**YIGITHAN SANLIK 64117**

**COMP341 Project 2 Report**

**Written Q1:**

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

I have used newPos, newFood and newGhostStates features that are already defined in the project’s starter code. These three features are all implemented on successorGameState which means that newPos gives the pacman’s position, newFood gives the all food positions and newGhost states gives the states of all ghosts in the game’s successor state. I have also used getGhostPosition() to find each ghost’s position by looping through ghost states given by newGhostStates.

In my evaluation function, I find the distance of the closest food and closest ghost from the pacman’s position in the successor game state. Then, I used this values to calculate the successor game state’s score. While calculating this score, I use food distance as a positive value and ghost distance as a negative value. Using closest ghost’s distance to the pacman as a negative value is a good idea because pacman should be as far as possible to the ghosts and being closer to the ghosts should affect the score in a negative way.

**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 dieing) 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?**

Running both algorithms 20 seconds, I have results of that MinimaxAgent got 143 points and AlphaBetaAgent got 715 points. By this results, we can say that AlphaBetaAgent is more efficient than MinimaxAgent in this environment with an equal time of 20 seconds. Another result that I have observe that the first 6 seconds of the run were exactly the same, but AlphaBetaAgent increased its point gaining pace. These difference derives from the implementation of these two algorithms. The main difference of these algorithms that we use value of v to find the max and min values in MinimaxAgent while we are using alpha and beta values to calculate max and min values instead of using a single v value. This difference makes AlphaBetaAgent to have a lower complexity. As a result, AlphaBetaAgent has a better performance in the long run.

**Written Q3:**

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

Pacman behaved exactly the same in both runs but in AlphaBetaAgent pacman moved faster after 6th second of the run. The main reason of this is that basically both of our algorithms have the same logic and calculates the same values. However, we use two variables which are alpha and beta in AlphaBetaAgent while using a single v value in MinimaxAgent in the implementation of max-value and min-value functions. These two implementation styles causes complexity difference and this complexity difference causes time efficiency performance.

**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.**

After runing the ExpectimaxAgent algorithm multiple times, I have observed that the result of this runs were not exactly the same like it was in the MinimaxAgent and AlphaBetaAgent algorithms. This is because of that we use an expect-value function instead of min-value. This expect-value function gives a randomness to the algorithm which ends up with various results for the same run.

Analyzing in the context of time efficiency, ExpectimaxAgent has a closer performance to the MinimaxAgent. I think this is because that ExpectimaxAgent uses also a single v value like MinimaxAgent. Using alpha and beta values, AlphaBetaAgent gives better performance results.

**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.**

I have used the same evaluation function with a small change. At the first question, my evaluation function was using successor game state’s values. At fifth question, I have used current game state instead of succesor’s game state. Using scared ghost states may develop my evaluation function.

**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**

I did not change my evaluation function except state choice, so I do not have differences in the context of feature weights.