Anmol Chhabria – achhabria7
Assignment #4
CS 4641

Markov Decision Processes and Reinforcement Learning Assignment

Markov Decision Processes and Reinforcement Learning

Markov decision processes (MDP) are state-transition systems with discrete times. A MDP is described by a set of states (*S*), a set of actions (*A*), transition probability function *T(s, a, s')*, real-valued reward corresponding to an action *a* from state *s* to state *s' R(s, a, s')*, and a discount factor γ. The transition probability *P(s, a, s')* corresponds to the probability that the agent will next be in state *s'* after carrying out action *a* in state *s*. The transition probabilities for an MDP follows the Markovian property, only the current state *s* and current action *a* matter for the next state *s'*. The next state *s'* is independent of all the previous states and actions. The discount factor represents the difference in importance between present and future rewards. Given S, A, *T(s, a, s')*, *P(s, a, s')*, and γ, the MDP computes a policy π for the problem. The policy maps every state *s* in S to an action *a* in *A*. Because of the Markovian property, the action will only depend on the current state *s*, not any previous states. The objective is to compute the optimal policy π^*, which will result in the optimal rewards. In this analysis, MDPs will be solved using Value Iteration, Policy Iteration, and Q-Learning. Q-Learning is a type of reinforcement learning algorithm. Reinforcement learning algorithms solve problems that can be described like MDPs, but they do not require knowledge about the environment to solve the MDP. Reinforcement learning agents learn the behavior based on the feedback provided by the environment through an exploration and exploitation tradeoff.

MDP Problem Selection

The problems selected are two mazes. Each state is represented by a cell in the maze. The start state for both mazes is the bottom left cell. The possible actions for an agent are *up*, *down*, *left*, and *right*. An orange square represents a goal, which is a terminal state. The obstacles are represented by walls. Every maze is surrounded by walls. If the agent carries out an action that results in hitting a wall, then the agent stays in the same state and is penalized. Every step taken has a penalty cost, or negative reward, of -1. The agent receives a positive reward for reaching the goal.

The stochasticity in the problem is modelled with the parameter *PJOG*. The stochasticity can take a value of between 0 and 1. Every state s has N successors. If in a particular state *s* the agent performs an action *a,* the agent has a probability of (*1-PJOG*) of ending up in the valid successor of the state *s,* while the agent has a probability of *PJOG/(N-1)* of ending up in any one of the *N-1* successors of the state.

Small Maze

The small maze is a 10x6 maze with 60 possible states. The small maze has two possible goals. One of the goal states is closer to the agent's start state as well as the wall obstacles. On the other hand, the other goal is farther from any obstacles, but it is farther from the start state. As the stochasticity increases, we could guess that the agent will choose the goal that is farther from the obstacles.

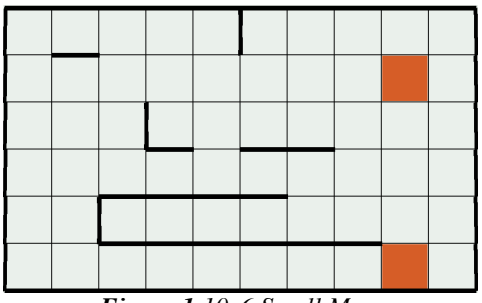

Figure 1 *10x6 Small Maze*

Large Maze

The large maze is a 45x45 maze with 2,025 possible states. The larger maze has more complex wall obstacles than the smaller maze. In addition, there is only 1 goal state, which is surrounded by obstacles. It is expected that the agent will take the safer route to the goal as its actions become more stochastic.

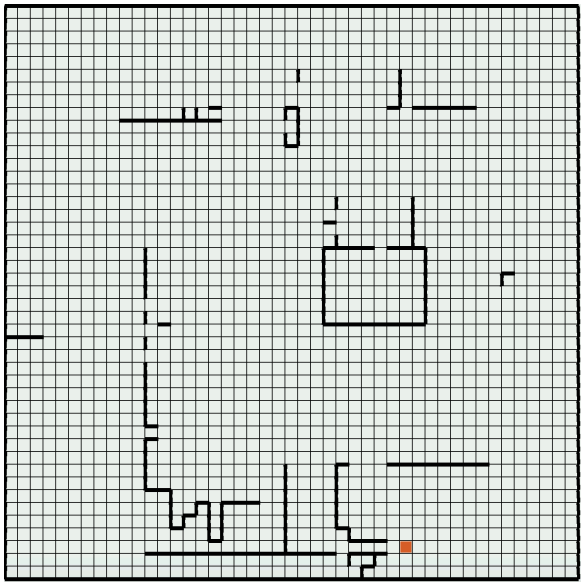

Figure 2 *45x45 Large Maze*

Value Iteration

Value iteration works backward by using the utilities and rewards to compute the optimal policy. The process starts with arbitrary utilities $V_0(s)$. Subsequently, the utilities $V_{i+1}(s)$ for each iteration $i+1$ are updated based on neighbors and discounted values $V_i(s)$ from iteration i. $V_{i+1}(s)$ represent the maximum possible rewards that can be obtained in $i+1$ iterations. The value iteration update used in the maze MDP problems is the following:

$$V_{i+1}(s) = \min (\text{a } \in \text{A}) \left\{ \sum_{s\prime} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right] \right\}$$

The process is repeated until convergence. The values propagate from their neighbors, so they eventually converge. Using this backward induction approach, the value at each state is calculated. The value update is calculated using Bellman's equation. It is important to notice that the value iteration update is non-linear, and it can have an exponential time complexity.

$$\pi^*(s) = \min (\text{a } \in \text{A}) \left\{ \sum_{s\prime} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right] \right\}$$

Approach

For the large maze as well as the small maze, the stochasticity (PJOG) values were varied from 0.0 to 0.9, with intervals of 0.1. The number of iterations taken to converge and time taken to run the algorithm were recorded for each *PJOG* value. For each stochasticity value, I saved the screenshots of the maze with the utility values. The darker shades of blue represent utility values that are less desirable, while the lighter shades of blue indicate preferred utility values. Since we are trying to minimize utility values, lower utilities are preferred over higher utilities.

Small Maze

As the stochasticity value increases, the number of iterations taken for the utility values to converge increases. This is explained by the increase in stochasticity and as a result, the number of unintended actions. After a stochasticity of 0.6, the number of iterations taken to converge increases exponentially. However, at a stochasticity of 0.9, the number of iterations reduces instead of increasing.

The average runtime for the small maze is 36 milliseconds. As the stochasticity increases, so does the runtime for the smaller maze. The graph for the runtime of the small maze looks very like the graph for the number of iterations to converge. This is expected given that more iterations will take more time to run.

