CS 440 MP1

Thomas Nelson

September 27, 2014

Problem #1

(a) If epsilon is set to 0.0, then the agent does not score any points at all. Instead, it just goes with its default action, which is to move north. It then gets stuck in a corner because it keeps trying to move north. Because epsilon is zero, the agent never tries anything other than what it thinks is the best action. This leads to the agent never learning if the other options are better or not. This is a very good example as to why some randomness is needed in choosing which action to take in a QLearning Algorithm.

(b) This time, the agent performed much better. The average reward over 1000 episodes was 133.3595. The reason the agent performed so well this time is that with the randomness of the epsilon, the agent explores more options. Since it explores more options, the agent has more data to calculate which action is the best action in each state, instead of just going north all the time.

(c) When the value of epsilon was changed to 0.5, the average reward decreases quite drastically. The average reward over 1000 episodes was -28.079. What went wrong here is that the agent was too willing to explore, and even though it probably had enough data to make a pretty good choice, half of the time it took a random action. So instead of avoiding the states where it has recieved negative rewards, the agent might just move right into them.

Problem #2

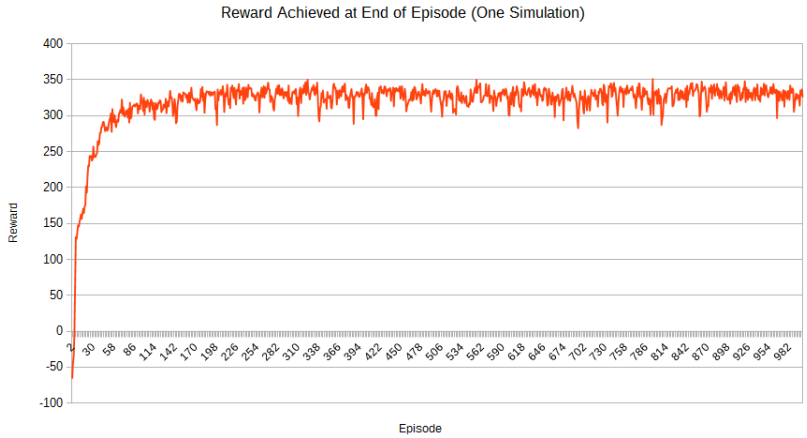
(a) When the agent does not know where the thief is, performance is terrible. The average reward over 1000 episodes was -12.8125. Since the agent does not know where the thief is, it cannot accurately predict whether or not a given action will cause it to run into the thief. This leads to it running into the thief several times, making its reward negative.

(b) When the agent does know where the thief is, the performance sees a substantial improvement. The average reward over 1000 episodes in this case was 273.1885. Obviously, since the agent now knows where the thief is, it can learn to not take actions that would cause it to run into the thief. Avoiding the thief means less penalties, which means a higher total reward.

(c) In order to search for the best learning rate and epsilon values, I used a form of binary search. I would pick two values, and then run the simulation for 10000 steps and 1000 episodes for each value. The simulation would be run for each value 3 times, and the average reward for all 3000 episodes would be calculated for comparison. The midpoint between the two values would then be tested, and the test would be recursed using the midpoint and the value that scored the highest. The number of recursions were limited to cut down on running time. The test for epsilon was done first with rate = 0.1 and then the test for rate was done with the value of epsilon found in the first test. The values that were found to perform the best were rate=0.225 and epsilon=0.0095.

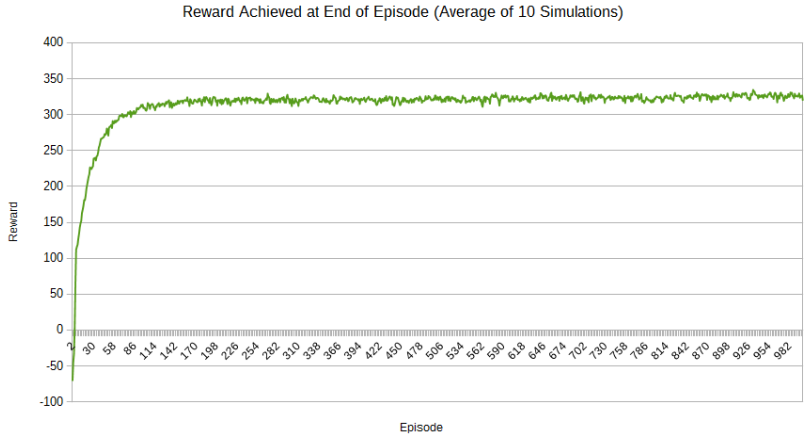
Problem #3

Here is a plot of the reward achieved at the end of each episode for one simulation:



The reward starts out under zero but quickly grows to above 300. After that, there is not much improvement. Also, due to the random aspect of the execution of the simulation, the plot has a lot of peaks and troughs, where the episodes had unusually high reward and unusually low rewards.

Here is a plot of the reward achieved at the end of each episode averaged over ten simulations:



This plot has the same general shape as the plot of one simulation, however this plot doesn't have peaks and troughs that are as large as the ones in the other plot. This is due to the averaging process "smoothing out" the outliers. Since this plot is a lot smoother and doesn't have quite as much noise, we can now see that the agent continues to improve throughout the entire simulation. However, that improvement is very slight after the initial large improvement.