Jared Saul (jsaul3)
CS 440: Introduction to AI
MP1.2 (due Monday, September 29, 2014)

1-A) With epsilon set to 0.0, my robot agent will never make impulsively random decisions, but rather will *only* take actions that have the highest Q-value. At first, everything has the same Q-value of 0 and it breaks ties by walking around randomly, but soon enough, it slips on a tile somewhere, breaks its packages, and incurs a negative penalty. I observe that my robot had a first episode with an overall negative reward (typically between -4 and -9 on most runs), followed by 999 episodes of zero reward. My hypothesis for this is that my robot has incurred negative penalties before by walking on slippery tiles, and since it only picks actions with the highest Q-value, it purposefully avoids all slippery tiles in the future since they have Q-values less than 0. As a result, it never gets any positive rewards for delivering packages, because it is too afraid to walk on the necessary slippery tiles.

1-B) With epsilon set to 0.1, every step my robot takes has a 10% chance of throwing caution to the wind and walking in a random direction. As a direct result, it is sometimes able to momentarily overcome its fear of walking on slippery tiles, and has a much better opportunity to successfully deliver its packages (a perfect example of Exploration vs. Exploitation, as mentioned in lecture.) Indeed, out of the 1000 episodes of 10000 steps each, we can see that the first few episodes have vastly negative overall rewards from the robot messing up a lot, but those numbers become better and better as the robot starts a new episode with the knowledge of what it has done wrong in the past and the ability to avoid it. Eventually, somewhere around the 25-30-episode mark, the robot starts to learn what sequence of actions results in a positive reward, and finally gets only positive overall rewards per episode as it starts reaping good rewards nonstop. It still has a 10% chance to walk randomly due to epsilon, but when lucky and following its Q-values, it sometimes gets an episodic reward of over 250 points!

1-C) With epsilon set to 0.5, my robot is walking randomly 50% of the time—practically every other step. I observe that the overall episodic rewards are *horrible* (only 22 out of 1000 were positive), *but*, there's still the chance that sometimes it learns a good policy. As the lecture slides state, if epsilon is too low, the robot will *slowly* converge to a *good* region around π^*, but if it's too high, the robot will *quickly* converge to a *poor* region around π^*. Indeed, even though it's doomed to make many mistakes during the simulation due to the 50% randomness rate, it also learns a lot of what works and what doesn't even when it didn't want to take that action. Several tests on this robot shows that its final chosen policy can make some mistakes—walking on more slippery tiles than necessary or getting stuck after delivering packages, for example—but at least it is far better than when epsilon was 0.0.

2-A) My agent, in this world and with no knowledge of the thief, has abysmal performance. In all 1000 episodes (overall multiple runs, even) it doesn't have a single overall positive reward, and its chosen optimal policy is to walk north for a little bit and then stand around doing nothing. This is actually pretty easily explainable! In a world where my robot agent does not know the location of the thief, it ends up getting mugged and losing its packages many, many times. In the end, its policy of standing around (or in reality, walking west into a wall non-stop) tells me that my robot feels it is safer to *never deliver a package at all* than it would be to attempt to deliver one and fail. At least then it would have an overall reward of 0.0, which would actually indeed be better than all 1000 of its simulated runs!

2-B) Upon setting `knows_thief` to `y`, the performance difference is substantial! By knowing where the thief is, my robot agent has knowledge of when he is most likely about to get mugged, and when it is safe to cross the vertical line the thief is in. In this run, my robot's first six episodes had negative overall rewards, the next six had overall positive rewards, the next fifteen were in the hundreds of points, and every episode after #27 had *at least* 200 points in the final overall reward. The reasoning behind this is obvious: encountering the thief results in such a large negative reward that the robot would surely go out of its way to avoid meeting up with it, and the extra state knowledge of if the thief is close enough to attack the robot on its next turn is all the information that is needed. In fact, this extra state information makes my robot plan ahead—if the thief is likely to be at the crossroads in WorldWithThief in *two turns*, my robot walks against the left wall to give the thief a chance to move onward. All of this planning pays off—my robot agent's optimal policy never slips on a slippery tile, and never encounters the thief!

