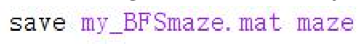
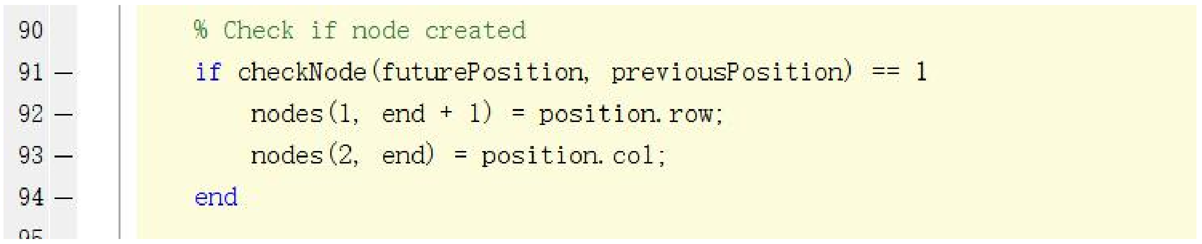
## Compare DFSMazeGeneration with my BFSMazeGeneration

I change two part of the previous code.
First, I change the name “my_DFSmaze.mat” to “my_BFSmaze.mat” in “main.m” line 41.



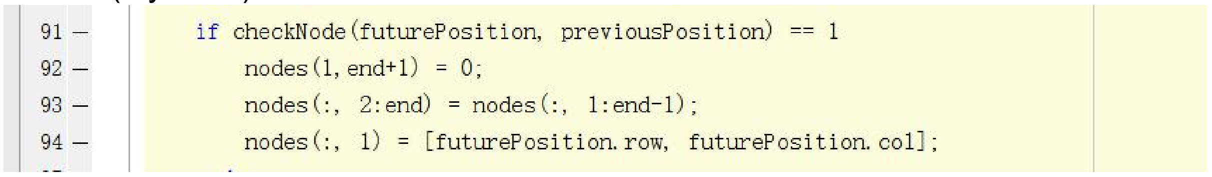
```
save my_BFSmaze.mat maze
```

Second, I change line 92, 93 in the file “move.mat”.
In DFS (previous code):



```
90        % Check if node created
91 -      if checkNode(futurePosition, previousPosition) == 1
92 -          nodes(1, end + 1) = position.row;
93 -          nodes(2, end) = position.col;
94 -      end
```

In BFS (my code):



```
91 -      if checkNode(futurePosition, previousPosition) == 1
92 -          nodes(1,end+1) = 0;
93 -          nodes(:, 2:end) = nodes(:, 1:end-1);
94 -          nodes(:, 1) = [futurePosition.row, futurePosition.col];
```

## Why I modify them?

The first modify is obviously because the maze name is different

The second modify because the different between BFS and DFS is that **in BFS, oldest unexpanded vertices are explored at first, and DFS will first explore the vertices which along the edge.** So we can summarize that there are two way to implements change from DFS to BFS:

1. In DFS, we put nodes in the last of queue and expand the last node of queue each time until there is no position to expand, then we remove it and access the last node of the new queue. In BFS, we can only change that when we remove the nodes that no position to expand, instead, access the first node of the new queue. However, if there still no position to expand we need to remove the first one and need to add a judge statement to avoid only one nodes in the queue which may case ‘Index out of bounds’.
2. Also, we can choose to add new nodes in the first of queue and not modify other place which mean when no position to expand we can still use the way to explore the last node of queue as DFS and remove from last.

