Markov Decision process

Value Iteration | Policy Iteration | Q-learning

1. Introduction:

1.1 Algorithm explanation

1.1.1 Value Iteration vs. Policy iteration: For understand their difference, we first should understand what they are. Basically, they are used in MDP to help us find the best path. Policy is telling us what we should do in order to gain the max reward. For example, we should go left to earn 2 points reward instead of go other directions to earn nothing. In this way, the policy iteration is we keep updating the entire policy map by each iteration and eventually we will get the best policy for each move. Now, the value iteration is very similar to policy iteration. In value iteration, we will update numbers in each state instead of updating the action choices. For example, during the first iteration, the values in the map is 1, 2, 3. Then in the next iteration, they may become 2, 2, 3. Now, we should have a good understanding of two of them.

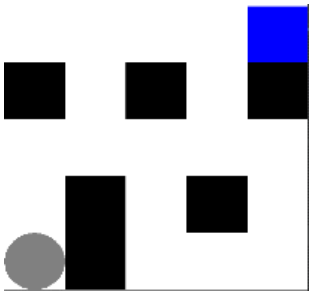
1.1.2 Q-learning: Knowing that there is the Q-function: Q(s, a). And since we've already known the value iteration and policy iteration. In general, using Q-function to express the value iteration and policy iteration and work on the reinforcement learning, this process is so-called q-learning. Typically, q-learning is very easy to implement compare to the rest other algorithm.

2. Two interesting MDPs

2.1 Introduction of MDP world

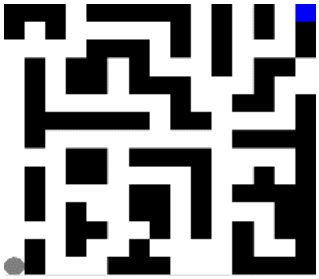
Recall what we've learned in CS3600 about MDP. We start with the grid world where different states locating in a 2-dimensional coordination system. Specifically, in this grid world, we can make a move and maybe receive some rewards (positive or negative). And sometimes there are walls in this grid world so that we may not move through the wall. Now I think we all clear about what the grid world looks like. In this assignment, I will still use the grid world as the basic structure of my two "interesting MDP problems". They are interesting because they will show us how MDP would work in the real world and let us know how does MDP would help us solve the problem we met. And more importantly, since they are grid, it's very easy to understand what happen inside of them. There is one grid world we are going to call it the easy grid world, and in this way the other one would be the hard grid world.

2.2 Easy grid world



The easy grid world is made of 5 x 5 grids. The blue grid is the goal state, the black grids are the walls. We want to find the unique shortest path from the start state to the goal state. If the state is the goal state, then we will get the reward: 100, otherwise we will get -1. All the setup is clear enough and the calculation will be easier and faster when there is only rewarding in the goal state. This is a easy grid problem not only the volume of combinations of paths is small, but also the path from the start state to the goal state is pretty straight forward, means we won't have to keep go back and forth just like in the maze.

2.3 Hard gird world



In general, the hard grid world is not quite different from the easy grid world. The basic rules are simple. In the hard grid world, there just exists more different paths, but still we should be able to find the shortest path from all possible paths. This grid world is closer to the real maze so that the calculation run and running speed will bigger than the easy grid world. The hard grid world is made of 15 x 15 grids.

2.4 Grid world explanation

We can treat the grid world as a record of policy. For example, if you see blank, then it means you can move to this state. If the state is blue (which you will see later), then it means you should go to this state. If the state is red, then you should avoid this state because you will not get any better if you go this state.

3. Algorithm implementation:

3.1 Burlap library

This is the instructor provide library written in Java. In this library, I could make my own grid world and that's how I got the two above pictures. It can do value iteration and policy iteration. Though I do need to give some self-defined hyperparameters. I set the reward is 100 when the agent get to the goal state, while the penalty is -1 if the state is not the goal state. In Movement.java, I set the move probabilities. Basically, if the agent want to move to north, then it will have 80% chance to end with north, while it also has 0.2/3 chance end with south, west, or east. Please notice that if the next move is actually a wall state, then the agent will stay in the original place. In the GridWorldLaunch.java, I use '0' stands for paths and '1' stands for walls.

