University of Rutgers

Assignment 1: Fast Trajectory Planning

Author:
Xuan Wang
Xunjie Zhu

Supervisor:
Abdeslam Boularias

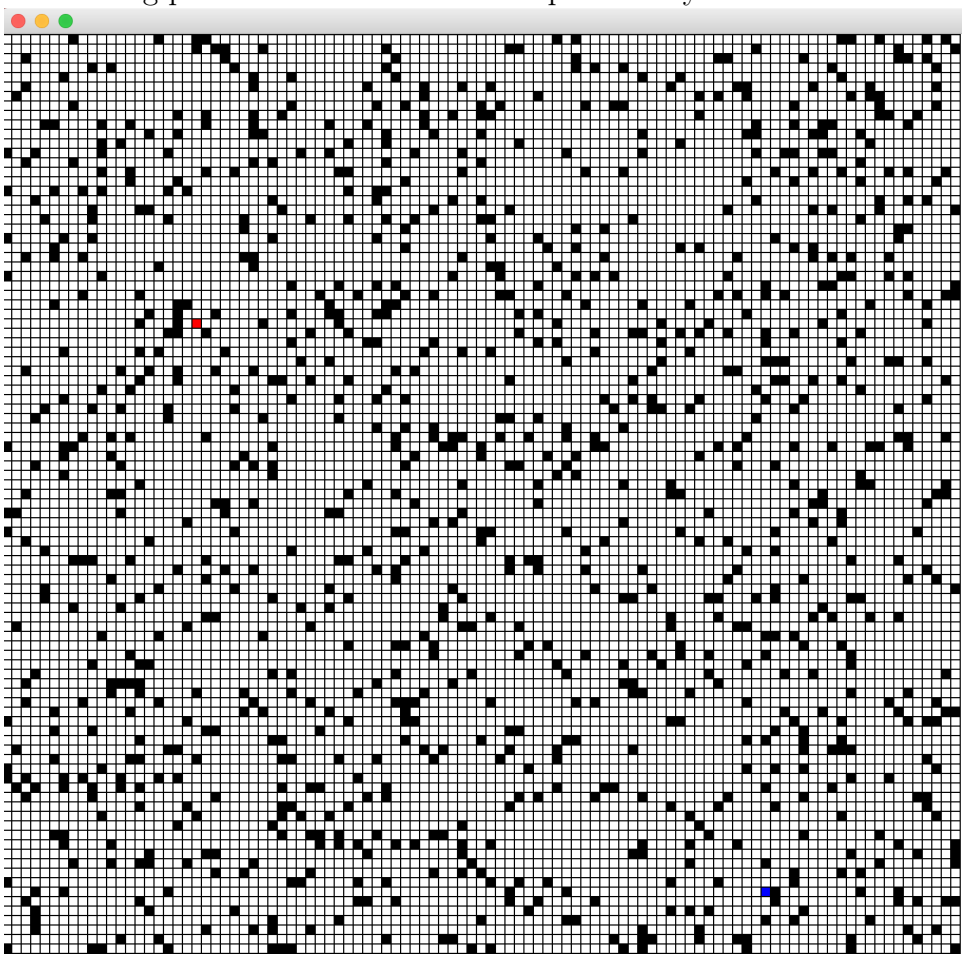
February 26, 2017

Part 0

For the maze, define a maze class, use *numpy* external library to generate a 2-d 101*101 int array representing the maze. Then use *networkx* library to convert the maze into an undirected graph by adding edges for each pair of adjacency nodes, use *dfs_preorder_nodes()* method to find a traverse list by DFS, for each visited node make it blocked of 30% chance

```
self.array = np.zeros((m,n))
traverselist = list(nx.dfs_preorder_nodes(self.graph,(0,0)))
p_list = [1,1,1,0,0,0,0,0,0,0]
traverselist = self.DFS()
for i in traverselist:
   self.array[i[0],i[1]] = random.choice(p_list)
```

