ECE448 MP1 report 4 credits students

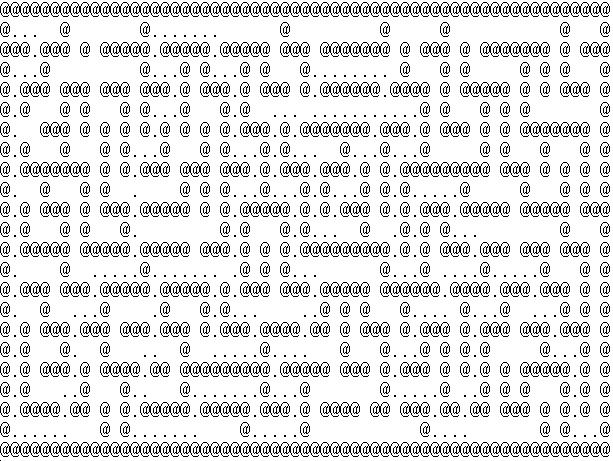
Yu\_Fan Fang,Chieh Hsu,Yu\_ming Huang

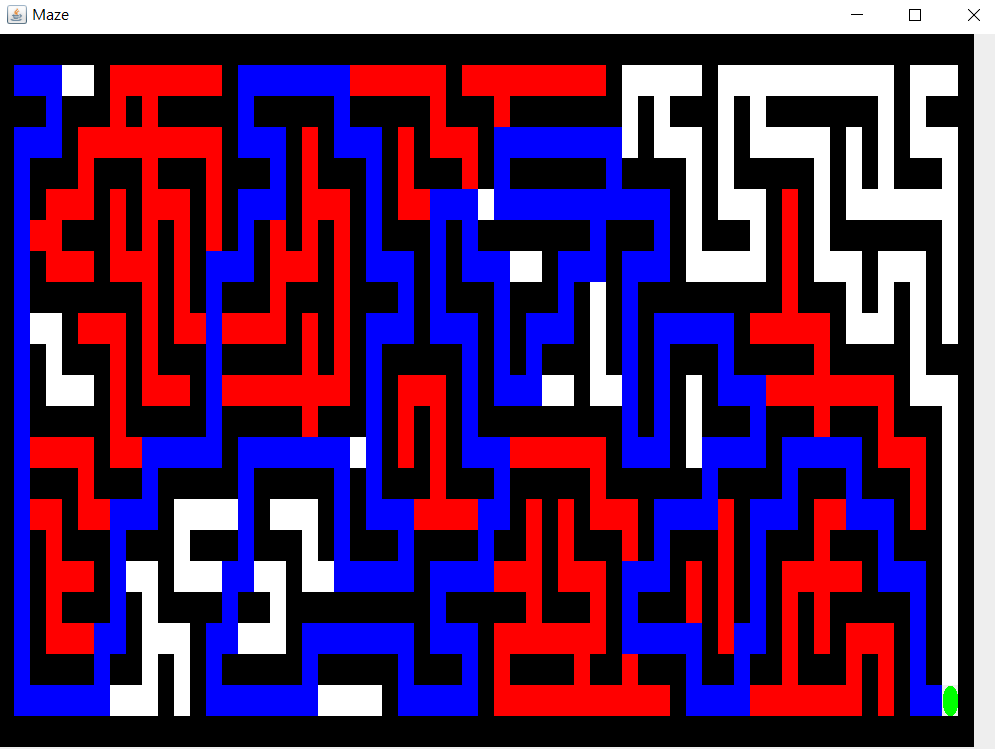
**1-1:Single point**

**One:mediumMaze**

1. **depth first search**

(1)





(2)

Total past cost:283

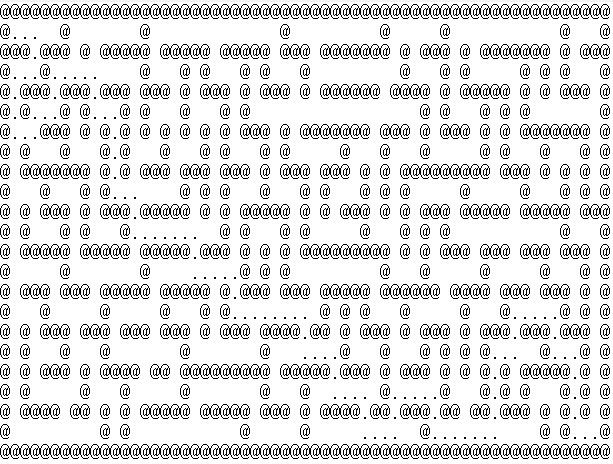
(3)

