Question 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)

According to the MazeGeneration-master code, the routes are generated by a randomized version of the DFS algorithm. The lines from 32-36 of code is used to implemented 'move' function in the file main.m, which is the access to the implement of tree search approach. These lines implement the generation process using number of elements in 'nodes' as the condition of the while-loop. Inside the loop, 'move' function is called with the inputs of 'maze', 'position', 'nodes', 'difficulty' every time and outputs the updated version of 'maze', 'position', 'nodes', which are passed as arguments for next loop. The process is continued until there are no more nodes left. Finally, the exit is connected to the rest of the maze by 'adjustEnd' function.

The file move.m contains the details of how 'move' function is worked with DFS algorithm and how relevant data structure is implemented. The lines from 43-98 of code is used to implement tree search approach. The structure of 'nodes' is a 2 by n matrix, where n is number of nodes added into the list. The first row contains all the row values of nodes and the second row contains all the column values of nodes. The algorithm adds or removes a node from the end of the list. Starting from a random node, a random neighboring node that has not yet been visited is selected. Then a random direction which is accessible is selected to reach a new node. This new node is marked as visited an its information is stored for backtracking. This process continues until a node with no unvisited neighbor is discovered. Then another node with an unvisited neighbor is met via backtracking and a new path is generated. Above steps are repeated until every node is visited.

Question B

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

In this algorithm, that the function is recursively called may cause stack overflow issues. Every time, a list of nodes will be generated with a series of backtracking information (position) and be required to store in some spaces. The algorithm can be rearranged into a loop by storing relevant 'position' in the maze itself, which means the computer doesn't have to compile information from other spaces resulting in less time and space complexity.

According to the DFS algorithm, it searches the node as far as possible along each branch until every child has been visited, resulting a low average branching factor and contain many long paths.

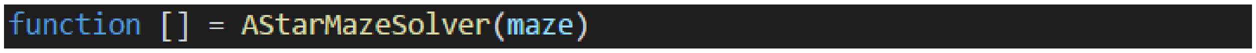
Question C

Write a maze solver using A* algorithm. (5 marks)

Based on the AStar demo, some parts of the code have been changed as follows:

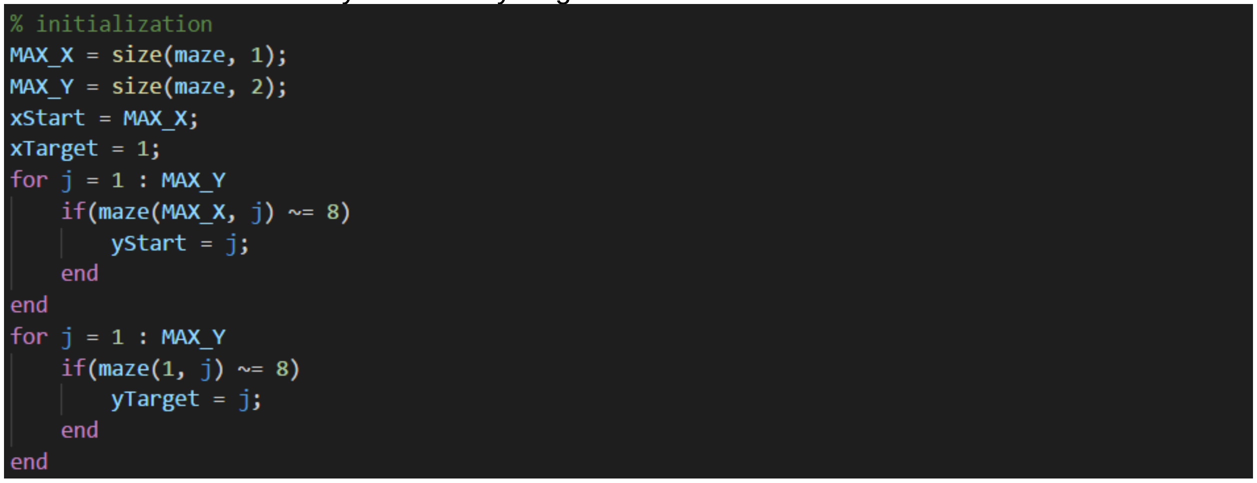
a) Changes in file AstarMazeSolver.m

In order to call the solver by command 'AStarMazeSolver(maze)' within the Matlab command window, a function is declared in the top of this file with 'maze' as the only input, which is assumed to be generated by the maze generator in advance.



