Nilay Mehta

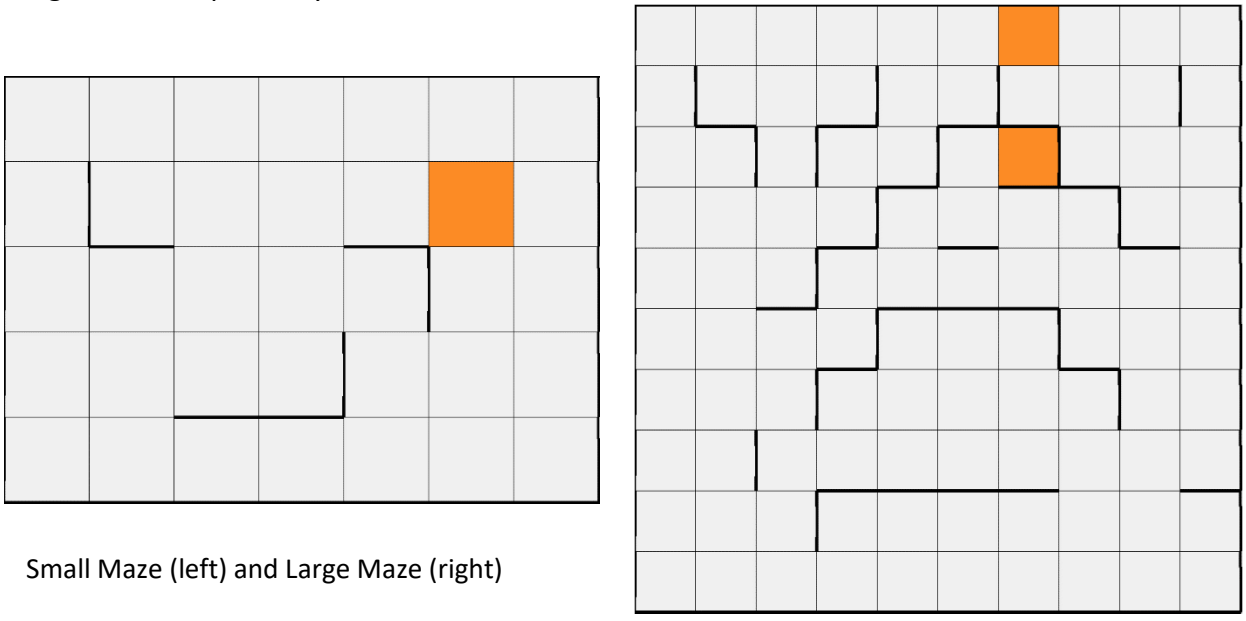
CS 4641 – Machine Learning

4/22/2018

Markov Decision Processes

Introduction: The purpose of this paper is to explore Markov Decision Processes (MDPs) and analyze the performances of various algorithms for path finding policies to reach a goal state destination. Each of the algorithms, value iteration, policy iteration, and Q-Learning, will be discussed for experiments performed over two MDP problems encompassing both a small and large number of states. First, we will examine the two policy making algorithms (value iteration and policy iteration) and then a reinforcement learning algorithm, Q-Learning, will be analyzed alongside the performance of the previous algorithms. To model these techniques, we will be using "RL_sim", a project created by Carnegie Mellon University students Rohit Kelkar and Vivek Mehta. Our two MDP problems are mazes that were built and tested by me using this application and are shown below.

The maze on the left is the smaller maze that I constructed based on the initial sample mazes to run the experiments. It is a 5 x 7 grid with one goal state (indicated by the orange block) and a few adjacent walls that carry a penalty if the agent moves in the direction of the wall. The larger maze is a 10 x 10 grid with two goal states and additional walls, again carrying a penalty if the agent moves into the wall. From here on out, both mazes will be referred to as 'small maze' and 'large maze' respectively.



Small Maze (left) and Large Maze (right)

Before we move on, it's important to understand Markov Decision Process's. MDPs are a mathematical model for decision making that carry some general principles. First, we have agents in an arbitrary state S that can take an action A (for us, this means up, down, left, right) to end up at the subsequent state S' defined by a transition function (defined by T(S, A, S')). Agents do not maintain knowledge of the previous state when assessing the next state to transition to and there are rewards associated with certain states or actions (reward for

reaching goal state vs penalty for hitting a wall). MDPs attempt to find a policy that maximizes the discounted sum of rewards that in turn produces an optimal solution. The experiments we will be performing will traverse these two environments to produce the optimal policy. Unless otherwise specified in the experiments, the parameters of each algorithm are assumed to be their default values. Further information, including how to run the experiments, can be found in the readme file.

Value Iteration:

Value iteration is one of the simplest ways of calculating a policy for an MDP. This algorithm works by estimating the value of a certain state by using the immediate rewards for each state to decide the next best state and action. Ideally, this converges to a solution where the value of a state is the maximum aggregate of the future rewards. This a greedy algorithm and the function for calculating this is called the Value equation as shown below. This is performed for k time steps until our values converge. The default value to check if the algorithm has converged is 0.001.

$$V_{t+1}(s) = \min_{a \in A} \left(PCost + \sum_{s'=S} (P(s'|s,a) \cdot x_{ss'}) \right)$$

where,

PCost = PathCost

$P(s'|s,a)$ = Prob. of transition from s to s' after action a.

