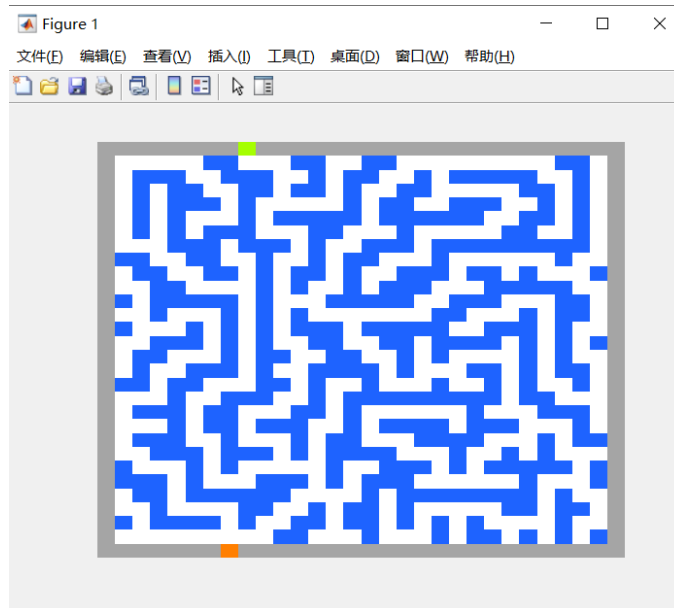


BFS Maze Generator:



(This is the maze generated by BFS algorithm)

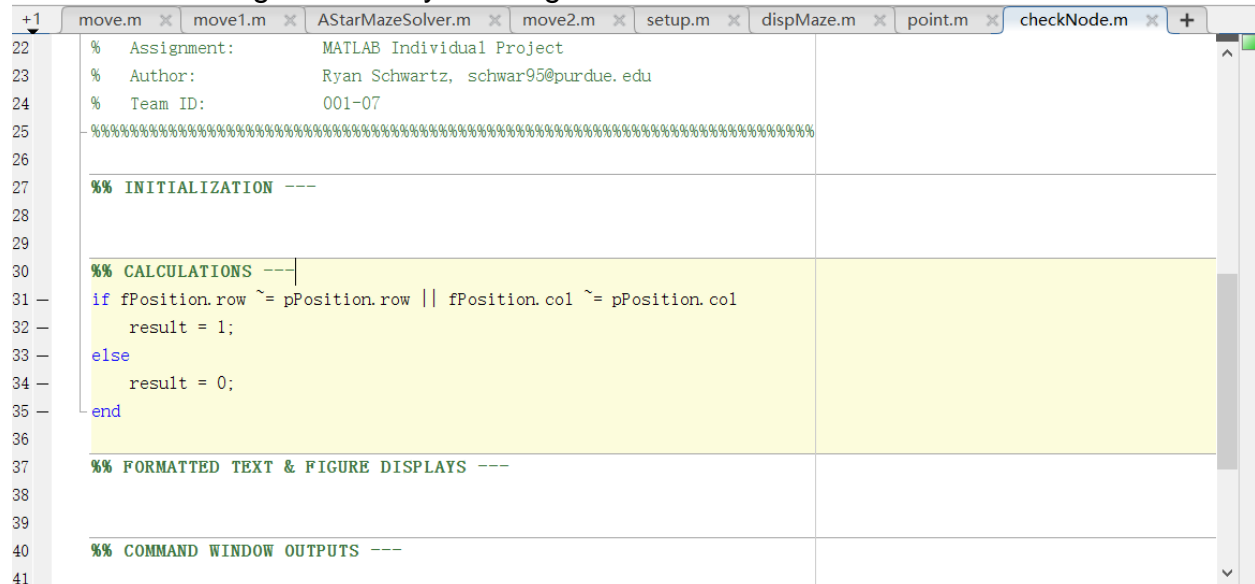
```
编辑器 - C:\Users\79993\Documents\MATLAB\BFSMazeGeneration-master\move.m
main.m  move.m  move1.m  AStarMazeSolver.m  move2.m  setup.m  dispMaze.m  +
34  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35
36  %% INITIALIZATION ---
37  % directions = [up, down, left, right]
38  % up, down, left, right can be 1 for OK or 0 for NO
39  [directions] = validateMove(maze, position);
40
41  %% CALCULATIONS ---
42  % Check if that route can continue or not
43  if any(directions) == 0
44      position = point(nodes(1, 1), nodes(2, 1));
45      nodes = nodes(:, 2 : end );
46  else
47      % sum(directions) = 1, 2, 3
48      % Random direction
49      locations = directions .* floor(100.0/sum(directions));
50      % Adjusts difficulty
51      % Increase in upward tendency if difficulty is 1
52      locations(1) = locations(1) * (1 - ((difficulty - 5.0)/14));
```

