Search in Pacman

Harbin Institute of Technology (Shenzhen)

Intelligent Computation Research Center

Date: 10/09/2017

目录

[1 Introduction 1](#_Toc495178525)

[2 Analysis and implement 1](#_Toc495178526)

[2.1 Question 0 1](#_Toc495178527)

[2.2 Question 1 2](#_Toc495178528)

[2.2.1 Algorithm Descriptions 2](#_Toc495178529)

[2.2.2 Implement 3](#_Toc495178530)

[2.3 Question 2 6](#_Toc495178531)

[2.3.1 Algorithm Descriptions 6](#_Toc495178532)

[2.3.2 Implement 7](#_Toc495178533)

[2.4 Question 3 10](#_Toc495178534)

[2.4.1 Algorithm Descriptions 10](#_Toc495178535)

[2.4.2 Implement 11](#_Toc495178536)

[2.5 Question 4 14](#_Toc495178537)

[2.5.1 Algorithm Descriptions 14](#_Toc495178538)

[2.5.2 Implement 16](#_Toc495178539)

[2.6 Question 5 17](#_Toc495178540)

[2.6.1 Algorithm Descriptions 17](#_Toc495178541)

[2.6.2 Implement 18](#_Toc495178542)

[2.7 Question 6 20](#_Toc495178543)

[2.7.1 Algorithm Descriptions 20](#_Toc495178544)

[2.7.2 Implement 22](#_Toc495178545)

[3 Conclusion 24](#_Toc495178546)

[3.1 Algorithm performance comparison 24](#_Toc495178547)

[3.2 Experiences from project 24](#_Toc495178548)

[4 Task Assignment 25](#_Toc495178549)

# 

# 1. Introduction

A Pacman agent needs to efficiently find paths through a maze, either to reach a particular location or collect remaining food quickly. In this project, we will build general search algorithms and apply them to Pacman scenarios. We just need to fill in portions of search.py and searchAgents.py during the assignment.

There are two parts of algorithms in the project, which are the search algorithms and food collecting algorithms. The first part only has one dot which includes Question 1, Question 2, Question 3, Question 4. The second part has many dots which includes Question 5, Question 6.

# 2. Analysis and implement

## 2.1 Question 0

Question 0 is initializing the Pacman game. In this question, SearchAgent is provided, so that we need not define a new agent. In order for the search agent to successfully work, it requires a search function. We are provided with tinyMazeSearch which returns the actions required to solve tinyMaze. The command is shown as follows.

python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch

The result is shown in figure 2-1.

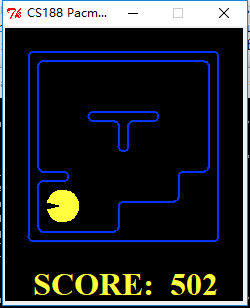


Figure 2‑1 The result of Q0

The output of console is shown in figure 2-2.

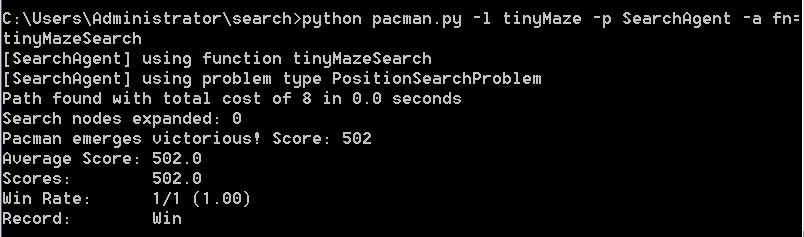


Figure 2‑2 The output of Q0 console

## 2.2 Question 1

### 2.2.1 Algorithm Descriptions

Question 1 is implementing the depth-first search (DFS) algorithm. Depth First Search algorithm is a kind of algorithm used to traverse the search tree or graph. It traverses the tree nodes as deep as possible along the depth of the tree branches. When a node’s sides have been explored, search will be back to the start node of the first found edge. This process will be continued until all nodes have the paths from source node to themselves. If there is a node not found, choose one of them as the source node and repeat the above process. The whole process is repeated until all the nodes are accessed. We use a stack which follows the principle of first in last out (FILO) as the basic storage structure. The procedure is shown in figure 2-3.

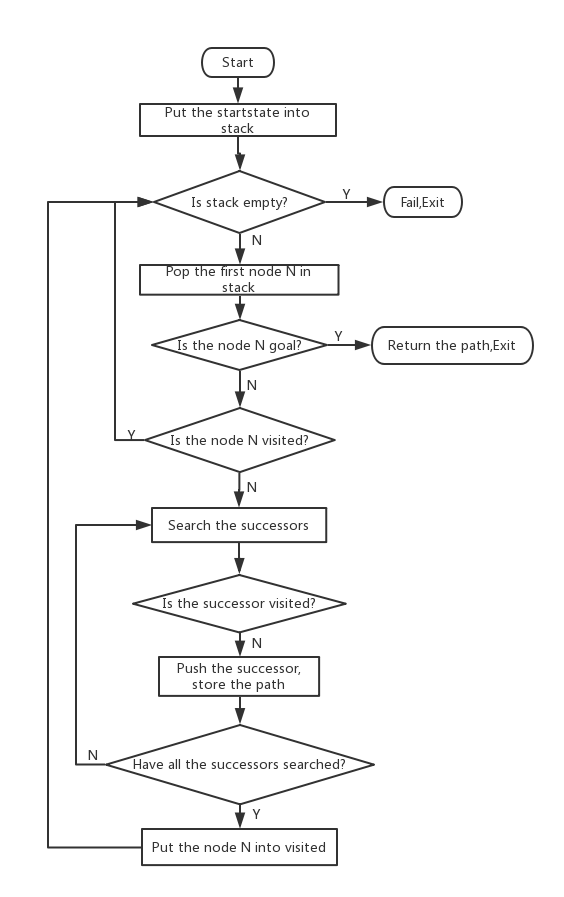


Figure 2‑3 The procedure of DFS

### 2.2.2 Implement

We implement the DFS algorithm in the depthFirstSearch function which is shown in figure 2-4.

