# Assignment 4 - Markov Decision Processes and Reinforcement Learning

Arpitha Dudi
(GT ID- 903272081)

## Abstract

This report analysis few techniques of reinforcement learning to an agent to make decisions. Reinforcement learning is an area of machine learning which places an agent into a world with defined rules, and allows the agent to learn based on the stimuli (penalties, rewards) he receives from the world. Here we explore two interesting Markov Decision Processes (MDPs) using two planning algorithms, Value Iteration and Policy Iteration, and one learning algorithm of choice, Q-Learning.

## Implementation

The implementation of the MDPs along with the analysis was done using code written in Java that used the BURLAP Reinforcement Learning package. The behaviors of the three algorithms were explored, using various parameters to observe the impact on the convergence, computation time and policy. Each experiment was run 3 times, and the average is reported.

## Algorithms

### Value Iteration

Value iteration is a planning algorithm**.** The essential idea behind value iteration is this: if we knew the true value of each state, our decision would be simple: always choose the action that maximizes expected utility. But we don't the initial state's true value; we only know its immediate reward. But, for example, a state might have low initial reward but be on the path to a high-reward state.
The utility of a state is the immediate reward for that state, plus the expected discounted reward if the agent acted optimally from that point on:

$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U(s')$

Note that the value for each state can potentially depend on all of its neighbors' values. If our state space is acyclic, we can use dynamic programming to solve this.
Otherwise, we use value iteration.

1. Assign each state a random value
2. For each state, calculate its new U based on its neighbor's utilities.
3. Update each state's U based on the calculation above: $U_{t+1}(s)=R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_t(s')$
4. if no value changes by more than δ(max delta), halt.

This algorithm is guaranteed to converge to the optimal (within δ) solutions but this takes a long time to converge in specific situations

### Policy Iteration

Policy iteration is another type of planning algorithm that starts with a policy and iteratively tries to improve it in order to converge. It tries to computes the optimal policy as follows.

1. Create a random policy by selecting a random action for each state.
2. While not done:
   a. Compute the utility for each state given the current policy.
   b. Update state utilities
   c. Given these new utilities, select the optimal action for each state.
3. If no action changes, halt

**Q-Learning**

Q-learning is a reinforcement learning method tries to find an optimal action/selection policy for an MDP.

The way this works is as follows: we assign each state an estimated value, called a *Q-value*. When we visit a state and receive a reward, we use this to update our estimate of the value of that state.

Our Q-value can be written as:

$Q(s, a) = R(s, a) + \gamma \max_{a'} Q(s', a')$

This says that the value of taking action (a) in state (s) is the immediate reward for the (s,a) pair, plus the value of the best possible state-action pair for the successor state.

We can use this to update the Q-value of a state-action pair as a reward r is observed:

$Q_{t+1}(s, a) = r + \gamma \max_{a'} Q_t(s', a')$

## Grid World

Grid World is a 2-dimentional plane in which an agent is present. The job of the agent is navigating the plane to reach the final destination by avoiding the walls. In order for the agent to navigate it has to perform 4 actions in any cardinal direction. The objective of the agent is to maximize the net reward by reaching the terminal state in the fewest number of actions and once it reaches the final destination, it gets a large reward (like 100). There is an actions probability of success that the agent will be able to move successfully in the direction that it intended. For example, if the agent wants to go down, there is an 80% chance the agent will actually go Down, but a chance to go Up, Left or Right (the remaining 20% is split equally 3 ways). The agent's priorities of the present and future, a discount factor is applied to the expected value of each action. For Value Iteration and Policy Iteration max delta was used for the algorithms to end when the delta in utility between consecutive iterations is below a threshold. For Q-Learning, the parameters to consider are learning rate which is the measure of the importance of recent and older information when an agent makes a decision and epsilon ε a threshold for random generation below which an agent chooses to explore. Q_Initial value is the value assigned to all states before updating the values with "truth". When Q_Initial is high, this initially makes all states viable for the agent to pursue, whereas when Q_Initial is low, all states are less viable and thus the agent is encouraged to exploit more than explore. The exploration strategy determines how the agent decides whether to explore or exploit.

Here we explore two types of grid world a 10x10 grid (small and low difficultly) and 11x11 grid (large and high difficult).

## Grid World (Small and Low Difficulty)

Here the algorithm is applied to a 10x10 grid with a low difficult problem which is generated randomly as shown in the Figure 1. The grey agent starts at (minX, minY) and tries to move to the blue target (maxX, maxY). The black sections represent the walls which the agent should avoid. The cost per step is -1, the terminal reward is 100 and the actions are probabilistic. By default, the discount is 0.99 and the experiment was done for 200 iterations.

