CS7641: Assignment4 pchordia6@gatech.edu

Sequential decision-making is the branch of algorithm for strategy selection where decision-making is impacted by the environment. In deterministic environments, the outcome can be determined based on specific state, where as in stochastic the outcome may not be always as expected. Most real world problems are stochastic in nature and Markov Decision Process provides a mathematical framework for finding the solution of such problems.

I will be using grid world Frozen lake problem from Open AI-gym to explore MDP's by using Policy iteration and Value iteration. Same problem I have used to explore the reinforcement-learning problem.

Why Frozen Lake is an interesting problem?

In frozen lake gird, S is the Starting point, G is the goal, F is point where Foot can be placed and H is the Hole. If agent hoes on Hole, it drops in frozen lake and game starts again. What makes problem interesting is that same problem can be used with different grid sizes. I have used 4*4 and 8*8 grid to analyze the environments. It is an apple–to-apple comparison to understand the agent's computational performance, reward, and behavior of different size grids.

Open-AI gym gives us the flexibility to keep the environment stochastic by using the slipperiness, which can make the agent to get unexpected result occasionally. Frozen lake is a simple problem, but having different number of states and sizes enable us to understand the behavior of different algorithm.

The target is to find the best possible route from start to goal in least number of steps by avoiding the obstacles (Hole) and considering the slipperiness of environment. It can assumed that smaller grid size can have better performance, but the stochasticity can lead to different results. It is possible that bigger grid size gives the agent more chances to learn and hence can lead to better or similar results as smaller. Smaller grid has less number of steps to take which can lead to better performance. All this kind of comparison will be good to observe leading us to understand the algorithm better.

Below are the 2 grids which are used for analysis. If observed on the 8*8 grid, obstacles increases near to the goal making it difficult for agent to reach to goal.

```
"SFFF",     "SFFFFFFF",
"FHFH",     "FFFFFFFF",
"FFFH",     "FFFHFFFF",
"HFFG       "FFFFFHFF",
            "FFFHFFFF",
            "FHHFFFHF",
            "FHFFHFHF",
            "FFFHFFFG"
```

By these two problems, I have tried to understand the effect of different number of states, gird size on agents performance on rewards, computational time and efficiency and ability to tackle new unknown environment.

Implementation and Methodology: I have used python for understanding and testing the value iteration, policy iteration and q-learning algorithm. The gym environment comes up with different levels of in build environment which can be used for understanding the algorithm. Initially we used frozen lake and taxi environment for testing. Taxi environment is not a stochastic environment so changed the environment to Taxi 8*8 Grid.

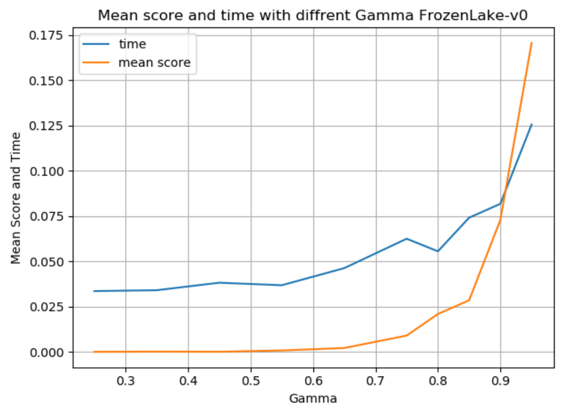
Markov Decision processes: In MDP, the agent interacts with environment by taking steps to reach the goal. At each of these steps, the state of agent changes and the reward function. The goal of agent is to find the optimal policy such that it maximizes the rewards. The rewards function in MDP can be different, it can be that at every step, a small negative reward is given to agent, which makes agent to reach the goal faster, where it gets maximum reward or give no reward at all until it reaches goal. It depends on the implementation. MDP has 5 elements, States, action, transition model, reward model and discount factor. The actions performed by agent is known as policy.

