CSE 468/568 Assignment 8
A* Path Planning

The objective of this assignment is to plan a path for a robot from a given starting point to a destination. Create a new package called `lab8`, and place the world files (`playground.pgm` and `playground.world`) from the associated file `(lab8.zip)` in the appropriate sub-folder.

The objective of the assignment is to use A* planning algorithm to find a route from a default start point, e.g. (-8.0, -2.0) to a target goal, e.g. (4.5, 9.0). Please go through the tutorial on ROS Parameters. The goal should be defined as two parameters `goalx` and `goaly` both of which should be of type `double`. This allows us to set a new before launching.

There are a couple of challenges in implementing A* planning as discussed in class. The first challenge is to derive a graph representation of the workspace. This depends on the map representation that the estimation block provides us. Typical examples of such representations are occupancy grids - a grid representation with 1s and 0s with 1 indicating an obstacle in that cell and 0 representing an empty cell. For this assignment, we have provided you such an occupancy grid. It is the file `map.txt`. It grids the world as 1m x 1m cells. You should import this into your program as the map. You can simply paste the array into your code and read it appropriately as a 2D matrix with dimensions of (20, 18).

The second challenge is the heuristic for the estimated cost between the current node and the goal. Given you know the current location and the goal, you can use Euclidean distance between the current location and the goal as the heuristic cost. ϵ is the coefficient of the heuristic function, used to scale the heuristic cost. You can start with $\epsilon = 1$, and tune only if required, based on the paths returned by your A* algorithm.

Your A* path planner algorithm should output a global path from the start node to the goal node. The global path essentially contains a list of nodes/checkpoints, such that when the robot moves to each one of them in succession, it will eventually reach the final goal location. First, print the path to the screen. Then command your robot to execute the plan. This should be broken down to 1-step at a time, until the path is finished. You need to be able to convert a world coordinate to a cell/node, and vice versa. At each step, you need to turn and move towards the center of the next cell. Once the robot reaches a certain **proximity**, it can move towards the next cell.

You may refer to this video for a walk-through of the A* algorithm.

Test to see if you can launch everything by running the command:

```
$ roslaunch lab8 lab8.launch
```

Submission Instructions

You will submit `lab8.zip`, a compressed archive file containing the lab8 package (folder). The folder should contain the world file, the pgm file, a launch file `lab8.launch` and the controller in appropriate sub-folders. The folder should compile if we drop it into our catkin workspace and call `catkin_make`. You should also print the path that your A* algorithm generates in the terminal. The format of this output should be a modified version of the `map.txt` array. Empty cells are shown with spaces, obstacles with 1s, and a * is placed over cells that are part of your path. You should add a space after each character in a row, to adjust for character width/height ratio. Here is a sample output:

```
                        1   1
                        1   1

1
  1
    1       1 1 1 1 1 1
    1       1 1 1 1 1 1
      1                     * 1 1
        1                   * 1 1 1
        1 1                 * 1 1 1
          1                 * 1 1 1
          1 1             *     1
            1 1 1         *
              1           *
      * * * * * * * * * * *
      *         1
      *         1 1       1 1 1 1
      *         1 1 1     1 1 1 1
      *       1 1 1       1 1 1 1
                1 1       1 1 1 2 1
```

For additional points, you can use `matplotlib` to visualize the path better.

Please take care to follow the instructions carefully so we can script our tests. Problems in running will result in loss of points.

Please use UB learns for submission

The assignment is due Monday, May 2 before midnight.

Tips

Diagonal Movement

You can design your algorithm based on the 4-neighborhood (only consider nodes to its top, bottom, left, and right). If you decide to go with 8-neighborhood, you need to validate possibility of diagonal moves before selecting the corner cells. i.e you may not move from the center cell to the top right corner cell, if the up cell and right cell are both obstacles. This would allow your path to "go through" an obstacle. An example of such unacceptable path is shown below:

```
                        1   1
                        1   1

1
  1
    1       1 1 1 1 1 1
    1       1 1 1 1 1 1
      1 * * *                 1 1
      * 1                     1 1 1
    *   1 1                   1 1 1
    *     1                   1 1 1
    *     1 1                   1
            1 1 1
              1

                1
                1 1       1 1 1 1
                1 1 1     1 1 1 1
              1 1 1       1 1 1 1
                1 1       1 1 1 2 1
```

Map

You could copy paste the map from `map.txt` directly to your script. You can then rearrange it as you see fit for your implementation.