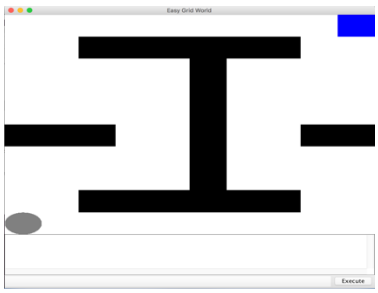
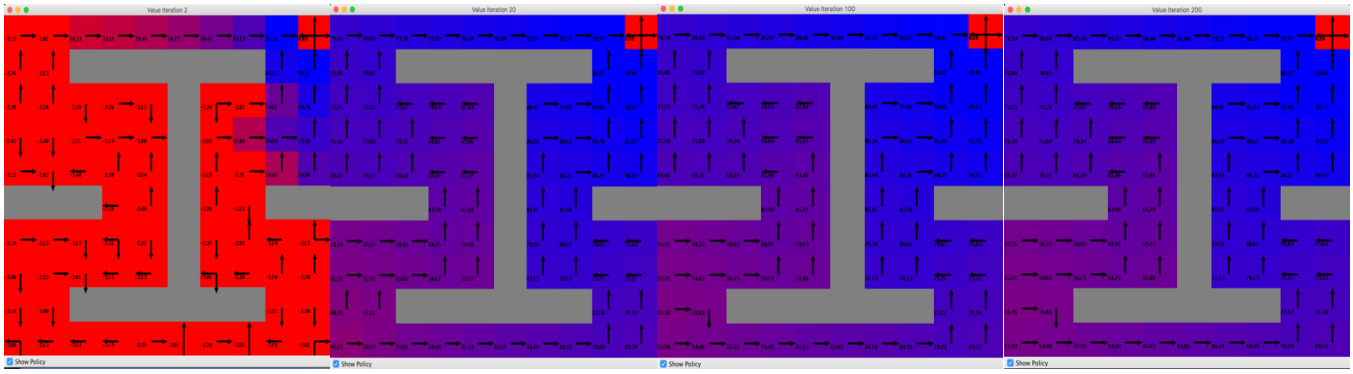



Figure 1: Grid world

**Value Iteration(VI)**

From Figure 3, Reward graph above shows that that Value Iteration had a large negative utility for the first 6 iterations. This is also reflected in the number of steps (>200) in the initial process before converging very quickly at I=10. Also the convergence(delta) and the number of actions correlate, which demonstrates the model behaves as expected and after the drop it remains at a constant level.

The running time grows linearly with the number of iterations, which is expected since the time will increase linearly with the increased number of actions. The spikes in time are again, due to the probabilistic nature of the agent's actions. The policy constructed using VI is shown in Figure 2 for Iterations 1, 20, 100 and 200. At I=2, the nearest states to the reward are evaluated for utility. VI continues to evaluate nearby states and at I=20, here VI first ensures the agent avoids walls before refining the specific path it should take. There not much change between I=100 and I=200.