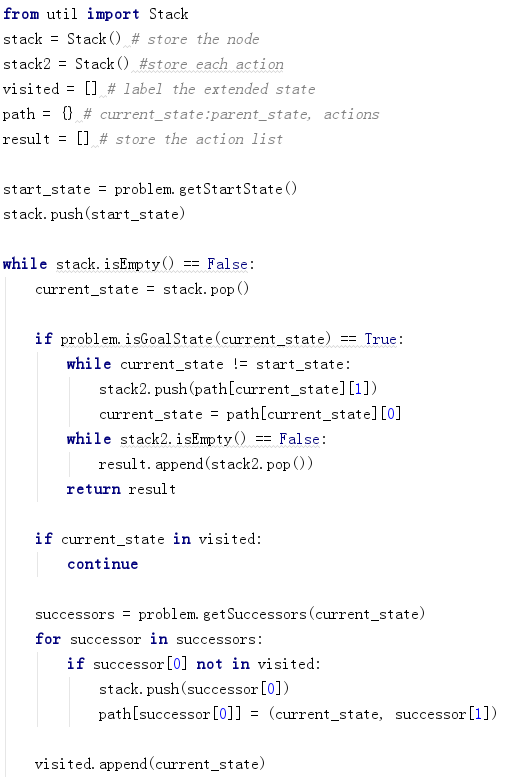


Figure 2‑4 The code of DFS

We test the DFS algorithm in both tinyMaze and mediumMaze. The command to run the DFS algorithm is shown as follows.

python pacman.py -l tinyMaze -p SearchAgent -a fn= depthFirstSearch

python pacman.py -l mediumMaze -p SearchAgent -a fn= depthFirstSearch

The results are shown in figure 2-5 and 2-6.

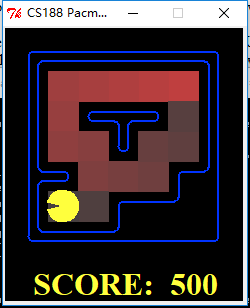


Figure 2‑5 The result of DFS in tinyMaze

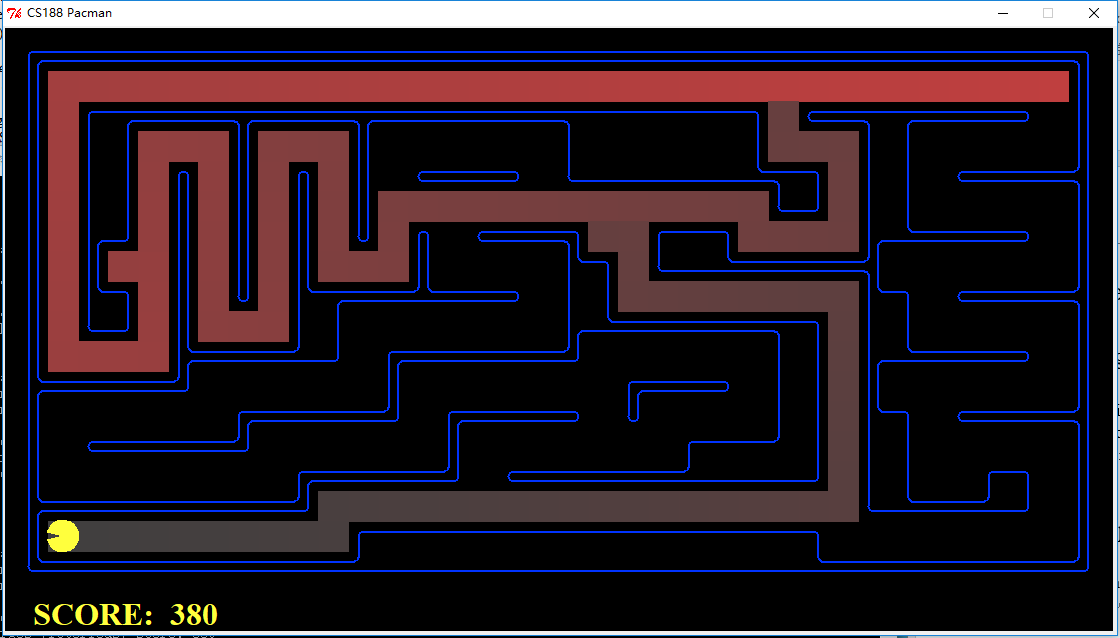


Figure 2‑6 The result of DFS in mediumMaze

The outputs of console are shown in figure 2-7 and 2-8.

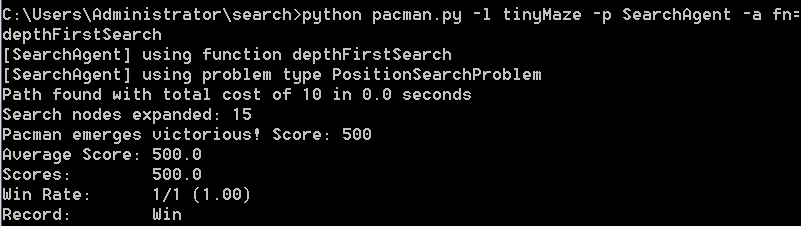


Figure 2‑7 The output of DFS console in tinyMaze

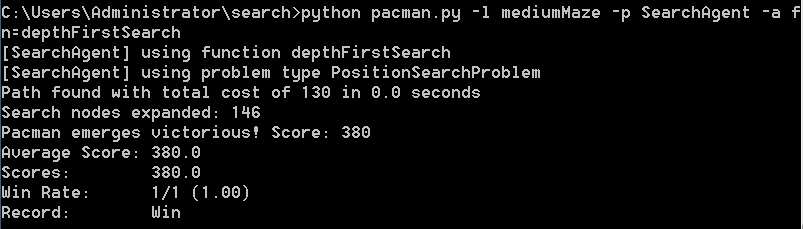


Figure 2‑8 The output of DFS console in mediumMaze

## 2.3 Question 2

### 2.3.1 Algorithm Descriptions

Breadth-first search(BFS) is a search algorithm which starts at the tree root or some arbitrary node of graph. The core is that exploring the neighbor nodes first before moving the next level neighbors.

