Matthew Farrell
CS440-ONL
Graduate Student
MP1 – Part 2

1.
(a). For WorldWithoutThief and the settings given, the Robot does not move logically at all, and in fact appears to simply move to the northwestern part of the map and remains there for the rest of the simulation. In the episodes output, the initial episode has an extremely negative reward total, which is keeping the robot from potentially attempting to cross the slippery areas again in future tests. Since the robot sees negative rewards and never takes a random movement choice despite the previous bad reward total, the potential benefit of taking actions to cross this area is never realized and the slippery locations are simply avoided consistently. Without some attempt at exploration despite having previously seen some negative reward, no positive reward is found that may be beyond it. This results in a large series of 0 total reward episodes.

(b). Setting epsilon to 0.1 gives the robot some chance to explore and results in rewards that start off in the negatives for the initial episodes, but increase quickly past the first 10-20 episodes that it runs (totals range from around -20 to the positive 240's). The robot is now relatively quickly able to learn that the slippery areas more often than not will lead to positive rewards and which slippery areas are presenting the least likelihood of dropping the packages. The potential of the states beyond the "wall" of slippery locations to offer rewards slowly works its way backwards as Q-values are updated and so the ideal states for the robot to travel to becomes the most direct route between delivery locations that avoids high-slipperiness spots (these high-sliperiness spots' Q-values will be lower due to the increased rate of dropping). The trick is for the robot to have enough randomness in the beginning to find those initial rewards, and those values will slowly work backwards through the route since a Q value is now stored that is positive and will be added in to preceding states' Q-values (and then their predecessors will gain higher Q values as well, etc., etc.). The PolicySimulator for WorldWithoutThief confirms this by showing the robot crossing directly over to the closest house for delivery, but then avoiding the more slippery red areas in preference for the safer normal slipperiness locations (choosing the best odds of not dropping). The robot is able to learn that despite the movement being longer, the expected reward is higher for traversing around the high-slipperiness locations when carrying packages. The slippery areas are appropriately traversed without concern when the robot is carrying no packages, since those states will never have earned a negative reward (cannot drop on a slippery area when carrying no packages).

(c). When setting epsilon much higher, to 0.5 or above, the total rewards become very erratic and tend to be much farther in the negative than anything run previously. Since epsilon is controlling the amount of randomness in the Robot's movement, it would make sense that such a high percentage (such as the 50% suggested) would create movement that often takes the robot off the ideal path and into random areas of the map. This has a profound effect, especially after the robot has been through a number of episodes and has a better idea of the correct path. The robot is often unable to take the action to states whose rewards are growing, because of the high random factor introduced by the high epsilon setting. The high epsilon will often result in the robot unnecessarily stepping into slippery areas and causing negative rewards. Even if the robot is not necessarily hitting a slippery location when moving at random half the time, it could very well take the robot on a very circuitous path towards the goal and often move it away from the next location where a high reward resides. This also resulted in a policy which, when tested, gave greatly varying results as to the robot's movements. It was indeed still able to learn a correct path to deliver items to the homes, but just as often the resulting policy would have the robot remain stationary from the beginning (or sometimes bounce back and forth forever between two spots).

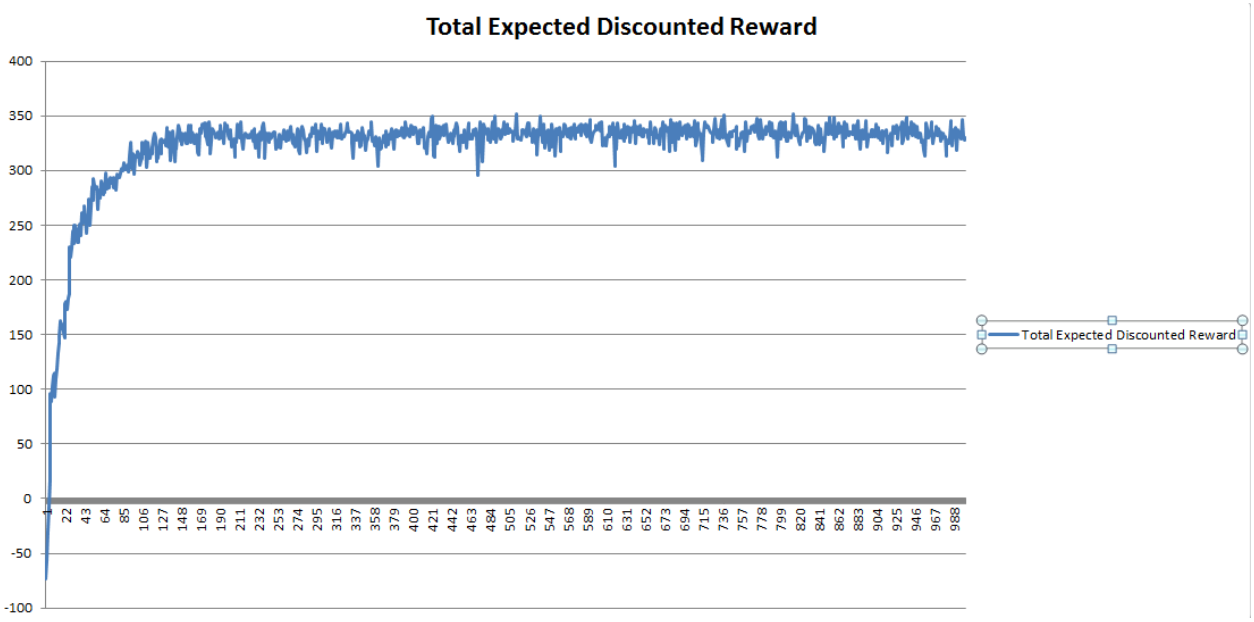
2.

