Assignment 4: Markov Decision Process

Tian Mi, tmi7
CS 7641: Machine Learning

Introduction

In this project report, I conducted reinforcement learning experiments on two grid world problems. The methods I tested includes two planning algorithms of value iteration and policy iteration, and also a learning algorithm of Q learning. The algorithms will be tested on two grid world MDP settings and I'd like to compare both the policy output and the iteration/time required to converge. The two grid worlds were designed to be of very different sizes to get a more comprehensive comparison of algorithms performances.

About Markov Decision Process

A Markov decision process contains 5 tuples (S, A, P, R, γ):

S is a finite set of states;
A is a finite set of actions (A_s is the finite set of actions available from state s);
$P_a(s, s')$ is the probability that action a in state s will lead to state s';
$R_a(s, s')$ is the immediate reward received after transitioning from state s to state s';
$\gamma \in [0, 1]$ is the discount factor, which represents the decreasing of importance for future rewards.

With the MDP defined, the algorithms can be applied to generate policies based on the expected reward of a set of states or actions, making improvements, and iterating until improvements are no longer seen. MDPs are particularly useful in situations that require large sequences of actions, and where the actions are not entirely reliable and decisions are better made using expected value.

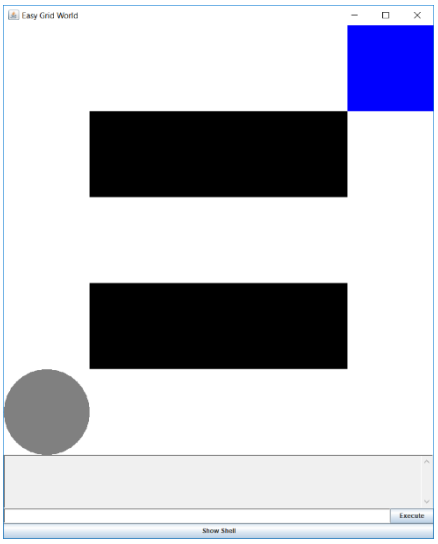
Implementation

The implementation of MDP problem and three learning algorithms was copied from public github repository and is based on the BURLAP grid world. The parameters of the grid world setting and learning methods includes the following. One thing that's important to my setting of MDP is the **transition probability** for each action. There is 80% chance an action will result in a state change follow this action, and there are 6.67% chance that the state will change to the other 3 directions. This kind of probability distributed action results can model the real world better.

Parameter	Values
Reward Function	100
Cost Function	-1
Terminal State	Top Right
Transition Probability	0.8 (correct), 0.066 (incorrect)
Discount Factor	0.99, 0.5, 0.1
Max Delta	-1
Max Iterations	100, 500

