Design

Student ID: 1698774 Name: Qianyi Lu

Please run the java program on eclipse as the arguments on eclipse and on linux system might has position difference.

My chosen is using A* search algorithm. A* search is represent as F(n)=g(n)+h(n)

The g(n) represent the cost of present path, which should be steps the robot has already taken.

The h(n) represent the cheapest cost to reach the target end point. For such a maze, the distance can be simply treated as the sum of the difference between the x,y positions of present point and target end point. As this robot can only move in four directions but not eight directions, this approximation has been proved as Manhattan distance. So the h(n)=Xend+Yend-Xpresent-Ypresent.

The robot's every action can be treated as a state. Creating a class to save such a state is a good choice, which should has four int variables to store the present x/y position, the g(n) and h(n). The class should also contains a string to store the actions robot has already taken.

When the maze's data has been read in, the robot should has an initialization state which is at the start point. Every time this robot tries to move, it first checks whether the state list is empty which means all the path has been check and none of them can reach the target end point. If it is not empty, check whether it is already at the end point, if yes, the program stops. After these two checks, this loop starts to check whether there is a blank space on its four directions. Create new states for those positions which it can go to. Finally, delete the original state as the robot has move away. After such process, there might be several states. Sort them as A* search, which means put the state with smallest g(n)+h(n) at the front. Then on every round the program makes a new decision, it always deal with the first data in the state list. To prevent the robot moves in circle or checking different ways to reach the same point which is useless, another array should be set to store those points some state has been reach. If a point is first time being reached by a state, that path must be one of the shortest path by A* search. So if this point is reached second times, the second path can be ignored.

Some other problems might happen due to the maze data's input. As the input is read by line, a better way to deal with the data is treating "row" as "column". Then the first position of a point is also the first level of the array storing the maze.

The test cases with outputs are attached below.

(Testgird_samll.txt and Testgird_large.txt have been changing to meet question's request)

```
Input:   ./robotplanner Testgird_samll.txt 0 0 4 0
Output:     R R U R R D
Input:   ./robotplanner Testgird_samll.txt 2 2 0 4
Output:     L U L U
Input:   ./robotplanner Testgird_samll.txt 2 2 2 2
Output:     Already at the target.
Input:   ./robotplanner testgrid_large.txt 0 3 19 0
Output:     R D R R R U R R R R R R R R R U U U U R R R R R D D D D D R D
D
Input:   ./robotplanner testgrid_large.txt 18 13 5 14
Output:     No path to reach.
```