Brad Ware
Markov Decision Processes
CS 4641 Project 4

# Markov Decision Processes – MDP's

MDP's are mathematical framework problems that model decision-making in multi-step situations where uncertainty is present. The uncertainty about MDP's is that when the agent takes an action from a state, probability is incorporated into the next state he enters. The goal is for the agent to navigate the world to the goal state from start state using a set of actions. But, each state transition satisfies the Markov property where they are completely independent of each other and previous actions. If a sequence of actions leads the agent to the goal state then the sequence is referred to as a policy. MDP's represent many optimization problems and can be solved through dynamic programming and reinforcement learning.

# MDP & Algorithms

In this paper I will be exploring MDP's through 3 algorithms: Policy Iteration, Value Iteration, and Q-learning. Policy and Value Iteration are both planning/dynamic-programming algorithms that require knowledge of the MDP problem. Q-learning is a Reinforcement Learning (RL) technique that actually doesn't require any previous knowledge about the MDP. For my MDP problems I used two mazes, one 10x10 and the large being 45x45. Both of these mazes were found under the RL_SIM package on Carnegie Melon's course website, and the code to run Value Iteration, Policy Iteration, and Q-Learning was found there too. The goal of this analysis is to analyze all three algorithms running on large and small MDP problems.

# Value Iteration

Value iteration is an iterative method for computing an optimal polity for a given MDP. It gives a random utility for each starting out, and then computes the utility of each state starting from the goal working backwards to the current state. As it works backwards it refines the estimates of the utility for each state. It repeats this process until the computed utility at the current state is 'close enough' to the original utility. Value iteration is not an efficient algorithm as it can have an exponential runtime. In all of my value iteration data, the precision threshold was 0.001. In value iteration, I manipulated the transition probability of the agent to explore a random state (not chosen). I iterated the value starting at 0 through 0.8.

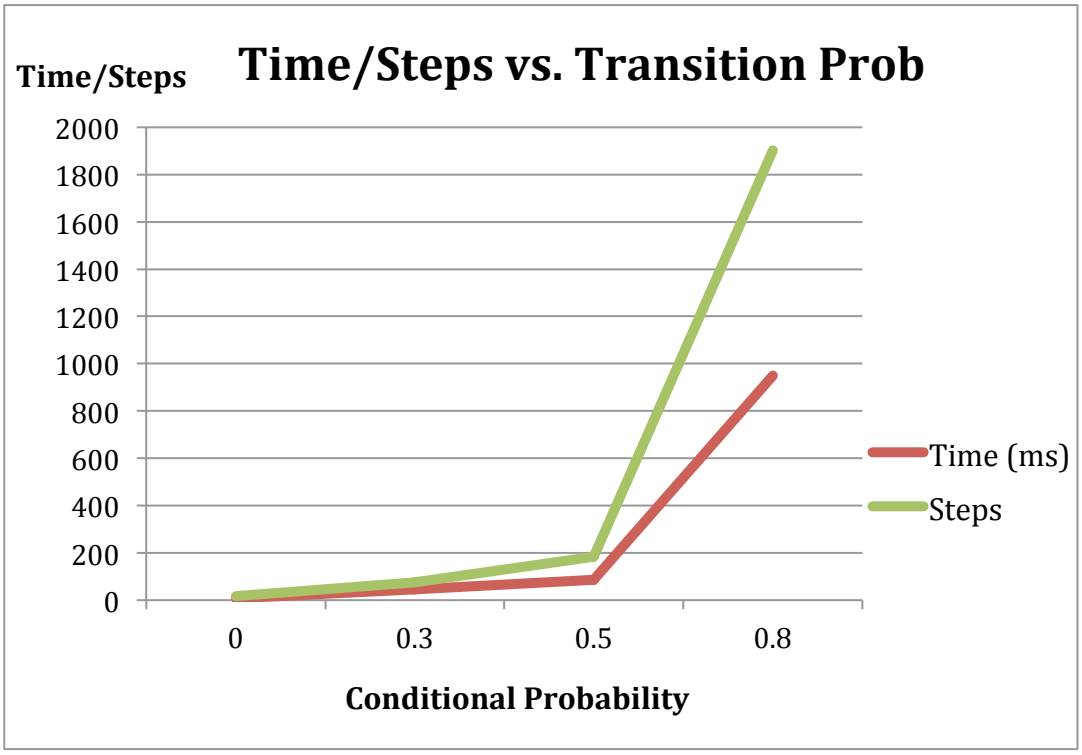
# Value Iteration – Small Maze

For value iteration, I hypothesized that as the transition probability increased, the time and steps taken would as well. A table on the next page captures my results of steps and time taken as the transition probability is manipulated. As you can see,

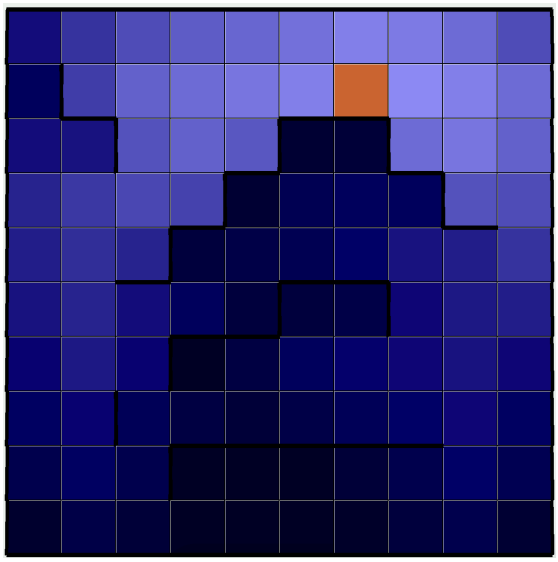
not only did raising the transition probability increase the time and steps, but it also was an exponential increase! This seems very logical as in a deterministic world as it's very simple to dictate the optimal moves toward the goal.

