Closed Topic

How many queens can be added to a given chess board?

This is a twist of the Eight Queen Puzzle.

The dimensions of the board, the maximum amount of queens that can be seen by a given queen (typically 0) and a in initial set of queens on the board will be given as input.

You are expected to provide as output the locations of as many queens as can be added to the given board while still satisfying the view restrictions.

There are 2 main differences between this assignment and the classic Eight Queen Puzzle:

- The size of the board is variable on width and height. This means non square boards are possible.
- The amount of queens a given queen is allowed to see is configurable, so it can (and sometimes will) be greater than 0. Queens are not considered obstacles, so in case there are 3 queens in the same row of a board, each one of them will be seen the other 2 (and maybe other queens on the board).

Technical requirements

What to implement

You are expected to implement a web server listening to HTTP POST requests at `http://0.0.0.0:8080/max_queens`

For this implementation you can use your language/technology of choice.

Requirements

1. The solution will be reviewed under Ubuntu 14.04, so make sure it can run there
2. Your solution should live on a public github repository. You will provide the Jury with the details of that repository during the hackathon
3. Your repository must implement a Makefile which offers:
    - `make install` for installing all necessary dependencies. This command will be run as root.
    - `make go` for running the server and have it listening at 0.0.0.0:8080
    - The time consumed by `make install go` cannot take longer than 2 minutes
4. The client verifying your solution will perform HTTP requests with a timeout of 2 seconds, so your server should respond quicker than that

In case you are really interested in how your code is gonna be run, asume a docker container defined like this:

``` FROM ubuntu 14.04

RUN apt-get update && apt-get install -y \ build-essential python2.7 python2.7-dev python-pip python-virtualenv

EXPOSE 8080

WORKDIR /solution

CMD ["make", "install", "go"] ```

## Detailed Input

The input format is JSON. The following snippet shows an example input:

```
{ "rows": 8, "columns": 8, "max_queens_on_sight": 0, "initial_queens": [{"x": 0, "y": 0}, {"x": 1, "y": 2}] }
```

`x` and rows refer to horizontal rows of the board. `x` can take values between `0` and `rows - 1`. `y` and columns refer to vertical columns of the board. `y` can take values between `0` and `columns - 1`. `initial_queens` may be empty

Asume the given boards are always possible, meaning no queens have been placed outside of the board or seeing too many other queens.

## Detailed output

```
{"added_queens": [ {"x": 3, "y": 6}, {"x": 2, "y": 4} ] }
```

Only new queens should be returned, and only new queens will be counted. Return an empty list of added queens is a valid response.
```

Evaluation

The same list of boards will be presented to each one of the solutions.

The winner will be the solution able to come up with the most validly added queens over all the boards. In case of a tie, the solution with the fastest response time wins.

The total number of queens is the result of adding the added queens per input board. On each one of the boards, in case the provided solution was invalid, the server timed out or any other error occurred, the amount of added queens will be considered 0.

The overall response time is the result of adding the response time of each input board. On each one of the boards, in case the server times out, the provided solution was invalid or any other error occurred, the response time will be that of a timeout. Otherwise, the response time of a particular board is the time ellapsed since the HTTP request is issued until the response body has been JSON decoded.