Mengjie Wang, Jun Zhou

September 16, 2018

CS420

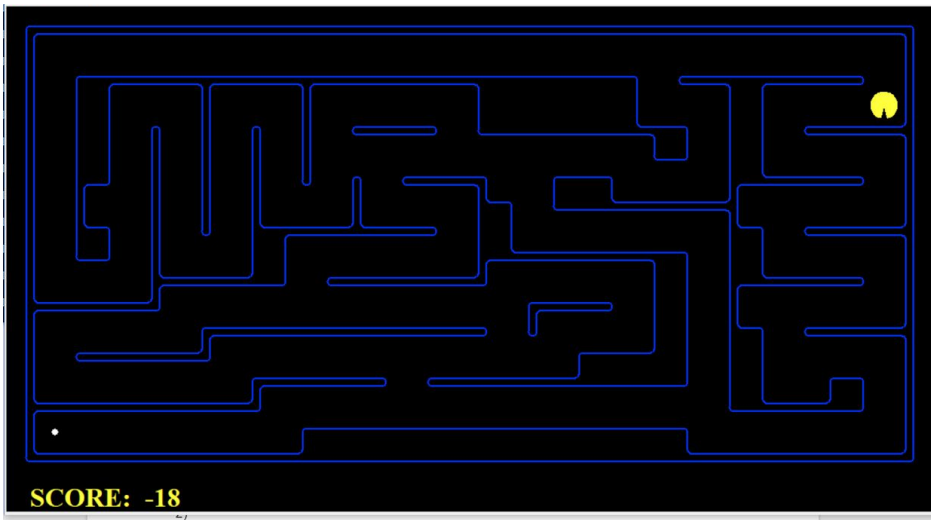
Professor Tao

Project Report

1)

The RandomAgent just select a random action from its action set, so it does not care which action will result a status closer to the food or get more score. Because stop will always be a legitimate action, it will stay over the position and do nothing sometime. Although random agent will not try to find an action that leads to the foods, it will not run into exceptions as all the actions it chooses from it legitimate. Also, after a long time, the agent will lead the pacman to the food in a small graph.

2)



3)

This BetterRandomAgent is basically the same as RandomAgent, except the fact that the pacman will never stay the same position and do nothing at any situation, since this BetterRandomAgent eliminates the stop action from its action set. This will slightly increase the efficiency for the pacman to finally reach the food although it still take a long time as it is just randomly choose the action.

4)

In the medium or large layout, ReflexAgent react almost exactly the same as the BetterRamdomAgent, because it only consider if the next status that the action leads to have a food or not. If there is a food in the nest status, the agent will take that action. If none of the action leads to a food, it will just randomly choose one in its action set. So in a bigger layout, since there will be only one food and it is hard to get, the improvement of ReflexAgent over BetterRandomAgent will be hard to notice.

However, if we compare the ReflexAgent and RandomAgent in the openSearch layout where there are food everywhere, we notice that the ReflexAgent has a much better performance. Since if there is food near this pacman, it will go to it. This will be much faster than just randomly choose an action and hope it leads to the food in a food intense layout.

5)

His position: gameState.getPacmanPosition()

Legal Action: gameState.getLegalPacmanActions()

Successor state after the specified pacman move: generatePacmanSuccessor(action)

Get Agentstate: getPacmanState()

Get ghoststate: getGhostStates()

Ghost position: getGhostPosition()

Get current score: getScore()

Capsules position: getCapsules()

Food number: getNumFood()

Get food in layout: getFood()

Get walls in layout: getWalls()

Does a position have food: hasFood(x,y)

Does a position have wall: hasWall(x,y)

Is game lose: isLost()

Is game win: isWin()

6) Reflection:

In this project, from the changes of four agents we developed, we could see the different level of rationalities represented by these agents. The goal here we need to reach is letting pacman to eat all the foods. These four agents provide four different solutions.

In the first agent (DumbAgent), at the first place, we simply code it to go west and disregard the environment. However, this might lead to problem when you cannot go west anymore. It rarely solved any maze with this one action. As in the pacman.py, we are given function that can return the legal action that this pacman can take. In order to avoid pacman wasting time on stucking in the illegal action or one direction, we changed our code to create a agent(RandomAgent) that pacman could randomly choose action from legal actions. This new solution is more flexible than DumbAgent, but this agent did not performed well in solving our problem because by randomly choosing actions, most of time the pacman just hang around without move towards the food. Also these was a new problem, which was the STOP action is always legal and STOP is an useless action for pacman. In BetterRandomAgent, we removed the STOP action from legal actions to make our agent act more efficiently.

The last agent, ReflexAgent acts a little bit more rational than three other three agents. Besides randomly choose the next action like RandomAgent, when it detect a food is around itself, it will choose to move to the food and eat it. Otherwise it will randomly choose next action from illegal actions. This change allowed the agent avoid moving away from the food when it comes next to it, which makes the agent seems more rational. But it is still not always able to solve the problem. In openSearch maze, since the food is all around the maze, the pacman act not very bad with the leading of the foods. But in other maze which has only one food away from pacman, it revealed the same problem like RandomAgent.

From the performances of four agents, we observed that if we use more conditions for judging next corresponding action, the agent will act more rationally. It will be able to deal with more complex situation with more strategies of making decisions and act wisely.