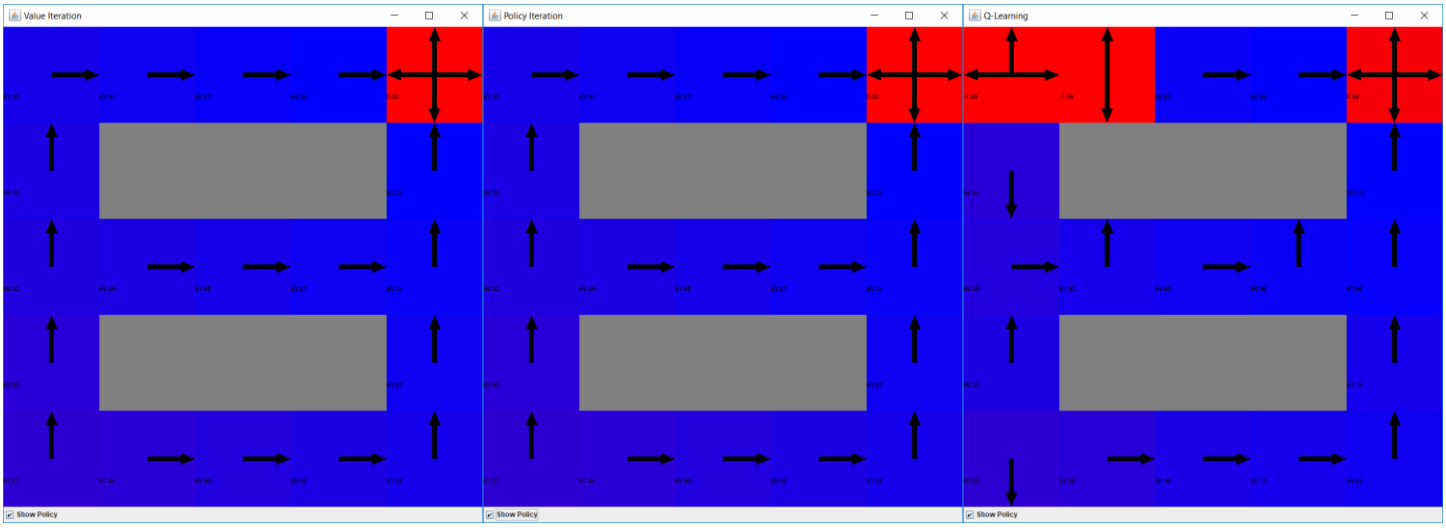
There are some additional parameters to consider for Q learning. The **Q_Initial** value is the value assigned to all states before updating the values, and the absolute values determine an initial Exploration-Exploitation Dilemma trade-off. Whereas larger Q_Initial value will encourage more exploration at the beginning and vice versa. The **exploration strategy** determines how the agent decides whether to explore or exploit. The decision process is a function of the approach (such as epsilon greed, random, or a form of simulated annealing) and the value of **epsilon ε** (a threshold for random generation below which an agent chooses to explore). Lastly, the **learning rate** is a relative measure of the importance of recent and older information when an agent makes a decision. Because the above three parameters all control the same Exploration-Exploitation trade-off, I only use the default values for easy comparisons.

Grid World Problem 1

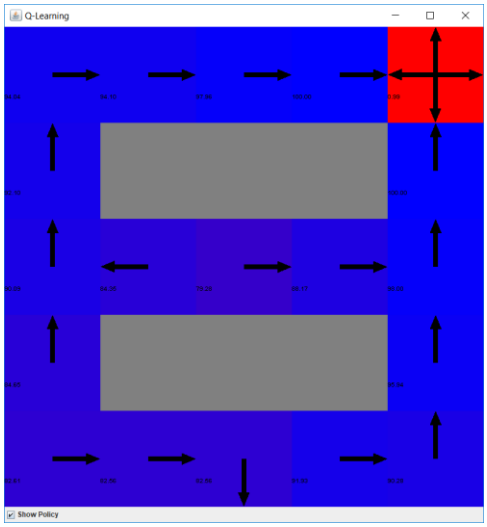


The setting of my first grid world is very simple. It's in a 5X5 grid space and contains three equally rewarded routes starting from bottom left end to top right end. A main focus of this grid world is to see if the utilities output of the three routes will be the same. I can also evaluate some other behaviors of the three algorithms, such as how many iterations it takes to converge, the clock time of each iteration and will the optimal policy change for each run. Although the total rewards of the three routes are the same, I would expect a unique choice of action at each intersection point due the probability distribution of action results. Take point (1,3) for example, there is a larger than 0.066 chance the actual action will be go down if I choose to go right, but this chance is much smaller if I choose to go up. The choice of decision by different algorithm would be different at such intersections. I will also change the iteration time and discount factors to see if the optimal policy changes.