$x_{ss'} = V_t(s')$, if transition from s to s' is safe

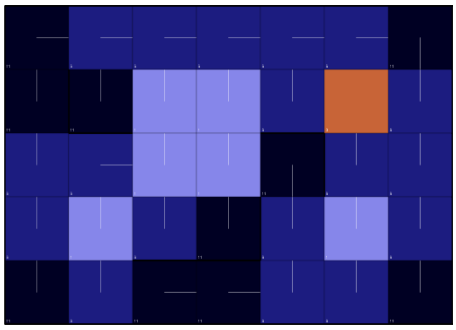
$= Penalty + V_t(s)$, if transition from s to s' is not safe

For both of the environments, we are affected by a noise parameter PJOG which can carry a value between 0 and 1. The noise affects the agent's intended movement because it is a probability that the 'optimal' action will not be performed according to our policy. For example, if we have a PJOG of 0.1, this implies that we have a probability of 0.9 that the action produces the intended state and a probability of 0.1 that another action will be taken and the agent will be pushed into another neighboring state. This also means that the transition function is not completely deterministic. The PJOG will be the parameter we alter and observe the performance for this algorithm.

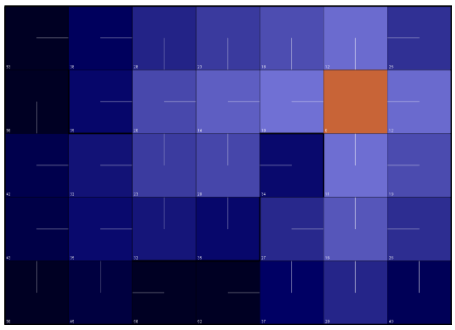
We first ran the algorithm over PJOG values of 0.1, 0.3, and 0.5 and recorded the steps to converge and the time in milliseconds below:

	Value Iteration (steps)	Time(ms)
Small Maze (PJOG 0.1)	21	3
Small Maze (PJOG 0.3)	44	4
Small Maze (PJOG 0.5)	110	7
Large Maze (PJOG 0.1)	40	14
Large Maze (PJOG 0.3)	81	22
Large Maze (PJOG 0.5)	182	98

