

2017/2018 COMP1037 Coursework 1 – Search Techniques

1. This part is based on the A* matlab demo (A_Star). You need to understand how this demo works and answer the following questions. **(6 marks)**
 - (a) You are required to implement Greedy Search and Uniform Cost Search algorithms based on the A* code (In the report, show what changes you have made and explain why you make these changes. You can use screenshot to demonstrate your code verification). (2 marks)

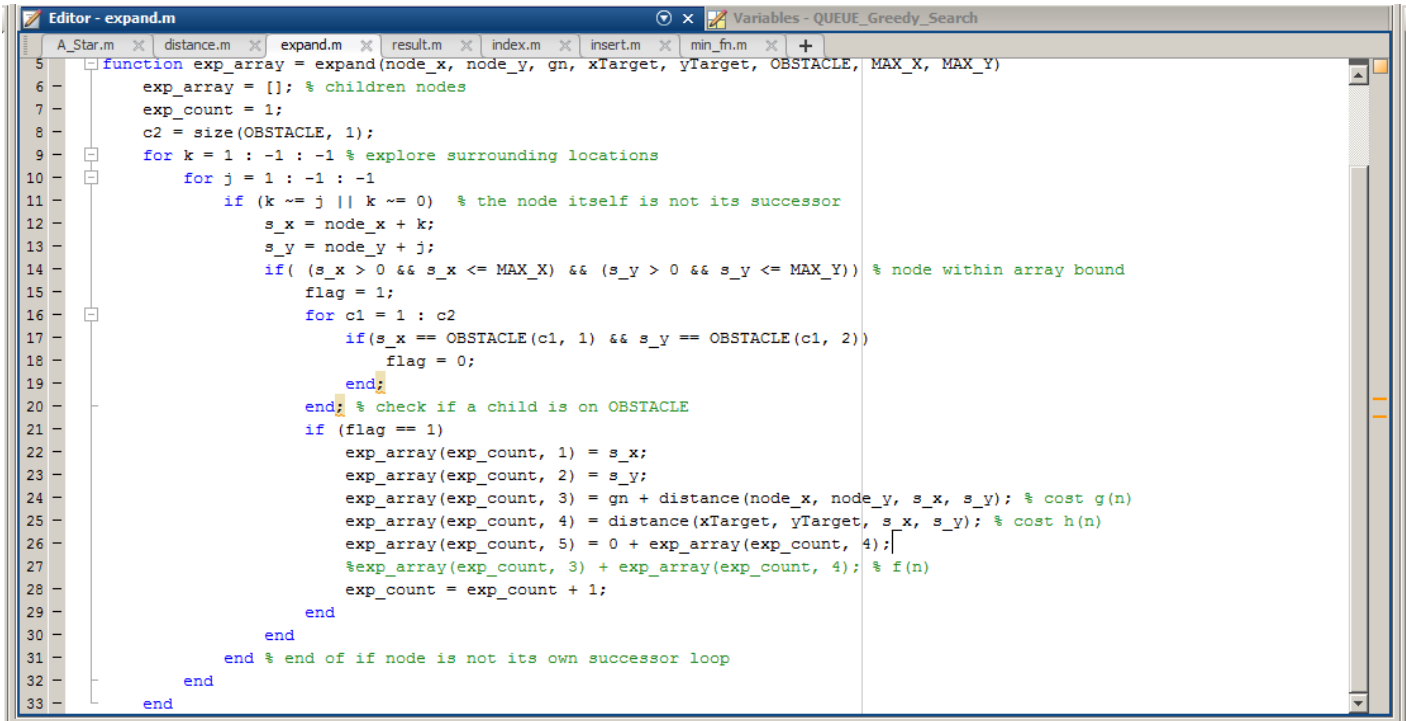
Solution:

In order to implement Greedy Search and Uniform Cost Search based on the A* algorithm code. First, the theoretical base in A* algorithm has been referred as follows:

$$f(n) = g(n) + h(n)$$

Here, $f(n)$ represents an evaluation of the state, which is the estimated cost of the cheapest solution through n . $g(n)$ stands for the cost from the initial state to the current state and $h(n)$ stands for the cost from the current state to a goal state.