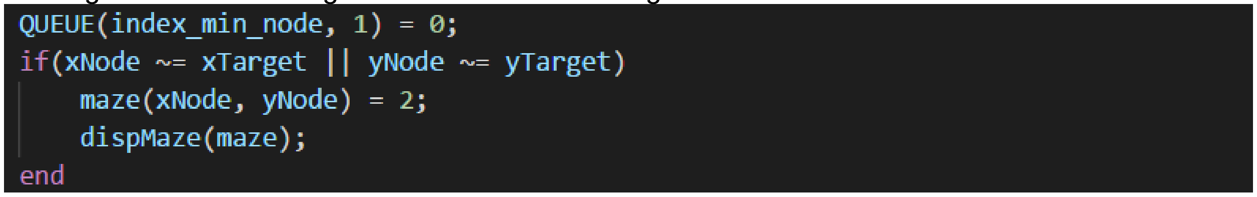
```
function [] = AStarMazeSolver(maze)
```

The initial step is to find the starting point and the ending point. Since the maze is a square, the build-in function 'size' is used to calculate the bound. Here, the 'xStart' is assigned to be MAX_X and 'xTarget' is assigned to be 1. The two for-loops is used to traverse all the nodes on the bounds to find the 'yStart' and 'yTarget'.



```
% initialization
MAX_X = size(maze, 1);
MAX_Y = size(maze, 2);
xStart = MAX_X;
xTarget = 1;
for j = 1 : MAX_Y
    if(maze(MAX_X, j) ~= 8)
        yStart = j;
    end
end
for j = 1 : MAX_Y
    if(maze(1, j) ~= 8)
        yTarget = j;
    end
end
```

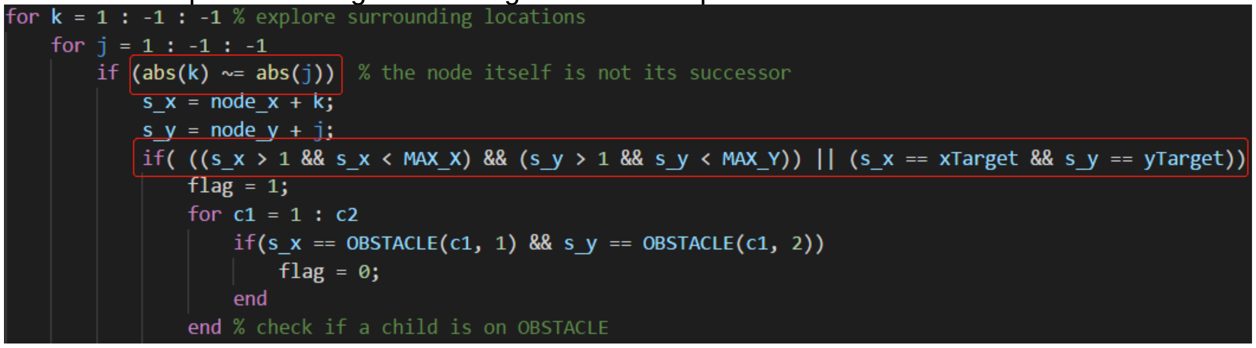
The 'OBSTACLE' contains all the nodes that cannot be passed, represented by the blue color with the code 'maze(i, j) == 0;'. In the search process using a while-loop, similar codes are implemented to display and mark the passing nodes. Information of the node is inserted into the 'QUEUE'. The node is displayed to be red by 'maze(i, j) == 2;' and marked to be a visited node by 'QUEUE(index, 1) = 0'; Here, if-statement is used to make sure that the colors of starting node and the target node cannot be changed.



```
QUEUE(index_min_node, 1) = 0;
if(xNode ~= xTarget || yNode ~= yTarget)
    maze(xNode, yNode) = 2;
    dispMaze(maze);
end
```