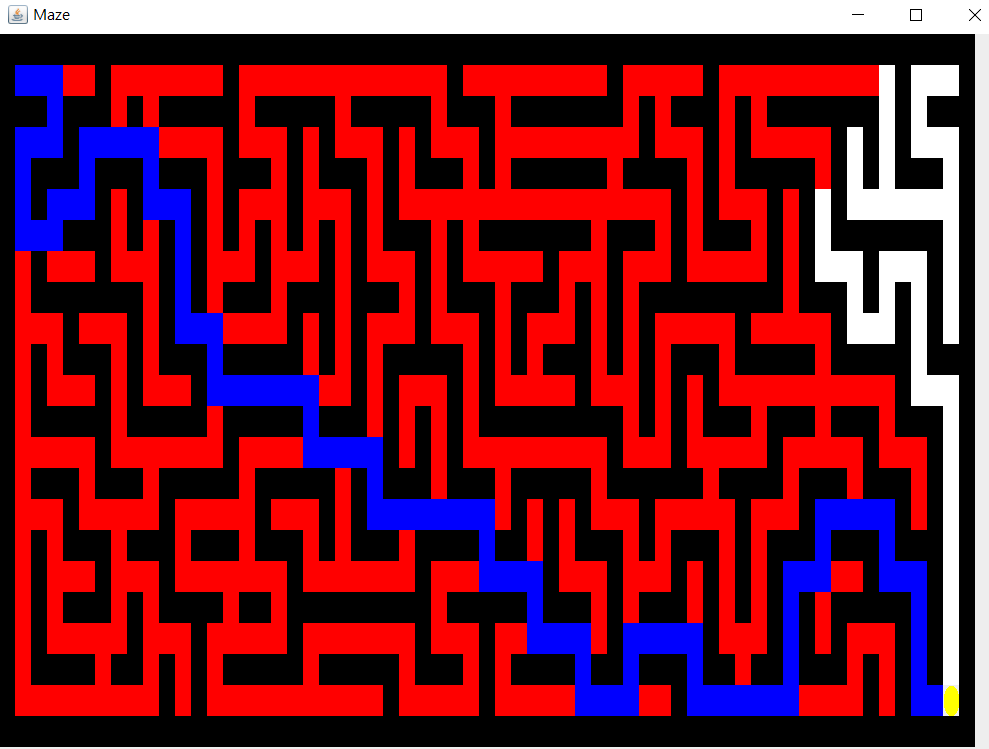
Node expanded:528

------------------------------------------------------------------------------------------------

1. **breadthFirstSearch**

(1)





(2)

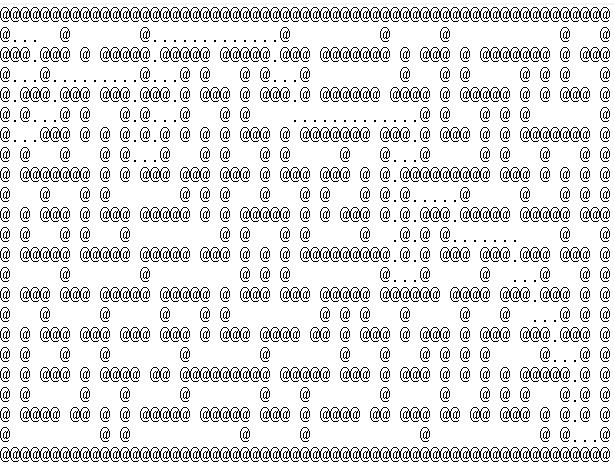
Total path cost:105

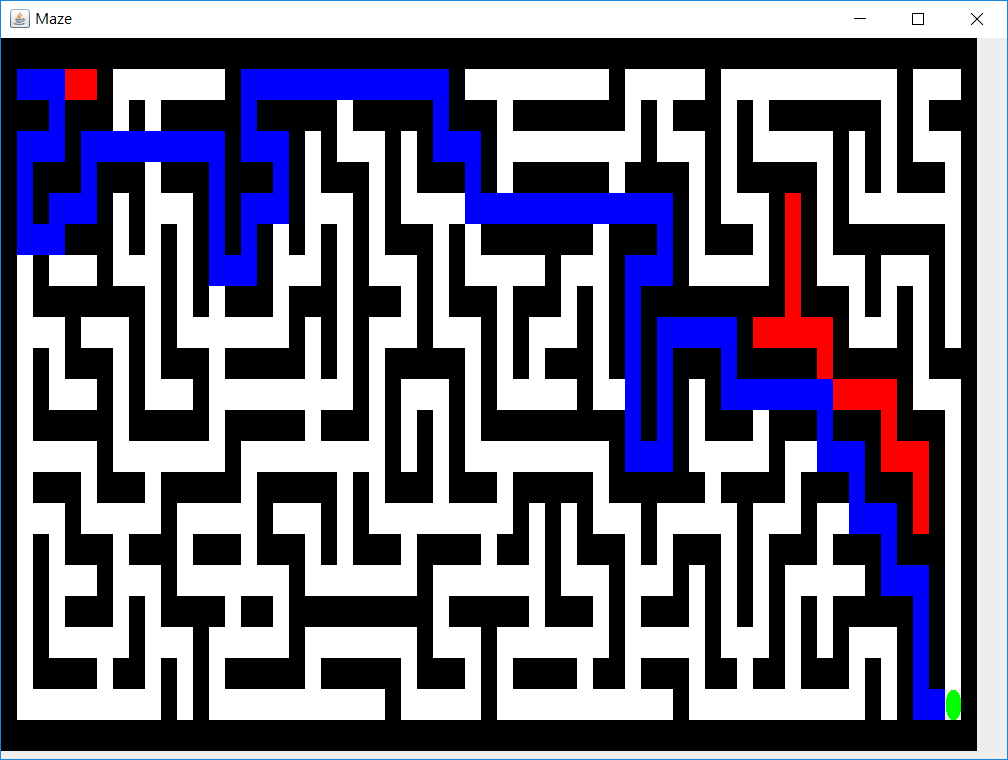
(3)

Total expanded node:629

1. **best first search**

(1)





(2)