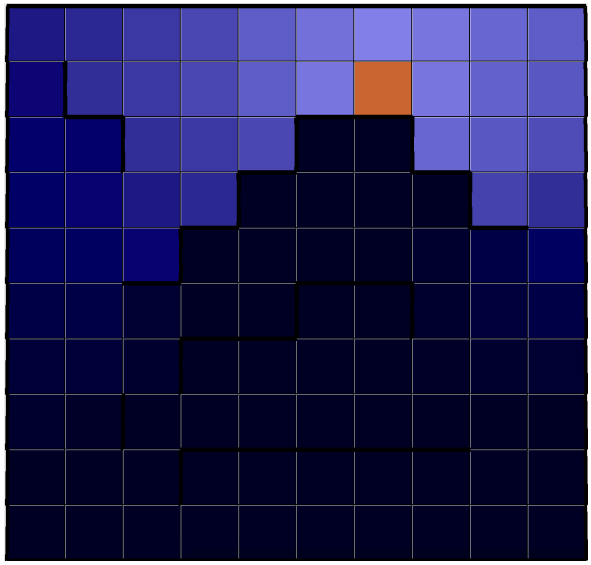
| Transition Probability | 0 | 0.3 | 0.5 | 0.8 |
|---|---|---|---|---|
| Time Taken (ms) | 1577 | 3490 | 8631 | 111657 |
| Steps Taken | 74 | 181 | 428 | 5769 |

However, as I increased the transition probability that it would move to a random state, it took the agent significantly longer to end the game. To really see the difference in changing the probability, I have constructed graphs to see the time and steps taken versus the transition probability.



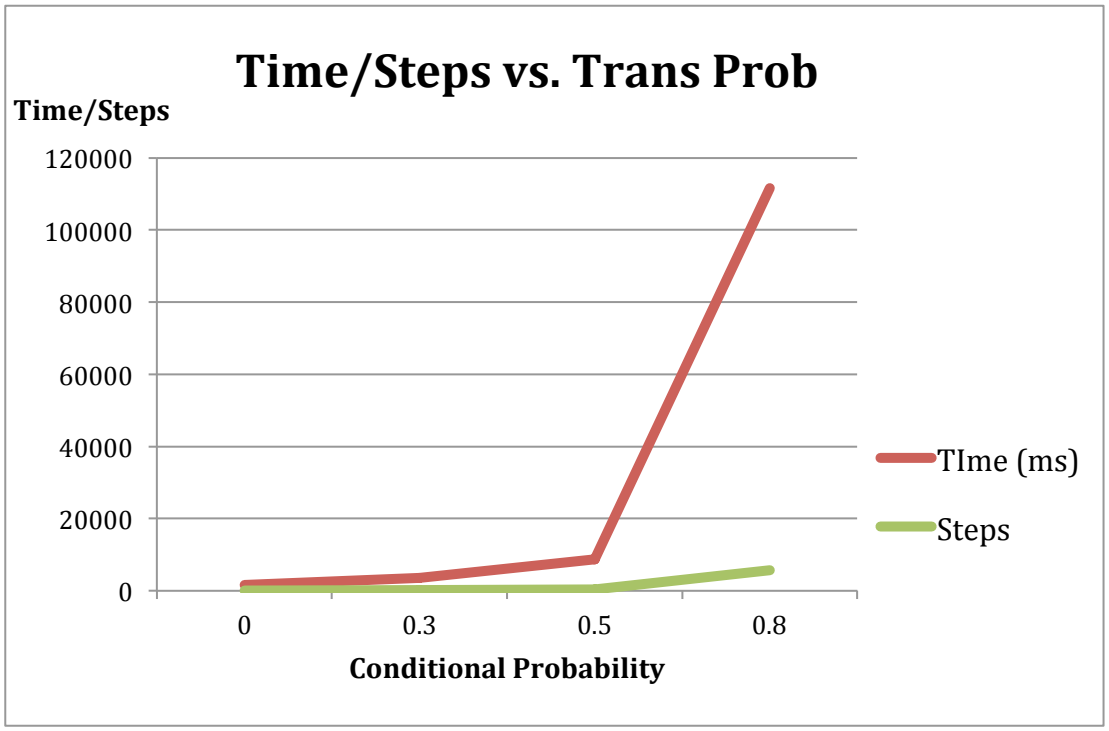

The most interesting aspect is the major jump in time and steps taken when the probability reached 0.8. The gap was not nearly as large when the increase was in the range of 0 – 0.5, but as soon as the transition probability exceeded 0.5 the increase became exponential. You can really tell the agent's patterns by looking at the pictures below. The one on the left is when the transitional probability was 0.3, and the *right* was at 0.8. The shading of each square (state) is based on how much it was visited. Clearly, when the transitional probability is higher the agent explores much more sporadically, as the right photo has a lot more black squares.