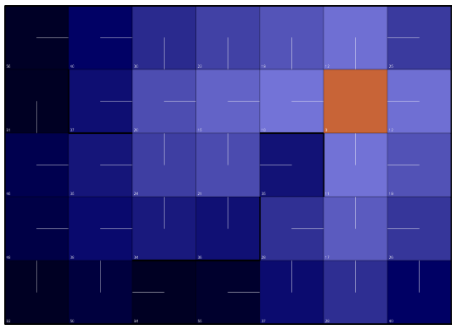
Small Maze: Iteration 1

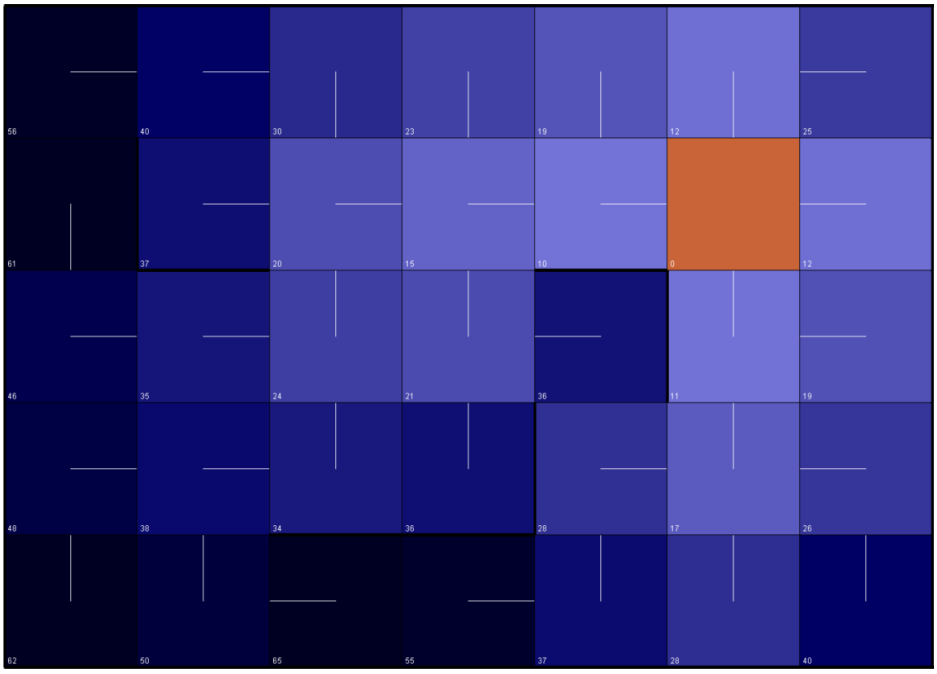


Iteration 15



Iteration 30



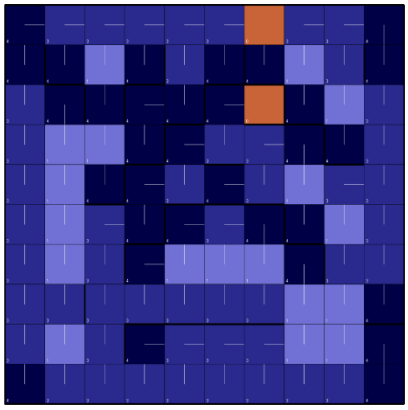
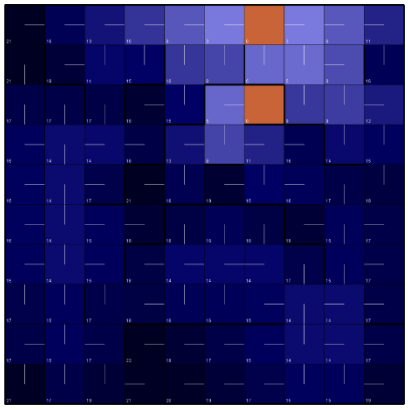
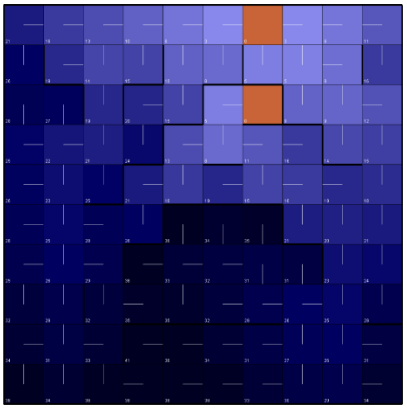


Iteration 44 (Convergence) with PJOG 0.3

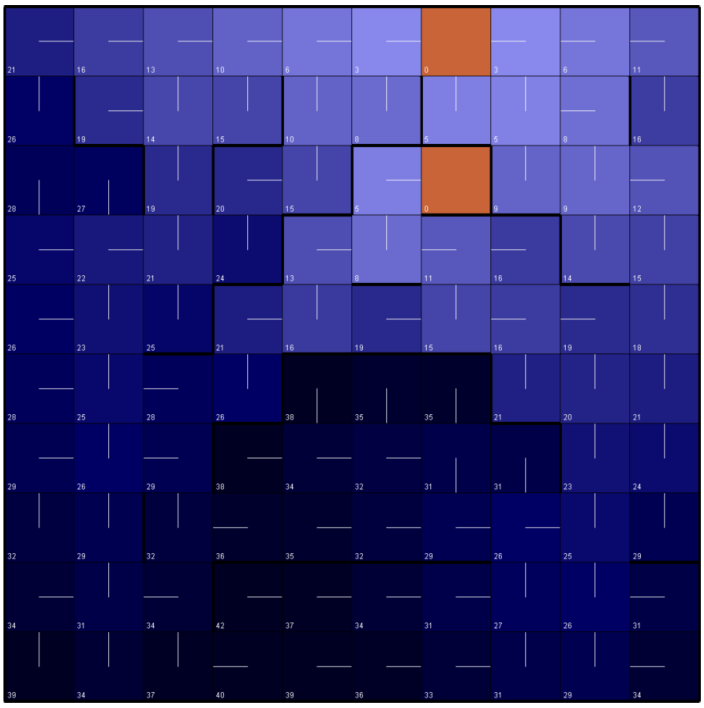
The images above show the iterations until convergence for a PJOG value of 0.3 until iteration 44. The light-colored tiles are the preferred states while the dark colored tiles are the less preferred states. The first iteration is fairly arbitrary because it is the agent's first passthrough and the policy is nowhere near optimal, but as we get to iteration 15 and 30 we can see that the tiles start to normalize and the tiles along the column and row of the goal are clearly preferred tiles.

One interesting observation that I noticed that is more enunciated in the larger maze than in the smaller maze is that the optimal policy tended to take longer paths to the goal states in order to avoid walls. This might be exaggerated in the larger maze because of the higher number of walls and this can be seen in the top right corner of the image where the wall is present. Even for the smaller maze, we can see that the square one down and one left of the goal state strongly prefers to go left rather than right even though right would yield a direct and shorter path to the goal state.

Large Maze: Iteration 1 | Iteration 14 | Iteration 28

Another observation that was interesting is that between the ranges for PJOG increased, we can see that for both of the iterations counts they increase by factors greater than 2. As expected, the larger map has a significantly larger number of states and multiple goals but it seemed to follow the same linear multiple when it came to iteration count changes for PJOGs. From 0.1 to 0.3, we see that iteration counts are about twice and from 0.3 to 0.5, we again multiply the iterations by about 2-2.5 again.