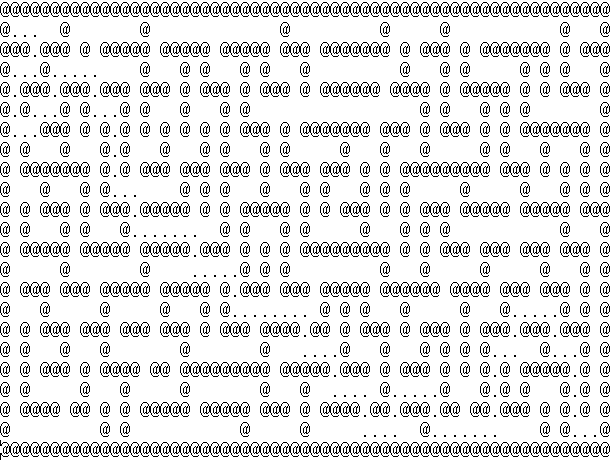
Total path cost:117

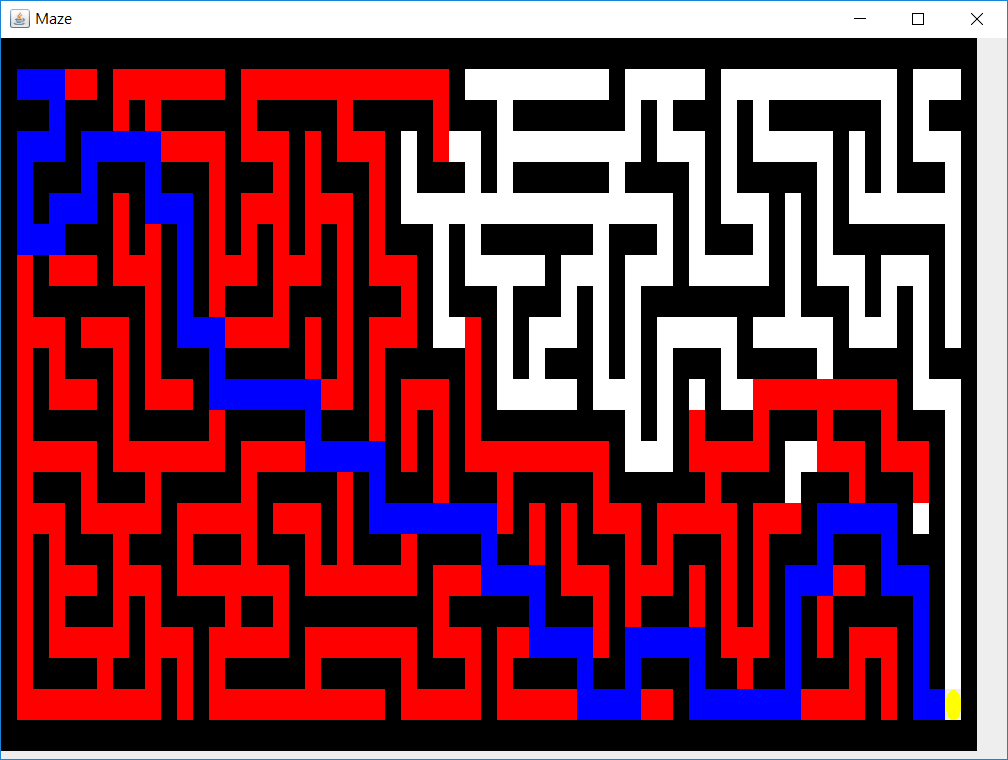
(3)

Total expanded node:138

1. **A\* search**

(1)





(2)

Total path cost:105

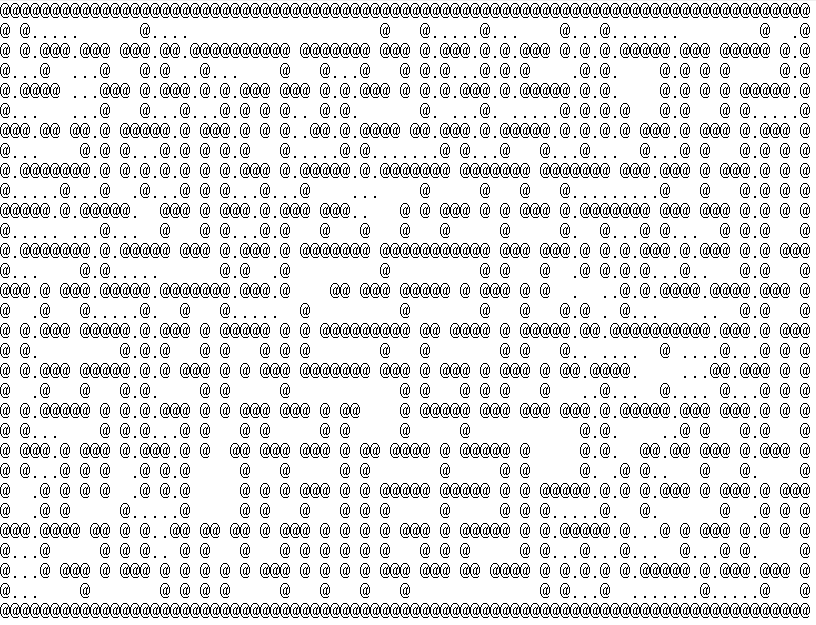
(3)

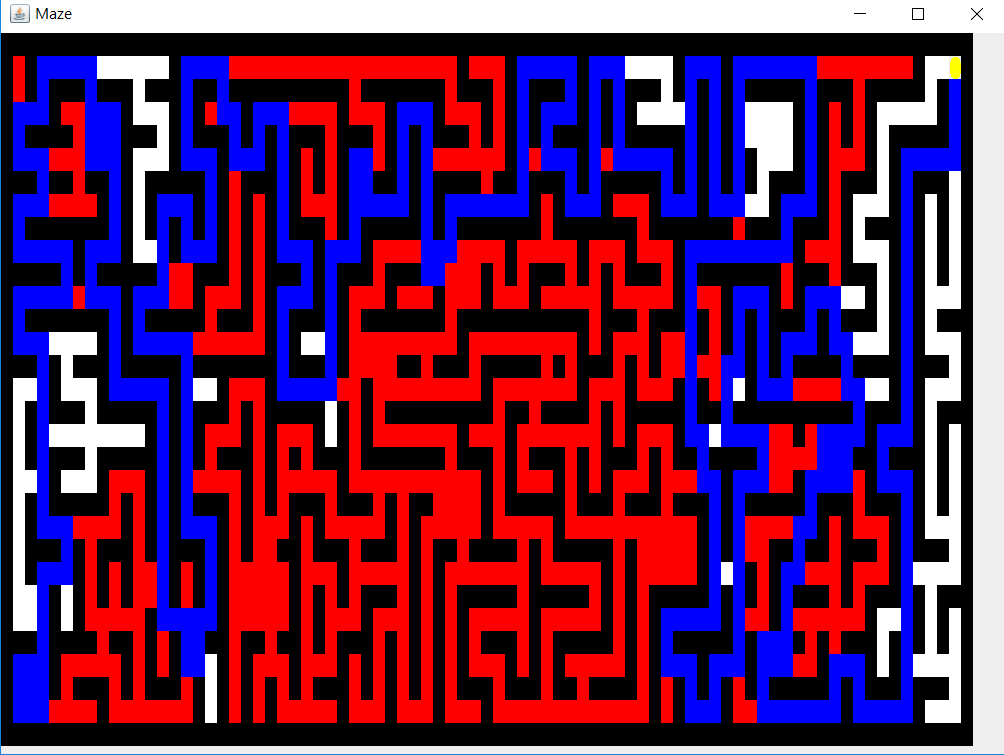
Total expanded node:473

**Two:bigMaze**

1. **depth first search**

(1)





