Markov Decision Processes Analysis

John Seon Keun Yi

CS4641

Introduction

In this assignment we analyze three reinforcement learning algorithms: value iteration, policy iteration and Q-learning. The algorithms are performed in two different Markov decision processes. (MDPs) An MDP provides a mathematical framework for decision making, particularly when the outcome of a decision is partly stochastic. That is, when the outcome following a decision or action is sometimes random, and other times is as expected. In this analysis, the three algorithms are run on different MDPs and the results between the algorithms are compared through various metrics.

GridWorld

I used two "GridWorld" problems for the MDPs, similar to the examples we went over in class. GridWorld refers to a world in which an agent (robot) moves through a discrete state space represented as a block of grids. Each state is a cell in the grid, and there exists obstacles and one or more terminal (goal) states. I used RL_sim to generate the two problems and test them on the three algorithms.

Basic rules apply in this world. The agent can move in four directions: up, down, left and right. Thus, there are four actions the agent can take. Also, to add in the stochastic nature of MDPs, a variable is added to represent noise in the environment. Noise of an action, named "pjog" in RL_sim, determines the probability that the agent will take some other action and end up in different states. For example, if pjog is 0.3, and the action taken is a=RIGHT, the agent will instead take action UP, DOWN and LEFT at a probability of pjog/(number of actions-1) = 0.1 each.

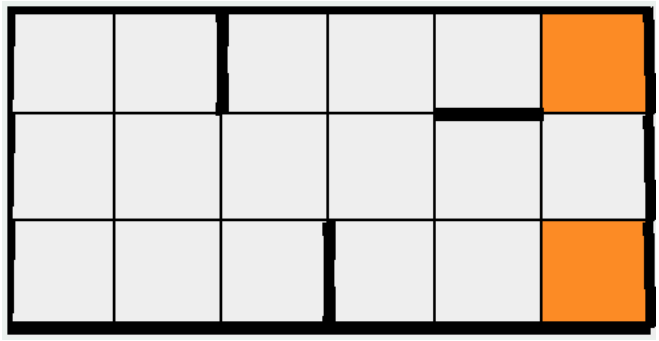

Figure 1. Map of the small maze.

When the agent makes a transition of one state to another, it gains a path cost of 1. However, when the agent hits a wall, it stays on the same state and receives a penalty of -50. The purpose of the penalty is to encourage the agent to avoid hitting walls and converge quickly. Finally, the goal state(s) is marked as an orange block in the maze.

The first MDP problem is a small maze, as shown in Figure 1. The small maze is a 3x6 maze with 18 states. The start state is the lowest leftmost block. The walls are represented as bold lines in the maze. There are two goals in this maze. The reason I put two goals is that I thought it would be interesting to see which one of the goals the algorithms prefer, and if the preference changes between algorithms.

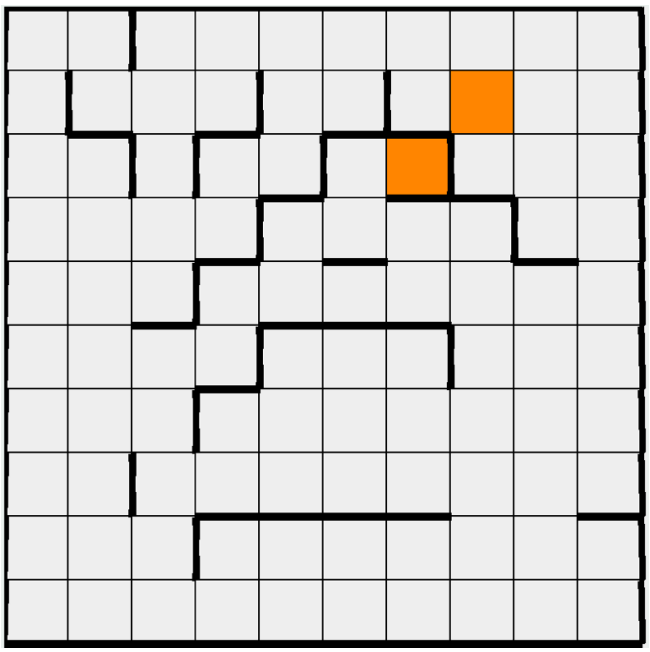

Figure 2. Map of the big maze.