The change of codes I made are in the above picture. The only change is in the first if-statement. I rewrite line34-line45 to implement the key principle of BFS algorithm. The biggest difference between BFS and DFS algorithm is that BFS stands for breadth first searching and DFS stands for depth-first searching. Therefore, the enqueue and dequeue pattern differs greatly. BFS uses a queue structure which allows first in first out while DFS uses a stack structure which allows first in last out.

In this maze-generating situation, when implementing BFS algorithm, we need to add the futurePosition node, which is determined by a random choice, to the last position of queue. The algorithm keeps searching level-by-level and adds random future position nodes to the last position of the queue. When there is no children nodes to expand,

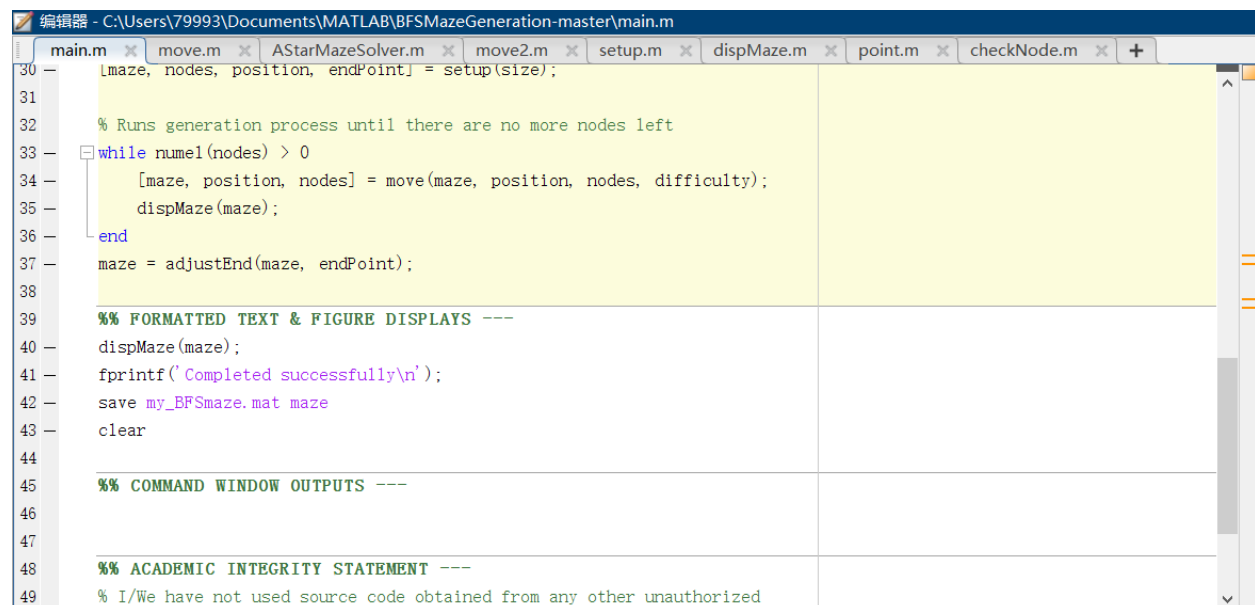
2019/2020 COMP1037 Coursework 1 – Search Techniques

