

2018/2019 COMP1037 Coursework 1 – Search Techniques

(The answer is attached below)

This part is based on the maze generator demo (MazeGeneration-master). The maze generator is a project written by some student using Matlab. He has adopted a tree search approach to randomly generate a maze with user-defined size and difficulty. **(15 marks)**

- (a) Read the MazeGeneration-master code, identify which line(s) of code is used to implement the tree search approach, explain the logic and the data structure used by the student to implement the tree search. (3 marks)
- (b) Identify the logic problem of this maze generator if there is any. (1 marks)
- (c) Write a maze solver using A* algorithm. (5 marks)
 - i) The solver needs to be called by command '**AStarMazeSolver(maze)**' within the Matlab command window, with the assumption that the 'maze' has already been generated by the maze generator.
 - ii) In the report, show what changes you have made and explain why you make these changes. You can use a screenshot to demonstrate your code verification.
 - iii) The maze solver should be able to solve any maze generated by the maze generator
 - iv) Your code need display all the routes that A* has processed with RED color.
 - v) Your maze solver should be about to display the final solution with BLACK color.

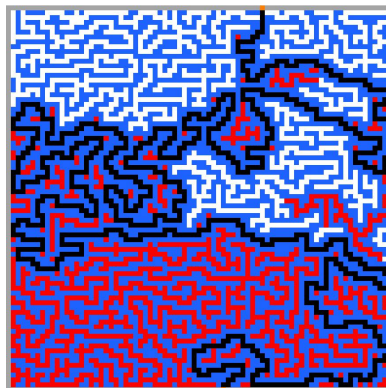


Figure 1. Sample output

- (d) Based on the previous AStarMazeSolver, implement a maze solver using the DFS algorithm. Similar to question (c), the solver need by called and ran by command '**DFSMazeSolver(maze)**'. (3 marks)
- (e) Based on the previous AStarMazeSolver, implement a maze solver using the Greedy search algorithm. Similar to question (c), the solver need by called by command '**GreedyMazeSolver(maze)**'. (1 marks)

- (f) You are required to use the Matlab basics from the first lab session to show the evaluation results of the three searching methods you've implemented in (c), (d) and (e) (hint: bar/plot) with respect to the '**total path cost**', '**number of nodes discovered**' and '**number of nodes expanded**'. Explain how you can extract the related information from data stored in variable '**QUEUE**'. (2 marks).

(a)

In the move function,in fact,most of the move function.

Data Structure:graph,queue

Logic:First use valid-move to check if the surrounding can move,then we use the random function to choose a path and thus nodes are created,every time it changes direction a node is created,then it will not stop until it reaches a wall or explored space(DFS),meanwhile it could change direction and new node could be created.

(b)

The maze seemed to be randomly generated however it is not because it is using an algorithm,if the algorithm is truly random then there is some circumstances that the end will never be reached.In a infinity loop, and the maze should have ended circles,ended loops like crop circles which are never reachable by us yet every time you opened the maze-generator,it would still find the end. And the total amount of blue blocks is always the same,a straight line could never be reached,so this is not a random maze.



(c)

(i)

Implemented

(ii)

1.

```
%% INITIALIZATION ----
```

```
% Color map
```

```
cmap = [.12 .39 1;
```

```
1 1 1;
```

```
1 0 0;
```

```
1 .5 0;
```

```
.65 1 0;
```

```
0 0 0;
```

```
0 0 0;
```

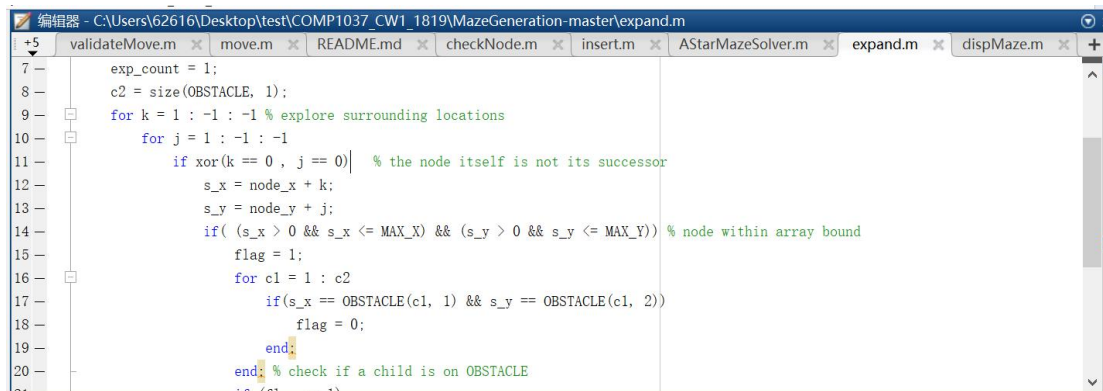
```
0 0 0;
```

```
.65 .65 .65];
```

Change number 2 into RED "1 0 0",pay attention to line 3.