Value Iteration

The first algorithm applied to solve the two MDP problems is value iteration. Value iteration applies arbitrary utilities (values) to each state and updates the utilities by adding the immediate reward of the state with the expected discounted reward of the state if the agent takes optimal actions onward. For each state, value iteration simply chooses the action that maximizes the utility of the following state. The process of value iteration is as follows:

1. Start with arbitrary utility values assigned to each state
2. Update the utility of each state based on the utilities of its neighbors
3. Use the Bellman update equation to update the utility for each state

$$\widehat{U_{t+1}}(s) = R(s) + \gamma \max_a \sum_{s\prime} T(s,a,s')\widehat{U_t}(s')$$

4. Repeat steps 2 and 3 until convergence

After convergence, the optimal policy π^* for each state s is determined by the equation below. Essentially the action that maximizes the expected utility is selected.

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s\prime} T(s,a,s')U(s')$$

Value iteration requires some randomness in the agent's actions in order to find the optimal path. If there is no noise in an action, the value for the transition function $T(s,a,s')$ will be always 1, and the optimal path will be stuck in a local maximum. In RL-sim, the variable pjog takes care of the noise. As explained in the introduction, pjog determines the probability that the agent will take some other action and end up in different states. To test the relationship of randomness of action with the result of value iteration, I varied pjog from 0.1 to 0.5 and compared the number of iterations (steps) to converge. Going over 0.5 made the agent to stay in the starting state, since taking some action is more likely to result in a wrong action.

Figure 3 plots the number of steps required to converge depending on the pjog values for the two MDPs. As one can see, the number of steps increase as the randomness (pjog) increases. Such behavior is displayed in both MDPs. The big maze takes more steps to converge overall, since it has a larger number of states. The numbers spike at pjog of 0.5, because this is essentially a fifty-fifty situation where the change of performing the right action is 0.5. From this plot, I concluded that although adding a component of randomness is important, adding too much of it will increase

the time till convergence.

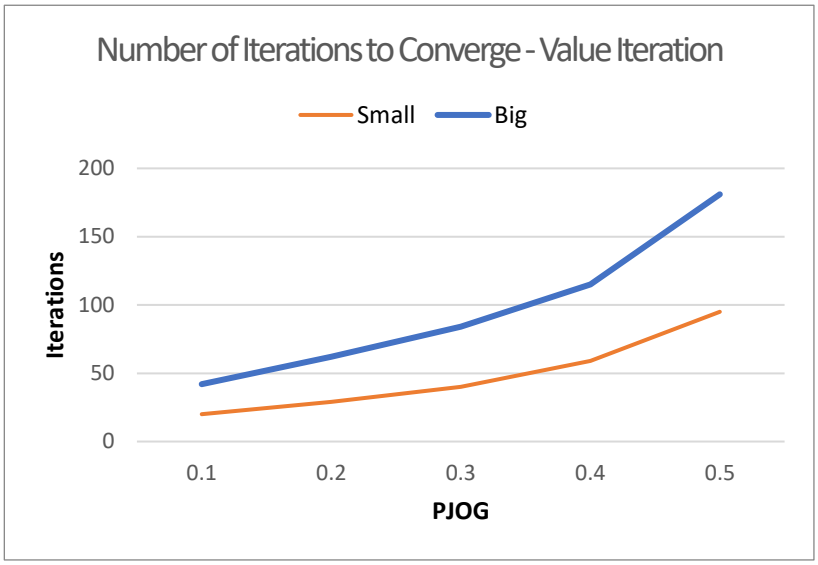

Figure 3. Randomness (PJOG) vs number of iterations to converge.

