Report on the development of a SODOKU SOLVER S.E Project

By *El Amrani Ismail*

4 November 2024

1. Introduction

The objective of this project was to design and implement a Sudoku solver using a set of defined deduction rules to progressively solve a given Sudoku grid. The solver operates by applying these rules iteratively to fill in cells based on logical constraints inherent to the game. In cases where the rules are insufficient to completely solve the grid, user intervention is requested to continue the process. The solution was developed in Java, with a focus on modular design and extensibility, adhering to object-oriented programming (OOP) principles.

2. Architecture and Design

The project's design is centered around a modular architecture that ensures separation of responsibilities and ease of extensibility:

-DeductionRule (Abstract Base Class): This serves as a common interface for all specific deduction rules. It defines a method apply that derived classes implement to apply specific deduction logic to the Sudoku grid.

-**DR1, DR2, and DR3 (Specific Rules):** These classes represent distinct deduction strategies. Each one is focused on a specific aspect of Sudoku constraints rows, columns, and sub-grids (3x3 blocks) respectively.

-**SudokuSolver (Main Logic Class):** This class manages the overall solving process, applying the deduction rules in sequence until no further progress is possible or until the grid is completely solved.

-**Main (Program Entry Point):** This class initializes the solver, loads the Sudoku grid from an input file, and oversees the interaction with the user during the solving process.

3. Component Descriptions

The deduction rules are the core of the solving strategy, each targeting a specific structural aspect of the Sudoku grid:

-**DR1 (Row-based Deduction):** This rule focuses on examining the rows of the Sudoku grid. By scanning each row, it identifies which values are already present and determines potential values for empty cells. If a cell has only one valid value based on the row constraints, that value is assigned. This approach leverages a straightforward logical constraint: each digit from 1 to 9 must appear exactly once in each row.

-DR2 (Column-based Deduction): Similar to DR1, this rule applies to columns. It evaluates which values are already present in a column and deduces potential values for empty cells. This method helps to progressively narrow down the possible values for each cell within a column, ensuring compliance with the Sudoku rule that each number must appear once per column.

-DR3 (Sub-grid Deduction): This rule addresses the 3x3 sub-grids within the Sudoku grid. It checks which values are already placed in a sub-grid and attempts to deduce unique values for any empty cells within that sub-grid. The rule ensures that each number appears exactly once in each 3x3 block, further reducing the search space for possible values.

SudokuSolver

The SudokuSolver class encapsulates the main solving logic. It reads the grid from a specified text file, applies the defined deduction rules, and manages the state of the grid during the solving process. The solver uses an iterative approach, repeatedly applying the rules until no further progress can be made. If cells remain unfilled, the solver prompts the user to provide input manually, reflecting the limitation of the current rule set for particularly complex grids.

The SudokuSolver class also includes functionality for printing the grid state, verifying whether the grid is completely solved, and handling user input gracefully.

Main

The Main class serves as the program's entry point. It initializes the SudokuSolver, loads the grid from a file, and controls the overall flow of the program. It provides a simple user interface for displaying the initial and final states of the grid, as well as handling any necessary user interactions.

4. Program Functionality

The Sudoku solver operates in the following manner:

1.The grid is loaded from a text file, where each line represents a row and values are separated by commas. Empty cells are represented by 0.

2.The solver applies the defined deduction rules (DR1, DR2, DR3) in sequence, attempting to fill in cells based on logical constraints.

3.If no further deductions can be made and the grid remains incomplete, the user is prompted to provide a value for a cell.

4.The process repeats until the grid is either completely solved or deemed unsolvable without further user intervention.

This approach provides a balance between automated deduction and user interaction, reflecting the inherent complexity of certain Sudoku puzzles.

5. Results and Testing

The solver was tested with a variety of Sudoku grids to assess its performance and robustness:

-Simple Puzzles: These grids were fully solved using only DR1, demonstrating the efficacy of row-based deductions for straightforward cases.

-Medium Difficulty Puzzles: These required a combination of DR1 and DR2 (column-based deductions) to reach a solution, illustrating the complementary nature of the rules.

-Complex Puzzles: Solving these grids necessitated the use of all three rules, including DR3 for sub-grid deductions. In some cases, user intervention was required to complete the solution, highlighting the limitations of purely rule-based approaches for highly complex puzzles.

-Very Difficult Puzzles: For grids that could not be solved entirely through the rules, the solver correctly identified the need for user input, ensuring that all logical avenues were exhausted before requesting manual intervention.

6. Challenges and Solutions

During the development of this project, several challenges were encountered:

-Optimization of Deduction Logic: Balancing the complexity and efficiency of each deduction rule was a key challenge. The modular structure of the rules allowed for isolated testing and refinement, leading to improved performance without compromising correctness.

-Handling Complex Cases: In cases where the rules alone were insufficient, the decision to involve user input provided a practical workaround without overcomplicating the logic. This approach maintains flexibility while acknowledging the inherent limitations of deterministic rule sets.

7. Conclusion

This project allowed me to create a modular Sudoku solver using a rule-based approach. By applying object-oriented principles, I was able to clearly separate different components and make the solution flexible for future improvements. While the solver works well for many types of puzzles, there is potential to enhance it further with more complex strategies.

I would like to thank *Mr. Régins*, our Software Engineering professor, for assigning this project. It was a valuable and enjoyable experience that improved my coding, design, and problem-solving skills while tackling the challenge of Sudoku solving.