COMP 7404 Computational Intelligence and Machine Learning

The University of Hong Kong

Department of Computer Science

3035348102 ZHANG Yupeng

Question 1:

Reflex Agent: In this question, I think the best way for the pacman is to minimize the distance to food and capsules. In the meantime, maximize the distance to ghosts. When it comes to end of the game, it will return the half of minimum value of ghostDist. Otherwise, it will return the previous value minus minimum value of foodDist. The ghostDist stores the Manhattan distance. Finally, I could get more than 1000 scores on average, however, there're one case that pacman died.

Question 2:

Minimax: In this question, I implement the minimax algorithm from the slides. There's one function named *maxValue*, which is called by pacman, while the other one is *minValue* function, which is called by ghost. When the *agentIndex* equals to 0, it means the current turn is pacman. Besides, considering more than 1 ghost, I also make the *minValue* function has an alternative case.

Question 3:

Alpha-Beta Pruning: This question is quite similar with the previous one, I implement the AlphaBetaAgent algorithm from the slides. It will prune the unnecessary branches, which helps machine reduce a lot computation work. Of course, it also considers the alternative case which has more than 1 ghost. Notice that in the first depth, which is *currentDepth* equals to 0, the pacman needs to select a direction, so it will return both *utilityValue* and *actions*. While in other cases, it just follows the value.

Question 4:

Expectimax: This question is quite similar with the second one, I implement it from the slides. The only different one is that ghost will choose the mean value rather than minimum value. So, I redesigned the function and named it *meanValue*.

Question 5:

A Better Evaluation Function: This question is quite similar with the first one, I use the same strategy and it works. Finally, I get more than 1000 points on average and no case that pacman died.

Question 6:

3-Board Misere Tic-Tac-Toe (Notakto): After reading the helpful paper "x-onlyttt", I understand that the p-position is the key point to solve this problem. The p-position types are as followed: {a, bb, cc, bc}. If a player can pick this p-position, he can always win the game. Therefore, our problem transforms to find the p-position. I designed the *evaluationFunction* method in the *TicTacToeAgent* class to compute and specify the types of board. What's more, there're only eight types, which includes {1, a, b, c, d, ab, ad, c2}.