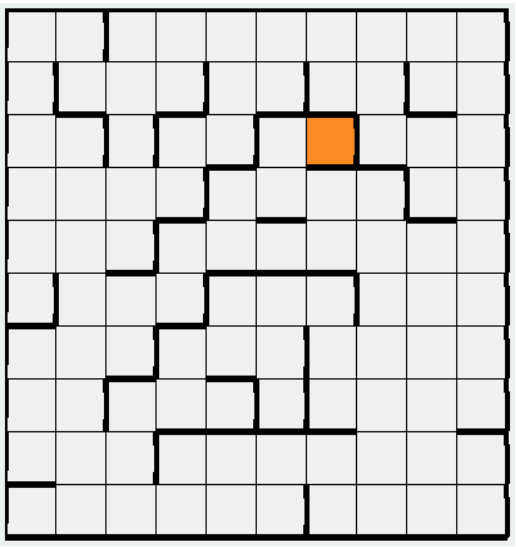
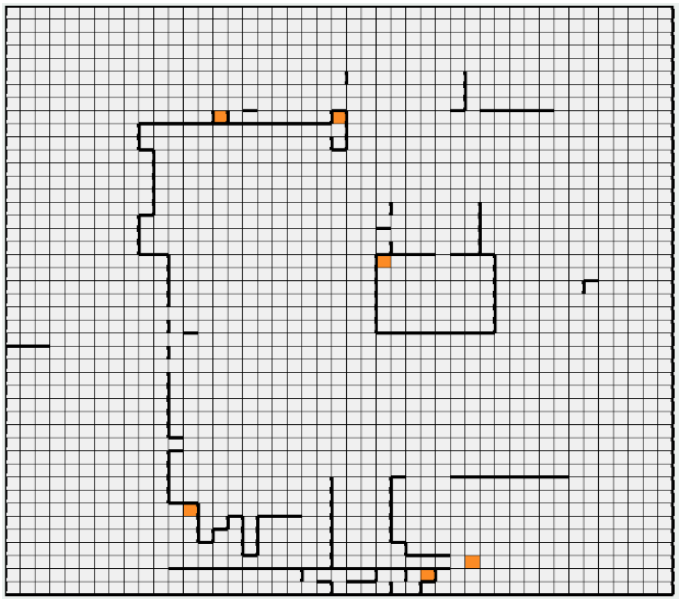
Nathan Dass
CS 4641 – Assignment 4

MDPs and Reinforcement Learning Report

MDPs

For this assignment, I used two different mazes to compare the results of value iteration, policy iteration, and Q-learning. One of the mazes is a small 10x10 maze with 100 total states, 1 goal state, a penalty of 50 points for hitting a wall, and a high number of walls for the size of the maze. The second maze is a large 45x45 maze with 2025 total states, 6 goal states, a penalty of 50 points for hitting the wall, and various configurations of goal states and walls.





The goal that is closest to the bottom of the large maze is hidden in a mini maze with some dead ends and zig-zags. There are also goal states inside a large box, a small box, a corner, in the open, and surrounded by walls on three sides. I placed the goal states in a variety of places to see if this affects the algorithms used.

While both MDPs represent the same world, I find them to be very interesting due to the differences in the way they are arranged and configured. I created these mazes with the hope that I would obtain results that would highlight aspects of the three algorithms. I used the Reinforcement Learning Simulator created by students at Carnegie Mellon University for a visual representation along with the implementation of these algorithms.

Value Iteration

Value iteration always chooses the state that maximizes the expected utility. Since we don't know the true value that a given state has, we add the immediate reward of the state to the expected reward if the agent were to make the optimal move from that point onwards.

