Sudoku Solver Visualizer Using Java

Name : Devisri Prasad

Registration Number : 12220538

GitHub Link :
https://github.com/Devisri-prasad/sudoku-visualizer-

Table Content P.no

1.Introduction

Sudoku, a logic-based number placement puzzle, has become a global phenomenon enjoyed by millions. The challenge involves filling a 9x9 grid so that each column, each row, and each of the nine 3x3 subgrids contain all the digits from 1 to 9 without repetition. The game's allure lies in its simplicity juxtaposed with the mental rigor it demands, making it a favorite among puzzle enthusiasts and casual gamers alike.

Despite its straightforward rules, solving Sudoku puzzles can range from trivial to extremely complex. This complexity provides an excellent platform to explore various algorithmic techniques in computer science, particularly in the realm of constraint satisfaction problems. One such technique, the backtracking algorithm, is a recursive, brute-force approach that incrementally builds candidates for the solutions and abandons a candidate ("backtracks") as soon as it determines that this candidate cannot possibly lead to a valid solution.

2.Problem Statement

The objective of this project is to develop a Java application that not only solves a given Sudoku puzzle but also visually demonstrates the solving process using the backtracking algorithm. Sudoku puzzles, while simple in their rules, can vary significantly in difficulty, requiring sophisticated methods to find solutions efficiently. The problem statement for this project encompasses several core challenges and goals that need to be addressed to achieve a comprehensive solution.

However, the problem extends beyond merely solving the puzzle. The solver must be integrated into a graphical user interface (GUI) that visually represents the solving process in real-time. This visualization aspect introduces several additional requirements and complexities. The GUI must be designed to dynamically update as the backtracking algorithm progresses, highlighting cells as numbers are placed or removed. This real-time feedback is crucial for users to follow the solving process and understand how the algorithm works.

3. Methodology

3.1 Backtracking Algorithm

The core of the Sudoku solver is based on the backtracking algorithm, a well-known technique for solving constraint satisfaction problems. The algorithm follows these steps:

1. **Incremental Construction**: Start with an empty cell and attempt to place a number from 1 to 9 in it.
2. **Validity Check**: After placing a number, check if the placement is valid. This involves ensuring that the number does not violate Sudoku rules—each number must be unique in its row, column, and 3x3 subgrid.
3. **Recursive Search**: If the number placement is valid, move to the next empty cell and repeat the process. If the placement is invalid or if no valid placements are possible for a cell, backtrack by removing the last placed number and trying the next possible number.

4. **Backtracking**: If a cell cannot be filled with any number, backtrack to the previous cell and try the next possible number there. This process continues until the puzzle is solved or all possibilities are exhausted, indicating that the puzzle is unsolvable.

This recursive approach is implemented in Java, leveraging the language’s support for object-oriented programming and recursion.

4. Implementation

The implementation uses Java’s Swing framework to create a user-friendly GUI that visualizes the solving process. The development process includes the following steps:

4.1 Environment Setup

Development Tools: The project is developed using IntelliJ IDEA or Eclipse IDE, both of which offer robust support for Java development.

Java Version: The application is built using Java SE 8 or higher, ensuring compatibility with modern Java features and libraries.

4.2 GUI Design

The GUI is designed to provide an intuitive and interactive user experience. Key components of the GUI include:

- **Grid Panel**: A 9x9 grid of JTextField components represents the Sudoku board. Each cell in the grid corresponds to a cell in the Sudoku puzzle.
- **Control Panel**: The control panel includes buttons to start the solving process, adjust the visualization speed, and display the elapsed time. These controls allow users to interact with the solver and observe the algorithm's progress.

4.3 Key Components

Timer: Displays the elapsed time since the solving process started.

Speed Controls: Allow the user to adjust the speed of the solving visualization.

Solve Button: Initiates the solving process.