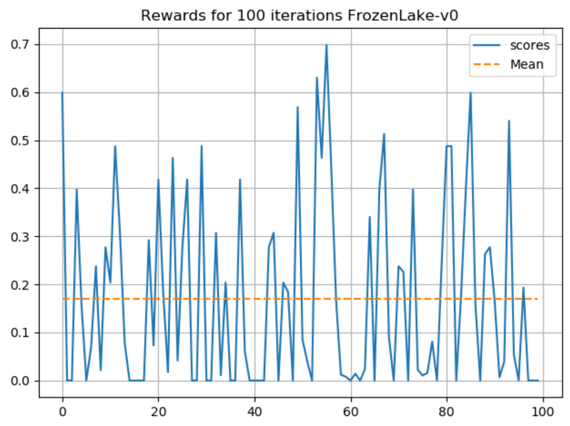
Value iteration: The Basic idea of Value iteration is to calculate the immediate reward in which agent is plus al the discounted rewards for future actions in which agent can be assuming agent will choose the optimal action. This directly establishes that there not only current but also all the future starts are also very important to maximize the reward in optimal steps.

Frozen lake 4*4 has 16 possible states, so there will be 16 possible Bellman equations for each state. We arbitrarily start with some state and then at each state we update the value, which is used in the next state. This is repeated until we reach equilibrium.

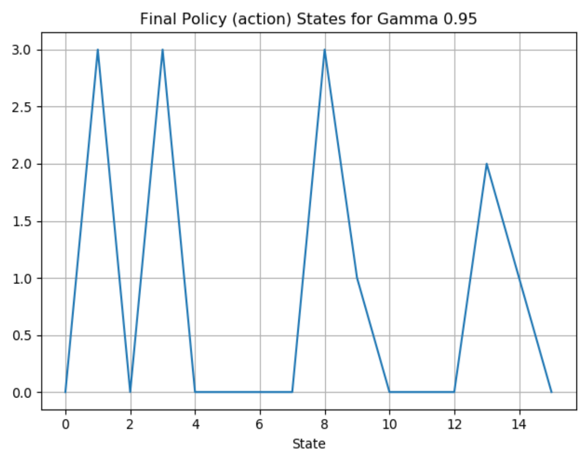
I tested the frozen lake 4*4 grid across different gamma values from 0.25 to 0.95 with difference of 0.10. This will help us understand impact of varying current and future rewards on the Bellman's equation. Higher value of gamma means more weightage to current rewards but also giving weightage to future rewards. Low value of gamma signifies low weightage to current rewards.

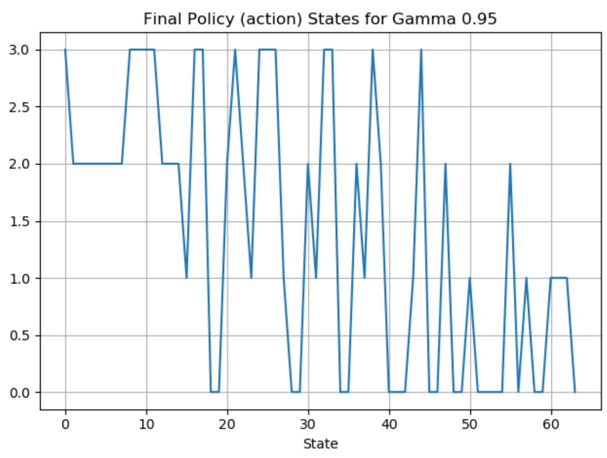
It can be clearly seen from the graph that as the gamma increases the reward increase. The highest score is ay the gamma 0.95. For this it can be clearly deuced that current reward plays a crucial role in convergence of algorithm, however as the gamma increases the time taken by algorithm also increases. Gamma =1 will give only importance to current reward and all the future rewards will be zero, which can result large or infinite time for agent to reach the goal. With gamma at 0.95, we ran the value iteration algorithm for 100 episodes and it can be clearly seen that maximum reward is received at around 55 iteration.





At 0.95 gamma and 55 iteration, the agent was able to reach the goal in minimum number of steps. The reward of zero indicated that agent got into the hole and started again. It can also be seen that as the gamma increases the count of convergence step also increase for finding the best policy.

