For this assignment, I will implement and analyze various reinforcement learning algorithm based on the classic 2D grid problem (similar to the problem. Basically there will be a 2D world in which the robot can move around in 2D grids; however, the robot is not a very good robot. When asking the robot to move to a particular direction, it only has 80% chance to follow the command, while having 10% chance to slide sideway (10% each side). There is three possible objects in the 2D world, namely the charging station, the trap, and obstacles. The charging station is considered a positive reward, and the trap will be something that the robot wants to avoid. Both charging station and the trap are absorbing states. On the other hand, obstacles will be a place that the robot cannot reach.

The reason why I choose this problem is that this type of 2D problem the basic set up for reinforcement learning, and the 2D grid can be easily generalized to a variety of classic videos games and real-world problems. To change nature of the game or problem, one can modify the reward matrix, the amount and location of various objects (charging station, trap, and obstacles), and changing how the robot will behave, and the size of the grid. The possibility to easily modify the natures of the problem allow us to apply the RL algorithm to a real-world situation. For example, designing the best path for the robotic vacuum, logistic, etc. One typical drawback of reinforcement learning is that RL algorithms typically need a lot of data. In the 2D world problem, the reward matrix, transition matrix, the size of the of the world, the behavior of the robots, and the objects in the world are given; however, for real-world problem, this information are typically absent and requires a large amount of effort to collect.

Problem 1: A simple 3X4 grid with 1 obstacle, 1 charging station, and 1 trap

First of all, I choose this problem because it is relatively small with few objects in the grid, thus the optimal route can be easily discerned by a human. This is very important because this provides a better understanding of the output of the various algorithm.

For the 1st problem, the reward matrix is shown in Table 1. There is a charging station (+1 reward) on the top right, a trap right below it (-1 reward), and obstacles called "wall". When the robot stays in any other gird, it loses a little bit of power (-0.04 reward).

Table 1: Reward matrix for problem 1. +1 is the reward for charging station, -1 is the reward for the trap.

-0.04	-0.04	-0.04	+1
-0.04	-0.04	-0.04	-1
-0.04	wall	-0.04	-0.04
-0.04	-0.04	-0.04	-0.04

Value Iteration Algorithms:

The value iteration algorithm finds the expected utility for each state in the world. Once the optimal utility function is found one can obtain the optimal policy. For my implementation, I define the stopping criteria (convergence) as when the utility between two iteration does not change a lot. According to Russell and Norvig, Artifical Intelligence A Modern Approach, the convergence can be defined using the following formula:

$$||U_{k+1} - U_k|| < \epsilon \frac{1-\gamma}{\gamma}$$

Where epsilon can be set by the user, and gamma is the discount factor.

For this specimen problem, the only parameter that can be turned is the discount factors and the epsilon value, since all other parameters are already defined given the problem. Discount factor describes the preference of agent for the current rewards over future reward. When discount factor is close to 0, rewards in the distant future are viewed as insignificant, while when the discount factor is closed to 1, the discount rewards behaves more like additive rewards.

Figure 1 shows how much iteration does it need to converge for various discount factors and epsilon values. As the epsilon becomes larger, the number of iteration needed increases. This is typical because we are requiring the utility matrix to change by a smaller degree. On the other hand, as the discount factor increases, the number of iteration needed increases and this because when updating the utility matrix, a large discount factor means that we are changing the utility matrix by a lot; thus a larger discount factor will result in more iterations to converge. Typically the discount factor is set to value greater than 0.9.