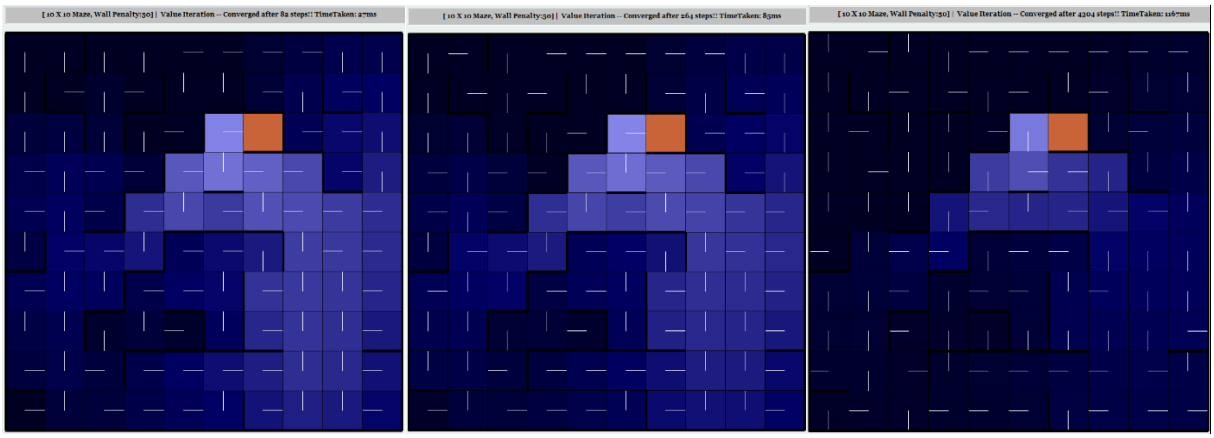
Nathan Dass
CS 4641 – Assignment 4

Small maze			
PJOG	**Precision**	Steps	Time
0.2	**0.0001**	82	27
0.2	**0.001**	76	23
0.2	**0.01**	69	24
0.5	**0.0001**	264	85
0.5	**0.001**	225	74
0.5	**0.01**	186	51
0.8	**0.0001**	4304	1167
0.8	**0.001**	3503	955
0.8	**0.01**	2701	761

Big maze			
PJOG	**Precision**	Steps	Time
0.2	**0.0001**	89	1357
0.2	**0.001**	81	1216
0.2	**0.01**	73	1084
0.5	**0.0001**	363	5371
0.5	**0.001**	312	4636
0.5	**0.01**	262	3883
0.8	**0.0001**	6102	91040
0.8	**0.001**	4857	81063
0.8	**0.01**	3621	62919

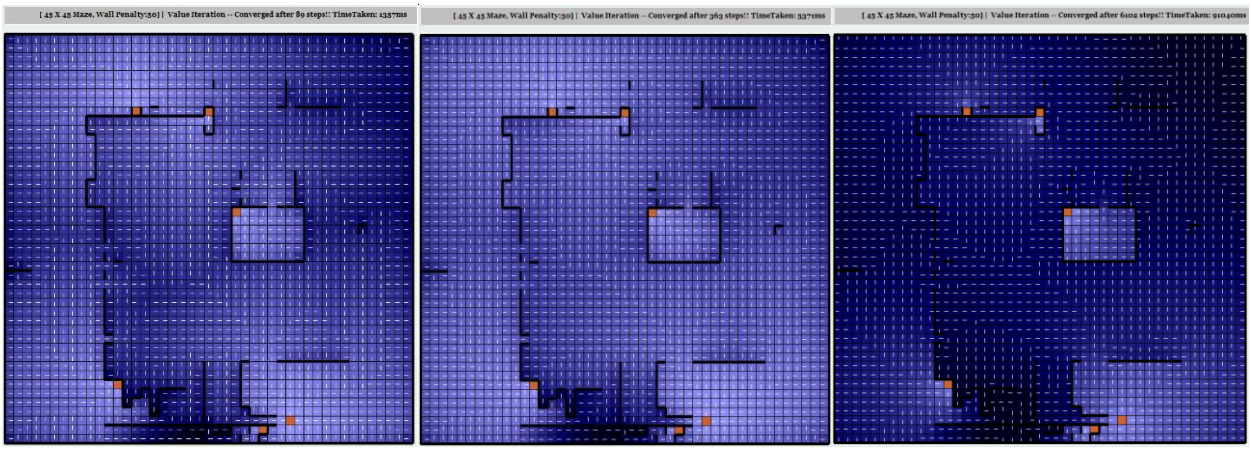
The two tables above show the steps made and times taken to execute value iteration with differing levels of PJOG and Precision. In this case, PJOG models the noise in the environment. A PJOG of 0.2 implies that there is a 20% chance that the agent makes move against its will to any of the neighboring states. Precision is used to check whether the algorithm converged or not.

From the tables, we can see a few trends. First, increasing the PJOG significantly increases the number of steps made and the time taken before the algorithm terminates. Additionally, a change in the precision does change the steps made and the time taken, but the magnitude by which it changes is smaller than if the PJOG changes. Both of these make sense because if the agent is making a lot of moves that are suboptimal, so it would naturally take longer to converge to a solution. However, if the PJOG remains the same and the precision changes, the nature of the agent is not actually changing. I also find it interesting how the number of steps made increases between the small and big maze, but the time taken increases by significantly more between the small and big maze. This is probably due to the fact that each individual step takes significantly longer for the big maze than it does for the long maze since there are more decisions for the agents to make and there are many more incorrect actions that the agent could take.



The pictures above show the results after each instance terminates with each iteration for the small maze. The precision in each of the pictures is 0.0001 and the PJOG changes from 0.2 to

0.5 to 0.8 from left to right. If the value for PJOG was held constant and the precision was changed, the results were identical for a PJOG of 0.2 and 0.5 and only changed a few of the outside stages for a PJOG of 0.8. Interestingly, a change in PJOG from 0.2 to 0.5 barely changes the optimal choice for most of the states, however a PJOG of 0.8 resulted in completely different choices for almost all of the states. This probably happens because since the PJOG is so high, the agent is actually more likely to follow the true optimal path by going in a direction that is not along the true optimal path because of the very high chance that the agent will be forced to go in a direction that it did not actually choose to go in. As such, it would make sense that the optimal path for a very high PJOG is in a direction that does not align with the true optimal path.



