# 1st Project Report

- Introduction of algorithm implementation

To implement the depth-first search algorithm (Q1), I apply the structure of stack. The agent walks along the deepest part of the selected road. When it finds no beans or hits the wall, it will turn. From the figure 1, we can see that the agent doesn't actually go to all the explored squares on his way to the goal. The reason is that the DFS algorithm excludes the case that the child node of the descendant node are the ancestor nodes in the process of extending the node.

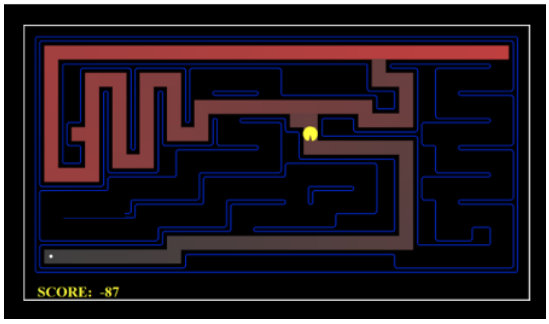


figure 1:DFS

The solution found by DFS algorithm for mediumMaze should have a length of 130. However, this isn't a least cost solution. Considering the implementation process of DFS algorithm, the agent always prioritizes the currently selected path and ignores the selection of the shorter path that may exist.

The BFS algorithm (Q2) takes this into account. The BFS algorithm adopts the queue structure, which expands all the child nodes under the selected parent node. As you can see figure 2, the agent has surveyed all the alternative routes it can take before starting the operation, and selects the most efficient route.

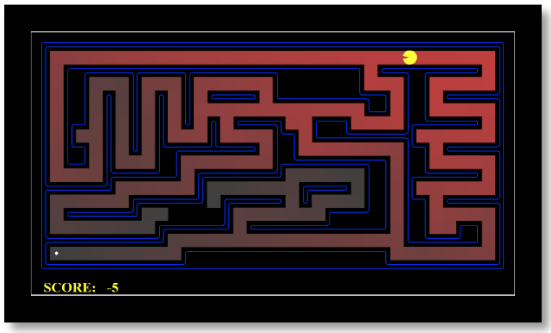


figure 2:BFS

While the BFS will find a path with the fewest actions to the goal, we may want to find a path that is "best" in other senses. In ghoulish places, we can charge more for dangerous steps, and where food is abundant, we can charge less for dangerous procedures. The UCS algorithm (Q3) can be applied to these cases. USC algorithm adopts the data structure of priority queue, which blurs the concept of nodes and only sorts the cost of each road, choosing the path with the lowest cost first. As can be seen from figure 3, the agent preferred the road with more beans, while in the figure 4, the

agent preferred the road without monsters.

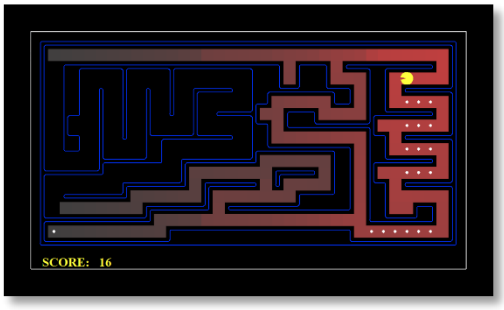


figure 3

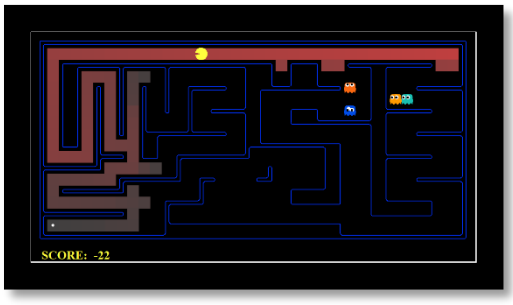


figure 4

The above three algorithms are all uninformed search algorithms, but A* algorithm is an informed algorithm (Q4), which is more efficient than the above algorithms. g(n) is the cost of the path represented by node n while h(n) is the heuristic estimate of the cost of achieving the goal from n. By summing h(n) and g(n) to calculate the cost, the agent can find beans faster, because the A* algorithm expands fewer nodes than the BFS algorithm.

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 617
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
```

figure 5

```
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
```

figure 6

To find the four dots, I store the state of the four corners beans with the four_corners variable (True if eaten, False if not). I use successors variables to store the currently uneaten dots. The isGoalState function is used to determine whether the agent had eaten all the dots (Q5) .

The cornersHeuristic function (Q6) is used to calculate the heuristic estimate of the cost of achieving the goal, namely h_n. The specific design idea of the function is as follows. Firstly taking the current state of the agent as the starting point, and each of the four corners as the end point. Then calculating the Manhattan distance between the two points and select the minimum distance among the four distances. Taking the point of minimum distance as the starting point of the new state and the remaining three corners as the end point, carry out the same calculation, and proceed successively to obtain the shortest distance h_n.

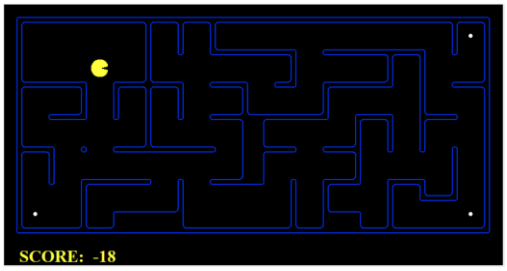


figure 7:the answer of Q6

To eat all the dots (Q7), I borrowed ideas from github and define a new function called food_distance, which helps calculate the heuristic distance from the current state to the all dots. And using the A* algorithm of Q4, it's easy to find the answer.

Sometimes, even A* algorithm with good heuristics, it is difficult to find the optimal path among all the points. In these cases, I let the agent always greedily eat the closest dot (Q8). However ClosestDotSearchAgent won't always find the shortest possible path through the maze. For example, in the figure 8, if the agent always greedily eat the closest dot, then it will end up eating the middle bean. In this case, the agent doesn’t find the shortest.

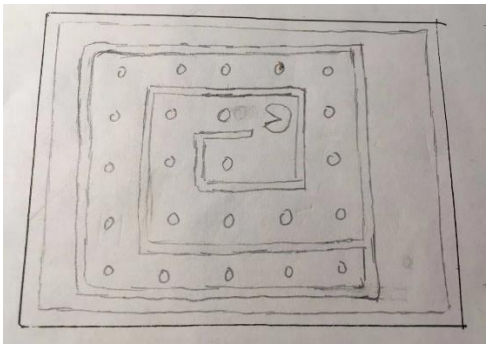

figure 8: Special case

- node and state

For specifying a node, its ancestor and descendant nodes are determined to be immutable while for a state, the previous state corresponding to each state can be changed due to different paths selected. For this game, node can not only be regarded as the current position of the agent, but also can be expanded before the agent moves forward, so as to provide multiple paths for the agent to choose. State only refers to the current position of the agent in the process.