COSC 370 – Artificial Intelligence
Project 2

Purpose: Use gradient ascent/descent algorithms to solve two classic problems.

Task: For this project, you will use two algorithms - hill-climbing and simulated annealing - to solve two problems. This will require you to code the following:

- Representations of the problems.
- Evaluative functions.
- The algorithms. This will include determining the initial temperature and cooling function for the simulated annealing functions. You will do both algorithms for both problems.
- Display functions.
- Driver function to run your code based on information below (no driver will be provided for this project).

The two problems:

The n-Queens Problem: On an n x n chessboard, place n Queens such that no Queen can attack any other Queen.

Sudoku: Traditionally, a 9x9 grid divided into 9 3x3 distinct regions where each region, row, and column have the numbers 1 through 9, non-repeating. We will be adjusting this problem to be the *Sudoku the Giant* variant published by Nikoli that features a 25x25 grid, divided into 25 5x5 regions. For this problem, I will be providing several text files with solvable puzzles.

Your code should ask which of the two problems that the user would like solved. If the user chooses the n-Queens problem, you should prompt the user for the number of Queens that the solver should target. This should handle Queens up to 200 in number. If the user chooses the Sudoku problem, you should prompt the user for the filename of text file that contains the problem to be solved. You should also ask if the user wants verbose output or not. You may assume that the user will not enter nonsensical input, and that the text file will be formatted correctly.

Your code should then generate the initial state of your problem, display it in some clear manner (including the evaluation value), then calculate the neighboring move to be taken. Display each move, including the evaluation value. **Your program should use your hill-climbing algorithm first, then reset back to the initial state and use simulated-annealing.** At the end of the program, you should output the final states reached by each algorithm, whether or not the algorithms reached an optimal state, and how long it took to get to that final state. Note – since simulated annealing is a form of hill-climbing, it would be strategic to work on the hill-climbing algorithm first and get everything working for hill-climbing, then move on to simulated annealing.

If the user asks for verbose output, output **all** neighbors that were considered before moving, with the neighbor moved to as the last neighbor output. Yes, I know that this is quite a lot of stuff to display.

You are required to work in teams of 4 for this project. Teams will be assigned during class on 2/18. For the purposes of this project, you are free to use Java 10+, C++, Python 3.5+, or Common LISP. If you are using C++, be sure to check for compilation against g++ on the CS server (let me know if you don't have access through 251). **There will be a mandatory team evaluation rubric to be filled out by the end of the project**. This rubric will be available via the course website and Blackboard. Team evaluations can adjust grades up to 20% and turning in the rubric on time will be worth 10 points.

Learning Targets: hill-climbing and simulated annealing implementation

DUE: March 4th at 11:59pm via Blackboard. Team evaluation rubrics due at the same time via email.

Rubric:

-75 Does not compile/interpret/etc.
-50 Does not run to completion
-15 excessively long run-time (over 20 minutes)
-25 missing/incomplete algorithm (hill climbing and simulated annealing for n-Queens and sudoku)
-15 missing/incomplete verbose output option
-15 missing regular output (timing, final states, optimal check)
-15 does not reset to initial state correctly between algorithm runs
-10 lack of user input file I/O for Sudoku problem
-10 did not turn in team evaluations on time (3/4 at 11:59pm via email)
20% adjustment possible due to team evaluations