Ryan Brooks
CS4300
3/20/2021

3.1 Reflex Agent

1. What feature (or features) did you use for your evaluation function?

   I just used the score of the state summed with the inverse of the distance to the closest dot. This was to incentivize Pacman to increase his score and head towards any dots that he was far away from. I later expanded upon and improved this simple evaluation function in the last code section by adding many additional metrics and reweighting existing ones.

3.2 Minimax

1. When Pacman believes that his death is unavoidable, he will try to end the game as soon as possible because of the constant penalty for living. Give an explanation as to why the Pacman rushes to the closest ghost in this case?

   Pacman assumes that the ghosts are acting optimally when we use minimax. Consequently, he knows for certain under this assumption that the ghosts will kill him in a certain amount of turns. Let us denote this number of turns as x. If Pacman loses 1 point for every turn, we can express the number of points he finishes the game with as p-x*1-500 = p-x-500 where p is the current number of points he has. Thus, to maximize his points he must minimize x, the amount of turns he lives for. This is why he rushes the ghost in this no-win scenario.

3.4 Expectimax

1. You should find that your ExpectimaxAgent wins about half the time, while your AlphaBetaAgent always loses. Explain why the behavior here differs from the minimax case.

   We know from the previous question that with minimax Pacman will rush the ghosts in a no-win scenario to maximize the number of points he ends with. However, this operates under the assumption that the ghosts are acting optimally. expectimax, on the other hand, assumes that the ghosts are acting randomly and that there is consequently a chance that they will not act optimally. Thus, unlike with minimax, expictimax Pacman will usually take more risks and in scenarios where the ghosts are acting sub-optimally, this can pay off. Ultimately, it is the case that in the test case given for this problem the ghosts are acting sub-optimally. Thus, the AlphaBetaAgent, by using minimax, unnecessarily kills itself prematurely while the ExpectimaxAgent has a chance of success being able to win roughly half of the time.

3.5 Evaluation Function

1. What features did you use for your new evaluation function?

   Firstly I took into account whether this was a terminal state. If the state was a winning state, the returned value was infinity, implying it to be the best state. If the state was a losing state, the value was negative infinity, implying it to be the worst state.

   Otherwise, I returned a value that comprised of a summation of many factors, each with their own weighting. The summed factors and their weights are as follows:

A. The score of the state. The higher a score is, the better the state. Thus this was multiplied by a weight of five.

B. The inverse of the distance to the closest inedible ghost if there was one. This was multiplied by a weight of -10. Subtracting this ensures that I maximized the inverse distance between Pacman and the closest ghost or in other words minimized the distance.

C. The inverse of the distance to the closest edible ghost if there was one. This was multiplied by a total weight of 20. I wanted Pacman to go towards edible ghosts, so I added to the value of a state more if Pacman was closer to the edible ghost. Thus he would approach and eat edible ghosts to acquire points. This was then multiplied by a factor related to the number of ghosts (G), G^-.5 to make it a less impactful metric when more ghosts are present.

D. The inverse of the distance to the closest dot. This was given a weight of 7. I once again took take the inverse of this distance. This time I left its multiplier positive to entice Pacman to approach the closest dot instead of running away like in the case of the inedible ghosts. This was once again multiplied by a factor of G^-.5 to make it less impactful in the case of more ghosts.

E. The number of remaining dots. This was weighted by -35. I used this value to penalize Pacman for having more dots on the screen, thus giving him bonuses as he cleared the screen of dots.

F. The number of remaining capsules (big dots). This was weighted by -500. Pacman needed to be incentivized to eat these if he was close to them. By eating the big capsules he gain points as well as the opportunity to earn more by eating ghosts.

The combination of these metrics ultimately found the following value for each non-terminal state:

value = 5(A) + -10(B)*G^(-.5) + 20(C)*G^(-.5) + 7(D) + -35(E) + -500(F)

5 Self Analysis

1. What was the hardest part of the assignment for you?

   The hardest part of this assignment for we was trying to fix a major issue with my better evaluation function. The issue actually persisted into the final version of my code even after spending hours trying to fix it. The issue arose when Pacman would become adjacent to the final dot on the board. In my evaluation function, when a state was a winning state I would return float('inf') (infinity) to show that it was the best possible state. Despite this, after Pacman approached the final pellet he would sit there until the ghost "forced him" into it. I couldn't figure out why this value did not entice Pacman to eat the final dot without the ghost's "help" despite the amount of time I sent on trying to fix it. Ultimately, while it did not impact my test cases greatly, it did get on my nerves.

2. What was the easiest part of the assignment for you?

   The initial reflex agent's evaluation function's implementation was the easiest part of the assignment for me. I tried a very simple solution which ended up working immediately. Even if my initial idea hadn't been good enough I still had more refined ideas for other implementations in my head. Thus, this was extremely trivial compared to the rest of the code questions.

3. What problem(s) helped further your understanding of the course material?

   I would say that implementing minimax, expectimax, alpha-beta pruning, and the better evaluation function really helped my understanding of the material. It was difficult at first to wrap my head around how to implement these functions even though I understood them so well. Over time, however it got easier to translate my understanding of them into practical code that worked with the existing model. The biggest difficulty I overcame in this was figuring out how to output my recommended action from the alpha-beta pruning minimax calculations. This too both reinforced my practical knowledge of the course material and allowed me to translate it into a practical, real-world type of setting.

4. Did you feel any problems were tedious and not helpful to your understanding of the material?

   I felt like the implementation of the initial evaluation function for the reflex agent was too easy to really be helpful. But it was also to short and simplistic to be tedious so it really wasn't a problem.

5. What other feedback do you have about this homework?

   I think this homework was good practice for implementing adversarial search functions, thus I have no other feedback.