Rapport

SUDOKU SOLVER

Author : MOUSSA BOUDJEMAA Merwan Malik

M1 Informatique - Software Engineering

Contents

1. Introduction

The Sudoku solver project is designed to solve Sudoku puzzles using a series of deduction rules: **DR1**, **DR2**, and **DR3**. Each rule applies progressively more complex techniques to deduce values for empty cells, attempting to solve puzzles from easy to hard. The solver dynamically transitions between rules based on puzzle complexity, with user interaction at each stage if manual input is needed.

2. Project Structure and Design Patterns

Architecture: The project includes main components like `SudokuSolver` (solving logic and state management), `SudokuGrid` (grid storage and management), and `DeductionRule` subclasses (`DR1`, `DR2`, `DR3`).

Design Patterns:

- **Singleton Pattern**: Used in the `SudokuGrid` class to ensure a single, shared instance of the grid across the entire application. This centralizes grid management and keeps all components working with a consistent grid state.
- **Observer Pattern**: Allows `DeductionRule` classes to observe changes in the grid's solved state. When the grid is solved, each rule is notified and displays the corresponding difficulty level, making notifications seamless without tightly coupling the rules to the main solver.

- **Template Method Pattern**: Used in the `DeductionRule` class, defining a general `applyRule` method that each subclass (`DR1`, `DR2`, `DR3`) customizes with its specific solving logic. This standardizes rule application while allowing flexibility in solving techniques.
- **Strategy Pattern**: Allows `SudokuSolver` to select and apply different solving strategies (deduction rules) dynamically, based on puzzle complexity. By treating each rule as an interchangeable strategy, this pattern adds flexibility and simplifies future rule additions.
- **Factory Pattern**: The `RuleFactory` class centralizes the creation of `DeductionRule` objects, making it easy to instantiate different rules based on difficulty. This decouples rule creation from the main solver, simplifying management and future extensions.
- **State Pattern**: Manages transitions between difficulty levels (`EasyState`, `MediumState`, `HardState`) as the solver applies each rule. Each state defines which rule to apply, allowing smooth progression through solving stages based on the puzzle's needs.

3. Program Flow

1. **Initialization**: Loads a Sudoku grid from `sudoku_sample.txt` or a specified file.
2. **Rule Application**:
    - **DR1 (Easy)**: Attempts simple elimination.
    - If DR1 fails, the user chooses to either manually enter values (M) or proceed to the next rule (N).
    - **DR2 (Medium)** and **DR3 (Hard)** follow similar options.
3. **Completion**: If the puzzle is solved, a message indicates the difficulty level (Easy, Medium, or Hard) based on the rule that succeeded. If no rule can solve it, the puzzle is marked as unsolvable.

4. Implementation Details

- **SudokuGrid**: Manages the grid state with a Singleton pattern, providing cell access, grid loading, and validation.
- **DeductionRule**: Abstract class with subclasses (`DR1`, `DR2`, `DR3`), each using a different solving approach.
- **UserInputHandler**: Handles manual inputs, allowing the user to assist rule application.
- **RuleFactory**: Dynamically creates `DeductionRule` instances, centralizing rule creation.
- **SolverState**: Uses states (Easy, Medium, Hard) to manage rule progression and stop when the grid is solved.

5. Deduction Rules

DR1: Basic Deduction Rule

- **Technique**: Simple elimination identifies singleton cells based on unique candidates in rows, columns, and blocks. Solves simple grids by filling cells with only one possible value.

DR2: Medium-Difficulty Deduction Rule

- **Technique**:
  - **Elimination**: Similar to DR1, with hidden pairs and triplets.
  - **Unique Candidate**: Assigns values if a candidate appears in only one cell of a row, column, or block.
  - **Block-Line Interaction**: Eliminates possibilities across blocks based on restricted cells.
  - **Repetition**: Repeats elimination steps until the grid is either solved or DR3 is needed.

DR3: Hard-Difficulty Deduction Rule (Backtracking)

- **Technique**: Uses backtracking, placing numbers and checking validity recursively. If DR1 and DR2 fail, DR3 tries each cell's candidates through trial and error, reverting if necessary.
- **Purpose**: Can handle complex puzzles, marking the grid as hard if it's solved through this method.

6. Testing and Validation

- **Test Cases**: Various grids tested, from simple to complex

-Easy grid solvable by DR1 :

```
1,6,0,8,2,5,7,3,4
5,8,2,4,0,7,9,6,1
7,3,4,0,9,1,5,8,2
3,5,7,1,8,4,2,0,6
2,1,6,9,7,3,0,5,8
9,4,8,5,6,2,3,1,0
6,7,3,2,1,0,8,4,5
4,0,1,7,5,8,6,2,3
8,2,5,3,4,6,1,0,9
```

-Medium grid solvable by DR2 :

```
1,6,0,8,2,5,0,3,4
5,0,2,4,3,0,9,6,1
0,3,4,6,9,0,5,0,2
3,5,0,1,8,4,0,9,6
2,1,6,0,7,3,4,0,8
9,4,8,5,0,2,3,1,0
6,0,3,2,1,0,8,4,5
4,9,0,7,5,8,0,2,3
0,2,5,3,0,6,1,7,9
```

-Hard grid solvable by DR3 :

```
1,6,0,0,0,5,0,3,0
0,0,2,0,0,0,0,6,1
7,0,0,0,0,0,5,0,2
0,5,0,1,0,4,2,0,0
2,0,6,0,7,0,0,5,0
0,4,0,0,0,0,3,0,0
6,0,0,2,0,0,0,0,5
4,0,1,0,0,8,0,0,3
0,2,0,0,0,6,0,0,0
```

-Unsolvable grid :

```
5,1,6,8,4,9,7,3,2
3,0,7,6,0,5,0,0,0
8,0,9,7,0,0,0,6,5
1,3,5,0,6,0,9,0,7
4,7,2,5,9,1,0,0,6
9,6,8,3,7,4,5,2,1
2,5,3,1,8,6,0,7,4
6,8,4,2,5,7,3,0,9
7,9,1,0,3,0,6,0,8
```

- **Expected Output**: Solved grids display difficulty messages (Easy, Medium, Hard); unsolvable grids output an error message.
- **Error Handling**: Manual entry errors halt the solution process, and the program can restart from scratch.

7. Conclusion

- **Summary**: The Sudoku solver effectively uses multiple design patterns to solve puzzles of varying difficulty. It’s modular, flexible, and can be extended with additional rules if needed.
- **Limitations**: Performance may decrease for highly complex puzzles with extensive backtracking.
- **Future Improvements**: Potential additions include advanced solving techniques and optimized performance for high-difficulty puzzles.