b) Changes in file expand.m

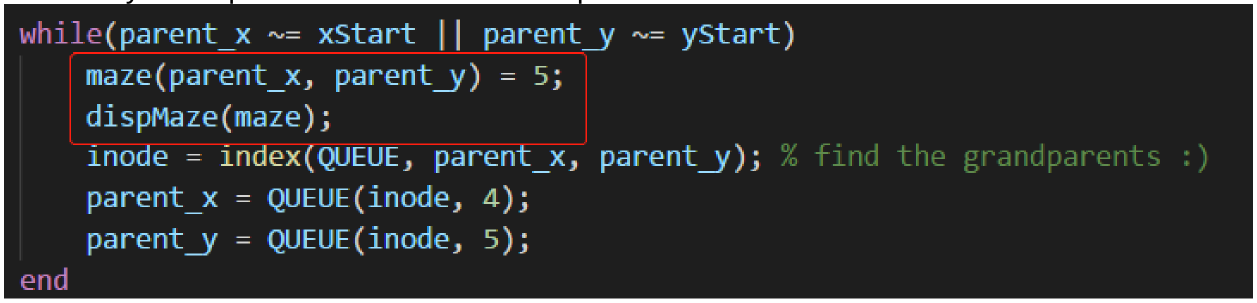
Because the node(x, y) is expanded to find its children and determine whether (x-1, y), (x+1, y), (x, y+1), (x, y-1) are its children or not, the nested for-loop is used. Meanwhile, the nodes on the corner and itself will not be checked. Also, the case that border-nodes should be excluded except the starting and ending nodes. The updated codes are shown below:



```
for k = 1 : -1 : -1 % explore surrounding locations
    for j = 1 : -1 : -1
        if (abs(k) ~= abs(j))  % the node itself is not its successor
            s_x = node_x + k;
            s_y = node_y + j;
            if( ((s_x > 1 && s_x < MAX_X) && (s_y > 1 && s_y < MAX_Y)) || (s_x == xTarget && s_y == yTarget))
                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
```

c) Changes in file result.m

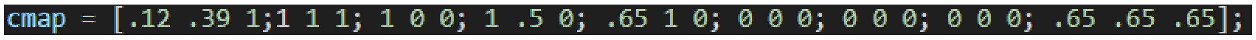
To find the final path, the last node is backtracked to its parent nodes until the starting node is reached. Codes related to ‘Optimal_path’ are removed to avoid some extra steps. The final path can be obtained by accessing nodes and its parent nodes stored in the QUEUE. Codes regarding ‘plot’ are replaced by color-changing steps. The node on the final path is displayed in black by the implementation of a while-loop.



```
while(parent_x ~= xStart || parent_y ~= yStart)
    maze(parent_x, parent_y) = 5;
    dispMaze(maze);
    inode = index(QUEUE, parent_x, parent_y); % find the grandparents :)
    parent_x = QUEUE(inode, 4);
    parent_y = QUEUE(inode, 5);
end
```

d) Changes in file dispMaze.m

Different types of nodes should be displayed by different colors. In this file, the corresponding Matlab-RGB values are changed in the color map. For example, if maze(i, j) == 2, the node will be red.



```
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];
```

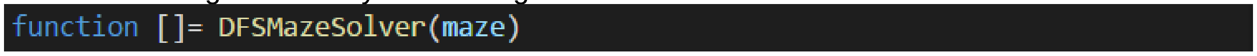
Question D

Based on the previous AStarMazeSolver, implement a maze solver using the DFS algorithm. The solver is called and ran by command ‘DFSMazeSolver(maze)’. (3 marks)

Based on the code of AStarMazeSolver, some parts of the code have been changed as follows:

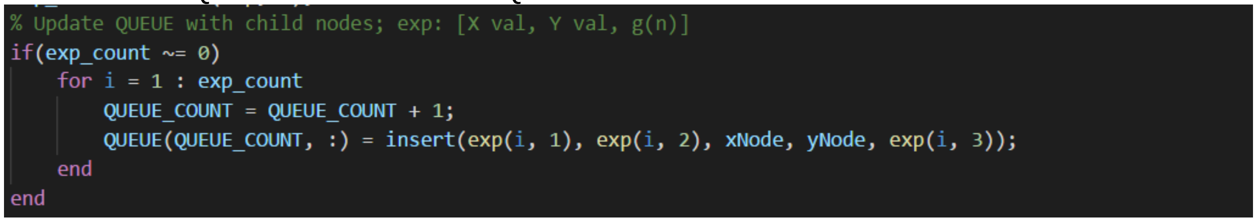
a) Changes in file DFSMazeSolver.m

In order to call the solver by command ‘DFSMazeSolver(maze)’ within the Matlab command window, a function is declared in the top of this file with ‘maze’ as the only input, which is assumed to be generated by the maze generator in advance.