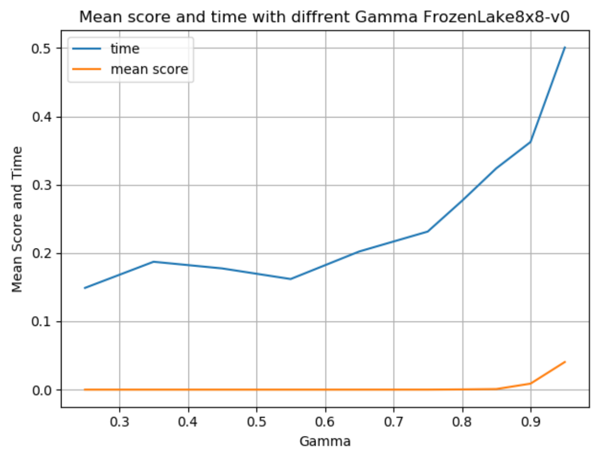


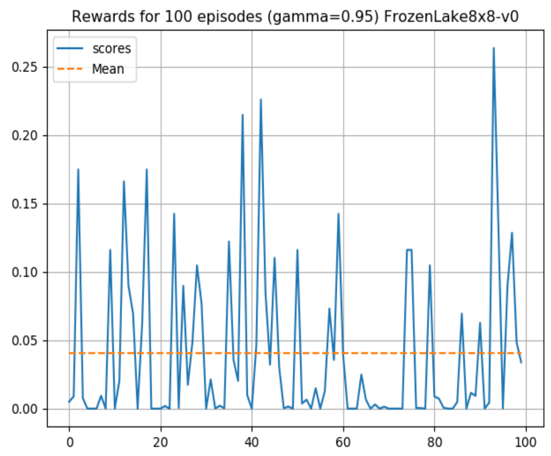


Above diagram shows the optimal policy with action values on y-axis. Left shows that the optimal policy for the Frozen late 4*4 grid and right shows the optimal policy for 64 states of 8*8 grid.

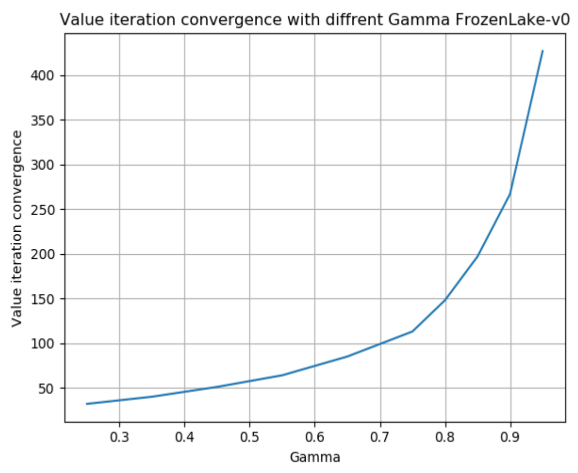
When we compare frozen lake 8*8 performance, the time required for reaching at goal at every gamma is much higher than frozen lake 4*4. This is because the number of states is higher than 4*4 grid. The rewards however is much lower than the 4*4 grid. The algorithm did not performed good when the gamma values are small and it can be clearly seen that after discount factor of 0.85 the reward function is noticeable. This is because as the number of states are more, the discount factor seems to tend nearly zero as it is a series.

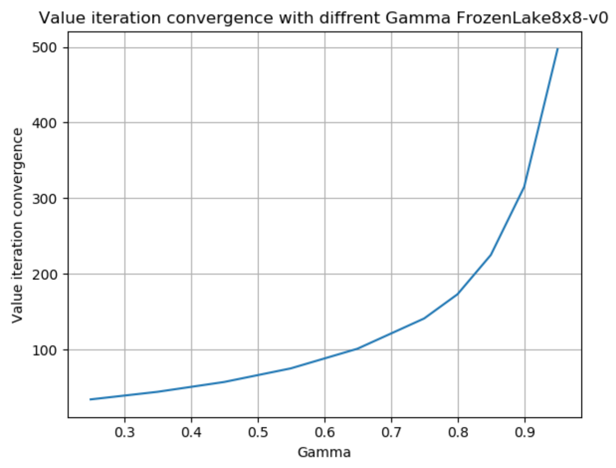
When the algorithm ran for 100 episodes at discount factor of 0.95 (because it gave the highest score), the highest reward seems to be achieved much later in the iterations. This means that the agent learned to reach the goal in later than 4*4 grid, because the number of states are more.





The graphs below shows the convergence of value iteration at different gamma for Frozen Lake 4*4 and Frozen Lake 8*8. It can be seen that 8*8 grid took more number of iterations to converge at every gamma value which can be because of number of states. X axis shows the gamma value and y axis shows the number of iterations for convergence.

