**Assignment 4**

**Markov Decision Processes**

**Introduction**

In this assignment, we explore Markov Decision Processes (MDPs). We will use two versions of grid world problem, one easy (has a small number of states) and other hard (has a large number of states). We will use value-iteration as well as policy iteration algorithms to find the optimal policy. We will also use a learning algorithm, Q-learning to learn the policy. We will run various experiment to compare the algorithms on time taken to converge, and do they converge to the same solution, etc.

**Markov Decision Process and Reinforcement learning**

MDPs has the following five elements:

* States: Set of states. In the grid world problem, each of the grids will represent the states.
* Actions: Set of possible actions for the given state of the agent in the environment. Different sets of action might be possible for different sets.
* State transition: The state to which our agent goes after taking an action from the given state
* Reward model: The reward our agent receives on going to a particular state.
* Discount factor: The present value of a future reward.

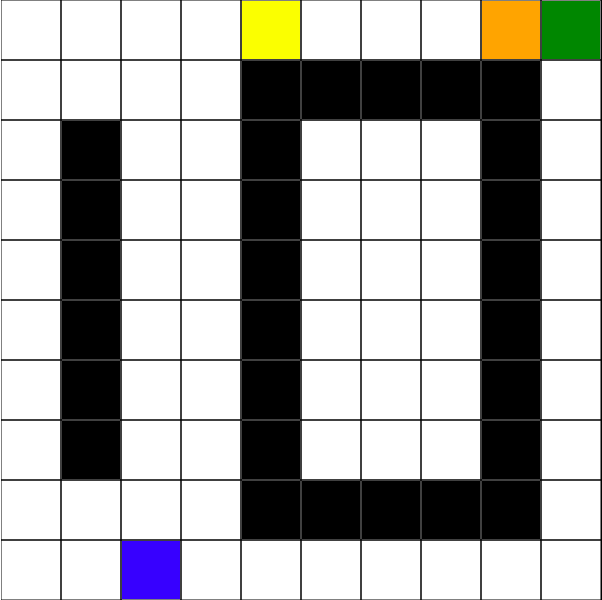
Let’s first look at the reinforcement learning problem. We have an agent which interacts with the environment (for example: a robot trying to walk in real world). The agent takes some action and corresponding to which it gets reward (i.e. robot has taken a good/bad step) and its current state changes to a new state. There might be only a set of actions available for the agent if it lies in a particular state (For ex: In the grid world problem, if the agent is at top left corner, the agent cannot move left or top. If the agent tries to take such an action it will remain in the same state.) The policy defines what action our agent should take if it lies in a particular state. Our agent tries to ‘learn’ a policy such that the total rewards it receives is maximized. Thus, we have to find such an optimal policy. We will use MDPs to model and solve the reinforcement learning problem.

**Two-problems**

We describe below the two problems we choose to show our analysis. We have choose such a problem which can modelled using MDPs. Each of the grid represents the state whereas we can move in 4 directions represents possible action. Thus, for any given state we have 4 or less possible actions. Moreover, a reward is associated depending on which grid our action take from the current state. Our agent takes the given action with probability 0.8 and can move with equal probability in other directions. This problem easily corresponds to real world situation, where we have to find path from one place to other place and the grid corresponds to rewards such as if there is a river or hill. This path might be discouraged. Hence, we could model real world problem using grid world problem.

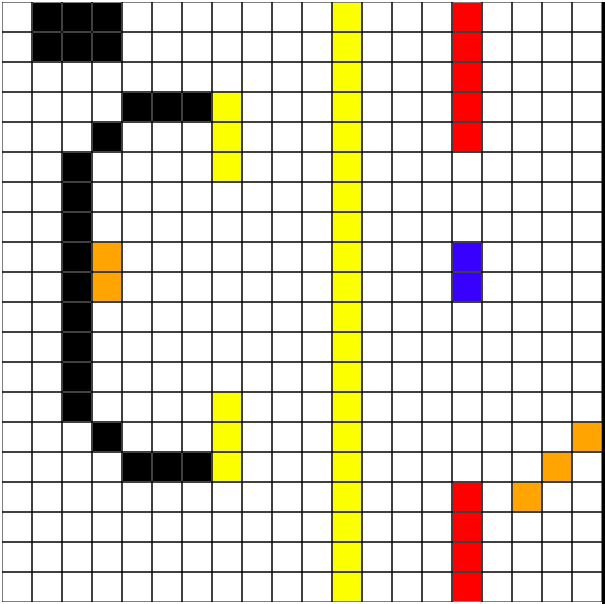
**Grid world (Easy)**

In the easy version, our grid world has a size of 10x10. Thus, we have 100 grids/states. Individual reward of each grid is colour coded. Green grid has reward of 5, blue has 3, white has 0, yellow has -1 and orange has -3. Black grids are brick/wall. We cannot cross them. (Ignore black labelled as 1 below)

The starting point is bottom left and the terminal state (goal) is top right. We have to find a policy to reach the terminal state from the starting point.

**Grid world (Hard)**

In the hard version of grid world problem, we choose a grid of size 20x20. Thus, we have 400 states/grids. Individual reward of each grid is colour coded. Green grid has reward of 5, blue has 3, white has 0, yellow has -1 and orange has -3. Black grids are brick/wall. We cannot cross them. (Ignore black labelled as 1 above). Again, the starting point is bottom left and the terminal state (goal) is top right. We have to find a policy to reach the terminal state from the starting point.

**Implementation**

We have used burlap library to implement the algorithms in python using jython. We implement the policy-iteration, value-iteration and Q-learning to solve the two MDP problems.