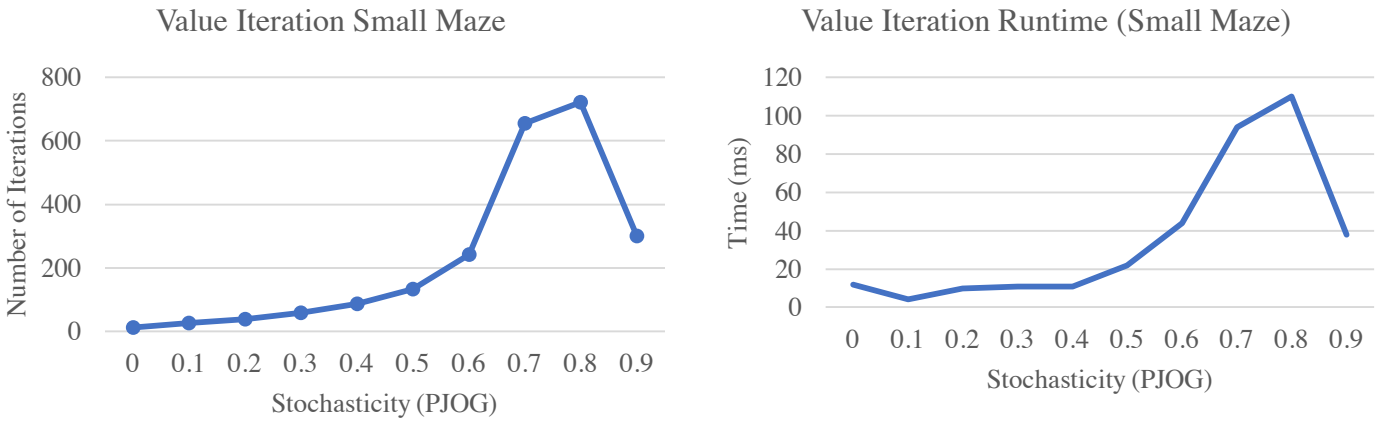

Figure 3 *Value Iteration: Number of Iterations vs Stochasticity (Left), Time vs Stochasticity (Right)*

The mazes with the value functions for 0.0, 0.3, 0.5, and 0.8 stochasticity values are shown below. For the 0.0 stochasticity, the shorter route to the goal (cell (9, 1)) is taken since the agent's actions are completely deterministic. For the 0.3 stochasticity, the longer route to the goal (cell (9, 5)) is preferred, given that the goal in (9,1) is closer to the wall obstacles, which makes it riskier. This is similar in the mazes with 0.5 and 0.8 stochasticity.

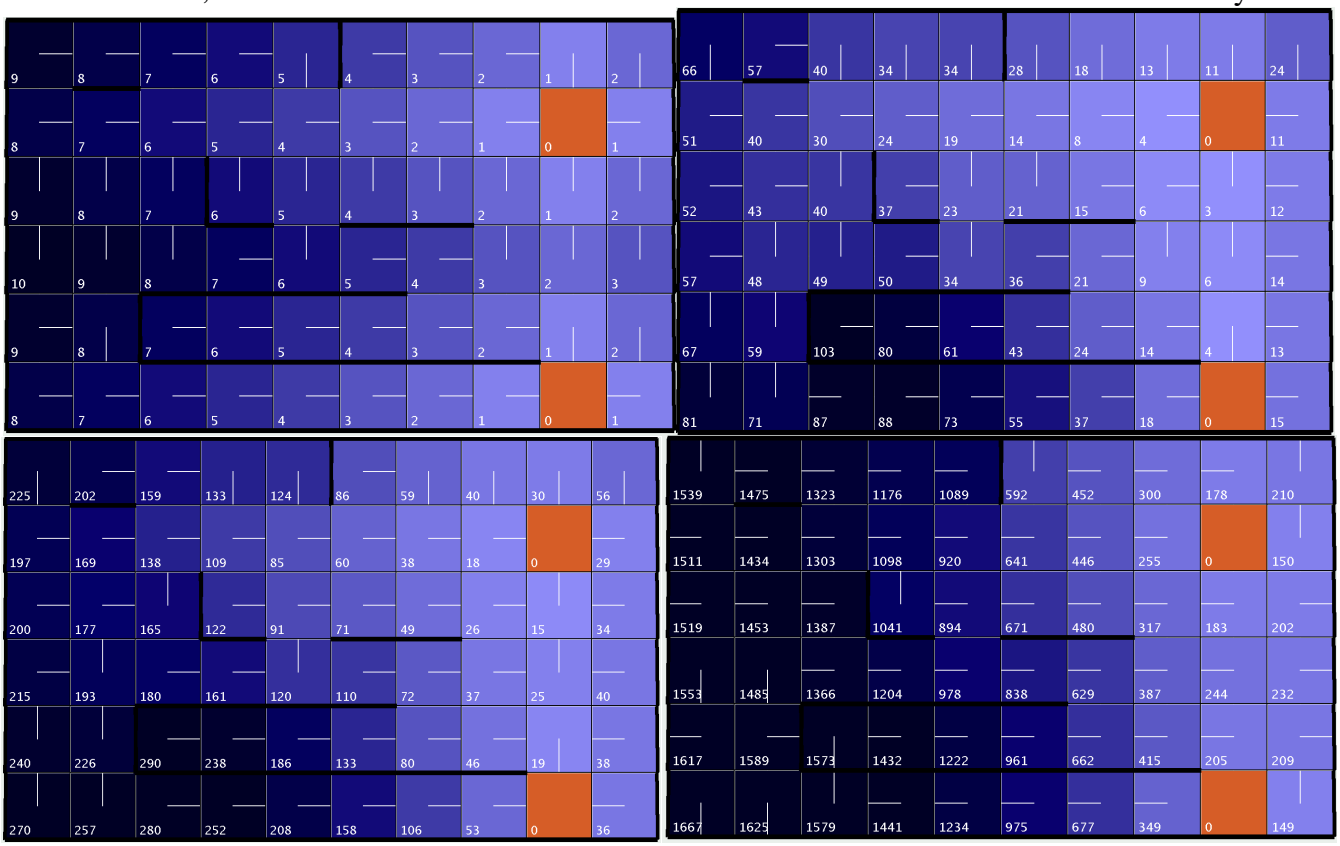

Figure 4 *Small Maze Value Iteration 0.0 (top left), 0.3 (top right), 0.5 (bottom left), 0.8 (bottom right) stochasticity*