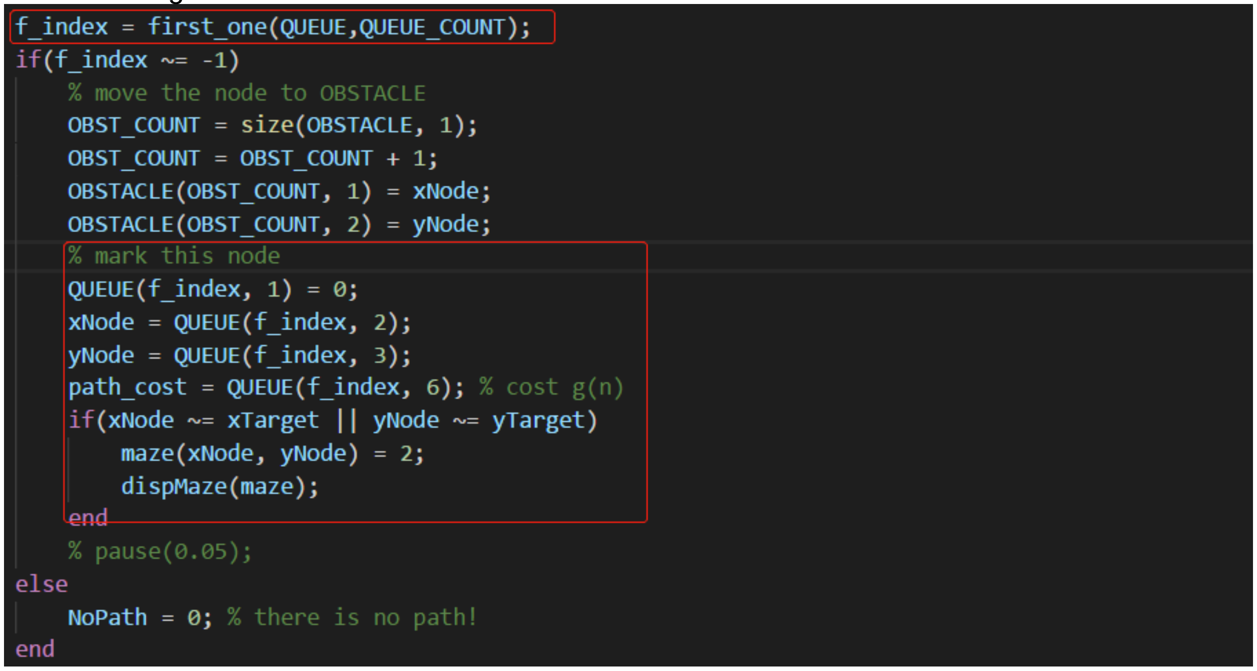
```
function []= DFSMazeSolver(maze)
```

The DFS algorithm is different from the A* Algorithm that the optimal path is not guaranteed to be found. Thus, the codes related to ‘goal_distance’, ‘hn’, ‘fn’ are deleted. According to the DFS algorithm, the first node in ‘QUEUE’ is expanded every time in a while-loop <u>(Here, the bottom of the node is regarded as the first node)</u>. The nodes are added from the bottom of the ‘QUEUE’. This node is marked by ‘QUEUE(i, 0)’, indicating that this node has been discovered and expanded. If this node has children, then all the children will be added from the bottom of ‘QUEUE’ to form a new ‘QUEUE’. The codes are as follows:



```
% Update QUEUE with child nodes; exp: [X val, Y val, g(n)]
if(exp_count ~= 0)
    for i = 1 : exp_count
        QUEUE_COUNT = QUEUE_COUNT + 1;
        QUEUE(QUEUE_COUNT, :) = insert(exp(i, 1), exp(i, 2), xNode, yNode, exp(i, 3));
    end
end
```