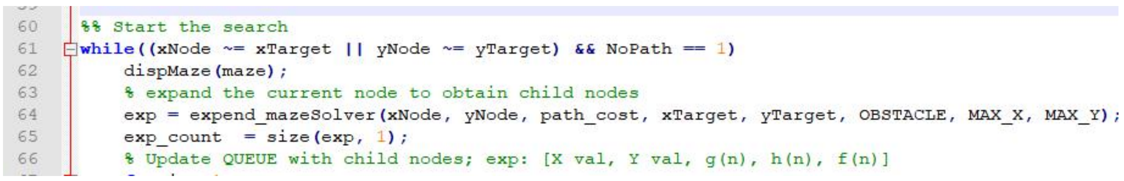
I choose the second way to implement change from DFS to BFS because first, it only need to change two line of previous code can minimize the modify, also, it do not need to add a statement to judge if it is only one nodes in the queue.

## Modify in Astar

### 1, Add “dispMaze.m” file in the folder

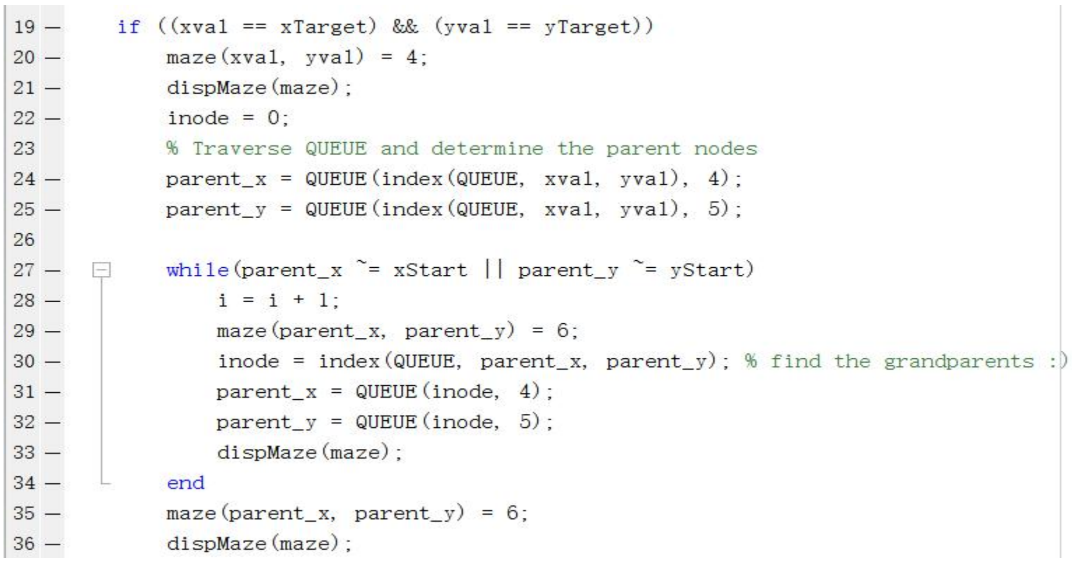
It is used to draw the maze, and I add it at:

Line 62 in the “AStarMazeSolver.m” to update path which is choose to expand(be processed)



```
60  %% Start the search
61  while((xNode ~= xTarget || yNode ~= yTarget) && NoPath == 1)
62      dispMaze(maze);
63      % expand the current node to obtain child nodes
64      exp = expend_mazeSolver(xNode, yNode, path_cost, xTarget, yTarget, OBSTACLE, MAX_X, MAX_Y);
65      exp_count  = size(exp, 1);
66      % Update QUEUE with child nodes; exp: [X val, Y val, g(n), h(n), f(n)]
```

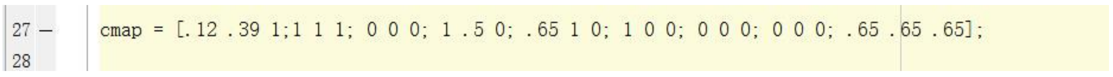
Line 21, line 33 and line 36 in the “result.m” which is use to update the optimal path.



```
19 —    if ((xval == xTarget) && (yval == yTarget))
20 —        maze(xval, yval) = 4;
21 —        dispMaze(maze);
22 —        inode = 0;
23          % Traverse QUEUE and determine the parent nodes
24 —        parent_x = QUEUE(index(QUEUE, xval, yval), 4);
25 —        parent_y = QUEUE(index(QUEUE, xval, yval), 5);
26
27 —        while(parent_x ~= xStart || parent_y ~= yStart)
28 —            i = i + 1;
29 —            maze(parent_x, parent_y) = 6;
30 —            inode = index(QUEUE, parent_x, parent_y); % find the grandparents :)
31 —            parent_x = QUEUE(inode, 4);
32 —            parent_y = QUEUE(inode, 5);
33 —            dispMaze(maze);
34 —        end
35 —        maze(parent_x, parent_y) = 6;
36 —        dispMaze(maze);
```