Converged on Iteration 40 (PJOG 0.1)

In terms of the iteration counts relative to the number of states for both mazes, this is expected because as the number of states grow, the number of expected rewards that affect the other states values increases too. As a result, the number of iterations and the time taken to converge to a solution increased from the small to the large maze. It is relevant to point out that the time growth on the second maze was much more drastic from a PJOG of 0.3 to 0.5 and did seemed to follow an exponential growth.

Policy Iteration:

Policy iteration is the second policy making algorithm we will be exploring. The main distinction this algorithm and Value iteration is that with Value iteration, we try and approximate the value of each state using the surrounding states. With Policy iteration, we are directly manipulating the policy and then comparing policies and selects the optimal policy. To accomplish this, we assume the action taken at each state through the current policy rather than computing the states on the board. This can be shown in the equation on the next page.

$$V_{t+1}(s) = Path\_Cost + \sum_{s' \in S}(P(s'|s, \pi(s)) \cdot x_{ss'})$$

where,

$P(s'|s,a)$ = Prob. of transition from s to s' after action a

$x_{ss'} = V_t(s')$, if transtion from s to s' is safe

$= Penalty + V_t(s)$, if transtion from s to s' is not safe

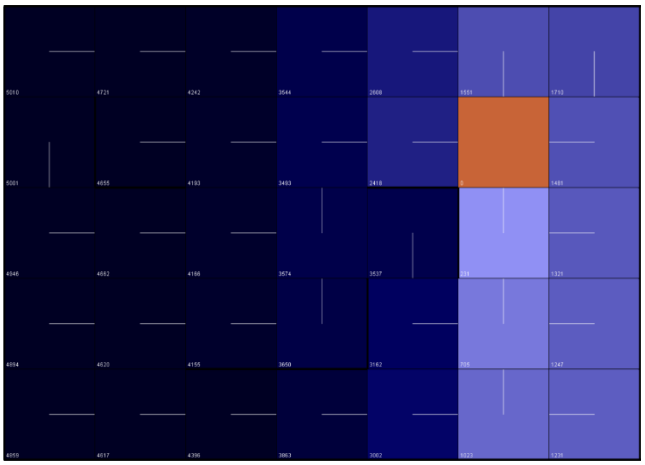
Again, the value that we will be modifying for this algorithm is the PJOG over the values 0.1, 0.3, and 0.5. The number of steps and time taken to reach these solutions is shown below:

	Policy Iteration (steps)	Time(ms)
Small Maze (PJOG 0.1)	5	8
Small Maze (PJOG 0.3)	4	8
Small Maze (PJOG 0.5)	4	9
Large Maze (PJOG 0.1)	8	75
Large Maze (PJOG 0.3)	7	76
Large Maze (PJOG 0.5)	4	60

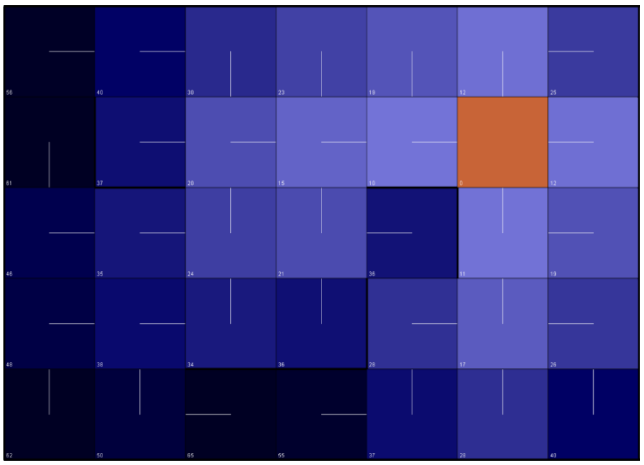
We can see that the solution that the Policy Iteration algorithm produced is the same as the one produced using Value iteration on the small maze. However, across all accounts, policy iteration took significantly less steps and time. These results indicate remarkable improvements when considering the efficiency of each step. There was a noticeable increase in the time it took to converge. However, this is understandable because each step is more involved in the sense that the state values need to converge and then the policy needs to update.

The values for the larger maze is very similar to the solution we obtained to the values of policy iteration and the steps are shown on the following page. We do see some differences between policies, particularly around the middle of the board near the walls, but the changes and differences in tile shades are negligible. The number of steps is much more significant for this