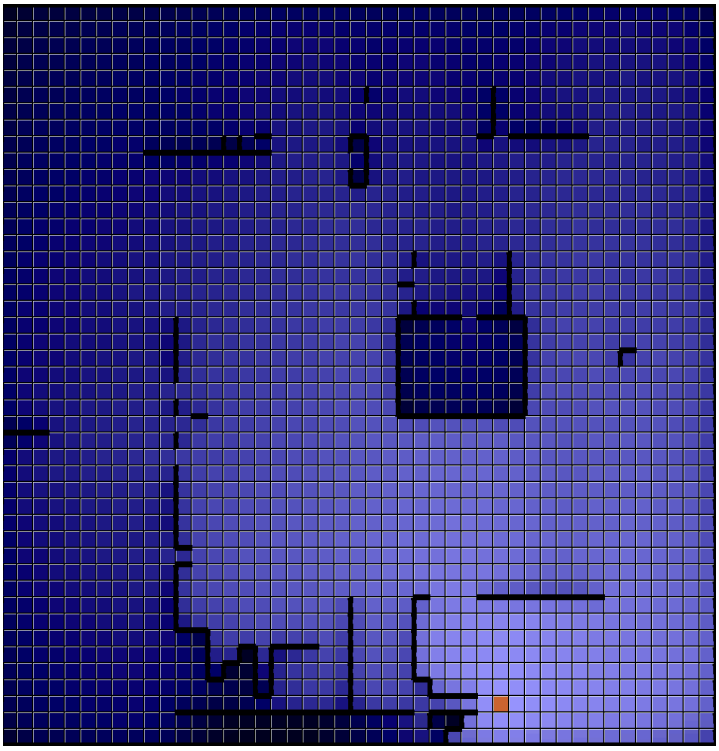


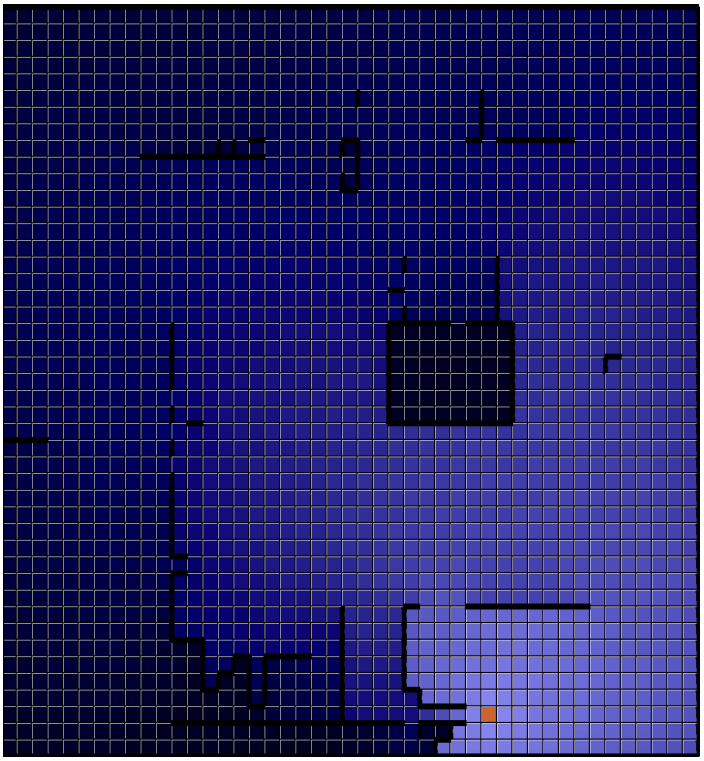
## Value Iteration – Large Maze

Before running value iteration on my large maze, I predicted that increasing the transitional probability would have little effect on the time/steps taken. My reasoning was that it would be much clearer to see in a smaller MDP the effect of random movements, where in a large maze it's harder to quantify a path anyways. Well I was wrong. Increasing the transition probability in my large maze had almost the exact same effect as when using it on my smaller maze. I have graphed the results below. What's interesting is that the time and



step values switched roles. In the small maze the amount of steps increased at an exponential rate, but with the larger maze time increased at the highest rate. Because the maze size was only 45x45, this capped the amount of steps possible for the agent. Below are screenshots of the agent's trail, where darker squares (states)





received more traffic. The one on the left had a transition probability of 0.3, and the one on the right had 0.8. It's easy to see the major difference in exploration that the picture on the right has, when the agent had a higher probability of choosing a random state.

## Value Iteration - Summary

In all, value iteration behaved as expected (to an extent). Both the large and small maze saw a massive increase in steps and time when the transition probability was increased. An interesting aspect is that the mazes flip-flopped the category with the highest increase: steps for the small maze grew exponentially while for the large maze it was time. This can be attributed to the actual state size of the mazes; the large maze had 20 more times the amount of states than the small one. Therefore, the value iteration algorithm took much longer to process it's move as the transition probability increased.

