An Analysis on Markov Decision Processes and Reinforcement Learning

Ryan Reed

1 – Selection of two MDPs

For this assignment I am using two separate mazes provided by the RL_Sim package, particularly the small and medium mazes. Starting with something simple such as a maze allows someone such as myself to grasp the concepts within MDPs and RL on a small scale while still understanding how they work. Mazes are also interesting ML problems because they are a very easy way to apply theory into a real world setting. In ML I believe it is especially important to start small, understand the concepts on a small scale, and use that knowledge to expand and develop currently existing knowledge. As kids we all have completed mazes, beaten Atari games, and played checkers with friends. By understanding the thought process behind these games, i.e. how they are played and completed, and utilizing ML learning to solve these tasks, they can provide insight into larger AI problems. Mazes also have tremendous flexibility. They can be extremely challenging or very straightforward, so it will be interesting to see how the structure of the maze impacts MDP algorithms, if at all.

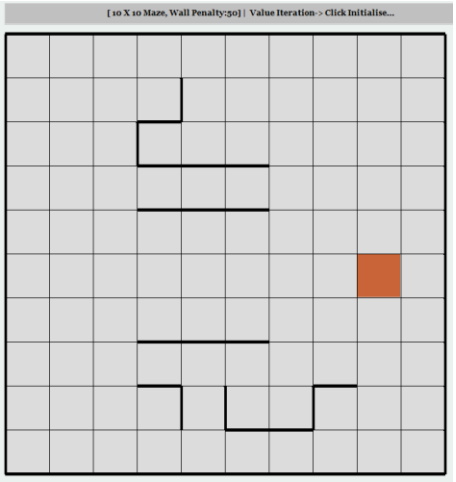

Figure 1a – Medium maze (10 x 10) provided in the RL_Sim package, 100 different states

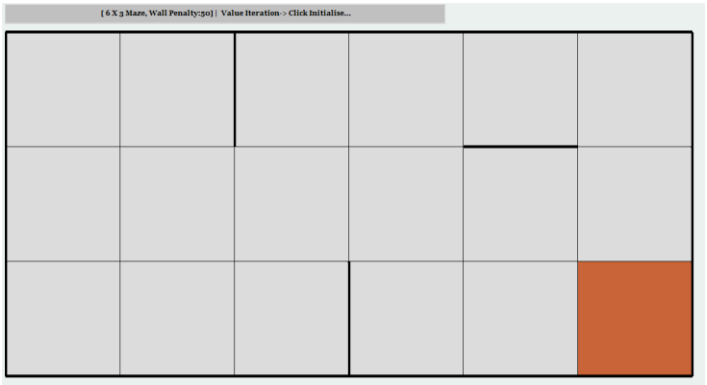

Figure 1b – Small maze (6 x 3) provided in the RL_Sim package, 18 different states

2 – Value Iteration

Comparing convergence properties for the two state spaces yields interesting results at an initial glance. The two state spaces appear to have similar convergence rates when altering the

PJOG metric which “models noise in the environment….PJOG = 0.3 implies that 70% of the times an action produces the intended next state and 30% of the times the action causes the agent to get pushed against its wall to any of its neighboring states.”

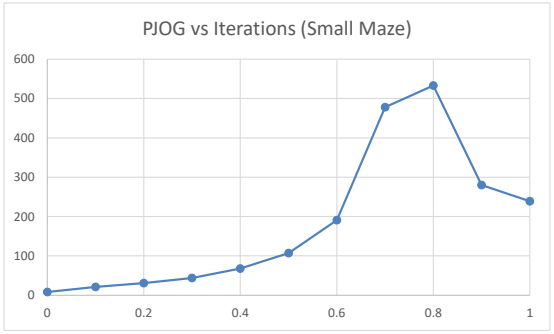

Figure 2a – PJOG vs Iterations to converge for the Small Maze (Precision set at .001)

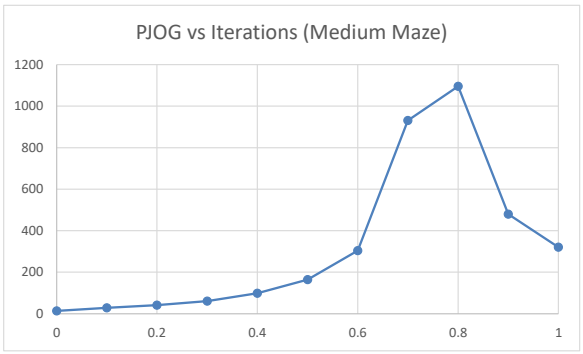

Figure 2b – PJOG vs Iterations to converge for the Medium Maze (Precision set at .001)

