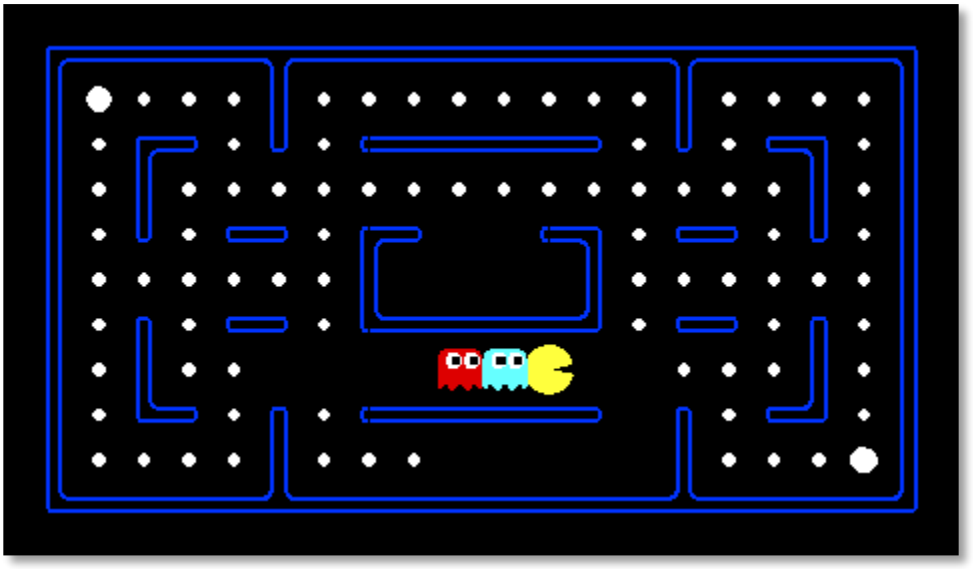
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ



Project 2 - Pacman

ARTIFICIAL INTELLIGENCE

Anna-Aikaterini Kavvada | NKUA | 01/12/2018

Table of Contents

Q1: Reflex Agent

In this question a simple evaluation function has been implemented. In this function we take into consideration both pacman's and ghosts' positions in the grid. For pacman we calculate its Manhattan distance from the foods available in the grid. If the distance of pacman from the food is 4 or less we add 1 point. If the distance is anywhere between 4 and 15 we add 0.2 points, since it is not such a good case as the previous one. If the distance is greater than 15 then we add 0.15.

In ghosts' case, we check if the ghost's new position is the same with the new position of our pacman agent. If so, that is the worst possible scenario, thus we decrease the points to avert this move. In another case, if the Manhattan distance between the pacman's new position and the ghost's is lesser than 3.5, the ghost is a close threat and we decrease by 1 point.

(The points given for each action were decided after multiple test trials, and these were the ones giving the best possible outcome for these restrictions.

(Also see the inline comments of the code)

Q2: Minimax

For this question, three functions were implemented:

- MiniMaxDecision(*self*, *gameState*, *agent*, *depth*)
- maxValue(*self*, *gameState*, *agent*, *depth*)
- minValue(*self*, *gameState*, *agent*, *depth*)

The function MiniMaxDecision(self, gameState, agent, depth), checks the terminal cases for the algorithm and is responsible for calling the appropriate function according to the agent that we are dealing with, pacman which is the max agent or a ghost which is a min agent. In case the agent is pacman (agent == 0 or agent == self.index), the function calls the maxValue() function, when the agent is a ghost (agent != 0) MiniMaxDecision() is calling minValue() function.

(Also see the inline comments of the code)

Q3: Alpha-Beta Pruning

The Alpha-Beta Pruning question, is almost the same as Q2. The difference is that we added values alpha and beta (a and b) in the functions we use, so that we can prune the branches of our search tree when needed. For instance, the declarations of the functions we had in Q2 now are:

- MiniMaxDecision(*self*, *gameState*, *agent*, *depth, a, b*)
- maxValue(*self*, *gameState*, *agent*, *depth, a, b*)
- minValue(*self*, *gameState*, *agent*, *depth, a, b*)

The other change made was adding a condition in the end of both maxValue() and minValue() functions, that checks for pruning. If there is no need to prune, function maxValue() updates a's value or if we are checking a MIN agent, a ghost, minValue() updates b's value. These updates take place, if only the prior if case was not true.

(Also see the inline comments of the code)

Q4: Expectimax

For this question, three functions were implemented:

- value(*self*, *gameState*, *agent*, *depth*)
- maxValue(*self*, *gameState*, *agent*, *depth*)
- expectimaxValue(*self*, *gameState*, *agent*, *depth*)

The value() function checks for the terminal cases of the algorithm and is responsible for calling the appropriate function according to the agent that we are dealing with. The only difference this value has with the MiniMaxDecision() in Q2, is that in the case where the agent is a ghost, we call function expectimaxValue, since in this question we are playing against an adversary, the ghost agents, that has a probabilistic behavior and make suboptimal choices.

The maxValue() function is the same as in Q2.

The expectimaxValue() function, is almost the same as th minValue() function in Q2. The difference is that here, we do not calculate the minimum value, but we calculate a probability value which takes into consideration the number of legal actions and the value of the new move. Then we add the probability value we just calculated to the one we had.

(Also see the inline comments of the code)

Q5: Evaluation Function

The evaluation function takes into consideration the distance of the food from the pacman agent, and the distance between the ghosts and the pacman agent. Also, we count the score, as a positive attribute, the case of existing power pellets. The function returns the negative value of the score for better results. (This happens because we give a great value to the state where a ghost is close to pacman thus boosting the score up high, whereas, when this does not happen and the pacman is getting victories the score is much lower).

The function scores 4/5 in the autograder.

(Also see the inline comments of the code)