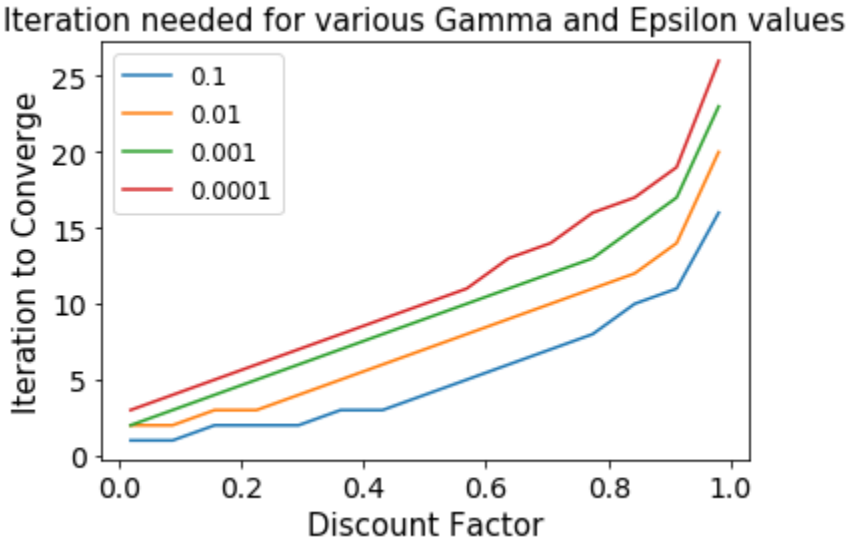

Figure 1: The iteration to converge vs different discount factors and different epsilon values for value iteration algorithm on problem 1.

Figure 2 shows the utility of every state as a function of iterations for gamma = 0.95 and Epsilon = 0.0001. Most of the utility value reach equilibrium after about 12 iterations. This value is smaller than the iterations to converges value calculated using the formula shown above the formula poses a relative strict requirement for convergence. From the plot below, one can observe that the closer the grid location to the top right corner (where the absorbing state is), the long it takes to converge. This makes sense because at large gamma value, the utility function consider the future reward to be more additive, thus when the grid is closer to absorbing state, it is changing the utility function by a lot.

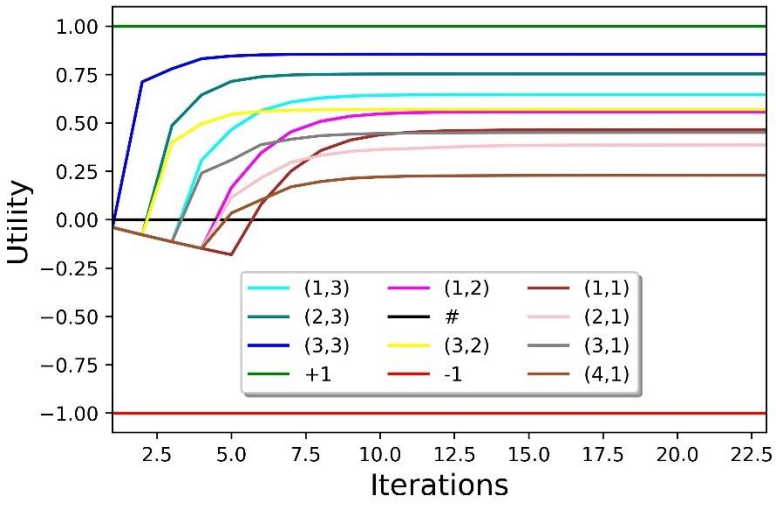

Figure 2: The utility value for each grid location vs iteration for problem 1.

For this problem, the optimal policy is shown in Figure 3(ref: https://mpatacchiola.github.io/blog/2016/12/09/dissecting-reinforcement-learning.html). This policy will be called “real optimal policy”.

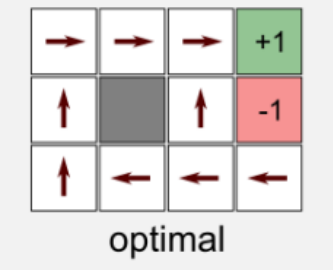

Figure 3: The real optimal policy according to reference.