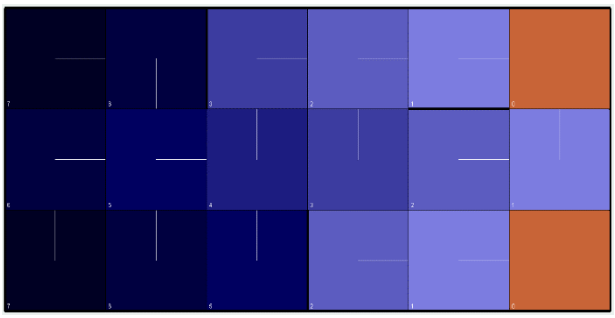

Figure 4. Result of value iteration with pjog=0.

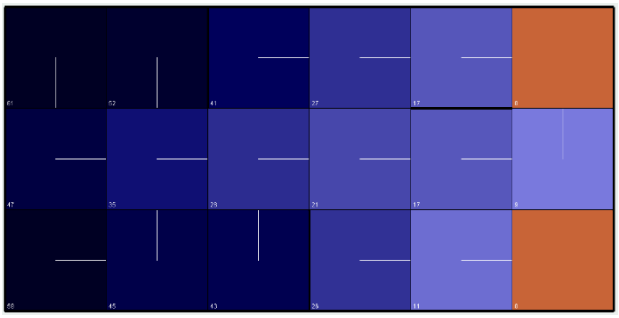

Figure 5. Result of value iteration with pjog=0.3.

Next, I analyzed if different pjog values lead to different convergences. For the small maze, all pjog values from 0.1 to 0.5 converged to the same result: it preferred the upper goal. The general path was similar to that of Figure 5. The agent preferred to use the middle row in order to avoid hitting the walls in case noise in action occurs. What if there were no randomness in action? Figure 4 display the result with pjog of zero. Although it converges to the same result as the ones with randomness, it takes a different path. Since there is no risk of error in action, the optimal path sticks to the wall instead of going though the middle row.

The big maze yielded different results. As seen in Figure 6, while pjog of 0.2 and above all converged to the goal on the upper right, value iteration with pjog=0.1 converged to the lower left goal. Without the walls, the lower goal has the shortest Manhattan distance. Thus, if there were little or no risk of noise, it is statistically more reasonable to converge to the lower goal. In the case of the big maze, the agent assumes that the risk of noise is low enough up to the pjog value of about 0.1. Although going through the path to the lower goal means the agent has a risk of hitting the walls marked in red, there is higher utility in converging to the lower left goal. When pjog is over 0.1, the agent determines that it is too risky to converge to the lower goal. Thus, it converges to the upper right goal -despite a longer "distance"- which avoids the risk of hitting the walls and ultimately has higher utility.

Policy Iteration

Unlike value iteration, which updates the utility of the states and the extract policies from it, policy iteration iterates directly in policy space. Policy iteration starts with an arbitrary initial policy π_0, computes the utility of each state given the current policy π_t, and improves the policy by choosing a new optimal policy for each state based on

