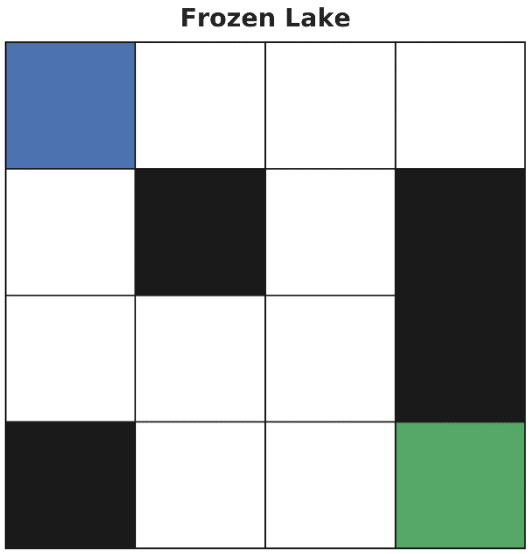
Reinforcement Learning

Introduction

The two problems that I have chosen are the Frozen Lake problem and the Forest Management problem. The Frozen Lake problem has a small amount of states and is represented as a grid world. The Forest Management problem is my large state non-grid world problem.

Frozen Lake

The frozen lake problem has to do with navigating over a frozen lake, avoiding "holes" in the lake and reaching a frisbee that has been erroneously thrown. It is interesting due to its stochasticity with regards to the action chosen when moving on the frozen lake. For my setup, for each action chosen, there is only a 1/3 chance that the action will correctly be followed, a 1/3 chance of heading left of the chosen direction, and 1/3 chance of heading right of the chosen direction. This results in some unintuitive policies that were calculated with the below algorithms. The map of the lake is shown on the right, with the blue square representing the starting point, the white spaces representing ice, the black square representing holes, and the green square is where the frisbee is located.

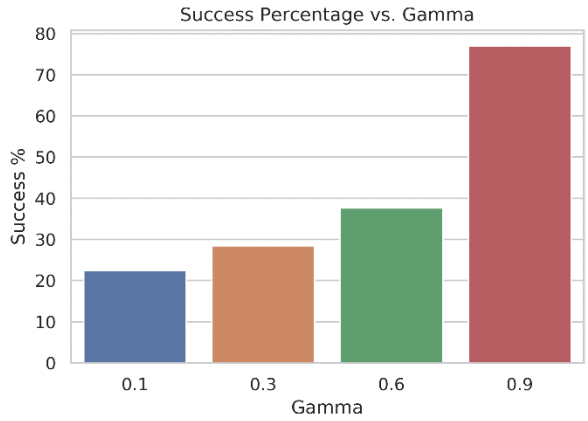


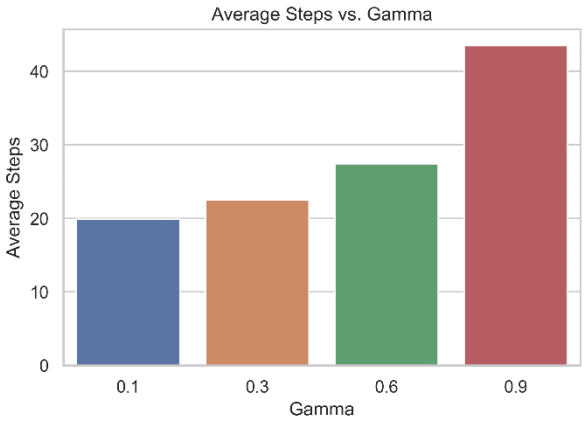
Value Iteration

For value iteration (VI), the true "value" of each state in the gridded lake is found by iterating through each location and having the utility of each state propagate out from its neighbors. Each time through the loop, the utility of the entire problem is calculated, and convergence is defined to be when the delta in utility falls below some given threshold, ϵ. For my approach, I varied both the input discount value (gamma), and the threshold parameter epsilon. The results can be seen for the time for each test as well as how many iterations it took to reach convergence. The overall run time for testing 4 different gammas as well as four epsilon values was only about 0.07 seconds, so the algorithm ran rather quickly.

Once the algorithm converged on a policy, the policy was tested 1000 times to see what the success rate of reaching the frisbee would be as well as how many steps it took to get there. It turned out that the largest gamma value combined with the smallest epsilon value yielded the best results, with an 82.5% success rate of reaching the frisbee in an average of 45 steps.

gamma	epsilon	time	iterations	reward
0.1	**0.01**	0	1	0.333333
0.1	**1.00E-05**	0	4	0.345235
0.1	**1.00E-08**	0	7	0.345239
0.1	**1.00E-12**	0.001	11	0.345239
0.3	**0.01**	0.001001	3	0.373333
0.3	**1.00E-05**	0	8	0.375101
0.3	**1.00E-08**	0.001001	13	0.375103
0.3	**1.00E-12**	0.001001	20	0.375103
0.6	**0.01**	0	6	0.44512
0.6	**1.00E-05**	0.001001	17	0.447647
0.6	**1.00E-08**	0.001001	29	0.447649
0.6	**1.00E-12**	0.003003	46	0.447649
0.9	**0.01**	0.002002	26	0.63754
0.9	**1.00E-05**	0.005005	77	0.639019
0.9	**1.00E-08**	0.008008	128	0.63902
0.9	**1.00E-12**	0.013012	195	0.63902