The pictures above show the results after each instance terminates with each iteration for the big maze. The precision in each of the pictures is 0.0001 and the PJOG changes from 0.2 to 0.5 to 0.8 from left to right. The results of the big maze follow the same trend as the small maze. With a change in precision, nothing changes when the PJOG is 0.2 or 0.5 and only a few actions change when the PJOG is 0.8. When the PJOG changes from 0.2 to 0.5, only a few actions change, but none of them are close to any of the goal states. The actions near the goal states remain the same. However, for a PJOG of 0.8, almost all (if not all) of the actions are reversed from the results obtained when the PJOG is 0.2 or 0.5 and this is probably due to the same reasoning as described above.

Policy Iteration

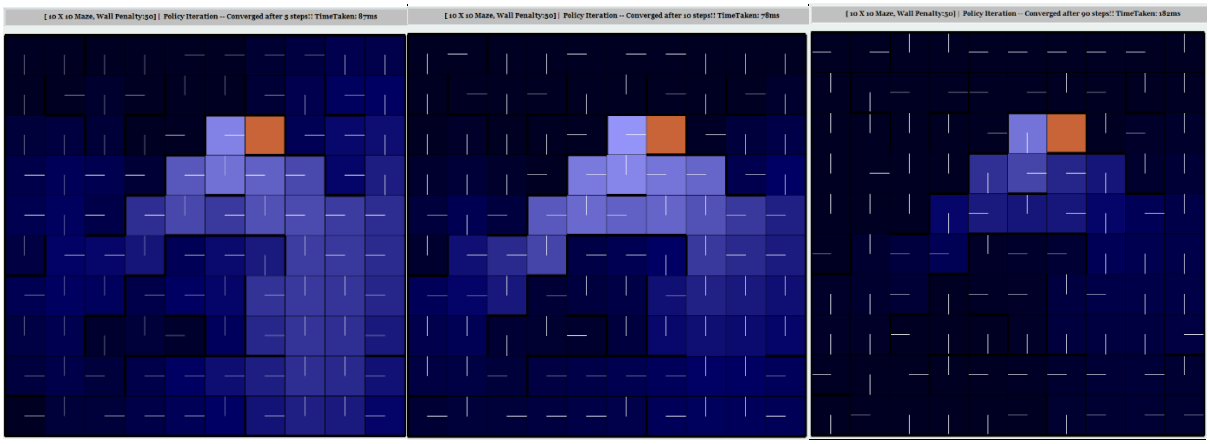
Policy iteration starts with a random policy and continuously updates the policy by updating the utility of each state according to the current policy until there is no change in the policy.

Small Maze			
PJOG	**Precision**	Steps	Time
0.2	**0.0001**	5	87
0.2	**0.001**	6	97
0.2	**0.01**	6	72
0.5	**0.0001**	10	78
0.5	**0.001**	10	43
0.5	**0.01**	10	43
0.8	**0.0001**	90	182
0.8	**0.001**	90	171
0.8	**0.01**	90	164

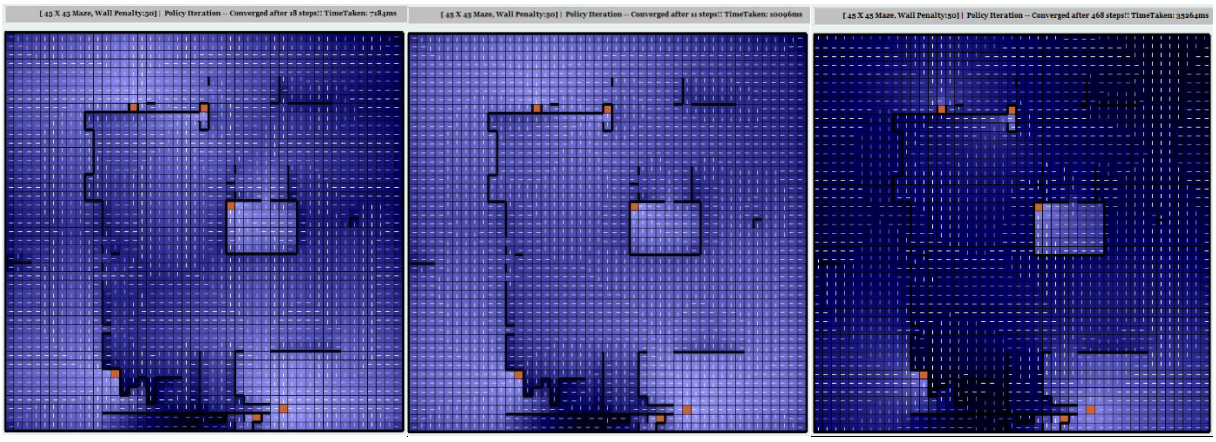
Big Maze			
PJOG	**Precision**	Steps	Time
0.2	**0.0001**	18	7184
0.2	**0.001**	18	6600
0.2	**0.01**	18	6156
0.5	**0.0001**	11	10096
0.5	**0.001**	11	8780
0.5	**0.01**	11	7440
0.8	**0.0001**	468	35264
0.8	**0.001**	1035	41538
0.8	**0.01**	497	27460

The two tables above show the steps made and the time taken to execute policy iteration with different levels of PJOG and Precision. Similar to value iteration, PJOG refers to the chance that the agent makes a move that it does not intend to and precision is used to check if the algorithm has converged.