Comparing figures 2a and 2b we can see that the larger state space of the medium maze adds a couple hundred more iterations at higher PJOG values. Clearly an increased state space *does* have an effect on convergence rate, likely because more state utility values means more iterations due to the increased evaluations for each individual state. However, it is important to note that both the small and medium size graphs are very similar in *structure* as displayed in in Figures 2a and 2b. Looking at both graphs we can see a huge increase in iterations taken at a JPOG of 0.7. This is because at this probability, the probability distribution of actions is basically uniform for all directions. Thus, the agent essentially is conducting a random walk and

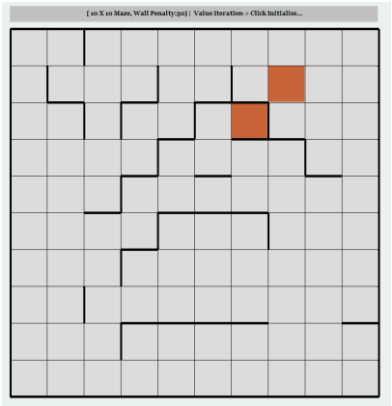

Figure 2c – Medium complex maze (10 x 10) provided in the RL_Sim package, 100 different states, 2 goals

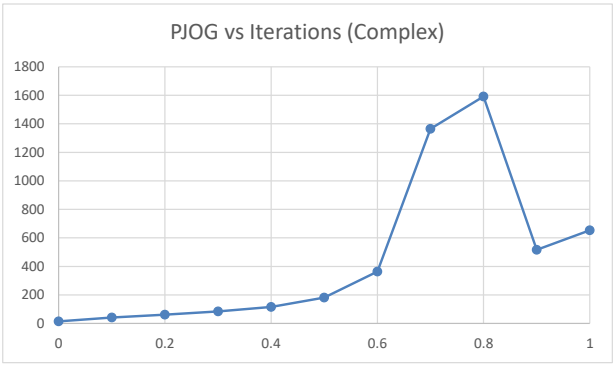

Figure 2d – PJOG vs Iterations to converge for the Medium Complex Maze (Precision set at .001, displayed in Figure 2.1c)

the optimal policy takes much longer to evaluate at each state. As we approach a PJOG score of 1 however, the probability distribution resides within the 3 remaining directions so we see a decrease in iterations as the agent only has 3 options to choose from, which takes less iterations to evaluate than a truly random walk.

Because of the similar graph structures between the two state spaces, I could not discern if the size of the state space was the only contributing factor. The initial mazes contain a few walls (with penalty 50) and a goal state, but nothing too complex. Thus I checked to see if the convergence rate was the same with a more complex maze, using the simplicity of the first two as a control. The more complex maze, provided by the RL_Sim package, contained several more walls and two goal states, and I checked to see if there were any differences in the number of iterations to converge. However, as displayed in Figure 2d, the graph structure does not really deviate from the simple medium maze, albeit with some increases in iterations. These increased iterations are likely due to the multi goal structure of the maze in the more complex version. Utilities have 2 goals to consider, and thus their expected cumulative rewards will correspond to *both* goals, thus taking more iterations to converge. Regardless, the overall structure does not change. The PJOG score, or randomness, is really the significant factor which can drastically change an MDP's number of iterations until convergence, as well as its policy.

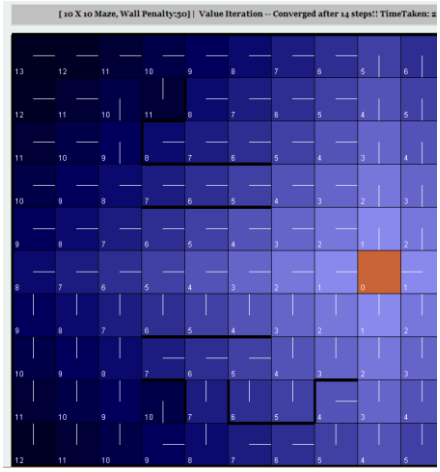

Figure 2g – Optimal solution for medium maze using value iteration (PJOG = 0)

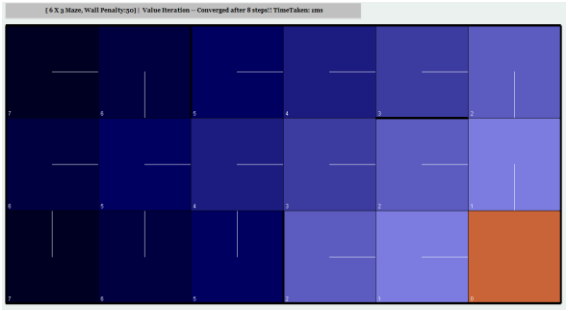

Figure 2h – Optimal solution for small maze using value iteration (PJOG = 0)

Finally, analyzing the actual solutions obtained from value iteration, higher utility values are given near the goal state. It is important to note that, because PJOG is 0, the walls do not have a significant impact on the solution as they would with a PJOG >0. I will discuss this in the next section.