which is the situation that if condition in line43 holds true, the node in the first position of the queue will be dequeued and used for searching another tree. The dequeue process is transformed into MatLab codes which are in line44-45 that position now represents the first node in queue and the node is dequeued after assigned to position. By doing so, the maze is now generated by BFS algorithm.



```
+1 | move.m | move1.m | AStarMazeSolver.m | move2.m | setup.m | dispMaze.m | point.m | checkNode.m | +
22 | % Assignment: MATLAB Individual Project
23 | % Author: Ryan Schwartz, schwar95@purdue.edu
24 | % Team ID: 001-07
25 | %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26 |
27 | %% INITIALIZATION ---
28 |
29 |
30 | %% CALCULATIONS ---
31 | if fPosition.row ~= pPosition.row || fPosition.col ~= pPosition.col
32 |     result = 1;
33 | else
34 |     result = 0;
35 | end
36 |
37 | %% FORMATTED TEXT & FIGURE DISPLAYS ---
38 |
39 |
40 | %% COMMAND WINDOW OUTPUTS ---
41 |
```

In the above codes, I changed the && in if condition to || because I think every point on path is a node and can take branches randomly/



```
编辑器 - C:\Users\79993\Documents\MATLAB\BFSMazeGeneration-master\main.m
| main.m | move.m | AStarMazeSolver.m | move2.m | setup.m | dispMaze.m | point.m | checkNode.m | +
30 | [maze, nodes, position, endPoint] = setup(size);
31 |
32 | % Runs generation process until there are no more nodes left
33 | while numel(nodes) > 0
34 |     [maze, position, nodes] = move(maze, position, nodes, difficulty);
35 |     dispMaze(maze);
36 | end
37 | maze = adjustEnd(maze, endPoint);
38 |
39 | %% FORMATTED TEXT & FIGURE DISPLAYS ---
40 | dispMaze(maze);
41 | fprintf('Completed successfully\n');
42 | save my_BFSmaze.mat maze
43 | clear
44 |
45 | %% COMMAND WINDOW OUTPUTS ---
46 |
47 |
48 | %% ACADEMIC INTEGRITY STATEMENT ---
49 | % I/We have not used source code obtained from any other unauthorized
```

I changed the file that save the maze to my_BFSmaze.mat

AStarMazeSolver:

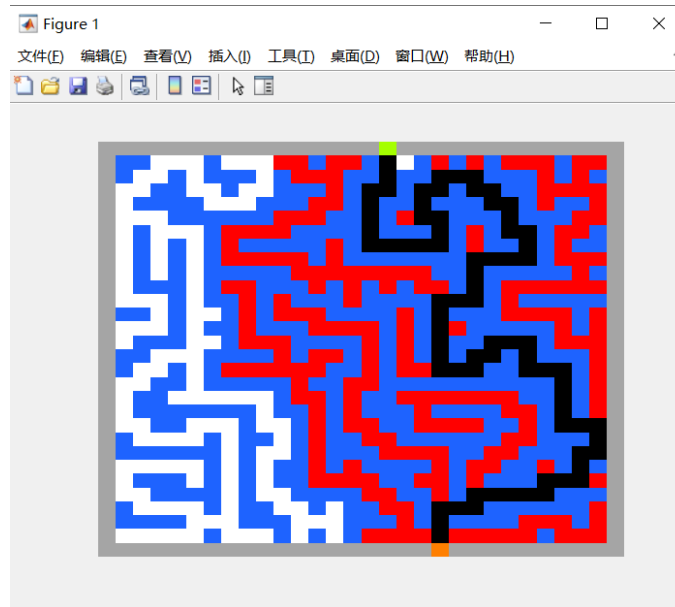
First of all, to modify the A* Algorithm code to the AStarSolver algorithm, we

need to look at the theoretical base of the A* Algorithm which is :