(2)

Total past cost:469

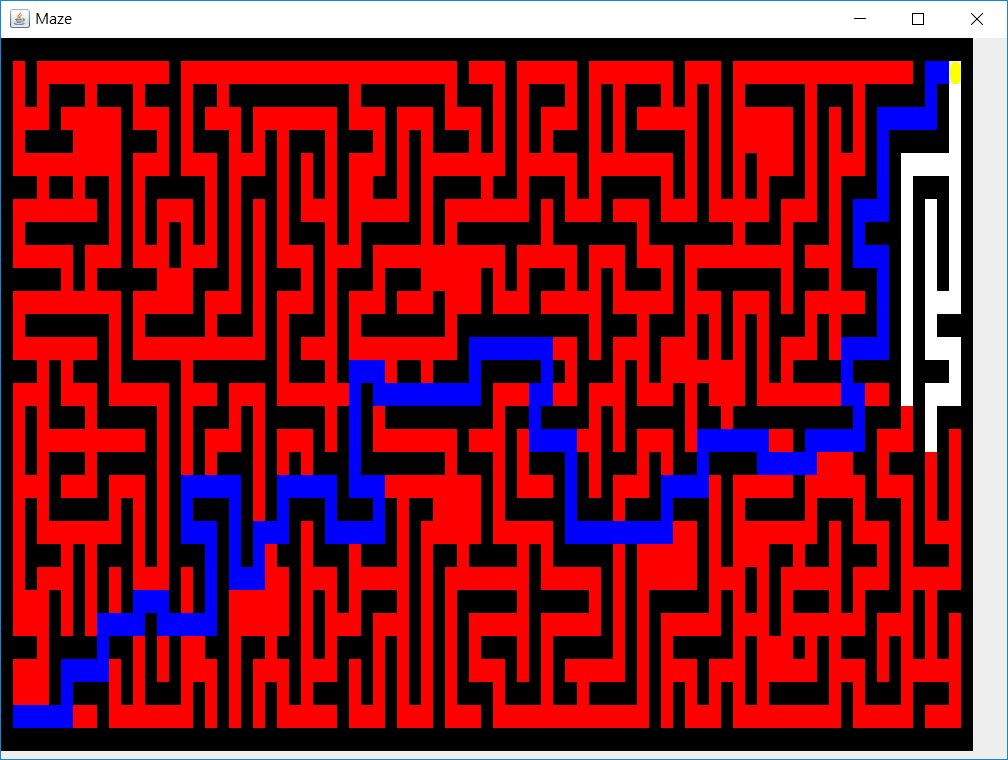
(3)

Total expanded node:1124

1. **breadth first search**

(1)





(2)