I found that when the discount factor is small, the algorithm does not output the maximum policy for some position. For example, when discount factor = 0.95, the policy is not optimal for the (3,3) position shown in the output shown in Figure 4. When discount factor =0.99, the policy is the same as the real optimal policy.

```
================== FINAL RESULT ==================
Iterations: 10
Delta: 7.82334e-05
Gamma: 0.5
Epsilon: 0.0001
The utility matrix converge to:
==================================================
[ 0.008594  0.125526  0.382436  1.      ]
[-0.040665  0.        0.066288 -1.      ]
[-0.062115 -0.053304 -0.019887 -0.074581]
==================================================
The optimal policy is the following:
==================================================
 >   >   >   *
 ^   W   ^   *
 ^   >   ^   v

==================================================
```

```
================== FINAL RESULT ==================
Iterations: 14
Delta: 2.21347416479e-05
Gamma: 0.7
Epsilon: 0.0001
The utility matrix converge to:
==================================================
[ 0.159273  0.328792  0.576359  1.      ]
[ 0.057191  0.        0.228775 -1.      ]
[-0.007985  0.008136  0.083938 -0.067743]
==================================================
The optimal policy is the following:
==================================================
 >   >   >   *
 ^   W   ^   *
 ^   >   ^   <

==================================================
```

```
================== FINAL RESULT ==================
Iterations: 24
Delta: 2.6645013968e-06
Gamma: 0.95
Epsilon: 0.0001
The utility matrix converge to:
==================================================
[ 0.646793  0.753141  0.855321  1.      ]
[ 0.557485  0.        0.569109 -1.      ]
[ 0.464533  0.386472  0.45105   0.229609]
==================================================
The optimal policy is the following:
==================================================
 >   >   >   *
 ^   W   ^   *
 ^   <   ^   <

==================================================
```

```
================== FINAL RESULT ==================
Iterations: 36
Delta: 4.68129301989e-09
Gamma: 0.9999
Epsilon: 0.0001
The utility matrix converge to:
==================================================
[ 0.811198  0.867567  0.917681  1.      ]
[ 0.761098  0.        0.660083 -1.      ]
[ 0.704744  0.654657  0.610743  0.38728 ]
==================================================
The optimal policy is the following:
==================================================
 >   >   >   *
 ^   W   ^   *
 ^   <   <   <

==================================================
```

Figure 4: Example of optimal utility value and corresponding policy for different gamma value for value iteration algorithm. Top left, gamma = 0.5, top right: gamma =0.7; bottom left: gamma = 0.95; bottom right: gamma = 0.9999.

From the results above, we can see that only at extremely high gamma value, the value iteration algorithm gets the (3,3) correct. This is probably due to the nature of the problem itself instead of the problem of the algorithm. Because the negative reward associated with the trap is too small, the penalty for going around from bottom right back to bottom left then up and right is still larger than the penalty of having 10% chance stepping into the trap.

If we change the reward matrix to the following by changing the reward of the trap from -1 to -100, then the reward matrix is shown in Table 2:

Table 2: A slight modification to the original game by changing the reward for the trap from -1 to 100.

-0.04	-0.04	-0.04	+1
-0.04	-0.04	-0.04	-100
-0.04	wall	-0.04	-0.04
-0.04	-0.04	-0.04	-0.04

Then with gamma equals to 0.95, the optimal policy will try to avoid the trap as much as possible, even trying to bang the wall when at (2,3) position (see Figure 5).

```
================== FINAL RESULT ==================
Iterations: 70
Delta: 5.02374367181e-06
Gamma: 0.95
Epsilon: 0.0001
The utility matrix converge to:
==================================================
[ 0.61978   0.724142  0.824415  1.      ]
[   0.532139    0.          0.274687 -100.      ]
[ 0.440925  0.364324  0.290587 -0.085512]
==================================================
The optimal policy is the following:
==================================================
 >   >   >   *
 ^   W   <   *
 ^   <   <   v

==================================================
```

Figure 5: Utility matrix and optimal policy when changing the reward of the trap from -1 to -100.

Policy Iteration Algorithms:

For policy iteration, the converging criteria I used is that both utility function and policy are reaching a equilibrium. A chart of iteration to converge vs. different Epsilon values and discount factor for the same problem shown in Table 1 is shown in Figure 6:

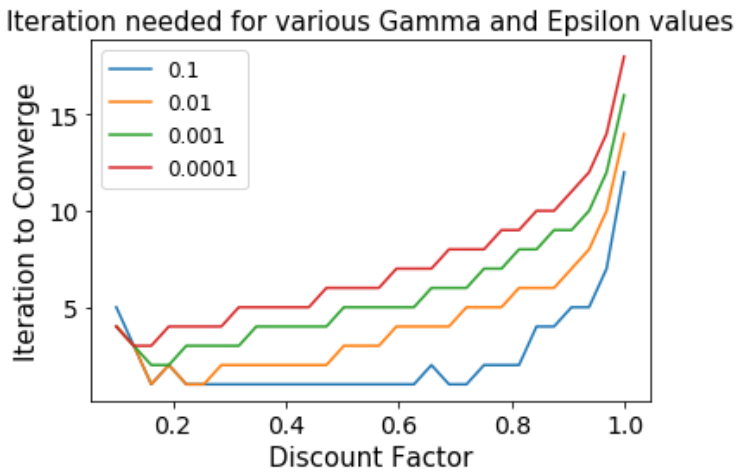

Figure 6: The iteration to converge vs different discount factors and different epsilon values for policy iteration algorithm on problem 1.