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

$f(n)$ is the sum of the $g(n)$ and $h(n)$, where n is the node to path, $g(n)$ is the cost from starting point to current node and $h(n)$ is a heuristic function that estimates the cheapest cost from current node to the goal. A* algorithm is often used in pathfinding problems and graph traversal problems. In this maze-pathfinding problem, as there is a function called “adjustEnd” to guarantee the maze is solvable, it must exist an optimal route to solve the maze. Therefore the A* algorithm is admissible.

(below is the result I get when calling “AStarMazeSolver(maze) in command window”)

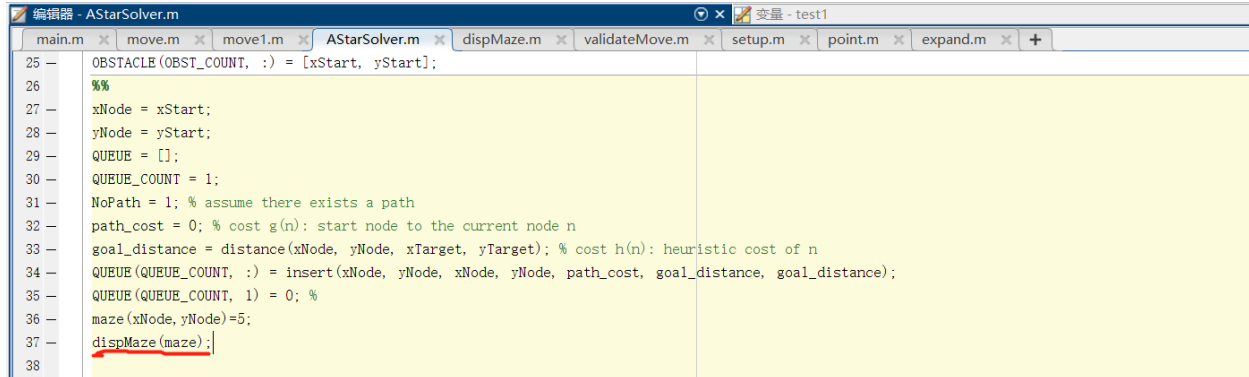


Changes on the given AStar code:

```
编辑器 - AStarSolver.m
main.m  move.m  move1.m  AStarSolver.m  dispMaze.m  validateMove.m  setup.m  point.m  expand.m  +
1  function [] = AStarSolver(maze)
2
3  maze_size= size(maze,1);
4  MAX_X = maze_size;
5  MAX_Y= maze_size;
6  OBSTACLE = [];
7  k = 1;
8  for i = 1 : MAX_X
9      for j = 1 : MAX_Y
10         if(maze(i, j) == 0)
11             OBSTACLE(k, 1) = i;
12             OBSTACLE(k, 2) = j;
13             k = k + 1;
14         elseif (maze(i,j)==3)
15             xStart=i-1;
16             yStart=j;
17         elseif (maze(i,j)==4)
18             xTarget=i+1;
19             yTarget=j;
20         end
21     end
end
```

