In Lab 1, you will complete a program for solving Sudoku puzzles. The initial program is lab1.zip, which you can download from the course web site. This will be done by implementing Sudoku as a CSP (see Chapter 4 of the textbook) and performing arc consistency and domain splitting. Initial code is provided for you on the course web site. Your grade on the lab will depend on the performance of your program when I run it on test problems. Your submission should also include a short description of how your program performs.

Sudoku

Do I really need to explain Sudoku? See http://en.wikipedia.org/wiki/Sudoku if you need an explanation.

Environment

The Sudoku environment is defined in Sudoku.java. It reads the puzzles from puzzles.txt and sends them to the agent. Each puzzle in puzzles.txt is formatted as a 81-character string. The environment expects to read a solution from the agent as an 81-character string, which presumably replaces the underscores with the numbers of the solution. The Sudoku environment then checks the result and prints success or failure to the console.

When there are no more puzzles, the Sudoku environment prints quit and finishes.

Interact

Interact.java is used to connect the Sudoku environment to a Sudoku agent. It does this by running the environment and the agent in different threads. Both the environment and the agent need to implement the Runnable interface.

Interact.java creates Java pipes so that the environment and agent have their own input and output streams. It monitors these streams so that any line output by one will be input to the other. Interact.java also prints to the console anything output by the environment and agent. The environment and agent can use System.out to print anything to the console.

The picknumber program is an example of how Interact.java works with an environment and an agent. You can download picknumber.zip from the course web site.

Agent

Your task is to complete an agent program (Solver.java) to interact with the Sudoku environment. Your program should read in a puzzle, find and print out a solution, and repeat. Solver.java is already coded to read in puzzles and quit when it is supposed to, but it does not print out solutions.
The way Solver.java should find a solution is by formulating the problem as a constraint satisfaction problem (see Chapter 4 of the textbook), and applying the generalized arc consistency algorithm (see Section 4.5) and domain splitting (see Section 4.6). Domain splitting is a form of depth-first search (see Section 3.5.1).

Solving the Sudoku puzzles by any other method will earn a zero for this assignment.

CSPTest.java provides very simple examples of CSPs and shows the result of the GAC algorithm (implemented by the consistency method). Note that the GAC algorithm is distributed over the Constraints, that is, each Constraint applies arc consistency to its Variables. This will be covered in more detail in class.

For a Sudoku puzzle, a CSP needs to have 81 Variables and 810 Constraints. You can use Variable.java for Variables, but you should create a new class ConstraintNotEqual.java for Constraints. This class does not need to assume that the values are Integer objects; instead it should use the equals method to test whether any two values are equal.

A puzzle either assigns a specific value to a Variable or leaves the value unknown, with the possible values as the digits from 1 to 9. Each Constraint should specify that its Variables are not equal.

Without doing any domain splitting, GAC is enough to solve the first 10 puzzles in puzzles.txt. This should allow you to debug your CSP and ConstraintNotEqual.java.

When doing domain splitting, you need to be careful to keep a copy of the CSP after arc consistency has been applied, so that each recursive call starts with the same possible values of the Variables (except the selected Variable of course). With domain splitting, no puzzle should take longer than a second to solve. The instructor’s implementation takes about one second to solve all 20 puzzles.

Turning in Your Lab

Submit the folder containing all your Java files to Blackboard. Your program should run under the following conditions: after all the .class files are deleted, then:

```
javac Interact.java
java Interact
```

should solve the puzzles.

The score of your lab will primarily depend on the performance of the program. In addition to the puzzles in puzzles.txt, your program will be tested on an additional 20 puzzles, all them harder than the original 20. Your score will be the percentage of puzzles that you solve.