### 2. “dispMaze.m”

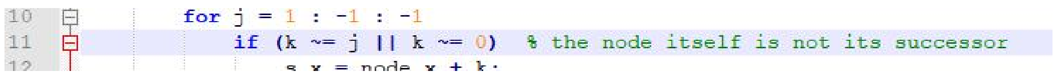
Based on the “dispMaze.m” in the BFS and add two color red [1 0 0] and black[0 0 0] to represent the processed vertices(value == 5) and optimal path(value == 6)



```
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
```

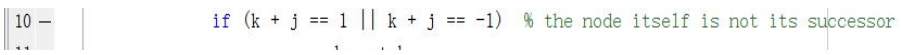
### 3. expend.m -> expend_mazeSolver.m

I modify the judgement in line 11 (in expend.m) from



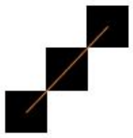
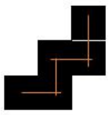
```
10      for j = 1 : -1 : -1
11          if (k ~= j || k ~= 0)  % the node itself is not its successor
```

to



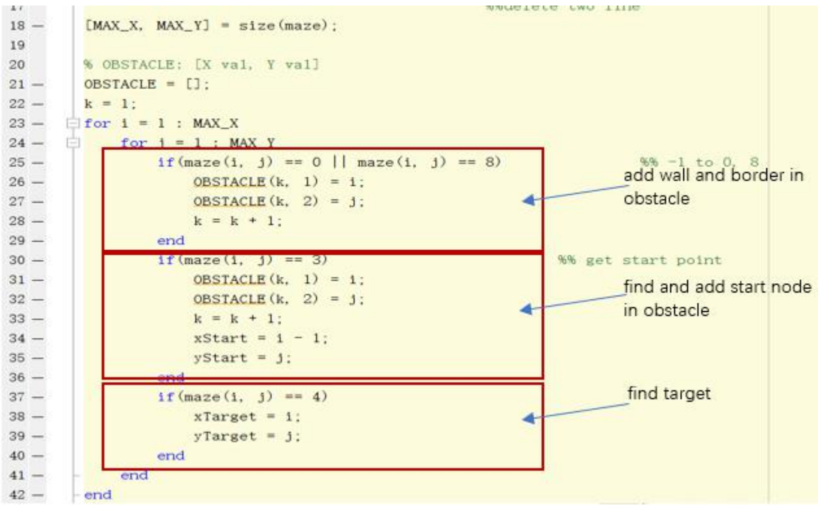
```
10 —            if (k + j == 1 || k + j == -1)  % the node itself is not its successor
```

Make sure it will not move slash and save it in “expend_mazeSolver.m”

