1. WorldWithoutThief
(a) ε = 0.0
In this case, I simulated my agent for 100 times. And here is the result of rewards (10000 steps):
Average Reward = 0.0, Standard Deviation = 0.0, Success Rate = 0%

In each simulation:
As episodes.txt shows, the robot received negative rewards for a few times and then all it received was 0.

Overall:
As PolicySimulator shows, the robot got stuck or went into a non-rewarded loop after a few steps.

The reason is that when ε = 0, the agent will strictly follow the greedy policy and will not explore other options, which means it will not find the path to maximize the reward. Instead, it will change strategies if it loses and be happy with any result where the agent wins. So in most cases, the agent loses in slippery locations and does not further explore the right side of the map.

(b) ε = 0.1
In this case, I simulated my agent for 100 times. And here is the result of rewards (10000 steps):
Average Reward = 112.60, Standard Deviation = 147.14, Success Rate = 37%

In each simulation:
As episodes.txt shows, the rewards fluctuated a lot in the first twenty episodes and afterward the rewards became very high on average.

Overall:
As PolicySimulator shows, the robot successfully delivered for about 37% of all cases.

Q-learning policy with reasonable amount of exploration enables the agent to explore the entire state space and converge to an optimal strategy even the agent temporarily loses in slippery locations. When ε = 0.1, the chance to successfully deliver is higher than the chance without exploration.

(c)
ε = 0.5
In this case, I simulated my agent for 100 times. And here is the result of rewards (10000 steps):
Average Reward = 138.44, Standard Deviation = 150.64, Success Rate = 46%

In each simulation:
As episodes.txt shows, the rewards in each episodes were mostly negative.

Overall:
As PolicySimulator shows, the robot successfully delivered for about 46% of all cases.

ε = 0.8
In this case, I simulated my agent for 100 times. And here is the result of rewards (10000 steps):
Average Reward = 134.69, Standard Deviation = 149.25, Success Rate = 45%

In each simulation:
As episodes.txt shows, the rewards in each episodes were mostly negative.

Overall:
As PolicySimulator shows, the robot successfully delivered for about 45% of all cases.

High exploration to improve one's knowledge is of no use if one never puts that knowledge into practice. In our case, large ε value will make the agent learn faster, but ignore what it has learned far too often. So in each episode, the rewards were mostly negative, because the robot chose random action and received a lot of penalties instead of using what he learned. I can't be sure that whether ε = 0.5 or ε = 0.8 is optimal for learning, but they both seem to work well in policy simulator.

2. WorldWithThief
(a) ε = 0.05, knows_thief = n
In this case, I simulated my agent for 100 times. And here is the result of rewards:
Average Reward = 0.0, Standard Deviation = 0.0, Success Rate = 0%

In each simulation:
As episodes.txt shows, the rewards in each episodes were mostly negative.

Overall:
As PolicySimulator shows, the robot did not deliver packages and the total reward is 0.

When the robot does not know the position of the thief, the chance to have its packages stolen is so high that it is more likely to receive penalty than getting reward. Consequently, even with certain amount of exploration, the robot chooses not to deliver in order to avoid penalty.

(b) ε = 0.05, knows_thief = y
In this case, I simulated my agent for 100 times. And here is the result of rewards:
Average Reward = 350.08, Standard Deviation = 7.20, Success Rate = 100%

In each simulation:
As episodes.txt shows, the robot received negative rewards for a few times, afterwards it received pretty high rewards on average.

Overall: The reboot delivered packages successfully in all simulations.

Since the robot knows the exact location of the thief, it can tactfully avoid the thief and deliver packages.

(c) best rate & best ε
Using rate = 0.1, and simulate for ~10 times:

ε	0.0001	0.001	0.005	0.01	0.05	0.1
episodes to receive ~100	12	7	7	5	9	10

episodes to receive ~200	80	75	38	20	15	50
episodes to receive ~300	190	170	175	130	>1000	>1000

I choose ε = 0.01 as the optimal value.