I = 2 I=20 I = 100 I = 200

Figure 2: Policy Maps using VI for easy world

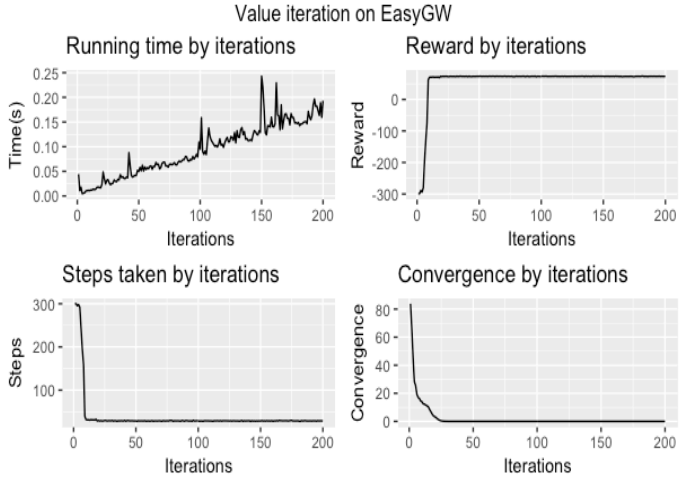



Figure 3: Parameter Maps for VI

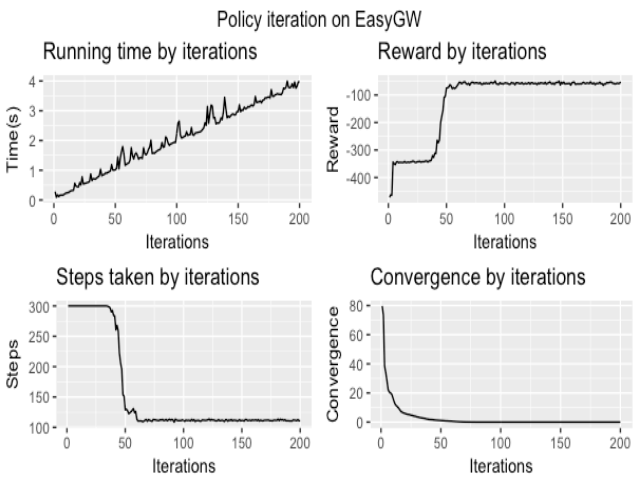
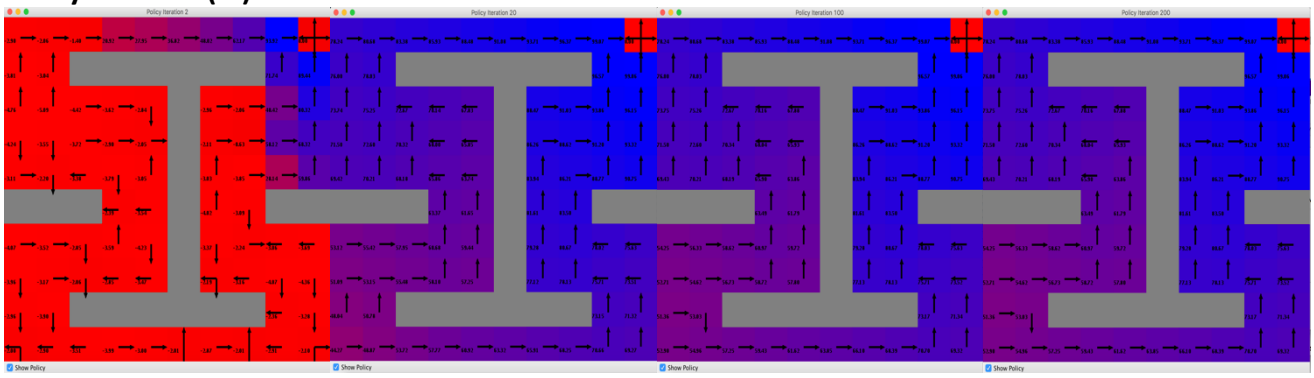



Figure 5: Parameter Maps for PI

**Policy Iteration(PI)**



I = 2 I=20 I = 100 I = 200

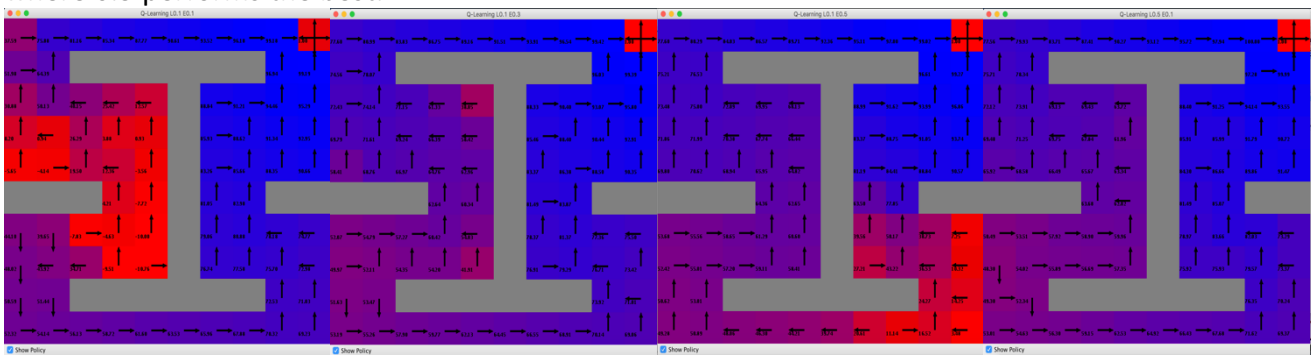
Figure 4: Policy Maps using PI for easy world

Reward graph is as shown in Figure 5 Policy Iteration converged at I = 12 which is slightly slower compared to VI which was at I=10. Also PI takes longer to perform iterations compared VI. While this is expected that expectation of PI takes longer per iteration, it is also expected that PI takes fewer

iterations because PI evaluates the utility of each action pair for a given policy, which takes more time to compute but provides more information to learn from. Here it takes additional 2 iterations more than VI which may be due to the probabilistic nature or the initial states assigned for first iteration. If we compare the results of PI and VI it behaves almost the same. The policy graphs for I >= 100, look nearly the same as shown in Figure 5.

