CSE 573 Final Project: Pacman Capture the Flag

Tanner Fiez Liyuan Zheng

March 15th, 2017

1 Overview

We designed two separate agents for the game. Each of these agents used a base class which used exact inference with time elapse. This was implemented in a very similar way to problems 1 and 2 in the homework 5. The HMM forward algorithm was used to update beliefs based off the noisy distance readings that our agents received. With time elapse we were able to use information about the legal moves an enemy agent could make to update the distribution over new positions that the enemy agent could be in. In order to choose actions, we used the Expectimax algorithm in a very similar way as in homework 2. By assuming that each enemy agent was in its most likely position based on our beliefs, we could run the algorithm as if we had full information. Due to the time constraints we were only able to run the algorithm to depth 2. We found that an aggressive approach yielded the best results. For this reason we designed one agent to be entirely offensive focused, by using an evaluation function that forced it to focus on getting into enemy territory without ever chasing enemy Pacman. Our second agent type was more defensive focused based off of its evaluation function, yet when the opportunity was right, we designed it to act in a similar way to our offensive focused agent. This joint design allowed us to balance the needs to defend our territory when needed but also take advantage of the chance to have both agents attack when our territory was not at risk. As is often the case in learning environments, we found the best way to improve our performance was to attempt to understand what caused our failure and success cases, and based off of this, modify our design.

2 Filtering Modifications

We used two modifications to our filtering approach using information that was available to us in addition to noisy distance readings that gave us significantly better inference. While the noisy distance measurements would give us many possible positions an enemy could be in, often several of these positions could be ruled out. The first modification was to check whether a possible position from a noisy distance measurement, was in agreement with the mode the enemy agent was in. For example, if the noisy reading indicated to us that the enemy agent was in a position that would make it a Pacman, we could check the mode of the enemy agent using the gameState.getAgentState().isPacman function, and if these were not in agreement, rule out that position and make its belief be zero in our update. The second modification was to check whether a possible position from a noisy distance measurement was in our sight range. If this was the case, we could rule out the position knowing that if in fact the enemy agent was in the position, we would have been able to get an exact reading to begin with, so the agent could not be in that position and the belief for the position could be updated to be zero.

3 Offensive Agent

As described previously, our offensive agent utilized our base class in which we had implemented our filtering algorithms to estimate position of enemies, and chose a move using the Expectimax algorithm. We found it to be important to use some notion of risk vs. reward for our offensive agent evaluation function. While in enemy territory, if an enemy agent was close, our first priority would always be to not get eaten. From this point we found that it was important at some point to focus fully on returning the food we had eaten to our side safely to recoup the points, instead of continuing to chase after food. After iterating on the possible

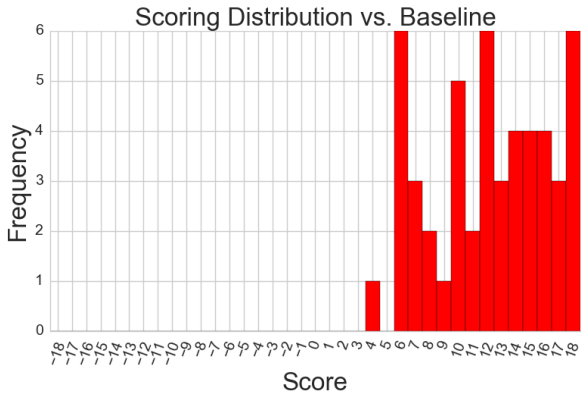
ways to do this, we ended up setting the number of food our agent would hold before turning to retreat to its own territory based on what the score of the game was. If our team was leading by 6 points or less we set the carry limit to be 6. If our team was leading by more than 6 points we set the carry limit to be 4. When the power pill was active we ignored these limits because our agent had far less risk. The intuition for these choices came from two factors. First, our offensive agent evaluation function, which the defensive agent also used when it was in attack mode, put a high priority of eating the power pill. When we did get the power pill we often were able to get many of the enemy food, and in particular, these were typically the easiest foods to capture because of their position. The second factor was that the food is much harder to get when our team had a higher score because the remaining food are in more difficult positions to reach and the enemy is unlikely to be disabled since to get the high score we likely did get the power pill. For these reasons we found a more conservative approach with a higher score to yield improved results. We found returning food to be a fundamentally harder task than defending a space. This is why we found having the fully offensive agent not ever worry about defending its own territory not to be a problem, but the ability of the defensive focused agent to be able to attack was very helpful since it is much easier to return food with two attackers.

4 Defensive Agent

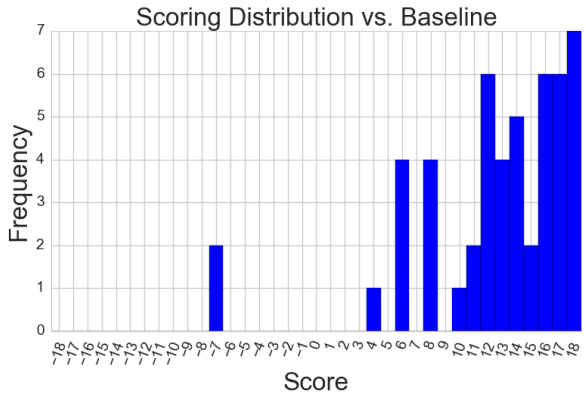
Like our offensive agent, our defensive focused agent utilized a base class which takes care of filtering for estimating position of enemies and moves are chosen using the Expectimax algorithm. The evaluation function for this agent is designed to balance the central priority of protecting our teams territory while also attacking when the opportunity is available. This is done by having an evaluation function that switches depending on whether an enemy has a Pacman, meaning that the enemy is on attack. If this is the case, the evaluation is simple in that it will choose the action that will minimize the distance to the closest attacking agent. If however, the enemy has no agents attacking we want to be able to use the agent to attack. When this is the case, we use the evaluation function that we designed for the offensive agent with one slight modification. This is that we do not decide to retreat based on the number of pellets the agent has collected. Instead we let the agent collect as many pellets as it can without switching to an evaluation function that is focused on returning to our territory. We found this to work well since we want to stay on the attack until we absolutely have to defend. In our simulations, rarely did this create large losses since we could typically recover and chase down the enemy agent.

5 Results

We tested our agents extensively against the baseline agents when trying to improve our evaluation functions. Because when updating the distribution of the opposing agents we assume they move randomly, we expect our methods to be scalable to different agents others have designed in the class for the tournament. The distribution of results for playing 50 games with our final agents versus the baseline as the red and blue team respectively are given below. Over 50 games as the red team, our agent beat the baseline team 100% of the time with an average score of 12.16. Over 50 games as the blue team, our agent beat the baseline team 96% of the time with an average score of 12.52.



(a) Playing as Red Team



(b) Playing as Blue Team

6 Work Distribution

We began the work by discussing possible approaches before settling on using the Expectimax algorithm with exact inference and time elapse. Since we worked together for the assignments, we were able to use a good amount of code that we had written for these previously with some modifications. We then spent a good portion of our time on the project working on the evaluation functions together by discussing strategy, trying new formulations before watching games to see where we were succeeding and failing as well as getting distributions of results. We contributed equally to this project and actively worked together on it.