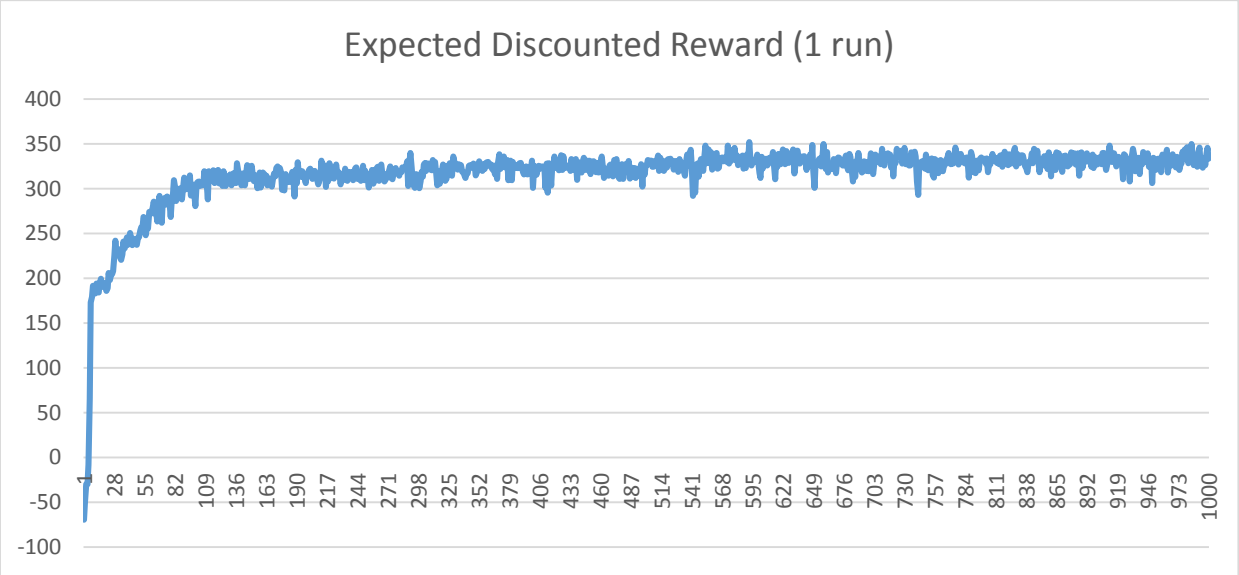
2-C) I'll set different parameters for the learning rate and epsilon, and record the average overall reward (sum reward of all episodes divided by 1000) for each pair below. The average overall reward itself is the average of several runs of Simulator.

Learning rate	Epsilon	Average overall reward
0.15	0.00	90.4
0.01	0.01	223.8
0.05	0.01	292.2
0.10	0.01	308.9
0.15	**0.01**	**320.2**
0.20	0.01	313.5
0.25	0.01	319.6
0.01	0.05	260.4
0.05	0.05	269.6
0.10	0.05	273.0
0.15	0.05	272.5
0.20	0.05	270.5
0.25	0.05	267.3
0.01	0.10	209.8
0.05	0.10	211.3
0.10	0.10	211.7
0.15	0.10	211.2
0.20	0.10	209.6
0.25	0.10	205.1

