Artificial Intelligence Project 02: Multi-Agent Pac-Man

Q1: Reflex Agent

As it is mentioned in the instructions on the project page that "A capable reflex agent will have to consider both food locations and ghost locations to perform well.", hence our evaluation function evaluate a state based on how far Pacman is from food and how scared are the ghosts. We are also considering the score for the successor state where Pacman will reach after taking that action as the good move will result in high score for successor state. We are adding inverse of Manhattan distances of foods from Pacman position as lesser the distance, greater should be the score, hence taking inverse will give more score to position with lesser distance to the food. Also, one important thing that is being considered is to evaluate the STOP state, in case of stop state, if all the foods are already eaten (Pacman reached goal state) then we assign maximum value to this particular STOP action, otherwise this STOP action is assigned minimum score, so as to not let Pacman go to the STOP state.

Autograder score for 10 cases: Average Score: 1226.1

Total Time taken for 10 cases: 03 Seconds

Q2: Minimax

Minimax algorithm in case of Pacman works by constructing a game tree in which every Max node has all the legal moves of Pacman and the Min node has all the legal moves of respective ghost.

This tree is getting constructed till a certain depth as specified by self.depth. For every Pacman move, all the ghosts take move one by one. This complete step is equivalent to depth/ply 1.

When all the nodes are expanded till self.depth, evaluation function is called to return the score at that particular state and this score along with action/move is propagated upward till the root node(Max Node). Max node (Pacman) always selects the move with maximum score among its moves while the min node (ghost) always selects the move with minimum score. This propagation of score based on Min and Max criteria will give the best move that Pacman should take at that particular game state to increase its chances of winning.

Also one important point is, if there are no more legal states to be expanded for a particular node and tree is also not expanded till self.depth, then score evaluation function is called for that particular node.

We are also assuming that maxScore can be 10^100 and minScore can be –10^100. Python 'sys' min and max score can be also taken, but for that we have to import sys library.

Time taken(Autograder): 1 Second

Q3: Alpha-Beta Pruning

Alpha-Beta Pruning is more efficient version of Minimax algorithm. In this algorithm, alpha is assigned to Max node and Beta is assigned to Min node. We start by assigning minimum value to alpha (-10^100) and max value to beta (10^100). Alpha values are updated at Max node and Beta values are updated at Min nodes. For every node we check whether alpha > beta, if this condition turns out to be true, then we stop expanding that particular node because this is the best score that node can have. This condition helps in reducing the number of branches that are being expanded, hence in a given time, depth of the search tree built can be increased compared to Minimax Algorithm for given time.

For example, in one of the test case: Minimax generated this solution(total 31):

"A B C D E F G H I J K L M N O P a b1 b2 c1 c2 c3 c4 d1 d2 d3 d4 d5 d6 d7 d8"

Whereas Alpha-Beta pruning generated the below solution(total 19):

"A B C D E F I K a b1 b2 c1 c2 c3 d1 d2 d3 d5 d6"

Time taken(Autograder): 1 Second

Q4: Expectimax

ExpectiMax algorithm is similar to Minimax algorithm, but in case of ExpectiMax, we cannot assume that our Min agent will always take a move that will increase its chances of success. Hence, instead of returning a minimum value from Min node, we are returning average of all the moves that Min agent can take. Hence, adding randomness to the moves that this agent(ghost) will make.

Time taken(Autograder): 1 Second

In case of trappedClassic, AlphaBetaAgent will always loose because ghost will always take optimal move i.e. try to block the Pacman resulting in eating Pacman. But in case of Expectimax, ghosts move is random so in some cases he will move away from Pacman allowing Pacman to eat the available food.

Conclusion:

Minimax evaluation will always fare better when we won't have stochastic agent (i.e. when we will have Deterministic agent). However, Minimax agent will expand every branch even when it won't be necessary to do so. Hence, runtime of Minimax can be improved by Alpha-Beta pruning. However, Minimax and Alpha-Beta-Pruning won't work with a stochastic agent, hence, Minimax algorithm should be modified to take care of a random agent. This modified Minimax algorithm is ExpectiMax algorithm.