The following picture is a visualization implement by scala



Part 1

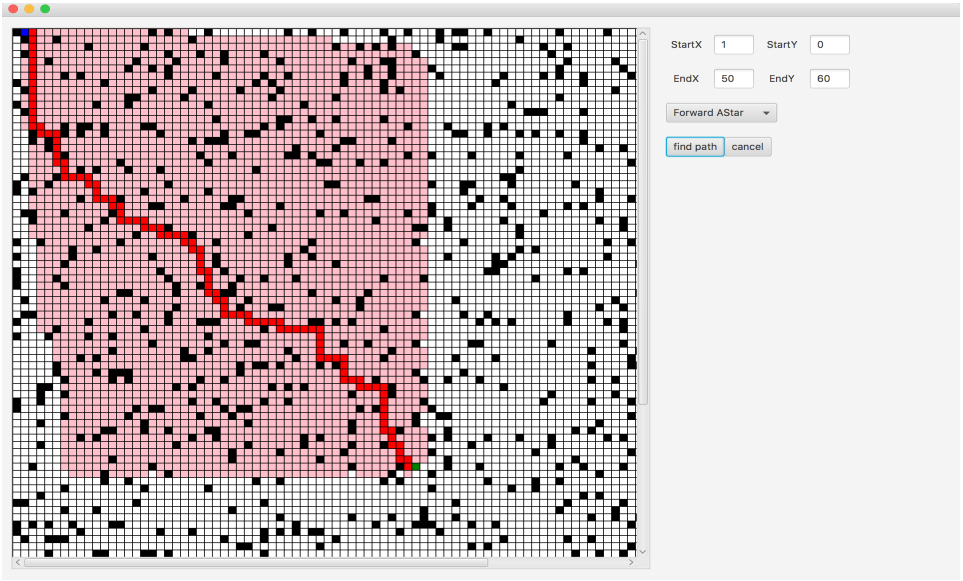
(a)While the initial agent explored three cells next to it which is unblocked, the three cells have same *g-values* 1, different *h-values*. Base on Manhhatan distance, the right one cell has the samllest *g-value* 2, so it has the smallest *f-value* 3, so the right one will be expanded first.

(b)Since the algorithm finds cell with minimum f value from open list and add it to close list, if open list is running to empty without found a end node, that means no more unvisited cells can be expanded and there is no path from start node to target node. It is always possible to find a path or find it impossible before all unblocked cells are added into close list, so the result can be found in finite time. The worst case is that the agent finds block and change path at each unblocked cell, and for each cell the agent goes through all possible unblocked cells and find a dead end. So the bound of agent moves is the number of unblocked cells squared.

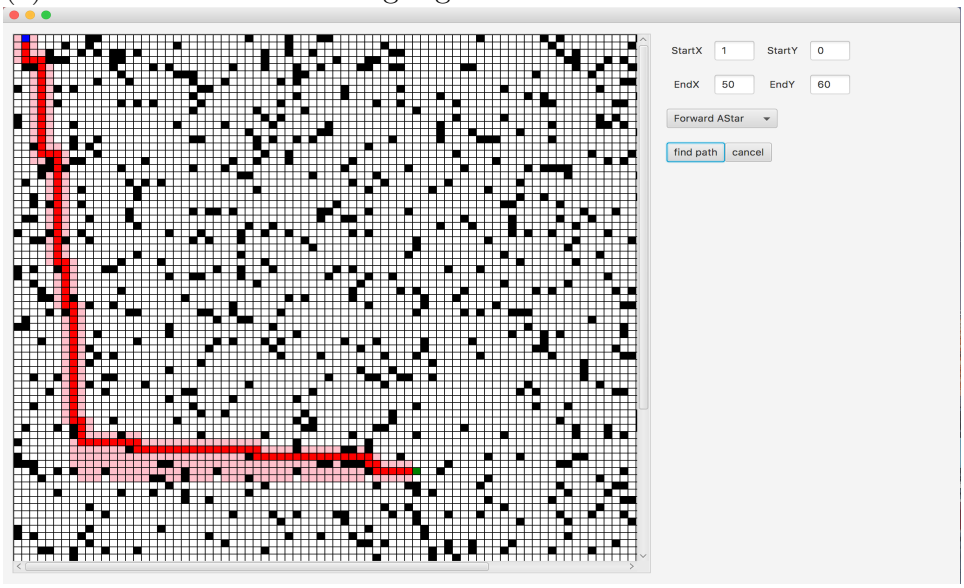
Part 2

Both version of Forward A* use the same generated maze to test, start point is in the upper left corner of maze, target point is in the lower right corner of maze and the following are the results:

