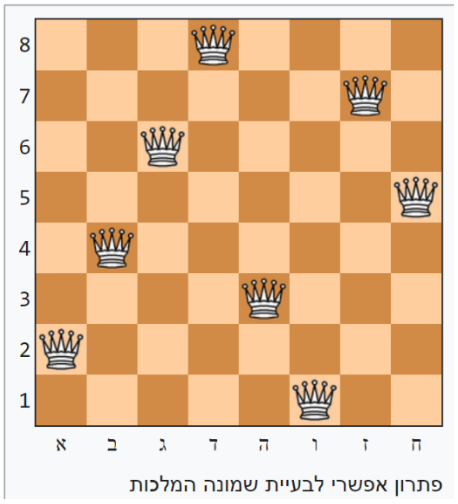
HW #2 – Solving the n-queen problem

We have discussed the 8-queens problem (https://en.wikipedia.org/wiki/Eight_queens_puzzle) in class.

Now is your time to have some fun with it!!!



I will give you a base code, dfs_queen, for the n-queen problem that uses DFS with backtracking. It works, but isn't nearly as efficient as many other possibilities that we'll discuss in class. Your homework is to implement two ways of solving the problem: one that uses search ordering heuristics to aid in the search (like forward checking) and is based on this code and a second that uses randomization for all elements and then a heuristic repair method (like hillclimbing) to find a solution based on that initial, random placement. I will then ask you to compare the two and let me know which one worked best over 100 runs. The first method is more likely to be deterministic and the second method will have stochastic elements making the distribution (and its average) more important (right?!). Pedagogic node: you could say that the methods that use heuristic repair think more "humanly" (at least more like *me*) and those using search ordering heuristics are more "rational".

The code that I provided already has two counters:

```
number_of_iterations, number_of_moves
```

which counts the number of iterations and queen moves needed to find the solution.

Your goal: make this number as low as possible **without *adding additional loops*.**

You **are** encouraged to make the next_row_is_safe function more sophisticated in the second method. Any iteration with the placement of a queen (even a randomly placed one) is considered an iteration, **including** for the first (or subsequent) placement of **all** n queens. I implemented one possibility place_n_queens which implements this idea and places n queens randomly. Note that the two implementations will likely need some tweaks to the code I provided.

Assignment: Compare 4 implementations and their counters:

1. The British Museum—This algorithm is the one to beat. For example, I found that solving the 8-queens problem this way took over 4 billion (!) iterations.
2. DFS with Backtracking – pretty straight forward and deterministic
3. Heuristic Repair/Stochastic Search—can be just like we discussed in class (with random initial placement) or not (only putting down one queen and then doing DFS). Check what works for you!
4. Forward Checking (only in the second week of class).

For each of the four implementations, run your code 100 times keeping track of the above statistics. Calculate the average, minimum, and maximum for each of the four implementations and record their values.

Try out your code for n-queens with larger values of n (say, 100-queens). Do your algorithms succeed in finding a solution?

Grade:

20 points – providing a working solution for the both the updated backtracking, stochastic and British Museum algorithms.

20 points – documentation of logic used

60 points – how well your code works

Submission: via Git as usual.