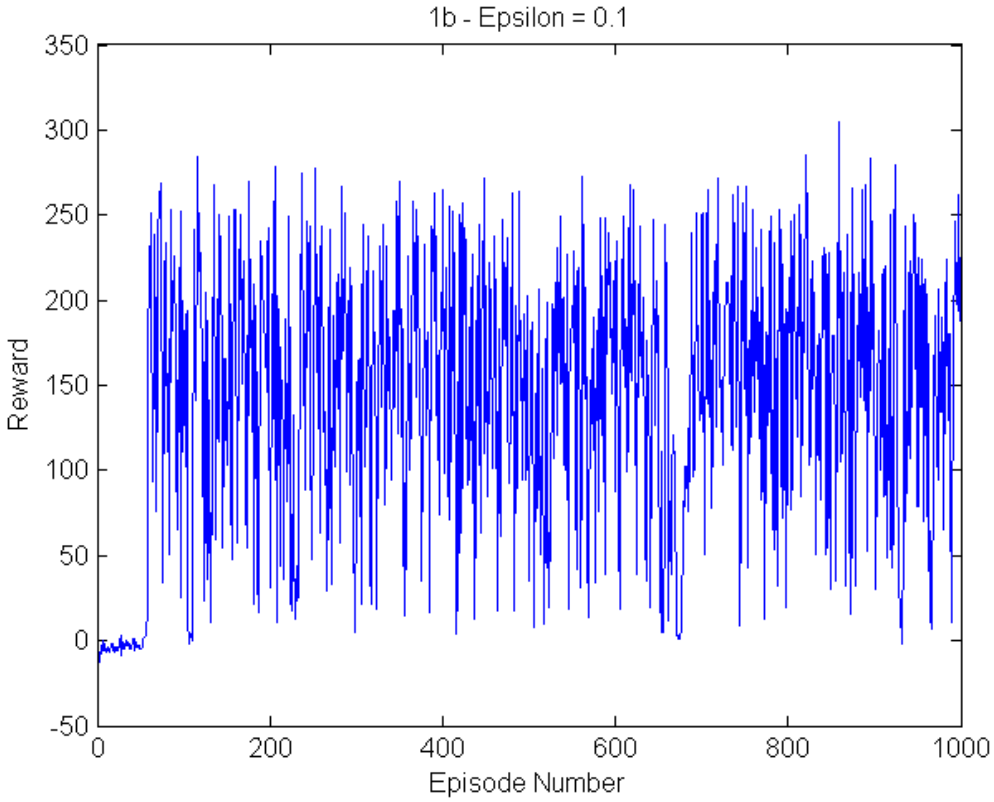
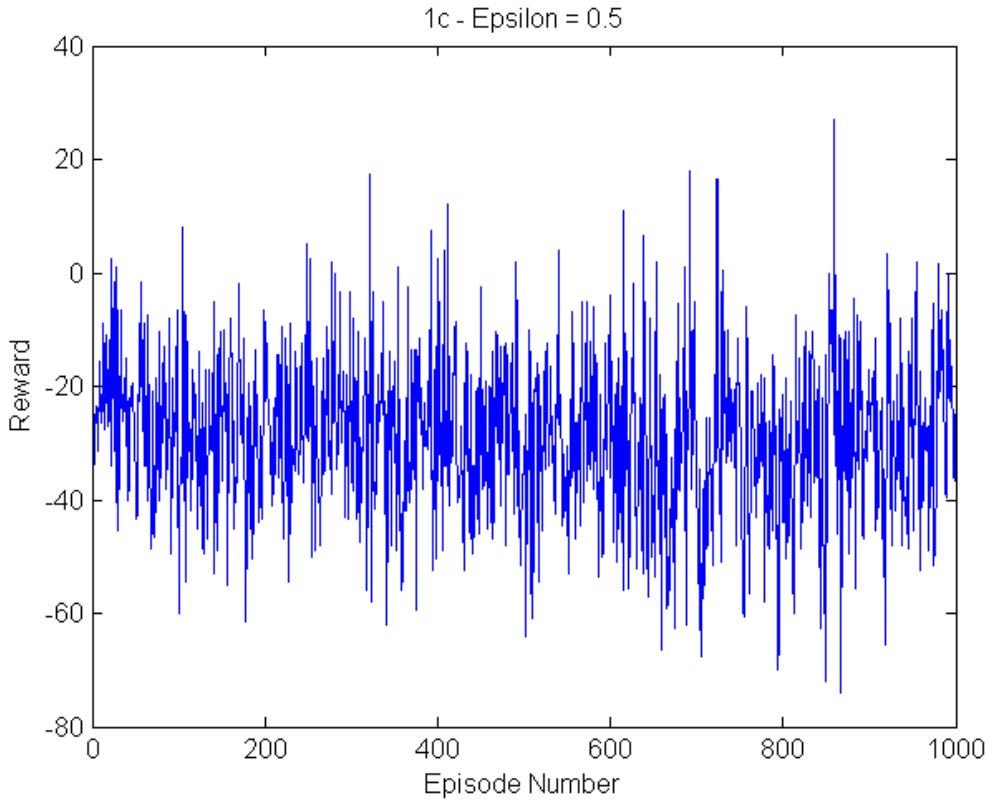
1. (a) Setting the discount factor to 0.9, the learning rate to 0.1, and ϵ to 0.0 results in very unfavorable behavior. The episodes often end up settling to zero reward, because as soon as the agent discovers a policy that returns a non-negative reward, it gets stuck on that policy. This is precisely akin to the example we saw in lecture, where one state contains a \$1 reward and another contains a \$100 reward. If the agent stumbles upon the \$1 reward first, it will continually visit that state and never go to the \$100 reward state, because there is no randomness in the system.