3.2 analysis.py

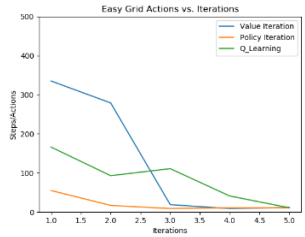
After the Burlap library, we need to make some analysis regarding the MDP results. So what we should analyze? That's come up with the second problem of this assignment. First we want to explore how many iterations it takes to converge, then we need to compare the speed of two types of iteration and explain the reason that causes the difference. Then I will try my best to come up more things to discuss.

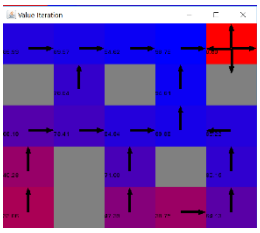
Now we should clear what we need to put in this python file. I gave four different iterations: 5, 100, 250, 1000. And by comparing the results based on these four iterations, we will figure out approximately how many iterations it will take to converge. There are the three kinds of data: value iterations, policy iterations, and q learning.

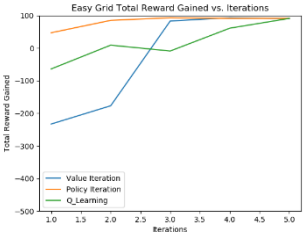
4. Easy Grid World

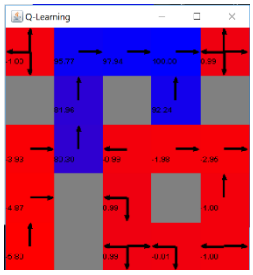
4.1 iteration = 5

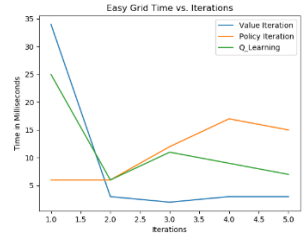
Running this code would be the faster one in this project. Now Let's see all six pictures.

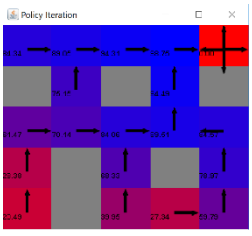












Now, how we supposed to read these six graphs? Let's focus on the right column first. The color changes from deep blue to hot red, which stands for most possible to move and least possible to move. This is make sense because if we check the first graph in the right column, the agent will stay in the top right spot because it cannot get better reward than staying there.

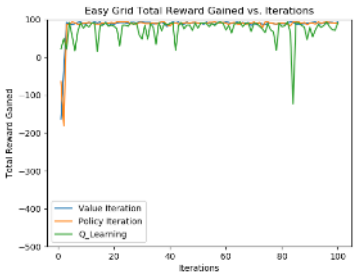
By observation of the first and second left column graphs, all three lines converge in iteration = 5. It at least tells us the iteration at least should be greater or equal to 5, and the third graph which is about time, showing us the time get decrement at 5 iterations than 4 iterations. In total, value iteration has the fastest convergence and Q-learning has the slowest convergence.

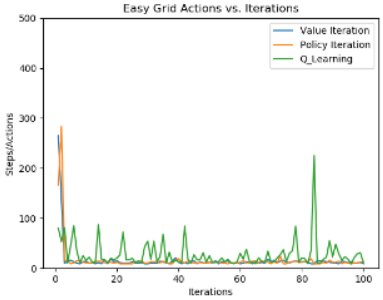
First we need to recall the formula: The way we built the q table is by

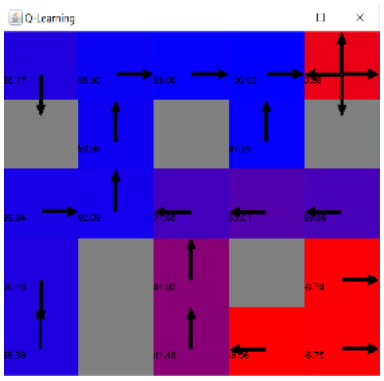
$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$, and for value iteration: $V(s) = \max Q(s,a)$. So the value iteration always picks the max value from q table. In my grid world, since we only give the goal state a very