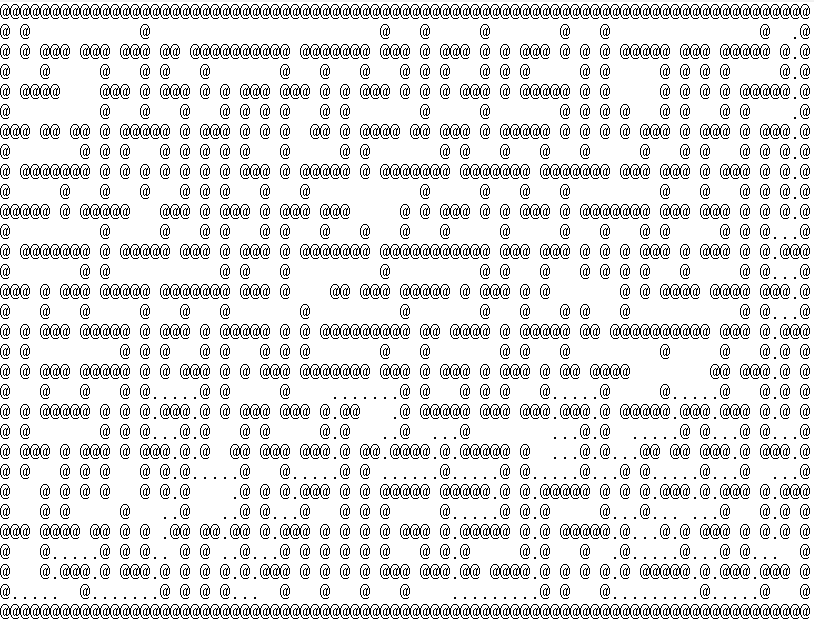
Total path cost:157

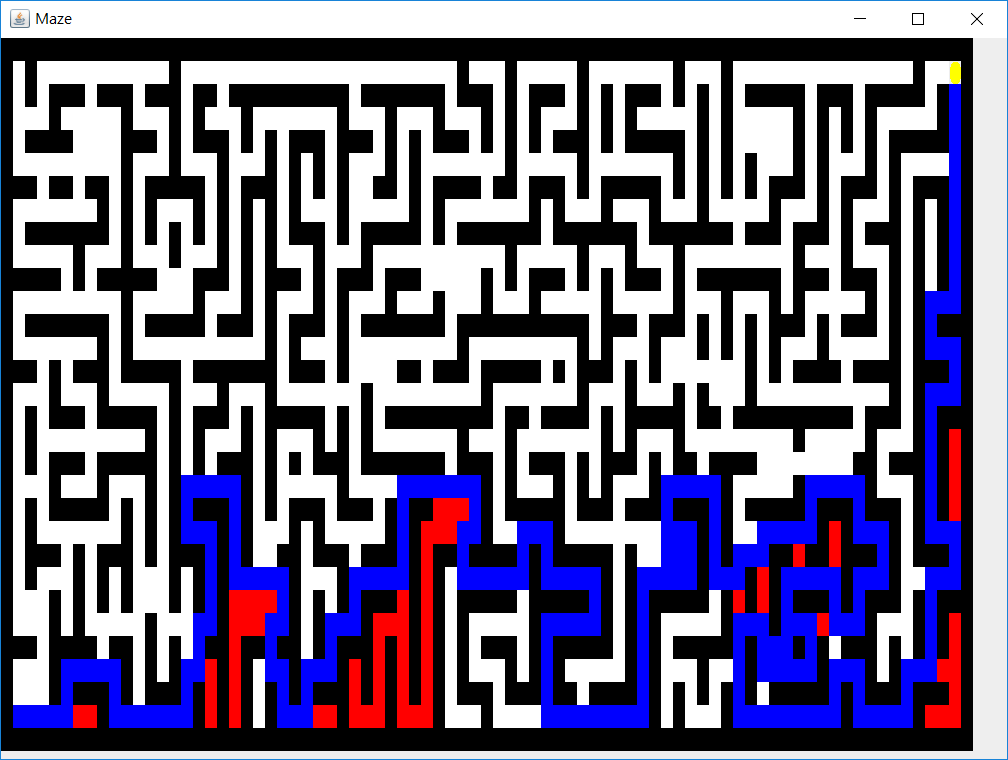
(3)

Total expanded node:1258

1. **best first search**

(1)





(2)

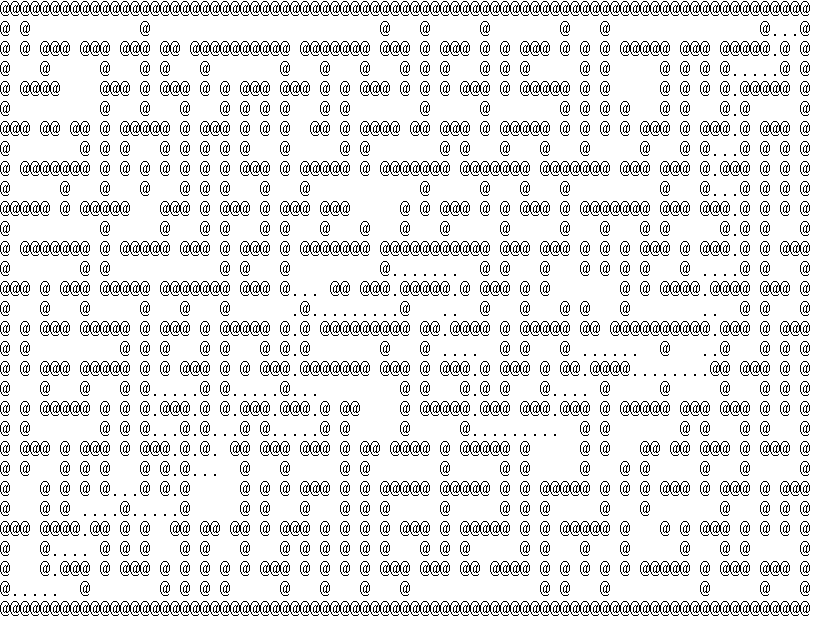
Total path cost:255

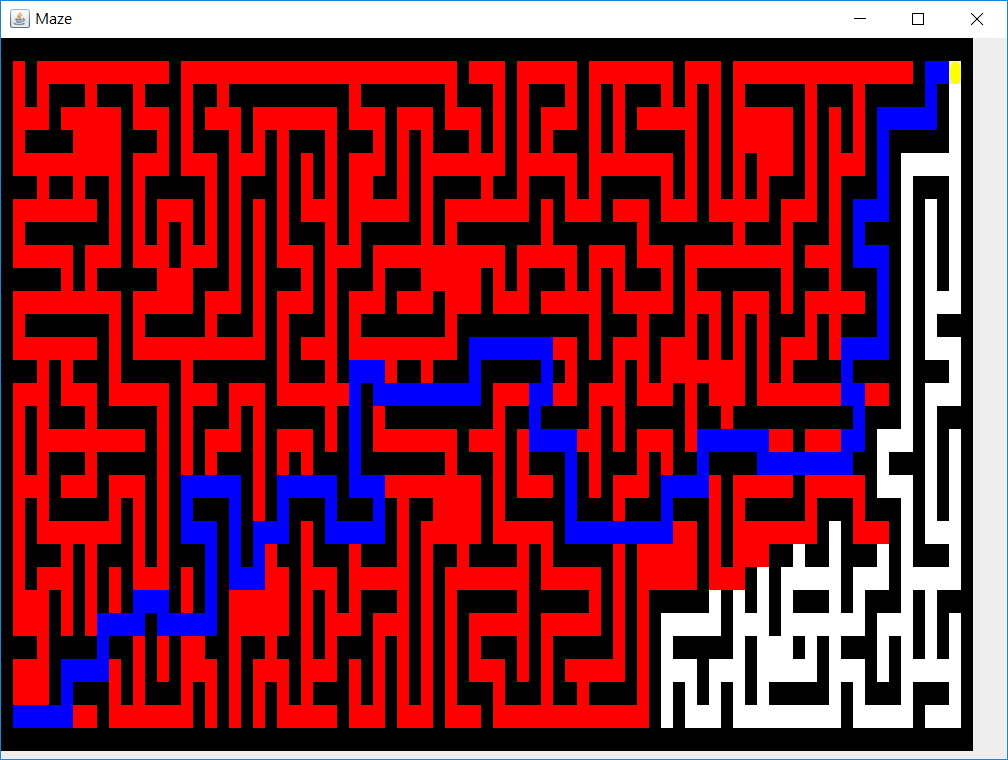
(3)

