(364-1-1441) Foundations of Artificial Intelligence

Problem Set 1: *To strive, to seek, to find, and not to yield*

Due: 29/11/2022

Title from the poem Ulysses *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. Problem 2 only requires 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: Of Kings and Bishops

Note: This same setting will be used in Problem Set 2 as well, so build your code sensibly and in a well-organized manner, as you will need it in the future.

To commemorate the upcoming British coronation, we will bring kings to meet bishops, to crown them. More specifically, 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 a king ♔agent in that location (marked in the output by `*`). The king agent moves (as the chess piece ♔) on straight lines (forward/back or left/right) or diagonally a distance of 1 each turn. A value of 3 indicates you have a bishop ♗agent in that location (marked in the output by `&`). The bishop agent moves (as the chess piece ♗) on diagonal lines only any distance it wants. 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.

In our boards we will always have the same number of bishops and kings, and the end board will have each king adjacent to a bishop.

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 0,1,2 and 3 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, 2 and 3 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. 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 (e.g., all 0). You need to explain in your submitted answers if your heuristic is admissible, consistent, or neither.

`detail_output` This is a binary variable. When it is false, your output is like the text above – you give the full chain of locations. 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 set of locations to your second set of locations) 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 (remember, this is only done once, for the first choice):

```
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>
-----
```

(the rest of the output as above)

Again, for the additional algorithms to come, additional details will be needed here, and they will appear in Problem Set 2.

2 Problem 2: Search

Describe a state space in which iterative deepening search performs much worst than DFS (for example, time complexity of $O(n^2)$ vs. $O(n)$).