AI – Project 2

Rules:

- Plagiarism of any kind will result in failure for the course. No exceptions. If you are not sure whether or not you are plagiarizing from the web, classmates, or any other person/source, it is your responsibility to clarify it with the professor before submitting the code.
- You can be in a team with at most another student from the class. You are allowed to discuss the assignment only with your team member or the professor.

  If you have registered for the undergraduate version of the course, then your partner should also have registered for the undergraduate version of the course.

  If you have registered for the graduate version of the course, then your partner should also have registered for the graduate version of the course.

  It is ok if one member is registered for CSC and the other one for EE.
- You are allowed to use the web to learn more about search strategies. You are not allowed to use or copy source code from the web.
- You are allowed to write the code in C, C++, Java, Matlab, Python, or any other programming language. You should submit (via Blackboard) all your source files and a README.txt file which should describe how to compile and run the code. The README.txt file should also contain the names and email address of each team member.

  You will receive 0 points if the code does not compile, i.e., generates errors during the compilation. If your team has two members, only one team member should submit.
- Submit a printout of your code in class. Submit the source code, README, and report as one zipped directory via Blackboard.

1 Grid Search on a Costmap

Given a grid, where certain cells are marked as obstacles, the objective is to find a path from an initial to a goal cell that avoids collisions with the obstacles. More specifically, each grid cell will have a numeric value in the range $[0, 1]$. A value of 0 indicates that the cell is an obstacle, which must always be avoided. A value of 1 indicates that the cell is free. The cost of passing through a cell (i, j) should be computed as

$$cost(i, j) = 1/val(i, j).$$

In this way, obstacle cells have a cost of ∞, while free cells have a cost of 1. Note that the cost increases as the value of the cell gets closer to 0.

The program will take the following arguments from the command line:

```
GridSearch method grid.txt rstart cstart rend cend path.txt
```

where `method` is the name of the search method, `grid.txt` is the name of the file with the grid, `rstart` is the row index of the start position, `cstart` is the column index of the start position, `rend` is the row index of the end position, `cend` is the column index of the end position.

The method name is as follows: `DFS` for depth-first search, `BFS` for breadth-first search, `AStarZero` for A*-search with the zero heuristic, `AStarEuclidean` for A*-search with the Euclidean heuristic, `AStarManhattan` for A*-search with the Manhattan heuristic.

The program should read the file `grid.txt` and search for a solution from `(rstart,cstart)` to `(rend,cend)`. It should then write the solution to the file `path.txt`. The moves are LEFT, RIGHT, UP, DOWN (no diagonal moves allowed).

Note that `DFS` and `BFS` treat the cells with value (> 0) as free cells since these methods just focus on computing a path. Essentially, these methods ignore the cost.

A* should take the cell-traversal costs into account so that it can find the shortest path to the goal.

The format of the input and output files are as follows:

Indices: all indices are assumed to start from 0.

Format of grid.txt

- first line contains the number of rows (nrRows) and the number of columns (nrCols)
- after that there should be a list of $nrRows \times nrCols$ numbers, each one specifying the cell value. Note that the i-th number in the list specifies the value for the cell with row $i/nrCols$ and column $i \mod nrCols$.

For example,

```
4 2
0 0 0.1 0.2 0.1 0.7 0 0.9
```

indicates a 4×2 grid. The cell values are specified in this order: $(0,0), (0,1), (1,0), (1,1), (2,0), (2,1), (3,0), (3,1)$.

Format of path.txt

- each line contains the row index and the column index of the i-th cell in the path

For example,

```
1 0
2 0
2 1
2 2
2 3
3 3
```

indicates the path $(1,0), (2,0), (2,1), (2,2), (2,3), (3,3)$

2 Creating Input Files for Testing

You can download any grayscale image from the web and use it for testing purposes. Support code is provided (in Matlab/Octave) to convert the grayscale images in the grid format. Support code is also provided to show the grid and your solution path.

Feel free to also create your own path-planning grayscale images. Open paint/gimp. Add some obstacles. Save your image in grayscale.

Follow these steps:

- Download a grayscale image from the web, say gray.jpg
- Convert the grayscale image to grid.txt, e.g.,

```
ImgToGrid('gray.jpg', 'grid.txt')
```

- Test that the conversion worked well by displaying the grid, e.g.,

```
ShowGrid('grid.txt')
```

- Run your program to get a solution. Suppose solution is stored in path.txt. Then visualize the grid and your solution, e.g.,

```
ShowGridAndPath('grid.txt', 'path.txt')
```

Note that even though the support code is in Matlab, you can write your program in any language that you like. The support code just provides image-to-grid conversion and visualization for the grid and your solution.

3 Additional Requirements for Graduate-Course Enrollment

If you are taking AI as a graduate course, you need to implement **Fringe** search in addition to DFS, BFS, and A*. See http://en.wikipedia.org/wiki/Fringe_search for more information on Fringe search.

4 What to Submit

All these in one zipped folder:

- README.txt: Your name, name of your partner, course number, instructions on how to compile/run your code, anything else you would like to note
- All source files