2018/2019 COMP1037 Coursework 1 – Search Techniques

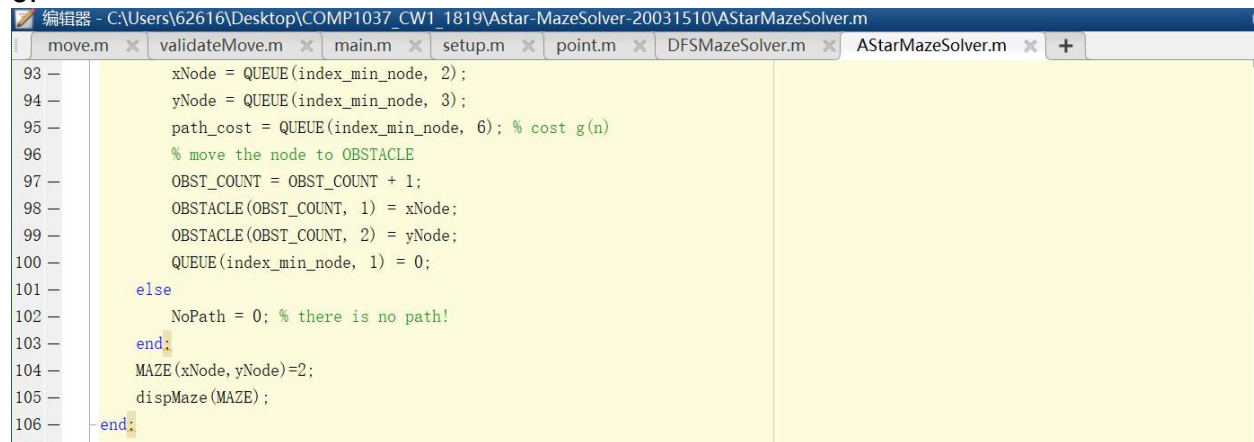
2.



```
7-      exp_count = 1;
8-      c2 = size(OBSTACLE, 1);
9-      for k = 1 : -1 : -1 % explore surrounding locations
10-         for j = 1 : -1 : -1
11-            if xor(k == 0 , j == 0) % the node itself is not its successor
12-               s_x = node_x + k;
13-               s_y = node_y + j;
14-               if( (s_x > 0 && s_x <= MAX_X) && (s_y > 0 && s_y <= MAX_Y)) % node within array bound
15-                  flag = 1;
16-                  for c1 = 1 : c2
17-                     if(s_x == OBSTACLE(c1, 1) && s_y == OBSTACLE(c1, 2))
18-                        flag = 0;
19-                     end;
20-                  end; % check if a child is on OBSTACLE
```

Watch for line 11, here to prevent the movement of diagonal, I change the code.

3.



```
93-      xNode = QUEUE(index_min_node, 2);
94-      yNode = QUEUE(index_min_node, 3);
95-      path_cost = QUEUE(index_min_node, 6); % cost g(n)
96-      % move the node to OBSTACLE
97-      OBST_COUNT = OBST_COUNT + 1;
98-      OBSTACLE(OBST_COUNT, 1) = xNode;
99-      OBSTACLE(OBST_COUNT, 2) = yNode;
100-      QUEUE(index_min_node, 1) = 0;
101-   else
102-      NoPath = 0; % there is no path!
103-   end;
104-   MAZE(xNode, yNode) = 2;
105-   dispMaze(MAZE);
106- end;
```

Put everything you discovered into 2, then everything will become red.

- (iii) implemented
- (iv) implemented
- (v) implemented

(d)

To implement DFS algorithm, we need to change the search method, we just expand the last available node in the queue, make sure this node is not expanded.

```
编辑器 - C:\Users\62616\Desktop\COMP1037_CW1_1819\DFS-MazeSolver-20031510\DFSMazeSolver.m*
move.m x validateMove.m x main.m x setup.m x point.m x DFSMazeSolver.m* x +
88
89     % DFS: find the last node in QUEUE expand it
90     for i=QUEUE_COUNT:-1:1
91         if QUEUE(i,1)==1
92             last=i;
93             break;
94         end
95     end

    % set current node (xNode, yNode) to the node with minimum f(n)
    xNode = aQUEUE(last, 2);
    yNode = QUEUE(last, 3);
    path_cost = QUEUE(last, 6); % cost g(n)
    % move the node to OBSTACLE
    OBST_COUNT = OBST_COUNT + 1;
    OBSTACLE(OBST_COUNT, 1) = xNode;
    OBSTACLE(OBST_COUNT, 2) = yNode;
    QUEUE(last, 1) = 0;
```