(a). The results of the test in WorldWithThief with knows_thief set to “no” were very negative. The rewards for the simulation were nearly unanimous in their negativity and not often even close to 0, most being in the negative double digits for total reward of an episode. The PolicySimulator confirms this result and shows the robot simply moving itself to the top left corner of the map and staying there, never delivering any packages. This is due to the lack of knowledge of the thief’s location. Since the robot has very high slippery areas in between the two delivery locations, this will get tested and often generate large numbers of negative rewards as the robot falls more often than not. Even when successful crossing these locations, the discounted rewards of reaching home 1 will likely not be enough to outweigh the falls in other attempts. If it then attempts to move through the thief column when traveling between the two houses, the lack of knowledge of how to avoid the thief creates a large number of stolen packages. The robot will also be learning to try and avoid the less-slippery column next to the thief, which will end up channeling the robot in the thief’s column for 3 moves (from center to the top row so it can move right to house #1). States that have the third column as the robot’s location will suffer strong penalties for these stolen packages, and cause the robot to learn to shy away from these areas. Now that the Q learning has made negative rewards for the area accessing the homes (slippery tiles and the thief blocking), especially home #1, the robot ends with many high negative reward totals, and the best policy ends up being to not attempt delivery and avoid all hazardous areas to get a 0 reward score which beats out the many negative reward totals found by attempting to cross the slippery/thief locations.

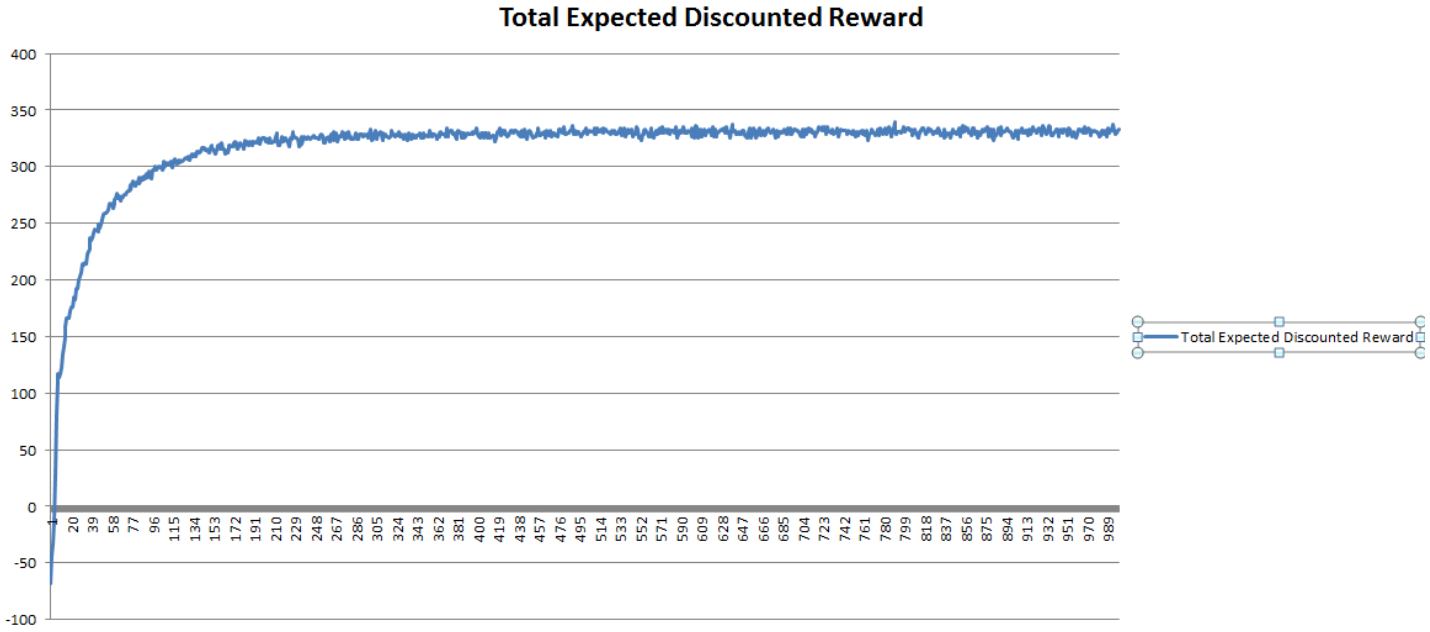
(b). Switching the knows_thief variable to “yes” makes all the difference and generates 3-digit positive rewards for nearly all episodes run with only a few large negative runs in the beginning as the optimal states are learned and then highly utilized because of their high Q-values (total rewards ranged from -65 in the first run to over 300 later in the run). The robot is (as before) able to learn about slippery areas to avoid, and in addition, now with the consideration of the thief’s position included with state information, generate negative Q-values for states where movement would take the robot into the same location as the thief. This enables the robot to precisely predict movements to avoid the thief as quickly as possible when holding packages, and to potentially delay its own movement forward toward a home if it knows the thief has moved into a position it needs to travel to next. Taking out the randomness of running into the thief creates a whole new set of states which have the thief position compensated for and quickly run up much higher Q-values than states where the thief will not come in contact with the robot. These higher Q-values are bolstered by the fact that the states the robot ideally travels now can both avoid the thief, and not need to risk stepping on any slippery locations, as a non-slippery path exists to both homes as long as you are able to avoid the thief.

