Chuangbo Tong, Haofan Wang

CSE140

06 December 2019

Written Report

Through this Pacman tournament, we are required to come up with our own Pacman strategy so that our Pacman agents could both defend our own food and eat opponents' food when we are competing with other teams' Pacman in class. During the process of coming up with "smart" ideas of arranging the strategies, we wanted to let both agents having the ability of changing defensive and offensive modes. However, due to the fact of createTeam() function only being able to specifically calling 2 classes of agents (defensive agent and offensive agent). So we gave up this idea, and when we later tried to let offensive agents defend based on certain circumstances, we found out that just by changing the weights of certain features would achieve what we wanted.

For an offensive agent, we wanted our agent to chase the nearest food whenever possible, but if the opposing Pacman was approaching, we would put escape as the first priority, and during that process, it would also try to get the capsule. To achieve this goal (or solve this problem), we chose q-learning as the main template for the offensive agent, but the final product was q-learning without learning. In the very beginning, we model the problem by implementing q-learning and evaluation functions in adversarial search, because we thought that through q-learning, our Pacman would be able to detect the dead-ends by learning, as we had done in P3, and to choose an appropriate action under different circumstances by evaluation functions. However, for the q-learning part, we were unable to train our Pacman to learn with q-learning, because even after we set up discount rate, learning rate, and epsilon, when we combined all the elements, including scores, together to train (or evaluate for) our q-value with 100 game tests, we kept getting zeros. After a long time debugging and ended up with no luck, we decided to change the algorithm to Alpha-Beta pruning, while using these q-values as evaluation functions. In our opinion, the product of features and weights, which was our q-value before, resembled the evaluation functions that we had done in P2, so all functions to compute q-values before were kept. Again, some difficulties were met, as the game would run into infinite loops. We returned

to the initial q-learning by putting more focus on giving different weights in various circumstances, because without learning, this final algorithm was basically assigning values computed by q-value (which in our case is essentially evaluation functions) to every position. To make our Pacman perform better at this point, it would be a matter of adding good features and setting appropriate weights. At the end of the unit test, we found out that eating capsules can earn lots of points since no enemy was chasing us. So we added a eat capsules feature. When Pacman was 1 unit away from the capsule, the Pacman would eat it no matter what. That was how we modeled the problem and how we solved the obstacles along the way. For the part of choosing weights, we underwent a lot of manual testing to see whether agents would choose a better path than before. We made eating food & capsules relatively big compared to other features so that the agent would intend to eat more, while others negative, for example, distance-to-enemy. The closer it gets to ghosts, the faster the score would decrease, so it would encourage ghosts to prefer running away than eating food. The drawback of this was that we were unable to prevent ghost going into a dead end, but we believed there would be a feature that could handle this case.

On the defensive agent part, we didn't implement any specific algorithms but only making agents going towards the center of the map so that it would have relatively equal distances to each incoming Pacman. Our initial goal, however, was to mark the middle of the map as the point where the defensive agent would always return to, and meanwhile maintaining the same level (same y-coordinate) as the closest opponent. For some reason, which we were unable to solve, our agent would only choose either one approach, but not both. With maintaining the same level as opponents, it indeed covered up some drawbacks of staying at the center of the map for some other maps because of how the layout was designed, but for most layouts this method alone was shown to be not sufficient enough after competing with other teams, so we eventually chose the method that stayed at central of map. Some obstacles were encountered: we found out that our agent would crash on some random map against other players. The reason was that the center point sometimes was a wall, which was illegal for Pacman to reach there; therefore, we first checked the point if it was a wall or not, and if it was, Pacman would go up 1 unit, and if reached the top of the map, Pacman went down. Eventually,

the Pacman would found a legal coordinate to stay. After we improved our method, the win rate increased a lot.

Throughout this project, we learned a lot of programming skills and teamwork strategies. We have very little experience of python before taking this class. We learned a lot of syntax of python and it was a very fun experience because it was the easiest language we learned in college. Also, we learned how to use weights and features to change the Pacman's behavior. During the coding experience, as a partner, we found out that communication with each other was a key point to avoid doing wasteful work. For instance, implementing a method or changed something in the code without telling the other partner that could potentially lead to time-consuming problems. This almost happened, but we successfully prevented it. We found out that making a plan before coding is very important, especially for big assignments like this one. It helped us code with a goal instead of trying to code this way and see if it works. These two ways of teamwork saved us a lot of time. It was a fun project, and a precious learning experience for both of us.