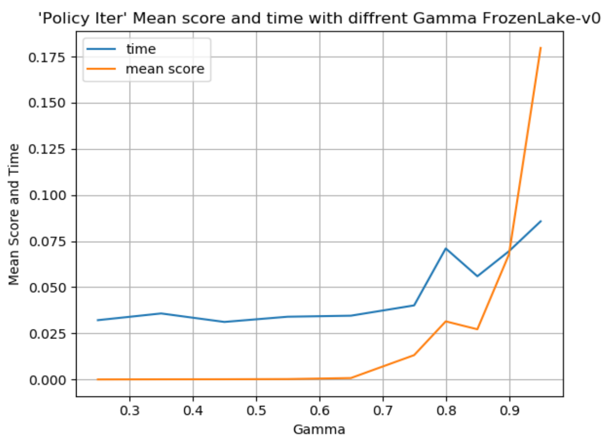


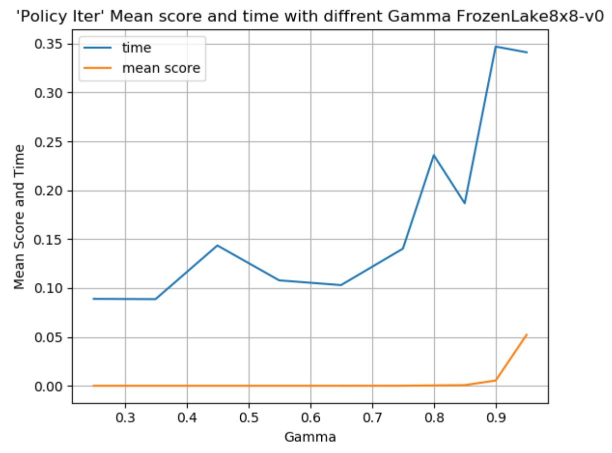


It can be said that value iteration depends on gamma < 1. If gamma = 1, the time for agent to reach goal can be infinite.

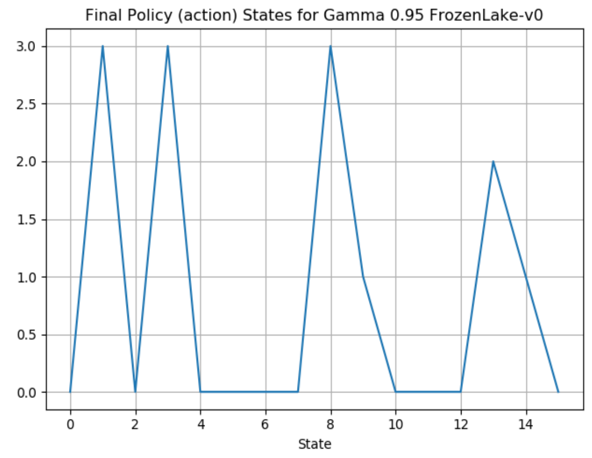
<u>Policy iteration</u>: Policy iteration algorithm calculates the utility of each for a given policy. Then it improves the policy based on one step look ahead basis. The algorithm keeps on following this policy evaluation and improvement until there are no changes in the utilities. As there are finite number of states the policy iteration algorithm is supposed to terminate. When there are finite state space, policy iteration algorithm is suppose to yield better results, for large state space, it may take large amount of time. The value function of Bellman's equation is used to find the better policies. The value function is iterated again and again to iterate the policy for finding the best optimal policy.

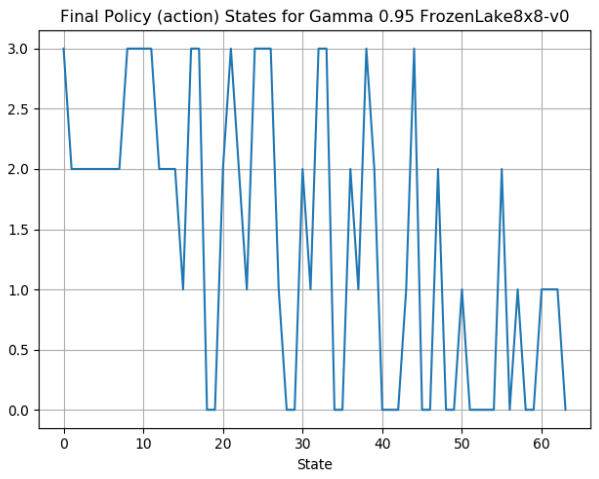
I have adopted similar strategy of varying gamma as done in value iteration. However, varying gamma seems to be more relevant to value iteration as it varies the long team and short term goals, in policy iteration gamma used in value function impacts the new calculated policy.