First, the implementation of Greedy Search could be carried out by setting $g(n)$ into zero for further computation of $f(n)$ each time (to ensure each time $h(n)$ is too large thus dominates $g(n)$), which makes all the states are making local optimum decision, even if global optimum is not the combination of several local optimums.

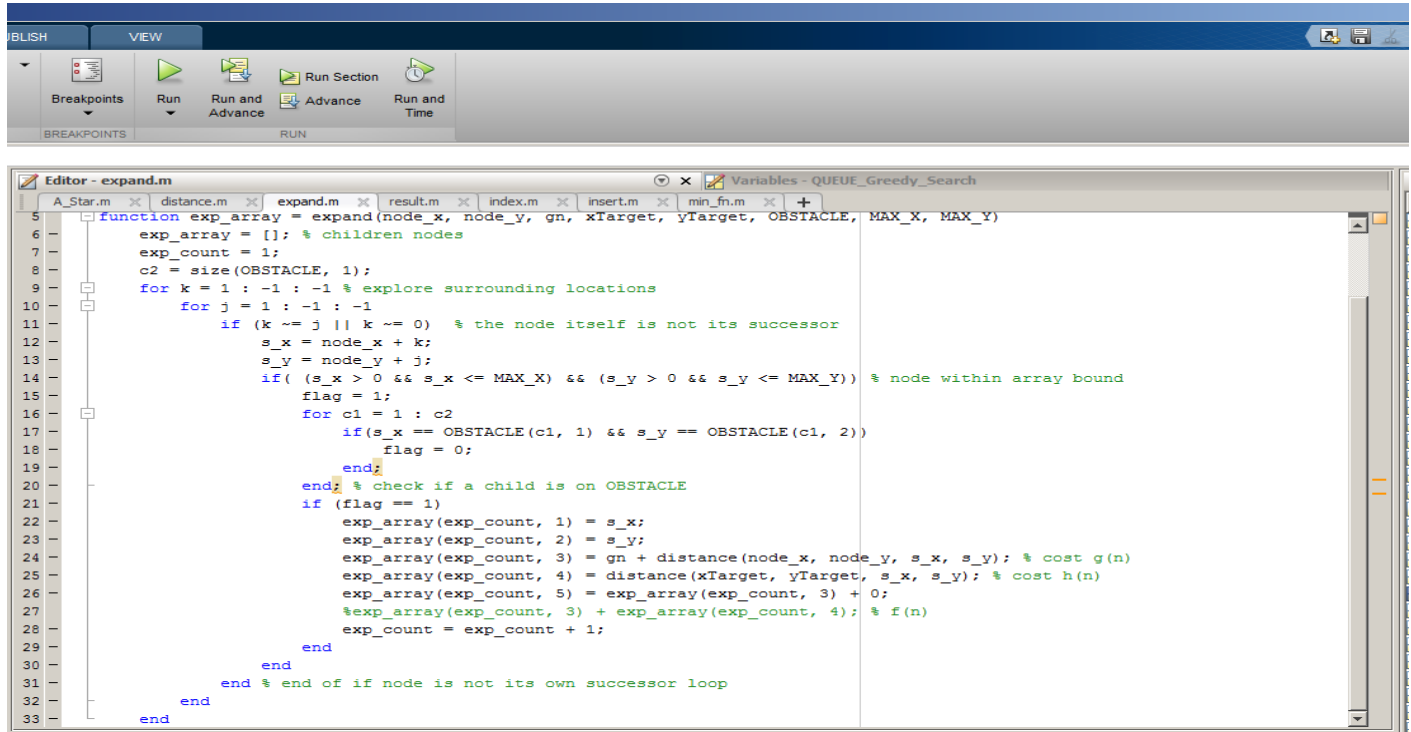


```
Editor - expand.m
A_Star.m distance.m expand.m result.m index.m insert.m min_fn.m +
Variables - QUEUE_Greedy_Search

5 function exp_array = expand(node_x, node_y, gn, xTarget, yTarget, OBSTACLE, MAX_X, MAX_Y)
6     exp_array = []; % children nodes
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 (k ~= j || k ~= 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
21                     if (flag == 1)
22                         exp_array(exp_count, 1) = s_x;
23                         exp_array(exp_count, 2) = s_y;
24                         exp_array(exp_count, 3) = gn + distance(node_x, node_y, s_x, s_y); % cost g(n)
25                         exp_array(exp_count, 4) = distance(xTarget, yTarget, s_x, s_y); % cost h(n)
26                         exp_array(exp_count, 5) = 0 + exp_array(exp_count, 4);
27                         %exp_array(exp_count, 3) + exp_array(exp_count, 4); % f(n)
28                         exp_count = exp_count + 1;
29                     end
30                 end
31             end % end of if node is not its own successor loop
32         end
33     end
```