3 – Policy Iteration

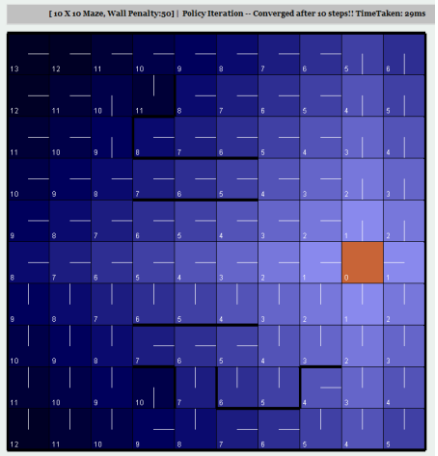

Figure 3a – Optimal solution for medium maze using policy iteration (PJOG = 0)

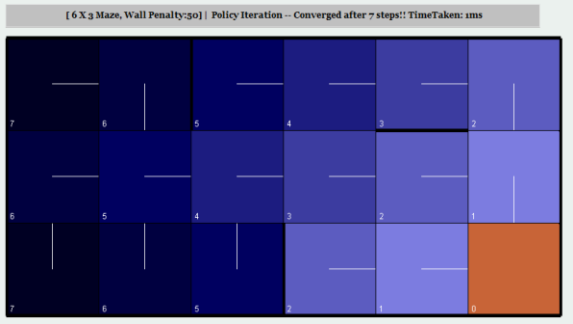

Figure 3b – Optimal solution for small maze using policy iteration (PJOG = 0)

Utilizing policy iteration now instead of value iteration, Figures 3a and 3b show how this method converges to the same solution as value iteration (Figures 2g and 2h). This is expected, because value and policy iteration both compute the same optimal utility values and polices for each state, but just take different approaches, with value iteration using a max over all of the actions, while policy iteration uses a fixed policy and attempts to improve the policy over several iterations.

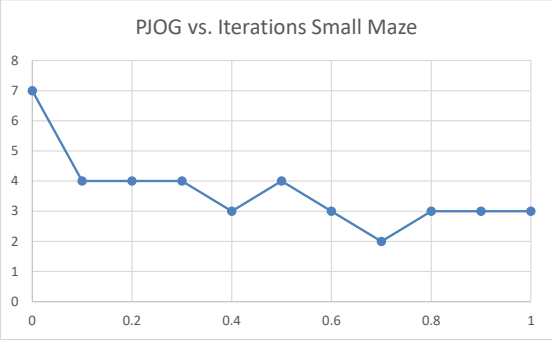

Figure 3c – PJOG vs Iterations to converge for the Small Maze (Precision set at .001)

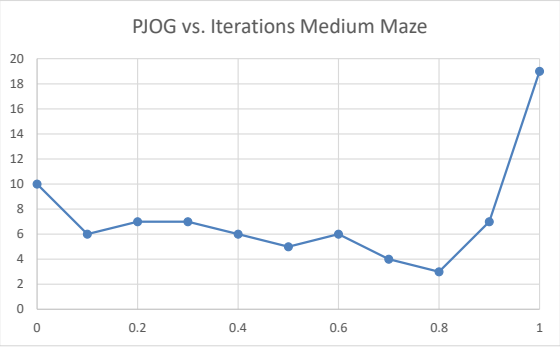

Figure 3d – PJOG vs Iterations to converge for the Medium Maze (Precision set at .001)

The convergence rates for policy iteration are, in contrast, quite different than value iteration. Ignoring PJOG values of 0.9 and 1, both state spaces have similar structures for their convergence rates, with downward trends as the action probability distribution approaches completely random (0.7-0.8). The difference in overall iterations is similar to value iteration as we see slightly more iterations for the medium size mazes with more state spaces. However, as

we approach true randomness, the number of iterations to converge using policy iteration *decreases*. This is likely because since we are updating the fixed policy to find improvements, a random walk does not leave much room for improvement over all of the states, thus why we see such a fast convergence rate. In fact, it is apparent that when any randomness is involved, the number of iterations to converge is less than when no noise is included at all. If we look at