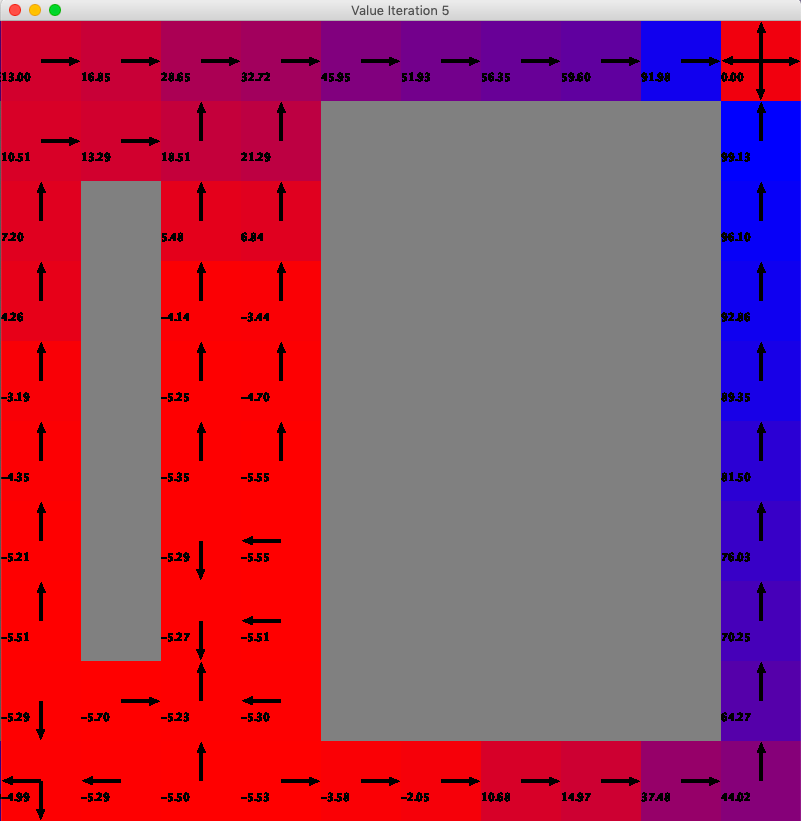
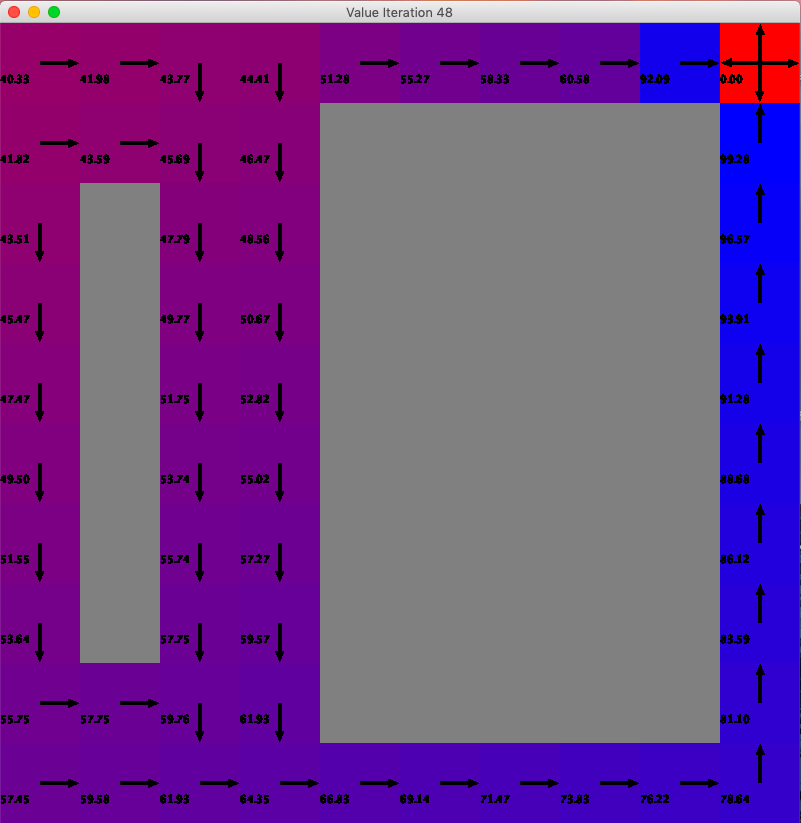
* Policy iteration: In policy iteration, we re-define the policy at every step. We repeat this process until the policy converges. It is guaranteed that the policy iteration will converge to the optimal policy.
* Value iteration: We iteratively computes the estimate of the value function V(s). The initial value of V(s) are set randomly. The value function quantifies the goodness of the state. We repeat this process until V(s) converges. The algorithm is guaranteed to converge.

Although both the above algorithms are guaranteed to converge, often policy iteration converges faster. This can be seen in our analysis later.