The 0.8 stochasticity maze shows some interesting results. Near the goals, the intended actions would naturally be those that would lead to the goal state. However, for example, the goal cell in (9,5) has surrounding cells that have actions that lead to states other than the goal. On the other hand, the 0.5 stochasticity maze still preserves the correct intended actions near the goals in cells (9,5) and (9, 1). This occurs in the 0.8 stochastic maze because the probability of the intended action is 0.2 (1-PJOG = 1-0.8 = 0.2), while the probability of the other unintended actions (including the one that would lead to the goal) is 0.26 (PJOG/(N-1) = 0.8/3 = 0.26). Therefore, intending to take an action that does not lead to the goal turns into the better strategy. By comparing the 0.5 and 0.3 stochasticity mazes, we can observe that a higher stochasticity makes the utility of each state less desirable. For instance, with 0.3 stochasticity the optimal action for cell (6, 6) is *down*. Whereas, with 0.5 stochasticity the optimal action for

cell (6, 6) is *right*. This is expected since 3 cells below there is another wall, even though this could result in a shorter path to the goal. Therefore, the 0.5 stochastic agent prefers a slightly longer route to the goal which is safer.

Large Maze

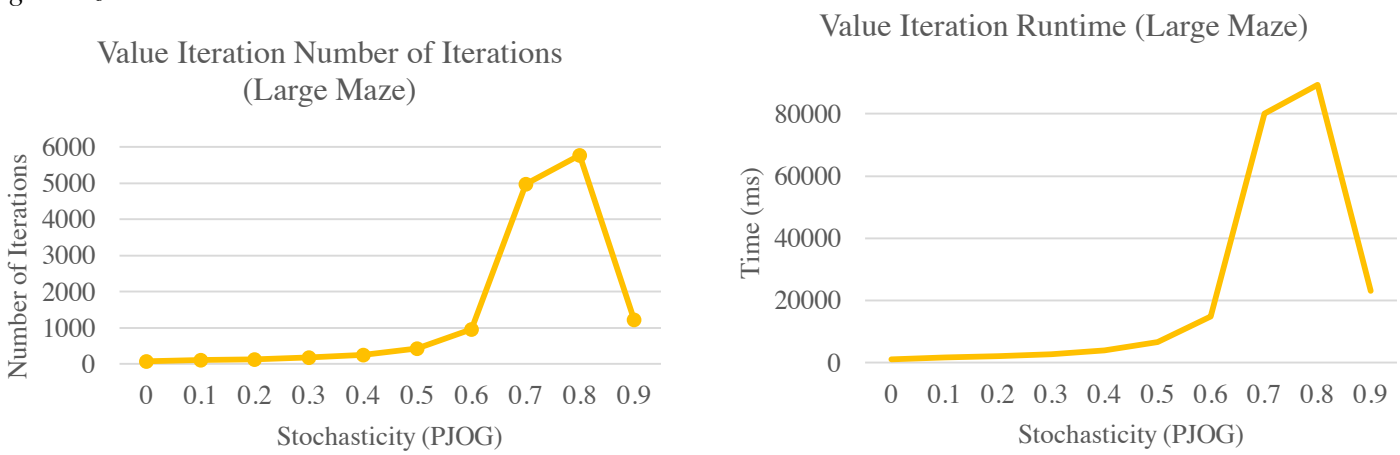

Figure 5 *Value Iteration: Number of Iterations vs. Stochasticity (left), Time vs. Stochasticity (right)*

Like the small maze, the large maze requires more iterations to converge as the stochasticity increases. However, the anomaly with a stochasticity of 0.9 prevails with the larger maze as well. The graph for the runtime of the large maze shows similar findings as the small maze. Both the runtime and number of iterations graphs have a similar shape. The larger mazes with 0.0, 0.3, 0.5, and 0.8 stochasticity are shown below. The 0.0 stochasticity shows that cells that are closer to the goal have a better utility than cells that are farther, as expected. As we proceed to a 0.3 stochasticity, the areas around the obstacles become much darker representing worse utilities for each state. Since there is a 0.7 (1 - PJOG = 1-0.3 = 0.7) probability that the agent's actions will result in its intended moves, but 0.3 probability that they will not, there is more precaution to avoid the obstacles. Comparing the mazes with 0.3 and 0.5 stochasticity, the optimal policies for both mazes differ mainly in cells that are in high proximity of the obstacles. That is because the agent with 0.5 stochasticity needs to take much safer routes than the agent with 0.3 stochasticity. The agent with 0.8 stochasticity follows a similar principle. The areas around the obstacles are darkest for the maze with 0.8 stochasticity. The agent with 0.8 stochasticity in the large maze presents an odd optimal policy for states around the goal. Instead of having optimal policies that face toward the goal, they face away. However, this is because the agent has 0.2 (0.2 = 1-0.8) probability of resulting in the intended action, as seen for the small maze.