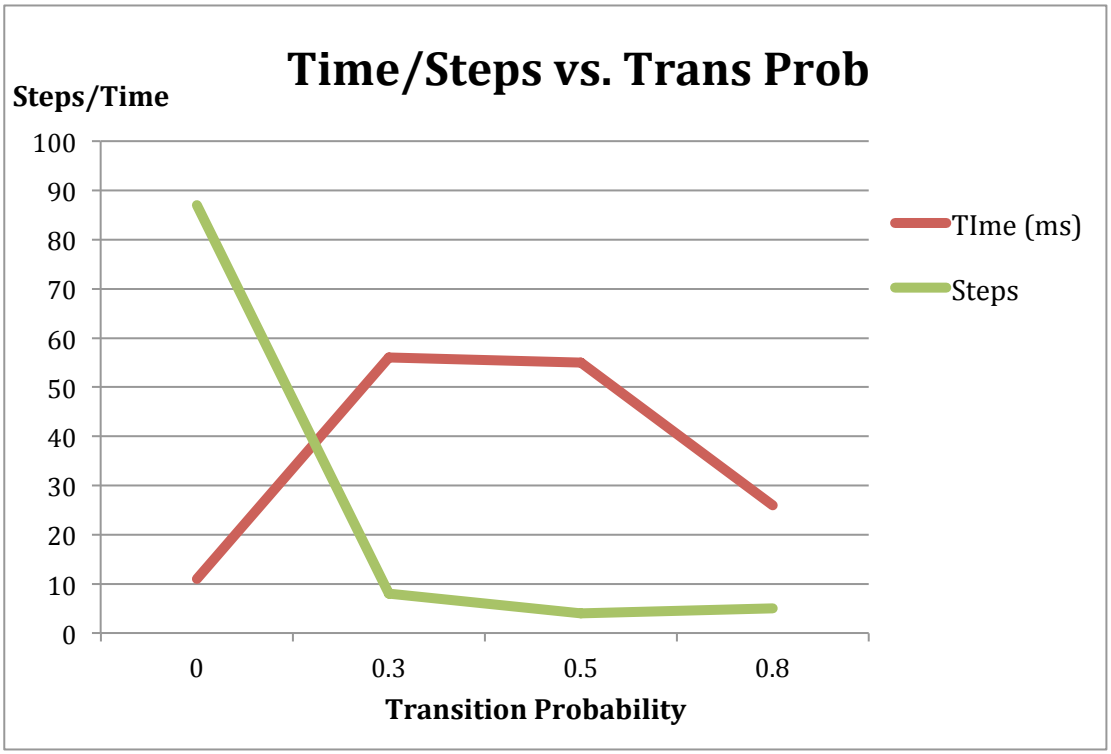
## Policy Iteration

Policy iteration is a faster iterative method where it starts with an approximate optimal policy and calculates the utilities based on that policy. It then updates the

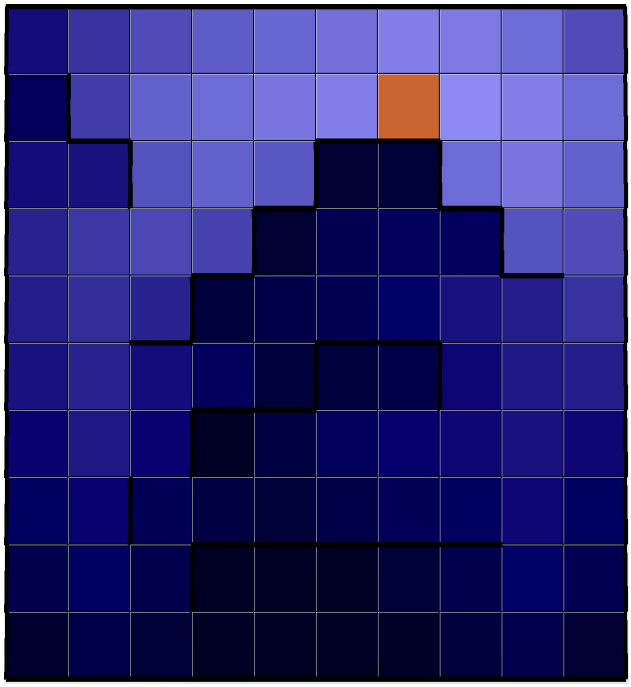
policy based on the policy formula and repeats this process until the policy stops changing. This iterative method has a smaller amount of iterations because policies converge faster than the utility. In all of my policy iteration data the precision threshold was 0.001, iteration limit was 500, and the value limit was 5,000. For policy iteration I again manipulated the transition probability, and it had much more sporadic behavior.