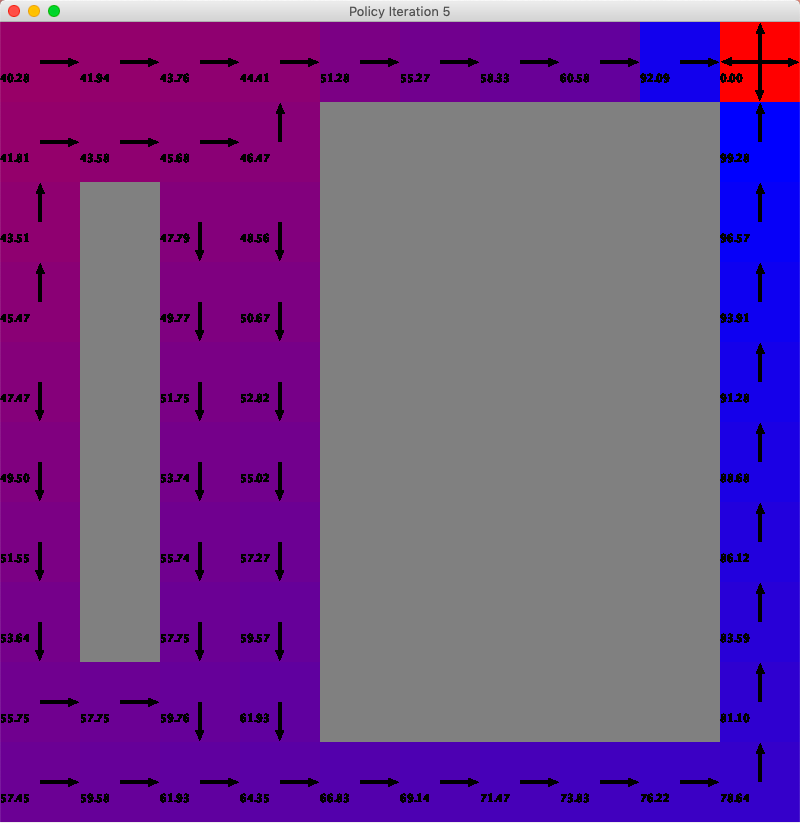
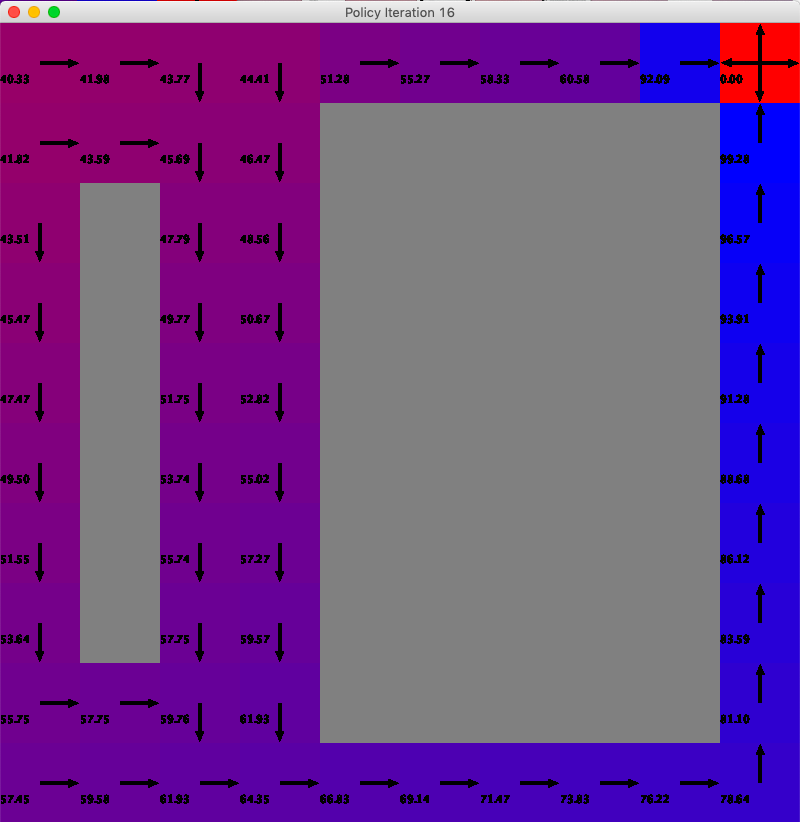
* Q-learning: It is model-free learning algorithm. The agent tries to maximize the sum of total reward. The future rewards are taken into consideration discounted by a factor. There is also a dilemma of exploration vs exploitation i.e. should we explore new states more or keep on improving already explored states. Thus, the given action is taken with a probability ‘epsilon’. The agent takes the given action with probability epsilon but it can take other possible action with probability 1-epsilon. Thus, we have three hyperparameters which include the learning rate, discount factor and epsilon.

**Easy Grid World**

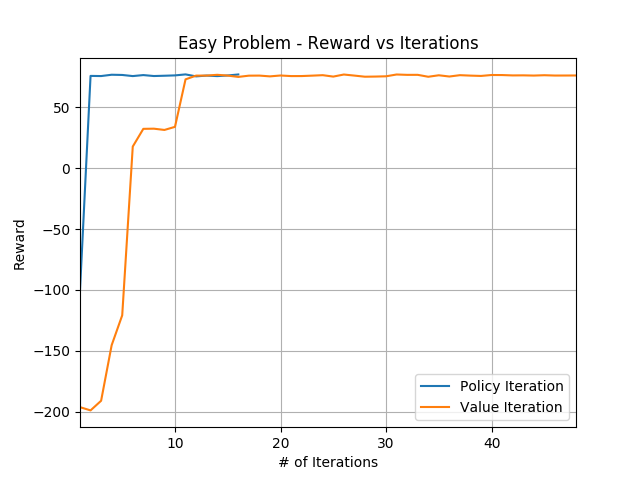
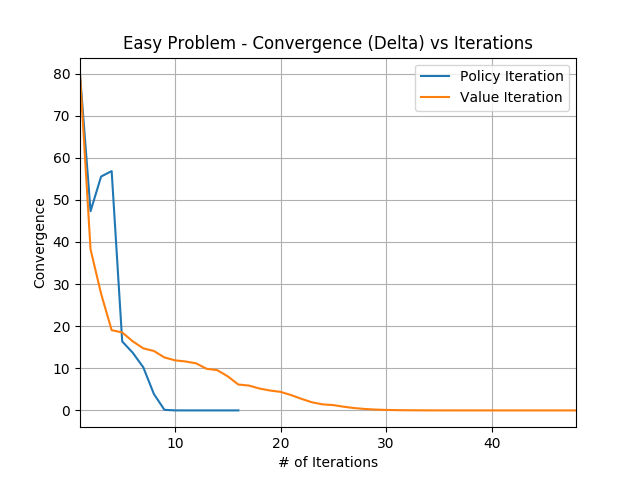
We show the policy found by the value iteration as well policy iteration. The first two graphs show the policy found using value iteration after 5th iteration and 48th iteration. The algorithm converges on 48th iteration. Blue colour shows how often the cells are explored. By the end of 48th iteration, value iteration is able to find the optimal policy. We can observe that most of the cells have become bluish in contrast with the reddish colour when the algorithm completed only 5 iteration. In the optimal policy, we can see that path from left to right to up is preferred and has better rewards in the corresponding paths. This can be attributed to the presence of negative reward grid in the top-left of terminal state. The policy discourages to take that route. The arrows below shows the optimal action for every state.