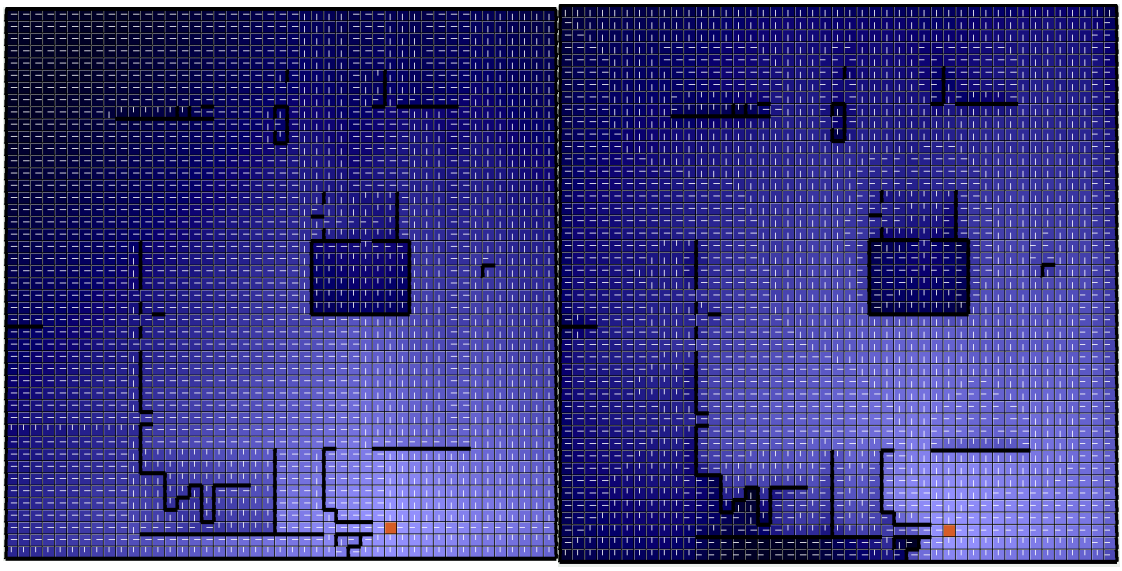

Figure 6 *Large Maze Value Iteration with 0.0 (left), 0.3(right) stochasticity*

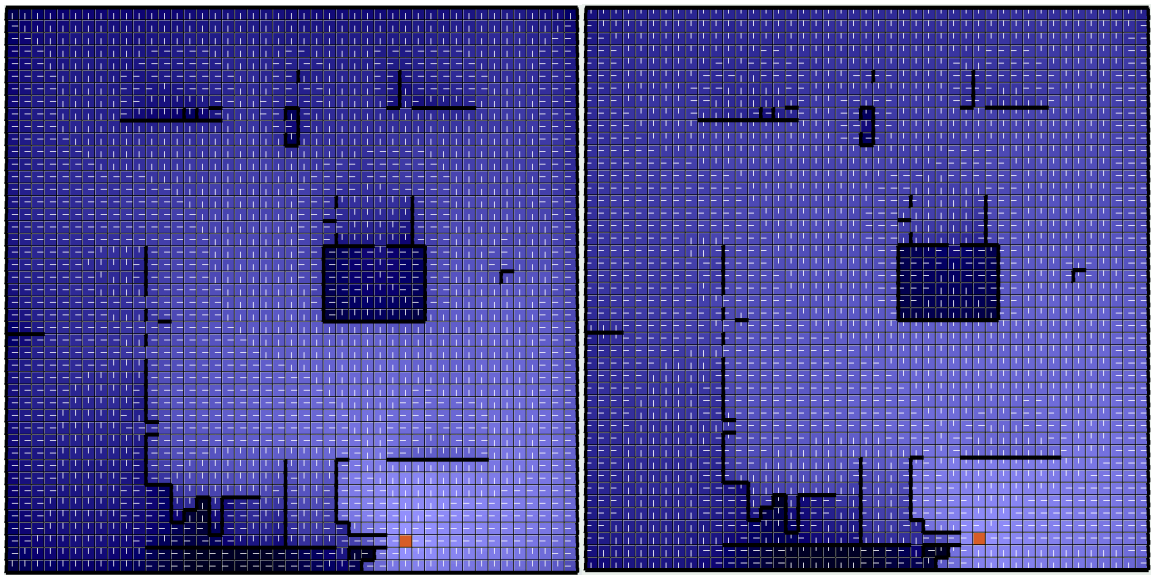

Figure 7 *Large Maze Value Iteration with 0.5 (left) and 0.8 (right) stochasticity*

Comparing Small Maze and Large Maze

Stochasticity (PJOG)	Iterations (Small Maze)	Iterations (Large Maze)	Time (Small Maze, ms)	Time (Large Maze, ms)
0	11	74	12	1162
0.1	27	106	4	1671
0.2	39	136	10	2119
0.3	58	181	11	2773
0.4	87	260	11	4034
0.5	132	428	22	6717
0.6	241	956	44	14778
0.7	656	4974	94	80103
0.8	722	5769	110	89325
0.9	300	1226	38	23063

From the data presented, we can observe that the large maze has higher runtimes and takes more iterations to converge. The average number of iterations for the small maze is 227, while the average number of iterations for the large maze is 1411. The average runtime for the small maze is 36 milliseconds, while for the large maze it is 22,575 milliseconds. Both show a drop in runtime and number of iterations with a stochasticity of 0.9.

Policy Iteration

Policy iteration will start with an arbitrary policy π_0 and calculate new utilities based on the policy. Then, it updates utility V_i corresponding to policy π_i at iteration i. The main difference between policy iteration and value iteration is that policy iteration uses the policy function to define the action, which eliminates the argmin and results in a linear system. This results in a linear system of equations that can be solved for all states s.

$$V_{\pi_i}(s) = \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V_{\pi_i}(s')\right]$$

The policy π_{i+1} at each state for iteration $i + 1$ continues to update, until there are no improvements possible. In policy iteration, we are concerned with finding the best policy. Therefore, we are not concerned with the actual values of the utilities, even though they are used in the update process.

$$\pi_{i+1}(s) = \min (a \in A)\{\sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V_{\pi_i}(s')\right]\}$$

Small Maze

For the small maze, the number iterations taken to converge to the optimal policy reduces exponentially as the stochasticity increases. As the noise in the environment increases, the policy will not matter as much. The probability of the actions leading to a position in one of the undesired states increases with stochasticity. The time shows a similar trend until the stochasticity reaches 0.4. With a stochasticity of 0.5 and above, the time increases.

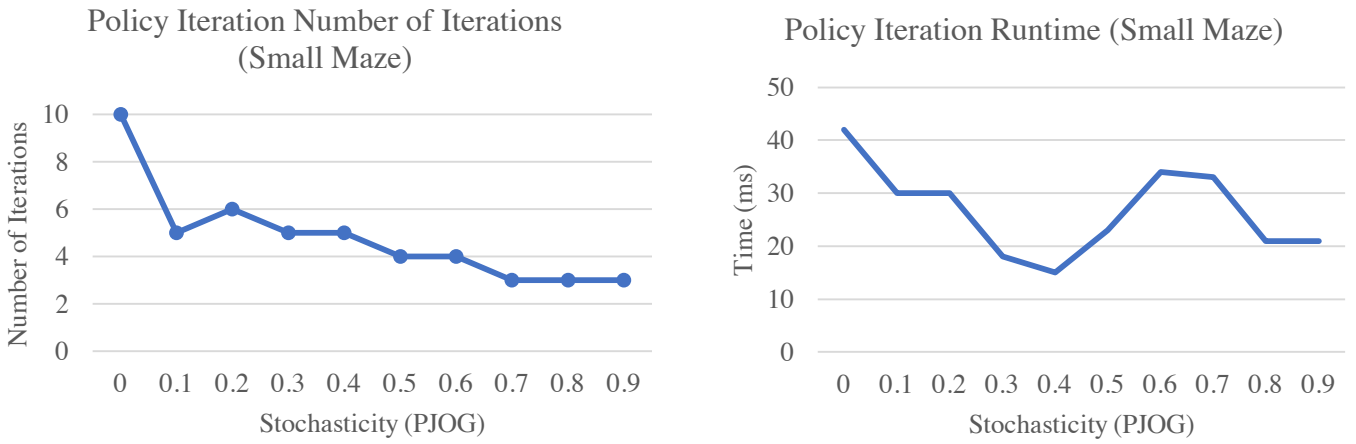

Figure 8 *Policy Iteration: Number of Iterations vs. Stochasticity (left), Time vs. Stochasticity (right)*

By looking at the mazes from the 0.0, 0.3, 0.5, and 0.8 stochasticity values we can see some similarities and differences compared to value iteration. The higher the stochasticity, the less desired are the utility of the areas around the obstacles, represented by the darker blue. This prevails in value iteration and policy iteration. Regardless of the stochasticity, the cells around the goal state have an optimal policy that would lead to the goal state. This only occurs in policy iteration, not in value iteration.The optimal policies derived through value iteration and policy iteration are similar at 0.0, 0.3, and 0.5 stochasticity. At a stochasticity of 0.8, the utilities for policy iteration become much lower, and the optimal policy differs is different from that obtained with value iteration.