From the tables, we can see that the precision has little to no effect on the steps for both the small and big maze except for a PJOG of 0.8 where the precision made a pretty large impact on the number of steps. This is probably because the agent almost never actually makes the intended action, so it is hard for the current policy to stick with one optimal actions at each state. Even with respect to time taken a change in precision had minimal effect on all trials of the small maze. This implies that the optimal policy is not really affected by a change in precision unless the agent has a lot of choices to choose from. This is probably because some of the states have very similar utilities and the policy switches back and forth between possible states. It is also interesting how there isn't a very big difference in steps between the small maze and the large maze for a PJOG of 0.2 and 0.5, but the time taken is exponentially larger for the big maze. I also noticed that the first few trials of policy iteration took a lot longer to run and the later trials were much faster. This could be attributed to the fact that the initialize policy is arbitrarily initialized, so it is significantly different from the optimal policy. However, with each iteration the policy becomes closer and closer to the optimal policy and the

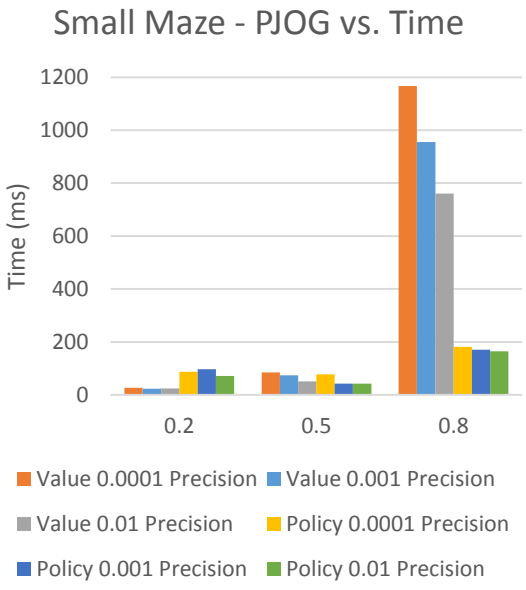


The pictures above show the results after each instance terminates with policy iteration for the small maze. The precision in each of the pictures is 0.0001 and the PJOG changes from 0.2 to 0.5 to 0.8 from left to right. If the value for PJOG was held constant and the precision was changed, the results were identical for all 3 values of PJOG. The results followed a similar trend for the results of the small maze from value iteration where the results were not too different when the PJOG was 0.2 or 0.5, but significantly different for a PJOG of 0.8.



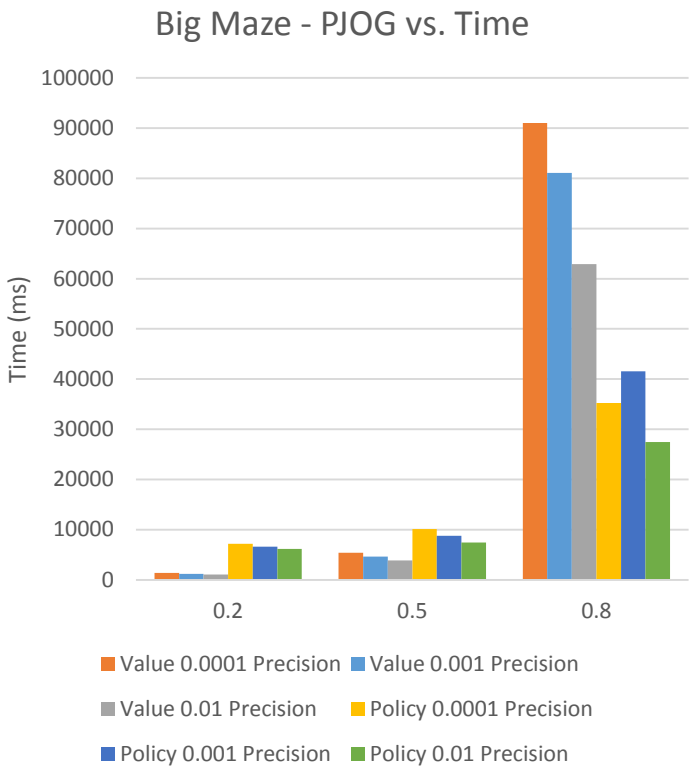
The pictures above show the results after each instance terminates with policy iteration for the big maze. The precision in each of the pictures is 0.0001 and the PJOG changes from 0.2 to 0.5 to 0.8 from left to right. If the value for PJOG was held constant and the precision was changed, the results were identical for all 3 values of PJOG. The results also followed a similar trend for the results of the small maze from policy iteration where the results were not too different when the PJOG was 0.2 or 0.5, but significantly different for a PJOG of 0.8.