(c). In order to locate the best possible learning rate and epsilon, I set up an automated simulator to run under the given conditions (10000 steps, 1000 episodes) and allowed it to run under every permutation from 0.01 to 1.00 for both learning rate and epsilon and collected the results. This number can of course vary from run to run since the top 10 results in the test were only separated by approximately 3.5 reward points. With this in mind, the particular outcome which showed the highest average reward over all of its episodes (which hopefully means it had the best balance of learning and maximum rewards later) was a learning rate of 0.16 and an epsilon of 0.01. The more noticeable number to myself is that the epsilon ends up being the absolute minimum value (without considering 0.00, ie not exploring) of 0.01. This makes some sense, as the amount of randomness wanted in the robot’s movements will be extremely minimal once we have sufficient knowledge of the world. This would help to boost reward in later episodes but perhaps slow down learning slightly in the earlier ones since it may not explore as easily. Obviously this is more than made up for in the rewards gained by not taking random actions later in the series of episodes and allow the maximum reward values to be much higher to compensate for the early negatives it may experience. A high epsilon would create somewhat of a ceiling of total reward for the simulation as it will spend slightly more time taking random steps despite knowing a near-ideal policy to follow. The ideal learning rate found is likely specific to this particular world and finds the ideal (or close to ideal) balance of future rewards to immediate ones.

3. First plotting the results of running the WorldWithThief simulation with 10000 steps and 1000 episodes (where knows_thief is yes) shows an initial very low reward value, but a quick rise after only 3 or 4 iterations of the simulation. After achieving those initial negative results and getting some learning under its belt, the Q-learning quickly ramps up and makes large gains on subsequent runs, topping out at an expected reward in the range of 330-340 with some bouncing around that number as some of the random events play out in movement and thief position. This shows that these ideal conditions provided are giving the best of both worlds in that it allows the policy to be formed by some random testing, but minimizes that randomness when a solid policy has been established for each state the robot could be in. Also ensuring that future rewards are properly balanced (by the learning rate) in the Q-value helps ensure the robot acts appropriately with respect to near and long term rewards. X Axis = episode #, Y Axis = reward



Taking 10 runs and averaging the results creates a similar graph to the previous one-run result, but has a smoothing effect when the results begin to approach the limit of maximum reward for this setup. The average tends to smooth out the variation of total reward caused by the random learning rate/selections across episodes. It again shows exponential growth as the algorithm learns and approaches the maximum reward limit (approximately 340) and then a levelling off as the algorithm has little to nothing left to learn about the world it is attempting to navigate. The limit the scores approach are limited by the fact that we are running the scenario under a finite number of steps, and would surely vary as the number of steps is increased and the robot allowed to deliver more packages and collect more rewards.



Both results from the single test and averaged 10 tests help confirm thoughts in question 2C about why the ideal epsilon is 0.01 (when testing in hundredth increments) since the ideal policy is reached relatively quickly, and this randomness harms the learning setup as many hundreds of episodes no longer need to experiment in such a small world of states, having approached the maximum reward possible in the given number of steps.