Our goal is to implement the breadth-first search (BFS) algorithm in the breadthFirstSearch function in search.py. Again, write a graph search algorithm that avoids expanding any already visited states. Test your code the same way you did for depth-first search.

The procedure is shown in figure 2-9.

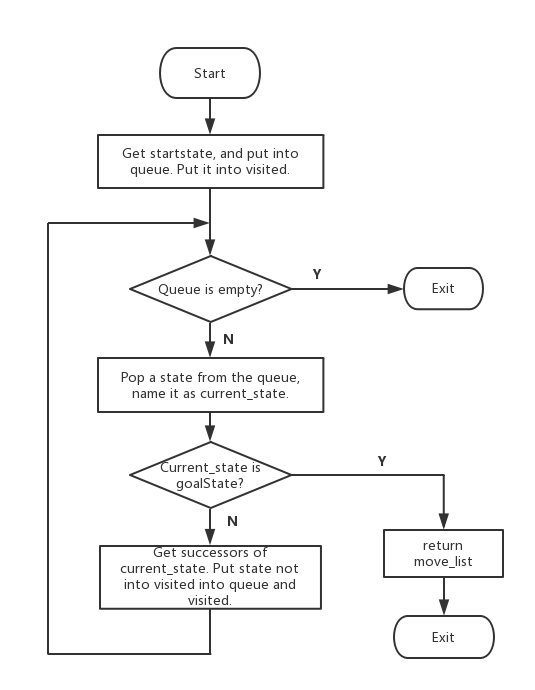


Figure 2‑9 The procedure of BFS

### 2.3.2 Implement

Here is the code:

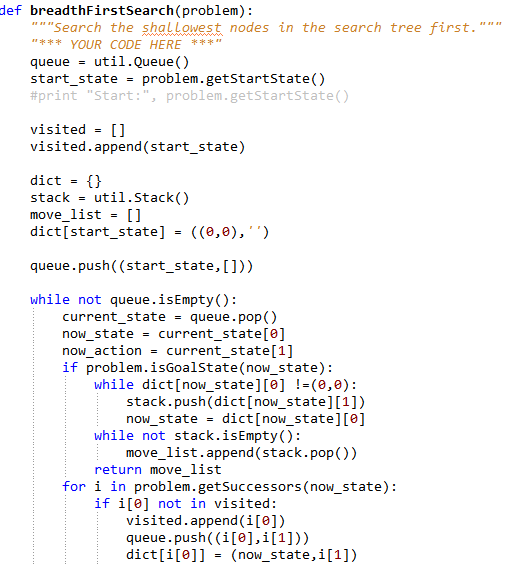


Figure 2‑10 The code of BFS

Run the command: python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs Then the result is show in figure 2-11 and figure 2-12. We get 442 score.

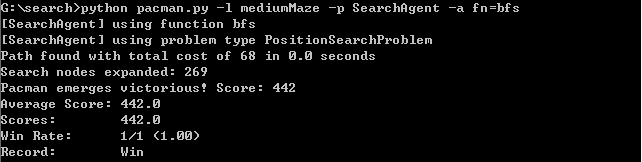


Figure 2‑11 The result of BFS in mediumMaze

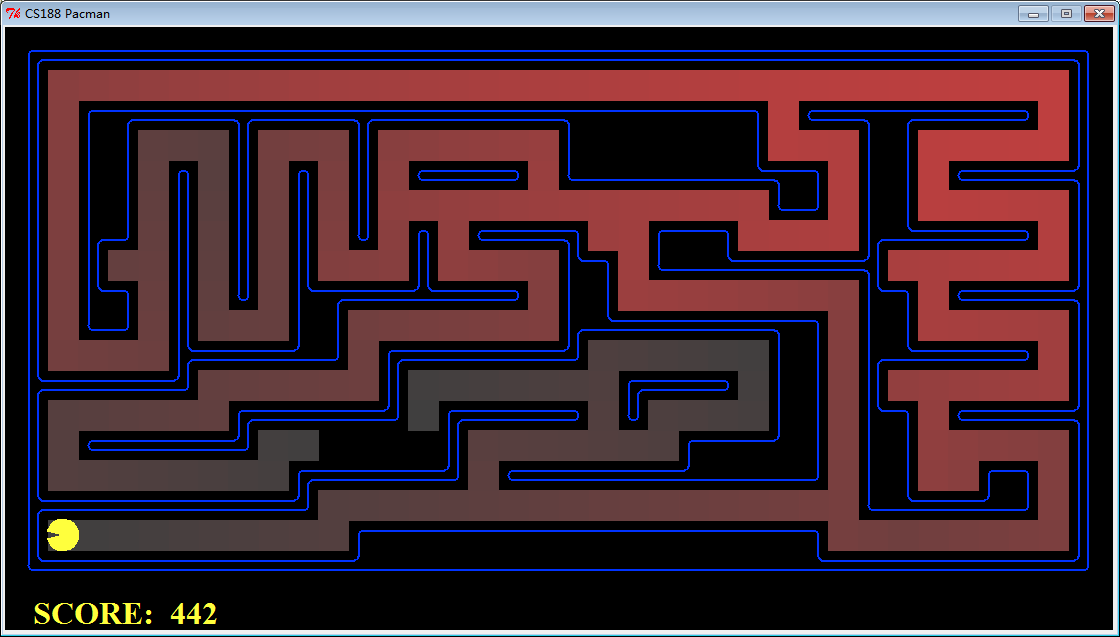


Figure 2‑12 The result of BFS in mediumMaze

Run the command: python pacman.py -l bigMaze -p SearchAgent -a fn=bfs –z .5

And the result is show in figure 2-13 and figure 2-14. We get 300 score.