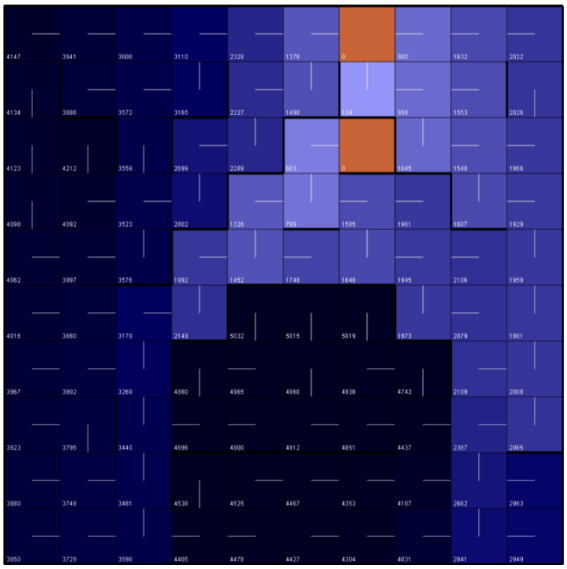
Small Maze: Iteration 1



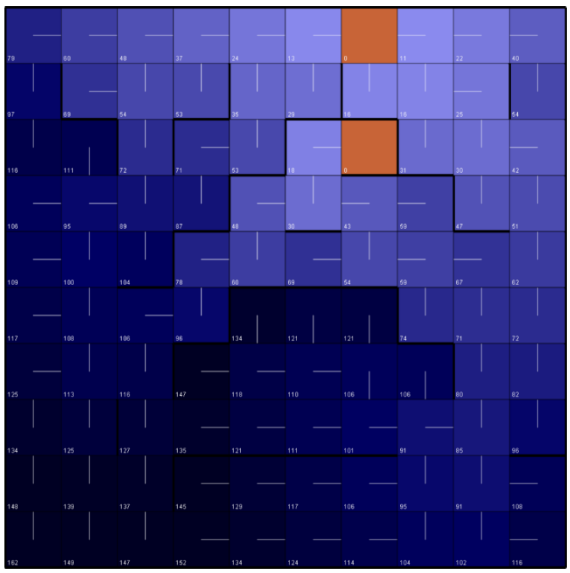
Iteration 4 (Convergence) with PJOG at 0.3

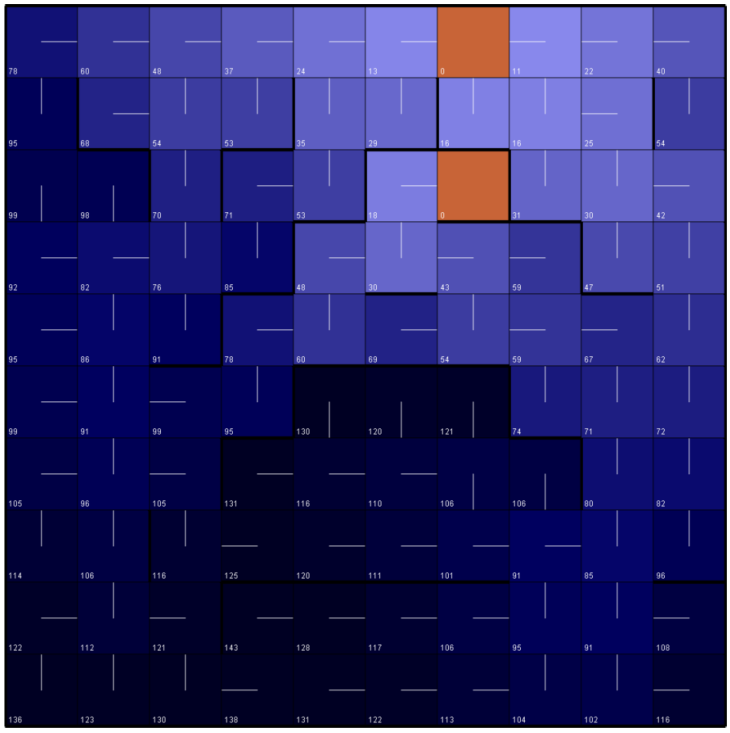


Large Maze: Iteration 1



Iteration 3





Left: Iteration 7 (Convergence) with PJOG at 0.3

maze and actually does result in a noticeable time difference between Value iteration and Policy iteration. This illustrates that one of the strengths of Policy iteration is for problem spaces that have a large number states and can result in both step and time savings.

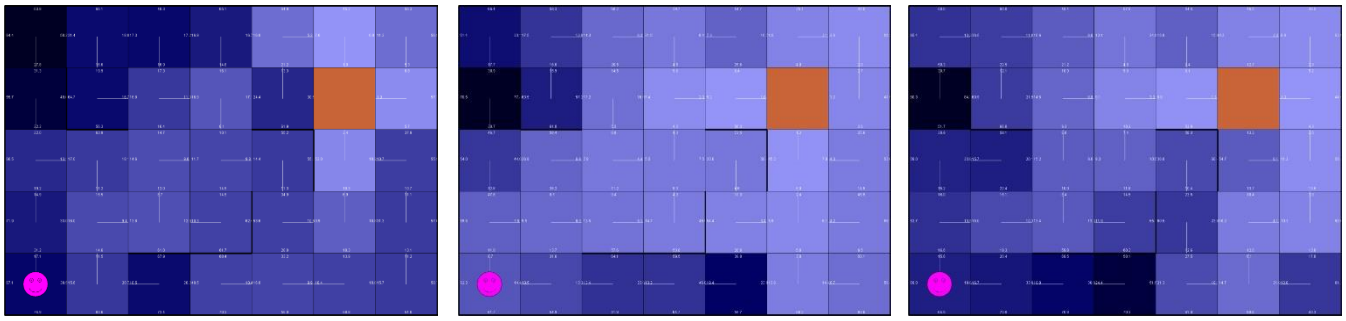
It was also interesting to see that even as we increased the PJOG value, the number of steps to converge improved (as well as the time but that is given because as steps decrease, so does time). This was not expected but logically it makes sense when factoring what was discussed earlier about how each individual step requires that the state values converge before the policy updates. This means that a higher PJOG value theoretically means higher 'exploration' because of more random actions taken until the states converge, in turn making each overall step more efficient and robust causing higher PJOG values to result in lower steps to convergence. These effects are inflated on the larger maze. It is also relevant to point out that this is different from Value iteration where the higher values for PJOG resulted in degraded performance, because of the reasons specified above.