(1) break ties in favor of small g value



(2) break ties in favor of larger g value

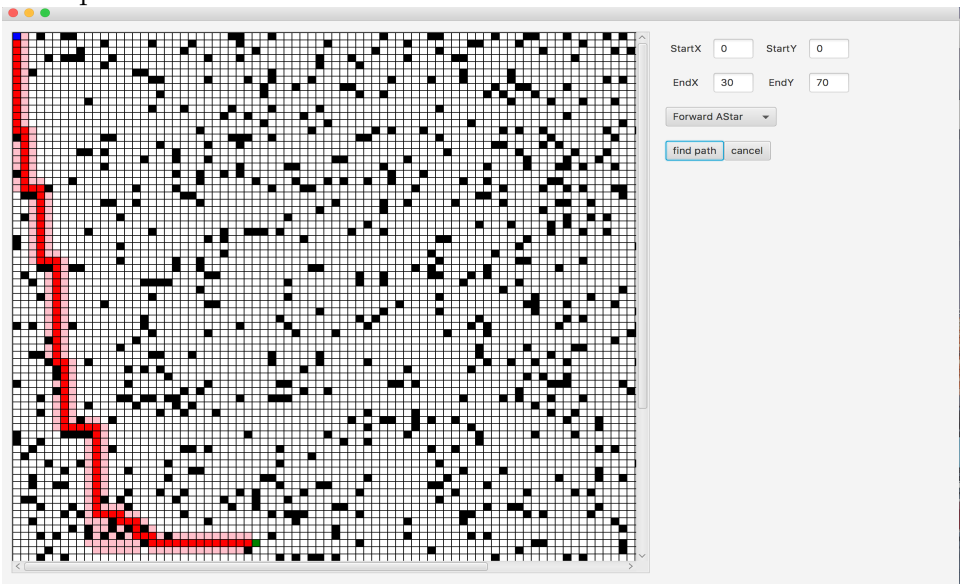


cell with pink color in graphs means this cell is expanded and added to close list, apparently choosing small g value to break ties cause more cells be expanded than choosing larger g value. It is because the

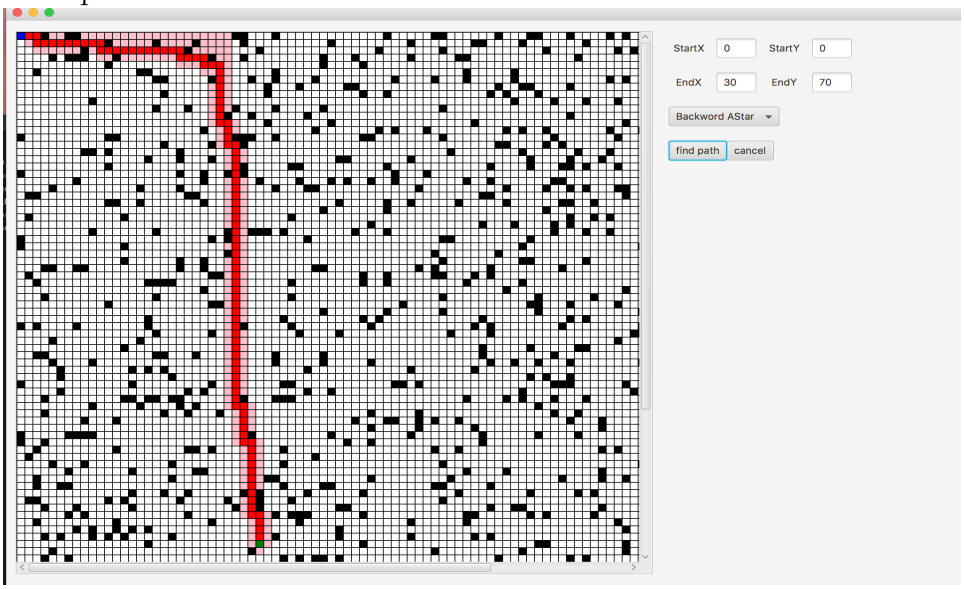
computer will always choose the farthest point away from the start point if a cell with larger g value has priority, and the search point will leave the start point as quickly as possible. If smaller g value is chosen, it is likely that cells having same f value around the start point will be searched level by level with the increasing of g value. *A1 B1 C1 D1 E1 E2 E3 E4* cells will be expanded by both in Figure 9.

Part 3

3.1. Repeated Forward A*



3.2. Repeated Backward A*



the results show that Repeated Forward A* and Repeated Backward A* have no significant difference, they have almost the same running time and same expanded number of cells and different paths. The cause of difference of paths is that they have the same break ties rule.