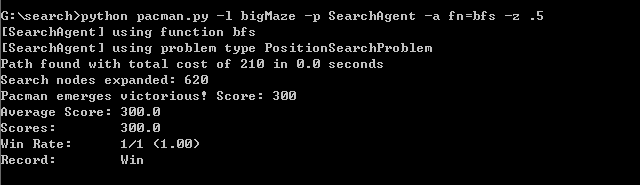


Figure 2‑13 The result of BFS in bigMaze

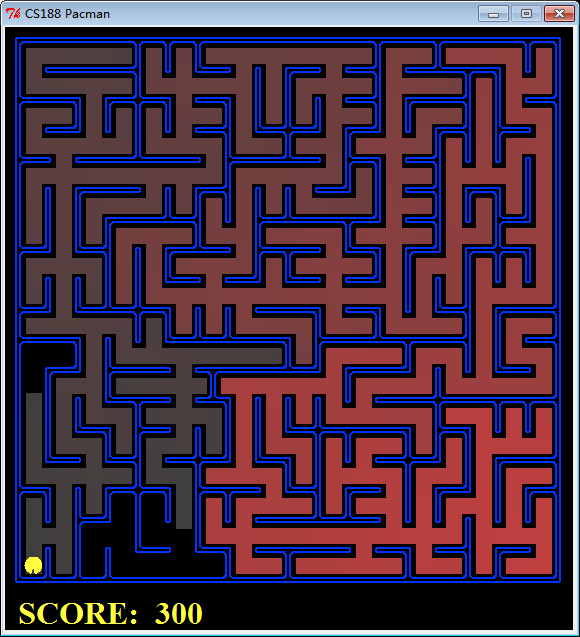


Figure 2‑14 The result of BFS in bigMaze

## 2.4 Question 3

### 2.4.1 Algorithm Descriptions

Question 3 is implementing the uniform-cost graph search algorithm (UCS) algorithm. Uniform-cost search does not care about the number of steps a path has, but only about their total cost. We use a priority-queue, which sorts the nodes based on priority when the nodes are pushed in the queue, as the basic storage structure. The procedure is shown in figure 2-15.

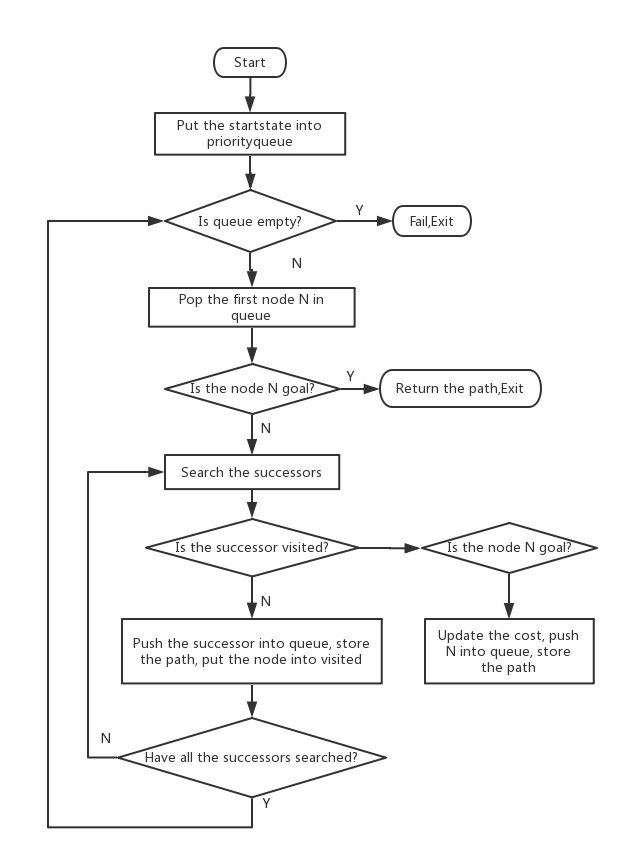


Figure 2‑15 The procedure of UCS

### 2.4.2 Implement

We implement the UCS algorithm in the uniformCostSearch function which is shown in figure 2-16.

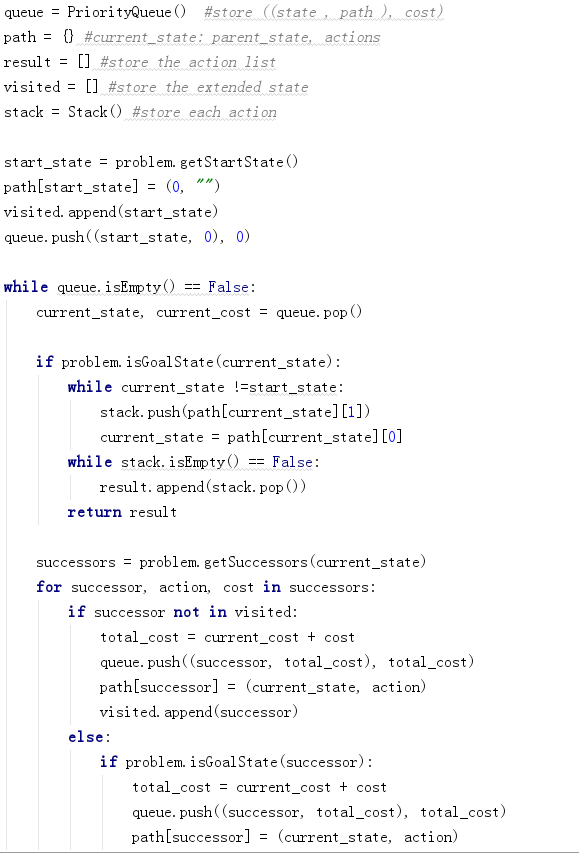


Figure 2‑16 The code of UCS

We test the UCS algorithm in mediumMaze. And we now observe different behavior in all three of these conditions, where the agents below are all UCS agents which differ only in the cost function . The command to run the UCS algorithm is shown as follows.

python pacman.py -l mediumMaze -p SearchAgent -a fn= uniformCostSearch

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

The results are shown in figure 2-17 2-18 2-19.