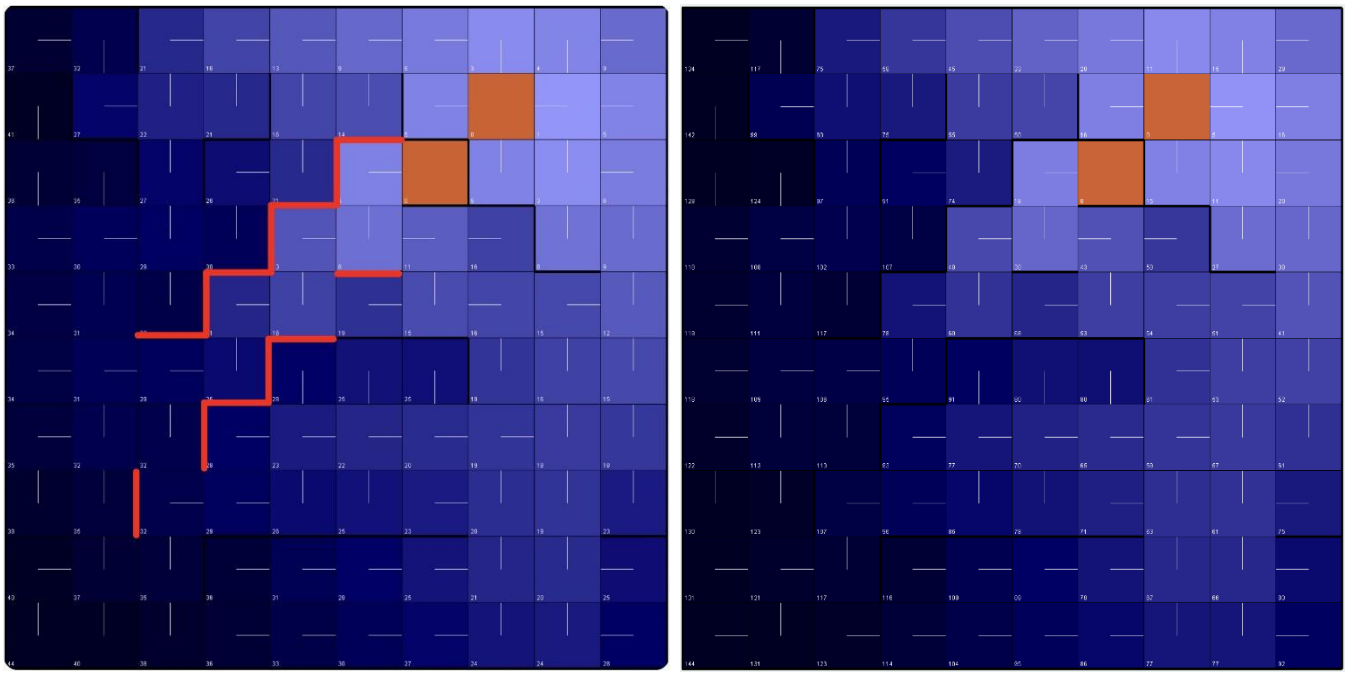

Figure 6. From left to right: result of value iteration with pjog=0.1 (left) and pjog=0.3 (right).

the policies computed. The value update equation is similar to the Bellman update in value iteration, only that this one has a fixed known policy in the transition function.

$$U_t(s) = R(s) + \gamma \sum_{s'} T(s, \pi_t(s), s') U_t(s')$$

Since the above equation can be solved in linear time, policy iteration generally converges faster than value iteration, although with a cost of greater computational expense.

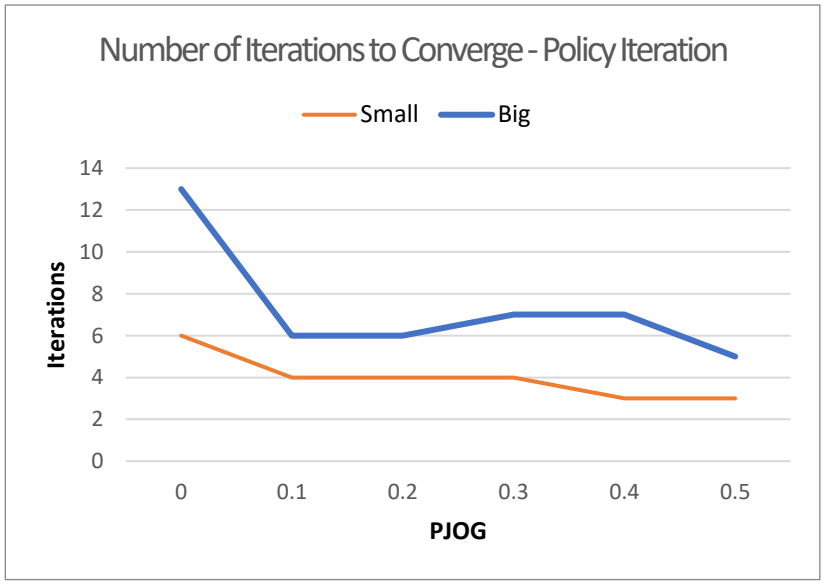

Figure 7. Randomness (PJOG) vs number of iterations to converge.