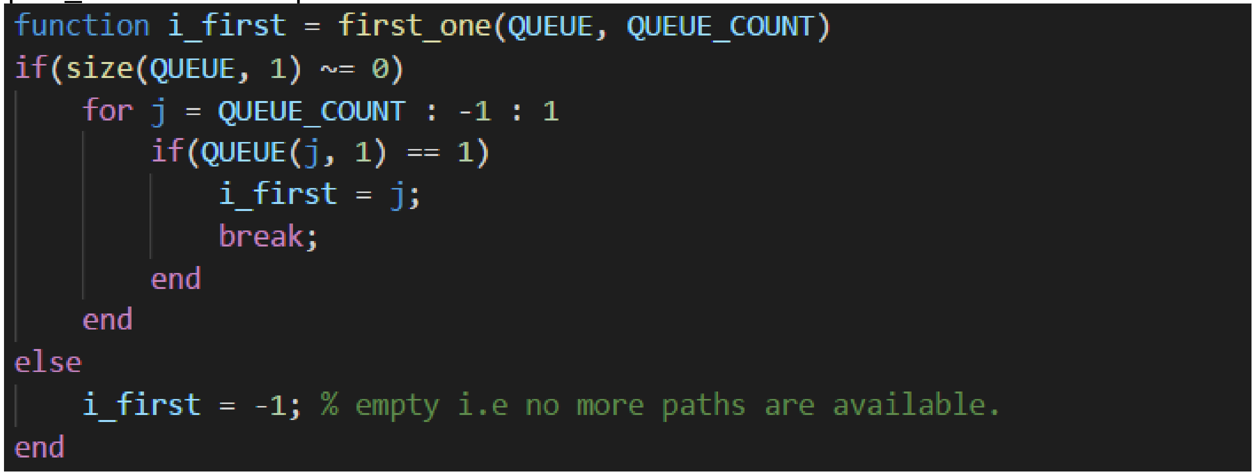
Otherwise, the second node in the ‘QUEUE’ which is not marked ‘QUEUE(i, 0)’ will be expanded in the next loop. This node in the ‘QUEUE’ will be found by ‘first_one’ funcition and its value of ‘xNode’, ‘yNode’, ‘path_cost’ are prepared for the next loop. Also, the color of the node is changed to red.



```
f_index = first_one(QUEUE,QUEUE_COUNT);
if(f_index ~= -1)
    % move the node to OBSTACLE
    OBST_COUNT = size(OBSTACLE, 1);
    OBST_COUNT = OBST_COUNT + 1;
    OBSTACLE(OBST_COUNT, 1) = xNode;
    OBSTACLE(OBST_COUNT, 2) = yNode;
    % mark this node
    QUEUE(f_index, 1) = 0;
    xNode = QUEUE(f_index, 2);
    yNode = QUEUE(f_index, 3);
    path_cost = QUEUE(f_index, 6); % cost g(n)
    if(xNode ~= xTarget || yNode ~= yTarget)
        maze(xNode, yNode) = 2;
        dispMaze(maze);
    end
    % pause(0.05);
else
    NoPath = 0; % there is no path!
end
```

b) Changes in file min_fn.m

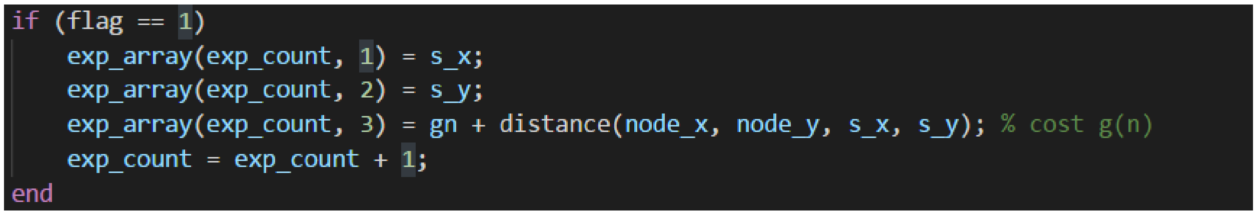
Here, the ‘min_fn’ is changed to ‘first_one’ function aiming to find the first node which is not marked ‘QUEUE(i, 0)’ starting form the bottom <u>(Here, the bottom node is regarded as the first node)</u>. It returns the index of that node in the ‘QUEUE’ for updating ‘xNode’, yNode’ and ‘path_cost’ for next loop in main.m.



```
function i_first = first_one(QUEUE, QUEUE_COUNT)
if(size(QUEUE, 1) ~= 0)
    for j = QUEUE_COUNT : -1 : 1
        if(QUEUE(j, 1) == 1)
            i_first = j;
            break;
        end
    end
else
    i_first = -1; % empty i.e no more paths are available.
end
```