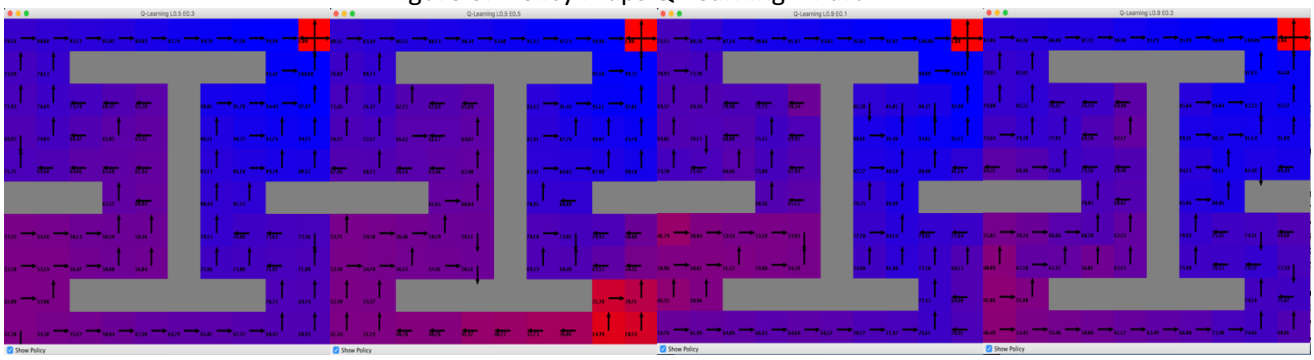
## Q-Learning

After applying two planning algorithms to the problems now we move on to applying Q Learning to solve it. For analysis, I ran Q Learning for learning rate (LR) value of [0.1,0.5,0.9], Epsilon (ε) value of [0.1,0.3,0.5] and for 10000 iterations. Consider Figure 6, Figure 7, Figure 8 where LR is constant at 0.1 indicates almost no learning done and Epsilon is varied. We can say the utility value increases form ε = 0.1 to ε = 0.3 due to the fact that at ε = 0.1 it does no exploring just becomes greedy might get stuck at local minima. The opposite happens when ε = 0.5 where it only explores and doesn't exploit at all. When the LR is constant at 0.5 it basically takes the average of the current and new Q-value. The utility value slight increase form ε = 0.1 to ε = 0.3 and at ε = 0.5 it performs badly. When the LR is 0.9 almost all the ε values perform similar where 0.5 performs the best.



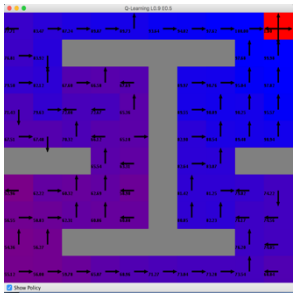
LR = 0.1 ε = 0.1 LR = 0.1 ε = 0.3 LR = 0.1 ε = 0.5 LR = 0.5 ε = 0.1