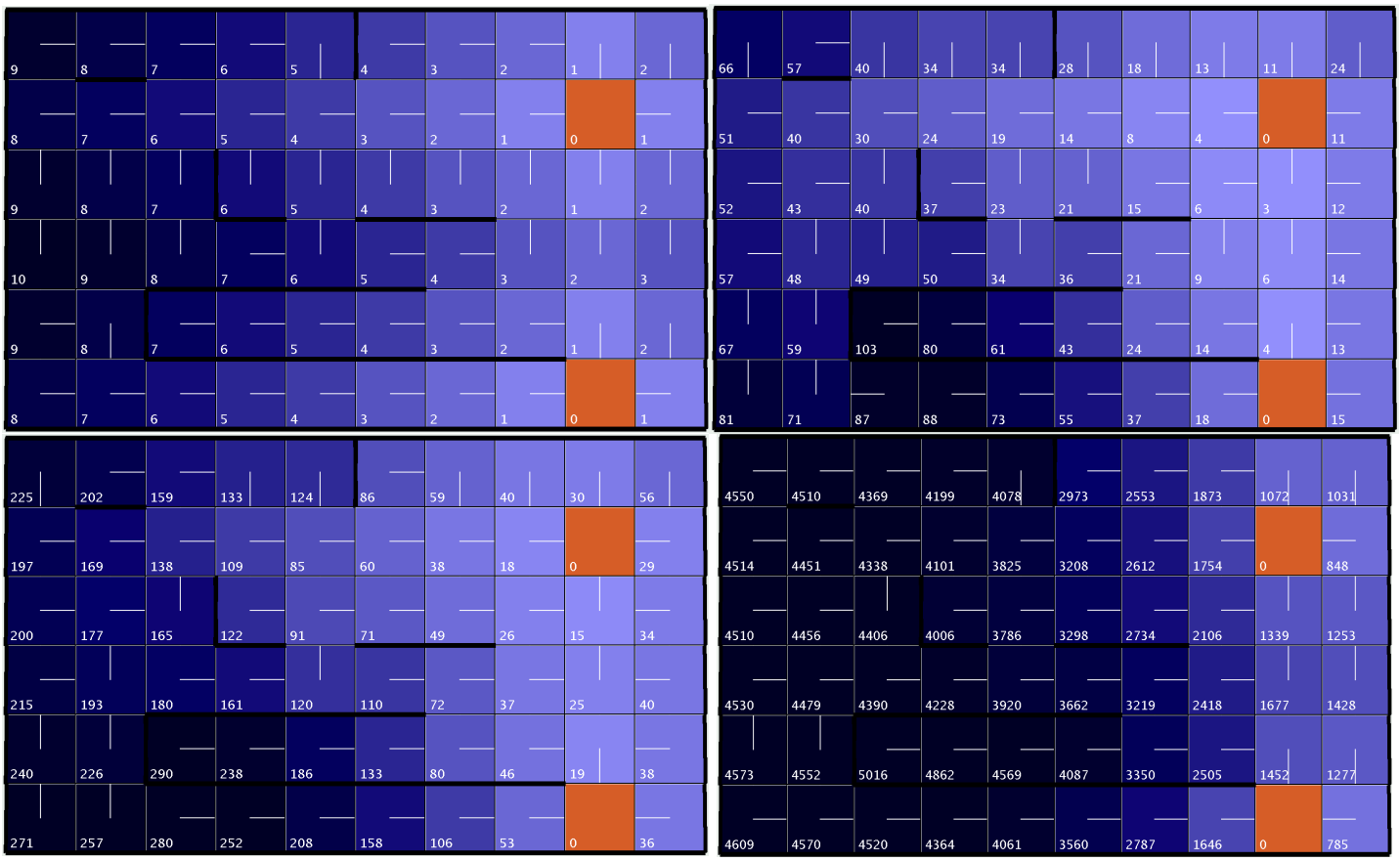

Figure 9 *Small Maze after Policy Iteration with 0.0 (top left), 0.3(top right), 0.5(bottom left), and 0.8(bottom right) stochasticity*

Large Maze

For the large maze, the number of iterations taken to converge reduces as the stochasticity increases. This is not the trend that we see in the runtime for the larger maze. The time reduces for a stochasticity of 0.1, but after this there is a general increasing trend in the time taken to find the policy. This could be attributed to the larger size of the maze and the increased randomness due to the stochasticity.

Figure 10 *Policy Iteration: Number of Iterations vs. Stochasticity (left), Time vs. Stochasticity(right)*