From  to 

**4. A_Star.m -> AStarMazeSolver.m**

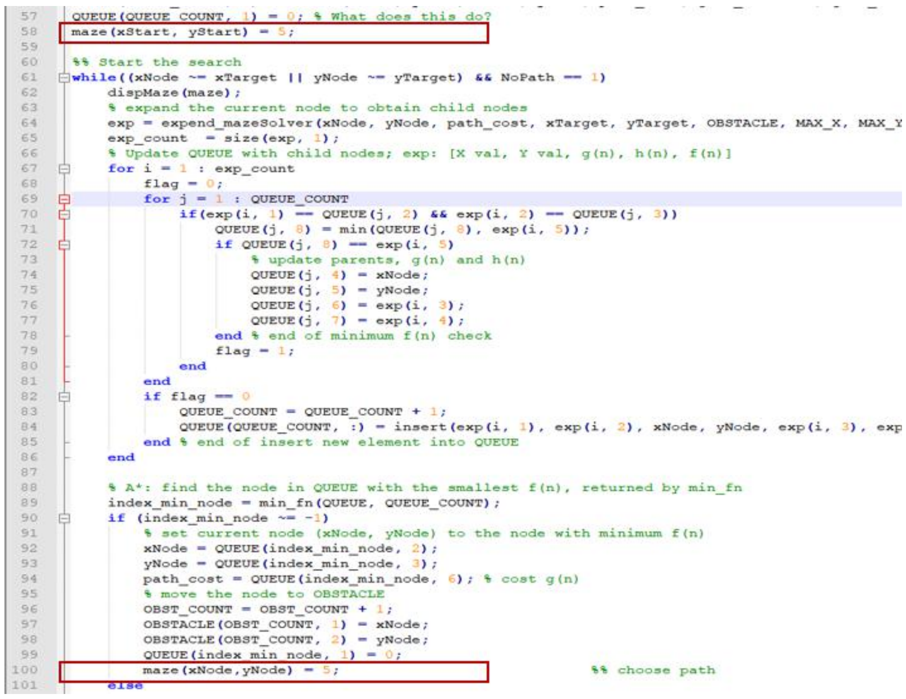
I remove the problem() and use loop to get the start point, target point. And change the if statement for add obstacle to list. From line 23 to line 42.



```
18   [MAX_X, MAX_Y] = size(maze);
19
20   % OBSTACLE: [X val, Y val]
21   OBSTACLE = [];
22   k = 1;
23   for i = 1 : MAX_X
24       for j = 1 : MAX_Y
25           if(maze(i, j) == 0 || maze(i, j) == 8)          %% -1 to 0, 8
26               OBSTACLE(k, 1) = i;
27               OBSTACLE(k, 2) = j;
28               k = k + 1;
29           end
30           if(maze(i, j) == 3)                            %% get start point
31               OBSTACLE(k, 1) = i;
32               OBSTACLE(k, 2) = j;
33               k = k + 1;
34               xStart = i - 1;
35               yStart = j;
36           end
37           if(maze(i, j) == 4)
38               xTarget = i;
39               yTarget = j;
40           end
41       end
42   end
```

add wall and border in obstacle
find and add start node in obstacle
find target

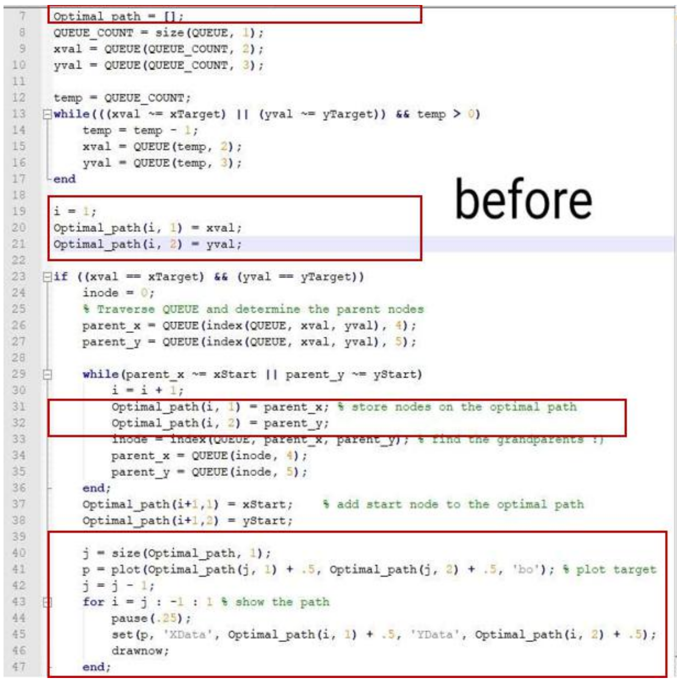
In line 58, I first set the value of start point to 5 because this point must be choose to expansive at first. Then, in line 100, I also do the same thing---- set the value of position to 5 because if next loop is exist, it will be expanded and become the route A* processed, so set value of this node to 5. If next loop not exist, represent this node is the target node and not suit to the definition of processed(children be explored and tested), I will change it back to 4 before draw it, so it will not be red.