Figure 6: Policy Maps Q-Learning – Part1



LR = 0.5 ε = 0.3 LR = 0.5 ε = 0.5 LR = 0.9 ε = 0.5 LR = 0.9 ε = 0.3

Figure 7: Policy Maps Q-Learning – Part2



LR = 0.9 ε = 0.5

Figure 8: Policy Map

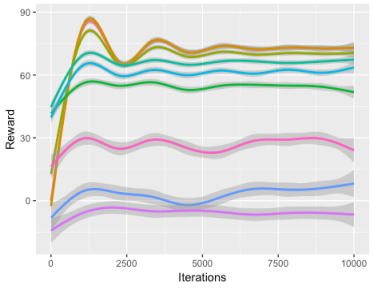



Figure 9: Reward Map

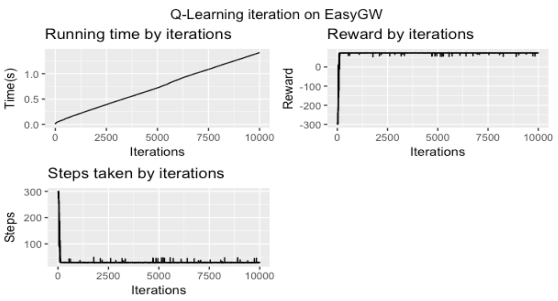



Figure 10: Parameter Map

From Figure 9 is a graph of reward w.r.t to iteration we can say that LR:0.1 and ε:0.3 out performs all other combination. Figure 10 running time graph compared to VI and PI is much less but number of iteration is large due to the fact that Q-learning algorithm must visit all the states as many times as possible for it to find an optimal policy. We can say that the algorithm convergence around I=110 which almost 10 times more than VI and PI.

## Grid World (Large and High Difficulty)

Here the algorithm is applied to a 40x40(1600 states) grid with a high level of difficulty which is generated randomly as shown in the Figure 11. This grid has walls for a total of 320 possible states it also a negative cost function of -2 for 160 states and the rest has a cost function of -1. The agent begins at the front of the store at (minX,minY) and it finishes shopping when it reaches the terminal state at (maxX,maxY) gets a reward of 100 here.

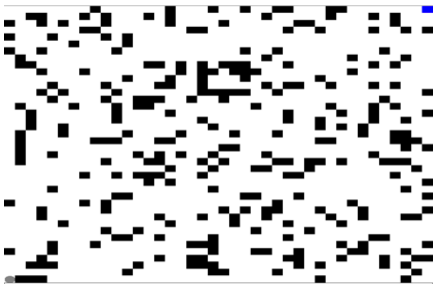
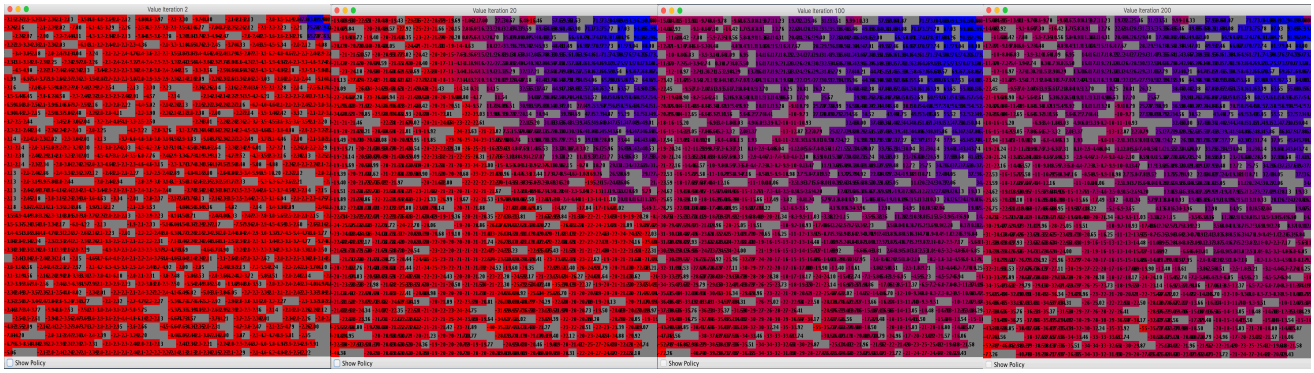


