Chris Saba

CS4200 Project 2

N-Queens (n = 8) OR 1 < n < ~~15

For my approach, I did the Hill Climbing algorithm, with ability to set the maximum allowed times the hill starting point randomizes, also known as “bad moves”. This algorithm is repeated 100 times to give an average of solutions found given in the output. Along with HC, I also implemented the genetic algorithm, which takes population, generations, and mutation probabilities as inputs. This was very interesting to see how well the genetic algorithm works. Both algorithms are given their own GUI chess boards, generated by Queens.java; as well as placed queen icons that are located in the package. This icon must be changed in the location of the Queens.java so that it will show in the GUI.

My analysis comes as follows; In terms of Hill Climb, I noticed that the accuracy (solvable percentage) is lower with 10-40 max iterations; being around 20-30% while anything above 100 almost guarantees 100% accuracy. If the genetic algorithm takes too long to compute, you may want to allow more generations to maximize the ability of finding a solution. Total for n=8, there are 92 possible unique solutions, and you will find that each time the program runs, you will most likely see a new solution. It was very amusing to see that I was able to configure an n size of 15, being able to solve quite easily with hill climb, and almost unsolvable with genetic. In terms of speed, this can be optimized to find solutions faster, with different inputs on the genetic side, but there are too many factors of randomness to pinpoint a direct solution.

Some questions can be answered in the README file located in the source folder. This will show you how to run the program and be able to see the results and analytics as shown.