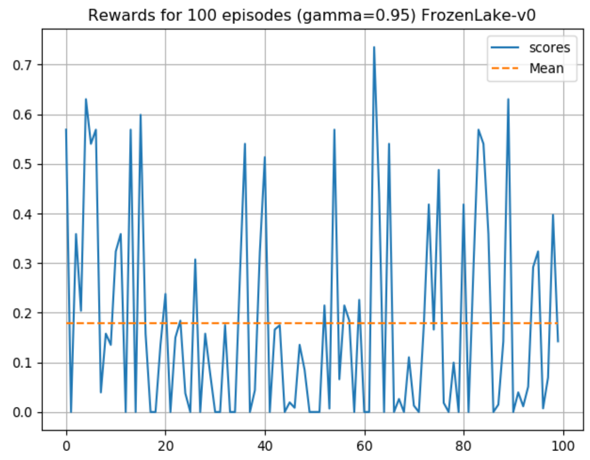
When we compare the frozen lake 4*4 and 8*8 grid for policy iteration, it can be seen that policy iteration took less time then the value iteration. The reward values for both policy and value iteration seems to be same. It can also be seen that the time is little zig-zag and for frozen lake 8*8, at the end time taken is less then the lower values of gamma. This clearly shows that policy iteration converges faster.

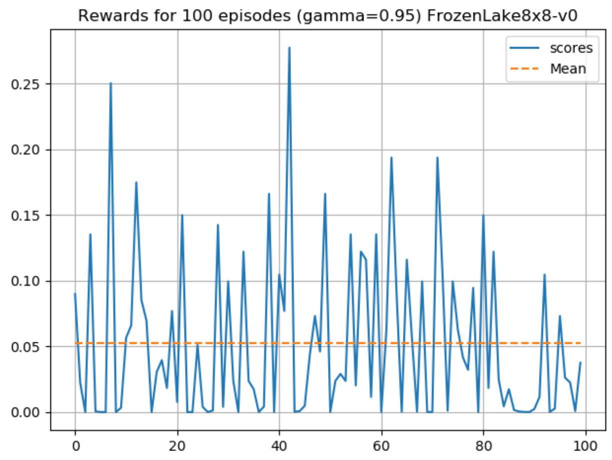




The plot above shows the final policy after improvement of policy. Each value on the y axis shows the action for the agent.

When the algorithm was run for 100 episodes on frozen lake 4*4 grid, it is seen that the highest reward value comes around the same value as the policy iteration. Rewards for 8*8 grid reaches highest values much earlier in policy iteration. This seems to be because even though the number of states are more, the continuous improvement of policy tend make algorithm converge faster.





<u>How many iterations does it take to converge? Which one converges faster? Why?</u>

For frozen lake 4*4 both value and policy, iteration took almost same number of iterations to converge. This can be because as the number of states are only limited to 16 for 4*4 grid.

When the same comparison is made for 8*8 grid, which has 64 states, it is clearly seen that policy iteration algorithm maximized the reward much earlier than the value iteration, which means that agent was able to reach to goal quickly as it learned the better route towards goal. Policy iteration took lot more time than value iteration.

It can be can be said that's as the number of states increases policy iteration seems to perform better than the value iteration because policy iteration involves policy evaluation + policy improvement, which means it takes into account the past results and build over that. Value iteration is value function+ one policy extraction and it is not used again because policy obtained is the optimal policy.

Do they converge to the same answer? Policy iteration performed better than the value iteration, because of continuous improvement of policy at every iteration using value function. As there is continuous improvement and no arbitrary utility state at each state, policy iteration seems to be better. Hence, they do not converge to the same answer. However, policy iteration took more time than value iteration and when computational performance is important, policy iteration seems to be less favorable.

How did the number of states affect things, if at all?

Number of states surely affects the performance of algorithm. For 8*8 grid, the number of states were 64, so it definitely has to take more steps to reach the goal increasing the time to execute. The mean rewards seems to be lower for the grid of 8*8 size. With the increase in number of states, the stochasticity increases, which means more number of obstacles and changes of agent to fall in frozen lake.

Value iteration seems to work in same manner as policy iteration for 4*4 grid, but for 8*8 grid policy iterations seems to perform much better computationally. Value iteration seems to take more time in convergence when underlying policy is not changing.