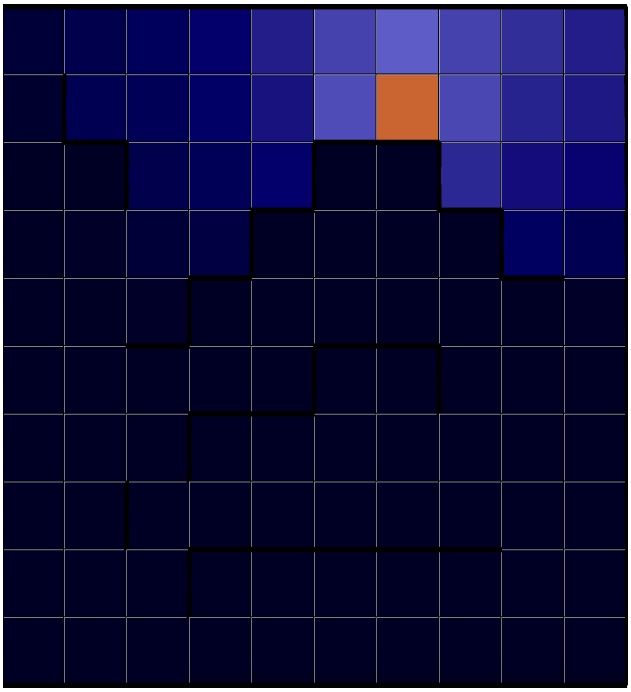
## Policy Iteration – Small Maze

As seen below, the algorithms' steps behaved exactly the opposite of value iteration, which decreased as transition probability increased. Time however was the lowest when there wasn't a transition, as it came down close to the original when tp was set to 0.8.




My only reasoning for time and steps decreasing is that maybe when the transition probability is so high my agent was actually moving into the right state purely based on probability. This would definitely explain why as the transition probability was 0.5 – 0.8, the time decreased greatly. To examine this further, let's take a look at the actual movements of the agent in each state. The one on the right is when tp was 0.8, and the left is 0.3.

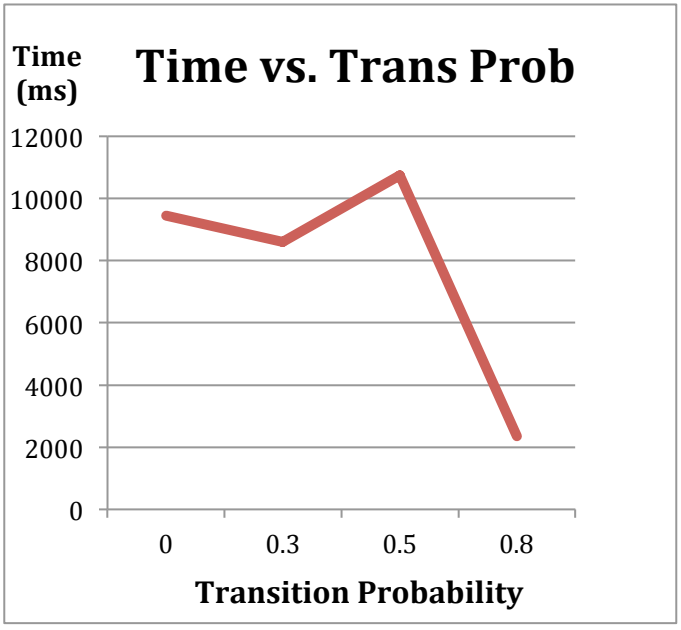


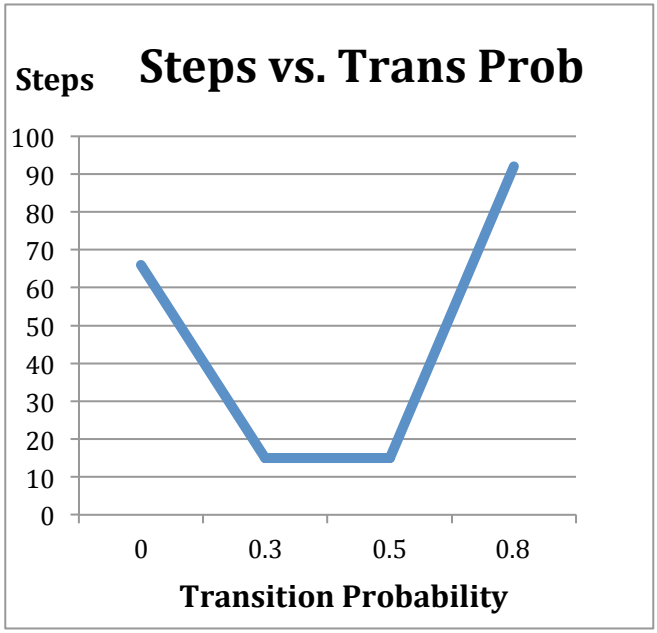


Again it's plain to see that the picture on the left has a much narrower path to the goal as the agent was exploring each state. When the transition probability was 0.8 the agent took a much longer and rigorous understanding of the maze.