c) Changes in file expand.m

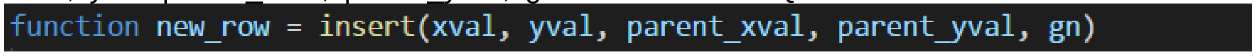
Since the distance between current node and target node will not be considered in the case of DFS algorithm, only ‘xChild’, ‘yChild’ and ‘gn’ will be stored in ‘exp_array’ (‘gn’ will be used for the last question).



```
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_count = exp_count + 1;
end
```

d) Changes in file insert.m

‘hn’, ‘fn’ are not considered in the case of DFS algorithm as the ‘expand’ function does. Only ‘xval’, ‘yval’ ‘parent_xval’, ‘parent_yval’, ‘gn’ are inserted to ‘QUEUE’.



```
function new_row = insert(xval, yval, parent_xval, parent_yval, gn)
```

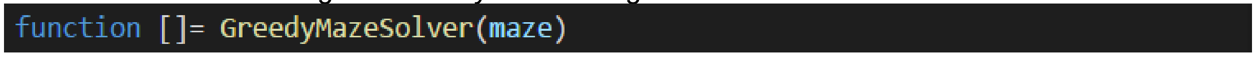
Question E

Based on the previous AStarMazeSolver, implement a maze solver using the Greedy search algorithm. The solver is called by command ‘GreedyMazeSolver(maze)’. (1 marks)

Based on the code of AStarMazeSolver, some parts of the code have been changed as follows:

a) Changes in file GreedyMazeSolver.m

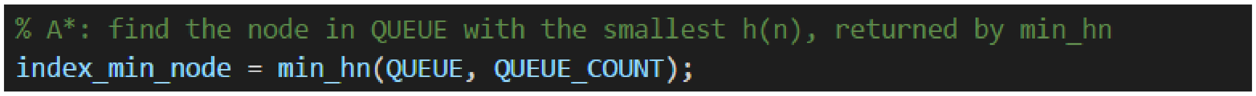
In order to call the solver by command ‘GreedyMazeSolver(maze)’ within the Matlab command window, a function is declared in the top of this file with ‘maze’ as the only input, which is assumed to be generated by the maze generator in advance.



```
function []= GreedyMazeSolver(maze)
```

In the initialization step, the codes related to ‘fn’ is removed, including parameters in some functions. According to the Greedy Search Algorithm, only should the heuristic cost ‘hn’ be considered in order to choose the next node to be expanded.

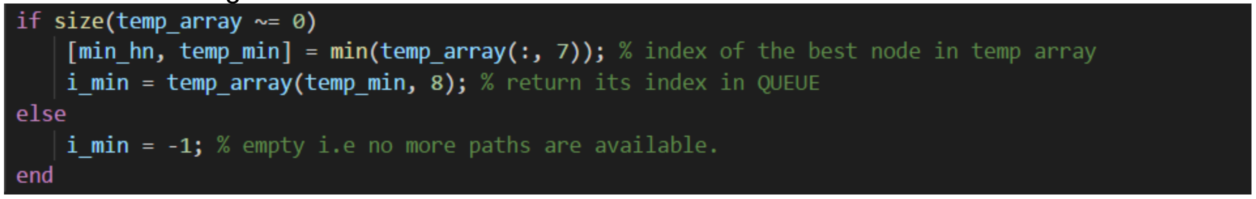
Since ‘hn’ is the only variable we need to focus on, every time a node is expanded, its children are inserted into ‘QUEUE’. Here, a new function ‘min_hn’, based on ‘min_fn’ function in the original code, is used to obtain the index of unvisited node with minimum ‘hn’ in the ‘QUEUE’. With its index, this node is regarded as the next node to be expanded and is put into ‘OBSTACLE’.



```
% A*: find the node in QUEUE with the smallest h(n), returned by min_hn
index_min_node = min_hn(QUEUE, QUEUE_COUNT);
```

b) Changes in file min_fn.m

Based on ‘min_fn’ function, ‘min_hn’ function is set up to to return the index of the node with minimum h(n) in QUEUE, because comparing ‘hn’ of different nodes is the benchmark to choose where to go next.