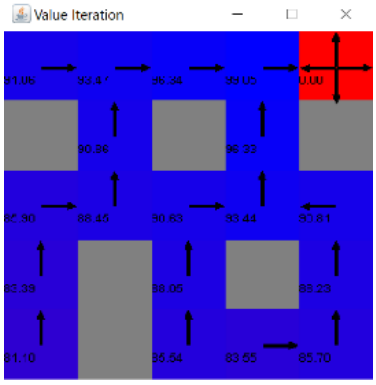
big reward, it will be most efficient if keep select the biggest value. Also we need to consider that this is a very simple grid world. It means the agent has very small chance end with the dead-end. Or I could say, the value iteration is more like greedy search while you don't have to apply any heuristic function and still be able to reach the best solution.

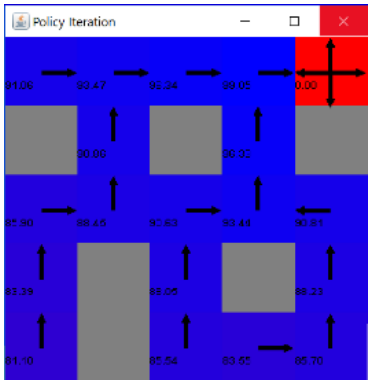
4.2 Iteration = 100











After we've done the five iterations work, let's see the three plot graphs. The turning point or we might call the changing point is nearly located in iteration around 5. After iteration = 5, the time consuming goes up very quickly, the reward stops getting better, and the action starts showing some kind "overfitting" behavior. For this easy grid world, as we can see, we could even find the best path by looking the whole grids. In this way, the agent must have found the best path but we still make it do the iteration again and again. By this, the iteration will be meaningless and that's why it shows no improvement in each plots.

Basically, q learning has the "over fitting" behavior when we iteration 100. But why this happens? In value iteration, the agent is learning the cost and reward when given a state, and it take a consider of the probability, while q learning is more like exploring by itself. The agent will receive whatever reward or penalty after it makes a move. Since there is no given information, the q learning agent will do as much exploring as it can,

while given large enough iterations. Now you might get understand why I am using the word “over fitting”. So it just gets too much learning. Well, but I have to confirm, this is not the same thing as the real over fitting.

All rewards eventually converge to nearly 100. This is making sense because after so many iterations in such an extremely easy grid world, all three should have a goo view of the grid world.

But in this case, which one has the best performance? In general, value iteration still has better performance since it takes shorter time than the policy iteration and has the same performance with policy iteration, showing by the other two plots. If we recall the analysis for iteration is 5, we will realized that value iteration also doing well in there. Hence, we need to revise the idea of value iteration. Basically, it is the algorithm that learning by rewards. The idea is that if we know the path, then we can make the move decision very quick. But we only know which move will bring the biggest reward. In this case we will choose that move. That’s the whole story of value iteration. In this grid world, we give the goal state a very big reward so that after one or two iterations, the move direction will very clear enough. And also, if I set the reward of the goal state is 1000 or 10000, then I believe that in such a easy grid word, the value iteration might only need two times of iterations to find the best move **because the target is too obvious**. Under the same environment, the policy iteration has very similar idea but it chooses to updating actions while makes it no advantages regarding to value iteration, but also its choice won’t hurt it too much.

4.3 we will stop here

As planned, we should keep going and try iteration 250 an 1000. However, since we’ve already seen the result in iteration 100, and understand the convergence will happen around iteration 5, we should stop here and turn to the hard grid world.