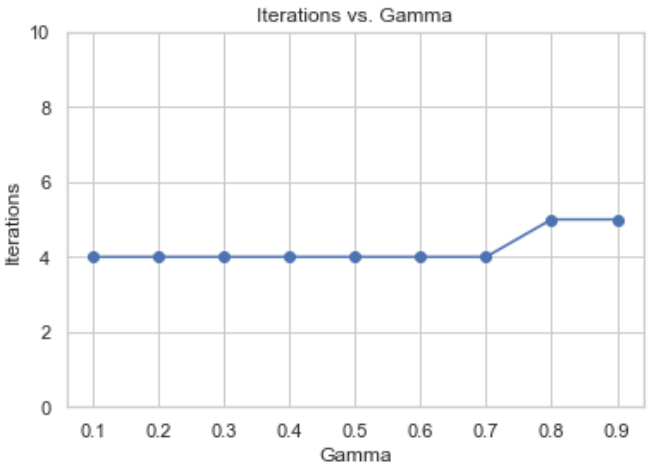


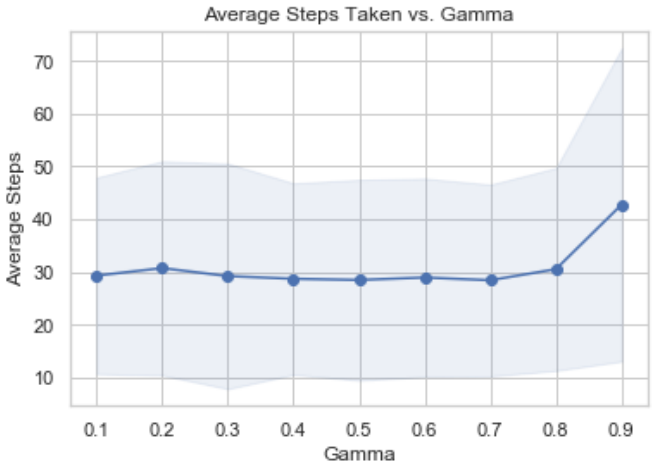
Policy Iteration

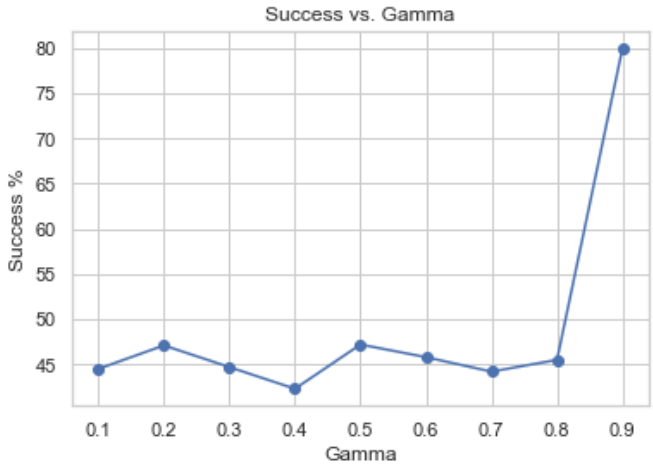
Policy Iteration (PI) was the second algorithm chosen to solve the problem, which is like Value Iteration except that instead of evaluating the utility of each state in the grid, we instead evaluate the utility of the current policy and try to iteratively improve. Convergence is defined as when the delta in policy improvement falls below some threshold epsilon. So, we start with an initial policy and then continue improving and evaluating until the policy is no longer changing. This is beneficial since we can now make jumps in policy space instead of value space.

When the policy converged on an answer for each given discount value, the policy was again tested over 1000 different episodes to gauge performance. The best performance again came with the highest value of gamma, which had an 80% success rate in an average of 43 steps. The 9 different values of gamma tested only took 0.03 seconds to run. This algorithm converged in considerably fewer iterations than value iteration did, only needing 4 or 5 iterations for each test.

gamma	epsilon	time	iterations	reward
0.1	0.0001	0.003004	4	0.345239
0.2	0.0001	0.001994	4	0.358992
0.3	0.0001	0.001	4	0.375103
0.4	0.0001	0.000986	4	0.394332
0.5	0.0001	0.002002	4	0.417861
0.6	0.0001	0.002002	4	0.447649
0.7	0.0001	0.001	4	0.487267
0.8	0.0001	0.002004	5	0.544196
0.9	0.0001	0.002002	5	0.63902