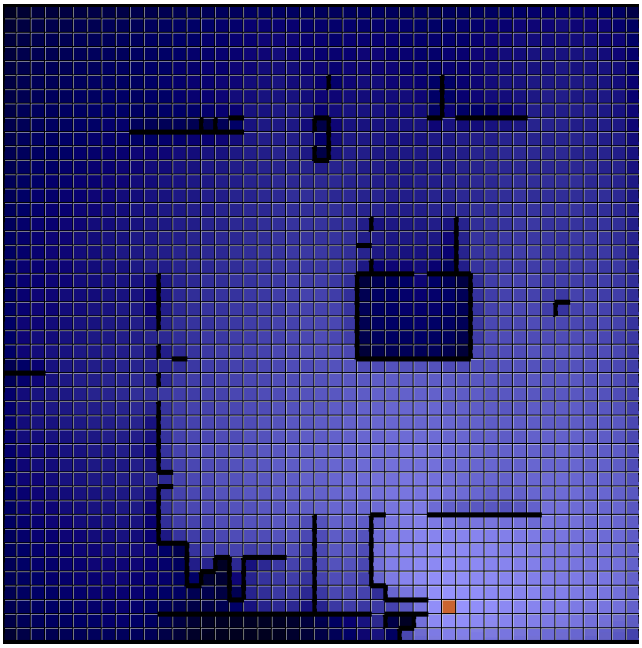
## Policy Iteration – Large Maze

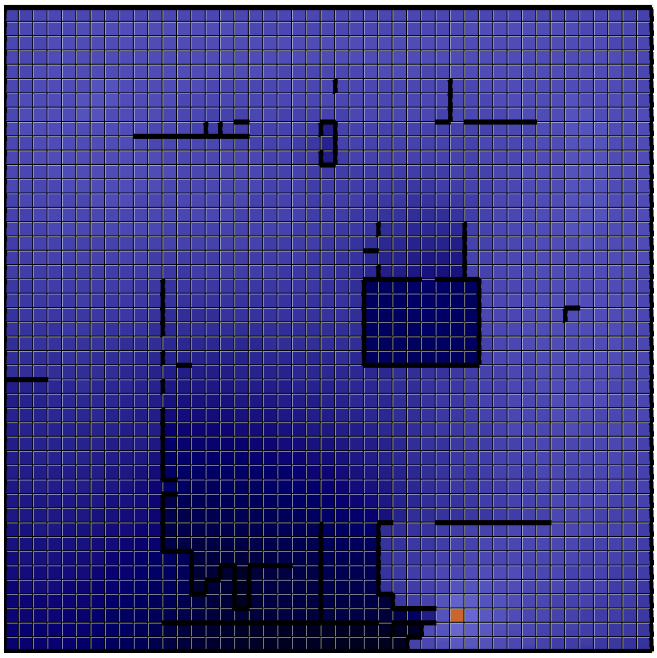
Policy iteration on the large maze actually behaved very differently than the small maze. If you reference the figures on the next page, it seems that time and steps actually reacted inversely to each other.



Because of this, I'm going to say that there is really no correlation between time and steps taken in the large maze, or if any, an inverse correlation. This can be attributed to the size of the maze, as the 45x45 has many more state possibilities, common patterns returned from policy iteration will be much less likely. To further this statement, let's look at the algorithms tendencies in each state, as the figure on the left had a transition probability of 0.3, and the one on the right was 0.8.





What's interesting here is that the transition probability affected how the algorithm explored the maze. When it was 0.3 (left), it seems to have gone up and explored the states near the top left and top right corner. However, as seen by the picture on the right, it explored much closer to the goal state, in the walled in error. This is attributed to the fact that the algorithm wanted to start from the top down, but if you increase randomness then it would explore the bottom first (which is more optimal in this maze!). That's why the time was so much lower when tp was 0.8, the randomness led our algorithm to the goal state.

## Policy Iteration – Summary

Policy iteration had very unpredictable behavior compared to value iteration. As expected, it did outperform value iteration in speed and the number of steps taken, which is what I predicted. However, it is interesting that range of time and steps for the small maze was relatively narrow (difference of less than 50). However, in the large maze this was not the case at all, as the results had to be put on two different graphs because of the range difference. It's also unique to think that in the small maze the time and steps were correlated closely, but for the large range they were inversely correlated. Even though value iteration's results grew as expected (exponentially with transition probability), policy iteration easily had better

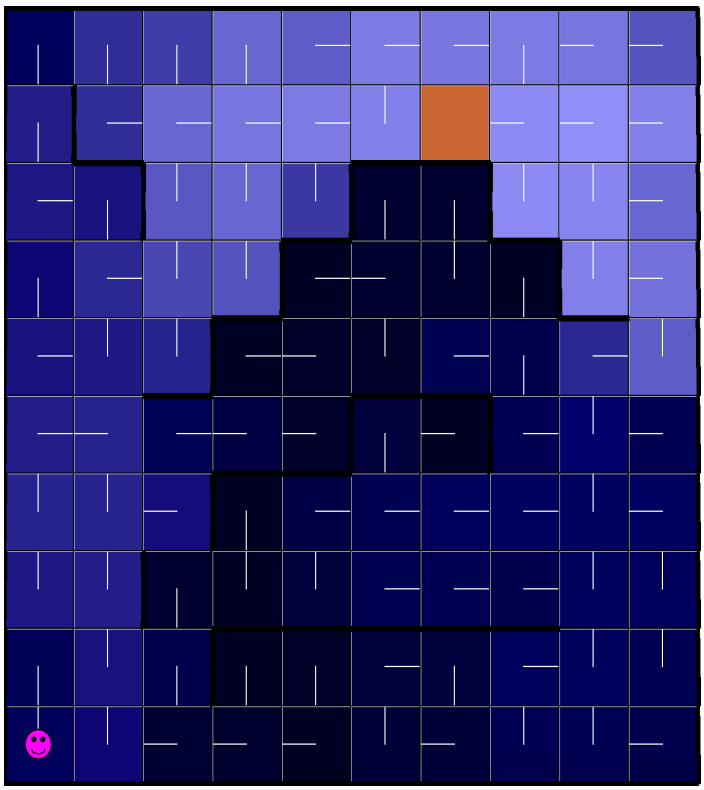
performance in time and steps. Therefore, I would definitely prefer the policy iteration algorithm in a large and small MDP problem.