5. Hard Grid World

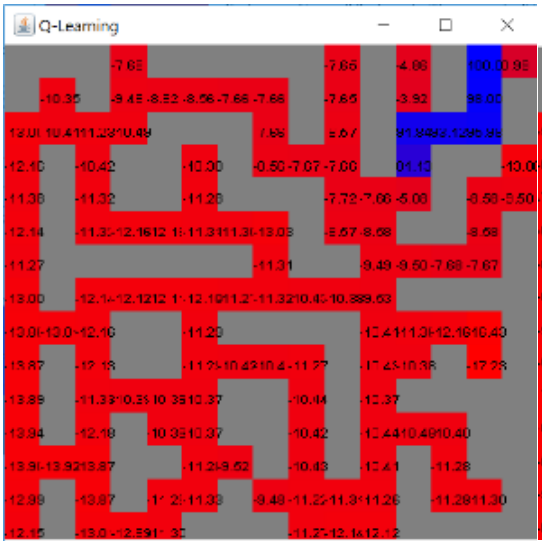
5.1 iteration = 5

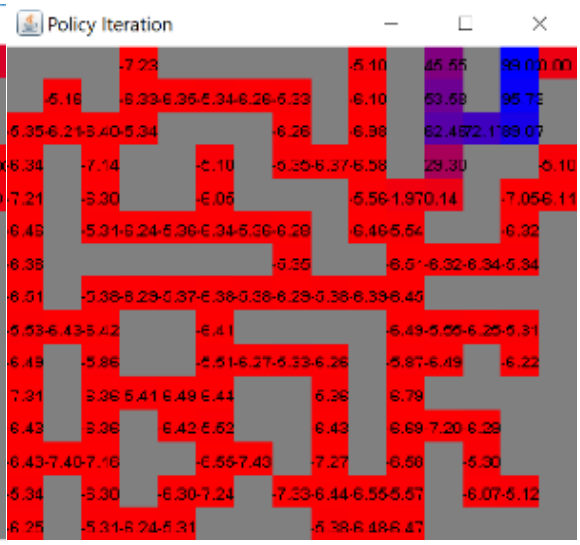
This time, I run with iteration = 5. The reason I am doing it is not because I want to get any valid result. It’s more about I want to show that iteration-5, which is a fine iteration for the easy grid world, will be completely useless when we are dealing with the hard grid problem.

We can treat the blue as highly movable or close to the goal state, and the red will be in the opposite side. In the hard grid world, I make it to be extremely not straight forward and I’ve already made sure the agent

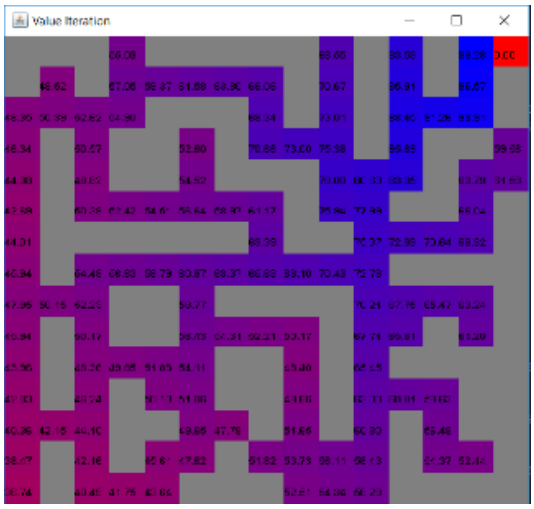
will definitely be trapped if it doesn't make enough iterations. This will clearly tells us that the size of problems are highly correlated with the iteration times.

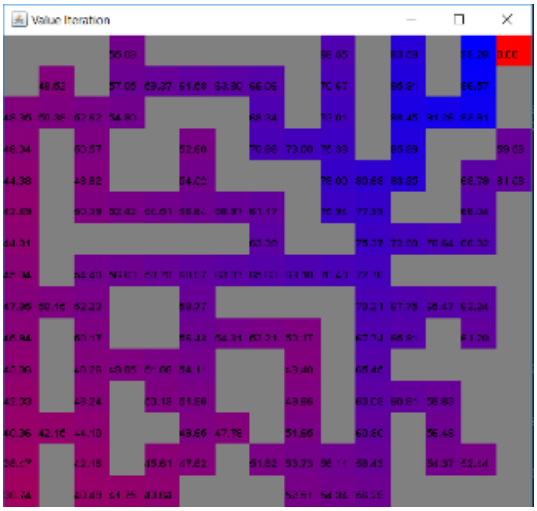
Furthermore, value iterations and policy iterations will have difference performance in the hard grid world, I predicted. For efficiency, we will skip iteration 100 and just start with iteration 250. Let's see what will happen.