Q learning: Q learning is the model free algorithm. In this, the learned action value function, directly approximates, the optimal action-value function, independent of the policy being followed. It simplifies the analysis and allows algorithm to converge faster. Value iteration and policy iteration both uses domain knowledge for transition function.

In short, in q learning create a table where we will calculate the maximum expected future reward, for each action at each state. This will let us know the best action to take at each state.

The goal of agent in q-learning is to maximize the cumulative reward. However, agent can fall in the traps, like in frozen lake environment; agent can fall repeatedly in the hole "H" or because environment is slippery, can slip always without reaching the goal. The agent need to do little exploration to find the big reward, and then follow the best found path to reach goal in minimum steps is known as exploration/exploitation trade-off.

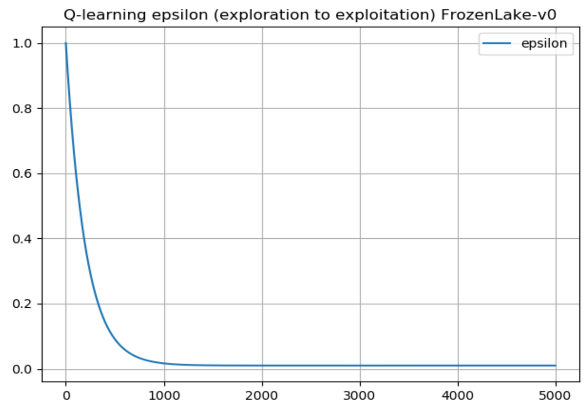
To follow exploration exploitation strategy:-

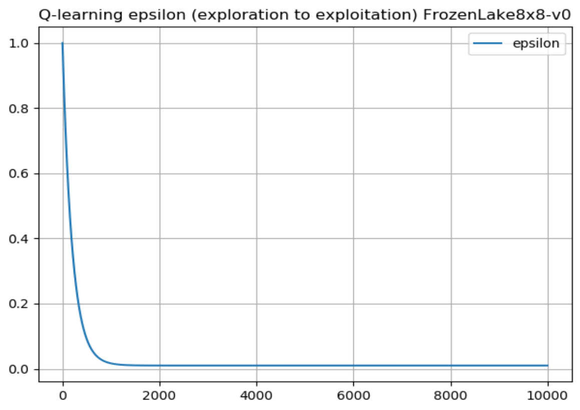
- We set epsilon (exploration rate) to maximum of 1 in the beginning. This is kept at maximum because initially we do not know anything about the environment (q table). We randomly choose our actions and explore the environment.
- We decide exploration or exploitation by randomly generating an number between zero and one and of that number of greater than 1 we do exploitation or else we do exploration.
- The main idea is that there are more chances of exploration at the beginning because the epsilon will be big and then later when epsilon will be less, there will be more exploitation. This makes sure that as the epsilon reduces, agent will become more confident in estimating q-values.

We take an action and observe the outcome state and reward and the update the q-table. We have used the discount factor of 0.95 because it gave the best performance in value and policy iterations also. Also, a small discount factor will make agent myopic by giving more importance to current rewards and a value of 1 or near will make agent to strive for rewards very long making the computation cost very high.

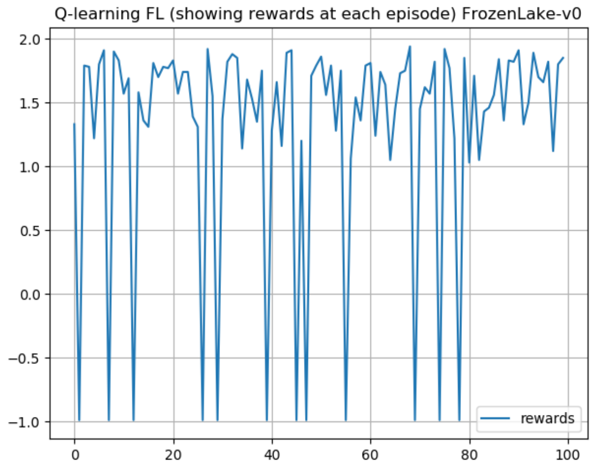
If we see the plots of epsilon, we can see that the plots are almost same for both frozen lake 4*4 and frozen lake 8*8. This is because we have adopted similar strategy for both the grids. However, the number of iterations for 4*4 is less than that of 16*16, because 8*8 has more states and hence more to explore. Epsilon reaches its lowest value bit earlier in 4*4 than the 8*8 grid.