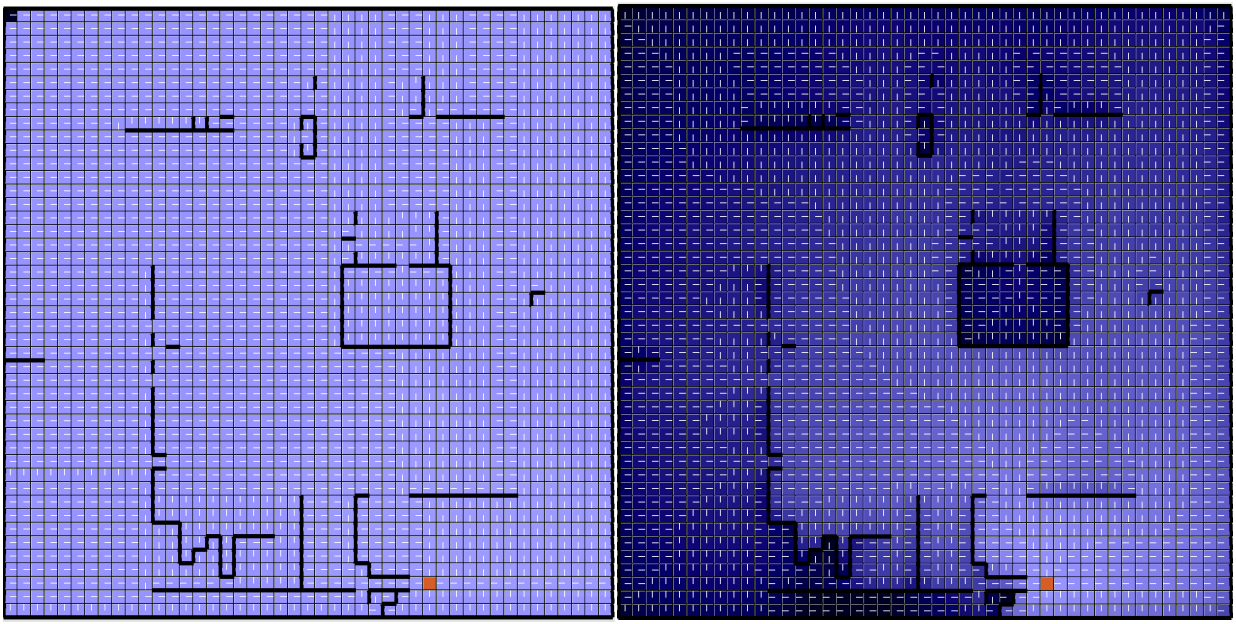

Figure 11 *Large Maze after Policy Iteration with 0.0 (left) and 0.3 (right) stochasticity values*

The mazes with stochasticity of 0.0, 0.3, 0.5, and 0.8 are shown in the images. The maze with 0.0 stochasticity shows an equal utility for all cells, except one corner. Due to the actions being completely deterministic, all cells have the same utilities and they are equally desirable utilities. As the stochasticity increases, the overall utilities in the states reduce. These result in much lower utilities near the obstacles and higher utilities farther from the obstacles. It is very interesting to point out that the maze with 0.3 and 0.5 stochasticity result in optimal policies that are exactly as those in value iteration. With the 0.8 stochasticity maze, the optimal policy differs at few locations near the obstacles. Other than that, the optimal policy derived for the large maze is pretty similar in policy iteration and value iteration.

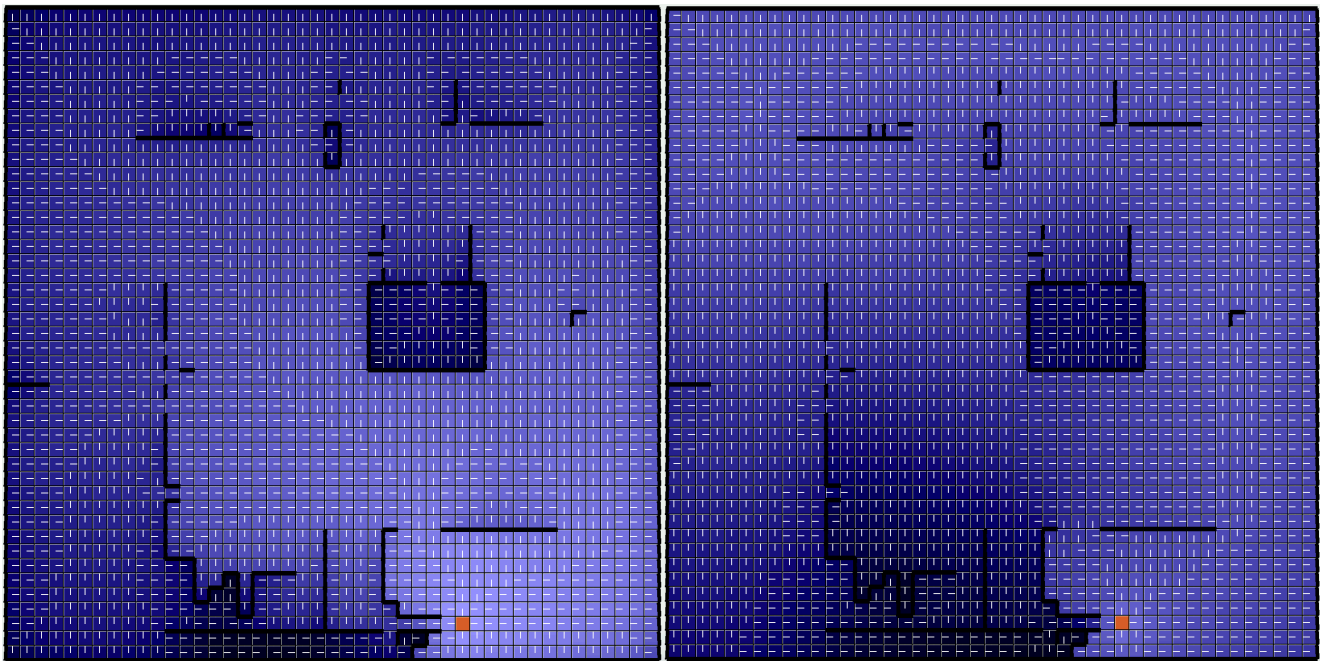

Figure 12 *Large Maze after Policy Iteration with 0.5 (left) and 0.8 (right) stochasticity values*

Comparing Small Maze and Large Maze

Stochasticity (PJOG)	Iterations (Small Maze)	Iterations (Large Maze)	Time (Small Maze, ms)	Time (Large Maze, ms)
0	10	66	27	8578
0.1	5	49	30	7987
0.2	6	17	30	9836
0.3	5	15	18	8113
0.4	5	13	15	8384
0.5	4	15	19	10155
0.6	4	8	34	12112
0.7	3	9	42	16377
0.8	3	92	21	2001
0.9	3	161	21	1783

For both mazes, there is a decrease in the number of iterations run as the stochasticity increases. For the small maze, the average number of iterations is 5, whereas the large maze takes an average of 45 iterations to converge. The average time taken for the small maze is 26 milliseconds, while the average runtime for the large maze is 8533 milliseconds. The larger maze requires higher more time and more iterations to converge, as expected.

Comparison Value Iteration and Policy Iteration

For the small maze and the large maze, the average time for value iteration was much higher than that for policy iteration. There is a similar trend for the average number of iterations: value iteration takes more iterations to converge than policy iteration. This is because value iteration is nonlinear and has an exponential time complexity. As the stochasticity increases, the average utility of each state decreases in value iteration and policy iteration. In policy iteration, as the stochasticity increased the number of iterations reduced, whereas value iteration showed that the number of iterations increased. Considering runtime and iterations taken to converge, policy iteration would be the recommended approach.