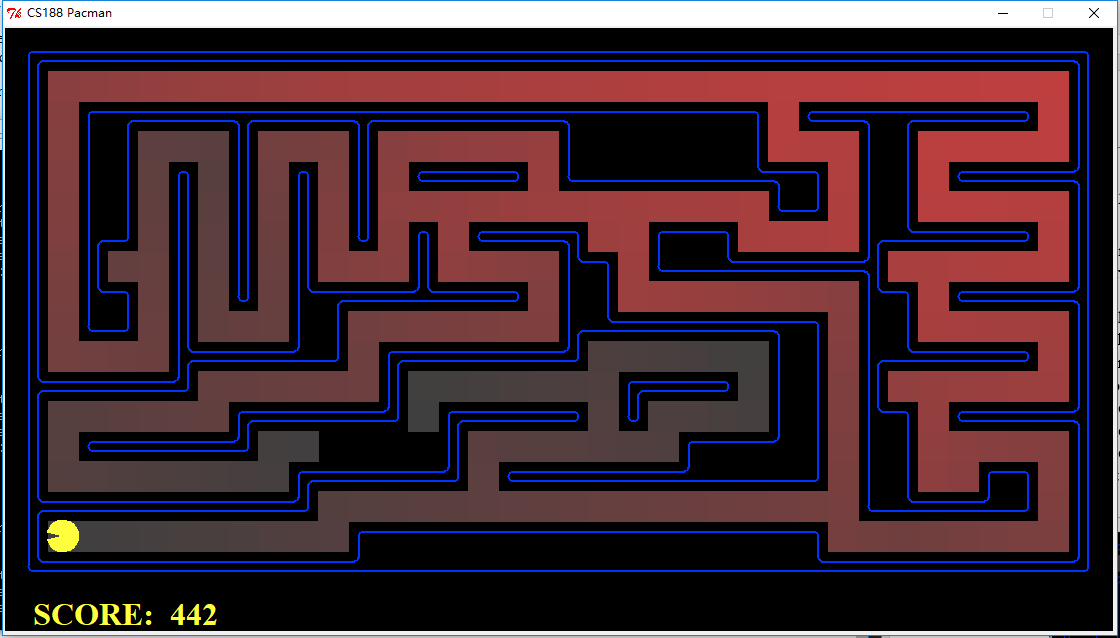


Figure 2‑17 The result of UCS in mediumMaze

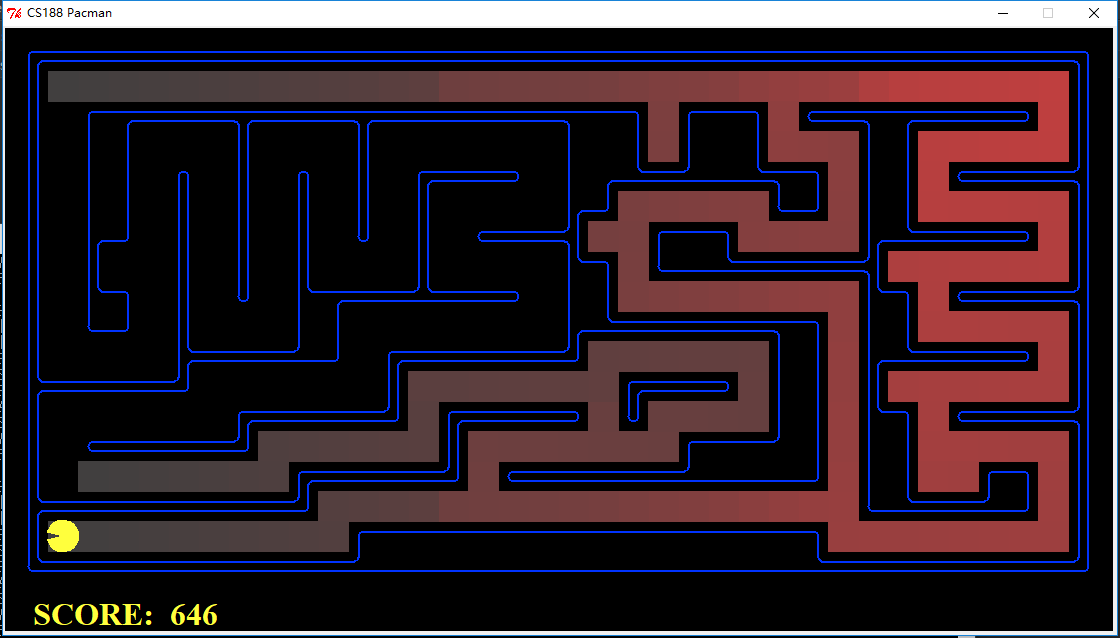


Figure 2‑18 The result of UCS in mediumDottedMaze



Figure 2‑19 The result of UCS in mediumScaryMaze

The outputs of console are shown in figure 2-20 2-21 2-22.