Q-Learning

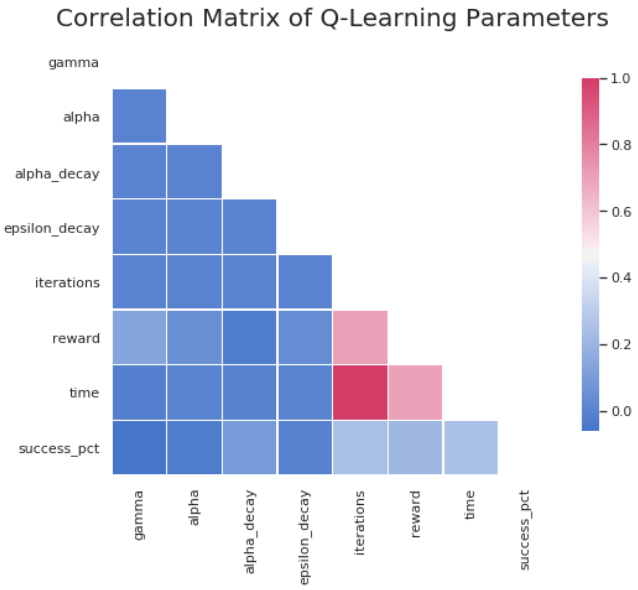
For my favorite reinforcement learning algorithm, I chose to use Q-Learning (QL) with an epsilon-greedy strategy. Whereas the VI and PI algorithms are more so planners that take in models and produce policies, QL attempts to learn what to do by exploring and quantifying the value of being in some state, leaving via a certain action, and proceeding optimally thereafter. This involves the exploration-exploitation dilemma in which we must decide the balance involved in using what we know (exploiting) and getting the data we need so that the algorithm can learn (exploring).

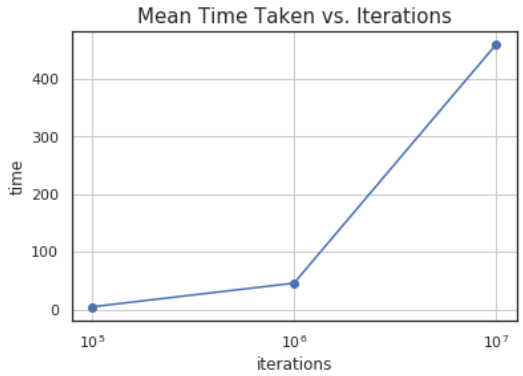
The exploring and exploiting are really determined by two quantities, alpha and epsilon, and their decay as we take more and more actions. Alpha, the learning rate, determines how large of leaps we are willing to take in the search for an optimal policy. The alpha decay dictates how quickly we eat away at that parameter, so we take smaller leaps as more iterations are run. Epsilon in this case represents how "greedy" we are with our current knowledge. A purely greedy method, setting epsilon equal to zero, means that we would take our highest value state each iteration through the loop and trying to find the optimal point from there. This would decrease the amount of exploration that is done which could put us in a local optima.

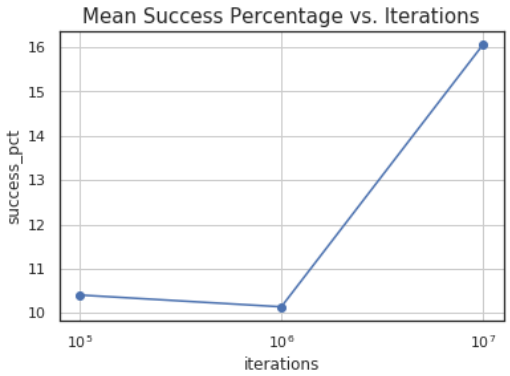
As far as convergence, the QL algorithm was continually run for the amount of iterations, and the policy it reached was returned once the iterations were up.

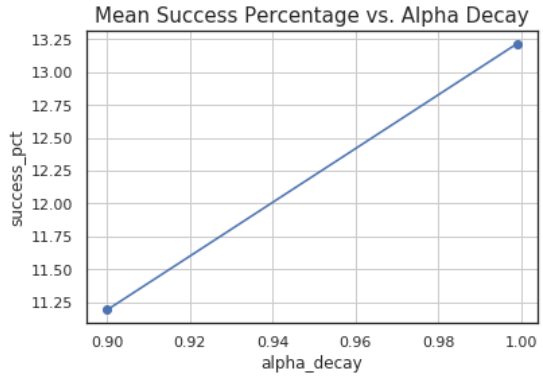
The correlation matrix below shows that the reward and success percentage were most affected by gamma, the alpha decay, and the amount of iterations run. The largest contributing factor to the time, reward, and success percentage was by far the number of iterations that were used.