** **

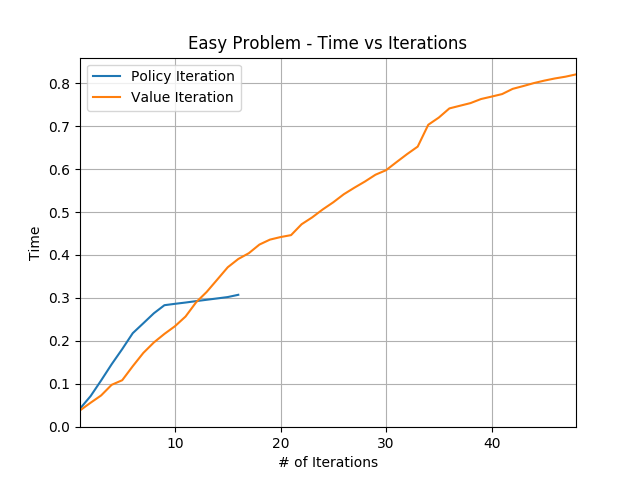
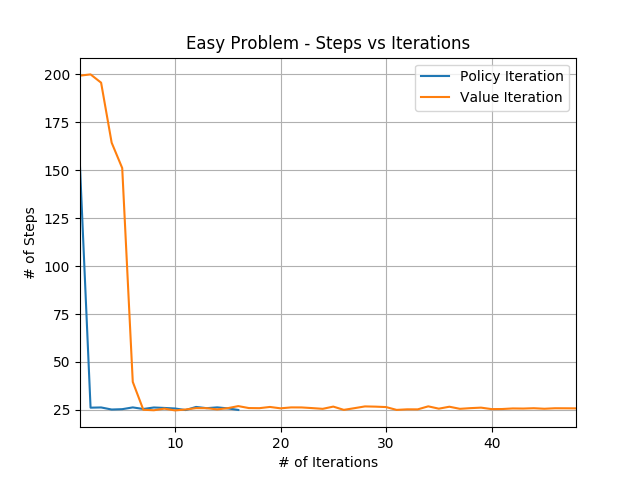
The below two graph corresponds to policy iteration algorithm for 5th and 16th iteration respectively. In the optimal policy, we can see that path from left to right to up is preferred and has better rewards in the corresponding paths. This can be attributed to the presence of negative reward grid in the top-left of terminal state. The policy discourages to take that route. The arrows below show the optimal action for every state. Policy iteration has almost figured out the solution by the end of 5th iteration and has explored most of the grids sufficiently. This is in contrast with the value iteration algorithm which takes much more iteration to converge, but they converge to same solution.

Next, we will compare both these algorithms w.r.t iteration, reward, steps required, time for convergence as a function of iteration.

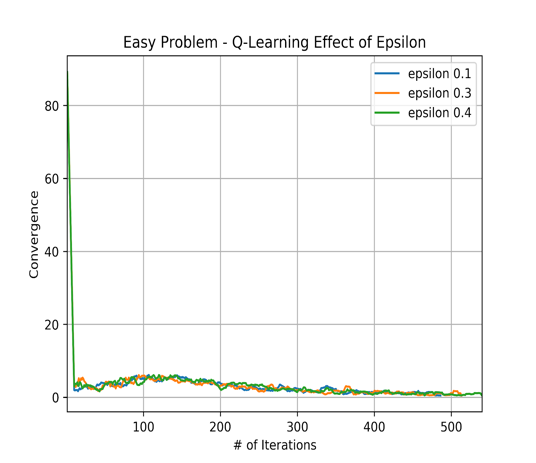
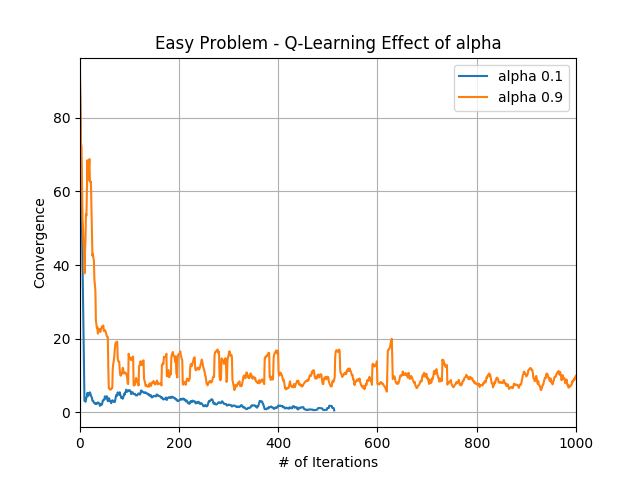


The first graph above show convergence w.r.t iteration for both the algorithms. We can observe that how quickly policy iteration converges. We have choose the convergence criterion as when the change between consecutive values is less than 10^-6. Similar trend can be seen in the reward vs iteration graph. We get a reward of ~80. For the policy iteration it only takes few iterations compared to value iteration.

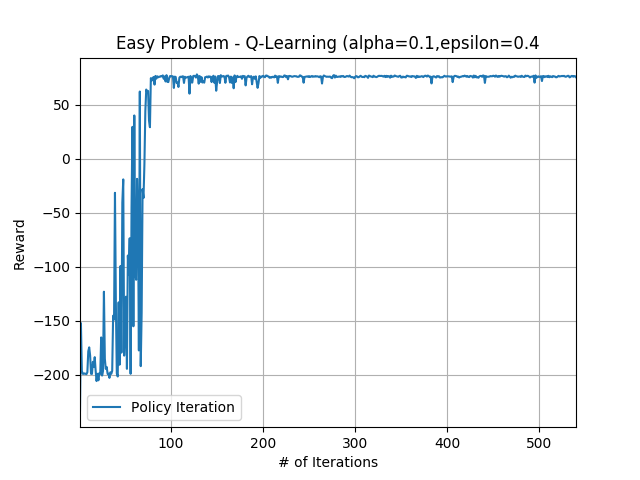


The third graph is the number of steps taken from starting state to terminal state. We have total 100 grids. We can see initially it took about 150 to 200 steps to reach the terminal state. But soon after the optimal policy is found it takes only 25 steps to reach the terminal state. Again policy iteration archives this in few iterations only. The fourth graph compares the time of both the algorithm. Policy iteration is computationally expensive and takes larger time to converge. This is beside the fact policy iteration runs for less number of iteration. Thus, the time comparison depends too much on the type of environment we are interacting/using.

