DS2020– Artificial Intelligence

Lab 2

Due on 14/2/2024 3.30pm

Instructions: Upload to your moodle account one zip file containing the following. Please do not submit hardcopy of your solutions. Late submission is not allowed without prior approval of the instructor. You are expected to follow the honor code of the course while doing this homework. This lab should be completed in pairs.

1. You are allowed to use the code from the previous submission to get started with the project. You will have access to the lecture slides during the lab but **will not have access to the internet.**

2. A neatly formatted PDF document with your answers for each of the questions in the homework. You can use latex, MS word or any other software to create the PDF.

3. Include a separate folder named as 'code' containing the scripts for the homework along with the necessary data files.

4. Include a README file explaining how to execute the scripts.

5. Name the ZIP file using the following convention rollnumberrollnumberhwnumber.zip

---

Simple Treasure Hunt

In this assignment you will continue experimenting different AI search techniques that we discussed in the class on a treasure hunt problem for a single agent. This is a quick recap of the problem and the input format:

This is a variation of the three musketeers game. In this version soldiers do not move and there is a treasure (diamond) protected by at least one neighboring soldier. Your task is to implement search techniques to figure out the minimum number of moves required by any of the musketeer to reach the diamond. A musketeer can only move up, down, left or right into a space containing a soldier. The musketeer cannot move into a space that neither contain a soldier nor a diamond. There can be up to three musketeers present on the board at random locations. Your implementation should find the path from musketeer to the diamond, which is shortest among the three of them. For example, if the three musketeers can reach the diamond in 5, 6 and 7 steps, your program should output the path that takes 5 steps.

Input Format:
You are given a 2-d list of integers, which describes the game board configuration. This game board is provided in the *input.txt* file. The conventions are as follows:

- 0 - for an empty space (there can be any number of empty spaces)
- 1 - for a musketeer (there can be a maximum of three musketeers)
- 2 - for a soldier (there can be any number of soldiers)
- 3 - for a diamond (there is only one diamond)

When implementing the different search techniques use the following convention for expanding a child node

Left -> Down -> Right -> Up (anti-clockwise)

For example, assume the board configuration around one of the musketeer is as follows:

0	2	2
2	1	2
2	0	2

Your search implementation should visit soldier (2) at the left of musketeer (1), followed by soldier (2) at the right of musketeer (1), and then soldier (2) above the musketeer (1). Had there been a soldier directly below the musketeer, it should be visited at the 2nd place i.e. after visiting the soldier at the left.

Output Format:
You are required to compute three lists described as follows:

1. *exploredNodes*- This is the list of nodes explored by the search algorithm following the convention described earlier.
2. *searchQueue* – This is a list of lists; each list is the search queue at a single iteration of the search algorithm. For example, assume the algorithm reaches the goal state in 5 steps, then the first list in *searchQueue* will be the queue at the first iteration, second list will the queue at the second iteration and so on.
3. *shortestPath* – This is the actual path from the start state (one of the musketeers) to the goal state (diamond)

For example, consider the following board configuration (4x4 for the sake of simplicity):

	y			
x	2	2	0	0
	0	1	2	2
	0	2	3	2
	2	0	2	0

The three lists given by the Breadth First Search on above input should be as follows:
exploredNodes = [[1, 1], [2, 1], [1, 2], [0, 1], [2, 2]]
searchQueue = [

[[2, 1], [1, 2], [0, 1]],
[[1, 2], [0, 1], [2, 2]],
[[0, 1], [2, 2], [1, 3]],
[[2, 2], [1, 3], [0, 0]],
[[1, 3], [0, 0]]
]
shortestPath = [[1, 1], [2, 1], [2, 2]]

Implementation Details:
For this lab you will be implementing the following algorithms

1. A* Search with Euclidean distance as the heuristic
2. Iterative Deepening A* Search

The input arguments of your code will be the file containing the maze (input.txt) and 0/1. 0 – corresponds to A search and 1 for iterative deepening A* search.*

Submit the all the code as a single zip file. The submission will be tested on game boards of size between 5 and 15. It is imperative that you follow these instructions, as your code will be checked automatically. Include a pdf file that describes your observations of the different search techniques.