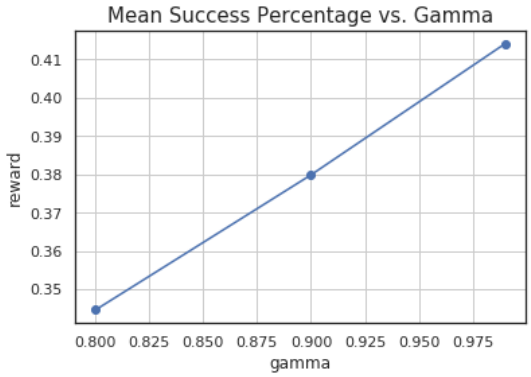
Gammas	[0.8, 0.9, 0.99]
Alphas	[0.01, 0.1, 0.2]
Alpha Decay	[0.9, 0.99]
Epsilon Decay	[0.9, 0.999]
Iterations	[1e5, 1e6, 1e7]







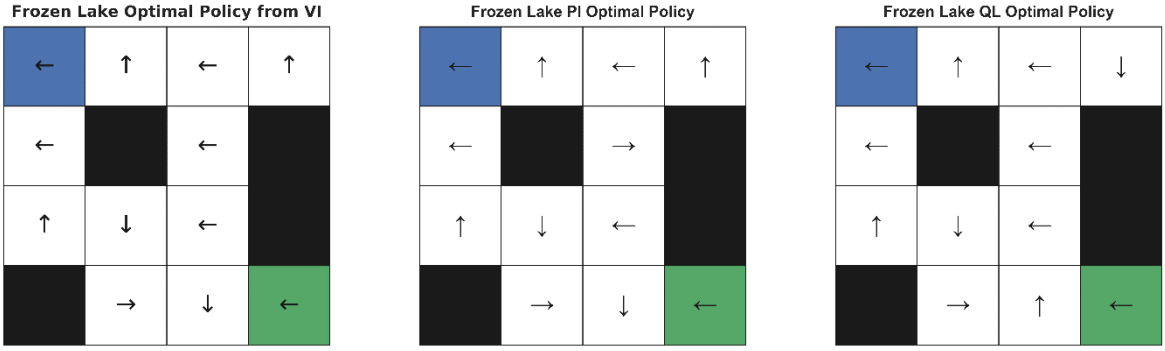




The amount of time needed to run all the tests was over 4 hours, as compared to the VI and PI algorithms which took less than a second. The optimal results produced a success rate of 66% in an average of 46 steps.

Comparison

All the different algorithms converged on just about the same optimal policy in the end, but not the exact same thing. The VI and PI solutions ran very quickly and return two different policies that result in the same reward. This is since the block in row 2, column 3 is essentially equivalent even though the policies dictate different directions due. This is due to the stochastic values chosen having equal weight between the chosen direction and the two orthogonal directions to it. The QL algorithm took much longer to reach its policy. The PI method converged the fastest and with the least amount of iterations due to the algorithm being able to work on improving the policy directly and having a little bit bigger epsilon value. I believe the QL algorithm had a slightly worse score due to the top right state where it chooses to go downwards instead of upwards like VI and PI do. This would have been an edge case of exploration and might could have been solved given more iterations or a larger initial exploration parameter.



With more states, all the algorithms would run and be tested in the same way, but the amount of time to solve the problem would go up. The amount of exploration needed for Q-Learning would most likely need to be increased in order to check every possible path to find the globally optimal policy. A method could also be created instead to check the current policy against several episodes to check for a level of sufficiency instead of trying to find the global optimum. This would help to decrease the amount of iterations needed and hopefully decrease the large amount of time it takes to learn.

Forest Management

For the next problem, a non-grid world problem with a large amount of states (500) was chosen called Forest Management. The problem deals with trying to find the optimal policy of when to cut down a forest, with the two actions that the user can take being to simply “Wait” and do not cut anything, or to “Cut” the forest. There is a reward for both actions, but there is also a

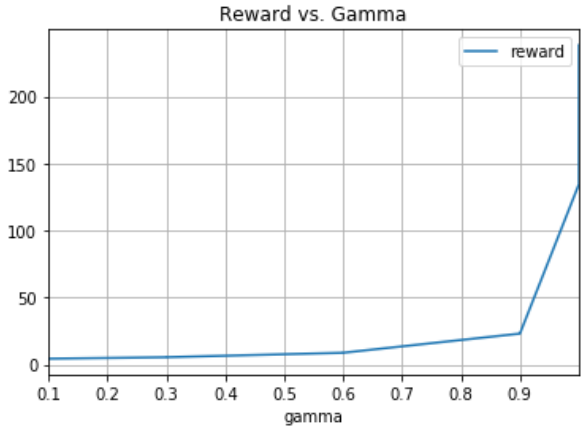
probability of a wildfire that burns down the wood and resets the state to the initial state. This problem is interesting because it is rooted in reality and can help to determine what the optimal course of action may be in order to both 1.) enjoy the wildlife that the forest can create and 2.) not let a fire get out of control and take away both the wildlife *and* the possibility of profiting from the selling of the wood that has been cut down.

