Raymond Hoagland, CS440 MP1 P2

1. In WorldWithoutThief

a) $\epsilon = 0.0$

In this case, the robot only uses random decisions when there is no best decision already marked. After that point, the robot simply chooses greedily every time, thereby never exploring the other options. This greedy behavior gives the robot the tendency to appear to get stuck. This occurs because the movement may not be well defined by the policy at certain nodes that were not greedily visited. Therefore, when the robot gets there, the robot may choose to pick moves that are otherwise unreasonable, such as moving back and forth between only two locations on the map.

b) $\epsilon = 0.1$

In this case, the robot behaves in much the expected fashion. It occasionally slips and therefore receives negative penalties, but is otherwise able to successfully deliver the packages to both recipients. This is because there is enough randomness allowed now that the robot is able to properly explore the various nodes in the world and define a policy that approximates the best path available.

c) $\epsilon = 0.5$

In this case, the robot starts to get stuck more often than in part b. There is so much randomness allowed that the robot is too often picking randomly and therefore is not able to well-define the best movement from a certain node. In some cases, however, the robot is still able to successfully deliver the packages to both clients.

2. In WorldWithThief

a) knowsThief = n

In these simulations, the robot moves upwards into the top left corner and then become stuck, choosing not to move in any direction. This pattern seems to be a result of the robot not understanding that there is a thief, and thus being unable to decide on an action that will provide a favorable outcome, as the thief seems to appear randomly and cause negative penalties.

b) knowsThief = y

In this simulation, the robot is able to successfully deliver the packages. The robot is able to identify where the thief is and actively avoid him, which allows a pattern to be determined. If the robot identifies that the thief is nearby, it tries to move away from it until the thief passes, at which point the robot proceeds to deliver the packages, a policy which is likely to end up with the best reward, as less packages will be lost to the thief.

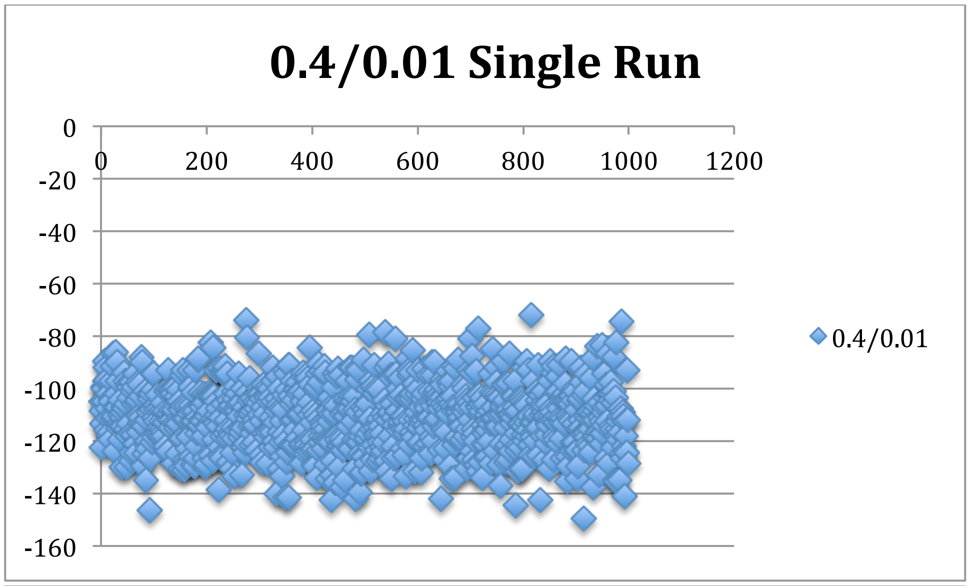
c) Identifying the best ϵ, learning rate values.

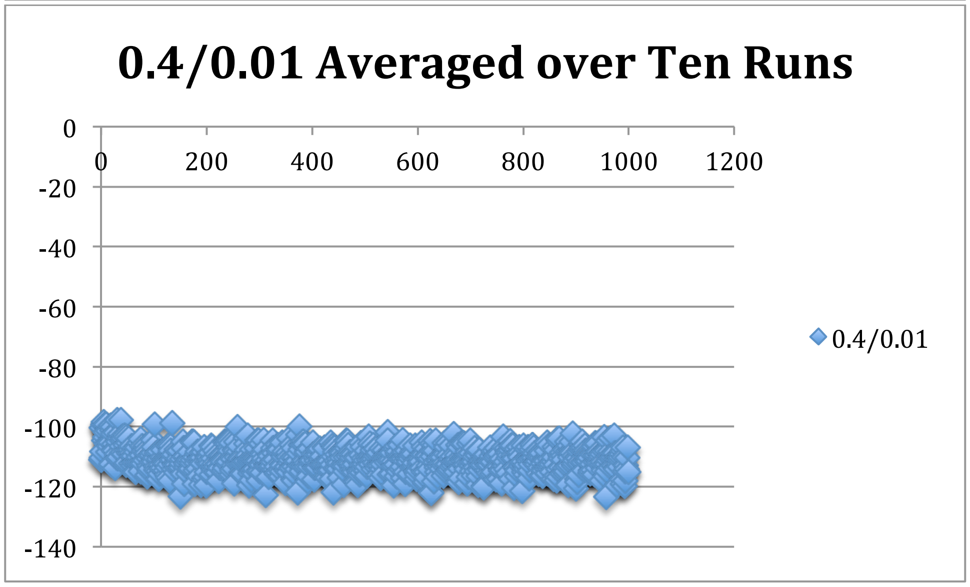
There is a tradeoff of sorts here, as making the amount of randomness too large can make it so that the robot spends too much time revisiting different nodes randomly, preventing it from defining a reasonable policy. Also, making the learning rate too high will mean that the robot only trusts the most recent values for rewards, and therefore may lose track of any important patterns that it should learn during the course of exploration.

On the other hand, having little to no randomness means that a strictly greedy approach will occur, in which the robot will stop exploring very quickly, while too low of a learning rate will leave the robot also unable to learn properly, as it will take too many steps to begin to recognize any patterns. Upon experimenting with various values for ϵ and learning rate, I decided that epsilon is best left quite small, such as $\frac{1}{100}$. This allows the robot to use some randomization to explore but ensure that for the large majority of the time, the robot sticks to greedy decision making. I also discovered that the ideal learning rate is somewhere in the middle, such as $\frac{3}{10}$. This allows the robot to take in new information at a reasonable rate while not eliminating old data too quickly either, thus allowing for the robot to learn at a modest pace.

3. Plotting Expected Reward (From Episodes)

On the first plot, the expected reward for each episode is very similar, mostly between -80 and -140. The robot appears able to determine the best policy after roughly the same amount of time during each episode. The plot of the averages reveals that the average for each episode is grouped together even more tightly. Almost every average falls within the range of -80 to -120. This means that on average, each episode takes the same amount of time for the robot to identify the best policy.


0.4/0.01 Single Run
0
-20
-40
-60
-80
-100
-120
-140
-160
0
200
400
600
800
1000
1200
0.4/0.01


0.4/0.01 Averaged over Ten Runs
0
-20
-40
-60
-80
-100
-120
-140
0
200
400
600
800
1000
1200
0.4/0.01