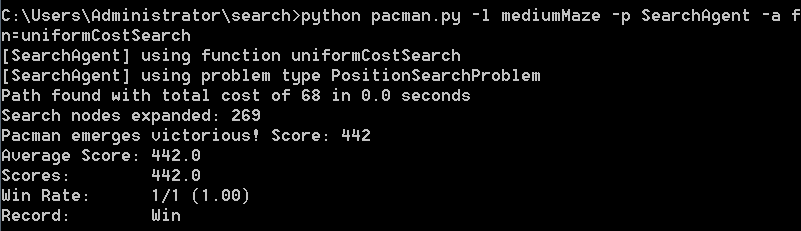


Figure 2‑20 The output of UCS console in mediumMaze

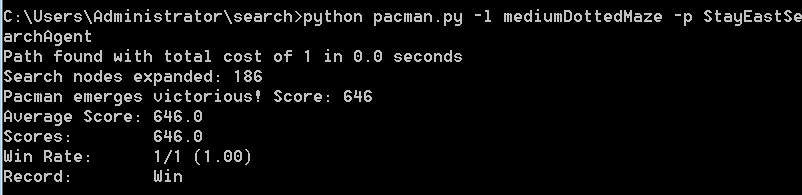


Figure 2‑21 The output of UCS console in mediumDottedMaze

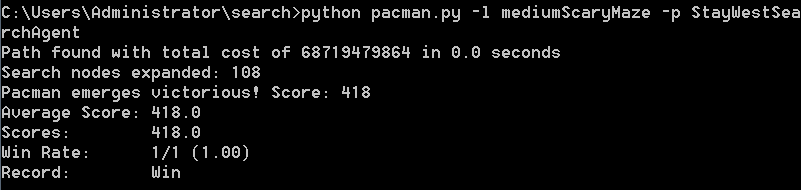


Figure 2‑22 The output of UCS console in mediumScaryMaze

## 2.5 Question 4

### 2.5.1 Algorithm Descriptions

A\* graph search is a typical heuristic Search algorithm. We select the node n which has the minimum f(n)=g(n)+h(n) to expand. N is the last node on the path. G(n) is the cost of the path from the start node to n. H(n) is a heuristic that estimates the cost of the cheapest path from n to the goal. We use PriorityQueue as the open list. We regard f(n) as the priority of n.

The goal is to implement A\* graph search in the empty function aStarSearch in search.py. You will need to pass a heuristic function into aStarSearch upon construction. The heuristic function should take one argument: a state in the search problem. See the nullHeuristic heuristic function in search.py for an example.

The procedure is shown on the figure 2-23.

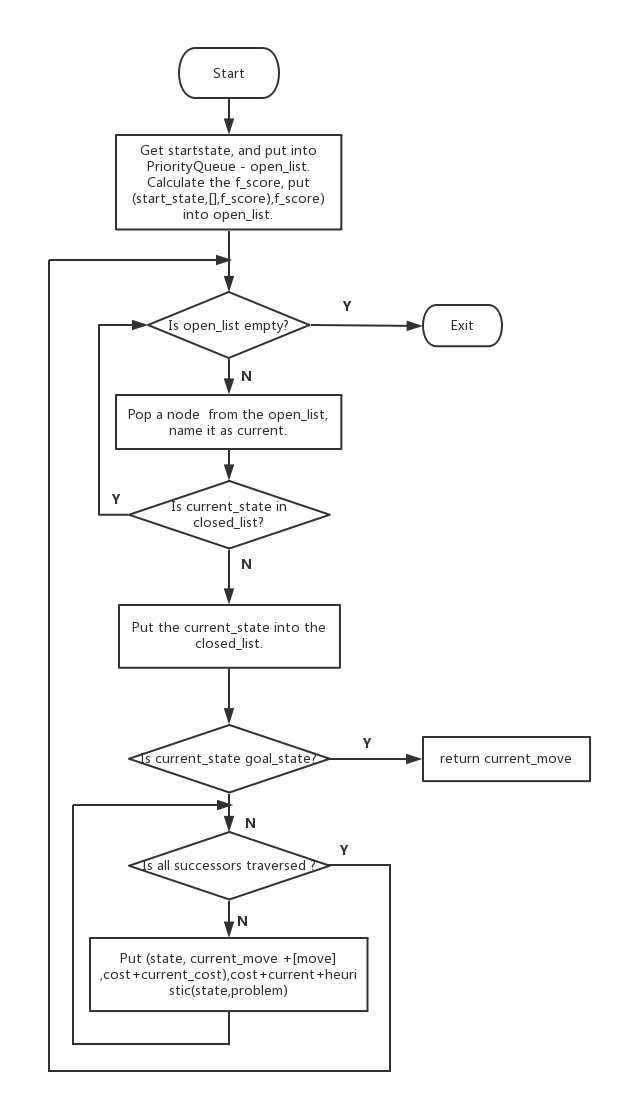


Figure 2‑23 The procedure of A\*

### 2.5.2 Implement

Run the command: python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic.

Here is the code:

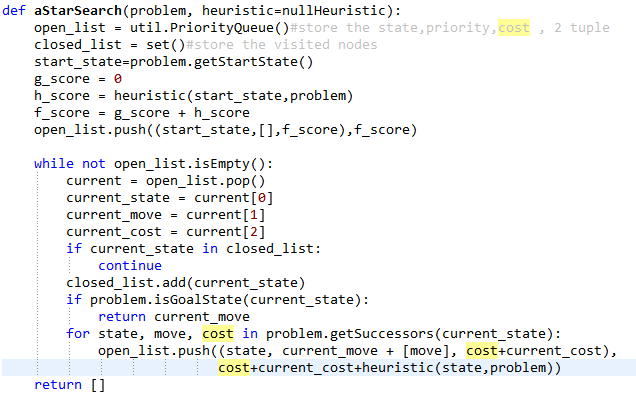


Figure 2‑24 The code of A\*

And the result is show in figure 2-25 and figure 2-26. We get 300 score.

