Artificial Intelligence
Machine Problem 1 – Robot Navigation

IMPORTANT NOTICE: You do not have permission to share the description of this assignment or discuss it with anyone outside the class. Also, you cannot share your code (or solution in general) with anyone, nor can you ask anyone for the code or solution. Failure to adhere to this policy will result in a zero grade for the assignment and may result in a dismissal from Lewis University.

Introduction

Assume you have designed a robot that needs to get out a maze by getting to the closest exit with least moves. The robot can generally move in the four cardinal directions: up, down, left, or right. However, the design of the robot does not allow movements in all four directions at the same time. Therefore, before it moves, it needs to select which of the movements disabled. You can assume you have the map of the maze with the exit marked. Furthermore, the maze itself allow for wrap-around movements, so if you are in the rightmost location, you can choose to keep going right and end up in the leftmost location, assuming it isn't blocked by a wall. The same applies to the other movements. Your job is to implement an algorithm that can decide which movement to disable so the path to the closest exit can be achieved with the least number of moves. The robot needs to start moving as soon as possible after it is placed in the maze, so the optimal decision and shortest path should be figured out in the least amount of time.

Requirements

To solve the problem, you will need to implement the A* algorithm to find an exit out of a maze, assuming only certain moves are possible. You can use this to then figure out which subset of moves allows for a path with the least number of moves. Your goal is to show the optimal choice, the instructions for solving the maze, and the maze configuration showing the shortest path to the closest exit.

You can assume the maze is a rectangular grid containing passageways, walls, and one or more exit points that the robot must reach. The robot can move in three of the four possible directions: left, right, down, or up. The maze is encoded in a space-separated file, containing numbers indicating what is at each location of the maze. Zero is used for spaces, one for walls, two for exits, and five for the starting point of the robot (see sample outputs at the end).

For this assignment, you are given base Python 3 code that executes a uniform cost search for a similar maze problem. So, download this code first, and then do the following:

1. Study the base code and make sure you understand how it works. Run it to check the path it finds.

2. Modify the code to allow for wrap-around moves.

3. Add code so you can run the search algorithm with one of the movements disabled. Have the code go through all four possibilities and compare the answers to see which provides the shortest path to the closest location.

4. Think of a heuristic function that can be used for this problem with an A* search. Then, modify the code so it uses the A* algorithm with that heuristic. Remember, heuristic needs to satisfy the admissibility and consistency properties. You can achieve this by relaxing the problem assumptions until it becomes easy to find the exact solution with a simple calculation. The exact solution to the relaxed problem can then be used as a heuristic for the original problem. Your heuristic should take into account the wrap-around moves and the fact that there are multiple exits (at known locations), but you don't need to take into account the disabled move (although that could make it better).

5. Modify the print statements from the base code to include your info in the printed heading:

```
Artificial Intelligence
MP1: Robot navigation
SEMESTER: Spring 2024
NAME: [your name here]
```

6. Your program needs to work correctly for all inputs. The maze itself is loaded from the `maze2024.txt` file. This file could be different for testing purposes. The shape of the maze could also vary, although the start and end coordinates will be fixed.

Additional Requirements

1. The name of your source code file should be `mp1.py`. All your code should be within a single file.
2. You can only import numpy, heapq, and queue packages.
3. Your code should follow good coding practices, including good use of whitespace and use of both inline and block comments.
4. You need to use meaningful identifier names that conform to standard naming conventions.
5. At the top of each file, you need to put in a block comment with the following information: your name, date, course name, semester, and assignment name.
6. The output format should **exactly** match the sample output shown on the last page. Note that for a different input, the output may be different. I will be testing on a different input than shown in the sample.

What to Turn In

You will turn in the single mp1.py file using BlackBoard.

Grading Rubric

Category	Unsatisfactory (0-1 points)	Satisfactory (2-3 point)	Distinguished (4-5 points)
Program Correctness	• Program does not execute due to errors • Incorrect results for most or all input	• Program works and completes most tasks appropriately • Program fails to work for special cases	• Program runs and completes all required tasks • Handles any required special cases • Executes without errors
Programming Style	• No name, date, or assignment title included • Poor use of white space • Disorganized and messy • No or few comments in the source code • Poor use of variables (improper scope/visibility, ambiguous naming).	• Includes name, date, and assignment title. • White space makes program fairly easy to read. • Well organized code. • Some comments missing in the source code or too many comments • Good use of variables (few issues with scope/visibility or unambiguous naming).	• Includes name, date, and assignment title. • Excellent use of white space. • Perfectly organized code. • Source code is commented throughout when needed • Excellent use of variables (no issues with scope/visibility or unambiguous naming).
Following Specifications	• Incorrect filenames • Incorrect specified identifier names • Source code organization different from requirements • Additional requirements not satisfied	• Correct filenames and class names • Few issues with other specified identifier names • Source code organization close to requirements • Some additional requirements not satisfied	• Correct filenames and specified identifier names • Source code organization satisfies all requirements • All additional requirements satisfied