2017/2018 COMP1037 Coursework 1 – Search Techniques

Also, when $h(n)$ has been set to zero in the summation, A* algorithm becomes Uniform Cost Search, because the cost between current state to a goal state didn't play a role during the decision making process, the code has been shown as follow:



```
function exp_array = expand(node_x, node_y, gn, xTarget, yTarget, OBSTACLE, MAX_X, MAX_Y)
exp_array = []; % children nodes
exp_count = 1;
c2 = size(OBSTACLE, 1);
for k = 1 : -1 : -1 % explore surrounding locations
    for j = 1 : -1 : -1
        if (k ~= j || k ~= 0) % the node itself is not its successor
            s_x = node_x + k;
            s_y = node_y + j;
            if (s_x > 0 && s_x <= MAX_X) && (s_y > 0 && s_y <= MAX_Y) % node within array bound
                flag = 1;
                for c1 = 1 : c2
                    if (s_x == OBSTACLE(c1, 1) && s_y == OBSTACLE(c1, 2))
                        flag = 0;
                    end;
                end; % check if a child is on OBSTACLE
                if (flag == 1)
                    exp_array(exp_count, 1) = s_x;
                    exp_array(exp_count, 2) = s_y;
                    exp_array(exp_count, 3) = gn + distance(node_x, node_y, s_x, s_y); % cost g(n)
                    exp_array(exp_count, 4) = distance(xTarget, yTarget, s_x, s_y); % cost h(n)
                    exp_array(exp_count, 5) = exp_array(exp_count, 3) + 0;
                    %exp_array(exp_count, 3) + exp_array(exp_count, 4); % f(n)
                    exp_count = exp_count + 1;
                end
            end
        end % end of if node is not its own successor loop
    end
end
```

- (b) You are required to use the matlab basics from the first lab session to show the evaluation results of the three searching methods (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)

Solution:

Analysis:

As for three main metrics, this report will explain each one individually about the methods of extracting. Firstly, the total path cost shall be stored into the variable "path_cost", which could be directly used in the results. Combined with the question 2, it's capable to get the path cost directly. If there is need for finding path cost in "QUEUE", the method is to find the minimum one in $g(n)$ whose first element is zero and $f(n)$ is zero. Secondly, "number of nodes discovered" and "number of nodes expanded" could be extracted through the variable "QUEUE". The first element in "QUEUE" is a state variable whose value is one or zero. The