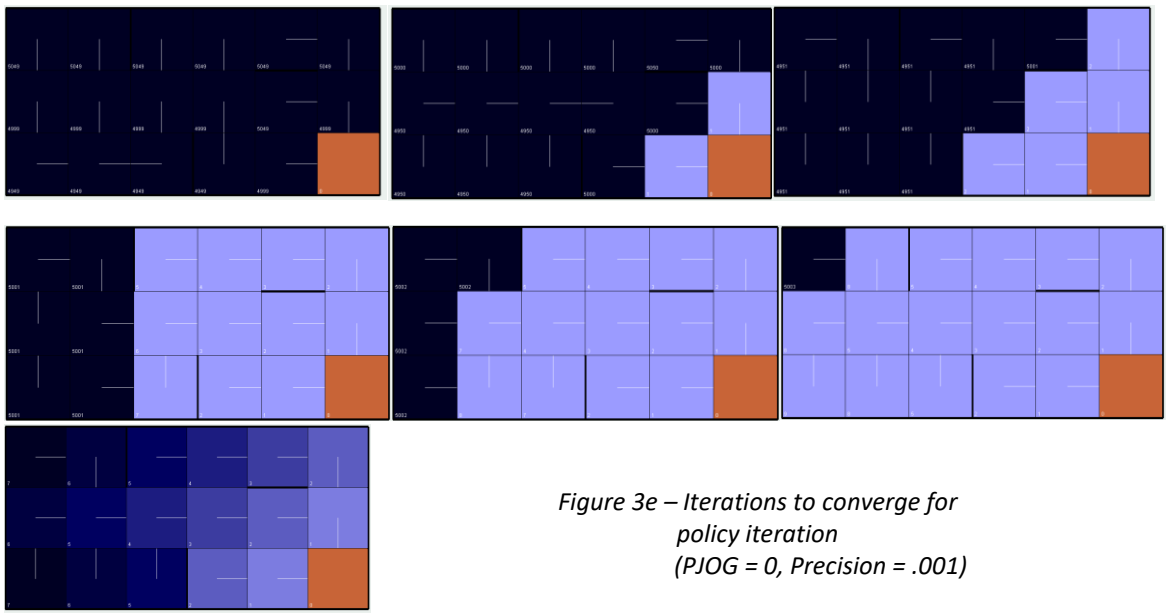

Figure 3e – Iterations to converge for policy iteration (PJOG = 0, Precision = .001)

the differences in solutions for the small maze between a PJOG of 0 and a PJOG > 0 (Figures 3e and 3f), we can see that randomness limits the options of the agent. The "optimal" solution restricts the agent because it wants to achieve the max *expected* utility, thus it must avoid walls at all costs. Looking at the state directly up and to the left of the goal state, we can see how noise creates different optimal solutions from its no-noise counterpart. In the no-noise solution, the optimal policy is to go right, as it has no probability of hitting the wall. In the PJOG = 0.4 solution however, the optimal policy is to avoid the wall at all costs, thus it chooses down as its optimal policy. This creates reduced iterations because for many states the optimal policy

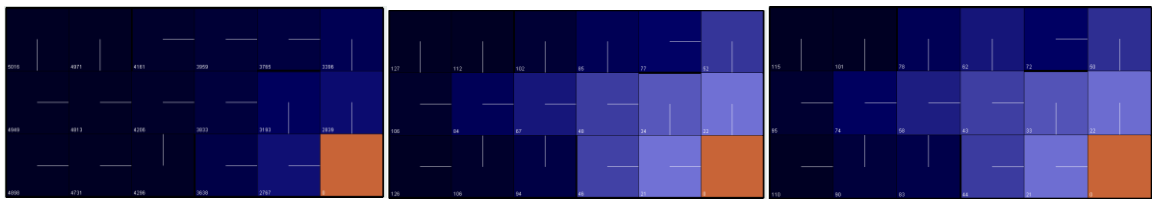

Figure 3f – Iterations to converge for policy iteration (PJOG = 0.4, Precision = .001)

is determined very quickly: avoid walls at all costs.