Q-Learning:

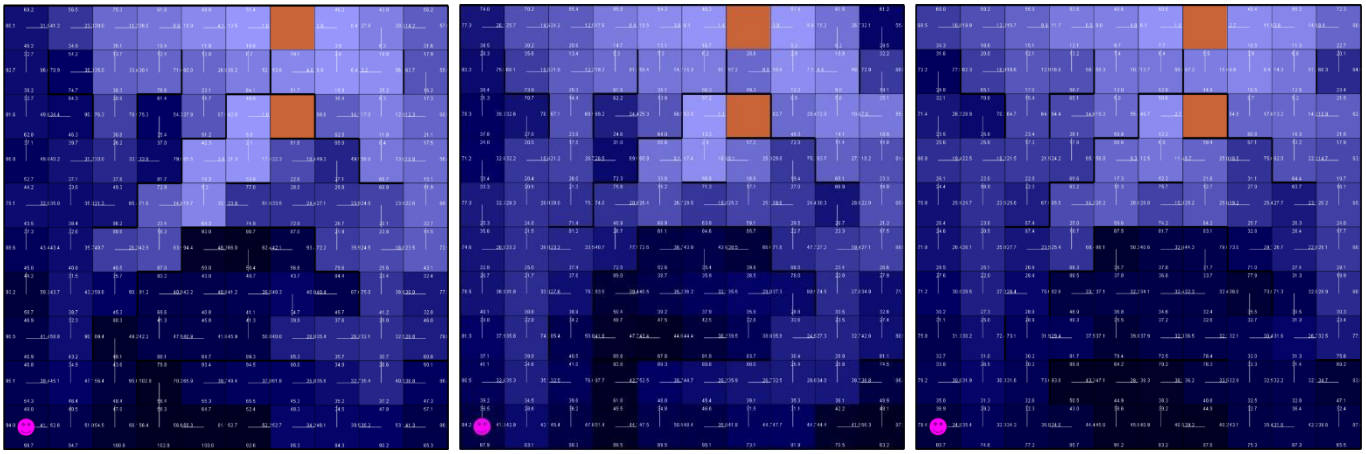
In this last section, we will use a reinforcement learning technique called Q-learning to solve the two MDP problem sets and evaluate its performance relative to our two policy-making algorithms. The inherent advantage of using an algorithm like Q-learning is that we do not need a structured model when approaching our problem. This algorithm makes dictions with very limited knowledge about the environment/world. The two techniques we discussed above require models to define our transition function and reward function that is utilized by our agent. Q-Learning attempts to learn a policy by estimating a value for a state and action ad storing these values in a Q-table. One we evaluate these values, we can choose an appropriate Q-value that either maximizes reward or minimizes penalty depending on your problem state. Because it is much more involved than the other algorithms discussed previously, the tradeoff is higher training times over more iterations. This is because in order to get optimal Q-values, we need to explore each state-action pair multiple times.

For this algorithm, there are a few parameters we can modify for our experiments. Aside from PJOG, particularly epsilon and the number of cycles. Epsilon is a hyperparameter for the epsilon greedy policy that helps with improving exploration. For example, if we have an epsilon value of 0.1, the agent will take the best action with a probability of 0.9, and another valid action with a probability of 0.1. The number of cycles indicates episodes run, where one episode is a traversal of the agent through the environment until it reaches the goal state. During my runs, I found

Small Maze (Epsilon = 0.3) 100 Episodes 500 Episodes 1000 Episodes



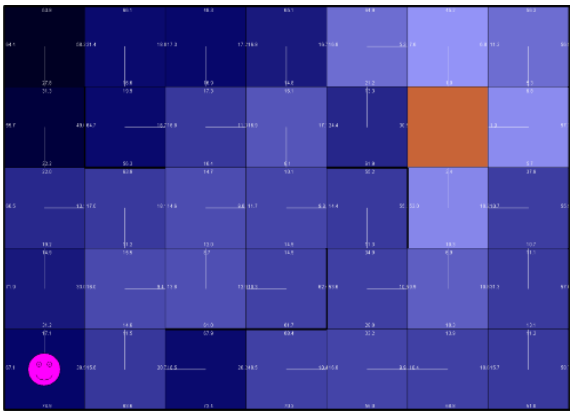
Large Maze (Epsilon = 0.3) 1000 Episodes 5000 Episodes 10000 Episodes