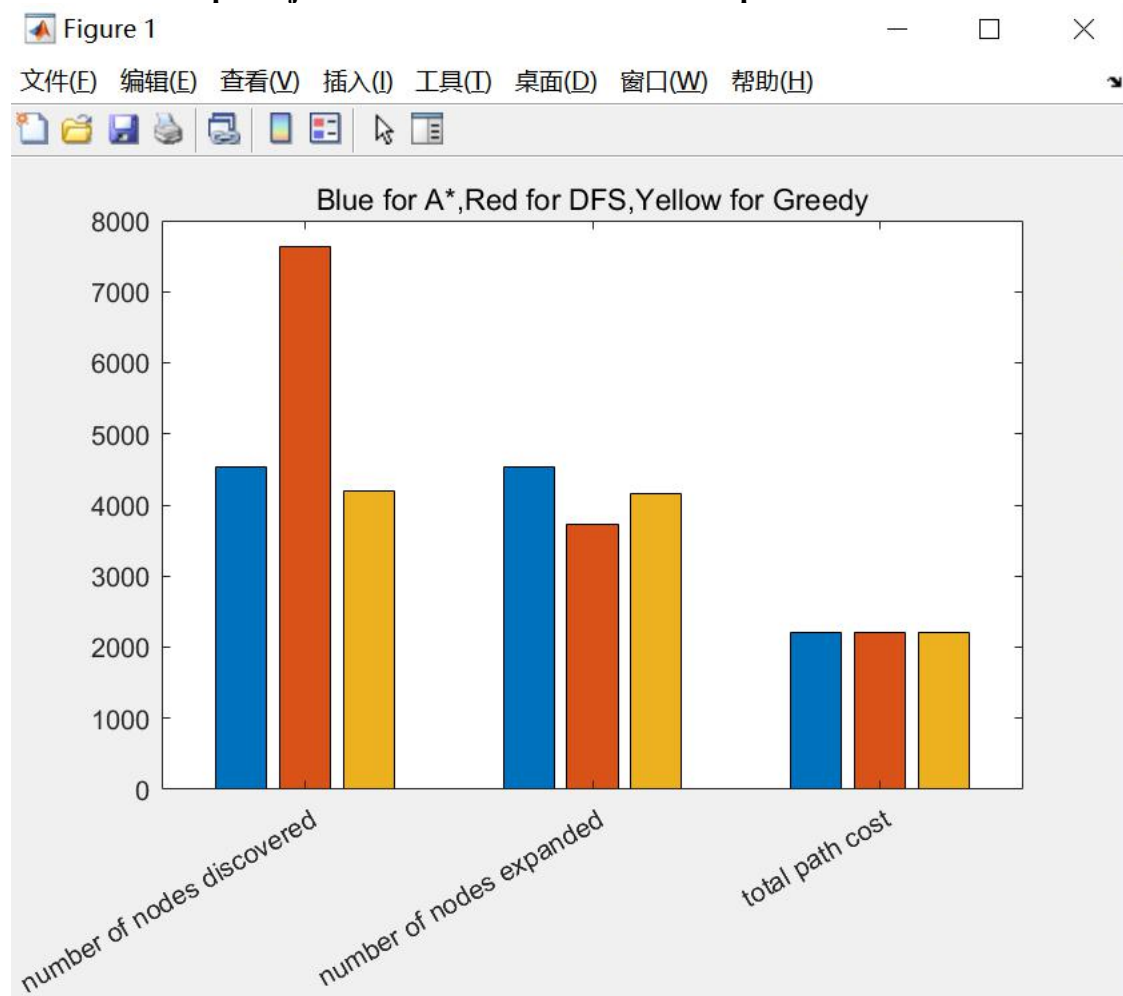
(e)

To implement greedy algorithm, instead of finding the minimal $f(n)$, we find the $h(n)$ which is the estimated cost to the goal. So in the `min_fn` function we change the value of the search to $h(n)$.

```
if size(temp_array) ~= 0
    [min_fn, temp_min] = min(temp_array(:, 7)); % index of the best node (lowest heuristic distance) in temp array
    i_min = temp_array(temp_min, 9); % return its index in QUEUE
else
    i_min = -1; % empty i.e no more paths are available.
end;
```

(f)

So basically in the queue we can find many quantities, the **total path cost** is $g(n)$ which is path cost, **numbers of nodes discovered** is all the nodes include the expanded nodes, they are either black or red so after you expand one node just add one time exp count, to find how much nodes expand just add one to a counter after expanded.



This is done by a 100x100 maze with difficulties of 10

Cost=[4537,7640,4196; 4532,3724,4161; 2215,2215,2215]