Team Members:
Rachel Chen
Sri Boinapalli

Sriracha Team Write-Up

We designed two Pacman agents that played capture the flag against another team of two in a maze. The four agents on the maze are either blue or red. Blue agents on the blue team try to eat dots from the red side/team while trying to defend dots on the blue side and vice versa for red agents. An agent is a ghost while on their own team's side of the maze and becomes a Pacman when it crosses over to the opposing territory. The agent returns to its starting position when eaten by an opposing ghost. To defend our own team's dots and eat the opposing team's dots at the same time, we made a defensive and an offensive agent. The defensive agent protects our team's dots and chases Pacman while the offensive agent seeks the other team's dots and avoids opposing ghosts.

For each agent, we calculated the dot product of the feature and weight vectors for each possible action that the Pacman can take. We wrote functions getFeatures(self, gameState, action) and getWeights(self, gameState, action). The feature vector calculates each of the parameters such as 'ghost distance' or 'food distance'. The weight vector returns the weight of each feature. The measure of the dot product of the two vectors makes the agents take the best action that maximizes the value of the dot product.

The Defensive Reflex Agent class has its own unique feature and weight vectors. The feature vector has parameters numInvaders, invaderDistance, nearCluster, and foodDanger. They measure the number of Pacman invaders, distance to nearest invader, distance to biggest cluster of our dots, and closest invader to dot distance respectively. The game state score goes up if the defensive agent is closer to a Pacman. The score goes down, however, with more invaders on our side and when the Pacman is closer to a certain dot. Our most successful strategy was making our ghost go towards the biggest cluster of dots. The nearCluster parameter measures the sum of all the distances from the ghost to the dots on our team. The smaller the sum, the higher the score became. Because our ghost was camped out at the most populated area of dots, it was located right where the Pacman tended to go. Our ghost would easily find the other team's Pacman and eat it before most of our dots were eaten.

The Offensive Reflex agent class, also had unique feature and weight vectors. The feature vector in the Offensive Reflex agent class has parameters goCatch, distanceToFood, and guardDistance. These parameters measure the distance between a Pacman and an enemy ghost and tells Pacman to react to the ghost, the distance between Pacman and food, and the location and distance the guards (enemy ghosts) are from Pacman, respectively. Other ideas we had for improving our offensive agent was to have our Pacman cross into enemy territory and to go toward the nearest large cluster of food first instead of heading toward the nearest food. However, when we implemented this into our code and ran it over one hundred iterations, we noticed a greater number of wins for the baseline agent than we did before the implementation of the new code. After watching one iteration of our agent versus the baseline agent, we noticed that our Pacman was not focused on eating any certain cluster of food and seemed confused about its next steps to take. Our best strategy that worked was to focus on our Pacman agent's distance to enemy ghosts and distance to food in order to optimize the amount of food eaten.

Our solution to capture the flag, ended up having an excellent defense system and slightly above average offense. For defense, we managed to grasp the main problem we needed to solve: to defend as many dots as possible while eating a Pacman that comes near. This prevented the main cause for losing, letting the other team eat most of our dots before we ate most of theirs.

Our offense was less exceptional, because we had trouble eating most of the other team’s dots while running away from the opposing ghost. Our offensive agent only ran away from the ghost when it was too close to a ghost and had no effective way of avoiding ghosts in general.

This project was very enjoyable and we learned a lot as a team and about the material of this course. This project was challenging but rewarding in the end and was interesting to see how we needed to implement different concepts we learned in this class. Both of us read over the different files we needed to design our agents and became quite skilled at understanding and managing feature and weight vectors so they can influence the score and the gamestate. By figuring out the balance between feature and weight vectors, we understood and worked out how to get our agents to work in sync instead of acting like two independent agents. Finally, we learned how to work with a partner using materials learned from this class. We learned how to properly communicate ideas so that the other person would understand and so both of us would be on the same page in what we wanted to accomplish.