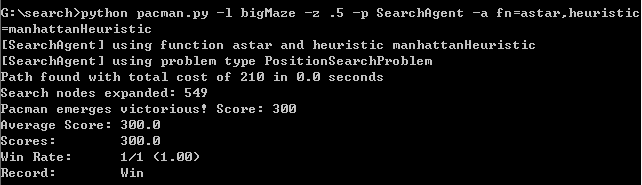


Figure 2‑25 The output of A\* console

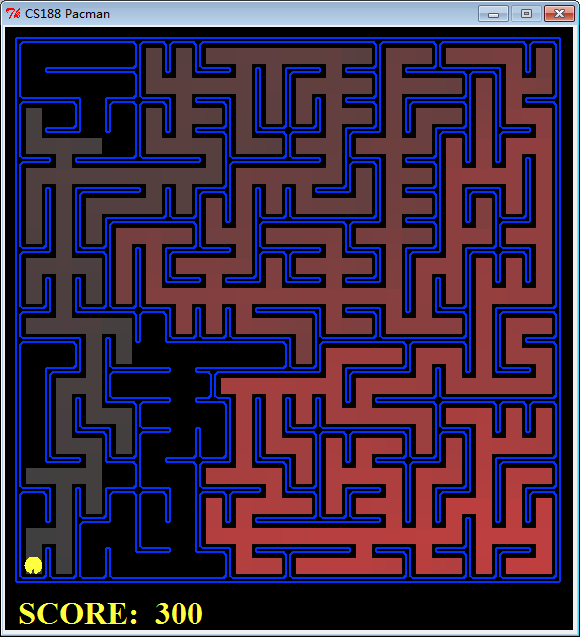


Figure 2‑26 The result of A\*

## 2.6 Question 5

### 2.6.1 Algorithm Descriptions

This question is to eat all the beans with as little steps as possible.If we have written our general search methods correctly, A\* with a null heuristic (equivalent to uniform-cost search) should quickly find an optimal solution to testSearch with no code change on our part.

And then, in the class FoodSearchProblem, we define the function foodHeuristic and construct the appropriate heuristic function to complete the bean searching problem.

### 2.6.2 Implement

Run the command: python pacman.py -l testSearch -p AStarFoodSearchAgent.

And the result is show in figure 2-27 and figure 2-28. We get 513 score.

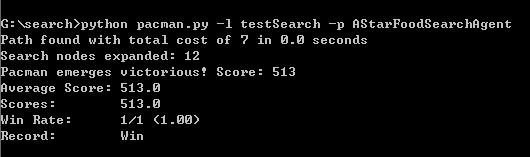


Figure 2‑27 The output of Q5(a) console

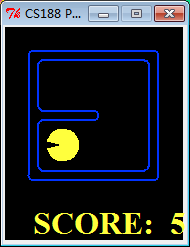


Figure 2‑28 The result of Q5(a)

Run the command: python pacman.py -l trickySearch -p AStarFoodSearchAgent

Here is the code:

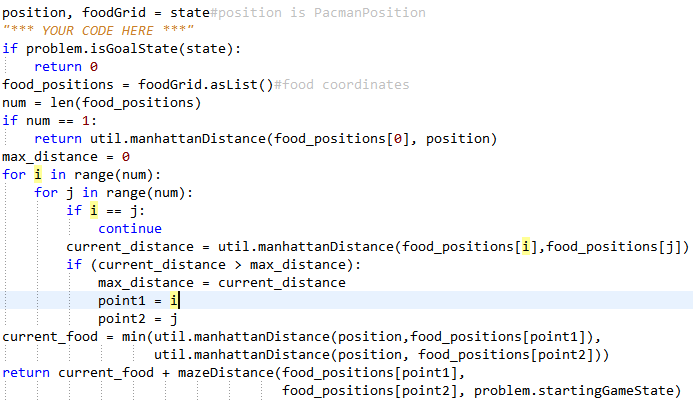


Figure 2‑29 The code of Q5(b)

And the result is shown in figure 2-30 and figure 2-31. We get 570 score.

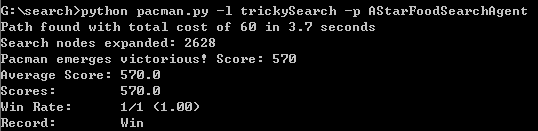


Figure 2‑30 The output of Q5(b) console

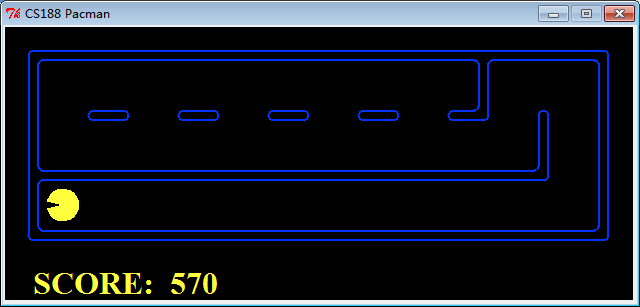


Figure 2‑31 The result of Q5(b)

## 2.7 Question 6

### 2.7.1 Algorithm Descriptions

Question 6 is implementing greedy graph search. A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. We use two different solutions to solve suboptimal search problem. The first solution is that implement greedy graph search in the function greedySearch in search.py and inherit a class GreedyFoodSearchAgent. We use a priority-queue, which sorts the nodes based on priority when the nodes are pushed in the queue, as the basic storage structure. The second solution is that implement the agent ClosestDotSearchAgent in searchAgents.py and the function findPathToClosestDot in searchAgents.py.

The procedure of the greedySearch is shown in figure 2-32.

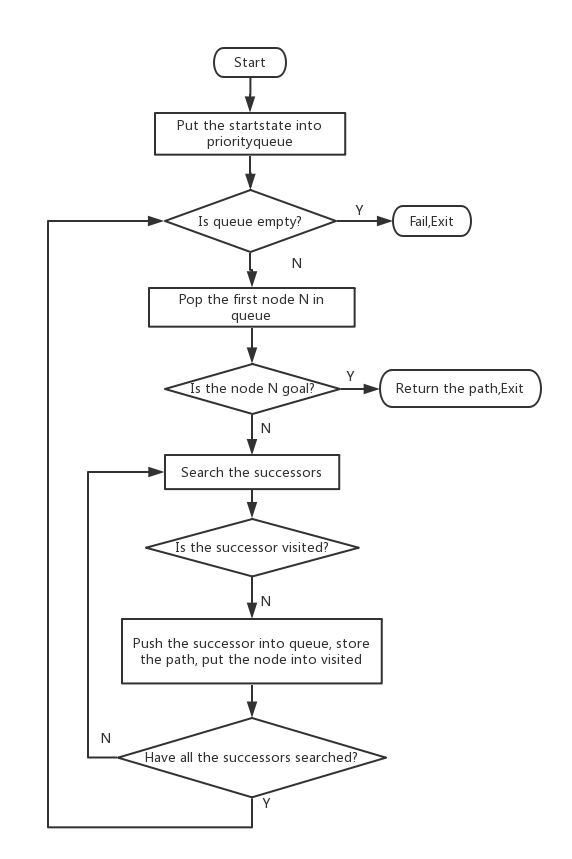


Figure 2‑32 The procedure of greedy search

### 2.7.2 Implement

In the first solution, we implement the greedy search algorithm in the greedySearch function which is shown in figure 2-33.

