Özgur Cem Birler
50118

I have implemented the evaluation function for Reflex Agent and better evaluation function by myself, but I couldn't do Minimax, AlphaBeta and Expectimax Agents, so they are taken from the internet:

Alpha Beta Agent:
https://github.com/opalkale/pacman-multiagent/blob/master/multiAgents.py

Minimax and Expectimax Agents:
https://github.com/romiphadte/AI-pacman/blob/master/multiAgents.py

Written Q1: What are the features you used in your evaluation function for your reflex agent? Why did you choose them? If you have too many, limit yourself to at most 5. Do you think using the reciprocals of some values is a good idea and why?

I have used position of foods and ghosts in the evaluation function.

I chose ghosts because Pacman loses the game if it is eaten by the ghosts. If the distance between Pacman and ghost is less than or equal to 1, the evaluation function is decreased by 99999 points which forces Pacman to be not close enough to ghosts. However, if the ghosts are scared, then it doesn't matter whether Pacman eats the ghost or not since eating a ghost doesn't provide any additional points.

I chose food because Pacman gains 9 points whenever it eats a dot, but since there are many dots we should assign a priority function for each food depending on the position of it and Pacman. In this case I decided that importance of a food should be inversely proportional with its distance to Pacman. The further the food is, the less important it is right now. So Pacman kind of acts greedy in moving to closest dots. For instance, if Pacman is 2 step closer to a food then when it takes a step, it will gain 10/1-10/2=5 points where as if its already 20 steps away, it will gain 10/19-10/20= 1/38 points.

Written Q2: Try the following lines of code: python pacman.py -p MinimaxAgent -l trickyClassic -a depth=2 -f python pacman.py -p AlphaBetaAgent -l trickyClassic -a depth=2 -f Run both them until 20 seconds (or less if pacman ends up dying) and see how far the pacman has got. You do not need to write additional time keeping code, a stopwatch should suffice. In your tests, were you able to see any speed difference between the MinimaxAgent and AlphaBetaAgent, between pacman actions? If so, why and if not why not? Is there any situation you came across that highlights this?

Yes, AlphaBetaAgent ate more food than MinimaxAgent in the first 20 seconds. Once the evaluation function returns a smaller value from all of the values of the previous branch, there is no need to check the other sibling actions in the current branch. Because Pacman has already had a better choice at the parent node of this (state, action), so the pruned (action, state) is not ever going to influence on the decision. With this idea, Pacman doesn't need to evaluate all of its possible successor states, thus it will perform less calculations and decide faster.

Written Q3: When you were running the tests in the previous question, did your pacman behave exactly in both cases? Why?

Yes, because because AlphaBetaAgent still uses the idea in Minimax but it doesn't iterate through all of the state action pairs. So it will choose the same actions since we use the same evaluation function, have the same possible states and same heuristic to play (search through the actions and states and move in a way that, at the end reach to the maximum value of the evaluation function, while assuming ghosts are trying to minimize the value). AlphaBeta pruning just increases the speed with eliminating the states that Pacman won't go beforehand, it doesn't provide different strategy for moving.

Written Q4: Now try the same with the ExpectimaxAgent; python pacman.py -p ExpectimaxAgent -l trickyClassic -a depth=2 -f Comment on how fast your code runs. Compare it with the MinimaxAgent and AlphaBetaAgent. Note that this comparison is trickier to do. If you are not able to conclusively see anything, write what you would have expected.

Expectimax Agent runs in similar speed with Minimax Agent and it is slower compared to AlphaBetaAgent. I would expect that Expectimax to be slower than AlphaBetaAgent because it doesn't prune some states so it still checks the all of the possible moves as in Minimax Agent. It is only different by assuming that ghosts don't move in a way that minimizes the score.

Written Q5: We are sure that you were able to write a better evaluation function than the one we used for the programming questions 2-4. Did you change anything from your evaluation function for the ReflexAgent? If so, what were the changes? What, if anything, is different in this case? If you have written something entirely different, comment on your new evaluation function

In the better evaluation function, I realized that the in the states the position of the foods, and the number of remaining of the food is more important than ghosts, because the ultimate aim is to eat all of the dots in the shortest time. Neither eating ghosts nor capsules give any additional point, instead this time my evaluation function uses only the number of remaining foods and the distance to foods. Selection of weights for these parameters are discussed in the next question.

Written Q6: For both the programming questions 1 and 5, you probably needed to tune your feature weights. If so, comment on how you selected your weights, what did you prioritize and why.

In question 5, I prioritized the weights of the number remaining foods more than the sum of the distances to each food because the ultimate goal is to eat all of the dots. Assume an instance where a cluster of dots nearby Pacman and a single dot very far away, in this case Pacman should move towards to the cluster instead of going to the furthest distance. I assigned a unit (one) weight to distance for each food and then tried to find the weight of the remainingFoodNumber feature by running the 5th question many times using different weights. At the end, I found that weight of 8 satisfies the requirements (that Pacman wins in 10 test cases and average score is more than 1000) Though I can't reason well why it is 8 instead of 7 or 9.

For question 1, the weight of each dot is inversely proportional with its distance to Pacman, as discussed in Q1 and I have decided to give extra 9 points for each eaten dot, as this is the same score gain in the actual play.