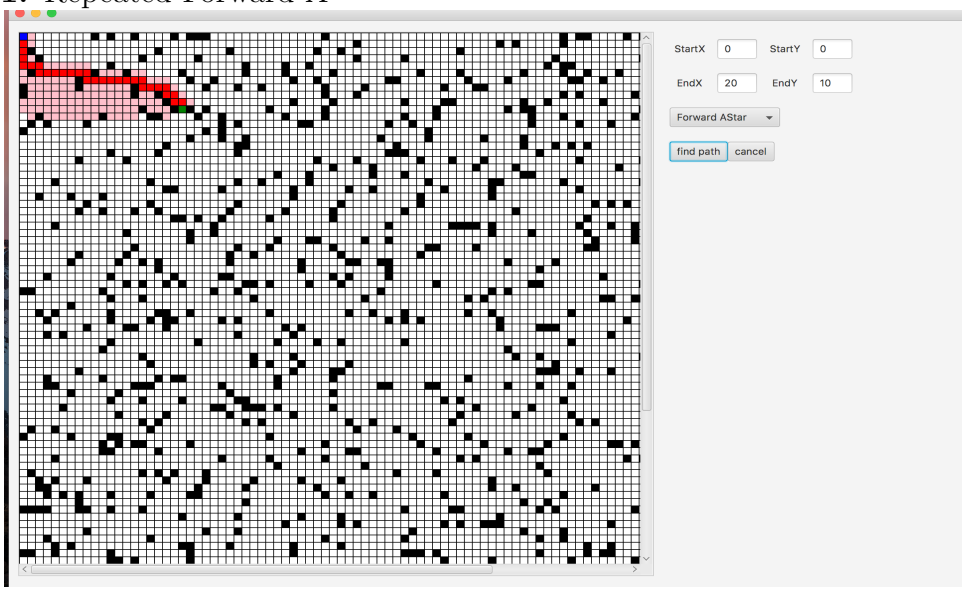
Part 4

The definition of Manhattan distance: The distance between two points measured along axes at right angles. If the agent only moves in four compass directions, it will only move along the axes we built, it assures that the Manhattan distances are consistent.

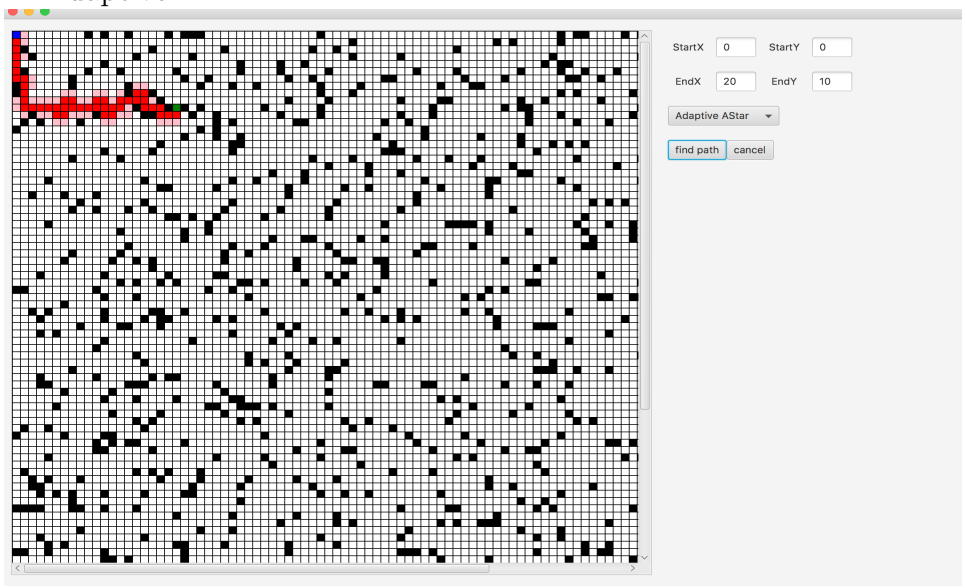
(OFFICIAL ANSWER)Adaptive A* solves a series of similar but not necessar- ily identical search problems. However, it is important that the h-values remain consistent with respect to the goal state from search problem to search problem. The following changes can occur from search problem to search problem: ? The start state changes. In this case, Adaptive A* does not need to do anything since the h-values remain consistent with respect to the goal state. ? The goal state changes. In this case, Adaptive A* needs to correct the h-values. Assume that the goal state changes from sgoal to s?goal. Adaptive A* then updates (= overwrites) the h-values of all states s by assigning h(s) := max(H (s, s?goal), h(s) ? h(s?goal)). (2) The updated h-values are consistent with respect to the new goal state [9]. However, they are poten- tially less informed than the immediately preceeding h-values. Taking the maximum of h(s) ? h(s?goal) and theuser-suppliedH-valueH(s,s?goal)withrespectto the new goal state ensures that the h-values used by Adaptive A* dominate the user-supplied H-values with respect to the new goal state at least weakly. ? At least one action cost changes. If no action cost decreases, Adaptive A* does not need to do anything since the h-values remain consistent with respect to the goal state. This is easy to see. Let c denote the orig- inal action costs and c? the new action costs after the changes. Then, h(s) ? c(s, a) + h(succ(s, a)) for s ?

Part 5

5.1. Repeated Forward A*



5.2. Adaptive A*



the results show that Adaptive A* expanded fewer cells than Repeated Forward A* and has less running time if the path is short enough. It is because

Part 6

The tree pointer can be represented as a two bit number .

```
val directions = List((0, 1), (0, -1), (1, 0), (-1, 0))
```

It is calculated by father node coordinate minus child node coordinate.

Another way is to use sparse matrix representing the maze since the quantity of blocked cells is small enough.