Sample Program Output

```
 Artificial Intelligence
MP1: Robot navigation
SEMESTER: Spring 2024
NAME: [your name]

INITIAL MAZE
[[1 1 1 0 0 0 0 0 0 1]
 [1 0 5 1 1 1 1 1 0 1]
 [1 0 1 1 1 1 1 0 2 1]
 [0 0 0 0 0 1 0 0 0 0]
 [1 1 0 1 0 1 0 1 0 0]
 [0 0 0 1 0 1 0 0 0 1]
 [1 1 1 1 0 0 0 1 0 1]
 [1 0 0 0 0 1 0 1 0 0]
 [1 1 1 1 0 0 0 1 0 1]
 [0 2 0 0 0 1 0 1 0 0]
 [1 1 1 0 1 1 0 1 1 0]
 [1 0 0 0 1 0 0 1 0 0]
 [0 0 1 0 1 1 0 0 0 1]]
SOLUTION AFTER DISABLED MOVE:  left
[[1 1 1 0 0 0 0 0 0 1]
 [1 0 5 1 1 1 1 1 0 1]
 [1 0 1 1 1 1 1 0 2 1]
 [0 0 0 0 0 1 0 0 0 0]
 [1 1 0 1 0 1 0 1 0 0]
 [0 0 0 1 0 1 0 0 0 1]
 [1 1 1 1 0 0 0 1 0 1]
 [1 0 0 0 0 1 0 1 0 0]
 [1 1 1 1 0 0 0 1 0 1]
 [0 2 0 0 0 1 0 1 0 0]
 [1 1 1 0 1 1 0 1 1 0]
 [1 0 0 0 1 0 0 1 0 0]
 [0 0 1 0 1 1 0 0 0 1]]
No solution
SOLUTION AFTER DISABLED MOVE:  right
START
Move 1 ACTION: left
Move 2 ACTION: down
Move 3 ACTION: down
Move 4 ACTION: left
Move 5 ACTION: left
Move 6 ACTION: left
Move 7 ACTION: up
[[1 1 1 0 0 0 0 0 0 1]
 [1 4 5 1 1 1 1 1 0 1]
 [1 4 1 1 1 1 1 0 2 1]
 [4 4 0 0 0 1 0 3 4 4]
 [1 1 0 1 0 1 0 1 3 3]
 [0 0 0 1 0 1 0 0 0 1]
 [1 1 1 1 0 0 0 1 0 1]
 [1 0 0 0 0 1 0 1 0 0]
 [1 1 1 1 0 0 0 1 0 1]
 [0 2 0 0 0 1 0 1 0 0]
 [1 1 1 0 1 1 0 1 1 0]
 [1 0 0 0 1 0 0 1 0 0]
 [0 0 1 0 1 1 0 0 0 1]]

Number of states visited = 8

Length of shortest path =  7
SOLUTION AFTER DISABLED MOVE:  down
[[1 1 1 0 0 0 0 0 0 1]
 [1 3 5 1 1 1 1 1 0 1]
 [1 0 1 1 1 1 1 0 2 1]
 [0 0 0 0 0 1 0 0 0 0]
 [1 1 0 1 0 1 0 1 0 0]
 [0 0 0 1 0 1 0 0 0 1]
 [1 1 1 1 0 0 0 1 0 1]
 [1 0 0 0 0 1 0 1 0 0]
```

```
 [1 1 1 1 0 0 0 1 0 1]
 [0 2 0 0 0 1 0 1 0 0]
 [1 1 1 0 1 1 0 1 1 0]
 [1 0 0 0 1 0 0 1 0 0]
 [0 0 1 0 1 1 0 0 0 1]]
No solution
SOLUTION AFTER DISABLED MOVE:  up
START
Move 1 ACTION: left
Move 2 ACTION: down
Move 3 ACTION: down
Move 4 ACTION: left
Move 5 ACTION: left
Move 6 ACTION: left
Move 7 ACTION: down
Move 8 ACTION: down
Move 9 ACTION: down
Move 10 ACTION: down
Move 11 ACTION: down
Move 12 ACTION: down
Move 13 ACTION: right
Move 14 ACTION: right
Move 15 ACTION: right
[[1 1 1 0 0 0 0 0 0 1]
 [1 4 5 1 1 1 1 1 0 1]
 [1 4 1 1 1 1 1 0 2 1]
 [4 4 3 3 3 1 3 3 4 4]
 [1 1 3 1 3 1 3 1 4 3]
 [3 3 3 1 3 1 3 3 4 1]
 [1 1 1 1 3 3 3 1 4 1]
 [1 3 3 3 3 1 0 1 4 3]
 [1 1 1 1 3 3 0 1 4 1]
 [4 2 3 3 3 1 0 1 4 4]
 [1 1 1 3 1 1 0 1 1 3]
 [1 0 0 0 1 0 0 1 0 0]
 [0 0 1 0 1 1 0 0 0 1]]

Number of states visited = 41

Length of shortest path =  15
BEST MOVE: disable right
SHORTEST PATH LENGTH FOR BEST MOVE: 7
```