2017/2018 COMP1037 Coursework 1 – Search Techniques

initialization code (highlighted) in the program show that zero is a state for the expanded node like this figure:

```
Editor - A_Star.m
A_Star.m  distance.m  expand.m  matlab_tutorial.m  +
This file can be opened as a Live Script. For more information, see Creating Live Scripts.

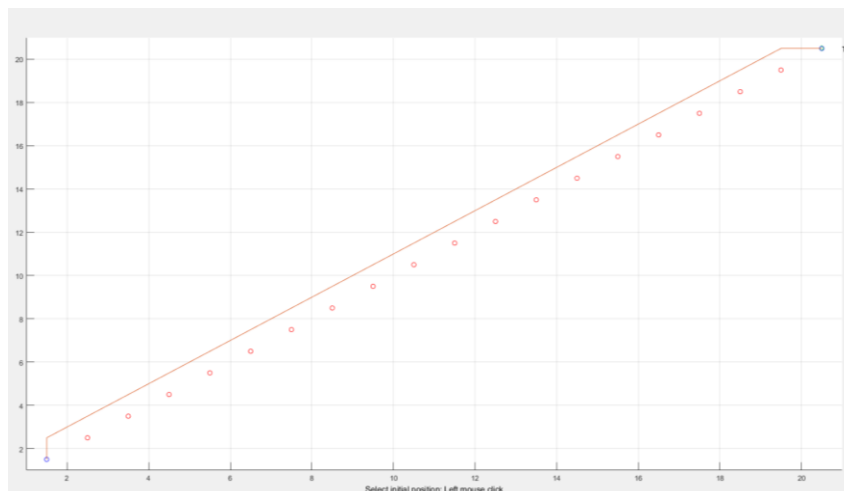
25 -     OBSTACLE(k, 2) = j;
26 -     k = k + 1;
27 - end
28 - end
29 - end
30 - OBST_COUNT = size(OBSTACLE, 1);
31 - OBST_COUNT = OBST_COUNT + 1;
32 - OBSTACLE(OBST_COUNT, :) = [xStart, yStart];
33 -
34 - %% add the starting node as the first node (root node) in QUEUE
35 - % QUEUE: [0/1, X val, Y val, Parent X val, Parent Y val, g(n),h(n), f(n)]
36 - xNode = xStart;
37 - yNode = yStart;
38 - QUEUE = [];
39 - QUEUE_COUNT = 1;
40 - NoPath = 1; % assume there exists a path
41 - path_cost = 0; % cost g(n): start node to the current node n
42 - goal_distance = distance(xNode, yNode, xTarget, yTarget); % cost h(n): heuristic cost of n
43 - QUEUE(QUEUE_COUNT, :) = insert(xNode, yNode, xNode, yNode, path_cost, goal_distance, goal_distance);
44 - QUEUE(QUEUE_COUNT, 1) = 0; % What does this do?
45 -
46 - %% Start the search
47 - while((xNode ~= xTarget || yNode ~= yTarget) && NoPath == 1)
48 -
49 -     % expand the current node to obtain child nodes
50 -     exp = expand(xNode, yNode, path_cost, xTarget, yTarget, OBSTACLE, MAX_X, MAX_Y);
51 -     exp_count = size(exp, 1);
```

So it's obvious that “number of nodes expanded” shall be the counter of all “QUEUE” variable's first elements' values are zero. Carefully, this report believes the final answer shall equal to the counter minus one, under the consideration of the start point has been particularly set to zero (like the highlighted code shown).

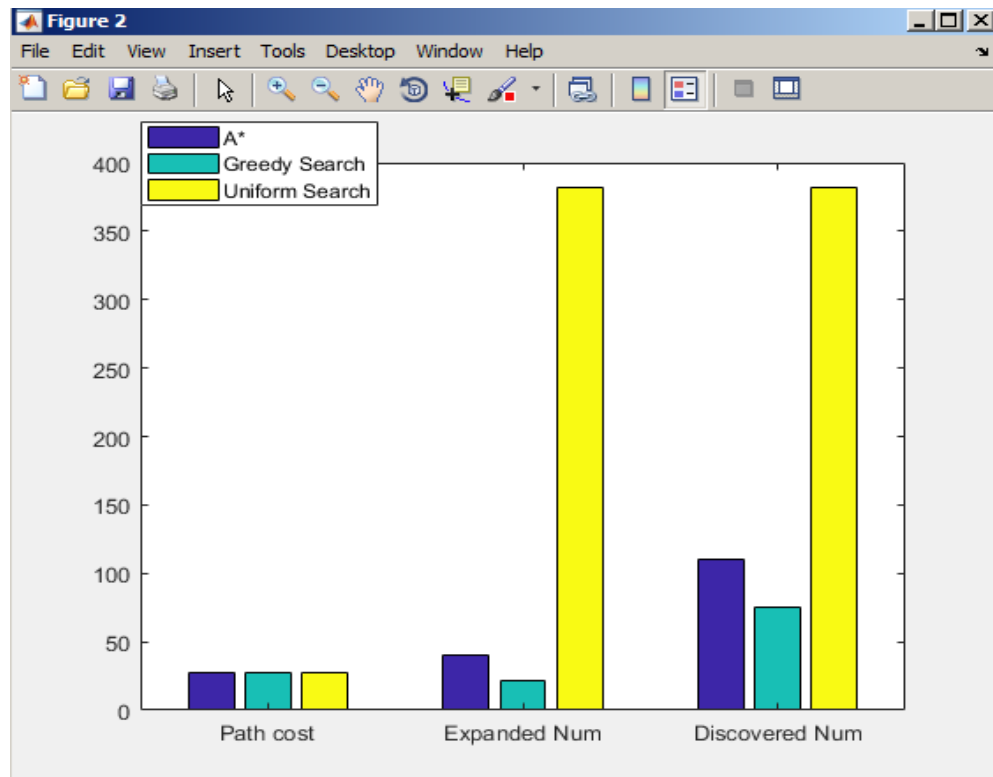
As for “number of nodes discovered”, this report believes that all the expanded nodes shall be countered into the nodes discovered, because all the expanded nodes shall be discovered first. Similarly, the start point shall be removed in this calculation process.