Total expanded node:322

**d.A\* search**

(1)





(2)

Total path cost:157

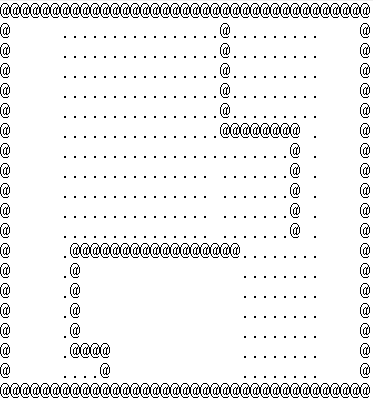
(3)

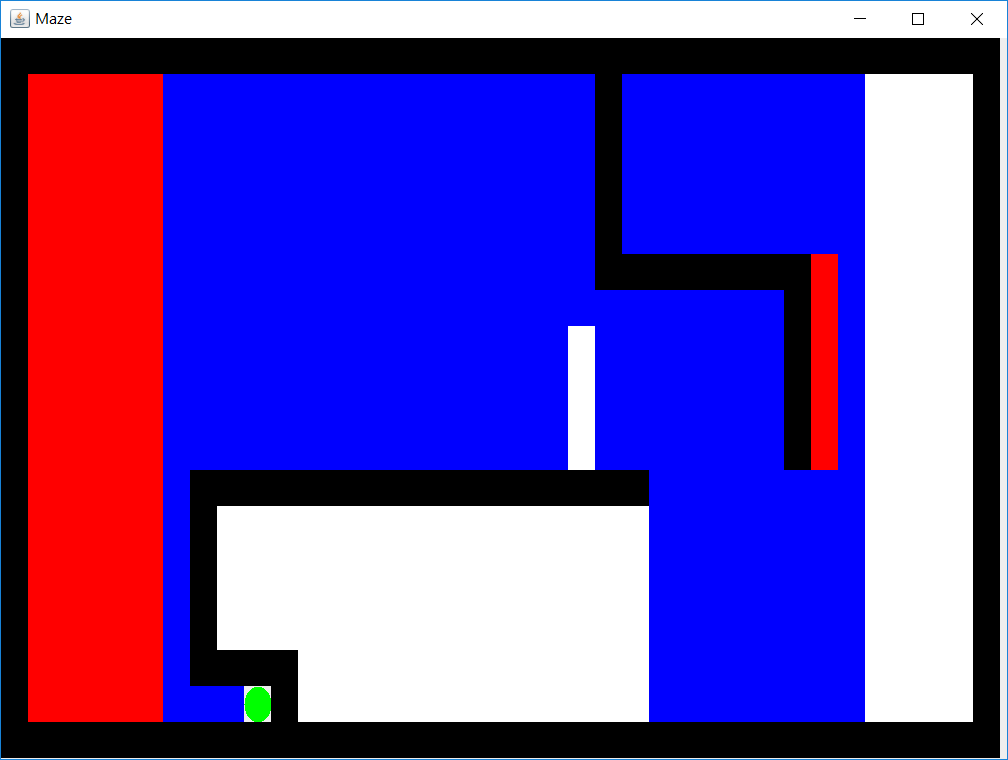
Total expanded node:1129

**Three:openMaze**

1. **depth first search**

(1)





(2)

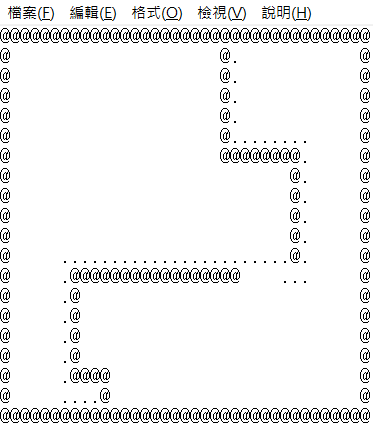
Total past cost:324

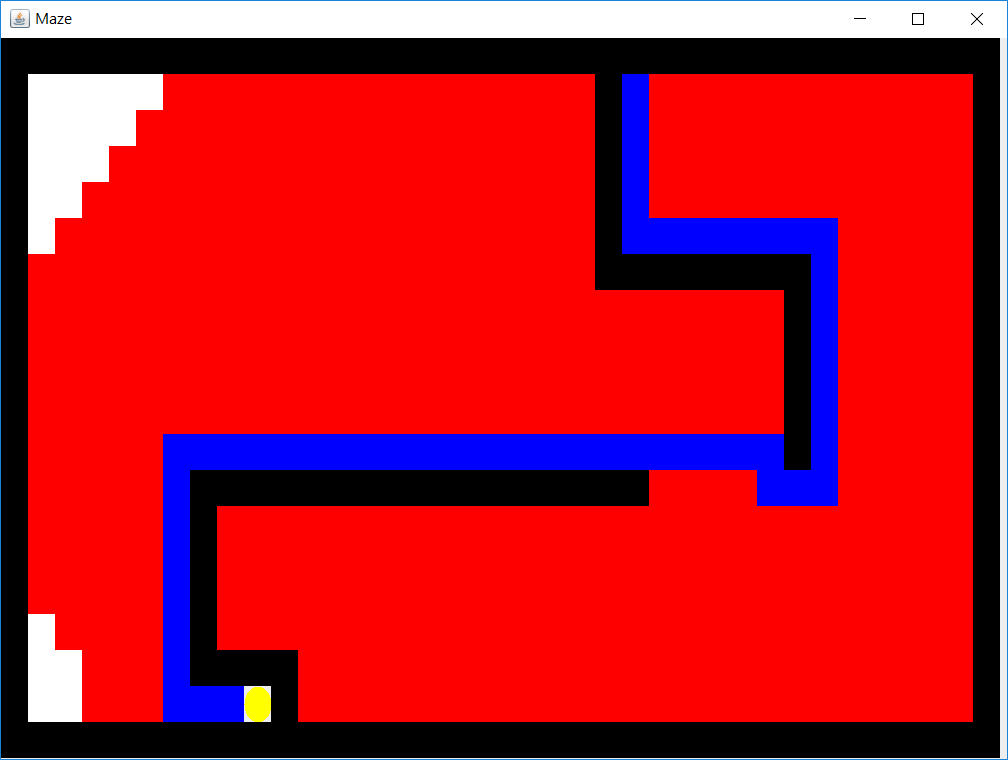
(3)