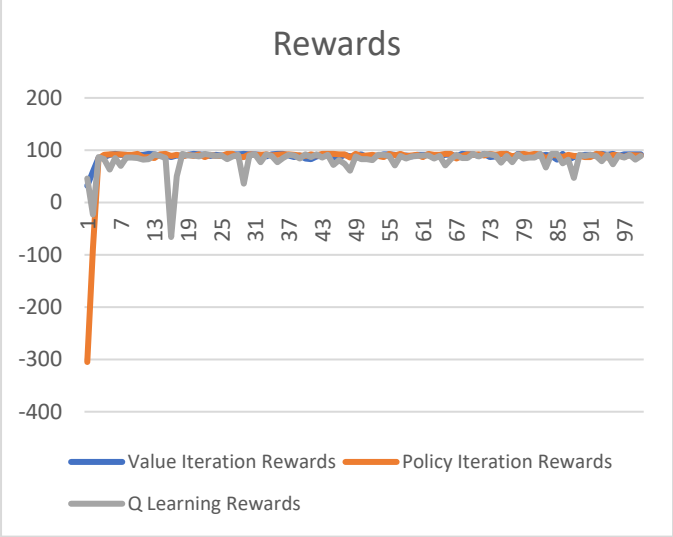
The optimal policies after 100 iterations for the three methods are:

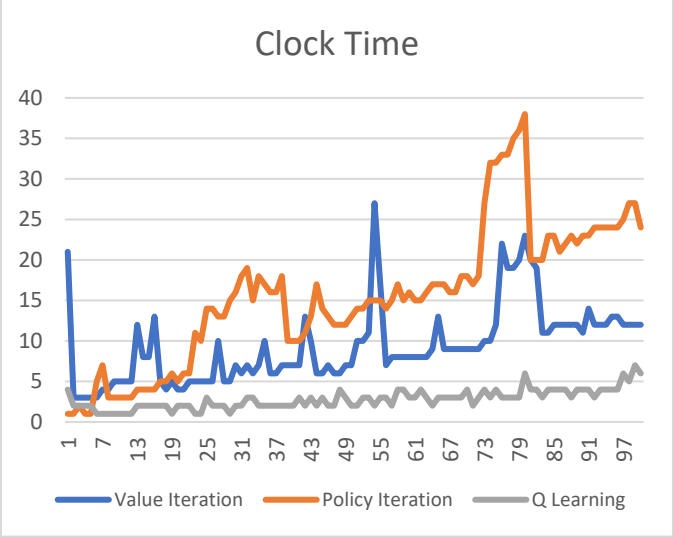


The outputs of value iteration and policy iteration are exactly the same, both of them show the "true" optimal policy given the reward function and domain knowledge. But the output of Q learning is quite different, the policy didn't identify the correct action for at lease 5 states. If I rerun the same Q learning setting, the policy output came quite different and all of my test run results have non-optimal actions. This seems to show that 100 iterations is too few for Q learning and the algorithm didn't really converge despite the reward function didn't change much. If I choose to run 1500 iterations, the result policy shown to the right is more like an optimal policy. The general trends of the three routes were formed and the final rewards are very close to optimal rewards. But the choices of action are not correct for two states of (3,1) and (2,3). The action choice at (2,3) can be explained since it's obvious that (1,3) has a much higher reward than (3,3), and I think this was because the QL algorithm explored one route more than another route. If the QL started by visiting some states more often than others by chance, then there is a large chance the output policy will also be biased. This can be seen as a problem of QL policy outputs.

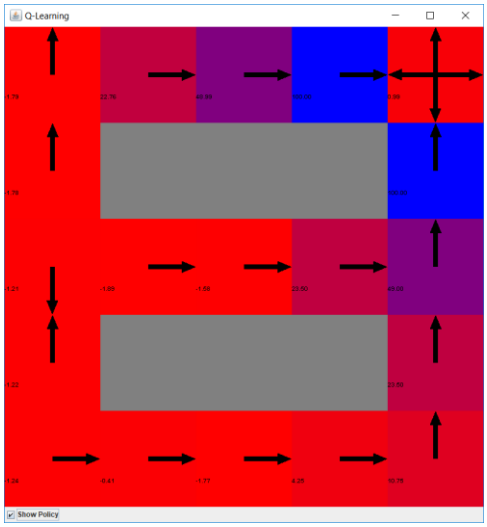


During the 100 iterations of all three algorithms, I kept track of the rewards of each iteration and the clock time it took for each iteration. The plots of these two metrics are the following:





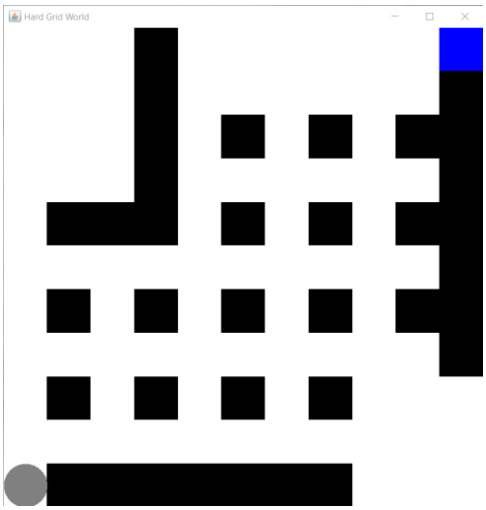
All three algorithms reached larger than 90 final rewards within 7 iterations. Even the Q learning run didn't calculated the correct rewards for each state, it's fairly simple to identify one optimal route for all algorithms. I felt like that's one drawback of my grid world design, that is there is no circling routes that may trap an algorithm and result in a region of very small reward values. Even through value iteration and policy iteration can find all three optimal routes within 100 iterations, the actual run time those two methods took are much longer than Q learning. In general value iteration takes around 3 times the time of Q learning and policy iteration takes twice as much as value iteration. Though we didn't feel much difference in this simple grid world problem, the computation simplicity of Q learning may be ideal for many real world complex problems. The small and sometimes large spikes in time are due to the probabilistic nature of the agent's actions, and we can see that Q learning again avoids any large spikes because the fast iteration time.



I also tried to modify some of the Q learning parameters like discount factor and learning rate. If I just change discount factor to 0.5, the algorithm still struggled to update the reward values of most states after 1500 iterations. As we can see in the policy output to the right. The same situation happened to learning rate as well. It's very obvious that setting 0.99 learning rate and 0.99 discount factor can provide best performance for this grid world setting. Which means more exploration here help all three algorithms to find optimal paths and converge much faster, and there is no need to do exploitation here given all three optimal routes can be easily identified. I'll continue to use the same parameters for my next grid world for all these benefits.