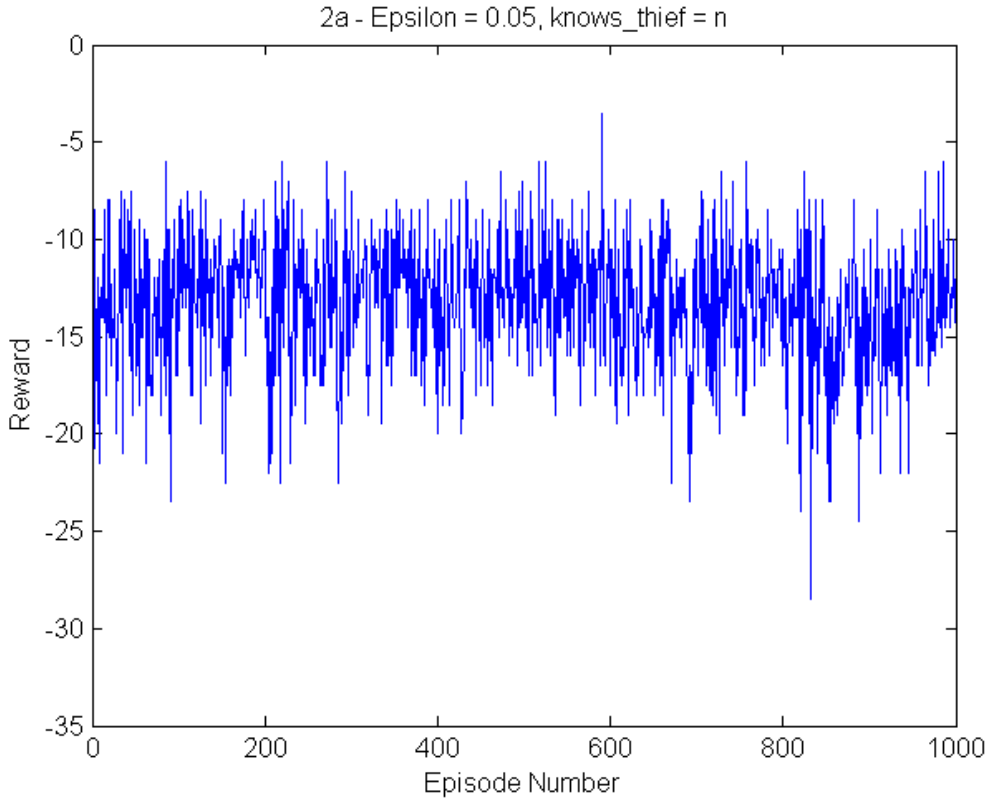
   (b) Setting ϵ to 0.1 results in a much, much better policy than setting to 0.0. The variance among episode rewards ends up being very high, but the mean is right around 150, which is significantly higher than the reward of zero achieved in 1a. This happens because ϵ is fairly high. The reward ramps to a steady state very quickly, but because of how often a random transition is chosen, the variance is high.



