CS 440 – MP1 Part 2 – Daniel Calzada (dcalzad2)

1. In WorldWithoutThief:
    a. After creating the policy with epsilon = 0 and running PolicySimulator, the agent appeared to get stuck in infinite loops, either going back and forth between two squares or repeatedly running into a wall. This is because as soon as the agent slips on a slippery square, the reward for that action will be negative and the agent will never try to perform that action from that state. After simulating enough times, most likely the agent will eventually slip on all the slippery squares barring access to the customers, therefore ensuring that the agent no longer delivers packages. When this happens, the agent has no choice but to loop infinitely between one or more states without delivering any packages.
    b. After setting epsilon back to 0.1, the agent works as expected, correctly delivering packages to the customers. This is because even if at one point the agent slips on a patch of ice, it may eventually try that path again and succeed, meaning that it may be able to deliver the packages and gain a reward.
    c. As epsilon increased to 0.5 (all the way up to 1), the robot seemed to take less optimal paths to the goal. One explanation for the cause is that as epsilon increases, the robot's actions when exploring become more and more random, so it has less experience following the optimal path than a robot with epsilon = 0.05 would have. Because of this, if it explores a possible but non-optimal path frequently, the q-values along that path will continue to increase along that path rather than on the optimal path. Then when the policy is calculated, the less-optimal path is chosen due to the higher q-values.
2. In WorldWithThief:
    a. With knows_thief = n and epsilon = 0.05, the agent gets stuck in an infinite loop without delivering any of the packages. This is because the reward for delivering a package is small (1.0) compared to the reward for running into the thief (-2.0). Running into the thief is much more likely than delivering the packages, so the agent thinks it is best to stay away from the thief and customers entirely in order to avoid getting the packages stolen. In fact, if the reward for delivering packages is changed to 1000.0, the agent will work as expected, delivering packages and avoiding the thief as much as it can (without actually knowing its location).
    b. With knows_thief = y, the agent functions as expected. This is because it now knows the position of the thief at all times and, by assigning negative rewards when getting

packages stolen, the agent can learn to work around the thief. When knows_thief = n, all it could do was try to get out of the thief's way, but most of the time it hit the thief. Now it can steer around it.

c. Using alpha = 0.2 and epsilon = 0.05, the agent converged to an optimal path in WorldWithThief, knowsthief = y in ~17 episodes. By changing epsilon or the learning rate, the agent converged more slowly. Of course, the output of the program is non-deterministic, so it is possible that there is a better configuration of alpha and epsilon, but the one aforementioned yielded favorable results when simulated.

   One reason could be that when alpha is larger, any random action taken that makes the agent move into a good spot will be treated (very quickly) as the best action. This, coupled with a low epsilon, will mean that the agent will be unlikely to explore any other paths, whereas a lower learning rate will make it more likely to explore all paths evenly and treat them all more-or-less the same. Contrarily, a small learning rate will still allow the policy to converge to the optimal policy, but this convergence takes longer. Likewise, changing epsilon will either make it too likely to explore a random path that it might ignore the good path, or make it ignore all paths once it finds one that works.

3. The plot using one run of the simulation converged quickly, but it contained a lot of noise so the exact total expected reward being converged to was difficult to determine. However, after averaging 10 runs, the values were approximately the same, but contained less noise. In the average of 10 runs, the values mostly converged after 17 runs, but continued to improve very gradually.

CS 440 – MP1 Part 2 – Daniel Calzada (dcalzad2)