CISC 440 - Artificial Intelligence & Robotics
Spring 2020

Assignment 04: Due Tuesday, 3/17 by 11:59pm

For this assignment, you will use a Java constraint solver named "Choco" to generate a random sudoku puzzle (http://www.choco-solver.org). Begin by forking and cloning the provided assignment as described in the "Assignment Setup and Submission" document in Canvas. You will need to finish the implementation of the SudokuGenerator class.

Sudoku

Sudoku PEAS task environment description:

Performance measure
- Time to solve a sudoku puzzle, where a solution contains each digit from 1-9 in each row, column, and 3x3 block

Environment
- 9x9 grid of cells with some number of cells containing fixed values from 1-9

Actuators
- Update the value in a non-fixed cell

Sensors
- Knowledge of entire board

SudokuGenerator (sudoku)

Unlike prior assignments, you are not solving a previously produced puzzle. Instead, you are generating a valid puzzle, so that provided code can remove some number of cells to provide to the player.

generateSudokuPuzzle() : String

This method will use the Choco library to create the required variables and constraints for the constraint solver.

Because we are writing code to generate "solved" sudoku puzzles, we will need 81 variables, one for each cell. The domain for each variable should be 1-9. To accomplish this using choco-solver, this link should help: https://choco-solver.org/docs/modeling/variables/. When creating the variables, you must name them (the first intVar, intVarArray, or intVarMatrix method argument) using the format "cell (#,#)" where the first # is the row and the second # is the column, as in: cell (1,1).

Our constraints will need to indicate that each cell's value must not be present elsewhere in the same row, column, or 3x3 square. To accomplish this using choco-solver, this link should help: https://choco-solver.org/docs/modeling/constraints/. The examples on this page include calls to the "allDifferent" and "arithm" methods on a Model instance. Importantly, for a constraint to be enforced, the "post" method must be called on it. We will not need to worry about "reification".

Use the provided "generateRandomSolution" method to solve the problem and record a "random" solution to the Solution parameter. To do this you will need to create and assign the appropriate variables and constraints to the Model. After "generateRandomSolution" has been called, generateSudokuPuzzle should return solution.toString().

Odds and Ends

- It may be helpful to look at an example or two. Here's an 8-queens example: https://choco-solver.org/tutos/first-example/first-model/.
- To dive deeper into the documentation, this is a good place to start: https://javadoc.io/doc/org.choco-solver/choco-solver/latest/org.chocosolver/org/chocosolver/solver/package-summary.html.
- You can view info on other constraint creation methods here: https://javadoc.io/doc/org.choco-solver/choco-solver/latest/org.chocosolver/org/chocosolver/solver/constraints/IIntConstraintFactory.html.

Your SudokuGenerator.java file will be used along with the other provided files when grading. Your grade be determined largely by the number of tests passed in SudokuGeneratorTest.java. While these tests are a good indicator of correctness, they do not guarantee a fully correct implementation. In other words, passing all tests may not translate to a 100% on the assignment.

Submit your assignment by pushing your code to your gitlab repository fork by the deadline.