Evaluation:

First, we set the same situation of obstacles in all experiments, which is as follow:



And the final results are shown as follow:



- (c) Design and implement another heuristic h_2 which is different from the one (h_1) is used in the A* matlab code, explain how h_2 works and show what changes you have made to change the heuristic function from h_1 to h_2 . Is h_2 optimal? Why? Which heuristic is better? Why? (2 marks)

Solution:

First, this report is going to introduce three different method for evaluating the distance. Which are Manhattan distance, Chebyshev distance and Euclidean distance, which are corresponded to h_3 , h_2 and h_1 in the following figure. And this report deduced h_2 is the optimal heuristic function.

```

1  % This function calculates the distance between any two cartesian coordinates.
2  % Copyright 2009-2010 The MathWorks, Inc.
3
4  function dist = distance(x1, y1, x2, y2)
5
6      %dist = sqrt((x1 - x2)^2 + (y1 - y2)^2); % h1
7      dist = max(abs(x1-x2),abs(y1-y2)); %h2
8      %dist = abs(x1-x2)+abs(y1-y2); %h3
9

```

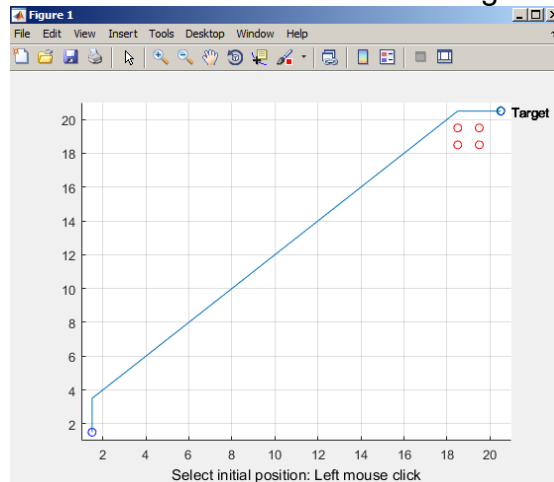
2017/2018 COMP1037 Coursework 1 – Search Techniques

In this paragraph, this report is going to explain why h2 is optimal. First, through checking the codes (in next figure) and testing with a few examples, it's obvious that this implementation has considered eight neighborhoods, which means they should be equal when it requires to measure the distance in eight different directions. So the Chebyshev distance is the best solution to abstract this situation.

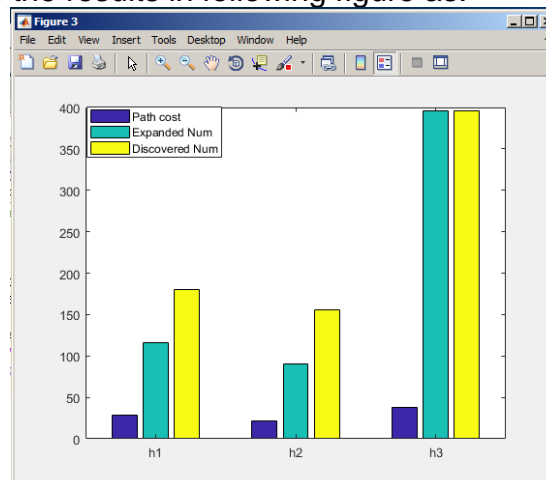
```
A_Star.m x distance.m x expand.m x PlotAnalysisResults.m x Untitled2.m x
1 % Function to take a node return an expanded array of chil
2 % None of the successors should be on OBSTACLE.
3 % Copyright 2009-2010 The MathWorks, Inc.
4
5 function exp_array = expand(node_x, node_y, gn, xTarget, 1
6 exp_array = []; % children nodes
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 (k ~= 1 || k ~= 0) % the node itself is not
```

