Assignment 3 - COSE361 (03)

2021320117 Bae Minseong

I. Result of pacman.py with layout test71.lay

```
(base) ➜  minicontest1 python pacman.py --agent MyAgent --layout test71.lay
Pacman emerges victorious! Score: 756
Average Score: 756.5061668396015
Scores:        756.5061668396015
Win Rate:      1/1 (1.00)
Record:        Win
```

II. Descriptions for MyAgent

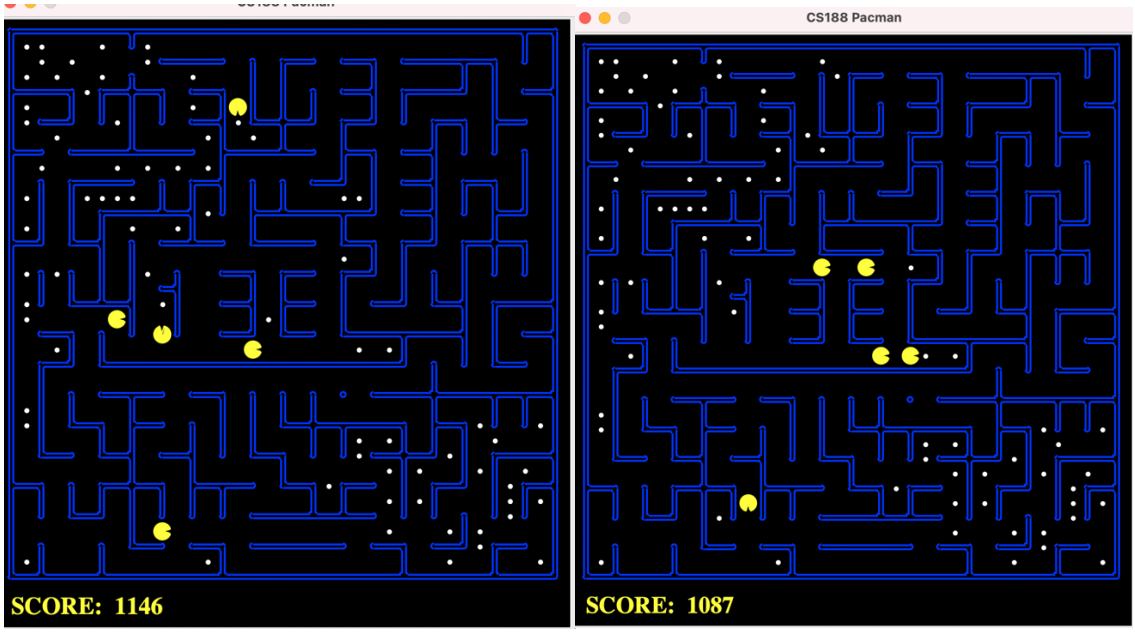
I will briefly explain the main algorithm and logical flow in this document and detailed comments will be in myAgents.py.
Basically, my algorithm is based on Breadth-First Search. Similarly, like ClosestDotAgent in the example code, I also implement the method findPathToClosestDot with BFS.

The key points for my findPathToClosestDot function are
1) Using dictionary and set structure provided by Python, I keep saving the information for next goal position (i.e., position for next closest food) and directions to go there. This information is used for checking whether the BFS function visit the position that will be visited by other pacman agent.
2) If the remaining number of foods are smaller than the number of positions that the pacmen will go, I implemented simplified version of my own BFS, reducedBFS to reduce the computation.
3) Basically, for the BFS and reducedBFS, there exists a computation limit. If the code computes the next state too many times, it terminates the computation and just make that pacman stop.
4) I used dictionary to save the information for actions and find solution by backtracking with that dictionary in BFS, because it is much faster than just simple list.append().

III. Discussions

Q1. Discuss cases where the agent implemented by yourself is better than the baseline.