Value Iteration vs. Policy Iteration

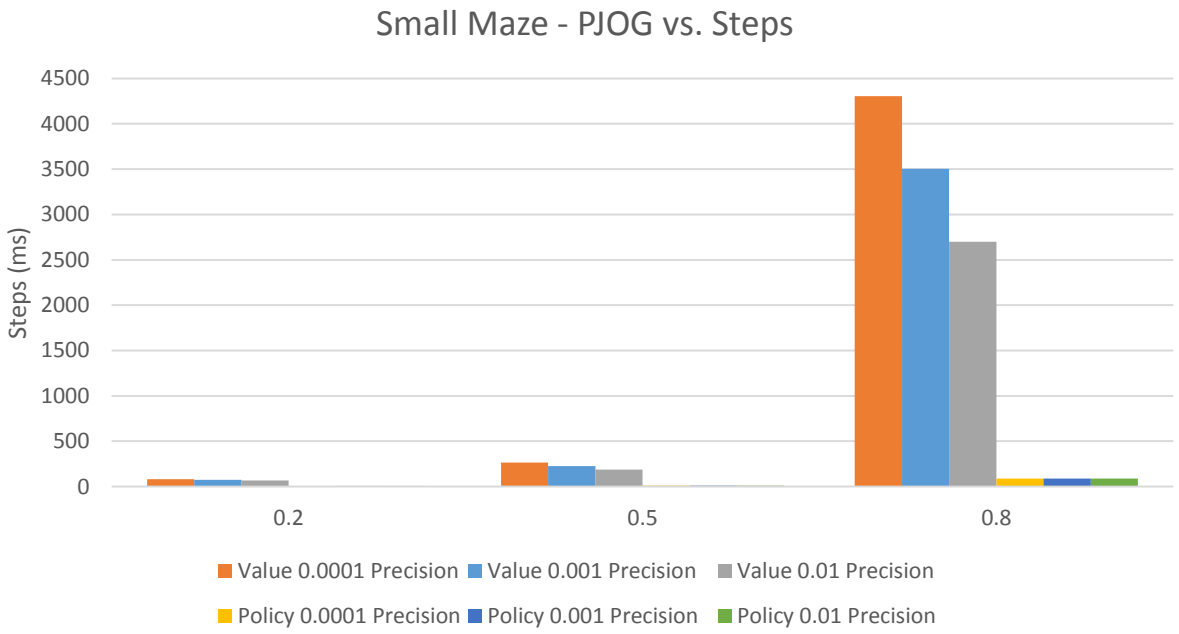


While both value iteration and policy iteration produce similar results, they are some fundamental differences in the algorithms themselves. Value iteration focuses on calculating the value at every state and terminates when the change in each value over each iteration is less than the threshold. On the other iteration, policy iteration focuses more on what the optimal action at each state is and terminates when there is no change in any of the optimal action. As such, value iteration can get stuck in some situations when some of the values are not below the threshold even though the policy isn't changing. In this case, policy iteration would terminate faster than value iteration.

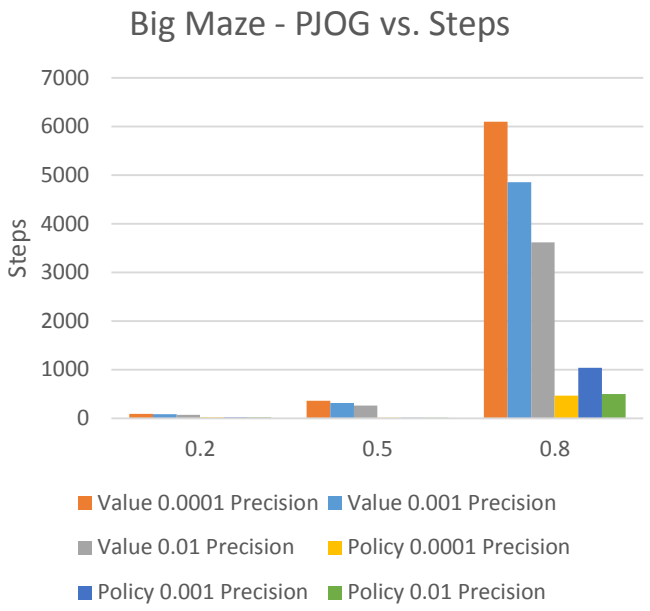
Nathan Dass
CS 4641 – Assignment 4



The graphs for the small maze and the big maze follow mostly the same trends except for when the PJOG is 0.5. When the PJOG is 0.2, policy iteration always takes longer than value iteration. This is probably because the agent usually makes the intended move, so it the utility can converge quickly. When the PJOG is 0.8, value iteration takes longer than policy iteration. This is probably due to the situation mentioned above where the change in the utility for a few states is not below the threshold, but the policy stops changing significantly earlier so policy iteration terminates before value iteration. Additionally, as the precision decreases, the time taken also usually decreases.