Experiment:

By applying A* algorithm, the parameters are different heuristic functions h1, h2 and h3. All obstacles' situation are the same as the next figure like:



Also, the report plots the results in following figure as:



With analysis, it's easy to find out h2 has the smallest number of expanded and discovered nodes, which proves that h2 is the most precise method for abstracting the distance in this situation, equivalently the optimal $h(n)$. Notice that different path cost is due to different evaluation standard, so they are not equal numerically here.

2. 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 depth-first approach to randomly generate a maze with user defined size and difficulty. **(9 marks)**

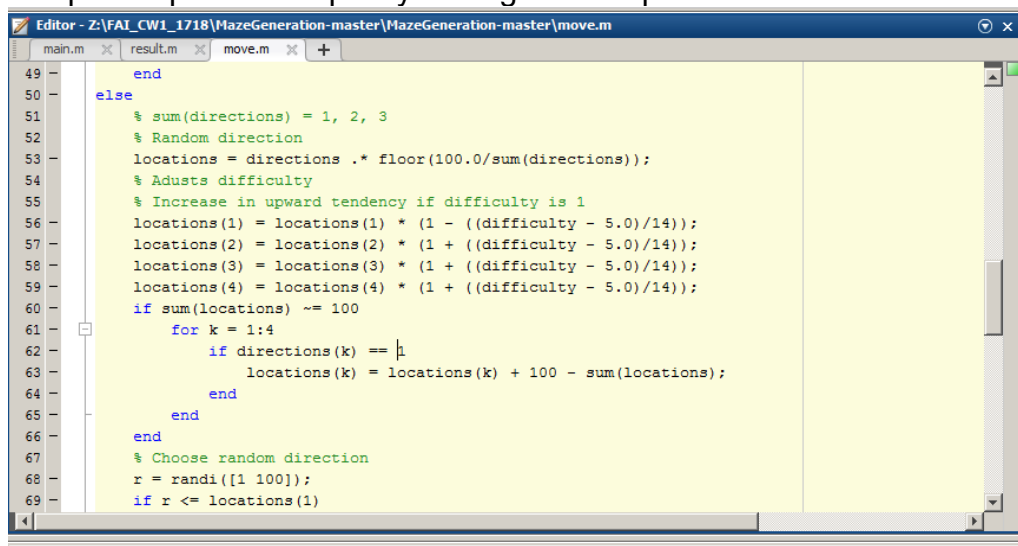
(a) In the demo code, show which line(s) of code is used to implement depth-first approach, explain the logic the student adopts to generate the maze. (2 marks)

Solution:

In the main function, this “calculation” part is the main part of implement depth-first approach, which is constructed with “move” function and “dispMaze” function. And this part aims to explain the details in move function.

```
29 %% CALCULATIONS ---
30 [maze, nodes, position, endPoint] = setup(size);
31 % Runs generation process until there are no more nodes left
32 while numel(nodes) > 0
33     [maze, position, nodes] = move(maze, position, nodes, difficulty);
34     dispMaze(maze);
35 end
36 maze = adjustEnd(maze, endPoint);
37
```

In views of this report, the most important part for maze generator is shown. And it's quite important to specify the logic in this part.