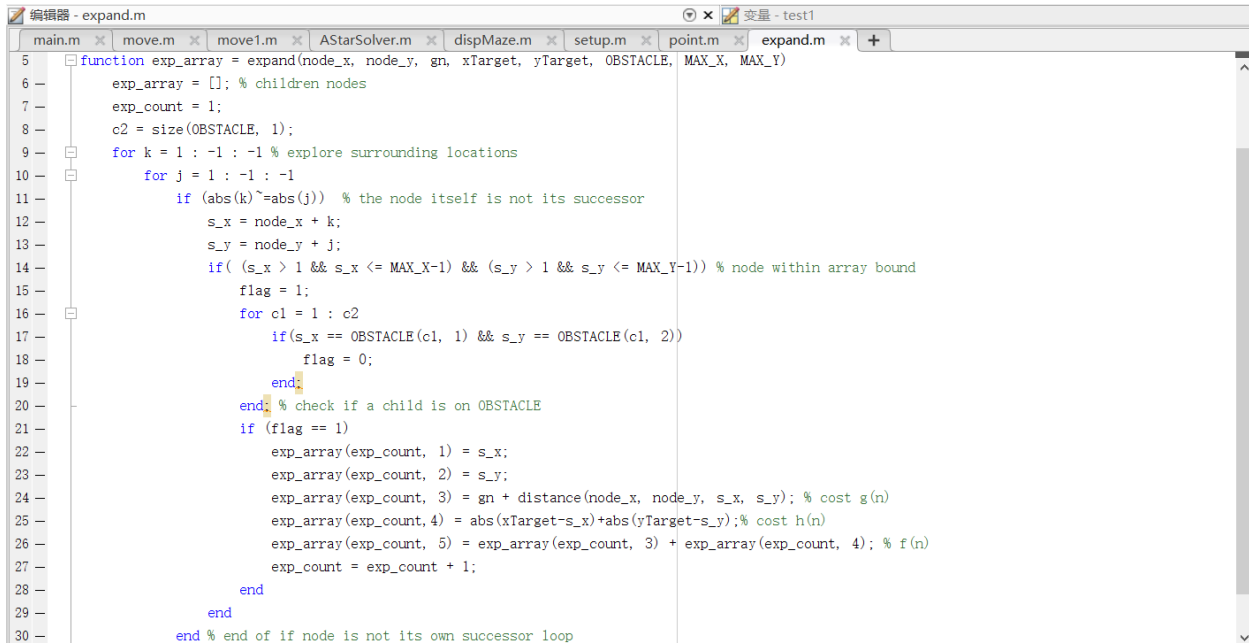
At first, I deleted “clf; problem();” two statements which are for a different pathfinding puzzle. Then in the nested for loop, I use the distinct values in the maze matrix to

identify and initialize queue OBSTACLE , starting point and target point.



```
25 — OBSTACLE(OBST_COUNT, :) = [xStart, yStart];
26 —
27 — %%
28 — xNode = xStart;
29 — yNode = yStart;
30 — QUEUE = [];
31 — QUEUE_COUNT = 1;
32 — NoPath = 1; % assume there exists a path
33 — path_cost = 0; % cost g(n): start node to the current node n
34 — goal_distance = distance(xNode, yNode, xTarget, yTarget); % cost h(n): heuristic cost of n
35 — QUEUE(QUEUE_COUNT, :) = insert(xNode, yNode, xNode, yNode, path_cost, goal_distance, goal_distance);
36 — QUEUE(QUEUE_COUNT, 1) = 0; %
37 — maze(xNode, yNode)=5;
38 — dispMaze(maze);
```

