N Queen Report

The implementation of the project was very interesting to me. I first started with the implementation of the board itself. The board was quite easy to implement, because all I had to do was generate random arrays of N size and place them together in the boardstate class. The class once constructed would contain how many collisions were found and whether or not the boardstate was solved yet. Detecting the collisions was simple as I just looped over all the indices to come with the final amount of collisions. I also had to implement a method that would generate a brand new board if there was a random change.

To implement the simulated annealing, I started with the first state that was randomly generated. I then had to adjust what the temperature and cooling rate, and the stopping temperature was going to be. I also had to detect what the probability was to accept an unoptimized move, which was quite difficult. In the end I came up with numbers that will fail some of the time but not all of the time.

To implement the genetic algorithm, it was very difficult to determine how to randomly select the parents. So I made my own compare function to determine which order the parents were to be chosen by the priority queue. I killed off half the bad population right off the bat with each generation. And then generate the children. I restore the children and the parents that were chosen to the population to come up with a full population all over again. This ensures that the most fit from the previous generation along with the kids are up for being chosen when creating the next generation of breeding pairs. This isn't exactly how it was stated in the pseudo code, but this was the best I could think of for the moment.

The simulated annealing algorithm had the most variables to play around with. It was interesting to tinker with the cooling rate. I found that the cooling rate, when lowered, would produce a result more often, but would also make the algorithm slow down because it was making more bad moves in the beginning. Messing with the initial temperature had the same affect when it was started higher. The stopping temperature didn't play much of a role in how fast or slow the algorithm was. It only made the algorithm run for longer before failing or finding a solution.

The genetic algorithm was also interesting to play around with the limit of generations that I allowed, and the amount of mutations. I felt like 30% mutation rate was a pretty good rate. But what was really interesting, was that there was a point where adding more to the population wasn't doing anything but slowing down the algorithm itself. It is much better to just have a limit on how large your population is, because the best will appear at the same rate after about 20 population size.