Compare to the chart obtained from value iteration earlier, it is obvious that the policy iteration algorithm converges faster, given that the discount factor and the epsilon values are the same. For example, when gamma = 0.9999 and Epsilon = 0.0001, it takes 25 iterations for Policy Iteration to converge while the Value Iteration needs 36 iterations.

This is because optimal policy typically converges before the value function. The policy iteration algorithm is jumping at policy space instead of value space, and policy iteration algorithm does not have to extensively compute the exact value matrix to find the best policy (because policy already converged).

The utility for each state for the policy iteration algorithm iteration when gamma = 0.95 and epsilon = 0.001 is shown in Figure 7. The trend and conclusion is the same as those shown in Figure 2

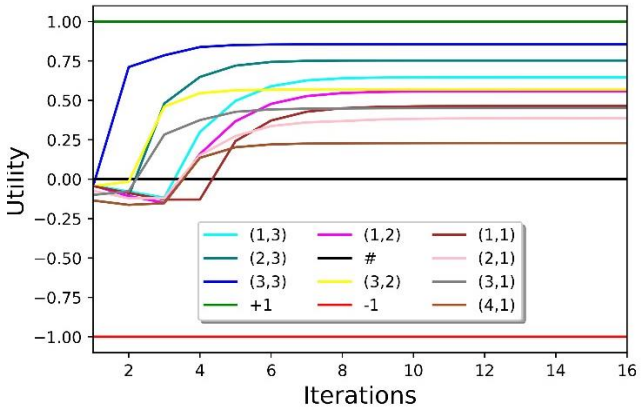

Figure 7: The utility value for each grid location vs iteration for problem 1 using policy iteration for gamma = 0.95 and epsilon = 0.001.

In terms of policies, I found that the policies obtained from the Policy Iteration Algorithm are pretty close to the Value Iteration algorithm. The utility function is pretty close too (see Figure 8 and Figure 4).