Reinforcement Learning

Q-Learning

Q-learning is a reinforcement learning technique. It is used to find the optimal policy for a finite MDP. The advantage of Q-learning is that it can make comparisons between the expected utilities of the possible actions without having a model of the agent's environment. Q-learning uses a function that is calculates the Quantity for some state *s* and action *a* combination. This uses an update like value iteration by using the old value to predict the new value. Q-learning uses the following: old Quantity value $Q(s_t, a_t)$, learning rate α_t (number between 0 and 1), the reward r_{t+1}, discount factor γ to predict the estimate of the optimal future value $\max_a Q(s_{t+1}, a)$.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t * ((r_{t+1} +) + \gamma * \max a\, Q(s_{t+1}, a) - Q(s_t, a_t))$$

The Q-learning algorithm implements some exploration policies to find the optimal policy for the MDP. It is epsilon (ϵ) greeedy, which adds some randomness to the algorithm. The agent will select the best action with $1 - \epsilon$ probability. The algorithm uses a form of 'random restarts' by changing the start state and gives the agent a high virtual reward for a state and action that have not been explored for a while. The learning rate has a decay function (decay = 1000/(1000 + number of episodes)).

Approach

For both mazes, Q-learning was run with a learning rate of 0.7. The epsilon values were varied from 0.0-0.8, and the PJOG values were varied from 0.0-0.8, with a difference of 0.1. Each maze was run over 100,000 cycles.

Small Maze

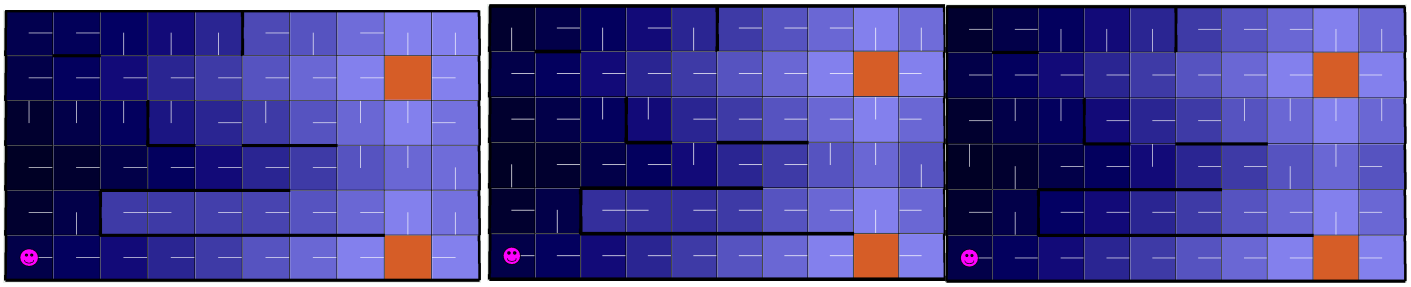

Figure 13 *Small Maze after Q-Learning PJOG = 0.0: Epsilon 0.25 (left), Epsilon 0.50 (middle), Epsilon 0.75 (right)*

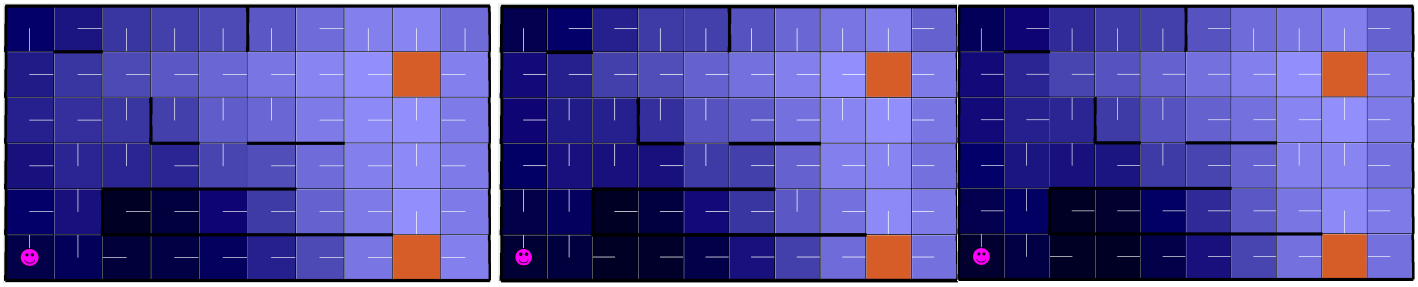

Figure 14 *Small Maze after Q-Learning PJOG = 0.5: Epsilon 0.25 (left), Epsilon 0.50 (middle), Epsilon 0.75 (right)*

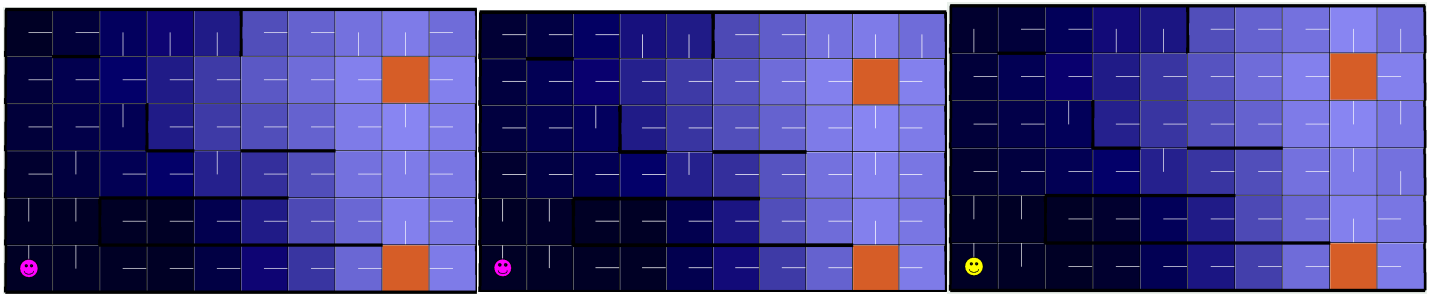

Figure 15 *Small Maze after Q-Learning PJOG = 0.8: Epsilon 0.25 (left), Epsilon 0.50 (middle), Epsilon 0.75 (right)*

For the small maze, the mazes with the different stochasticity values and epsilon values were compared to the policies obtained in value iteration and policy iteration. For 0.0 stochasticity, none of the policies derived in q-learning resemble those derived in policy iteration and value iteration. Even with no randomness (epsilon = 0), the policy differs at approximately 8 (13%) of the 60 states. For 0.5 stochasticity, epsilon value of 0.25 results in a policy that differs in 6 (10%) of the 60 states. For the 0.8 stochasticity, 0.5 epsilon results in a change in the optimal policy for 2 (0.04%) of the 60 states, while 0.75 epsilon results in a different policy for 5 (0.08%) of the 60 states. Therefore, an epsilon value of 0.5, a balance of exploration results in the best approximation to the optimal policy.