## Q-Learning

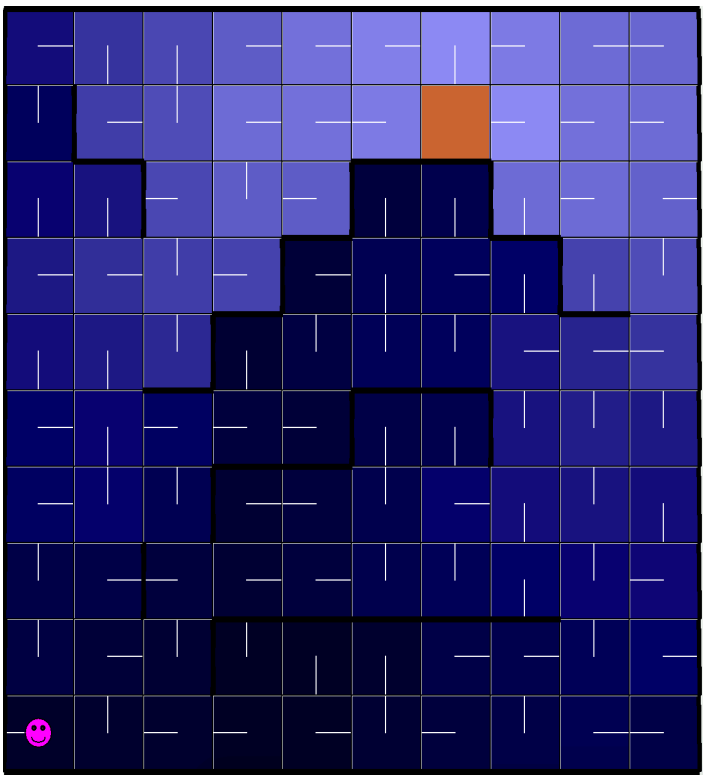
Q-Learning is a reinforcement learning technique that finds the optimal policy for an MDP. However unlike policy and value iteration, it requires no prior knowledge of the model it is trying to solve. It simply learns everything about the MDP as the algorithm iterates. Q-Learning works by learning an action-value function, which gives the expected utility taking an action from a given state. For the large and small maze, I manipulated the epsilon value starting from 0.1 and through 0.8. This value represents the exploitation vs. exploration rate, which is extremely similar to the transition probability. I also ran tests suites where the noise was toggled to 0.8 and 0.3. The learning rate and precision remained static at values 0.001 and 0.70.

## Q-Learning – Small Maze

For the small maze Q-Learning took about 1 second to run, and this was for every variation of variable combinations. Each episode rendered a different policy, and I ran 1000 cycles for each variation.