Figure 11: Grid world

## Value Iteration



I = 2 I=20 I = 100 I = 200

Figure 12: Policy Maps using VI for easy world

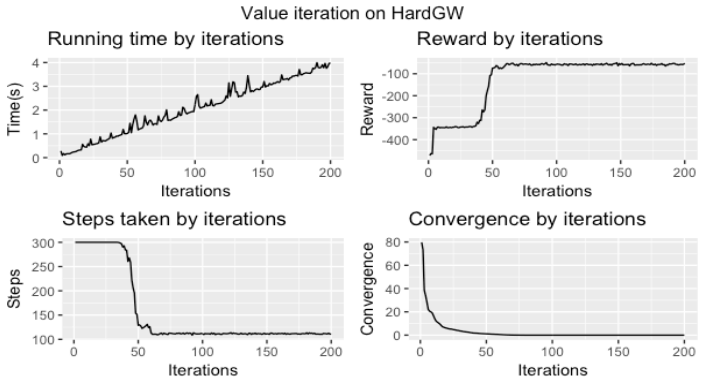



Figure 13: Parameter Maps for VI

Figure 14: Final optimal policy

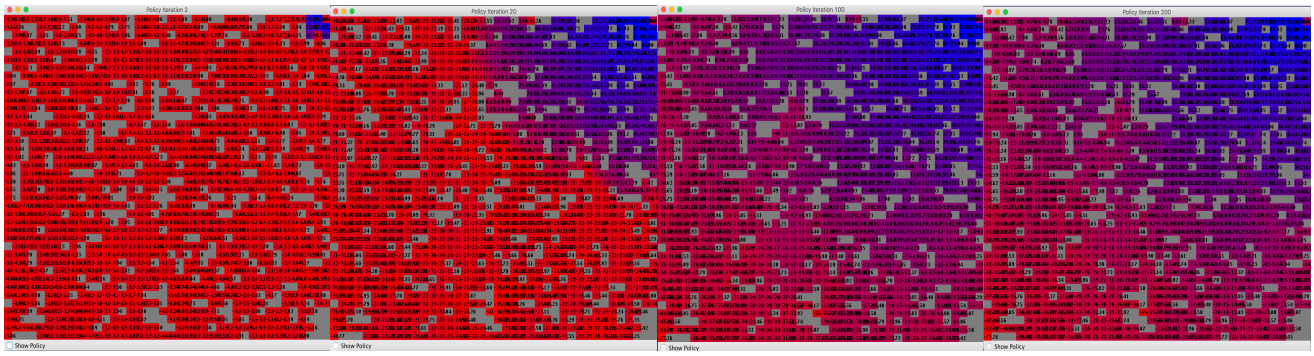
From Figure 13, reward graph shows that Value Iteration had a large negative utility for about 30 iterations which in kind off expected due non-updates of policy based on expected values. We can see that there are two sharp increases in the reward value one at 8$^{nd}$ iteration and the another around the 45$^{th}$ iteration. It finally convergence around I = 50 and after that the reward value is almost constant.

The running time lineally increases with iterations and the time value till I = 50 is less than a sec after it increases till 4sec. This might due to the fact that VI must make calculations for each action in the policy.

The small wavering along the trend line is indicative of the probabilistic actions of the agent. The policy for I=1, 20, 100 and 200 were constructed and are shown in Figure 12. At I = 2 we can see that only small number states at the terminated states are converged (blue color). VI continues to evaluate nearby states and at I=20, here VI first tries to explore the states near to the goal. There not much change between I=100 and I=200. Figure 14 shows the final optimal policy for VI.

**Policy Iteration**

Reward graph is as shown in Figure 16 Policy Iteration converged at I = 50 which is same as VI. Also PI takes longer to perform iterations compared VI almost double the time. The reward value graph range was increased by 100 compared to the VI. If we compare the results of PI and VI it behaves almost the same. Also the convergence(delta) and the number of actions correlate, which demonstrates the model behaves as expected and after the drop it remains at a constant level. The policy graphs as shown in Figure 15 for I >= 100, and the optimal policy as shown in Figure 17 looks nearly the same w.r.t VI.