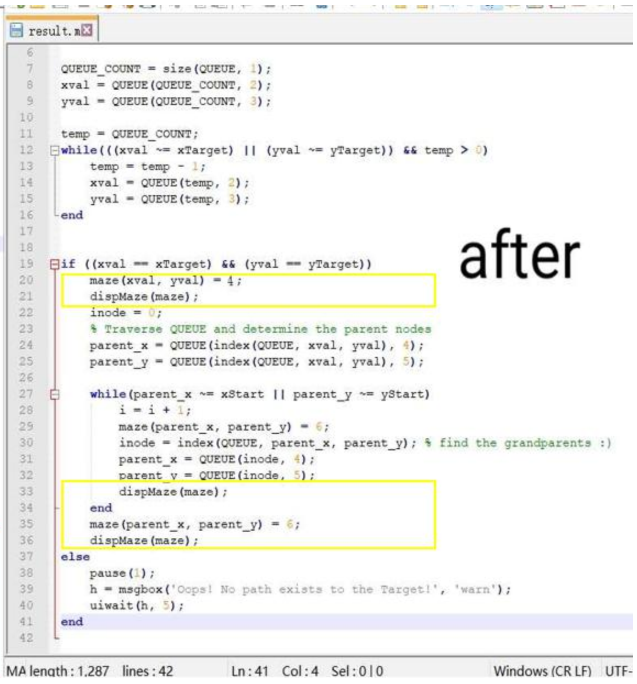
```
57   QUEUE(QUEUE_COUNT, 1) = 0; % What does this do?
58   maze(xStart, yStart) = 5;
59
60   %% Start the search
61   while((xNode ~= xTarget || yNode ~= yTarget) && NoPath == 1)
62       dispMaze(maze);
63       % expand the current node to obtain child nodes
64       exp = expend_mazeSolver(xNode, yNode, path_cost, xTarget, yTarget, OBSTACLE, MAX_X, MAX_Y
65       exp_count  = size(exp, 1);
66       % Update QUEUE with child nodes; exp: [X val, Y val, g(n), h(n), f(n)]
67       for i = 1 : exp_count
68           flag = 0;
69           for j = 1 : QUEUE_COUNT
70               if(exp(i, 1) == QUEUE(j, 2) && exp(i, 2) == QUEUE(j, 3))
71                   QUEUE(j, 8) = min(QUEUE(j, 8), exp(i, 5));
72                   if QUEUE(j, 8) == exp(i, 5)
73                       % update parents, g(n) and h(n)
74                       QUEUE(j, 4) = xNode;
75                       QUEUE(j, 5) = yNode;
76                       QUEUE(j, 6) = exp(i, 3);
77                       QUEUE(j, 7) = exp(i, 4);
78                   end % end of minimum f(n) check
79                   flag = 1;
80               end
81           end
82           if flag == 0
83               QUEUE_COUNT = QUEUE_COUNT + 1;
84               QUEUE(QUEUE_COUNT, :) = insert(exp(i, 1), exp(i, 2), xNode, yNode, exp(i, 3), exp
85           end % end of insert new element into QUEUE
86       end
87
88       % A*: find the node in QUEUE with the smallest f(n), returned by min_fn
89       index_min_node = min_fn(QUEUE, QUEUE_COUNT);
90       if (index_min_node ~= -1)
91           % set current node (xNode, yNode) to the node with minimum f(n)
92           xNode = QUEUE(index_min_node, 2);
93           yNode = QUEUE(index_min_node, 3);
94           path_cost = QUEUE(index_min_node, 6); % cost g(n)
95           % move the node to OBSTACLE
96           OBST_COUNT = OBST_COUNT + 1;
97           OBSTACLE(OBST_COUNT, 1) = xNode;
98           OBSTACLE(OBST_COUNT, 2) = yNode;
99           QUEUE(index_min_node, 1) = 0;
100          maze(xNode,yNode) = 5;                     %% choose path
101      else
```