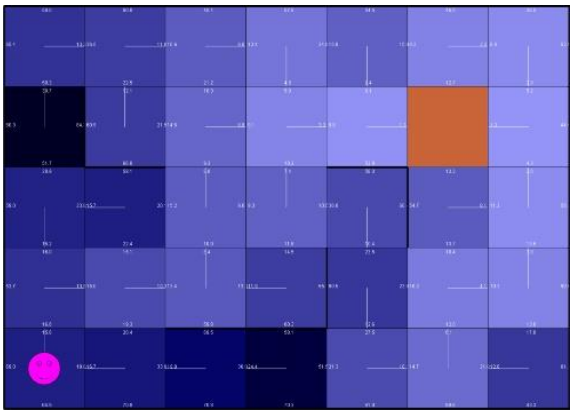
that higher PJOG values severely handicapped the algorithm and required much greater cycle steps to produce a useable policy. Because of this, we will keep the PJOG at 0.1 for the remaining experiments. We also kept the learning rate constant at the default value of 0.7.

For our first set of experiments, we kept our epsilon value constant at 0.3 and tested three values for the episodes. For the smaller maze, we ran the experiment over 100, 500, and 1,000 episodes. It becomes apparent right away that the main drawback of this algorithm is the time and number of iterations to converge to an optimal policy. Comparing the 1000 episode policy with our two previous optimal policies reveals a similarity of 76% between the solutions (nine of the blocks have differing actions). However, the agent still has a clear and straightforward path toward the goal state. For the larger maze we ran the experiment over 1,000, 5,000, and 10,000 episodes. The higher episode count becomes a necessity because of the larger set of states and subsequently an increase in the number of state-action pairs. The Q-learner does surprisingly well over 10,000 episodes with a nearly 91% similarity with the policies produced using the previous algorithms. The comparisons of the policies over the various episode counts illustrates the importance of episode count as a hyperparameter. As a general rule, we can see that as the number of episodes increases, our policy becomes more optimal. This is logical because each episode is a traversal over the environment, and as a result, a higher number of episodes would indicate 'familiarity' with the environment and be more likely to converge at an optimal solution.

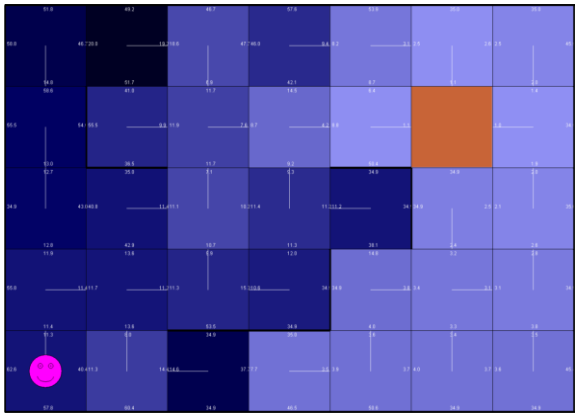
100 Episodes (Epsilon = 0.1)



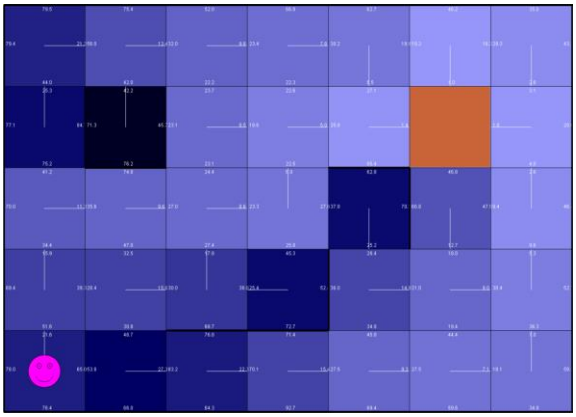
1,000 Episodes (Epsilon = 0.1)



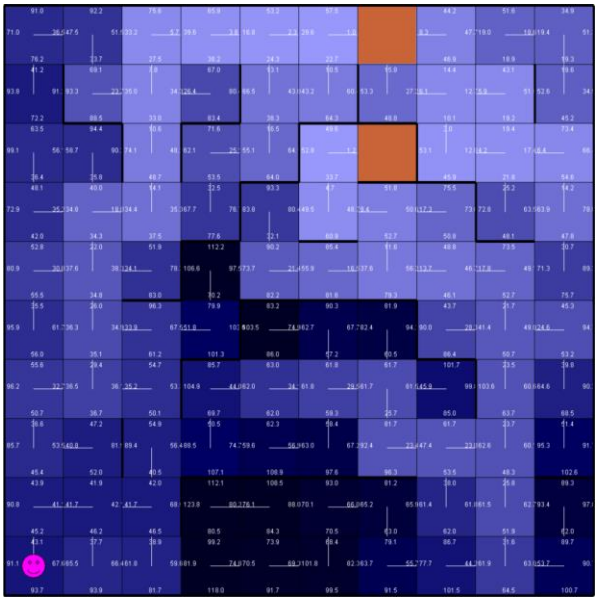
100 Episodes (Epsilon = 0.3)



