(364-1-1441) Foundations of Artificial Intelligence

Problem Set 2: *Then, in my wanderings all the lands... to scale the highest of the heights*

Due: 19/12/2021

Title from the poem Tiresias *by Alfred, Lord Tennyson*

You need to submit both code and written answers. Problem 1 is a programmatic one, but there are things there you need to submit in writing. Problems 2 and 3 only require written answers and not programming. You will submit one `q1.py` file containing your code, and one `answers.pdf` file containing your written, typewritten (not a scan of handwriting) answers, in English or Hebrew.

Make sure your code compiles, and the output matches what is requested. Your grader will not debug your code, and if it does not compile or output correctly, they will not be in charge of fixing your errors, even if their cause was a very minor mistake.

Also, to simplify your work process, as well as the grader's, please name your variables and methods in a meaningful way (e.g., name a function `heuristicCalculation` and not `myAwesomeFunction`).

1 Problem 1: A Board of Trouble Part II

(Problem description identical to previous problem set)

In this exercise you will build a function that starts from one board and tries to reach another using only a chain of legal moves. We will use 6×6 boards which you will receive as a 2 dimensional array, in which a value of 0 indicates the place is empty, and the value of 1 indicates there is a forcefield there, blocking your advance (marked in the output by `@`). A value of 2 indicates you have an agent in that location (marked in the output by `*`). Your agent can only move on straight lines (forward/back or left/right) a distance of 1 each turn. A piece can disappear if it makes a move to go beyond the final (6th) line, i.e., an agent making a forward move in line 6 will disappear. The *cost* is the number of moves that is needed to reach the goal, so we wish to minimize the number of steps that it will take to reach the goal board

For example:

```
Board 1 (starting position):
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
```

```
4:    @   @
5:*
6:  @
-----
Board 2:
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:
6:* @
-----
Board 3:
  1 2 3 4 5 6
1:*   *     *
2:      * @ *
3:@
4:    @   @
5:
6:* @
-----
Board 4:
  1 2 3 4 5 6
1:*   *     *
2:      * @
3:@         *
4:    @   @
5:
6:* @
-----
Board 5:
  1 2 3 4 5 6
1:*   *     *
2:      * @
3:@         *
4:    @   @
5:
6:  @
-----
Board 6 (goal position):
  1 2 3 4 5 6
```

```
1:*   *
2:      * @ *
3:@         *
4:    @   @
5:
6:  @
```

In this exercise you will only implement A* algorithm to solve this. However, additional algorithms will be coming, so keep that in mind...

You will write a function in python called `find_path`:

```
def find_path(starting_board,goal_board,search_method,detail_output):
```

The function takes 4 variables (in this order of input):

`starting_board` This is the beginning of your search. This is a 2-dimensional array populated with the values 0,1 and 2 as explained above. You can assume this is a valid board (though you can write a function to check this, which will probably help in your debugging).

`goal_board` This is the board you wish to reach from the starting board via the process. This is a 2-dimensional array populated with the values 0,1 and 2 as explained above. You can assume this is a legal board, with the forcefields in the same location (again, a function checking it's legality will probably be helpful to you)

`search_method` This will be an integer. It will have multiple possible values added in Problem Set 2, but for now you will only implement:

1. **This is just from the previous exercise, it won't be checked for correctness.** An A*-heuristic search. **You choose the heuristic**. In your submitted answers, containing the answers to the questions, you will detail your heuristic. It cannot be trivial (i.e., all 0). You need to explain in your submitted answers if your heuristic is admissible, consistent, or neither.
2. A hill climbing algorithm. It should be restarted 5 times (if it didn't find the answer).
3. Simulated annealing. **You choose the temperature**. In your submitted answers, containing the answers to the questions, you will detail your temperature schedule. It should not go longer than t=100.
4. A local beam search, with the number of beams (k) being 3.

5. A genetic algorithm. Population size is 10.

`detail_output` This is a binary variable. When it is false, your output is like the text above – you give the full chain of boards. The first one contains the *starting board*, and following that, boards with a single legal move from the board before them, until the last line contains the *goal board*. If no path was found from the starting board to the goal board, the output is `No path found`.

If the binary variable is true, for the first transformation (from the first board to the second board) you need to print out your work process, so for search method:

A*-heuristic search Print out the heuristic value of the board you are choosing. So the beginning of your print out will be:

```
Board 1 (starting position):
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:*
6:  @
-----
Board 2:
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:
6:* @
Heuristic: <your number here>
-----
```

If the binary variable is true, for the first transformation (from the first set of locations to your second set of locations) you need to print out your work process, so for search method:

Hill climbing No change in output.

Simulated annealing Show every action you consider and the probability of taking it. Multiple actions can appear on one board if the probability of taking a previous action is less than 1, and ended up not being chosen. The last action will always be the one taken in the next board. So, for example:

```
Board 1 (starting position):
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:*
6:  @
action:(2,4)->(3,4); probability: 0.5
action:(1,5)->(1,6); probability:1
-----
Board 2:
  1 2 3 4 5 6
1:*   *     *
2:      * @ *
3:@
4:    @   @
5:*
6:  @
-----
```

Local beam Show the "bag" of actions considered. So, for example:

```
Board 1 (starting position):
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:*
6:  @
-----
Board 2a:
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:
6:* @
-----
Board 2b:
  1 2 3 4 5 6
1:*   *     *
```

```
2:      * @ *
3:@
4:    @   @
5:*
6:  @
-----
Board 2c:
  1 2 3 4 5 6
1:   *   *
2:*     * @ *
3:@
4:    @   @
5:*
6:  @
-----
```

Genetic algorithm After showing the route (or lack thereof) show one of your genetic creations :

```
Starting board 1 (probability of selection from population::<number>):
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:*
6:  @
-----
Starting board 2 (probability of selection from population::<number>):
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
5:*
6:  @
-----
Result board (mutation happened::<yes/no>):
  1 2 3 4 5 6
1:*   *   *
2:      * @ *
3:@
4:    @   @
```

```
5:*
6:  @
```

2 Problem 2: CSP I

The 4-queen problem involves 4 queens on a 4x4 board, which should not threaten each other under chess rules. Formulate the problem as a CSP, and draw its constraints graph.

3 Problem 3: CSP II

What is the time complexity of the AC-3 algorithm we saw in class? Explain it fully, including your choice of the parameters you use to express the complexity.
(The course book has the answer for this, but you need to explain it in your own words)