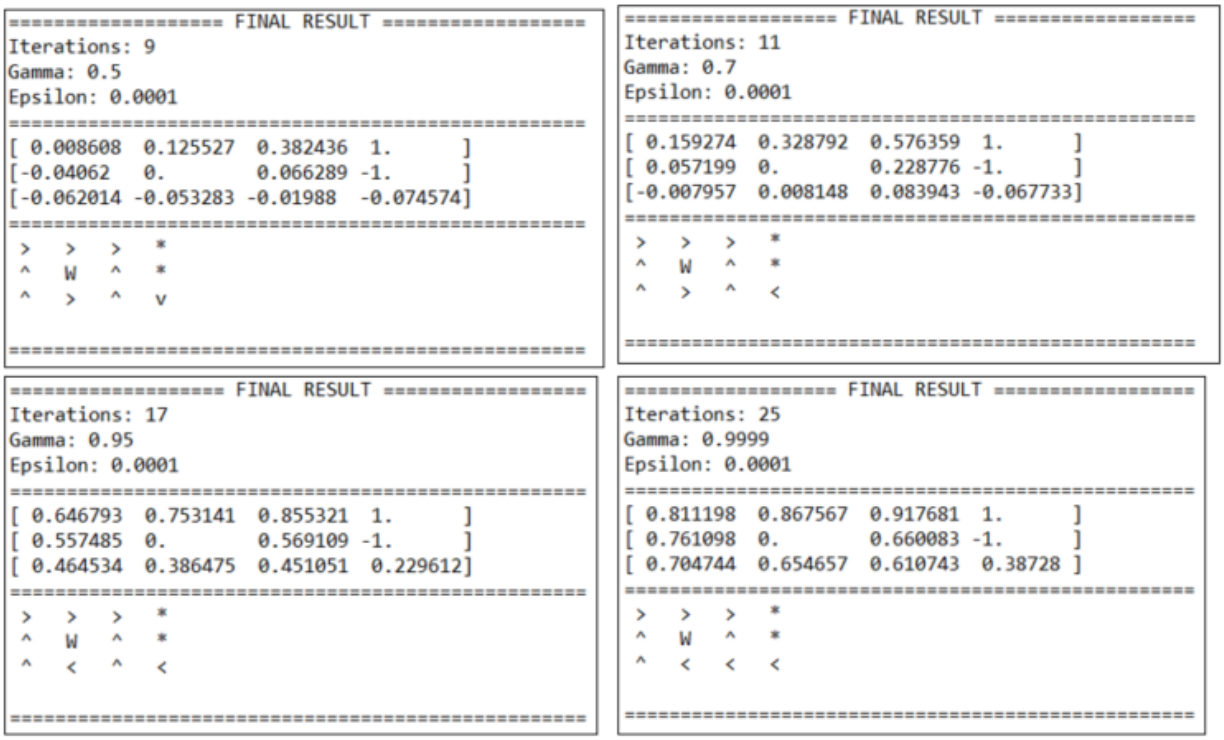

Figure 8: Example of optimal utility value and corresponding policy for different gamma value obtained from policy iteration. Top left, gamma = 0.5, top right: gamma =0.7; bottom left: gamma = 0.95; bottom right: gamma = 0.9999.

If I change the reward for the trap from -1 to -100, the result is also the same as value iteration; thus I conclude that these two algorithms generate almost the same result for this problem.

Q-Learning:

Q learning algorithm is a model-free learning method. It assumes that the Q learning agent does not know the transitional matrix and the reward matrix; thus, the agent has to explore the "world" and update the state-action matrix. It is typical that the Q learning method takes longer to converge because little information is given to the Q learning agent. Overall, for the same problem, even though the grid is only 3X4, it is much slower to obtain the optimal policy compared to value iteration or policy iteration.

When alpha (learning rate) is set to 0.1, and I tried different exploration matrix. If I set the exploration matrix as the optimal matrix obtained by value iteration or policy iteration, the time to finish every iteration is much faster compared to the situation where the exploration matrix is set to some random policy. In addition, the number of iterations needed to reach a relative equilibrium is also smaller. This is because a good initialization can help the algorithm to converge faster, thus it does not have to visit some states that often. This result agrees with Reference 2.

For the original problems, after running the Q learning algorithms for the game presented in Table 1, I found that even at high iteration, the optimal policy still occasional changes and never reach a stable equilibrium. The points that are oscillating are at the [2,3] position and bottom right position.

```
Policy matrix after 397001 iterations:
 >   >   >   *
 ^   #   ^   *
 ^   <   ^   v
```

```
Policy matrix after 398001 iterations:
 >   >   >   *
 ^   #   <   *
 ^   <   ^   <
```

To check out what happens if I magnify the reward, I reran the Q learning algorithm for the game presented in Table 2 (changing the reward of the trap from -1 to -100). It turns out the optimal policy is still oscillating at when the number of iteration is greater than 380000. It seemed like the Q learning algorithm is still having difficult to pin down 1 single optimal policy at the [2,3] position and bottom right position., even though it gets most of the location with optima policy.

The problem of Q learning is that it needs many iterations to obtain the optimal policy even for a small grid world, and for the most of the problems, it is hard to get an idea of the upper bound of iteration needed. In real world situation, it means that the robot has to collect an extremely large amount of data to achieve optimal policy, which is usually unrealistic.

Problem 2: A 8X8 2D world with multiple absorbing states:

For the second problem, I increased the grid size from 3X4 to an 8X8 grid so that I can examine the behavior of various algorithm for problems large state. I generate the example grid world using random function. For each row, there will be 1/8 chance for obstacles, 5/8 change for a slightly negative rewards (-0.04), and 2/8 chance for a trap with -1 reward. After I generate the random grid, I did some modification to make sure an optimal route can still be discerned by a human.

The reward matrix is shown below, where "wall" is obstacles, -1.00 is a trap, 1.00 is the charging station.

```
[-1.00, 'wall', -0.04, -0.04, -1.00, -1.00, -0.04, 1.00],
[-0.04, -0.04, 'wall', -0.04, -0.04, -0.04, -0.04, -0.04],
[-1.00, 'wall', -0.04, -1.00, -0.04, -1.00, -0.04, -0.04],
[-0.04, -0.04, 'wall', -0.04, -0.04, -0.04, -1.00, -0.04],
['wall', -0.04, 'wall',-0.04, -1.00, -1.00, -0.04,'wall'],
[-1.00, -0.04, -0.04, -0.04, -0.04, -0.04, -1.00, -0.04],
[-0.04, -0.04, -0.04, -0.04, -0.04, 'wall', -0.04, -0.04],
[-0.04, -1.00, -1.00, -0.04, -1.00, -0.04, -0.04, -0.04],
```

