Jawel BRIKI

Sudoku Solver: A Deduction-Based Approach

Introduction

This report details the development of a Sudoku solver that utilizes a deduction-based approach. Unlike traditional solvers that rely on backtracking algorithms, this project aims to solve Sudoku puzzles using only logical deduction rules. The report will outline the problems encountered during development, present the implemented solution, and elaborate on the design choices made.

Solution and Design Choices

As requested by the exercise, a base class DeductionRule was made, and all our deduction rules (DR1, DR2, DR3) derive from it. The idea is that DeductionRule can be abstract, as it does not contain any definition. In fact, had the language of choice been Java, it could've been an interface, but those don't exist in Python.
Each DR that derives from DeductionRule implements the apply method, which apply the rule exactly once on a grid, and updates the grid and/or the list of candidates of the cells of the grid.

The deduction rules I chose are the following:

- **DR1 Naked Singles:** If only one possible number can fit in a cell, that number is placed in the cell.
- **DR2 Hidden Singles:** If, for a certain number, a given cell is the only one in its "house" (a house is the name given to a row, column or 3x3 block) that has it as a candidate, that number is placed in the cell.
  The difference with Naked Singles is that the cell may have other candidates, while Naked Singles checks for cells with exactly one candidate.
- **DR3 Naked Pairs:** If 2 numbers can fit in 2 cells of the same house, these numbers are removed from the list of candidates of all the other cells in that house.

These rules are applied iteratively until the puzzle is solved or no further deductions can be made.

In terms of design, I opted for the Singleton pattern for my SudokuGrid, since only one grid should exist at all times, and all operations are made on the same grid, the pattern made sense there.

Problems Encountered

While DR1 and DR2 were relatively easy to implement, as soon as I started coding DR3, I realized my program would run into an infinite loop. The reason was the way I first coded my DR1 and DR2, as well as my method to update the list of candidates, didn't consider the fact that a rule could be applied without necessarily having to fill a cell (in fact, DR3 never fills any, it only updates the list of candidates), and was only computing the candidates based on cells already filled.
To resolve this problem, I added an extra check while updating my candidates so that if DR3 was applied, updating candidates wouldn't re-add the previously deleted candidates, while keeping the same logic for all other cells.