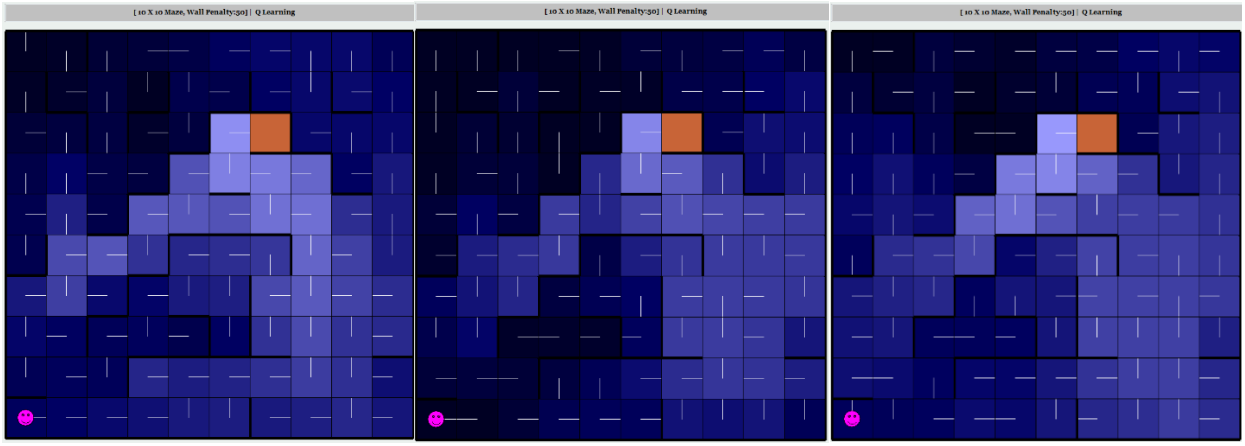


The number of steps also has a nearly identical trends between the algorithms. Value iteration always takes more steps than policy iteration and the number of steps increases dramatically when the PJOG is 0.8. Again, this is probably because of the uncertainty that is associated with whether or not the agent is making the right move. Additionally, it can be seen that the number of steps usually decreases as the precision decreases. This would make sense because we are not looking for as precise of an answer, so we don't have to iterate through more steps once a larger threshold of precision is reached.