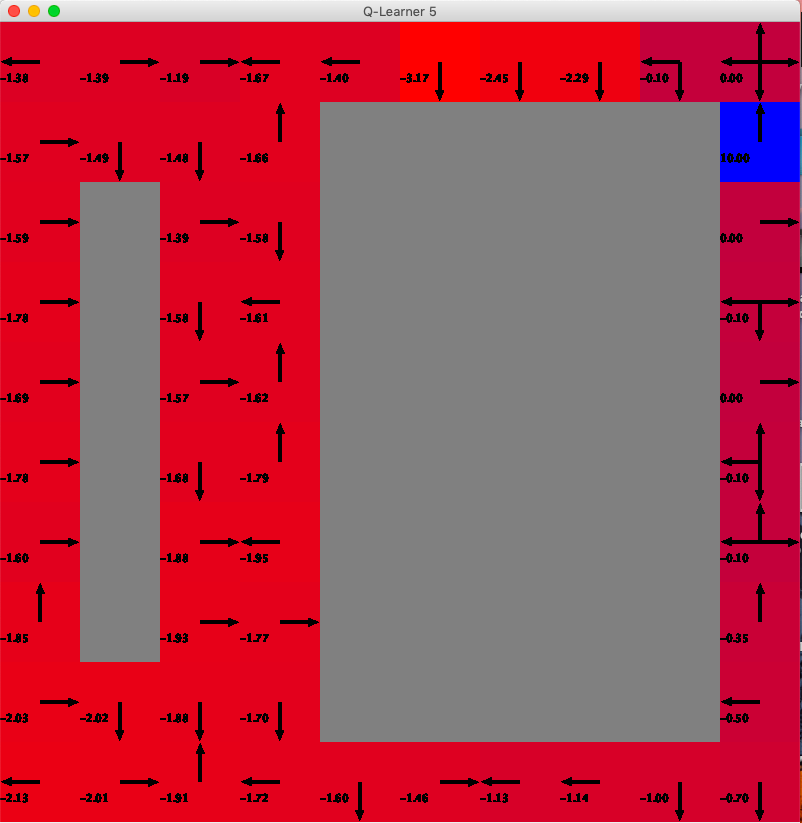
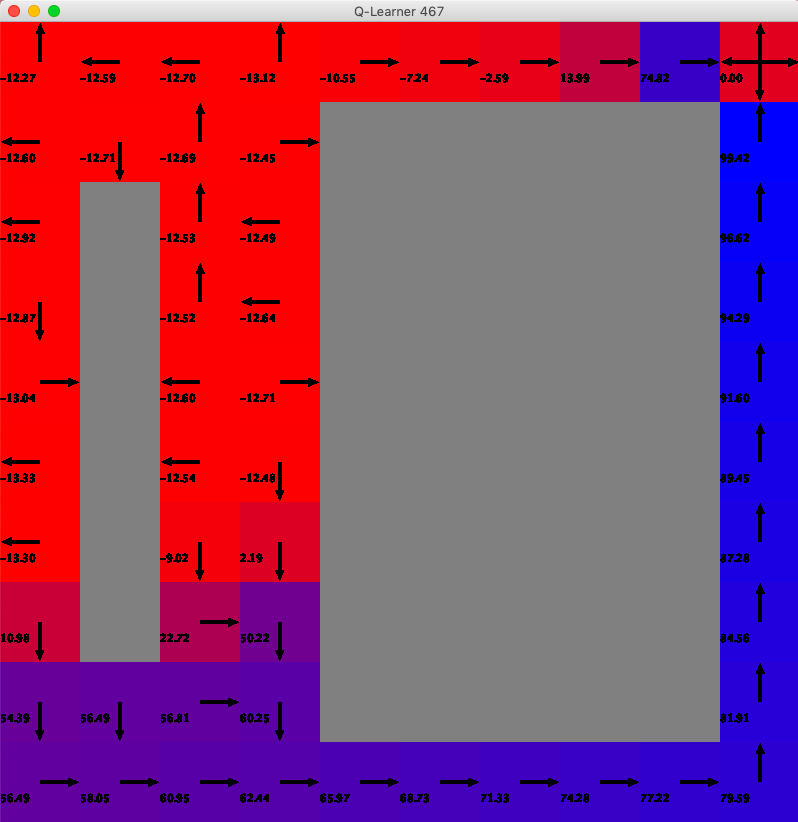
Now, we will describe the results of Q-learning for the easy problem.



A close up of text on a white background

Description automatically generated

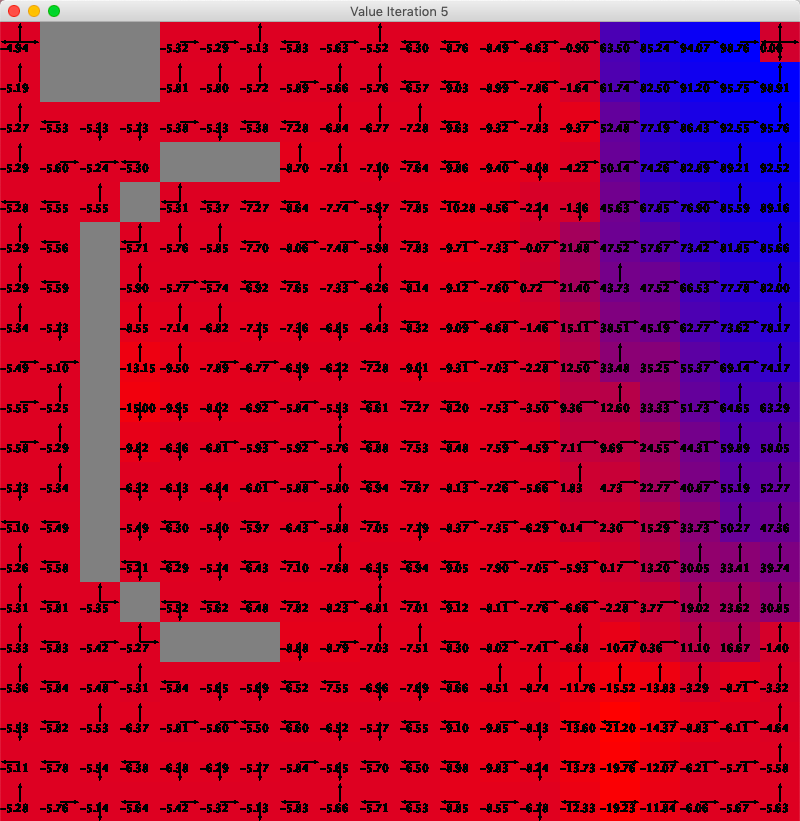
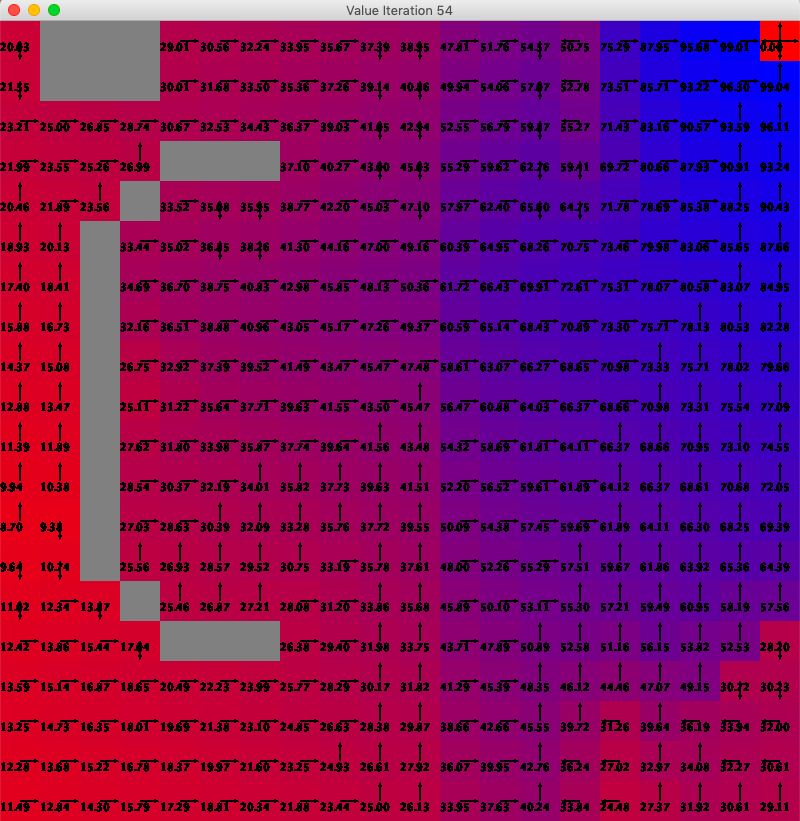
The first graph above shows the effect of alpha (learning rate). We plot the convergence w.r.t number of iterations for different alpha (or learning rate). We observe that a high alpha value results in destabilisation. A higher value of alpha relies too much on new exploration and discards the old values learnt. The next graph shows the effect of epsilon. There is almost negligible effect on changing the epsilon. The third graph shows the effect of discount factor. A **higher discount factor** gives here better convergence. The future rewards are fixed in nature. Hence, having a better estimate of future is better. Suppose we might have an environment which is dynamic in nature i.e. future rewards may change, in that case having a smaller discount factor would be beneficial. We have to choose the discount factor wisely depending upon the environment. Next graph show the reward w.r.t. number of iteration when alpha is set to 0.1, epsilon is set to 0.4 and discount factor set to 0.99. We can see that the reward converges to similar value as in policy iteration and value iteration.

