Implementation

The python classes along with test mazes for this project includes the following files:

* robot.py: This script establishes the robot class. This is the only script I have modified.
* maze.py: The provided script contains functions for constructing the maze and for checking for walls upon robot movement or sensing.
* tester.py: The provided script runs to test the robot’s ability to navigate mazes.
* showmaze.py: The provided script is used to create a visual demonstration of what a maze looks like.
* test\_maze\_01.txt - test\_maze\_03.txt: The three provided sample files specify mazes upon which to test my robot.
* test\_maze\_04.txt: The 12x12 maze which is designed by myself for testing the robustness of my robot.

In order to understand the architecture of the robot environments, I plot a diagram as shown in Figure x. The file {\verbatim tester.py} is the tester of the whole environment. Tester can call the {\verbatim Maze} class to obtain the given maze. Then, the tester can initialize a robot, and send the only parameter, the dimension of the maze, to the robot. While a robot has been initialized, its location and heading is always $(0,0)$ and `up’, respectively.

The architecture of the robot and its environment:

(A Figure)

For the purpose of clarity, I summarize some of the characteristics of the robot as below.

* The robot is always initialized by the tester. While the robot is initialized, its location is $(0,0)$ and heading is ``up’’.
* The robot does not know any information about the given maze except the dimension. Of course, the maze is always a square.
* The robot has to maintain the map of the maze when it visiting the maze.
* The robot has three sensors mounted on the front of the robot, its right side, and its left side. These sensors can detect the distances from the robot to walls.
* I implemented only one method (function) for the \texttt{Robot} class, \texttt{next\_move}. (Except the constructor, of course.)
* The \texttt{next\_move} function must then return two values indicating the robot’s rotation and movement on that time-step. Rotation is expected to be an integer taking one of three values: $-90$, $90$, or $0$, indicating a counterclockwise, clockwise, or no rotation, respectively. Movement follows rotation, and is expected to be an integer in the range $[-3, 3]$ inclusive.
* The robot will attempt to move that many squares forward (positive) or backwards (negative), stopping movement if it encounters a wall.

The search algorithms implemented for the robot are DFS and A-star. In some sense, my DFS could be viewed as an A-star using heuristic matrix with $0$ in each entry. For an A-star search, the heuristic matrix plays a very important role in getting a good performance. I particularly mention my design of the heuristic matrix here.

For a $n\cross n$ maze, my heuristic matrix ($H$) is defined as below.

In particular, the heuristic matrix for $12\times 12$ maze is defined as:

Now, we apply the heuristic matrix to the three given mazes, a preliminary result could be obtained.