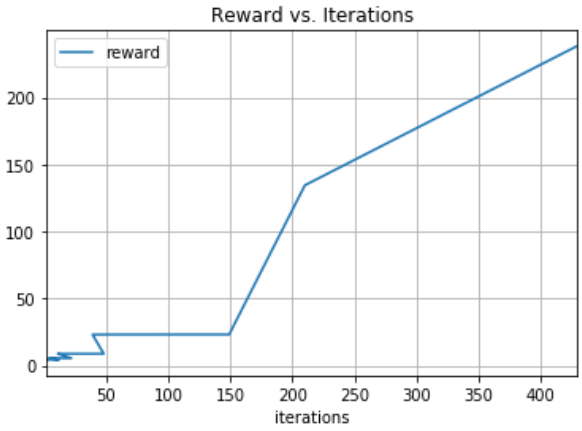
Value Iteration

For a description of VI or how convergence is defined, refer to the Frozen Lake Value Iteration section as they were the same for this problem.

The discount and threshold on the change in the utility value calculated were again varied to try to determine what the optimal policy would be, which took a total of 0.74 seconds. The results can be seen below, with the best result involving the largest gamma and smallest epsilon values.

gamma	epsilon	time	iterations	reward
0.1	**0.01**	0.001	2	4.36
0.1	**0.001**	0.001001	3	4.3933
0.1	**1.00E-08**	0.003003	8	4.396613
0.1	**1.00E-12**	0.003003	12	4.396613
0.3	**0.01**	0.001	4	5.460862
0.3	**0.001**	0.002002	6	5.489575
0.3	**1.00E-08**	0.005004	15	5.491933
0.3	**1.00E-12**	0.008008	22	5.491933
0.6	**0.01**	0.003003	11	8.797055
0.6	**0.001**	0.004004	15	8.808703
0.6	**1.00E-08**	0.008017	33	8.809994
0.6	**1.00E-12**	0.014013	48	8.809994
0.9	**0.01**	0.011001	39	23.08968
0.9	**0.001**	0.019018	50	23.14753
0.9	**1.00E-08**	0.041036	105	23.17236
0.9	**1.00E-12**	0.036043	149	23.17243
0.9999999	**0.01**	0.050036	210	134.4865
0.9999999	**0.001**	0.068062	232	144.9073
0.9999999	**1.00E-08**	0.082075	341	196.5374
0.9999999	**1.00E-12**	0.104094	429	238.22



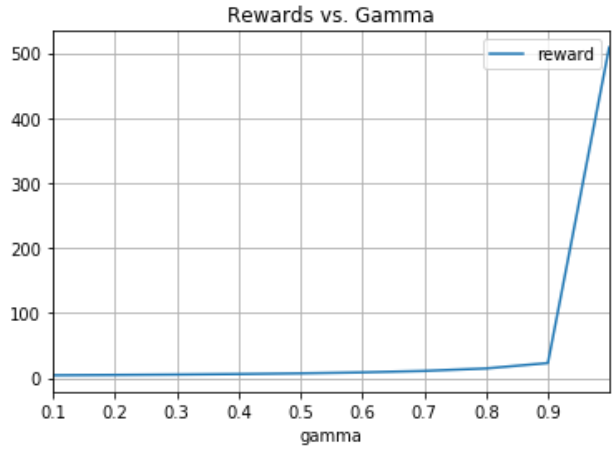


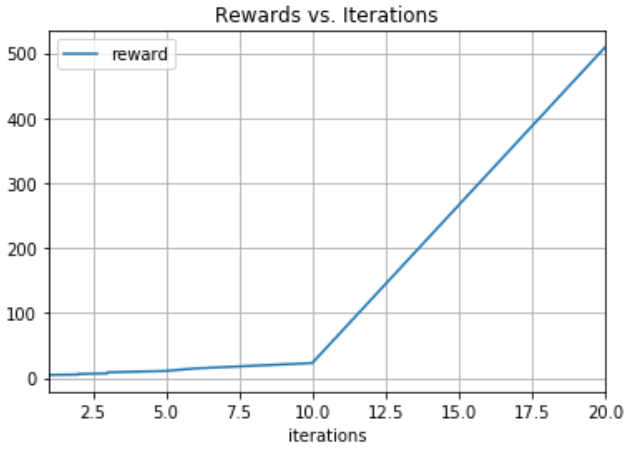
Policy Iteration

For a description of PI or how convergence is defined, refer to the Frozen Lake Policy Iteration section as they were the same for this problem.