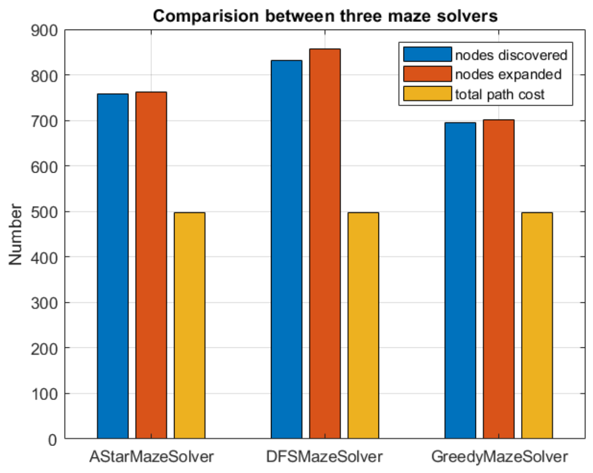


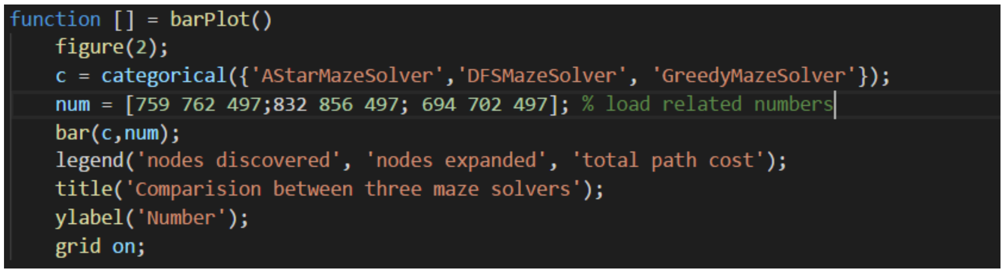
```
if size(temp_array ~= 0)
    [min_hn, temp_min] = min(temp_array(:, 7)); % index of the best node in temp array
    i_min = temp_array(temp_min, 8); % return its index in QUEUE
else
    i_min = -1; % empty i.e no more paths are available.
end
```

Question 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)

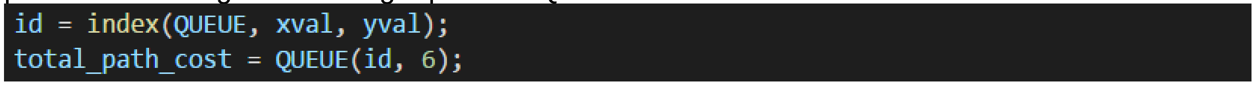
For this question, the evaluation results of the three searching methods are based on a maze with the size of 50*50 and the difficulty of 5.





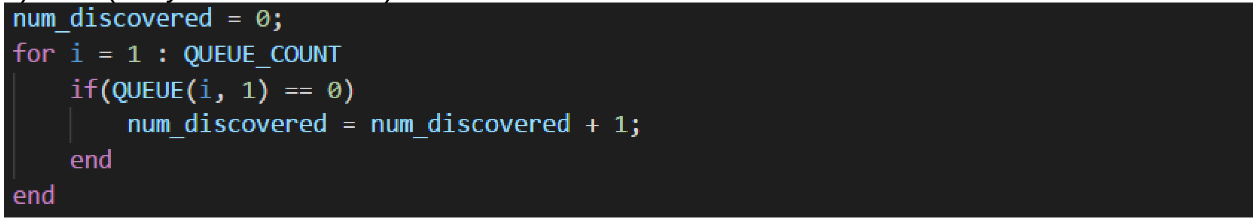
```
function [] = barPlot()
    figure(2);
    c = categorical({'AStarMazeSolver','DFSMazeSolver', 'GreedyMazeSolver'});
    num = [759 762 497;832 856 497; 694 702 497]; % load related numbers
    bar(c,num);
    legend('nodes discovered', 'nodes expanded', 'total path cost');
    title('Comparision between three maze solvers');
    ylabel('Number');
    grid on;
```

1) The total path cost is obtained by the code 'QUEUE(idOfTarget, 6)', because the information of the target node is stored in 'QUEUE'. In the process of 'expand', the current path cost stored in QUEUE will be accumulated and passed to its children. Hence, the total path cost is the 'gn' of the target point in 'QUEUE'.