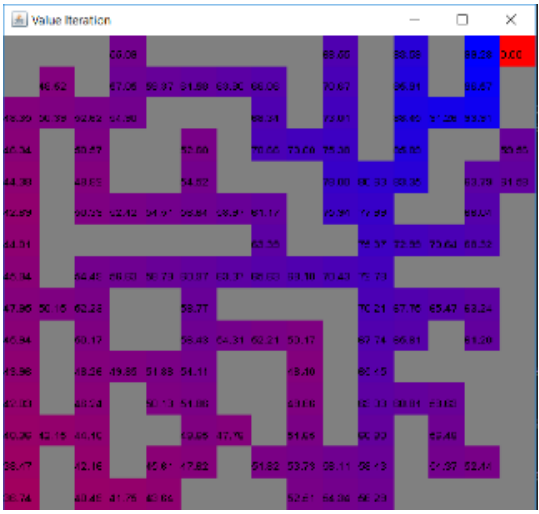




5.2 iteration = 250

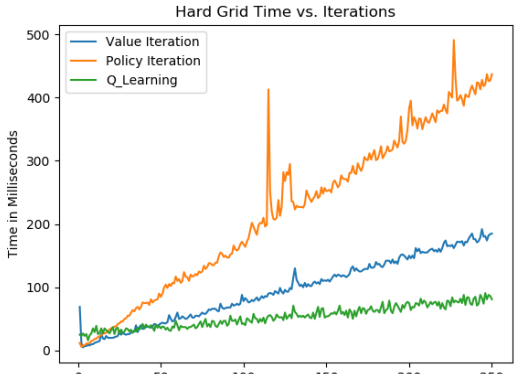


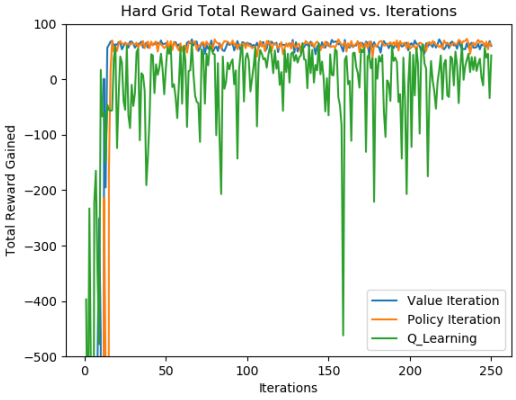


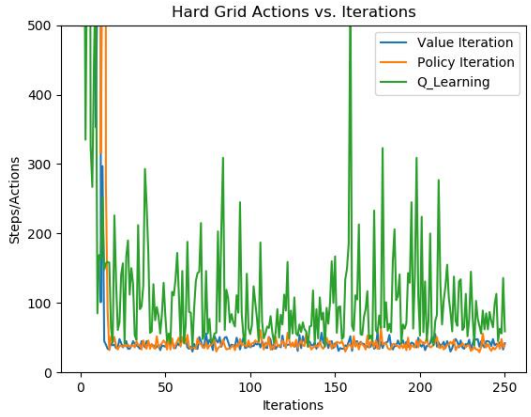


In general, the grid world map looks better than these map under 5 iterations. Around the upper and right region, we observe a large distribution of blue. This means that the agent will get more confidence once it reaches this area. And meanwhile, the red area only exists in goal state. Now let's explain more about the

red and blue. When the grid shows red, it just means the agent cannot receive better reward than stay in its current position. That's just why the goal state is always red. But the thing is we don't want the red shows in other non-goal state. If there exist red grids in non-goal state, it only tells us that the agent dones't learn enough so that it doesn't know where to go in the red states. But for now, the learning result is pretty good because the agent at least has an idea regard to the next move. Now let's see the plots graphs.







In the Time vs. Iterations plots graph, we saw that policy iteration takes the largest time and q-learning takes shortest time. Since the policy iteration is trying to find the policy directly, which means it always has a heavier computational burden. But if we use some representative values to find the policy, it will be faster than we just work on updating policy directly.