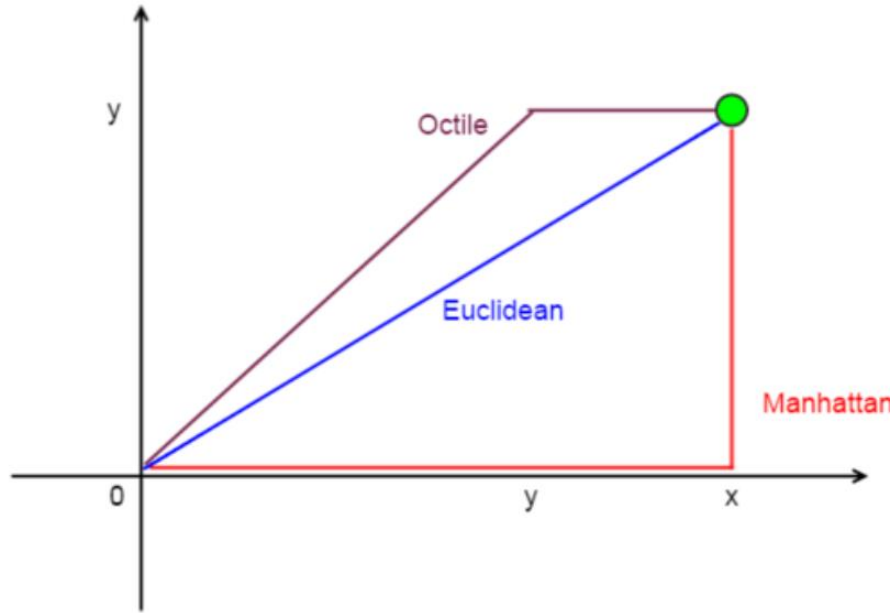
Then I add “maze(XNode,YNode)=5” to set the starting point to be red, corresponding to the index of the edited colormap. Also, I add “dispMaze(maze)” before the searching part begins to visualize the maze and paint the starting point red.



```
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 (abs(k)~=abs(j)) % the node itself is not its successor
12 —                 s_x = node_x + k;
13 —                 s_y = node_y + j;
14 —                 if (s_x > 1 && s_x <= MAX_X-1) && (s_y > 1 && s_y <= MAX_Y-1) % 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) = abs(xTarget-s_x)+abs(yTarget-s_y); % cost h(n)
26 —                         exp_array(exp_count, 5) = exp_array(exp_count, 3) + exp_array(exp_count, 4); % f(n)
27 —                         exp_count = exp_count + 1;
28 —                     end
29 —                 end
30 —             end % end of if node is not its own successor loop
```

In the above picture, I changed two if conditions in line 11 and line 14. For the condition on line 11, by using abs() function to guarantee that the expanded children nodes can't be the node itself and also can't in a diagonal relationship to the current node. For the if condition in line 14, I change the condition that check if the node is within array bound from set (0, MAX_X) to (1,MAX_X-1) because the nodes can't be on the borders.

In line 25, I changed the heuristic function from calculating straight distance between current node to goal to calculating the Manhattan distance. Because according to lecture slides, “if $h_1(n) \leq h_2(n)$, for all states n in the search space, we say h_2 dominates h_1 or heuristic h_2 is more informed than h_1 .” The below simply demonstrates the reason why Manhattan distance is the most suitable in this 2-dimension pathfinding problem since it has a higher value for heuristic function.



Therefore, implementing h_2 could result in less expanded nodes which will certainly enhance the efficiency of the algorithm.

```

67 % A*: find the node in QUEUE with the smallest f(n), returned by min_fn
68 index_min_node = min_fn(QUEUE, QUEUE_COUNT);
69 if (index_min_node ~= -1)
70     % set current node (xNode, yNode) to the node with minimum f(n)
71     xNode = QUEUE(index_min_node, 2);
72     yNode = QUEUE(index_min_node, 3);
73     path_cost = QUEUE(index_min_node, 6); % cost g(n)
74     % move the node to OBSTACLE
75     OBST_COUNT = OBST_COUNT + 1;
76     OBSTACLE(OBST_COUNT, 1) = xNode;
77     OBSTACLE(OBST_COUNT, 2) = yNode;
78     QUEUE(index_min_node, 1) = 0;
79     maze(xNode, yNode) = 5;
80     dispMaze(maze);
81 else
82     NoPath = 0; % there is no path!
83 end
84 end
    
```

In this picture, the index of the node with the smallest $f(n)$ in QUEUE is obtained by “min_fn()”, which is the index of node estimated to be the closest to the goal in the queue. After obtaining the position of this node, I add the value of the position to 5 and paint it on maze using dispMaze(maze).