Additionally, it is interesting to note how the no-noise solution reaches the optimal solution, and why it takes more iterations. In the steps in Figure 3e, we see how each time the policy is updated, it improves upon the farther states more and more until it converges to the

optimal solution. However the added noise in Figure 3f allows for optimal policy to be reached very quickly, likely because an "arbitrary" policy is not far off from an optimal solution, i.e. an arbitrary solution corresponds with the randomness of the high PJOG mazes.

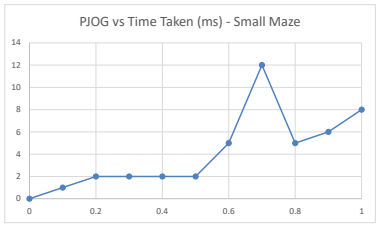

Figure 3g – PJOG vs Time Taken, small maze value iteration

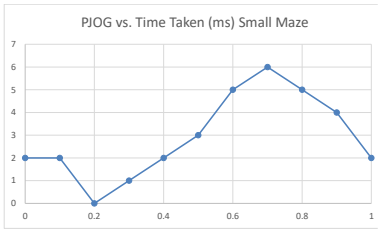

Figure 3h – PJOG vs Time Taken, small maze policy iteration

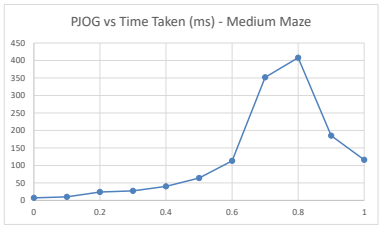

Figure 3i – PJOG vs Time Taken, medium maze value iteration

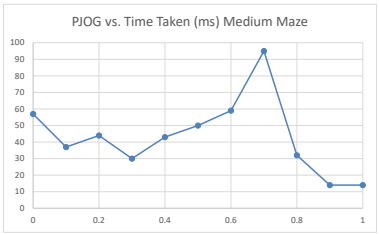

Figure 3j – PJOG vs Time Taken, medium maze policy iteration

Finally, figures 3g – 3j display the time taken for the small and medium mazes among both methods. Interestingly, along with taking less iterations, policy iteration is significantly more efficient, reaching an optimal solution in much less time. This can be attributed to the fact that value iteration requires all values to converge before reaching the optimal solution, which often takes a significant amount of time. Policy iteration takes advantage of the fact that the policy often converges far before the values do. It also only considers one change in policy, not a change for each state. These differences account for policy iteration's considerably faster performance.

4 – Q-Learning

Finally, instead of using value iteration or policy iteration to find the solution, I used Q-Learning to "solve" the small and medium maze MDPs. Q-Learning (and thus reinforcement learning) is very much like unsupervised learning because we do not have prior domain knowledge that we can exert onto the model through known rewards. Instead we are assuming that an MDP exists and find the optimal policy by observing actions and given rewards and forming an optimal policy. Thus, I will be using my policy and value iteration utility values as "ground truth" values when analyzing Q-Learning.

For a true base case, I set PJOG to 0 and tested how fast each maze would converge to an optimal solution based on the value iteration solution. In order to test this, I checked the Q values of the optimal directions (which I obtained from the value iteration solution) and took the average difference of 3 states' utilities between Q-Learning and value iteration (specifically, the 3 states directly next to the goal state). In figures 4a- 4d I display the average difference for both the small and medium sized mazes with epsilon values of 0.1 and 05. Interestingly, each maze only needed a few episodes before converging to an optimal or near optimal solution. As I will describe in my next section, this is largely because of the PJOG value of 0. Q-learning, with no randomness involved, quickly learns the underlying model and ideal policy without issue, because the agent will always act as it is intended, and thus rewards and expected maximization of those rewards are quickly computed.

Analyzing differences between state spaces, it is clear that a larger state space affects the number of iterations in Q-learning. I can hypothesize this is because q-learning tests rewards for different states, and thus more states means more actions to test which means more iterations.

Altering the epsilon, although only over 5 iterations, also had an impact on convergence. An increased epsilon in both cases increased convergence time, likely because Q-learning was taking time exploring the entire state space, thus delaying finding an optimal policy. I also believe the epsilon had a larger impact on the medium sized maze. In the medium maze, the solution did not converge after 5 episodes, and had a slower decrease in difference when the epsilon was at 0.5. I believe increasing the state space likely has an effect on q-learning's ability to 'learn' quickly, as it needs to evaluate rewards for each policy. However, I believe increasing the epsilon helps the RL model overall learn faster because it will explore the state space and learn the rewards of the overall model much more quickly.