4.4 Visualization

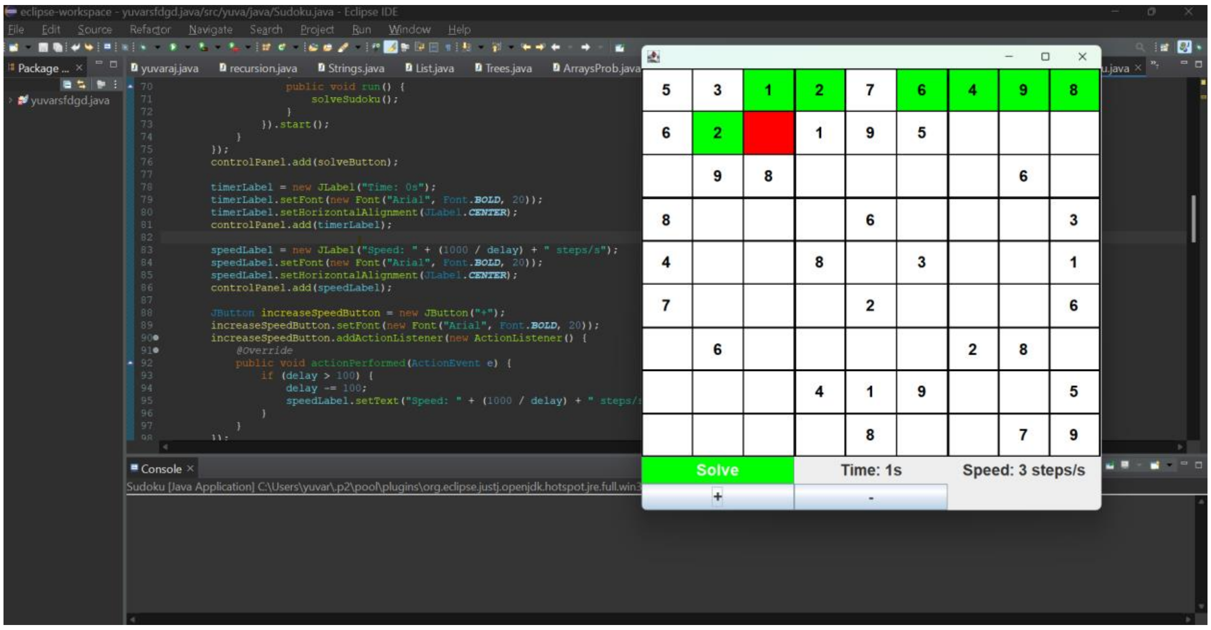
The solving process is visualized by updating the color and content of the cells:

- **Green Cell**: Indicates a cell where a number has been placed.
- **Red Cell**: Indicates a cell where a number placement was undone due to a conflict.
- **White Cell**: Resets the cell to its default state.

5. Conclusion

The Sudoku solver visualizer is an effective tool for understanding how the backtracking algorithm works in real-time. The Java implementation, using the Swing framework, provides a user-friendly interface to interact with the Sudoku puzzle and observe the solving process step-by-step. This project serves as both an educational tool and a practical application for solving Sudoku puzzles.

6. Screenshot



```java
                    public void run() {
                        solveSudoku();
                    }
                }).start();
            }
        });
        controlPanel.add(solveButton);

        timerLabel = new JLabel("Time: 0s");
        timerLabel.setFont(new Font("Arial", Font.BOLD, 20));
        timerLabel.setHorizontalAlignment(JLabel.CENTER);
        controlPanel.add(timerLabel);

        speedLabel = new JLabel("Speed: " + (1000 / delay) + " steps/s");
        speedLabel.setFont(new Font("Arial", Font.BOLD, 20));
        speedLabel.setHorizontalAlignment(JLabel.CENTER);
        controlPanel.add(speedLabel);

        JButton increaseSpeedButton = new JButton("+");
        increaseSpeedButton.setFont(new Font("Arial", Font.BOLD, 20));
        increaseSpeedButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (delay > 100) {
                    delay -= 100;
                    speedLabel.setText("Speed: " + (1000 / delay) + " steps/
                }
            }
        });
```

```
Sudoku [Java Application] C:\Users\yuvar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3
```

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