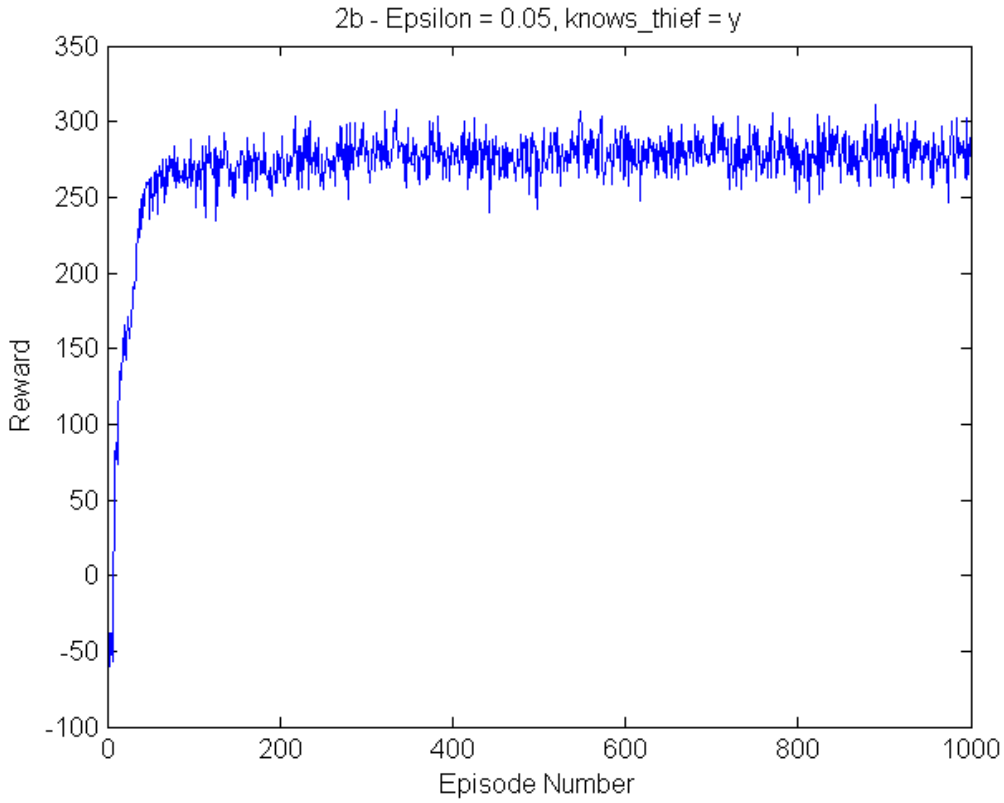
   (c) Setting ϵ to 0.5 results in a horrid policy. Every other action ends up being random, and the chance of getting penalized (by area) is greater than the chance of getting a reward, resulting in a low overall reward per episode. The policy never gets better eighter, because any time a useful state transition is found, a random action may happen regardless.


1c - Epsilon = 0.5
Reward
Episode Number
40
20
0
-20
-40
-60
-80
0
200
400
600
800
1000

2. (a) Setting ϵ to 0.05 for the WorldWithThief, knows_thief as n, results in an awful policy. The mean reward is right around -12, with variance between -5 and -20 approximately. This is somewhat expected - with the thief moving about randomly and the robot not knowing where the thief is, there is a high chance per episode of running into the thief.