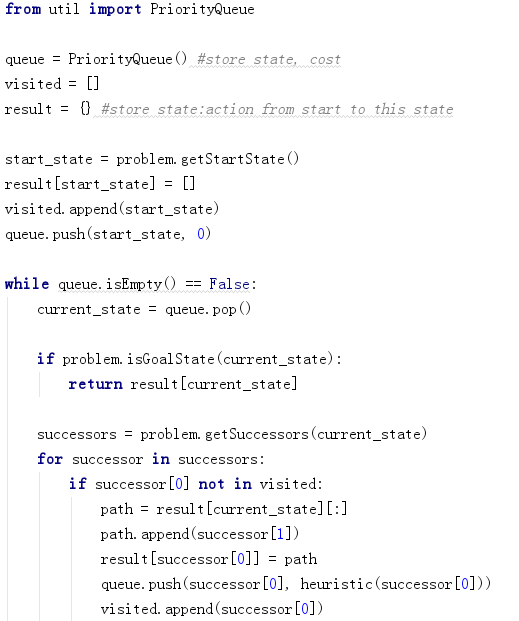


Figure 2‑33 The code of greedy search

In the second solution, we implement the greedy search algorithm in the findPathToClosestDot function which is shown in figure2-34.

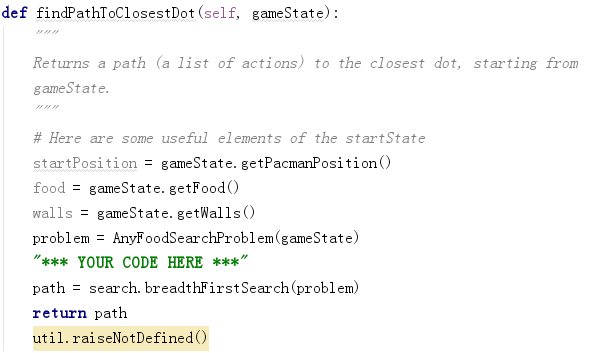


Figure 2‑34 The code of findPathToClosestDot

We test the greedy search algorithm in bigSearch. The command to run the greedy search algorithm is shown as follows.

python pacman.py -l bigSearch -p GreedyFoodSearchAgent -z .5

The result is shown in figure 2-35.

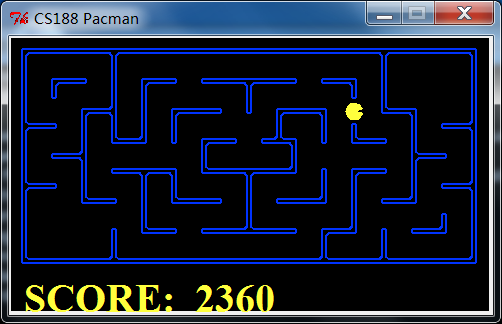


Figure 2‑35 The result of Q6

The output of console is shown in figure 2-36.

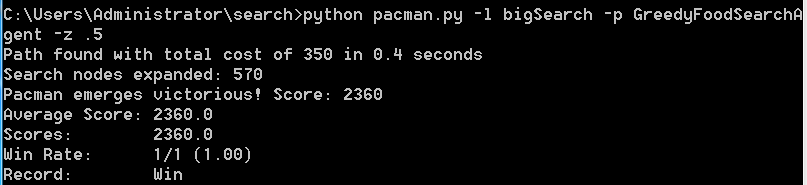


Figure 2‑36 The output of Q6 console

# 3. Conclusion

## 3.1 Algorithm performance comparison

The comparison of several algorithms performance is shown in table 1.

Table 1 The comparison of several algorithms performance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Question** | **Layout** | **Nodes expanded** | **Total cost** | **Total time** | **Score** |
| Q1 | tinyMaze | 15 | 10 | 0.0 | 500 |
| mediumMaze | 146 | 130 | 0.0 | 380 |
| Q2 | mediumMaze | 269 | 68 | 0.0 | 442 |
| bigMaze | 620 | 210 | 0.0 | 300 |
| Q3 | mediumMaze | 269 | 68 | 0.0 | 442 |
| mediumDotted | 186 | 1 | 0.0 | 646 |
| mediumScary | 108 | -- | 0.0 | 418 |
| Q4 | bigMaze | 549 | 210 | 0.0 | 300 |
| Q5(a) | testSearch | 12 | 7 | 0.0 | 513 |
| Q5(b) | trickySearch | 2628 | 60 | 3.7 | 570 |
| Q6 | bigSearch | 570 | 350 | 0.4 | 2360 |

## 3.2 Experiences from project

In this experiment, we implement the DFS, BFS, UCS, A\* and greedy search algorithms. Compared with DFS and BFS, UCS is more efficient and is equivalent to BFS when the search method fails. A \* algorithm is the most effective direct search algorithm by using the formula for preprocessing, which can omit a large number of unnecessary search path to improve the efficiency.

To be honest, We learned a lot from the project. Firstly, We have a deep comprehension of these algorithms by implementing them. Then, We can programme the project by Python deftlier. Thirdly, We know that how to complete the certain functions in part-filled project.

# 4. Task Assignment

|  |  |
| --- | --- |
| Name | Task Assignment (code and report) |
| Leng Jia(Group leader) | Q0、Q1、Q3、Q6 |
| Yan Hongyu | Q2、Q4、Q5(a)(b) |