5. **modify in "result.m"**
   In the whole file, I remove "optimal_path[]" and draw the maze each time instead when find its  parents node to make sure the black optimal path will generate from target to start like demo.

delete part      add part


before

```
Optimal_path = [];
QUEUE_COUNT = size(QUEUE, 1);
xval = QUEUE(QUEUE_COUNT, 2);
yval = QUEUE(QUEUE_COUNT, 3);

temp = QUEUE_COUNT;
while(((xval ~= xTarget) || (yval ~= yTarget)) && temp > 0)
    temp = temp - 1;
    xval = QUEUE(temp, 2);
    yval = QUEUE(temp, 3);
end

i = 1;
Optimal_path(i, 1) = xval;
Optimal_path(i, 2) = yval;

if ((xval == xTarget) && (yval == yTarget))
    inode = 0;
    % Traverse QUEUE and determine the parent nodes
    parent_x = QUEUE(index(QUEUE, xval, yval), 4);
    parent_y = QUEUE(index(QUEUE, xval, yval), 5);

    while(parent_x ~= xStart || parent_y ~= yStart)
        i = i + 1;
        Optimal_path(i, 1) = parent_x; % store nodes on the optimal path
        Optimal_path(i, 2) = parent_y;
        inode = index(QUEUE, parent_x, parent_y); % find the grandparents :)
        parent_x = QUEUE(inode, 4);
        parent_y = QUEUE(inode, 5);
    end;
    Optimal_path(i+1,1) = xStart;    % add start node to the optimal path
    Optimal_path(i+1,2) = yStart;

    j = size(Optimal_path, 1);
    p = plot(Optimal_path(j, 1) + .5, Optimal_path(j, 2) + .5, 'bo'); % plot target
    j = j - 1;
    for i = j : -1 : 1 % show the path
        pause(.25);
        set(p, 'XData', Optimal_path(i, 1) + .5, 'YData', Optimal_path(i, 2) + .5);
        drawnow;
    end;
```


after

```
result.m

QUEUE_COUNT = size(QUEUE, 1);
xval = QUEUE(QUEUE_COUNT, 2);
yval = QUEUE(QUEUE_COUNT, 3);

temp = QUEUE_COUNT;
while(((xval ~= xTarget) || (yval ~= yTarget)) && temp > 0)
    temp = temp - 1;
    xval = QUEUE(temp, 2);
    yval = QUEUE(temp, 3);
end


if ((xval == xTarget) && (yval == yTarget))
    maze(xval, yval) = 4;
    dispMaze(maze);
    inode = 0;
    % Traverse QUEUE and determine the parent nodes
    parent_x = QUEUE(index(QUEUE, xval, yval), 4);
    parent_y = QUEUE(index(QUEUE, xval, yval), 5);

    while(parent_x ~= xStart || parent_y ~= yStart)
        i = i + 1;
        maze(parent_x, parent_y) = 6;
        inode = index(QUEUE, parent_x, parent_y); % find the grandparents :)
        parent_x = QUEUE(inode, 4);
        parent_y = QUEUE(inode, 5);
        dispMaze(maze);
    end
    maze(parent_x, parent_y) = 6;
    dispMaze(maze);
else
    pause(1);
    h = msgbox('Oops! No path exists to the Target!', 'warn');
    uiwait(h, 5);
end

MA length : 1,287   lines : 42        Ln : 41   Col : 4   Sel : 0 | 0        Windows (CR LF)   UTF-
```