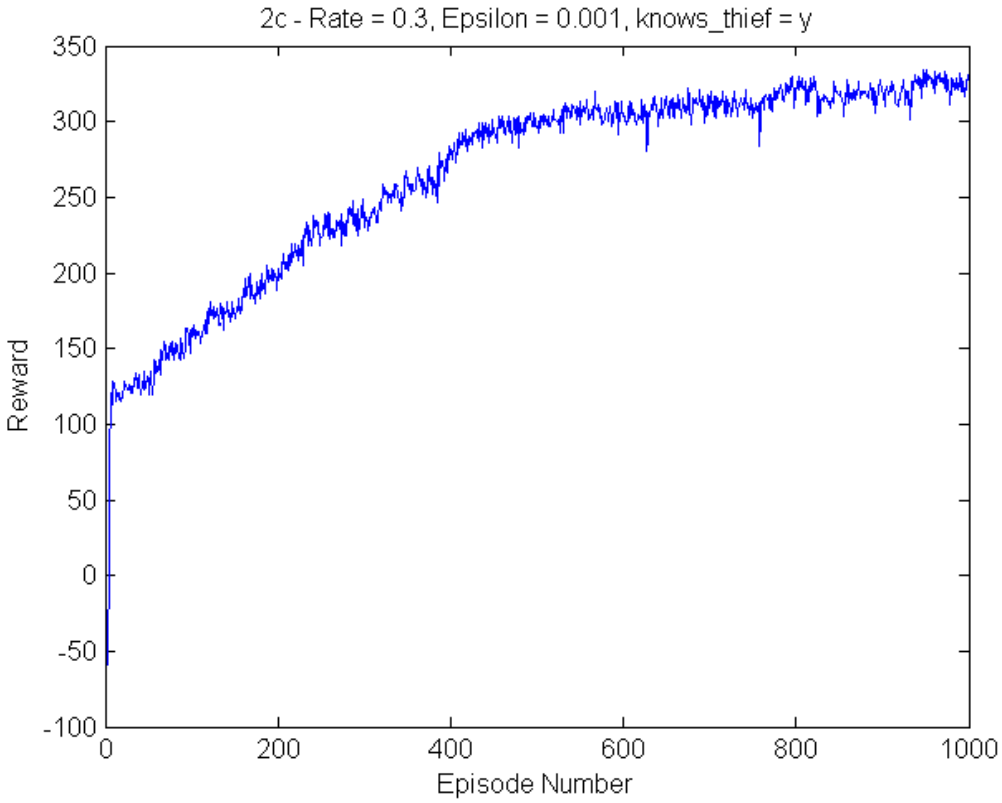


(b) Setting knows_thief to y results in a much, much better policy, for good reason! Knowing where the thief is essentially increases the number of possible states for the robot, where for every position of the robot, there are a number of possible states for wherever the thief could be. This allows our algorithm to assign explicit Q values for every position of the thief, and subsequently avoid it more often.



(c) The parameters I settled upon were learning rate = 0.3 and $\epsilon = 0.001$. To find ϵ, I ran through different orders of magnitude and tried to find which fit best. Values on the order of 0.1 resulted in much quicker policy convergence, but the reward variance was still fairly high, and the reward was lower than anticipated. Likewise, ϵ values that were too low didn't allow enough time for convergence in 1000 timesteps. The value $\epsilon = 0.001$ seemed to be an appropriate tradeoff between convergence time and final reward amount.

Adjusting the learning rate was a much less scientific process - for that, I simply slid the value from as low as 0.05 to as high as 0.5 and stuck with a value that yielded the highest mean reward.


2c - Rate = 0.3, Epsilon = 0.001, knows_thief = y
Reward
Episode Number
350
300
250
200
150
100
50
0
-50
-100
0
200
400
600
800
1000

3. A plot of learning rate = 0.3 and $\epsilon = 0.001$ for a single run can be found in 2c. The same plot averaged over ten runs can be found below. The averaged plot results in a smoother curve and a slightly quicker convergence.