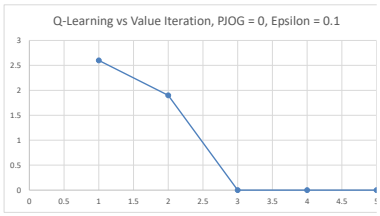

Figure 4a – Average Difference of Utility, small maze, PJOG = 0, Epsilon = 0.1

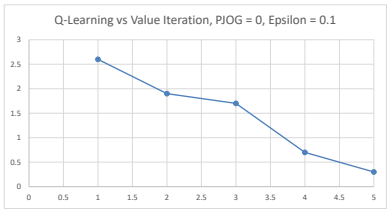

Figure 4b – Average Difference of Utility, small maze, PJOG = 0, Epsilon = 0.5

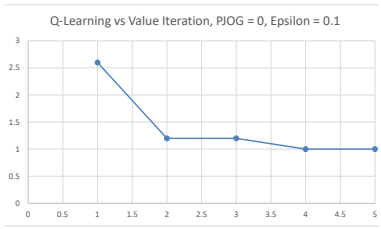

Figure 4c – Average Difference of Utility, medium maze, PJOG = 0, Epsilon = 0.1

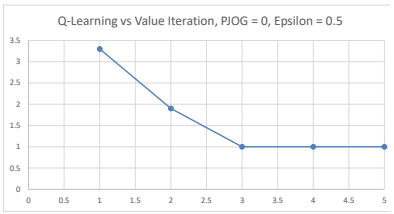

Figure 4d – Average Difference of Utility, medium maze, PJOG = 0, Epsilon = 0.5

As stated, I was curious to see how randomness would affect Q-Learning's ability to learn the underlying structure of the MDP, and based on Figures 4e and 4f, it had a very large impact. I attempted to find the average differences over several episodes, but there was no evident convergence to toward the optimal solution. Thus I upped the number of cycles to 1000 per iteration before finally seeing results. This alone unveils how an even marginally high PJOG value affects q-learning, as it requires many, many more iterations than a no-noise model. This is very intuitive – if randomness is a factor, Q-learning initially has no idea how the model is behaving or why it is behaving the way it is. There is no solid structure in the agent's actions, and it takes longer to learn.

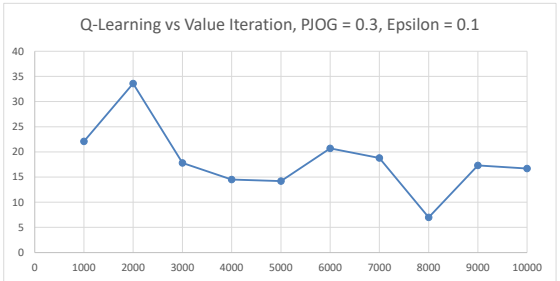

Figure 4e – Average Difference of Utility, small maze, PJOG = 0.3, Epsilon = 0.1

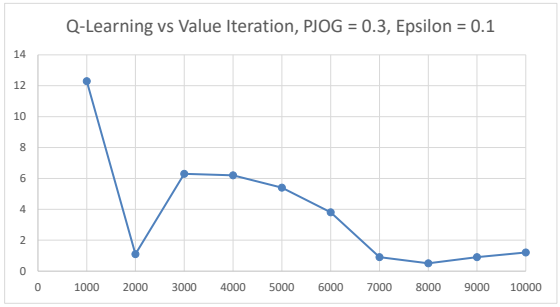

Figure 4f – Average Difference of Utility, medium maze, PJOG = 0.3, Epsilon = 0.1

5 – Conclusion

Overall Q-learning did not perform terribly in comparison to value and policy iteration. However, as I mentioned this is likely in part due to the size of the state spaces I used. In a maze with 2000 states and a high JPOG value, q-learning would very likely take forever to learn the underlying structure and create an optimal policy. This really emphasizes the importance of domain knowledge in mdp's and machine learning in general. Domain knowledge allows us to mirror real world situations very closely. This allows our algorithms to be precise and efficient, as was displayed here when comparing value/policy iteration vs. q-learning. Additionally, as an IE it was interesting to see how Markovian principles and probability distributions can intertwine with machine learning. Randomness, although it throws a wrench in many of our algorithms, is a huge part of real world settings and another aspect of domain knowledge. Since these algorithms help us take a more direct approach with machine learning by interacting with the world, I felt it was important to mention this. Taking all of this into account, MDPs and reinforcement learning are a much more 'hands on approach' to machine learning. They solve very cool problems (like DeepMind) and I'm excited to explore and use them more in the future!