*TP = 0.3, Epsilon = 0.1*

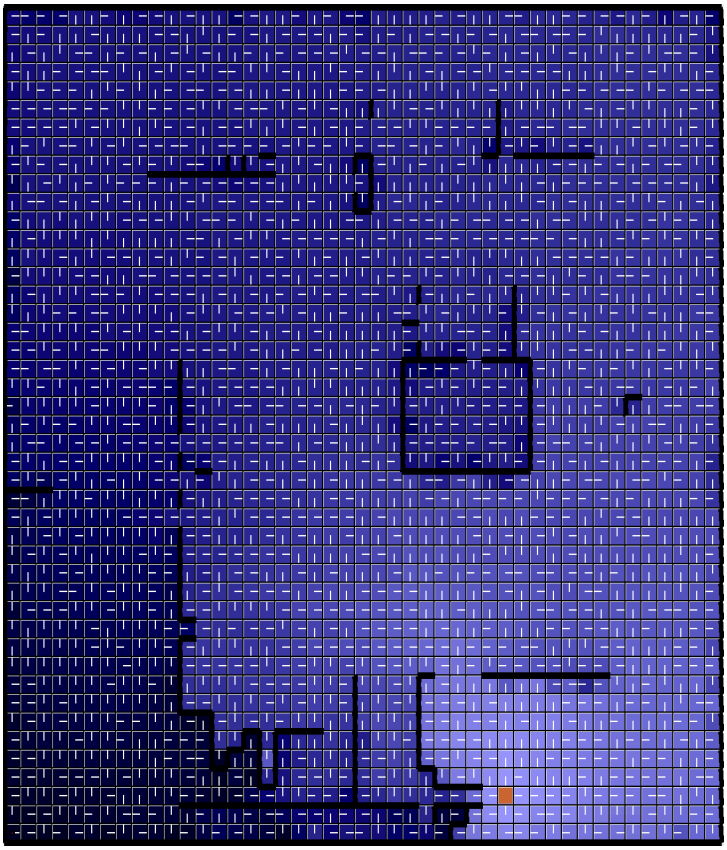


*TP = 0.8, Epsilon = 0.8*

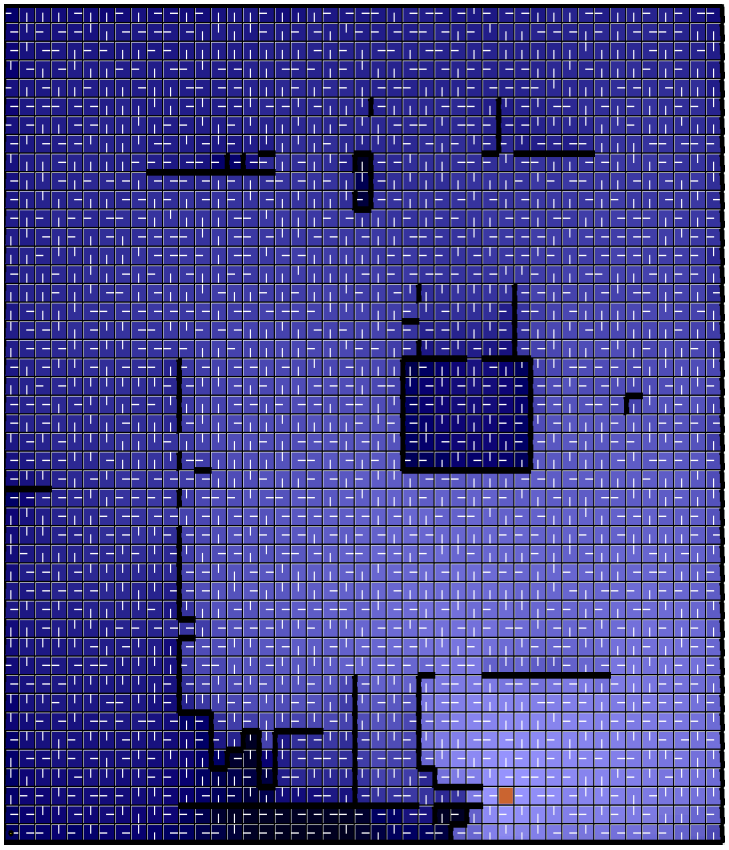
An interesting comparison here is how the manipulation of the epsilon value controls the pattern Q-Learning takes of the maze. Epsilon controls the probability into which Q-Learning arrives in the next desired state, so with a higher epsilon the less control it will have over its next move. The darker shaded states in the figure represent a lower utility, so the lightest state has the highest utility in the maze. That being said, it's clear that the policy on the right had a clear advantage to finding states that had a low utility. Because it didn't have a high probability factoring into it's moves, it was easily able to find the lowest utility states that are blocked from reaching the goal because of walls. Even though QL in the right figure could determine those states had very low utility, it was sub-optimal compared to the left.

## Q-Learning – Large Maze

The large maze actually ran decently efficient, and it took less than 5 seconds for Q-Learning to run 1000 cycles. This is unexpected because Q-Learning has no prior knowledge about my mazes, so I would have expected it to be inefficient. Therefore, I did not graph the time it took to run because there was such a slight difference.



*TP = 0.3, Epsilon = 0.1*



*TP = 0.8, Epsilon = 0.8*

In both of these figures, the epsilon and transition probabilities were manipulated. There wasn't a real difference efficiency wise, but look how much darker the states are in the bottom left corner with the left figure compared to the right. This is

because by manipulating the epsilon value the right figure was not able to detect the utility for every state as accurately as it could have. It's easy to see that states blocked from the goal by a wall should have a very low utility, however if Q-Learning incorporates probability into the next state it visits instead of pure logic, then it is much harder for the algorithm to fully compute the state space.

## Q-Learning – Summary

Q-Learning was definitely an interesting algorithm to analyze. First of all, it definitely performed much more efficiently than I expected. For having no prior knowledge of my mazes, it was able to run 1,000 cycles in under 5 seconds. Now this definitely might have changed if I continually ran 10,000 or even 100,00 cycles, but when I did a trial run at those numbers it didn't increase performance at all. So therefore I kept even run at 1,000 cycles. However, increasing the epsilon value did definitely make a difference in the utility calculations for states, especially around the walls. This can definitely be attributed to the fact that the algorithm desired to not ever enter these states, but because random probability was increased, it sometime had to. This affected it's optimal policy calculation because it had a shadowed view of the actual utilities for some of the states. But, this happens all of the time in real life, as no agent in a realistic world moves exactly as intended every time. There are obstacles that change an agent's perspective, options, and actions. I do think that if I used a large maze, say 100x100, Q-Learning's efficiency might have been much worse.

## Conclusion

After analyzing all three algorithms, value iteration performed as predicted, policy iteration was completely erratic, and Q-Learning was somewhere in between. Value iteration continually increased it's time and steps as the transition probability increased, and this became exponential as $tp > 0.5$. Policy iteration was all over the place – it initially decreased, leveled out around 0.5, and then increased again. This is most likely dude to the fact that when the transition probability is around 0.5, the algorithm unpredictably moves optimally, even when it is up to chance. Therefore having some probabilistic influence in the outcome actually benefits the algorithm, which is something I never thought I would say. However there is a ceiling for this, and when the transition probability reaches 0.8, the steps and time again start to increase. So be careful when setting the initial variables. Finally, Q-Learning behaved normally in terms of the way it explored the maze when the epsilon value was manipulated. Obviously with a lower epsilon value, it has a much better view of the true utility of states, which benefits the algorithm performance wise. However it did behave unexpectedly by running so quickly on my large maze. I expect that if the maze grew (like 100x100) the performance would drastically deteriorate. Overall this project was one of my favorites and really opened my eyes to the different strategies of exploring an MDP problem.