Ten different discount values were tested with a constant epsilon threshold of 0.0001 on the change in policy utility value from one timestep to the next. All the tests took 0.7 seconds to run and the best policy resulted again from the highest gamma value which took only 20 iterations.

gamma	epsilon	time	iterations	reward
0.1	0.0001	0.014567	1	4.396613
0.2	0.0001	0.012011	1	4.882699
0.3	0.0001	0.025023	2	5.491933
0.4	0.0001	0.025023	2	6.277574
0.5	0.0001	0.038034	3	7.329154
0.6	0.0001	0.038034	3	8.809994
0.7	0.0001	0.062057	5	11.05455
0.8	0.0001	0.076069	6	14.88372
0.9	0.0001	0.123112	10	23.17243
0.999	0.0001	0.25223	20	508.3859



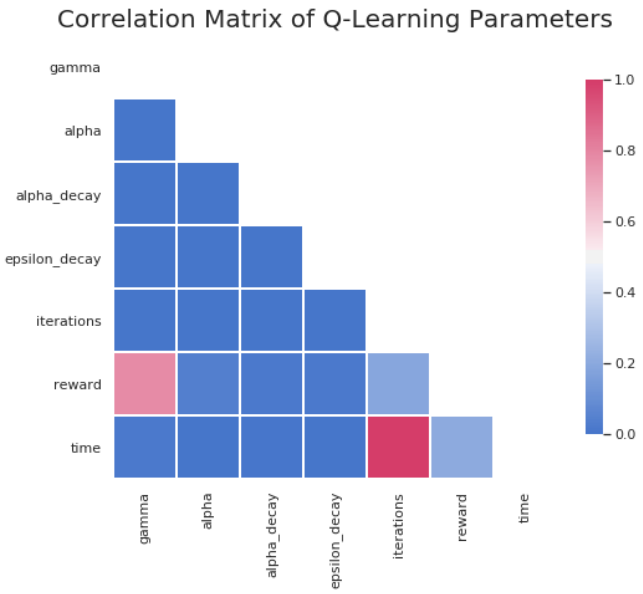


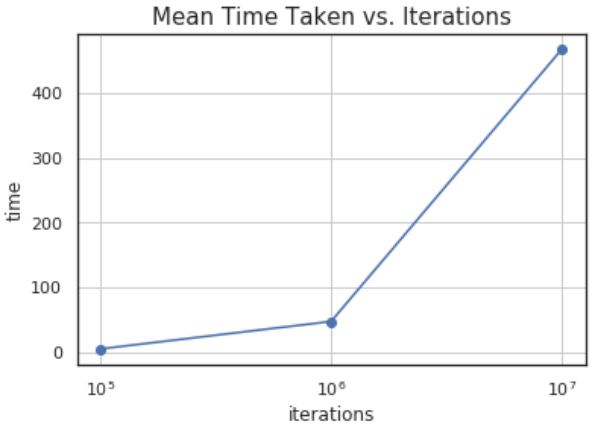
Q-Learning

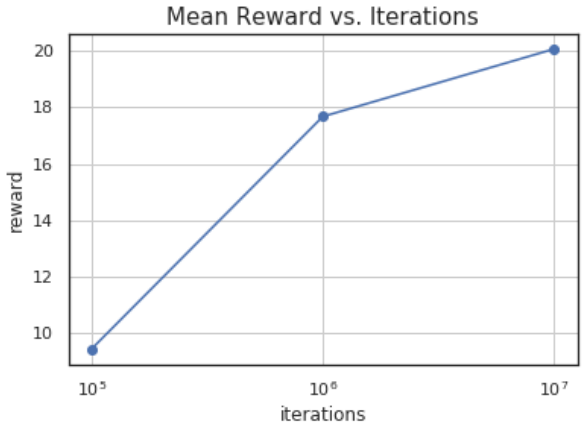
For a description of QL and the strategy used, refer to the Frozen Lake Q-Learning section as they were the same for this problem.

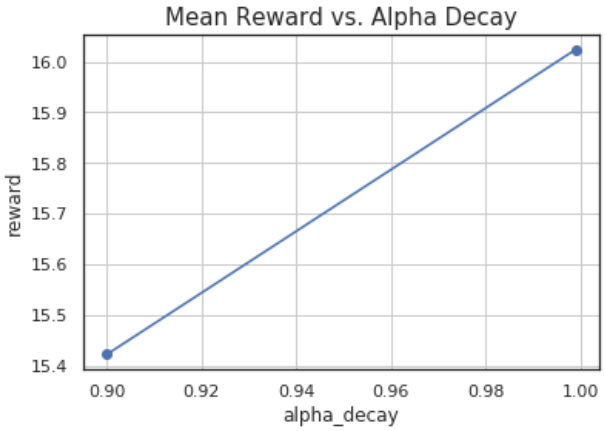
The same inputs were explored for QL for this problem as well, as can be seen in the table below. A correlation matrix was again created to try and investigate which of the parameters had the largest effect on the resulting reward gained and the amount of time it takes to run the algorithm.