Figure 7 plots the varying number of steps to converge with pjog values from 0 to 0.5. Unlike value iteration, the number of iterations show a decreasing trend as action stochasticity increases. As the pjog value increases, there is a

greater chance that the agent will end up in a different state. A higher pjog value allows the agent to explore more states and determine the optimal policy before it converges to the goal. Thus, while an increased randomness in action may increase running time, the agent will converge in fewer steps.

Like on value iteration, the big maze with more states took more steps to converge. However, the big maze performed quite similar to the small maze, with about two to three iterations more in general than the small maze. The small maze converged to the same result (upper goal) for all pjog values. As in Figure 5, the agent preferred the middle path to avoid hitting walls. The big maze preferred the lower left goal for pjog values lower than 0.1 but converged to the upper right goal for values bigger than 0.1.

Value Iteration vs Policy Iteration

Overall, policy iteration took much fewer steps to converge. While value iteration took at least 20 to at most 180 steps to converge for both maps, policy iteration took less than 10 steps for all cases (with pjog>0). This is because policy iteration directly determines the policy instead of indirectly determining the optimal policy based on utility as on value iteration.

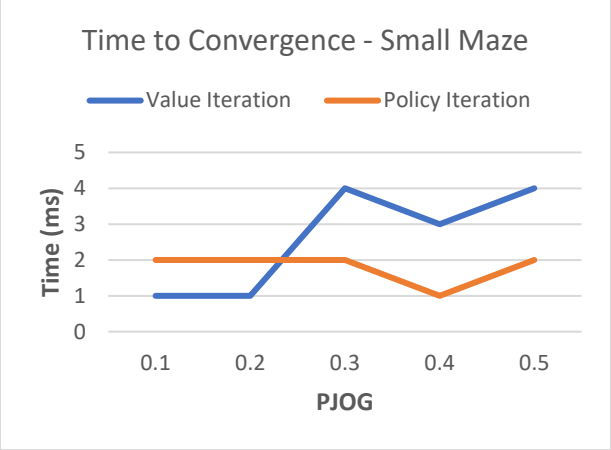

Figure 8. Time to convergence for value and policy iteration run on the small maze.

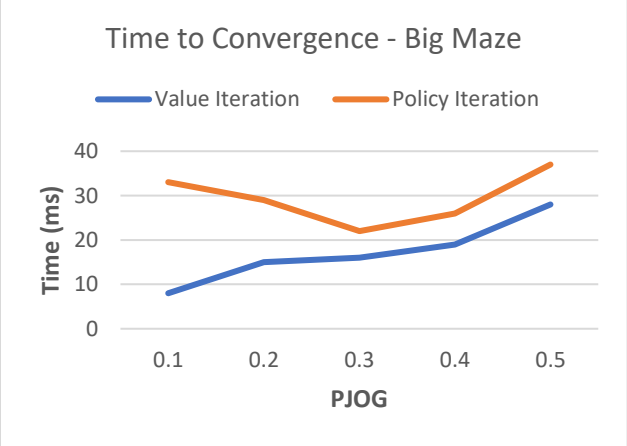

Figure 9. Time to convergence for value and policy iteration run on the big maze.

However, comparing running time returned different results. Figure 9 displays the running time to convergence with value and policy iteration run on the big maze. As one can see, policy iteration always took more time than value iteration. This is because, although policy iteration converges in fewer steps, each iteration takes a significantly longer time than that of value iteration. Interestingly, the computing time for policy iteration was not significantly longer than value iteration. The difference between times was reduced to about 5ms for pjog larger than 0.3, as the number of steps for policy iteration decreased and increased for value iteration. In fact, in Figure 8, both algorithms run on the small maze, policy iteration excelled in computation time after pjog of about 0.25. Although policy iteration takes more time each step, since it took such few steps (3) compared to that of value iteration (95), it ultimately took a shorter time to converge.