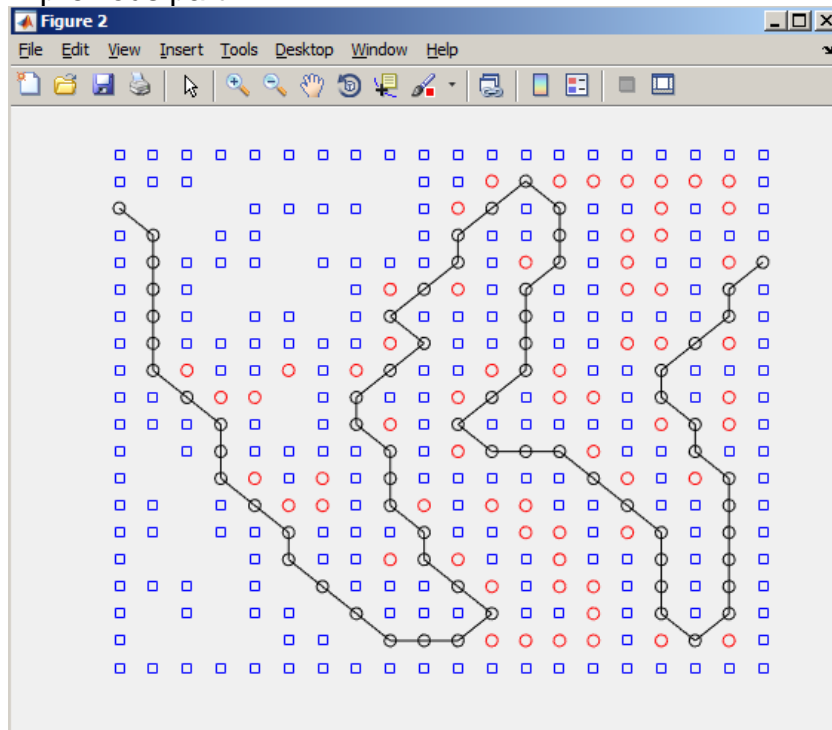
In this maze generator, there are four directions (up, down, left, right) to

randomize. By using the input of difficulty, different values would be generated based on the percentage based increase. Then, randomly, choose the direction to make next move, if it's valid.

(b) Identify the problem of this maze generator if there is any. (1 marks)

Solution:

In the current A star algorithm situation, there is an eight-neighborhood state, which means there are eight choices for a current state to move. But this maze generator only covers four. So when implementing the most optimal one, it may be better to change the heuristic function by using Manhattan distance. But there is still possible to give an eight-neighborhood solution. In the following figure, the choices near the target position (final black circle in the left) shows the left-down is the best solution. So still, here in question c, this report uses h2 which is mentioned in previous part.



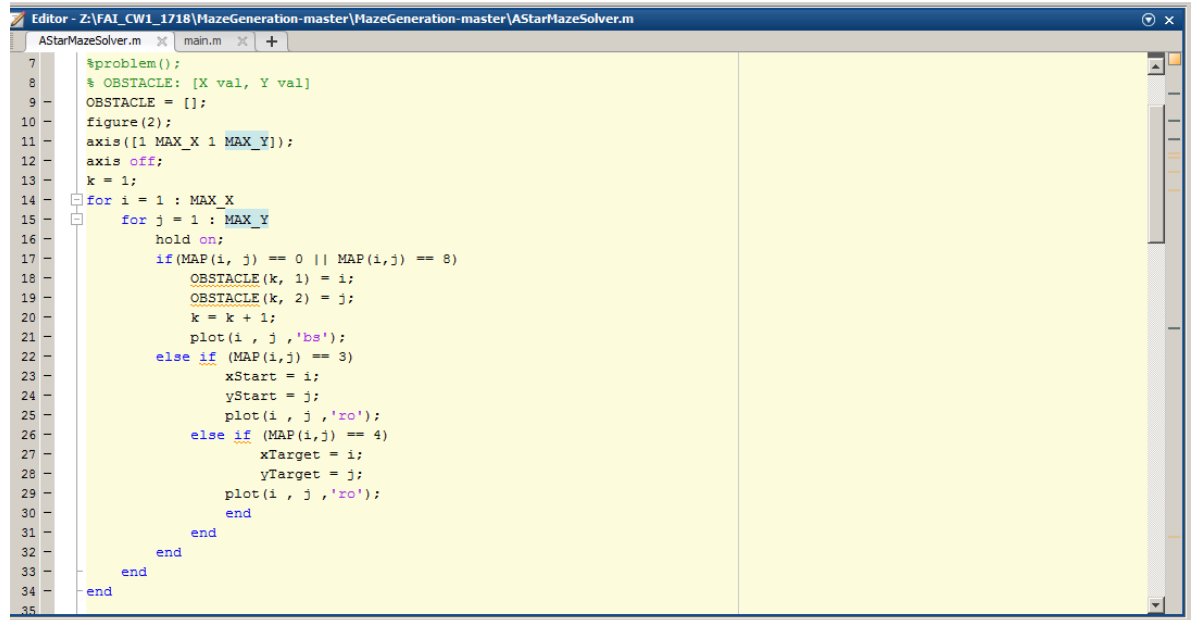
(c) Write a maze solver using A* algorithm. (6 marks)