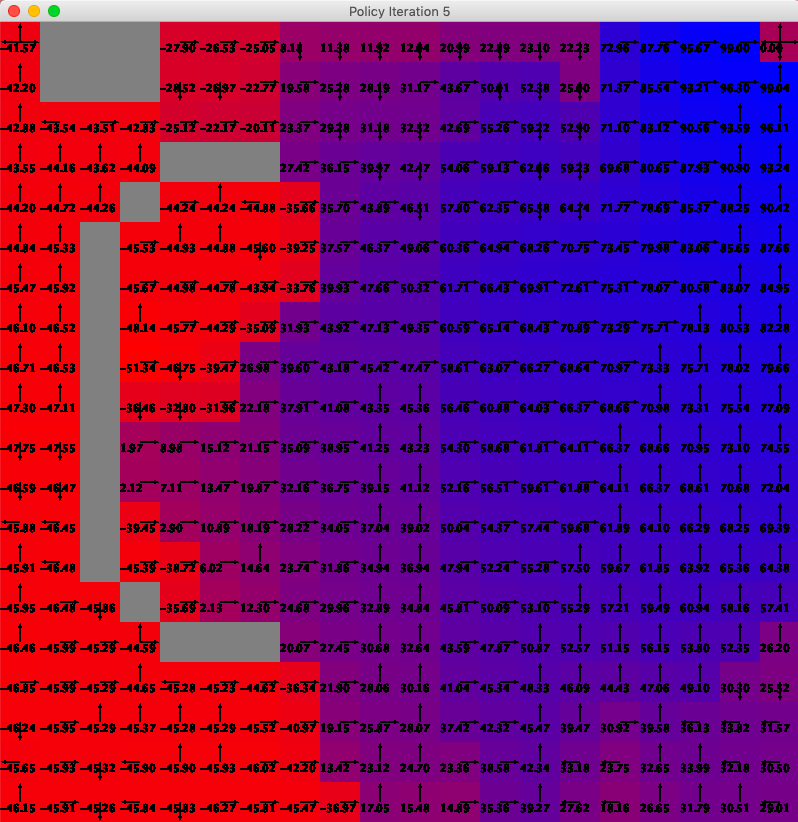
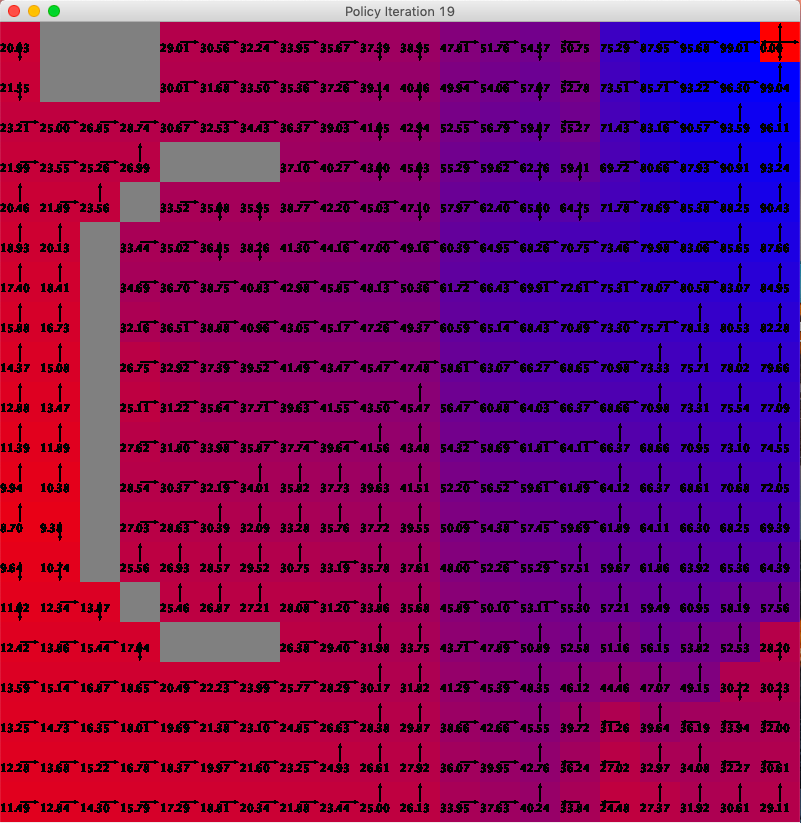
The above graph shows the policy after the 5th iteration and 467th iteration for the Q-learning algorithm. We can see that initially most of the grids are unexplored (left figure) and as the algorithm converges most of the grids are explored (right). The optimal policy we got from the Q-learning is different from the policy iteration and value iteration algorithm.