Value Iteration Algorithms:

By plotting the iteration needed for different discount factors and Epsilon values, one can obtain the plot below:

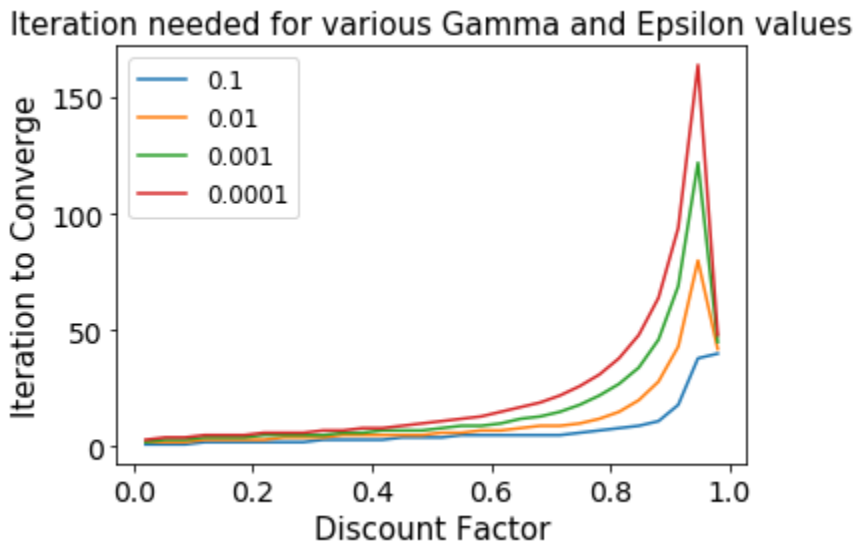

Figure 9:The iteration to converge vs different discount factors and different epsilon values for value iteration algorithm on problem 2.

The iteration needed to converge increases as Epsilon values become smaller or a larger discount factor is used; however, interestingly, at very high discount factor, the iteration needed sharply drops. This behavior is interesting and it is due to the problem set up (a randomly generated game). With a large discount factor, the agent is taking the future reward more seriously, and for this problem there is probably only few possible routes, resulting the agent to converge at high discount factors.

Compared to problem 1, this problem generally takes more iteration to converge due to larger amount of states.

Now let’s look at the optimal policy output for various discount factors:

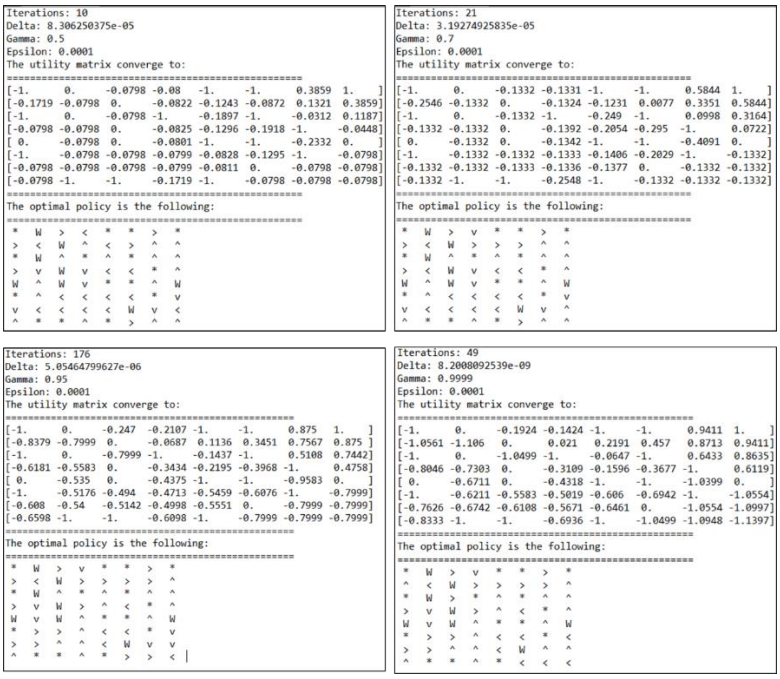

Figure 10: Example of optimal utility value and corresponding policy for different gamma value using value iteration algorithm. Top left, gamma = 0.5, top right: gamma =0.7; bottom left: gamma = 0.95; bottom right: gamma = 0.9999

At gamma = 0.5, the route is far from optimal. In fact, it does not even go to the charging station. When gamma = 0.95 and gamma = 0.9999, the optimal policy is basically the same.