The reward goes to convergence at around 30 or 40, shown as the Reward Gained vs. Iterations plots. But in the same plots graph, the Q-learning never get convergence. Regardless the time consuming, value iteration and policy iteration have the similar performance. For such a big grid world, value iteration couldn't do better than the policy iteration because we only give the goal state a reward. The value iteration should have a better performance if the reward is generally increasing until the goal state. For example, if the grid world is like that the agent will get 5, 10, 20, 50, 100 while it gets closer and closer to the goal state. In our grid world, if the grid gets this much big, than it just makes the value iteration do the same thing as the policy iteration because there is no **representative** value for value iteration algorithm to learn. And even if we give more penalty for leaving away from the goal state would also improve the value iteration algorithm performance

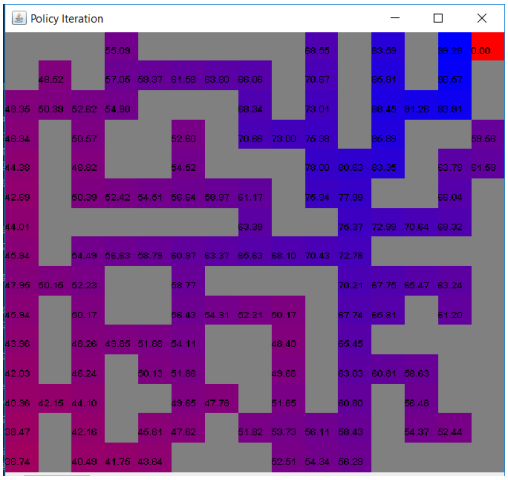
But as we can see, the variance for value iteration and policy iteration is very small, compare with the Q-learning. To explain this, we still need to back to the grid world. I design the grid world to make it less likely to trap the agent forever. If I make some trap and the trap will make sure the agent hardly gets out once the agent explores into the trap, then there will be a very high variance.

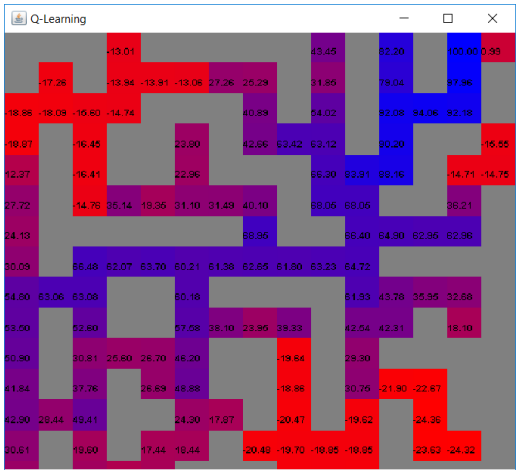
However, for Q-learning, it has the fastest computational speed, but the result is very bad. The reward, each time of q-learning iteration, never get to converge. I could provide one reason for that. Knowing that Q-learning is learn the policy based on the past sequence. More general, it tried to improve the policy based on what I got in last time. However, this is only work when the past action does influence the future situation. Well, I must say, not in all grid world that the past actions will not influence the future situation. But first, my grid world is pretty straight forward without much tricky traps. And also, if you take a look of the grid world blue and red pictures, you will find that the colors are changing very gently. It means that no matter which state the agent chooses, as long as the agent is trying to move to the upper right, it will also reach the goal state.

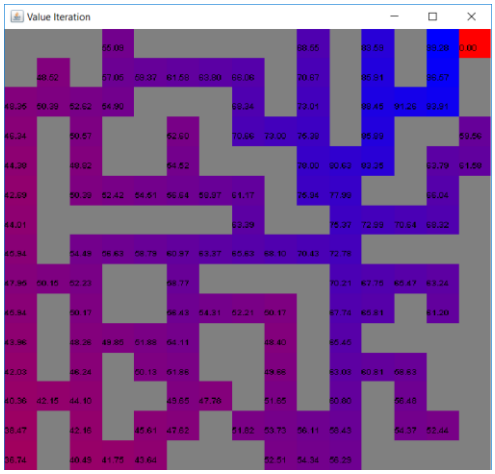
More precisely, think of us in a maze. If we have a compass and we've been told that we will get the goal as long as we keep north, then it really doesn't matter if we go east then go north or we go west then go north. What really makes it matters only if we go east and go north but this choice will eventually lead us to be trapped so that we have to back to the start and try west then north again.