Solution:

In this part, this report will individually explain how finish different tasks as the questions' order.

i) The solver need be called by command '**AStarMazeSolver(maze)**'

Based on the “AStar.m”, the function “AStarMazeSolver” has been created. Instead using the “problem.m”, this function takes the maze as a parameter as the input. Then the initial process is shown as following figure, including plotting part.



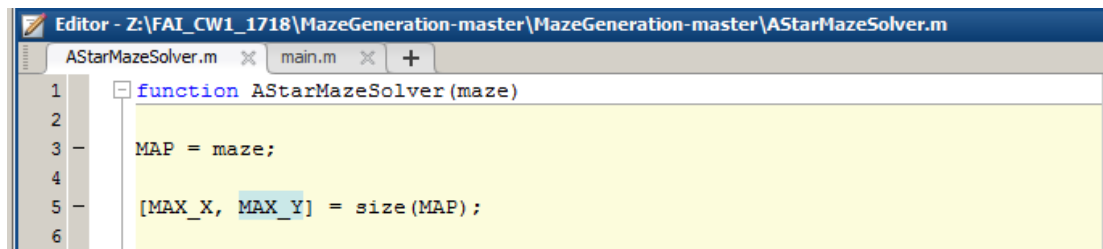
```
7 %problem();
8 % OBSTACLE: [X val, Y val]
9 OBSTACLE = [];
10 figure(2);
11 axis([1 MAX_X 1 MAX_Y]);
12 axis off;
13 k = 1;
14 for i = 1 : MAX_X
15     for j = 1 : MAX_Y
16         hold on;
17         if (MAP(i, j) == 0 || MAP(i,j) == 8)
18             OBSTACLE(k, 1) = i;
19             OBSTACLE(k, 2) = j;
20             k = k + 1;
21             plot(i, j, 'bs');
22         else if (MAP(i,j) == 3)
23             xStart = i;
24             yStart = j;
25             plot(i, j, 'ro');
26         else if (MAP(i,j) == 4)
27             xTarget = i;
28             yTarget = j;
29             plot(i, j, 'ro');
30         end
31     end
32 end
33 end
34 end
35
```

All borders and obstacles would be marked with blue squares meanwhile.

Note: the maze has shown there is a need to transform between x and y.
But this report thought it's no harms to the solution so this report didn't implement it. If it's required, it could be achieved through mathematics methods like setting x with MAX_X minus i instead of i.

- ii) The maze solver should be able to solve any maze generated by the maze generator

In order to fix this, as the previous figure shown, different sizes of mazes from maze generator will be taken into the consideration into the initial part. Especially, the next figure shows how to process this functionality.



```
1 function AStarMazeSolver(maze)
2
3     MAP = maze;
4
5     [MAX_X, MAX_Y] = size(MAP);
6
```

- iii) The maze solver should be able to find the optimal solution.

By applying the A star algorithm with the optimal heuristic function, it has been realized finally.

- iv) Your code need display the all the routes that A* has processed with RED color.

Before plotting the optimal solution, every nodes which A* has processed will be marked with red circles in advance.

- v) Your maze solver should be about to display the final result BLACK color.

By taking advantages of original “result.m”, this could be realized and finally all the optimal nodes would be marked with black lines and circles to demonstrate the optimal solution.