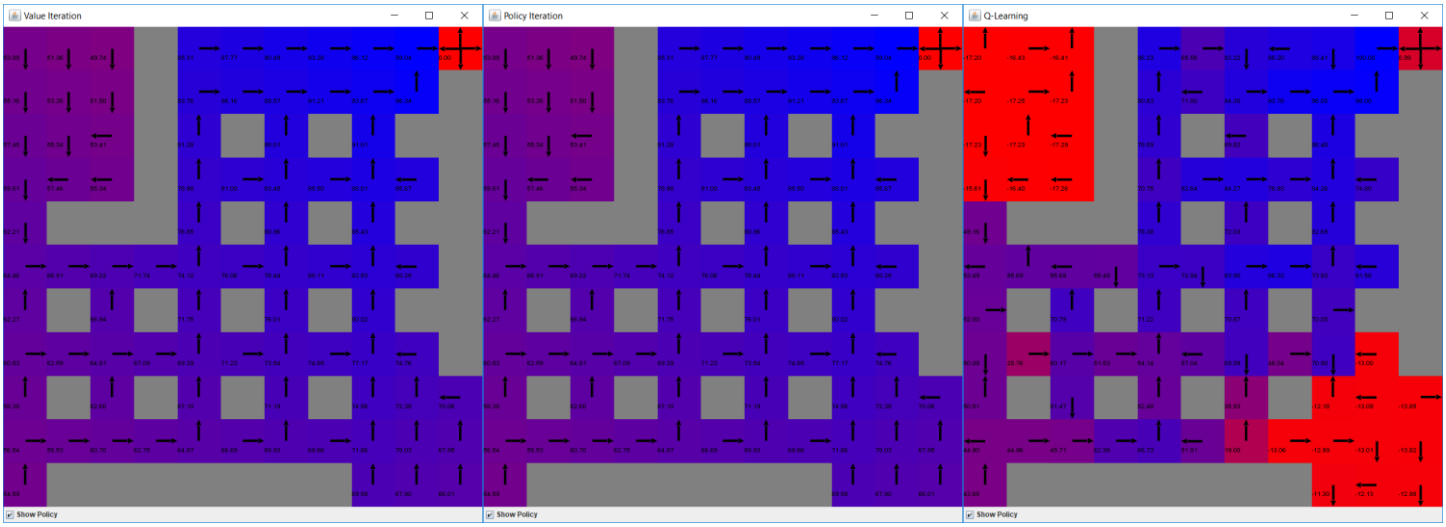
To conclude the pros and cons of these three methods. I would say planning based methods converge in very few iterations and can provide the correct reward values after some iterations. But the run time of one iteration can be very slow due to the probability distributed action results, it's even more slower for policy iterations. The learning based method don't know prior knowledge of the model and can explore to a near optimal solution very fast. Also, the run time of Q learning method is nearly linear to the number of iterations, that should be a much desired property for more complex problems.

Grid World Problem 2



The goal of my second grid world design is to include more alternative routes and add a dead end region to see what reward values each agent will give to that dead-end region. The actual design of my GW is to the right. It's a 11X11 grid world with multiple alternative routes from left bottom to top right. And in the top left section, I reserved a 4X3 space as a mostly sealed dead-end route. It would be interesting to see how different methods handle this section, will they just skip this part and kept the original low rewards all along or some algorithm will get inside here repeatedly and assign relatively high rewards here. I also add a smaller 3X2 free space to the bottom left, the rewards and policy results here maybe will have different properties than the dead-end section. Because this free space and the dead end space are on symmetric spaces with respect to the starting point, I think it's fair to compare the actual reward values within the two sections.

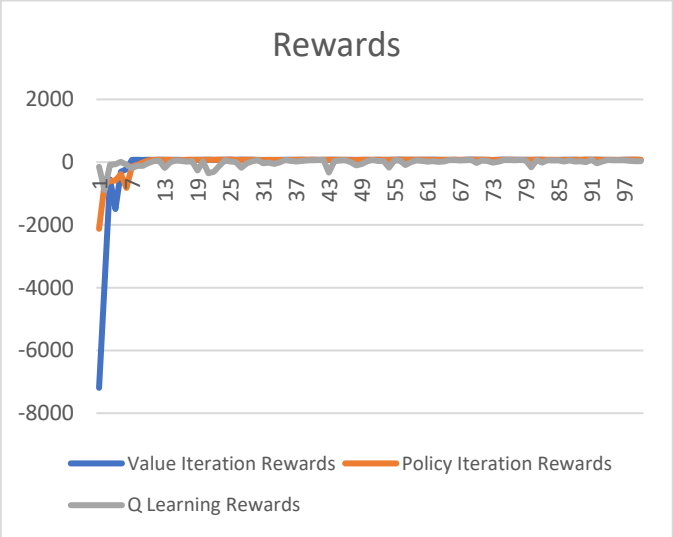
The final policy results after 100 iterations are:

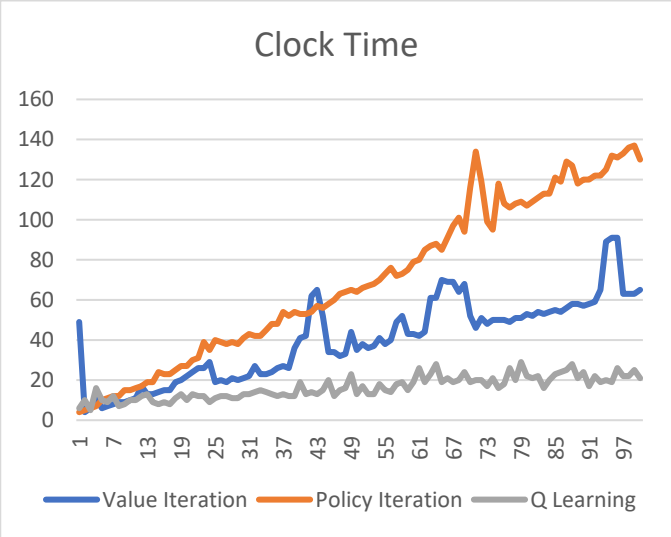


Once again, value iteration and policy iteration reached the same 'optimal' policy result. It seems that 100 iterations are more than sufficient for these 2 algorithms. In general, reward values are around 50 in the dead end section, while they are higher at around 70 in the free circling space. This agrees with my expectation that the dead end space is less preferred compared to the free space. The action that both policies took at the entrance state (8,2) of the free space is also very interesting. Both policies choose to enter the free circling space instead of going back, which makes sense because going into the free space is actually making process towards the final state as we can see. But on the entrance point of the dead-end space at (1,7), the policies choose to go back because they can only waste their moves inside that section.

As for Q learning final policy. Both the dead-end space and free space still have negative Q values after 100 iterations. This shows that QL only visited these two sections in the first couple iterations, and it abandoned those states quickly. That's a very nice feature for QL considering it didn't know the model and rewards function. The overall trends of Q values identified most correct routs along the way. But if we compare all the actions QL take with the optimal policy of VI results, a lot of the actions QL chose didn't make sense. Judging from my experience of the first grid world, QL needs a lot more iterations to achieve a nearly optimal final policy.

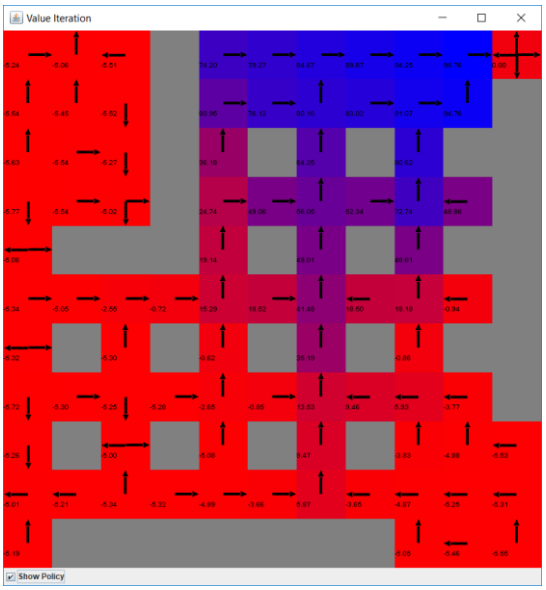
The rewards and clock time follow exact similar pattern as the first grid world. Only Value Iteration have much lower starting rewards in the first 7 iterations. That is reasonable given the methods need more trial and error in the beginning in this larger grid world. As we can see in the rewards plot, all algorithm reached convergence in less than 7 iterations, meaning it's still pretty easy to find an optimal solution in this MDP. I've tried to implement the most complex 2D grid world I can think of, I guess I need to use a multi-dimensional MDP to see the difference of how fast each method are. The clock time plot also identified Q learning as the most time sensitive methods that only requires less than 1/5 of policy iteration's time.