Since the convergence happens around iteration 30 or 40, it might be not quite different when we try the iteration 1000 because there won't be any improvement.

5.3 Iteration = 1000

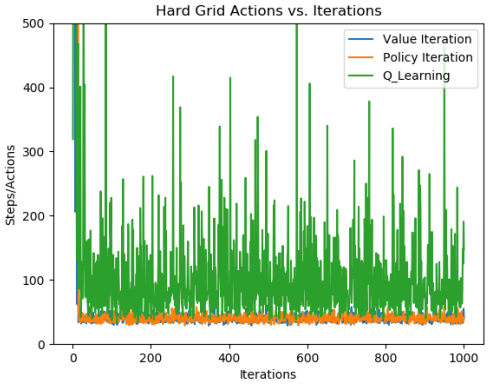


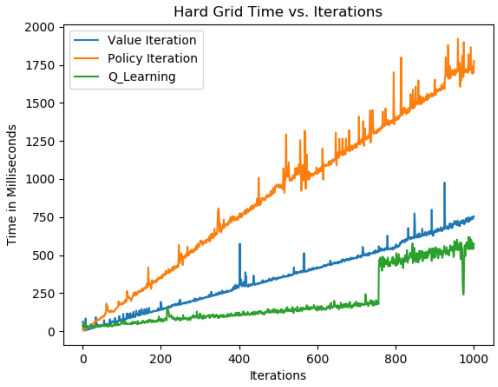


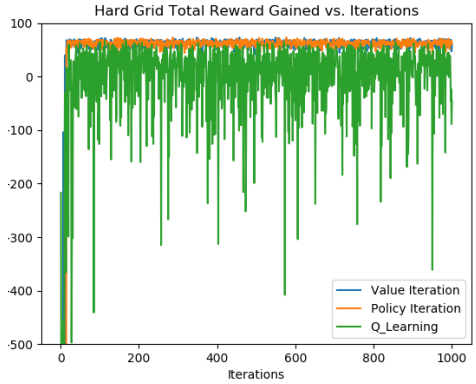


For now, let's focus on the q-learning graph. This time we have large enough iterations and there are some red appearance in the q-learning grid world. If you try to go along with the red, you will actually end with the dead-end. One thing that makes me astonishing is I don't even realize I make some dead-end trap when I made this grid. Let's focus on the q-learning grid graph. If we go along with the most blue state, then surprisingly, we are on the best path to the goal state. So I get the reason that q-learning has such large variance is just because it doesn't get enough iterations. In this way, I m hoping to see the convergence of q-learning in the plots graphs.

Here are plots:







Since we've already analyzed time issues in the last part, and the time complexity is same here. The reward of q-learning still doesn't converge but it generally provides a better.

6. Q-Learning

This section I want to talk about the behavior of Q-Learning in Hard Grid World, why Q-Learning doesn't converge but still return a good policy. First, let's understand why we need convergence. In machine learning, we train a algorithm on a dataset and we want this algorithm generates a steady model based on the dataset so that we will have more confidence when we input a new sample and get a result. Basically, this is the whole idea of why we are looking for the convergence. From this perspective, we could start with explaining why value iteration and policy iteration have better convergence. These two algorithms believe that they have knowledge of the grid world. When they are directing agent to explore, they might make wrong directions because what they know about the grid is wrong. That's why the beginning reward is very low. However, with the increasing of iterations, these two algorithms will improve they knowledge of the grid and generate better paths. In this way, the convergence will occur once the algorithms improve their

knowledge to the highest level. Differently, Q-Learning doesn't try to generate a well-trained model, unlike value iterations and policy iterations. Q-Learning is always focusing on the state that the agent exists, and what are its next options. But how to make a fair estimation? Q-Learning will use Q-table. Each time it receives a reward, it will update the q-table. So the q-table will help to build a fine policy in this grid world. Q-Learning would explore the whole grid world and keep updating the q-table. It will not based on the model so that this would also give it more chance to get better knowing of the grid world.