Policy Iteration Algorithms:

In Problem 2, the iterations to converge vs. different discount factor and Epsilon values is shown below:

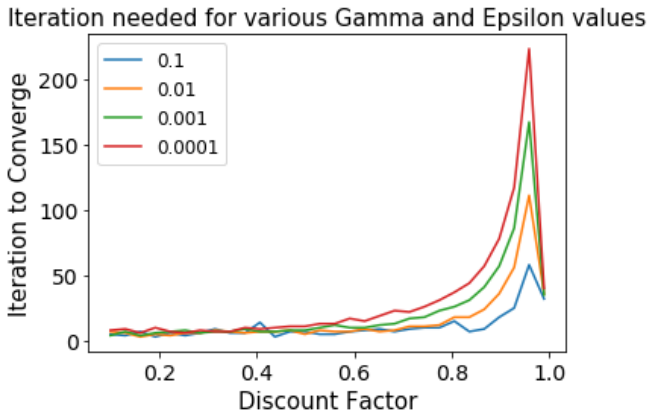

Figure 11: The iteration to converge vs different discount factors and different epsilon values for policy iteration algorithm on problem 2.

The behavior is generally the same as value iteration. As mentioned earlier, policy iteration typically takes fewer iteration to converge; however, for this problem, the iteration to convergence are almost the same except at very high discount factors, as shown in Figure 12:

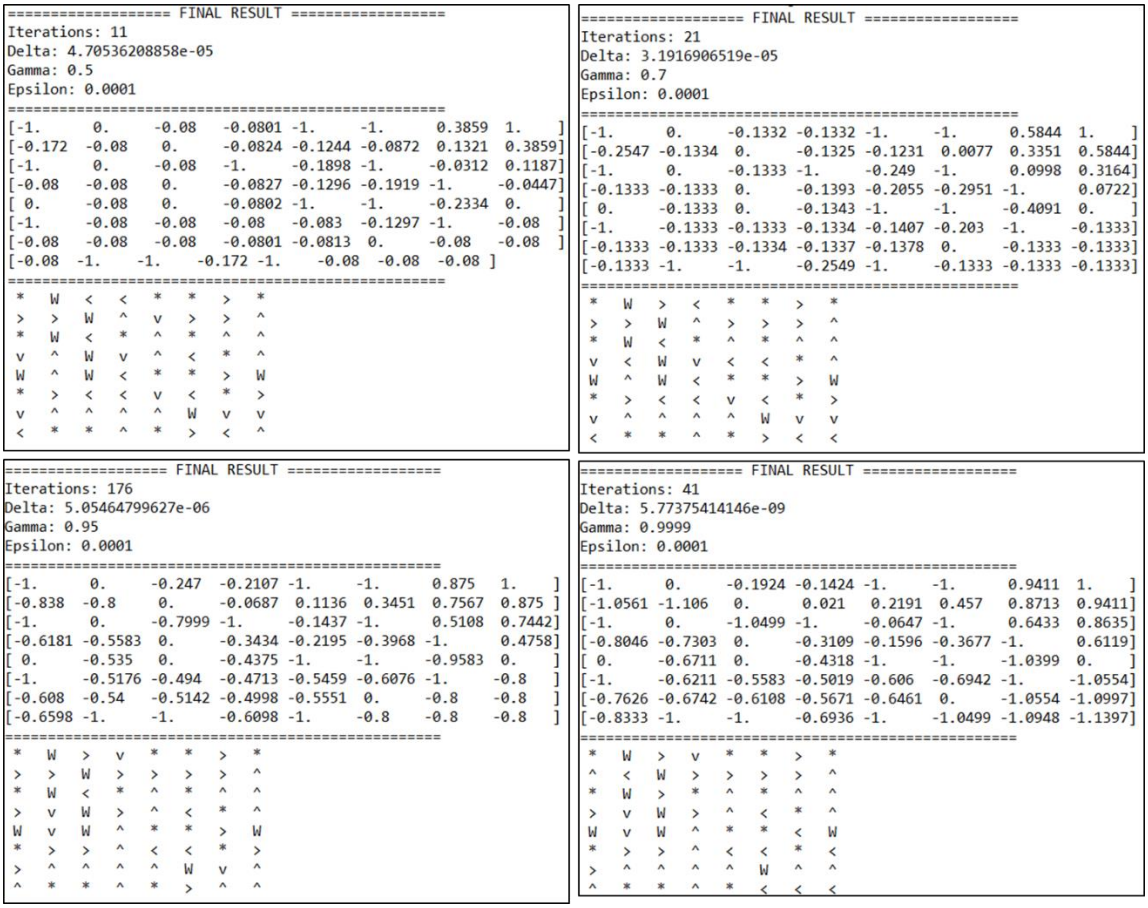

Figure 12: Example of optimal utility value and corresponding policy for different gamma value for the policy iteration algorithm on problem 2. Top left, gamma = 0.5, top right: gamma =0.7; bottom left: gamma = 0.95; bottom right: gamma = 0.9999.

However, the optimal policy provided by policy iteration seemed to be better than that of value iteration. If one looks at the [7,2] location (the right diagonal grid from the bottom left grid) for gamma value of 0.95, the value iteration suggests going right but the policy iteration suggests going up. For this specific problem, going up at [7,2] seemed to be the best because it avoids the possibility of going to the trap.

Compared to problem 1, this policy iteration algorithm generally takes more iteration to converge due to a larger amount of states.

Q-Learning:

Q learning algorithm for this 8X8 world is extremely time-consuming. I set the alpha to 0.1 and use a randomly generated exploration policy, it turns out that even after 750000 iterations (takes about an hour), the policy has not converged yet. Moreover, even at more than 750000, the policy is still very far away from the real optimal policy, as shown in Figure 13. In terms of exploration matrix, the conclusion is the same as problem 1: when the optimal policy is used as exploration matrix, the Q learning agent runs faster for each iteration compared to when a randomly generated matrix is used.

As a result, I conclude that the performance of Q learning algorithm, either speed-wise or policy-wise, is inferior to either value iteration or policy iteration.

```
Policy matrix after 799001 iterations:
 *   #   >   <   *   *   >   *
 >   >   #   >   >   >   ^   ^
 *   #   ^   *   <   *   ^   <
 <   <   #   <   <   <   *   ^
 #   <   #   v   *   *   ^   #
 *   >   ^   ^   <   <   *   >
 v   <   ^   >   ^   #   v   v
 <   *   *   ^   *   >   >   v
```

Figure 13: The policy matrix after about 790000 iterations in Q learning.

Summary:

For this assignment, I used three kinds of RL algorithm, namely, Value Iteration, Policy Iteration, and Q learning to solve a simple 3X4 grid problem and an 8X8 grid problem with more obstacles and more states.

For problem 1 (the 3X4 grid problem), value iteration takes longer to converge compared to policy iteration if the discount factor and the Epsilon converging criteria is the same. The iterations to convergence take longer if a large discount factor or a smaller Epsilon converging criteria is chosen. The utility values for grids closer to the absorbing states, which contains more extreme reward, converges slower compared to the grids further away from the absorbing states. At small discount factor, the converged policy is not optimal, while for larger discount factor, the converged policy is closer to optimal. For this problem, policy iteration takes less iteration to converge. If the discount factor and the epsilon values are set the same for both algorithms, then the obtained utility matrix and the policy are almost the same for both policy and value iteration. On the other hand, Q learning algorithm takes much longer to achieve equilibrium, but even at high iteration, the policy for some points still fluctuate a little bit from time to time. The policy obtained by Q learning is sub-optimal compared to value iteration and policy iteration. If the exploration policy is set to the "real optimal policy", then the speed for each iteration and the iteration to convergence are much faster compared to a randomly generated exploration policy.

For problem 2, most of the conclusions are the same as problem 1. The iterations needed for all three algorithms are larger due to larger problem size. In this problem, the iteration needed to converge for policy iteration and value iteration is the same. Q learning algorithm is extremely slow for such a problem size, and the policy after a large amount of iteration is still far from the real optimal policy

Reference:

Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,

Codes are modified from https://mpatacchiola.github.io/blog/2016/12/09/dissecting-reinforcement-learning.html

Thank you very much for your time!