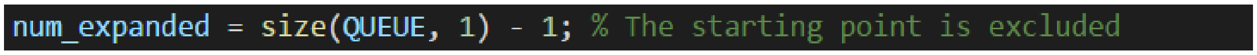
```
id = index(QUEUE, xval, yval);
total_path_cost = QUEUE(id, 6);
```

2) The number of nodes discovered is the number of nodes in 'QUEUE' whose QUEUE(index, 1) == 0 (They are in red color).



```
num_discovered = 0;
for i = 1 : QUEUE_COUNT
    if(QUEUE(i, 1) == 0)
        num_discovered = num_discovered + 1;
    end
end
```

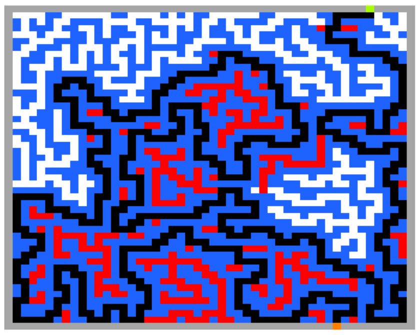
3) The number of nodes expanded is the number of nodes stored in the 'QUEUE' minus 1, because every expanded child is added into the 'QUEUE' and the starting node is not a child.

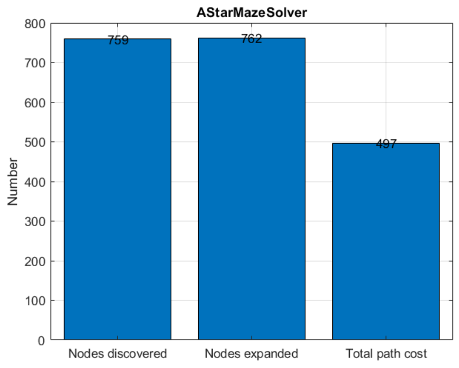


```
num_expanded = size(QUEUE, 1) - 1; % The starting point is excluded
```

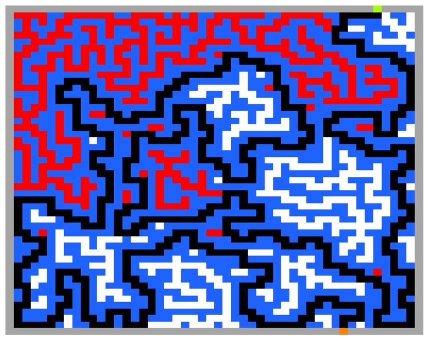
Appendix – evaluation maze (the size of 50*50 and the difficulty of 5)**:**

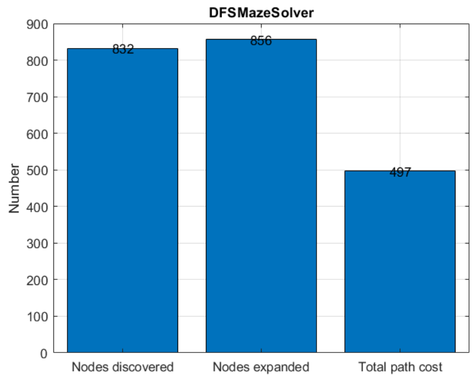
1.AStarMazeSolver



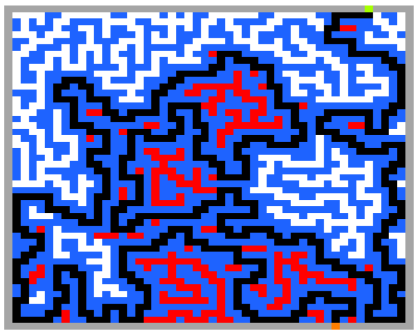


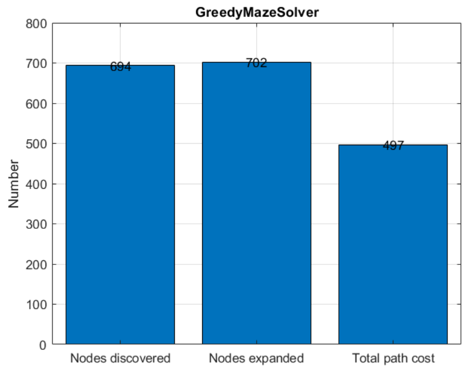
2.DFSMazeSolver





3. GreedyMazeSolver




GreedyMazeSolver
Number
Nodes discovered
Nodes expanded
Total path cost