I = 2 I=20 I = 100 I = 200

Figure 15: Policy Maps using VI for easy world

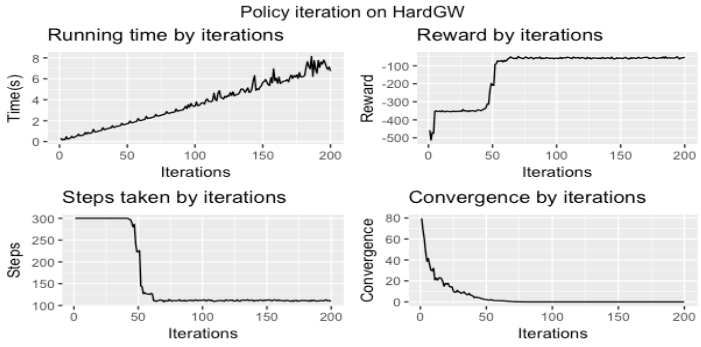



Figure 16: Parameter Maps for VI

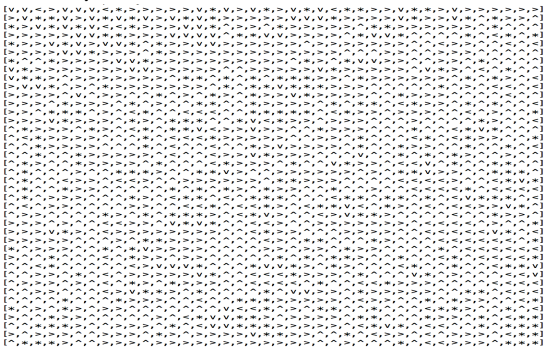
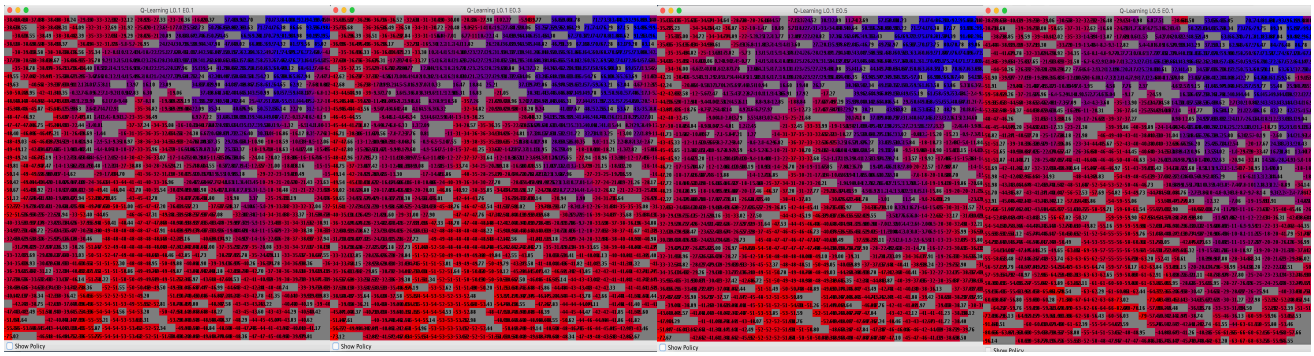


Figure 17: Final optimal policy

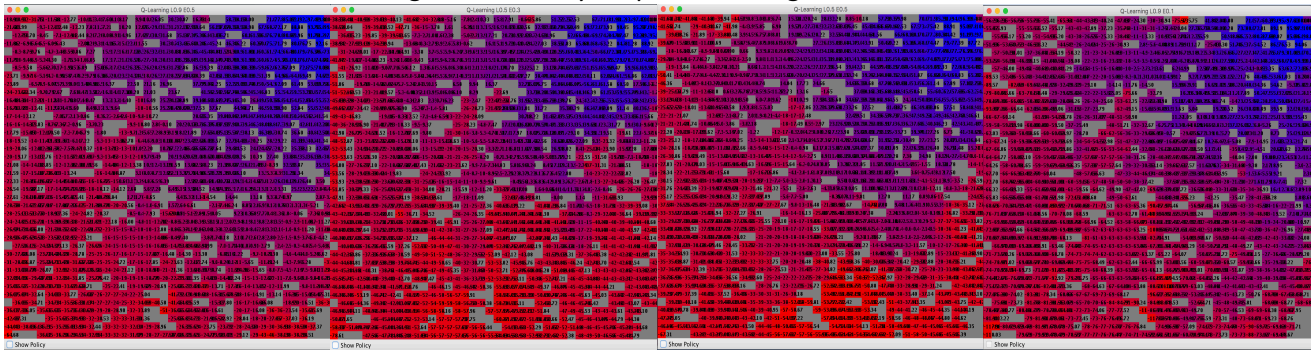
## Q-Learning

For this analysis again, ran Q Learning for learning rate (LR) value of [0.1,0.5,0.9], Epsilon (ε) value of [0.1,0.3,0.5] and for 10000 iterations. Figure 18, Figure 19 and Figure 20 shows the Policy maps for all the different configurations. Couldn't find much difference using all the policy maps so plotted the reward w.r.t iterations for all combinations as shown in Figure 21. One thing to note is that reward is negative for all combination and is never positive. Compared to VI and PI Q-learning performed the worst for this grid. Compared to all LR:0.1 and ε:0.1 performed slightly better than the rest which converges around I= 4000. Figure 22 shows the variation in learning steps and running time w.r.t. to iteration it can be seen through pulses that there is some randomness in the steps and never smooths out. It might due to fact that the algorithm has not yet learned enough.



