Project Name: The Good Guy

Approximate Q Learning with Linear Function Approximation/Boltzmann Exploration Optimization/A* Search

By Gurkirat Singh
A11593827

Youtube: https://youtu.be/QyykKAVv_X0
Github: https://github.com/endeavors/Approximate-Q-Learning

The goal of this project was to build an AI game based on Approximate Q-Learning, which uses features and weights to describe how good a particular action in a state is. Many of the concepts used in this project were based on my own independent research, hence, they were not taught in class.

To begin with, the red pentagon serves as a monster that is trying to capture the green dot (which is the "Good Guy") and it is the green dot's responsibility to free the prisoners captured behind the black wall without getting caught by the monster(s). I wanted the green dot to figure by itself how to get to the prisoners and I succeeded in doing that by using Approximate Q-Learning coupled with some great optimizations. Instead of calculating exact Q-values, we now approximate the Q-values and assign weights to features on the basis of how likely does a particular state represent the feature that we have chosen. This is called linear function approximation.

I built the GUI using Turtle Graphics framework and made sure that the GUI was a completely different entity from the Q-Learning classes and that the Q-Learning classes controlled what the GUI displayed, which is very much like a Model-View controller design pattern. I also made sure that the training of the game could be run without the GUI to make the computation and the process faster. I used the Boltzmann's exploration optimization to decrease my probability of exploration vs. exploitation as more episodes were completed. This inherently allowed me to add noise to the actions I could take from a particular state and fail early on in the game so that I could exploit from my failures later on in the game. I was able to systematically reduce the temperature (T) by using the process of simulated annealing, which eventually gave the probability of choosing an action *a* given some state *s*. I then sampled over the probability to choose my best action.

There are two kinds of features that I used in the game. One is the Manhattan distance to the reward (which are the prisoners) and the other is the number of ghosts that are going to attack the green dot in less than 2 steps. I then update my current weights based on the state I'm

currently in and the action that I have chosen to take. The following equation is the one that I use to compute the weight that correspond to each feature.

$$w_i \mathrel{+}= \alpha\big([r + \gamma * max_{a'} Q(s', a')] - Q(s, a)\big) * f_i(s, a)$$

Furthermore, I added path finding algorithm A* to my game just to optimize the way the green dot chose its path on the grid. A* simply taking the Manhattan distance from the green dot to the reward would exclude the positions of the walls if there are any in between. Hence, A* helps finding the shortest path *around* walls to help the green dot calculate the Manhattan distance even when walls are present. On a very large grid space, doing A* on every possible state would be very expensive and would defeat the purpose of Approximate Q-Learning, which is suppose to be faster. Hence, I only use this A* optimization when the green dot senses that it is near a wall (Example shown in video).

The monsters are able to find where the green dot is by also using Manhattan distance; however, simply using Manhattan distance would make the Monster(s) over powerful than the green dot. Therefore, I let the Monster attack the green dot provided some probability of attack specified by the user. Overly, the output of the game was extremely impressive. The green dot was able to learn gradually from its mistakes and hence was far better in releasing the prisoners as more and more training episodes were done (Example showed in the video). However, as the grid size grew, it became slower for the green dot to effectively calculate which actions were best in a state. This is because many times it encountered states that it had never seen before and continued to explore further on such states, which subsequently led to bad guesses. The game plays itself 10 times after training for a default of 500 episodes (which can be changed). The green dot was able to determine which strategies worked in evading the monsters without even specifying what strategy to use. Plus, the green dot was placed randomly (and so were the monsters) on the grid; hence, not only did it have to evade the monsters but also be able to get to the prisoners from any given state on the grid. This is the power of AI! ☺