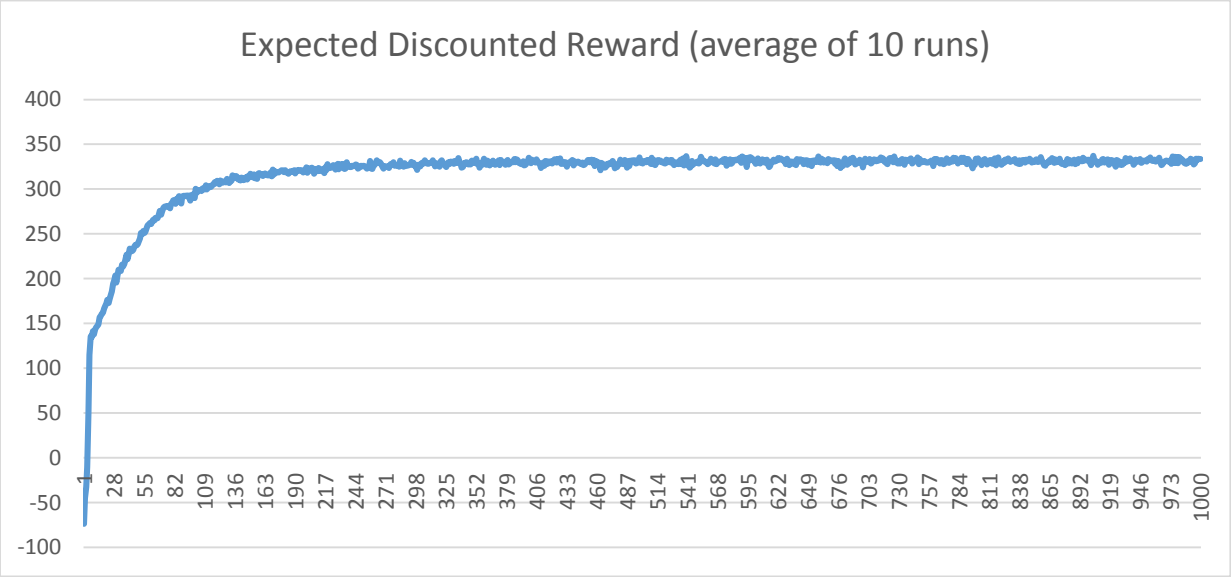
For my investigation, I ran three different values of epsilon (0.01, 0.05, and 0.10) with six different values of learning rate (0.01, 0.05, 0.10, 0.15, 0.20, and 0.25). My investigations showed that a learning rate between 0.10 and 0.15 typically obtains the best average overall reward, while a lower epsilon value does the same. (Afterwards, I tested setting epsilon to 0.0, but with uninteresting results.) I was quite surprised to see that such a low epsilon did so well, but in hindsight, it makes sense. As mentioned in lecture and in my answer to 1-C, a low epsilon will cause my robot agent to converge upon a good optimal policy, but extremely slowly. However, in this case, the robot's knowledge of the thief allowed

it to avoid bad choices, and the low value of epsilon meant that the robot spent less time randomly deciding to walk onto a slippery tile or into the path of the thief anyway. Given that this was definitely not the case in WorldWithoutThief, this experiment just goes to show that good values for epsilon and the learning rate may very well depend on the system and world itself!

3) Using a learning rate of 0.15 and an epsilon value of 0.01…



As mentioned in my answer to question 2-C, in this case I observe that my robot had negative overall rewards for the first 5 or 6 episodes, followed by a quick shot upward as it learned how to optimally deliver its packages. I notice that my robot seems to reach its peak somewhere around 500 episodes in, and doesn't really gain any improvements past that point. I assume that the graph for 10 averaged runs is going to look very similar to the graph above, but much less choppy.



I was right! It's still a little choppy at first (the runs take between 3 and 6 episodes to stop having overall negative rewards, so they climb in the high positives at varying rates) but with this graph I can definitely see the curve smoothen out. In fact, this graph makes it look like my robot reaches its peak performance around 333 episodes instead of 500.

The obvious thing to observe here, of course, is that the robot starts off making a lot of mistakes as expected, learns from those mistakes, and gradually begins to avoid getting caught by the thief or walking on slippery tiles. A few hundred runs in, my robot agent is finally masterful at waiting for the thief to pass before moving across to deliver packages, and this is evidenced by all of its scores past run #113 being over 300 points.