Total expanded node:419

1. **breadFirstSearch**

(1)





(2)

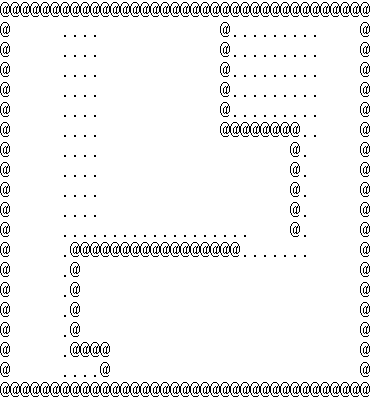
Total path cost:54

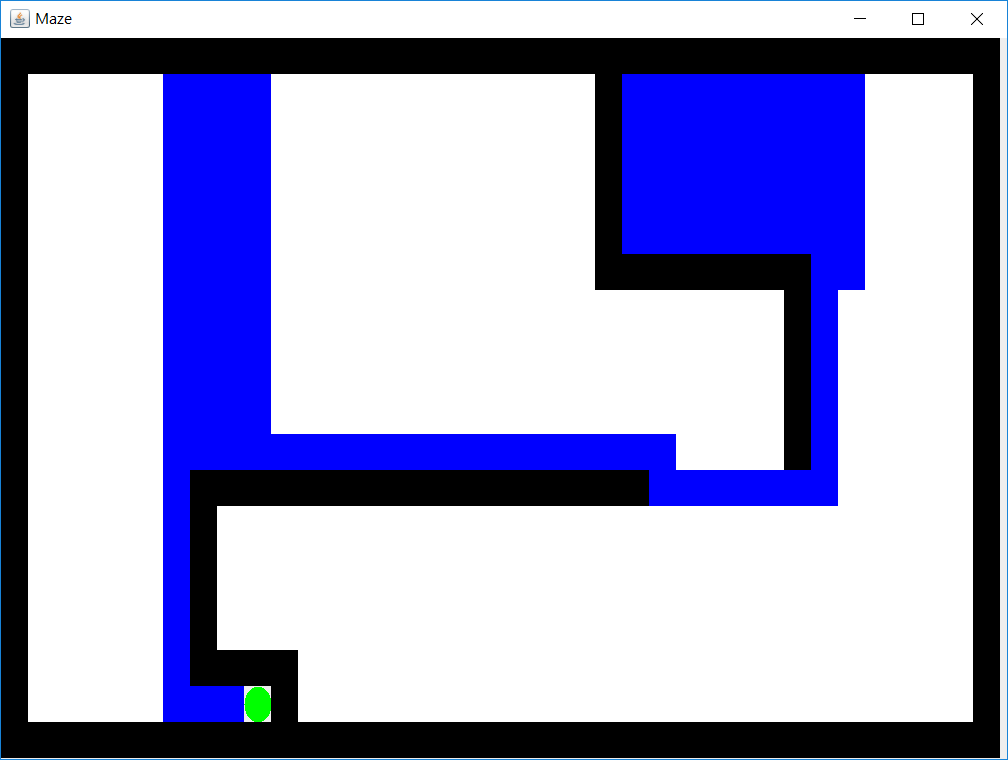
(3)

Total expanded node:565

1. **best first search**

(1)





(2)

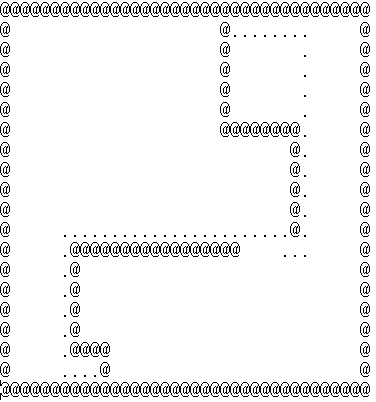
Total path cost:128

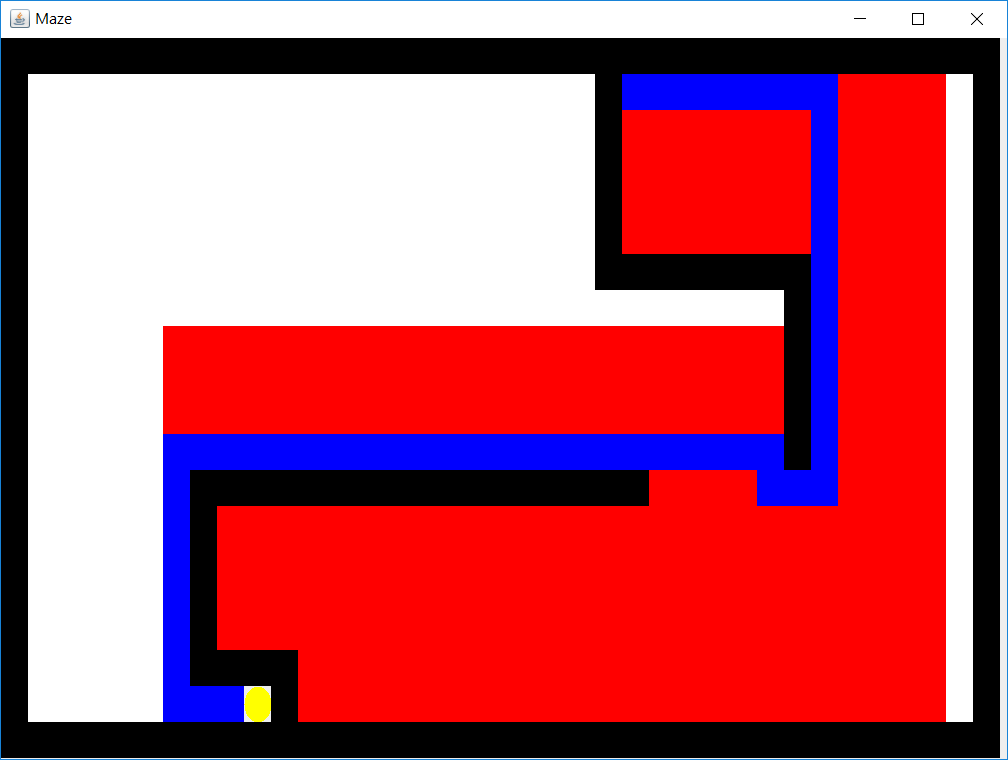
(3)