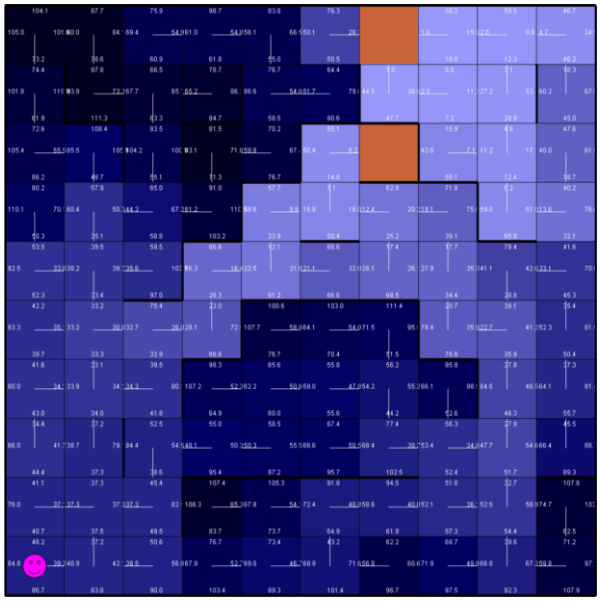
1,000 Episodes (Epsilon = 0.3)



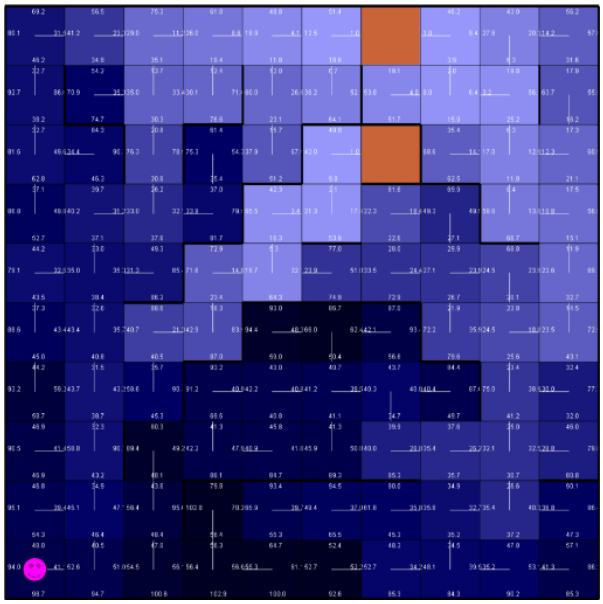
1,000 Episodes (Epsilon = 0.1)



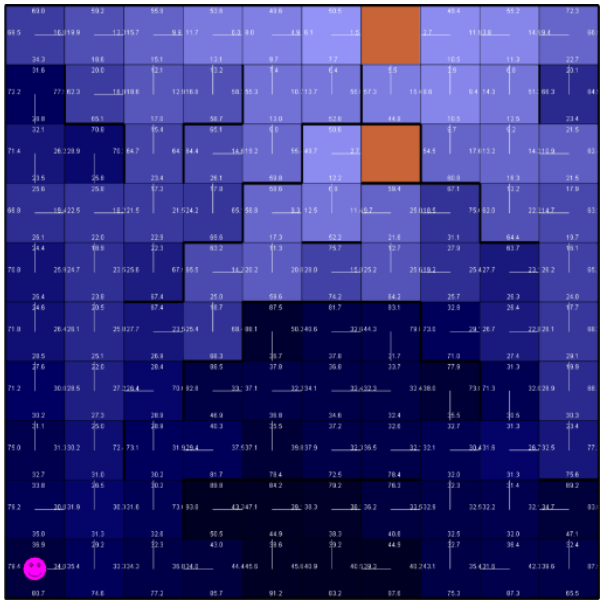
10,000 Episodes (Epsilon = 0.1)



1,000 Episodes (Epsilon = 0.3)



10,000 Episodes (Epsilon = 0.3)



Our last set of experiments looks at the exploration rate and how the epsilon value affects the convergence to our optimal policy over various episode counts. The two values tested for both mazes is epsilon = 0.1 and 0.3. We can see that even with a smaller number of episodes relative to our completion (10% of our total episode count), we get scattered results. Our policies for the smaller maze seems to perform better initially with the higher epsilon value but after completing the full set of steps, our policy for epsilon = 0.1 is closer to our optimal policy from the previous algorithms. The same occurs for our larger maze. Initially, the policies for our larger epsilon value is promising relative to the same policy for the lower epsilon count, yet after completing the full number of episodes, our policy is closer to the optimal solution for an epsilon value of 0.1. It is important to note that increasing the episode count also increases the

compute time in a seemingly non-linear fashion. This is why I believe that using the epsilon value as a parameter to increase exploration rate is a good way to model the optimal policy. Overall, the policies produced by Q-Learning are not optimal or as consistent at the two algorithms discussed earlier. However, the results are impressive when we keep in mind that the Q-learner has a much more limited knowledge of the scope of the environment, yet is still able to consistently produce near optimal solutions in a reasonable number of episodes.