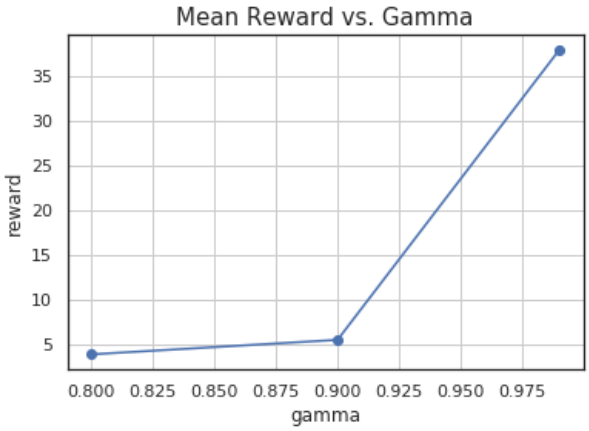
Gammas	[0.8, 0.9, 0.99]
Alphas	[0.01, 0.1, 0.2]
Alpha Decay	[0.9, 0.99]
Epsilon Decay	[0.9, 0.999]
Iterations	[1e5, 1e6, 1e7]









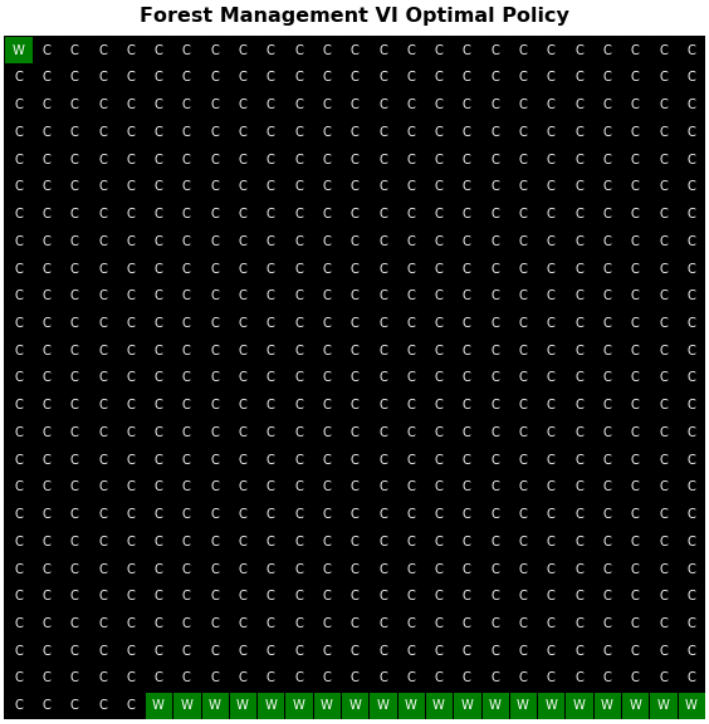


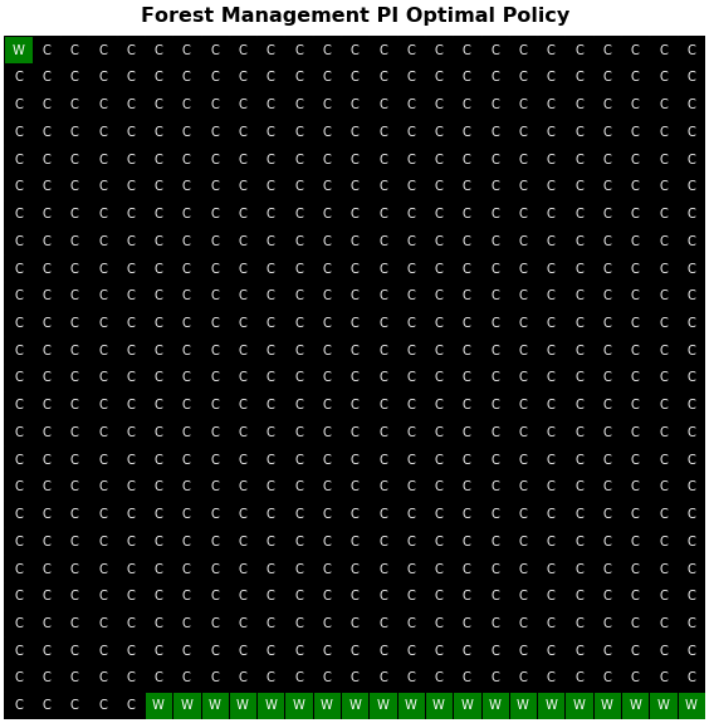
As can be seen from the correlation matrix plot, gamma and the number of iterations had the largest effect on gaining a higher reward from the policy. The algorithm took over 4 hours to run, with the number of iterations having the largest effect on time. The most successful run resulting in a reward of almost 48, and came from following values:

- Gamma = 0.99
- Alpha = 0.01,
- Alpha Decay = 0.9
- Epsilon Decay = 0.999
- Iterations = 10,000,000

Comparison

The resulting policies for the VI and PI algorithm both converged to the same policy, as can be seen in the following two charts. The policy is actually a 1-D array that has just been presented here as 2-D for the sake of fitting it on a page. The green boxes with a 'W' denote a "Wait" action, and the black boxes with a "C" signify a "Cut" action.