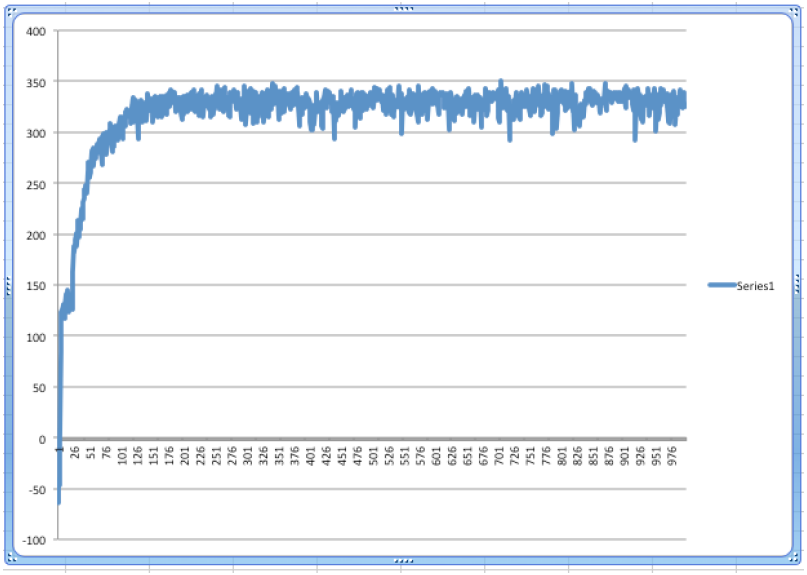
Using ε = 0.01, and simulate for ~10 times:

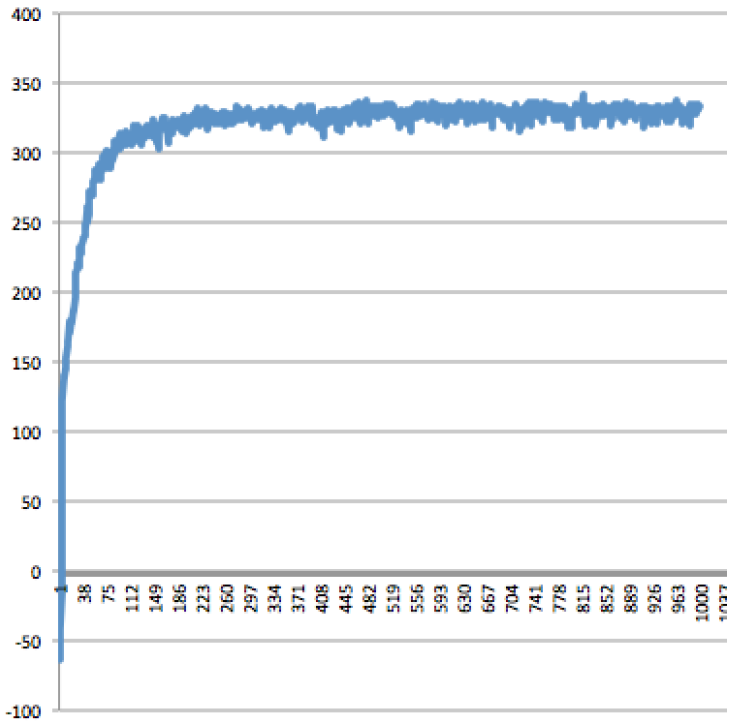
rate	0.1	0.2	0.3	0.4	0.5	0.6
episodes to receive ~100	6	10	7	7	9	7
episodes to receive ~200	18	25	23	23	15	15
episodes to receive ~300	155	100	98	130	fluctuate	fluctuate

For rate > 0.5, the rewards fluctuate a lot and they do not go stably above 300.
I choose rate = 0.2 as the optimal value.

So best ε = 0.01, best rate = 0.2.

3.





The first graph was generated by 1 simulation. The second graph was generated by averaging 10 simulations. Note that the second graph is smoother.

It is smoother because the average converged to a curve. It is reasonable to assume that the more simulations we perform, the more it converges to a curve line.