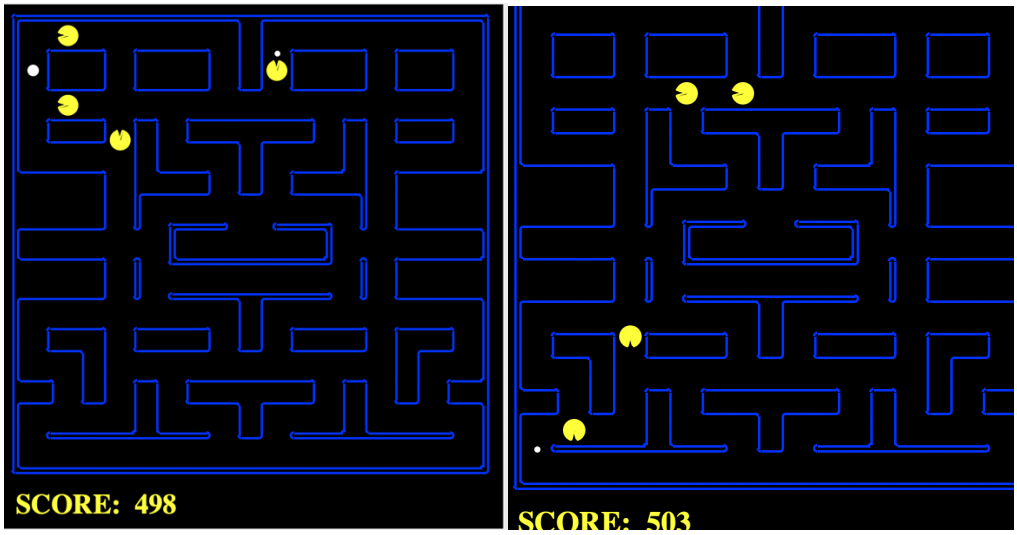
For almost every case that there are multiple pacmen in the game, my agent will be better than the baseline. Because first, the baseline code uses the algorithm based on uniform-cost search, so it keeps calculating the cost for the possible positions. However, AnyFoodSearchProblem don't have to consider cost of actions because each move just has same unit cost. In this perspective, computing costs will make the baseline algorithm more complicated. In addition, there exists a computation limit for my algorithm, so it will be much efficient than the baseline.
Also, my agent will be more efficient in multiple-agent games than the baseline because of saving and using information for positions that will be visited by other pacman agent. In the baseline algorithm, it just briefly checks whether the next position is already visited or not. However, my modified BFS algorithm with currentGoalList, checks if the next position is planned to be visited by other pacman agent although it is goal position. This makes a huge difference for eating all food in shorter time.
For example, in the layout test12.lay,



for the similar situation of remaining dots, (left one) pacman agents in myAgent algorithm are separated and find the dots individually but (right one) pacmen in the baseline algorithm make a flock because multiple pacmen move to the same target food until that food is eaten.

Q2. Discuss cases where the agent implemented by yourself is worse than the baseline.

Because of the 'Stop' usage in myAgent algorithm, there can be a case worse than the baseline. Let's look at the layout test72.lay.



In the myAgent algorithm, if the search faces the computation limit or there are no more dots to eat because other pacman agents are planning to eat them, the agent immediately stops. So, like in the left picture, pacman agents in myAgent don't move if other pacman plan to visit the dot although the pacman itself is closer to the dot. In the left picture, the undermost pacman is moving to eat the last dot although the rightmost pacman can eat the dot just with 1 move. It is very inefficient situation.

However, all the pacman agents in the baseline algorithm keep moving until their individual target dots are eaten. Like the right picture, all the 4 pacmen move to eat the last remaining dot, so the closest pacman will eat the dot.

So like this example, there can be cases myAgent is worse than the baseline one.

Q3. Ask yourself one question and answer.

Question: Suggest a way to improve your MyAgent algorithm.

Answer:

In my MyAgent algorithm, I made a computation limit to make the better score.
There is a computation limit value for each BFS and reducedBFS (601 in BFS, 18 in reducedBFS).
So, if we use **hyperparameter optimization** like Grid Search for these values, we can make a better performance for MyAgent algorithm.