**Hard Grid World**

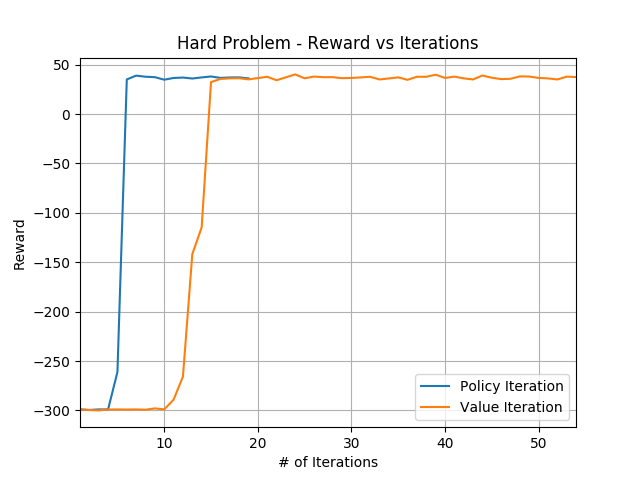
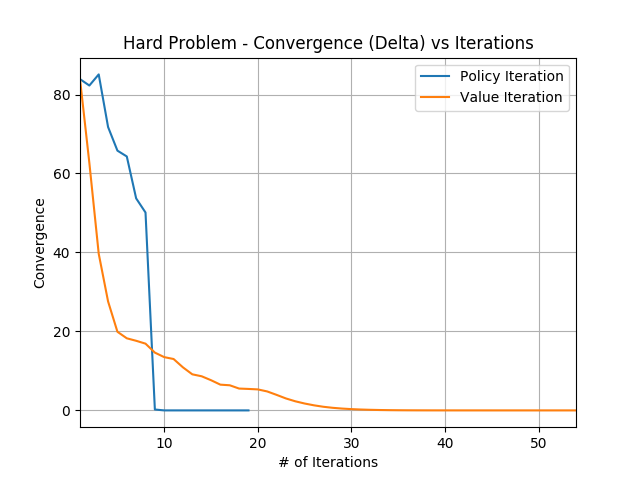
We show the policy found by the value iteration as well policy iteration. The first two graphs show the policy found using value iteration after 5th iteration and 54th iteration. The algorithm converges on 58th iteration. Blue colour shows how often the cells are explored. By the end of 58th iteration, value iteration is able to find the optimal policy. We can observe that many of the cells have become bluish in colour. This is in contrast with the reddish colour when the algorithm has completed only 5 iteration. The arrows below show the optimal action for every state.

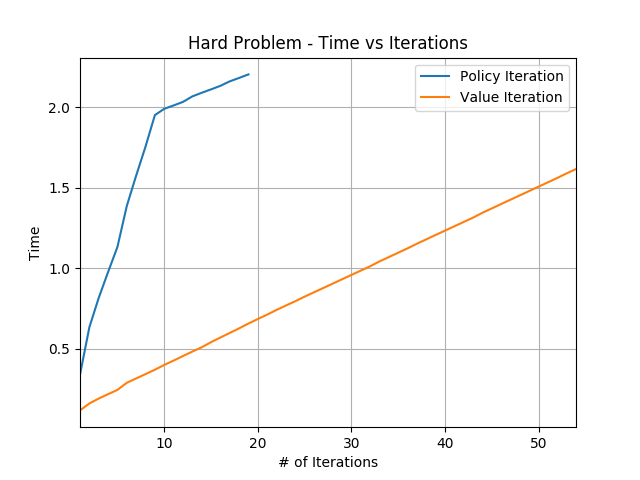
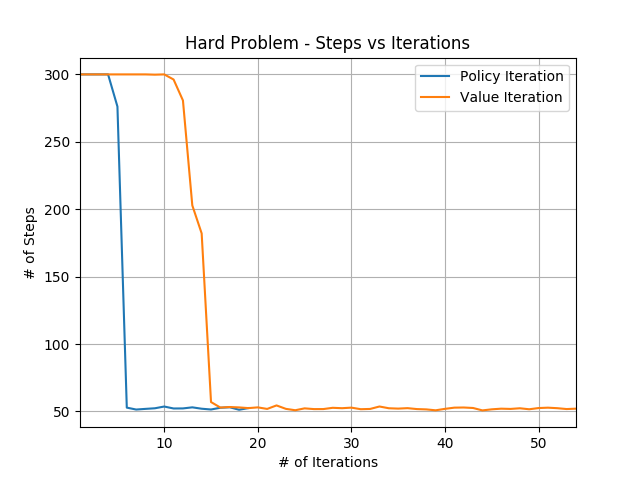
The below two graph corresponds to policy iteration algorithm for 5th and 19th iteration respectively. Policy iteration has almost figured out the solution by the end of 5th iteration and has explored most of the grids sufficiently. This is in contrast with the value iteration algorithm which takes much more iteration to converge, but they converge to same solution. Thus, policy iteration can give us optimal solution even for relatively larger problem in few iterations.

Next, we will compare both these algorithms w.r.t iteration, reward, steps required, time for convergence as a function of iteration.



The first graph above show convergence w.r.t iteration for both the algorithms. We can observe that how quickly policy iteration converges. We have choose the convergence criterion as when the change between consecutive values is less than 10^-6. Similar trend can be seen in the reward vs iteration graph. We get a reward of ~40 in both the algorithm but for the policy iteration it takes fewer iterations compared to value iteration.



The third graph is the number of steps taken from starting state to reach the terminal state. We have total 400 grids/states. We can see initially it took about 300 steps to reach the terminal state. But soon after the optimal policy is found it takes 50 steps to reach the terminal state. Again policy iteration archives this in few iterations only. The fourth graph compares the time of both the algorithm. Policy iteration is computationally expensive and takes larger time to converge. This is beside the fact that policy iteration runs for less number of iteration. Thus, the time comparison depends on the type of environment we are interacting/using. It is quite possible that policy iteration might takes lesser time than value iteration.

Now, we will describe the results of Q-learning for the hard problem.