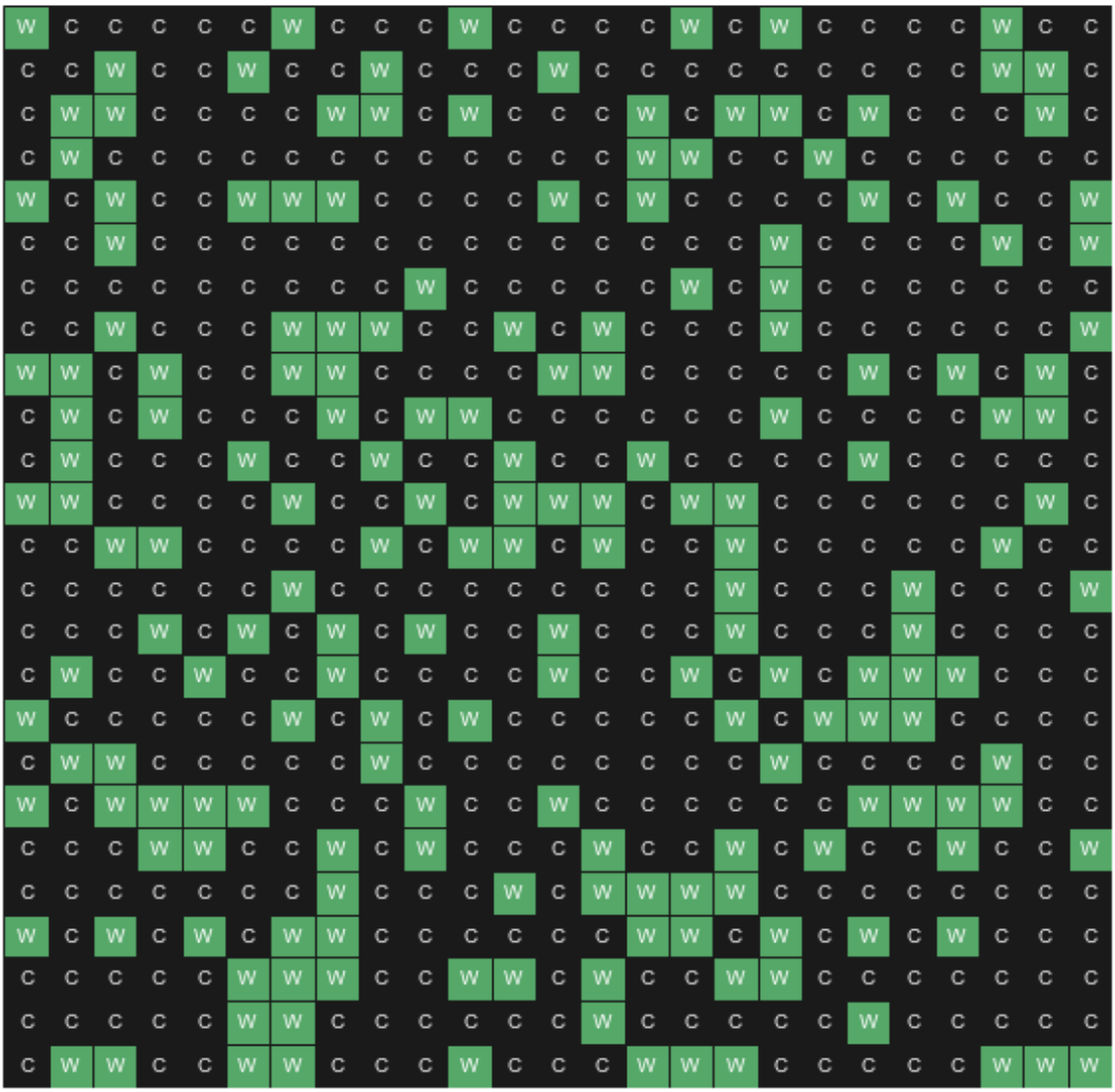


It turns out that it seems like the optimal thing to do is to simply cut every year and then to wait it out near the end of the policy. If given more time, it would be interesting to tweak the reward values for the "Wait" and "Cut" action to get a clearer picture of how the algorithms are choosing what to do. One could also determine the relationship between the probability of fire and the optimal policy to get a policy that could fit for different regions for a more real-world application.

Both the VI and PI algorithms ran very quickly, with the PI running just a little bit faster. The PI algorithm took the least amount of iterations to run.

The QL approach seemed to struggle a little bit more, taking a larger amount of time and resulting in a much lower reward value. As can be seen with the 'Mean Reward vs. Iterations' plot in the Forest Management Q-Learning section, a larger reward might could have been found with more iterations as the trend line is still going up.

Forest Management QL Optimal Policy



With fewer states, the Q-Learning algorithm would have had a better chance at finding a more optimal strategy as the exploration space would be much smaller. The VI and PI algorithms would again be able to quickly find the optimal answer since they have all model and reward knowledge already.