Assignment 2

Due Mar 8 by 11:59pm **Points** 60 **Submitting** a file upload
Available after Feb 21 at 12am

```
Submit your non-programming answers in a PDF document [TYPESET] along with the code (.java/cpp/py
/c) files together in a zip. Please include a brief description of how to run your program in the
PDF file.
```

Analytical Questions

1. In class, we discussed the problem of placing *k* queens in an *n x n* chessboard. In this problem, you will be considering the problem of placing *k* knights in a *n x n* chessboard such that no two knights can *attack* (it **moves** to a square that is two squares away horizontally and one square vertically, or two squares vertically and one square horizontally) each other. *k* is given and $k <= n^2$. Formulate this problem as a Constraint Satisfaction Problem. [**20 points**]
   - What are the variables? [5 points]
   - What is the set of possible values for each variable? [5 points]
   - How is the set of variables constrained? [10 points]

2. Explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining in a CSP search. [**10 points**]

Programming Question

1. **[30 points]** Solve an **N-queens problem using a standard genetic algorithm** (Fig. 4.8 in your book). You are recommended to play with different variable values and research different ideas before finalizing them. You will first have to design the related functions for this. You can use the example given in the book (Figs. 4.6 and 4.7) as your starting point. More specifically,
   - You will ask the user for the value of N (assume 4 <= N <= 20).
   - Run the genetic algorithm code for that setting.
   - Finally, display the following information (either on the console or using a GUI):
     - The final solution, i.e., placements of queens (a simple tic-tac-toe-like visualization with _'s and Q's would be enough, e.g., _ _ Q _).
     - Which generation the solution was found?
     - Maximum fitness was achieved in all the past generations along with the solutions displayed with those highest fitness values.
     - Run time to find the final solution.

- Keep asking for N until the user wants to stop.