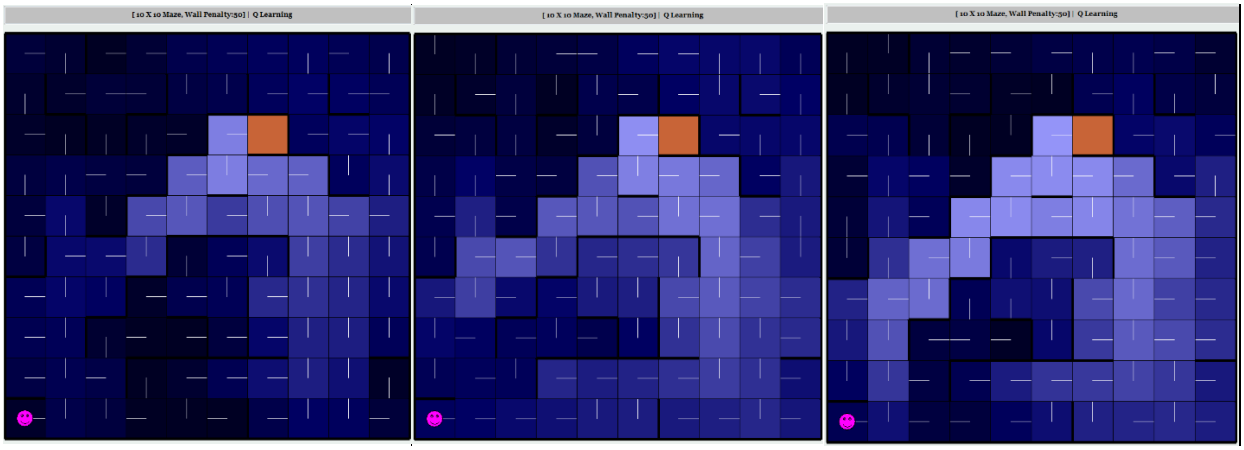


Q-learning

Policy iteration and value iteration work well to find the optimal policy. They assume that the agent knows the transition function and the reward for all states in the environment, but this isn't always true. Q-learning goes about this through exploration and exploitation where exploration refers to taking an action that is not the best action and exploitation refers to using the knowledge that has already been built into the policy and picks the best action. This is similar to how simulated annealing makes bad decisions on purpose to avoid getting stuck in a local maximum. In the CMU Reinforcement Learning Simulator, this value is represented through "Epsilon" where epsilon refers to the probability that the best action is taken and (1-epsilon) is the probability where any of the other actions besides the best action is taken. Another important parameter is "Learning rate" which is a value between 0 and 1 and represents how much the new policy changes the existing policy. For these trials, I used a PJOG of 0.2, a precision of 0.001, enabled the decaying learning rate option, and ran the algorithm for 1000 episodes where one episode represents transitioning from the start state to another state repeatedly until the goal state is reached. With decaying learning rate enabled, each successive policy has a smaller influence on the overall policy. Therefore, the initial policies have the largest impact.

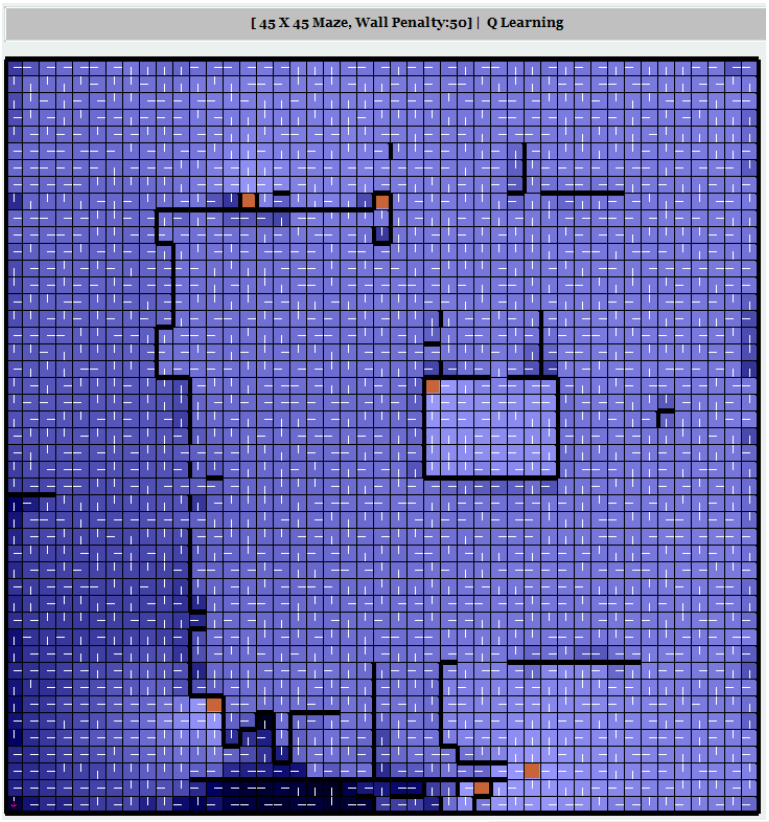


The pictures above show the results after each instance terminates with Q-learning for the small maze. The learning rate in each of the pictures is 0.5 and the epsilon changes from 0.2 to 0.5 to 0.8 from left to right. With an epsilon of 0.2, there are several instances where the suggested action is actually suboptimal and does not lead directly to the goal state. However, most of the suboptimal actions are on the edges of the maze. This would suggest that a lot of exploitation and not that much exploration is a good approach for this maze. With an epsilon of 0.5, there are two suboptimal actions that are near the goal state and more suboptimal choices that are relatively close to the goal state, but not on the edges of the maze. This would suggest that an equal amount of exploration and exploitation is not a good approach for this maze. With an epsilon of 0.8, there are no suboptimal actions that are close to the goal state, but there are many more suboptimal actions on the edges of the maze and often times, the action is to run into a wall. Given these observations, I would say that an epsilon of 0.2 is the optimal epsilon between these three epsilons.



