Artificial Intelligence Course

Project 2: Multi-Agent Pacman

First name	Last name	Student number
Hao	**Ban**	**2591928**
Shengdong	**Wang**	**2616227**

Comments about the assignment (if you have)

Compared to the last time, the experience of this mission is more interesting. Sometimes, I can even find that the algorithm of design can make better decisions than me.

Question 1: Reflex Agent (4 points)

In Reflex Agent part, we need to define an evaluation function. According the description, the most important thing is finding a suitable value can represent GameStates.

At first, I found three variables can influence game states, including distance between pacman and position, distance between pacman and ghost and **scared times**. However, I am not sure the measure of scared times is correct.

According the hints in project description, I try to use the reciprocal of important values (such as distance to food). But we need to consider the situation when the distance between pacman and food is zero.

We need to consider the distance between pacman (newPos in code) and food or ghost (a function named ghost.getPosition). We can't consider all the effects of food and ghost. We can give priority to **the most dangerous ghost**, that is, the closest to us. And then, we use two values to measure effect of food distance. One is **min value of food distance**, another is **the number of current remaining food**. After experimenting with different methods, we found that considering the number current of remaining foods helps to improve the evaluation results.

After parameter adjustment and increasing the weight of the number of current remaining food on the results, we finally got a evaluation function.

Question 2: Minimax (5 points)

According to the description of getAction function, we define miniMax function and make **(gameState, agent, depth)** as input. Output should include each step's **cost and action.** Moreover, we found related information in initial function.

When the depth of minimax has **reached max depth** or it is **without legal action**, the game is end. In each round, we need to judge next agent is pacman or ghost. If the number agent is equal to getNumAgents(), it means we should come into next round.

Because a single search ply is considered to be one Pacman move and all the ghosts' responses, we need to distinguish the first round and others. We got the legal actions in getLegalActions function, and recursively called minimax function until the game is end. And then, **pacman is the max agent and ghost is the min agent** in our game. We will update previous path in result. That's why we need to save the cost of each step.

Finally, the function will return the actions as result. And we can find a path until the game is end.

In a conclusion, I found that minimax is a conservative approach, but not a optimal method. We always think ghost will give us max threat in each round. On the other hand, minimax method need to find all possibilities, which cost so much time. So, we need to use alpha-beta pruning to reduce the scale of tree.

Question 3: Alpha-Beta Pruning (5 points)

Actually, Alpha-Beta pruning function is similar with minimax function. The difference is that we don't need access some extra nodes which can't be accessed in the end. So, we add α and β in our function. In the code, A means alpha and B means beta. According the pseudo-code in description, **A is the max's best option** on path to root, and **B is the min's best option** on path to root.

Except (gameState,agent,depth), we also need to add **A and B as our input**. So, in each round, we need to get the value of A and B. A is max one from (result[0], a), and B is min one from (result[0], b). Implementation of A and B also used conservative approach.

For **pacman**, it need to find the max value. **If existing min result [0] is less than B** which finds the min one, pacman doesn't need to access nodes in B. For **ghosts**, they need to find the min value. **If existing max result [0] is bigger than A** which finds the max one, ghosts don't need to access nodes in A. Therefore, before accessing more nodes, we added a condition to judge it. It will save much time in most cases.

When calling Alpha-Beta pruning function, we added negative infinity, -float("inf"), and positive infinity, float("inf") as initial values.

Question 4: Expectimax (5 points)

Actually, Question 3 and 4 is similar with Question 2. It's an improvement in one area. As described in question 2, we choose ghost to take the optimal path in each time. The case is different with actual situation. Therefore, in this Question, all ghosts **will be modeled as choosing uniformly at random form their legal moves.**

In Expectimax function, the **possibility** that ghosts will go **is same** in different and legal **actions**. It is equal to 1/len (gameState.getLegalActions(agent)). Make sure when you compute your averages that you use floats. So, it should be 1.0/len (gameState.getLegalActions(agent)).

In the pacman round, we didn't change anything from Question 2. In the ghost round, we used the probability to multiply each value and add them up to the final result.

Question 5: Evaluation function (6 points)