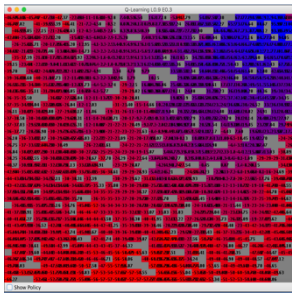
LR = 0.1 ε = 0.1 LR = 0.1 ε = 0.3 LR = 0.1 ε = 0.5 LR = 0.5 ε = 0.1

Figure 18: Policy Maps Q-Learning – Part1



LR = 0.5 ε = 0.3 LR = 0.5 ε = 0.5 LR = 0.9 ε = 0.5 LR = 0.9 ε = 0.3

Figure 19: Policy Maps Q-Learning – Part2



LR = 0.9 ε = 0.5

Figure 20: Policy Map

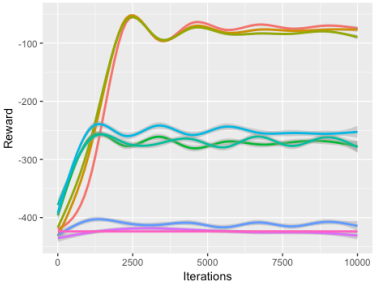



Figure 21: Reward Map

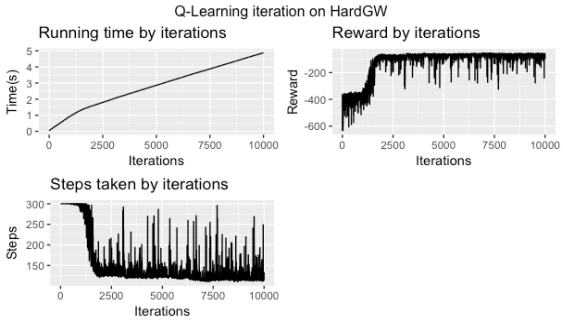



Figure 22: Parameter Map

## Discount factor variation

The default discount factor was 0.99, which makes the agent value future rewards almost the same as present rewards. Here we vary the discount factor to see the effect on the convergence. As shown in Figure 23 for small grid problem the Q-learning parameters where LR:0.1 and ε:0.3. Figure 24 shows the effect of discount fact for the hard grid. We can say that convergence occurs slightly faster for high discount factors irrespective of the grid size, due to the fact that the agent can explore different policies a little more if the final reward is not worth as much.

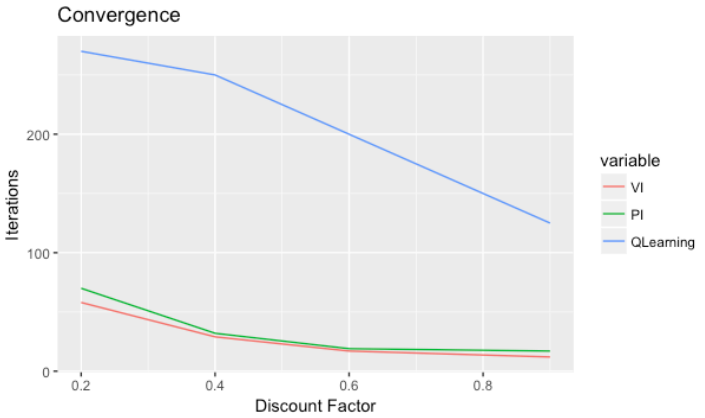



Figure 23: Small Grid: Discount factor

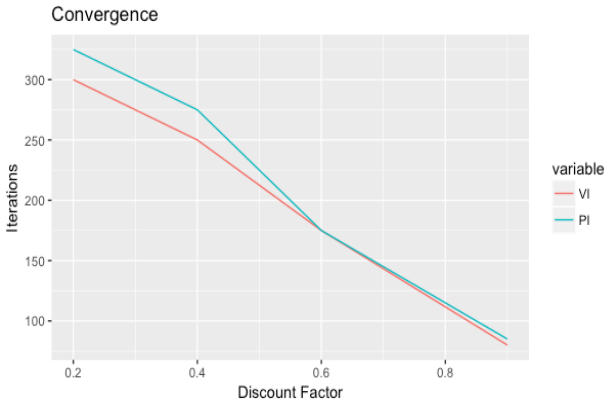



Figure 24: Small Grid: Discount factor

**Conclusion**

Value Iteration performs better than Policy Iteration because they converge to the same policy but Value Iteration converges faster and computes faster per iteration. Q-Learning computes the fastest of the 3 algorithms, but due to its greedy nature the ability to explore was not used for the QL converged reward.

**References**

1. http://uhaweb.hartford.edu/compsci/ccli/projects/QLearning.pdf
2. https://github.com/peridot121/omscs-cs7641-machine-learning-assignment-4/pull/1.
3. http://www.dudonwai.com/ai/artificial-intelligence