Xinyi Ma [xim002@ucsd.edu](mailto:xim002@ucsd.edu)

Yijun Zhang [yiz160@ucsd.edu](mailto:yiz160@ucsd.edu)

Yuanchi Ha [yuha@ucsd.edu](mailto:yuha@ucsd.edu)

**HW4 Writeup**

**Algorithm:**

**P1: evaluationFunction**

**P2: MinimaxAgent**

In the minimax agent class, we defined function min\_value and max\_value separately. The recursion call is realized by a function called get\_value.

The get\_value function consider 2 cases. If the current index of the agent is 0, which means the current agent is Pacman, then we call the max\_value function. Otherwise, we call the min\_value function.

In min\_value function, we first define the value as infinite to update it with smaller value later. Then we consider two distinct conditions. If the game is won or lost already, we return the evaluationFunction(gameState) directly. Otherwise, we consider another two conditions. If the index of current agent is the same as the number of ghosts, which mean that we already evaluated all ghosts, then we call max\_value function with pacman’s index inside the loop; otherwise, we recursively call min\_value with next ghost’s index.

In max\_value function, the logic is smiliar, except that we define value as negative infinite to start with, so that we can update it with bigger value later. Moreover, after considering whether the game is win or lost, we don’t need to consider another two conditions. Instead, we call min\_value with the first ghost’s index to update the value inside the loop.

Outside these three functions, we have the main function to start the game. First, we get all the legal actions to loop with. We also define the result action as direction.stop and score as negative infinity to be updated later. Then we start the game by looping through all legal actions and update the scores each time by calling min\_value with the index of the first ghost inside inside a max. If the score is better than the previous one, then we update the result action with the new action too.

**P3: AlphaBetaAgent**

In the AlphaBeta agent class, like what we did in the minimax agent class, we defined function min\_value and max\_value separately and function get\_value to do the recursion. Moreover, we update alpha and beta and return the value of based on its relation with alpha and beta inside the min\_value and max\_value function after each recursion call.

The get\_value function consider 2 cases. If the current index of the agent is 0, which means the current agent is Pacman, then we call the max\_value function. Otherwise, we call the min\_value function.

Similar to min\_value function in minimax agent class, we first define the value as infinite to update it with smaller value later. Then we consider two distinct conditions. If the game is won or lost already, we return the evaluationFunction(gameState) directly. Otherwise, we consider another two conditions. If the index of current agent is the same as the number of ghosts, which mean that we already evaluated all ghosts, then we call max\_value function with pacman’s index inside the loop, then we check if the value is already smaller than the best of max(alpha), if so, we return the value; otherwise, we update beta. If the index of current agent is not the same as the number of ghosts, we recursively call min\_value with next ghost’s index, then check value and/or update beta.

The max\_value function is exactly the same as the one in minimax function.

Outside these three functions, we have the main function to start the game. First, we get all the legal actions to loop with. We also define the result action as direction.stop, score as negative infinity, alpha as negative infinity, and beta as infinity to be updated later. Then we start the game by looping through all legal actions and update the scores each time by calling min\_value with the index of the first ghost inside inside a max. If the score is better than the previous one, then we update the result action with the new action too. After that, if score is already bigger than min’s best option, we return the result already, otherwise, we update alpha.

**P4: ExpectimaxAgent**

Similar to the minimax agent class, we we defined function min\_value and max\_value separately and function get\_value to do the recursion. Besides that, we calculate the expected value inside the min\_value function.

The get\_value function consider 2 cases. If the current index of the agent is 0, which means the current agent is Pacman, then we call the max\_value function. Otherwise, we call the min\_value function.

In the min\_value function, we first define the value as 0 to update it with expected value later. We also calculate the probability based on the length of legal actions. If it is zero, the the probability is 0, otherwise, it is the reciprocal of the length of the legal actions. Then we consider two distinct conditions. If the game is won or lost already, we return the evaluationFunction(gameState) directly. Otherwise, we consider another two conditions. If the index of current agent is the same as the number of ghosts, which mean that we already evaluated all ghosts, then we call max\_value function with pacman’s index inside the loop, then update value by adding the product of the probability and the value to it. If the index of current agent is not the same as the number of ghosts, we recursively call min\_value with next ghost’s index, then update value in a similar way.

The max\_value function is exactly the same as the one in minimax function.

Just like the minimax function, outside these three functions, we have the main function to start the game. First, we get all the legal actions to loop with. We also define the result action as direction.stop and score as negative infinity to be updated later. Then we start the game by looping through all legal actions and update the scores each time by calling min\_value with the index of the first ghost inside inside a max. If the score is better than the previous one, then we update the result action with the new action as well.

**P5: betterEvaluationFunction**

**Performance on test data:**

**P1: evaluationFunction**

Average Score: 1215.6

Scores: 1377.0, 1060.0, 1193.0, 1115.0, 1232.0, 1186.0, 1206.0, 1380.0, 1224.0, 1183.0

Win Rate: 10/10 (1.00)

Time: 4mins,42secs

**P2: MinimaxAgent**

Average Score: 84.0

Scores: 84.0

Win Rate: 0/1 (0.00)

Time: 55secs

**P3: AlphaBetaAgent**

Average Score: 84.0

Scores: 84.0

Win Rate: 0/1 (0.00)

Time: 57secs

**P4: ExpectimaxAgent**

Average Score: 84.0

Scores: 84.0

Win Rate: 0/1 (0.00)

Time: 56secs

**P5: betterEvaluationFunction**

Average Score: 1309.3

Scores: 1365.0, 1374.0, 1319.0, 1333.0, 1379.0, 1101.0, 1364.0, 1292.0, 1383.0, 1183.0

Win Rate: 10/10 (1.00)

Time: 2mins,24secs

**Analysis of result:**