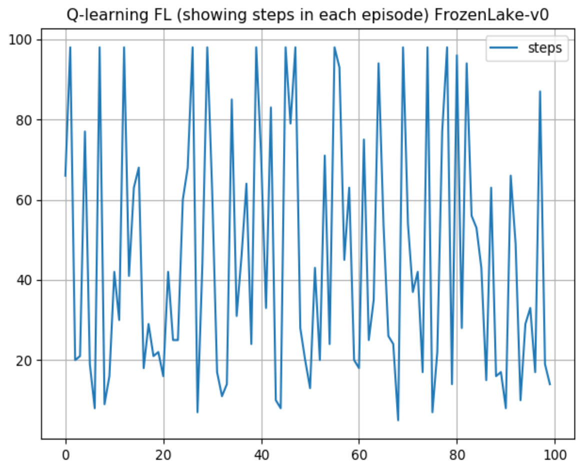
We also adopted a different rewards strategy for the q-learning. For every step it tool, we give agent a negative reward but for positive reward is very high.



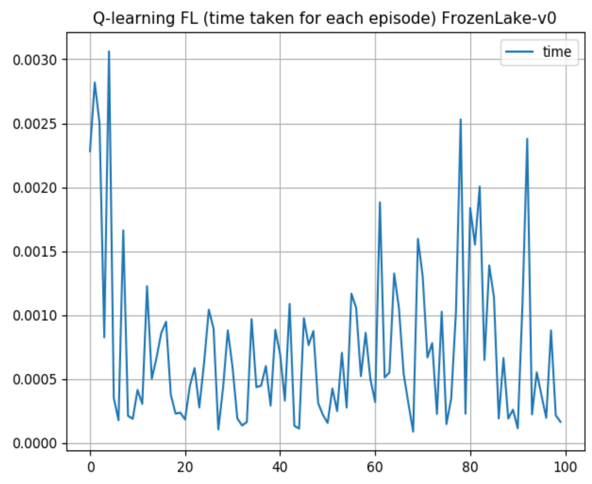


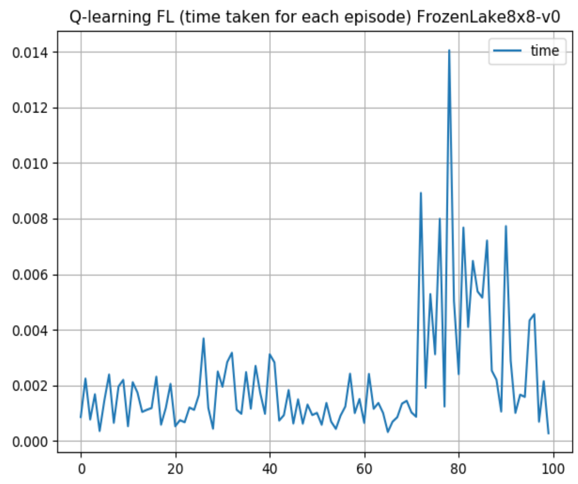
Looking at the rewards for 4*4 grid, it can be clearly seen that agent reached the target many times. All the highest points in the rewards are achieved when agent reached to the goal. There are few points when the rewards were minimum and those are the points when the agent fell in the hole.
However, if we closely observe, the rewards was almost positive and above 1 after episode 80, and the number of steps were also very less. This clearly indicated that the q-table learned enabled the agent to reach the goal in minimum steps and maximizing the rewards. If we closely observe, the number of steps and maximum reward was around episode 93.