Large Maze

For the large maze, the same epsilon values and stochasticity result in many more differences in the optimal policy obtained in value iteration and policy iteration compared to the optimal policy obtained in Q-learning. This is a result of larger number of states. With a higher number of states, there are more possibilities for the optimal policy. With Q-learning, all the states have much lower utilities than with policy iteration and value iteration at the same stochasticity. These lower utilities are what cause the changes in the optimal policy for Q-learning. The optimal policies differ a significant amount from those obtained by policy iteration and value iteration. The algorithm was given 100,000 cycles to converge, but there is a possibility that more cycles would have allowed the algorithm to converge to the optimal policy.

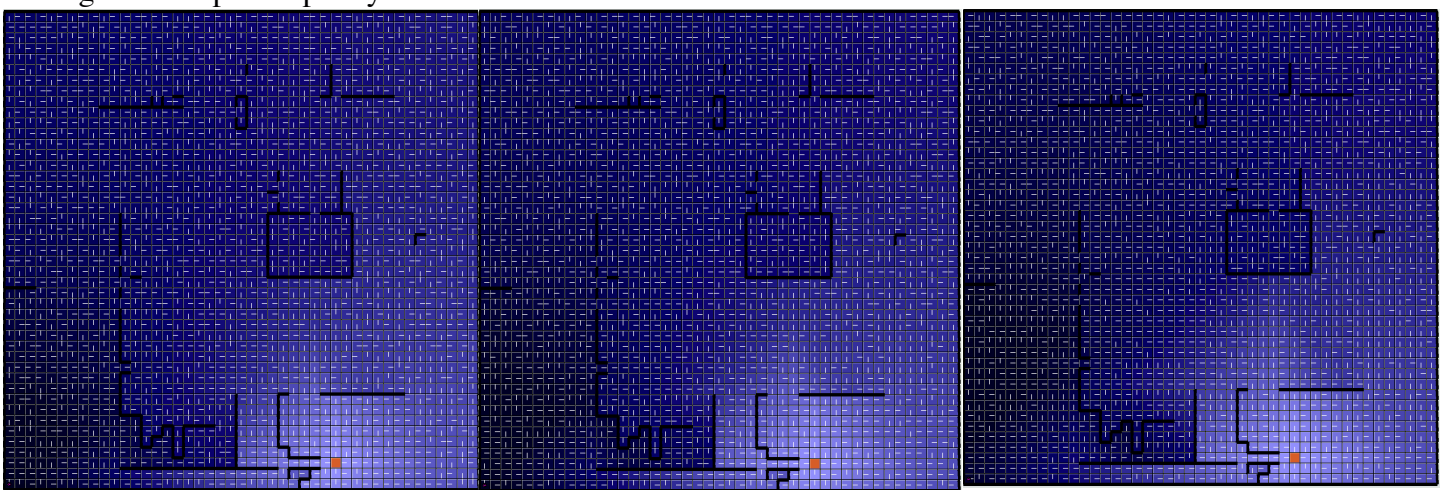

Figure 16 *Large Maze after Q-Learning PJOG = 0.0: Epsilon 0.25 (left), Epsilon 0.50 (middle), Epsilon 0.75 (right)*

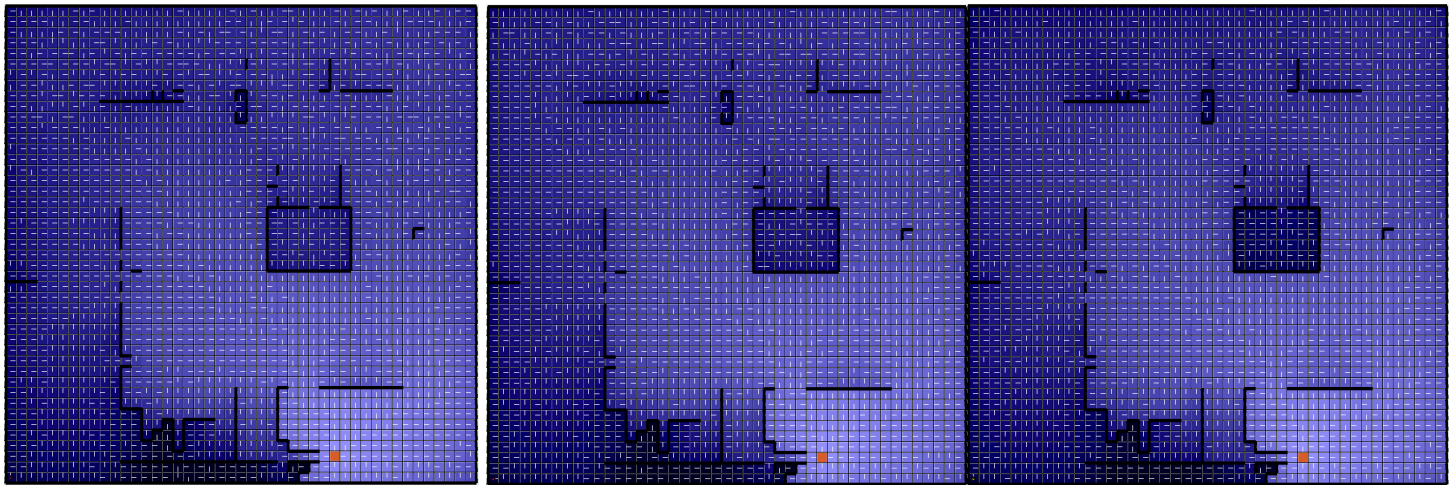

Figure 17 *Small Maze after Q-Learning PJOG = 0.8: Epsilon 0.25 (left), Epsilon 0.50 (middle), Epsilon 0.75 (right)*

Conclusion

After analyzing value iteration, policy iteration, and Q-learning approaches of solving an MDP, I have concluded that policy iteration is the recommended approach when the environment is known, while Q-learning can provide useful approximations for an unknown environment. As mentioned earlier, policy iteration has a lower runtime and lower number of iterations. Q-learning can take a higher number of cycles and higher runtime if the MDP has a high number of states. In addition, Q-learning requires parameterization to ensure there is a balance of exploration and exploitation. Q-learning is more practical, given that there are many more real-world problems in which the nature of the environment is unknown. This enables reinforcement learning to solve many more problems than policy iteration would be able to. Therefore, Q-learning has a lot more applications than policy/value iteration.