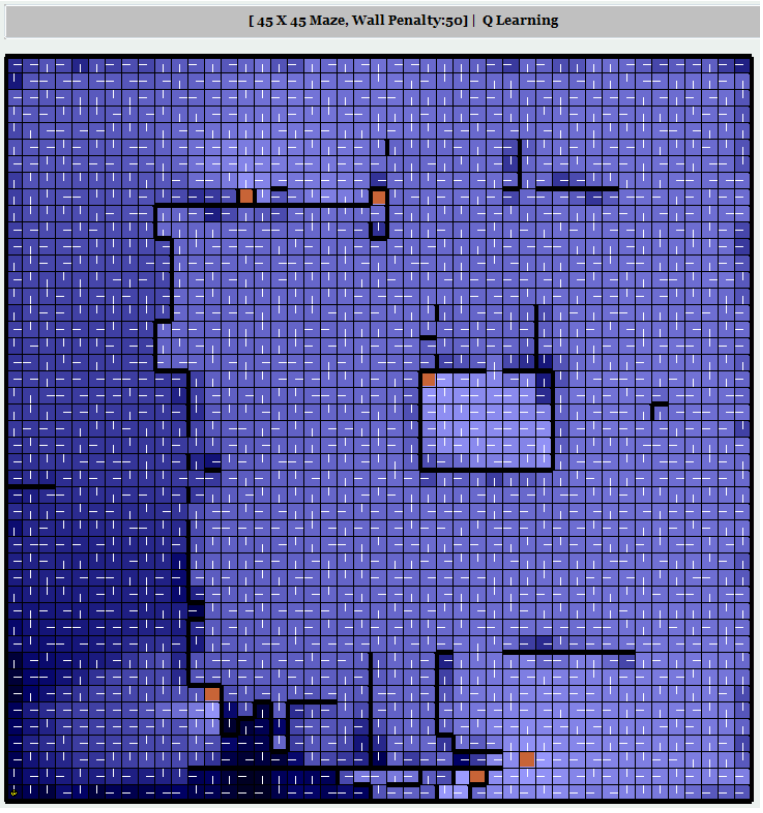
The pictures above show the results after each instance terminates with Q-learning for the small maze. The epsilon in each of the pictures is 0.2 and the learning rate changes from 0.2 to 0.5 to 0.8 from left to right. In general, it appears that as the learning rate increases, the number of suboptimal actions decreases. This would suggest that an epsilon of 0.2 and a learning rate of 0.8 would be the best combination. I think that this makes sense because there are not many states, so there is not much of a need for a lot of exploration. However, it is also easy to make an

incorrect action as a part of the policy, so making sure that later episodes still have a reasonable influence on the overall policy is important for a successful policy.



Interestingly, with Q-learning, only one combination of the learning rate and the epsilon resulted in all of the immediate neighbors of the goal states to be correct. This combination was an epsilon of 0.5 and a learning rate of 0.5 (pictured to the left). None of the eight combinations resulted in the immediate neighbors being correct, regardless of the epsilon or the learning rate. Additionally, the nearby neighbors of the goal states were usually suboptimal as well. This implies that Q-learning is pretty bad for a large maze. Even if there was only one path to the goal state, sometimes the algorithm would suggest moving directly away from the goal state.

The worst combination of epsilon and learning rate for the big maze ended up being 0.2 for the epsilon and 0.5 for the learning rate (pictured to the right). In this case, there are four immediate neighbors that do not suggest to move towards the goal state. I also find it very interesting that for the very bottom goal state with the path that does not really have many options to choose from, Q-learning suggests to move along the exact opposite path that the moves should be going.



In general, Q-learning was not effective for the big maze, but an equal balance of exploration and explolitation seemed to work the best.

Conclusion

From this assignment, I learned about different reinforcement learning algorithms, specifically value iteration, policy iteration, and Q-learning. While value iteration and policy iteration converged to similar solutions for both the big and the small mazes, Q-learning performs pretty poorly on the big maze. Value iteration is typically fast at computing the optimal policy, but also needs a large number of iterations to converge and requires full knowledge of the state space and the model. While value iteration is faster than policy iteration, full knowledge of the state space and the model is still required. On the other hand, Q-learning requires no prior knowledge of the state space or the model, but comes across the exploration vs. exploitation dilemma. Exploration leads to the agent taking less optimal actions, but can result in finding a more optimal solution in the future. Exploitation leads to the agent utilizing actions that are already known to produce good results, but can converge to a local optimum. Finidng that balance of exploration and exploitation is key to effectively finding solutions to MDPs when using Q-learning.

Small maze								
Value Iteration					Policy Iteration			
PJOG	Precision	Steps	Time		PJOG	Precision	Steps	Time
0.2	0.0001	82	27		0.2	0.0001	5	87
0.2	0.001	76	23		0.2	0.001	6	97
0.2	0.01	69	24		0.2	0.01	6	72
0.5	0.0001	264	85		0.5	0.0001	10	78
0.5	0.001	225	74		0.5	0.001	10	43
0.5	0.01	186	51		0.5	0.01	10	43
0.8	0.0001	4304	1167		0.8	0.0001	90	182
0.8	0.001	3503	955		0.8	0.001	90	171
0.8	0.01	2701	761		0.8	0.01	90	164

Big maze								
Value Iteration					Policy Iteration			
PJOG	Precision	Steps	Time		PJOG	Precision	Steps	Time
0.2	0.0001	89	1357		0.2	0.0001	18	7184
0.2	0.001	81	1216		0.2	0.001	18	6600
0.2	0.01	73	1084		0.2	0.01	18	6156
0.5	0.0001	363	5371		0.5	0.0001	11	10096
0.5	0.001	312	4636		0.5	0.001	11	8780
0.5	0.01	262	3883		0.5	0.01	11	7440
0.8	0.0001	6102	91040		0.8	0.0001	468	35264
0.8	0.001	4857	81063		0.8	0.001	1035	41538
0.8	0.01	3621	62919		0.8	0.01	497	27460