CSE 537- ARTIFICIAL INTELLIGENCE
PROJECT 2-REPORT

Following Files are submitted:
1. multiAgent.py: The modified (ReflexAgent, MinimaxAgent, AlphaBetaAgent) code.
2. output.txt: Sample outputs generated.
3. MiniMaxCount.txt: The number of nodes explored by MiniMaxAgent for each state.
4. AlphaBetaCount.txt: The number of nodes explored by AlphaBetaAgent for each state.

1. ReflexAgent:

The evaluation function is modified to make the Pacman play elegantly. Following two were considered:

(I). The distance of the Pacman from the food:
Pacman's distance (Manhattan distance) is calculated from all the food positions.
The distance which is less is given more score so that it picks the corresponding action as the next move to make.

(II). The distance of the Pacman from the ghost:
Pacman's distance from the ghost is calculated. If this distance is more, then higher score is given to it.

Observations:
(a). The Pacman played better, it made moves which will decrease the distance towards the food, that is, it will move towards the food.
(b). The Pacman was always escaping from the ghost by making moves which increase its distance from the ghost.
(c). For the **testClassic** board, Pacman won all the times. (Tested it by running the following commands:
python pacman.py -p ReflexAgent -l testClassic -n 10)

The total time for this execution was 29.4061 seconds. Each time it took around 3 seconds to complete.

2. Minimax Agent:

Minimax tree is generated to its entirety till the depth provided. The evaluation function is calculated when the depth is 0 or when the game is estimated to end. The values are propagated upwards.

A Min node(ghosts) will update its value only if the returned value is less . Similarly, the Max node will update its value only if the returned value is more.

Pacman will choose the maximum of all the values corresponding to each move and will make the move accordingly.

Observations:
(a). Since the tree is generated fully till the maximum, the time taken to complete the game is more

which is around 18.367 (Statistics are provided in *MiniMaxCount.txt* file) for the **smallClassic** board.
(b). The number of nodes expanded for each state is also provided in *MiniMaxCount.txt* file.

As it can be seen for the initial board configuration, the number of nodes expanded is 34 and for the next it is 160 and so on. This number is reduced by the pruning done using AlphaBeta Agent that is explained next.

3. AlphaBeta Agent:

AlphaBeta will not generate the tree completely till the depth. It will prune the tree based on the alpha and beta values which are updated at each Max and Min node respectively.

When the value of alpha is greater than or equal to beta at a particular node, its children will not be generated further, reducing the number of nodes in the memory and also the time for making decisions.

Pacman will choose the maximum alpha value and make the move accordingly.

Observations:
The results of running AlphaBeta agent on **smallClassic** board is provided in *AlphaBetaCount.txt* .

(a). Pruning the tree reduced the number of nodes expanded by a great number. For the initial configuration of **smallClassic** board, it expanded only 22 nodes, as opposed to Minimax which expanded 34.

Also, miniMax expanded 739 nodes, AlphaBeta agent expanded only 323 nodes. This is a huge difference ! This is one of the major optimizations.

(b). The time taken for alpha beta to complete the game is just around 8 seconds.

Optimizations:
1. The ReflexAgent evaluation function was very efficient for a single ghost, that it won all the times. It was efficiently moving away from the ghost and towards the food.
2. The AlphaBeta agent expanded very less nodes when compared to the Minimax nodes at each board configuration.

Discussion: (*Challenges*)

1. One issue that was found was with the ***getLegalActions*** function.
   If there are two ghosts like below:
   . . . G1 G2 . . .

   Ideally, the legal move for G1 should be only West but it returns both West and East.
   So, when it moves towards East, its getting overlapped.

Conclusion:

The AlphaBeta pruning algorithm is efficient when compared to other adversarial game algorithms as it reduces the number of nodes explored and can be increases the depth to which the nodes can be generated. Also, evaluation function is very crucial because, based on its values only, the Pacman's move is decided. So it needs to be efficient.