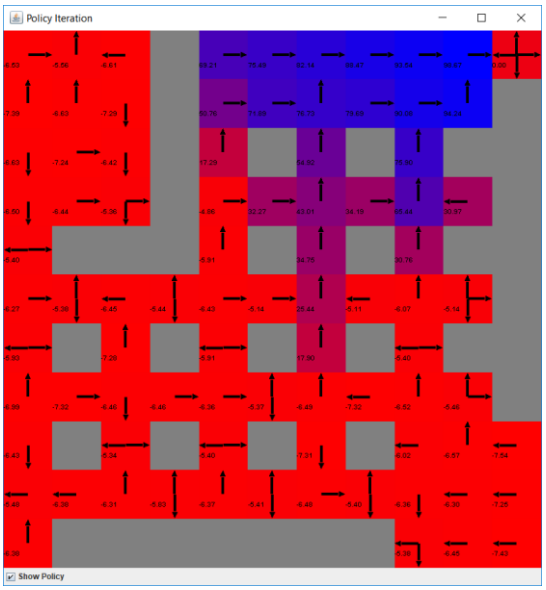




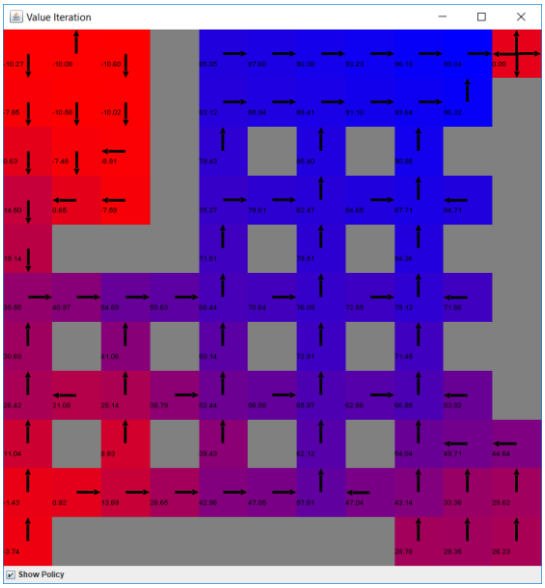
Given that the two of my grid world design gave very similar results, I thus want to explore more of the comparison between value iteration and policy iteration. If I take the policy snapshot at iteration 5 and 10 for theses two algorithms, the results look like:

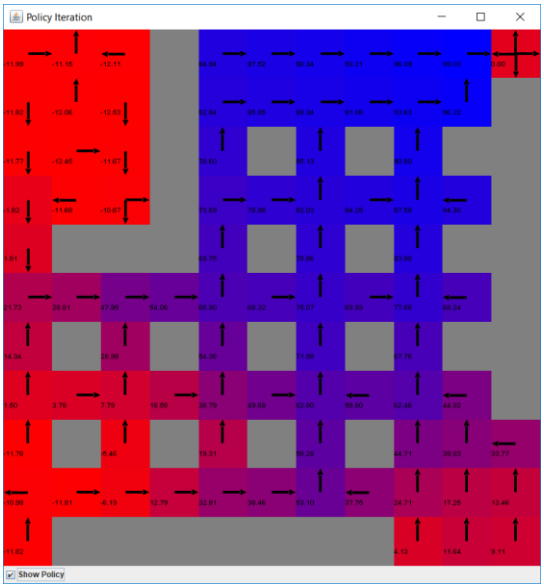
Iteration 5:





Iteration 10:





These plots also show the huge similarity between value iteration and policy iteration. I can only find some difference when comparing the actual rewards. The VI rewards are mostly larger than PI ones, which means they are most close to the final reward values. So by this measure, VI converges faster than PI while using less clock time all the way.

Conclusion:

After running these three methods in the two grid worlds, I'm most impressed by the performance of Q learning. It uses much less computation resources while identifying the optimal solutions as fast as the others. Q learning can also handle the useless information of the model best thus save even more resources. And all these benefits were built upon no prior knowledge of the model and rewards. To me, Q learning is the most suitable technique for real world problems. As for VI and PI, their advantage is to guarantee an optimal policy result after enough iterations. And I would prefer VI here since it's runs and converges faster than PI.