A screenshot of a cell phone

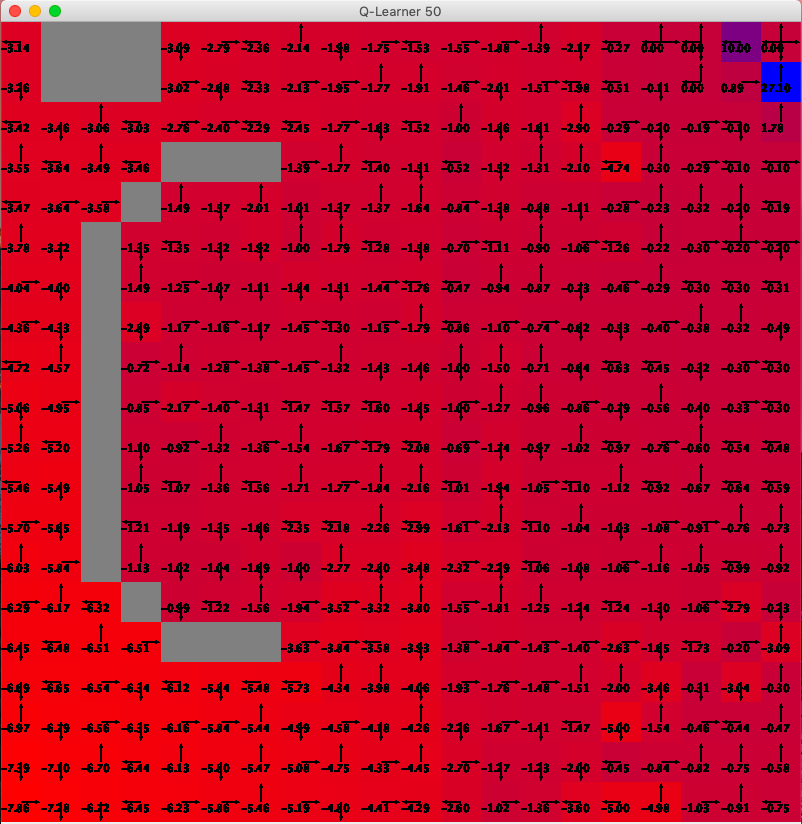
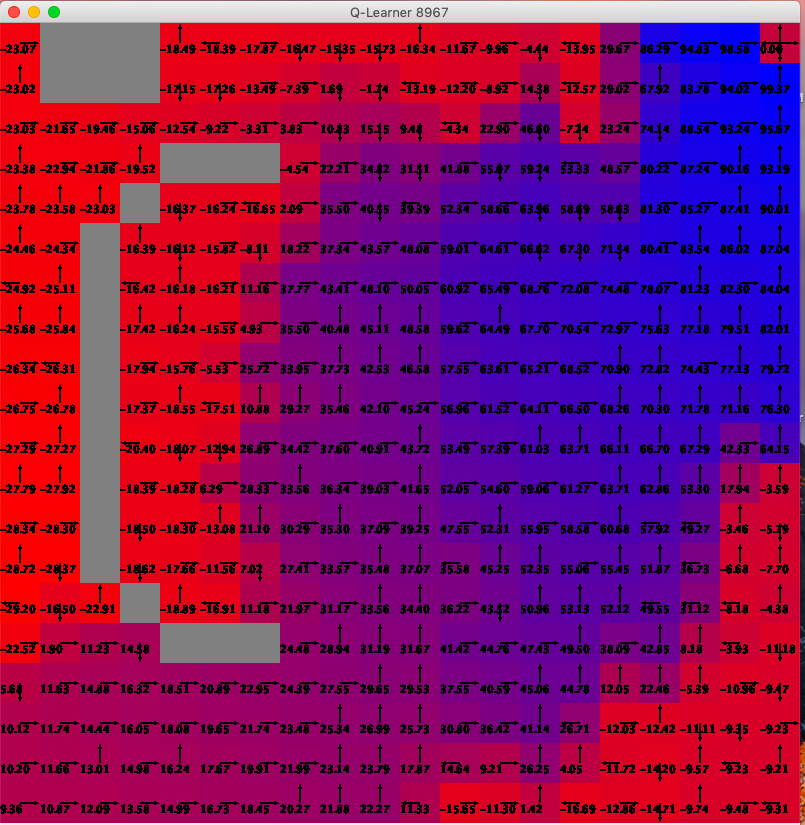
Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

The first graph above shows the effect of alpha. We plot the convergence w.r.t number of iterations for different alpha (or learning rate). Alpha is varied to 0.1 and 0.9. We observe that a high alpha value results in destabilisation. A higher value of alpha relies too much on new exploration and discards the old values learnt. This effect was less when the grid size was small. But as the grid size/number of states increased this destabilisation got amplified. The next graph shows the effect of epsilon. We vary epsilon over 3 values in [0.1, 0.3, 0.4]. There is almost negligible effect on changing the epsilon. Next graph show the reward w.r.t. number of iteration when alpha is set to 0.1 and epsilon is set to 0.4. We can see that the reward converges to similar value as in policy iteration and value iteration.

The above graph shows the policy after the 5th iteration and 8967th iteration for the Q-learning algorithm. We can see that initially most of the grids are unexplored (left figure) and as the algorithm converges most of the grids are explored (right). The optimal policy we got from the Q-learning is different from the policy iteration and value iteration algorithm.

The time mentioned in all the above graph is in minutes.

**Why Q-learning?**

We can see that Q-learning requires much more time to converge than the policy iteration and value iteration. But, we Q-learning gives us huge advantage that we don’t need a model. Often in real world scenarios we have limited knowledge about the world/environment. Thus, we cannot model the environment and hence value iteration or policy iteration might fail. Other versions of Q-learning such as DQN, DDQN can be easily used to learn about more complex environment. Next, the policy found from Q-learning might be different from policy-iteration or value iteration. This is because Q-learning sometimes converges in premature fashion (although it would be near optimal solution). We have to tune the alpha and epsilon w.r.t. number of iteration to get the optimal policy.

**Conclusion**

In this project, we studied two types of (Grid World) problems, one with small number of states (100) and other with large number of states (400). We ran a series of experiments using the three algorithms to understand each one of them . We saw that policy iteration converges faster but since it is computationally expensive it might take larger time to converge. Next, we saw the benefit of Q-learning. We saw that it took much longer time to converge but it is a model-free algorithm. We saw the effect of the three hyperparameters in the Q-learning viz. learning rate, epsilon and discount factor.