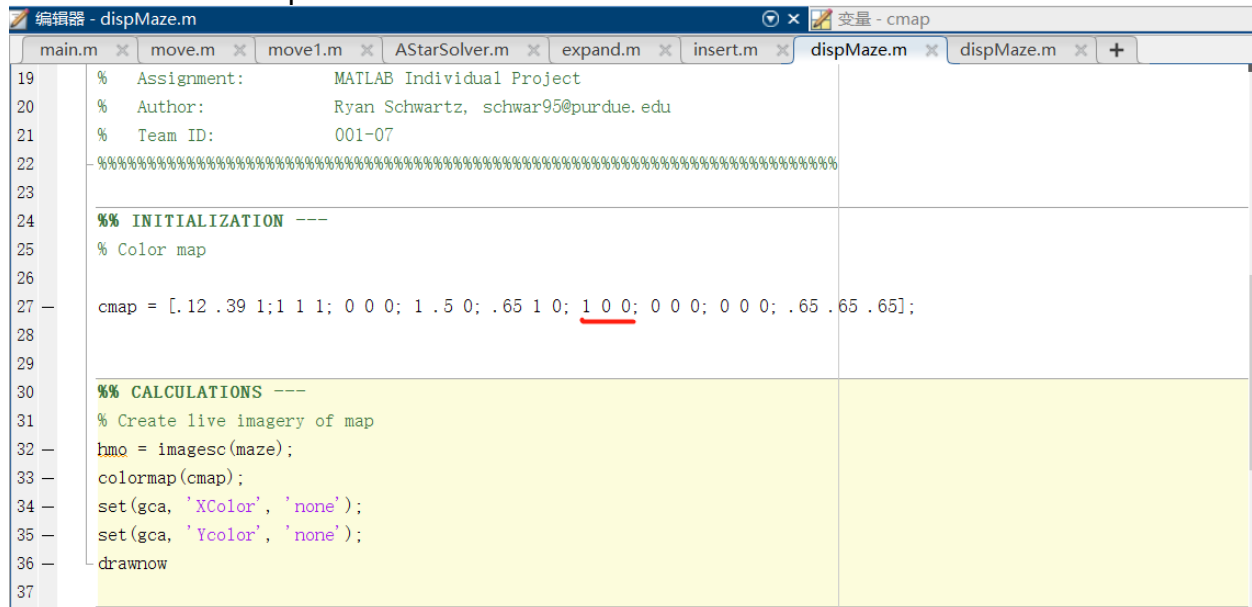
```

85 while ((xNode ~= xStart || yNode ~= yStart) && NoPath == 1)
86     maze(xNode, yNode) = 2;
87     dispMaze(maze);
88     for i = 1:QUEUE_COUNT
89         if (QUEUE(i, 2) == xNode && (QUEUE(i, 3) == yNode))
90             xNode = QUEUE(i, 4);
91             yNode = QUEUE(i, 5);
92         end
93     end
94 end
95     maze(xNode, yNode) = 2;
96     dispMaze(maze);
97 end
    
```

This purpose of this part is to paint the optimal route black. The while loop keeps

2019/2020 COMP1037 Coursework 1 – Search Techniques

tracking back the parent node of current node by using the data stored in QUEUE and setting the value of parent node position in maze as two which is the index of black color in the colormap.



```
19 % Assignment:      MATLAB Individual Project
20 % Author:         Ryan Schwartz, schwar95@purdue.edu
21 % Team ID:        001-07
22 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23
24 %% INITIALIZATION ---
25 % Color map
26
27 cmap = [.12 .39 1; 1 1 1; 0 0 0; 1 .5 0; .65 1 0; 1 0 0; 0 0 0; 0 0 0; .65 .65 .65];
28
29
30 %% CALCULATIONS ---
31 % Create live imagery of map
32 hmq = imagesc(maze);
33 colormap(cmap);
34 set(gca, 'XColor', 'none');
35 set(gca, 'Ycolor', 'none');
36 drawnow
37
```

The changes I made in dispMaze function is to change the cmap variable, set the 6th line to be (1 0 0) which is red in RGB color.