Total expanded node:127

1. **A\* search**

(1)





(2)

Total path cost:54

(3)

Total expanded node:358

-----------------------------------------------------------------------------------------------------------------

Implementation

(a)DFS

For the depth first search, we use stack as our data structure, and we keep trying each solution until it hits the end and has no place to go. We then pop the elements from the stack to return to the intersection and starts trying a new path. This often leads to the worst path cost, since no optimal solution is guaranteed. The solution is simply the first one we find.

(b)BFS

For the breadth first search, we are guaranteed the best solution, since we simply expand every node in each level. The data structure we use is Queue, since every node in each level is going to be expanded until we found the solution. So the space complexity is much higher. However, we can get the path with minimum cost.

(c)Best first search

This is greedy search, it’s similar to DFS, however, we add the heuristic function to make the solution more rational, rather then simply try each path without thinking. However, this kind of search can’t guarantee to find the best path cost, nonetheless, it’s usually much better than DFS.

(d)A\* search

This is the best method, it’s similar to BFS, however, we use the heuristic function to restrict the node to be expanded. So the total expanded node can be less, the heuristic function we choose is Manhattan distance, which is admissible. Since the grid is of Manhattan form, so no heuristic function can be less than the actual cost.

-----------------------------------------------------------------------------------------------------------------

**1-2:Multiple points**

1. Multiple points

b.Suboptimal

implementation:

We use 3 parameters for the heuristic function:

1.Whether or not we can eat the food pallet in next step(top priority)

2.Whether the state is visited(second top priority)

3.distance to the closest food pallet

In this case, we can eat all the food pallets with a total of 318 steps. The animation is given in the same file folder.

-----------------------------------------------------------------------------------------------------------------**2-1:Basic Sokoban**

**heuristic:**

For each step in the solving procedure, we define a data structure that records current position, the positions of all the boxes, the current Sokoban array state(what’s on each position, ex:human, box, storage point, wall, space, human on storage point, box on storage point), the path already taken, the direction already chosen(so as to avoid repeat visiting, thus we can be guaranteed to get the final answer), and the heuristic.

For the heuristic, we first calculate the summation of the distances from the boxesNotOnTheStorage to the closest storagePosition in order to move the boxes closer to the storage point. What’s more, to speed up the process, we would like to make the agent closer and closer to the box(so as to move them and change the state). So we add another heuristic that calculates the average distance from agent to all the boxes. Hence, we can be convinced that the agent will keep moving closer to the boxes and make the movement.

The heuristic is admissible because we calculate the distance from the boxStoragePoints to the closest storagePosition. Thus, no other total distance between boxes and storages found can be less then this.

For the distance between human agents and the boxes, we calculate their distance with Manhattan distance, which is also guaranteed to be admissible.

According to above, the heuristic is admissible.

a.

heuristic:

path length:34

answer:(up:0,right:1,down:2,left:3)

1, 2, 1, 2, 1, 2, 2, 3, 3, 1, 1, 0, 0, 3, 2, 3, 2, 1, 0, 0, 0, 3, 0, 0, 1, 2, 2, 2, 2, 0, 0, 3, 2, 2

node expanded:1119

running time:1.2 second

b.

heuristic:

input2.txt:

a.

path length:55

answer: (up:0,right:1,down:2,left:3)

0, 1, 2, 2, 0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 0, 3, 3, 2, 3, 0, 1, 1, 1, 2, 3, 1, 1, 1, 0, 0, 3, 0, 0, 3, 3, 2, 2, 0, 0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 0, 3, 2, 3, 1, 0, 3

node expanded:2024

running time:3.3 second

input3.txt:

a.

path length:34

answer: (up:0,right:1,down:2,left:3)

1, 0, 1, 1, 2, 2, 2, 2, 3, 2, 1, 0, 0, 0, 0, 3, 3, 3, 1, 2, 1, 2, 1, 2, 2, 3, 3, 2, 3, 3, 0, 0, 2, 1

node expanded:14795

running time: 107.2 second

b.

heuristic:

input4.txt:

a.

path length:146

answer: (up:0,right:1,down:2,left:3)

2, 2, 0, 0, 1, 1, 2, 2, 3, 3, 1, 1, 2, 2, 3, 3, 0, 3, 3, 0, 0, 1, 2, 1, 2, 2, 1, 1, 0, 0, 3, 3, 1, 1, 0, 0, 3, 3, 2, 2, 0, 3, 3, 2, 2, 2, 1, 1, 0, 0, 3, 1, 1, 1, 0, 0, 3, 3, 2, 3, 2, 1, 2, 2, 3, 0, 1, 0, 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 0, 0, 1, 3, 2, 2, 1, 1, 0, 0, 0, 2, 3, 3, 2, 3, 3, 2, 1, 1, 0, 0, 0, 3, 2, 1, 2, 2, 3, 3, 0, 1, 0, 1, 1, 1, 2, 2, 3, 3, 0, 3, 0, 1, 1, 3, 2, 2, 1, 1, 0, 0, 3, 3, 3, 2, 3, 2, 1, 1, 1, 3, 0, 0, 3, 0, 3, 2

node expanded:29112

running time: 362.3 second

b.

heuristic: