1. Sudoku Solver Project Report

By Nicolini Quentin

1.1. Table of Contents

1.2. Introduction

The **Sudoku Solver** is a Java-based application that reads, solves, and evaluates Sudoku puzzles using a combination of automated deduction rules and user interaction. It is designed to handle puzzles of varying difficulty levels and provides detailed feedback during the solving process.

---

2. Problem Statement

The task involved implementing a Sudoku solver with the following features:

- Solve puzzles using three distinct deduction rules (DR1, DR2, DR3).

- Interact with the user when the solver cannot deduce further moves.
- Detect inconsistencies and request a puzzle reset when necessary.
- Evaluate puzzle difficulty based on which rules were applied.

Additional requirements:

- Modular and maintainable code structure.
- GitHub repository with documentation and a detailed report.
- Support for evaluating puzzle difficulty levels.

---

3. Solution Design

3.1. Deduction Rules

Three deduction rules were implemented as subclasses of an abstract `DeductionRule`:

1. **DR1 - Naked Single**: Assigns a value to a cell if it has only one possible candidate.
2. **DR2 - Hidden Single**: Identifies cells where a candidate can only fit in one place within a row, column, or block.
3. **DR3 - Pointing Pair/Triple**: Eliminates candidates from rows or columns when they are confined to a block.

A `DeductionRuleFactory` centralizes rule creation.

3.2. State Management

The solving process uses a **State Pattern**, with the following states:

- **DeductionState**: Applies deduction rules.
- **UserState**: Engages the user for manual input when deduction rules cannot proceed.

State transitions are managed by a `SolverStateFactory`, enabling dynamic behavior changes during execution.

3.3. Grid and Iterators

The grid is represented as a linear array of 81 cells. Each cell can hold a value (1-9) or remain empty (`-1`). Custom iterators (`GridRowIterator`, `GridColumnIterator`, `GridBoxIterator`) facilitate row, column, and block traversal.

3.4. Logging and User Interaction

A `Logger` class provides multi-level logging, including:

- **TRACE**: Detailed debugging information.
- **INFO**: General application progress.
- **SUCCESS**: Highlights successful operations.
- **WARN**: Alerts potential issues.
- **ERROR**: Logs critical failures.

User interaction is handled through a `UserState`, which:

1. Prompts the user for cell positions and values.
2. Validates inputs against the grid's constraints.

---

4. Design Patterns Used

4.1. Factory Pattern

Classes:

- `DeductionRuleFactory`: Creates and manages deduction rule instances.
- `SolverStateFactory`: Dynamically creates solver states.

Purpose: Simplifies object creation, centralizes logic, and supports future extensibility.

4.2. State Pattern

Classes:

- `SolverState` (abstract superclass).
- `DeductionState`: Applies rules automatically.
- `UserState`: Handles manual input when deduction fails.

Purpose: Encapsulates solver behavior into states, enabling dynamic transitions between automatic solving and user interaction.

4.3. Iterator Pattern

Classes:

- `GridIterator` (interface).
- `GridRowIterator`, `GridColumnIterator`, `GridBoxIterator`: Facilitate traversal.

Purpose: Enables consistent and efficient traversal of grid components (rows, columns, blocks).

---

5. Implementation Details

5.1. File Input and Parsing

The application reads puzzles from a text file, where:

- Rows are separated by newlines.
- Cells are separated by commas (`3,8,0,1,0,0,5,9,0`).
- Empty cells are represented by `0` or `.`.

The `SudokuFileProcessor` class handles:

1. File reading and parsing.
2. Initializing grids with starting values.

5.2. Deduction Rule Application

Rules are applied in order of difficulty (DR1, DR2, DR3). If no rule applies and cells remain empty, the solver transitions to `UserState`.

5.3. Difficulty Evaluation

Difficulty levels are determined by:

- The hardest rule required to solve the puzzle:
  - **Easy**: Solved using DR1.
  - **Medium**: Requires DR2.
  - **Hard**: Requires DR3.
  - **Very Difficult**: Cannot be solved by DR3.
  - **Impossible**: Unsolvable due to inconsistencies.

---

6. Challenges Encountered

1. **Efficient Candidate Management**:
   - BitSet was used for fast operations on candidate sets.
2. **Handling User Errors**:
   - Validations prevent incorrect inputs during manual solving.
3. **Dynamic Transitions**:
   - Ensuring smooth transitions between states required careful management of dependencies.

---

7. Conclusion

The Sudoku Solver successfully meets the project requirements, including solving puzzles, user interaction, and difficulty evaluation. Its modular design and use of design patterns make it both robust and extensible.

---

8. References

1. Sudoku Solving Techniques
2. State Pattern
3. Java Iterator Pattern

Authors:
NICOLINI Quentin - 22100103