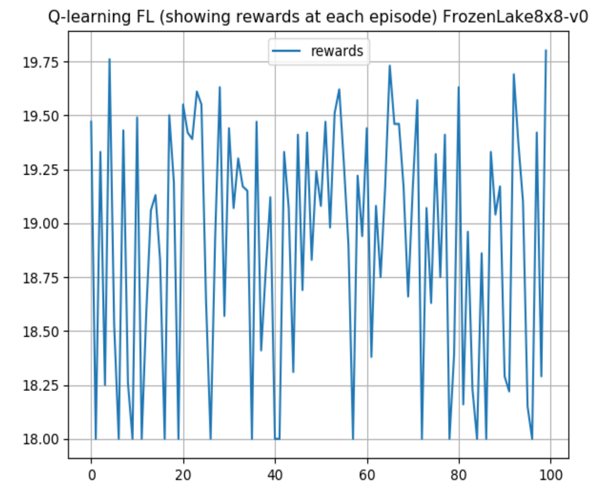


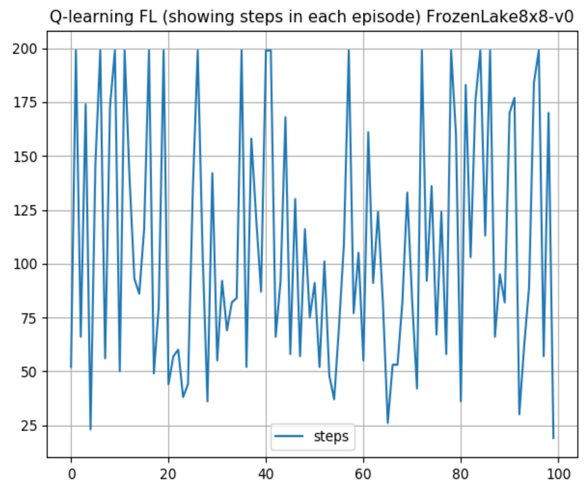
The time chart also shows that the minimum time required for the after to reach the goal was after episode 90. This gives us clear indication that the performance of agent to reach the goal and maximizing the rewards is after episode number 90 (Time chart for 4*4 grid is on next page).





(The charts for Frozen lake 8*8 is on next page) When we see the rewards and number of steps of agent on frozen lake 8*8 grid, it seems that the best results were obtained later (after episode 90). But the main difference in the 4*4 and 8*8 is that agent took lot more steps in 8*8 environment but reached the goal every time. This can be seen because the rewards are never negative, eve though agent has took maximum of 200 steps. Also, it can be seen that when the time was maximum (chart above), the number of steps were also maximum and reward was minimum.





Below are the q-table generated for 4*4 grid. The rows with value of zeroes are holes. The last row has the value of zeroes because it’s the goal. The 4 columns are the 4 actions which an agent can take. A similar queue table is generated for 8*8 grid.

```
[[  1.20902334e-01   9.29675193e-02   4.53196777e-02   4.32641729e-02]
 [  4.38724029e-03   1.20554214e-02   3.93787226e-04   1.35091618e-01]
 [  1.52037364e-02   4.80567051e-03   3.70413938e-02   1.29743879e-01]
 [  8.20870233e-03   4.17828423e-03   1.83544344e-04   5.74279986e-02]
 [  1.34309523e-01   6.30603127e-02   1.88076366e-02   4.88216384e-02]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  4.37947419e-04   5.17721620e-05   6.39340689e-02   4.32066042e-05]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  4.78979082e-03   1.08713256e-02   1.51107333e-02   2.38008281e-01]
 [  5.50561325e-03   3.84173405e-01   5.74343498e-03   5.01855505e-04]
 [  8.34469617e-01   2.32657202e-04   5.14349893e-04   1.49987785e-03]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  1.85714244e-02   6.67617379e-02   5.86564563e-01   9.65430244e-02]
 [  6.88294200e-02   9.31567708e-01   1.60856271e-01   4.25231576e-01]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]]
```

The q learning is a model free algorithm, which means that agent does not knows transition probabilities and rewards. The agent only discovers that there is a reward from going from one state to another using an action. When agent does so, it receives a reward. It finds out what transitions are available from that state by ending up in that state and looking at options.

Conclusion:

In Q-learning, the optimal plan is always to choose an action from a state and action pair. This is very important when we want an agent to be capable of entering any new situation without any prior knowledge and figure out what to do. Q-leaning seems to be good when state is space is too large to repeatedly enumerate. Having an agent to explore by its own can be computationally traceable.

When value and policy iterations are compared, they are more computationally expensive and may not perform as good as q-learning in an unknown environment. Knowing of value and policy may make policy iteration better choice, but policy and value are not always available and needs to be calculated which makes both value and policy iteration less preferable than q-learning.

References:

- https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419
- https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa
- https://en.wikipedia.org/wiki/Q-learning#Influence_of_variables
- Richard Sutton reinforcement learning 2nd edition
- Russell Norvig artificial intelligence a modern approach