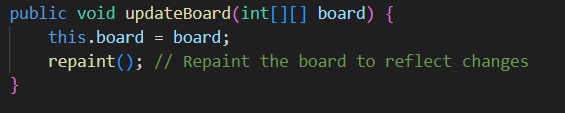
The paint Component method is responsible for drawing the grid. It first fills empty cells with a light gray color and then draws the numbers in the cells. Grid lines are drawn, with thicker lines every three cells to highlight the 3x3 subgrids.

A screen shot of a computer program

Description automatically generated

**3.3. updateBoard Implementation**

This method updates the board every time a change is made so that the changes are reflected while backtracking.

****

**3.4. startSolver Button**

This method start the backtracking algorithm to solve the board using backtracking and visualization process.

**4. Conclusion**

This Sudoku solver with a GUI demonstrates the integration of a backtracking algorithm with a graphical interface, allowing users to visualize the solving process. The combination of SudokuGUI for rendering the board and SudokuSolver for solving the puzzle creates an interactive and educational tool for understanding how Sudoku puzzles can be solved programmatically.

**A colorful squares with red and blue squares

Description automatically generated**

**5. GitHub Repository Link**

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

**6. Code**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.HashSet;

import java.util.Set;

public class SudokuGUI extends JPanel {

    private static final int SIZE = 9;

    private int[][] board;

    private Set<Point> rowHighlights = new HashSet<>();

    private Set<Point> colHighlights = new HashSet<>();

    private Set<Point> subgridHighlights = new HashSet<>();

    private JButton startButton;

    private void startSolver() {

        SudokuSolver solver = new SudokuSolver();

        new Thread(() -> {

            solver.solveSudokuWithGUI(board, this);

        }).start();

    }

    public SudokuGUI(int[][] board) {

        this.board = board;

        JFrame frame = new JFrame("Sudoku Solver");

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.setSize(600, 650);

        frame.setLayout(new BorderLayout());

        this.setPreferredSize(new Dimension(600, 600));

        frame.add(this, BorderLayout.CENTER);

        startButton = new JButton("Start");

        frame.add(startButton, BorderLayout.SOUTH);

        startButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                startSolver();

            }

        });

        frame.pack();

        frame.setVisible(true);

    }

    public void updateBoard(int[][] board) {

        this.board = board;

        repaint(); // Repaint the board to reflect changes

    }

    public void setHighlights(Set<Point> rowHighlights, Set<Point> colHighlights, Set<Point> subgridHighlights) {

        this.rowHighlights = rowHighlights;

        this.colHighlights = colHighlights;

        this.subgridHighlights = subgridHighlights;

        repaint();

    }

    @Override

    protected void paintComponent(Graphics g) {

        super.paintComponent(g);

        int width = getWidth() / SIZE;

        int height = getHeight() / SIZE;

        // Draw cells with different colors

        for (int row = 0; row < SIZE; row++) {

            for (int col = 0; col < SIZE; col++) {

                if (rowHighlights.contains(new Point(row, col))) {

                    g.setColor(Color.RED);

                } else if (colHighlights.contains(new Point(row, col))) {

                    g.setColor(Color.GREEN);

                } else if (subgridHighlights.contains(new Point(row, col))) {

                    g.setColor(Color.BLUE);

                } else if (board[row][col] != 0) {

                    g.setColor(Color.WHITE); // Non-zero cells

                } else {

                    g.setColor(Color.LIGHT\_GRAY); // Empty cells

                }

                g.fillRect(col \* width, row \* height, width, height);

                g.setColor(Color.BLACK);

                g.drawRect(col \* width, row \* height, width, height);

                if (board[row][col] != 0) {

                    g.setColor(Color.BLACK);

                    g.drawString(String.valueOf(board[row][col]), col \* width + width / 2, row \* height + height / 2);

                }

            }

        }

        // Draw grid lines

        g.setColor(Color.BLACK);

        for (int i = 0; i <= SIZE; i++) {

            g.drawLine(i \* width, 0, i \* width, getHeight());

            g.drawLine(0, i \* height, getWidth(), i \* height);

        }

        // Draw thicker grid lines for 3x3 boxes

        Graphics2D g2 = (Graphics2D) g;

        g2.setColor(Color.RED);

        g2.setStroke(new BasicStroke(3)); // Set the stroke width to 3

        for (int i = 0; i <= SIZE; i += 3) {

            g2.drawLine(i \* width, 0, i \* width, getHeight());

            g2.drawLine(0, i \* height, getWidth(), i \* height);

        }

    }

    public static void main(String[] args) {

        int[][] board = new int[][] {

                { 5, 3, 0, 0, 7, 0, 9, 0, 0 },

                { 6, 0, 0, 1, 9, 5, 3, 4, 0 },

                { 0, 9, 8, 3, 4, 0, 5, 6, 0 },

                { 8, 5, 9, 7, 6, 0, 0, 0, 3 },

                { 4, 0, 0, 8, 0, 3, 7, 0, 1 },

                { 7, 0, 0, 0, 2, 0, 8, 5, 6 },

                { 0, 6, 1, 0, 0, 0, 0, 8, 0 },

                { 0, 8, 0, 4, 1, 9, 0, 0, 5 },

                { 0, 4, 5, 0, 8, 0, 0, 7, 9 }

        };

        new SudokuGUI(board);

    }

}

import java.awt.Point;

import java.util.HashSet;

import java.util.Set;

public class SudokuSolver {

    private static final int SIZE = 9;

    private static final int SUBGRID\_SIZE = 3;

    private boolean isSafe(int[][] board, int row, int col, int num, SudokuGUI gui) {

        Set<Point> rowHighlights = new HashSet<>();

        Set<Point> colHighlights = new HashSet<>();

        Set<Point> subgridHighlights = new HashSet<>();

        for (int x = 0; x < SIZE; x++) {

            rowHighlights.add(new Point(row, x));

            colHighlights.add(new Point(x, col));

            subgridHighlights.add(new Point(row - row % SUBGRID\_SIZE + x / SUBGRID\_SIZE,

                    col - col % SUBGRID\_SIZE + x % SUBGRID\_SIZE));

            gui.setHighlights(rowHighlights, colHighlights, subgridHighlights);

            try {

                Thread.sleep(5); // to visualize the process

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            if (board[row][x] == num || board[x][col] == num ||

                    board[row - row % SUBGRID\_SIZE + x / SUBGRID\_SIZE][col - col % SUBGRID\_SIZE

                            + x % SUBGRID\_SIZE] == num) {

                return false;

            }

        }

        return true;

    }

    public boolean solveSudokuWithGUI(int[][] board, SudokuGUI gui) {

        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(board, row, col, num, gui)) {

                            board[row][col] = num;

                            gui.updateBoard(board);

                            try {

                                Thread.sleep(80); // to visualize the process

                            } catch (InterruptedException e) {

                                e.printStackTrace();

                            }

                            if (solveSudokuWithGUI(board, gui)) {

                                return true;

                            }

                            board[row][col] = 0;

                            gui.updateBoard(board);

                            try {

                                Thread.sleep(20);

                            } catch (InterruptedException e) {

                                e.printStackTrace();

                            }

                        }

                    }

                    return false;

                }

            }

        }

        return true;

    }

}