Policy iteration takes fewer iterations to converge at the cost of more expensive computation time. Value iteration takes less time each iteration but has to go through much more steps to converge.

Other than convergence time and number of steps to converge, value and policy iteration returned the same results. Figure 10 shows a comparison of value and policy iteration performed on the big maze, with the same pjog values of 0.3. One can see that they display the same results and converge to the same goal. This is because the objective of both policy and value iteration is to find the optimal policy in each state. Although the approach of the two algorithms are slightly different, they ultimately converge to the same goal in MDP problems.

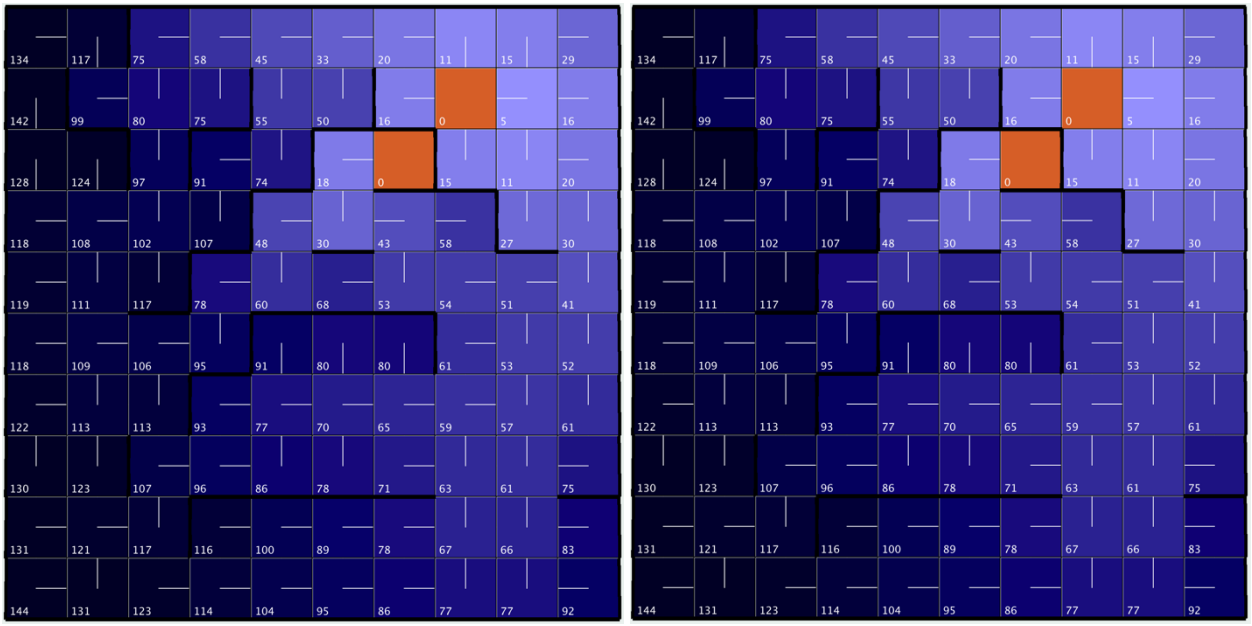

Figure 10. From left to right: result of value iteration (left) and policy iteration (right) performed on the big maze.

Q-Learning

The reinforcement learning algorithm I selected was Q-learning. Q-learning is used when the agent does not know the reward and transition functions for each state. Since the agent doesn't always know information about the environment such as in value and policy iteration, I would say Q-learning is more analogous to real-life situations.

Q-learning has two hyperparameters: epsilon and learning rate. Epsilon is used in the epsilon-greedy exploration policy. A high epsilon value makes the agent to explore more rather than take the optimal action. For example, an epsilon of 0.1 means that the agent has a probability of 0.9 to take the best action in the state, and probability 0.1 to take actions other than that. We have to allow some value of epsilon for the agent to explore and not get stuck in local minima, but too high a value will allow too much exploration and ultimately take long to converge to an

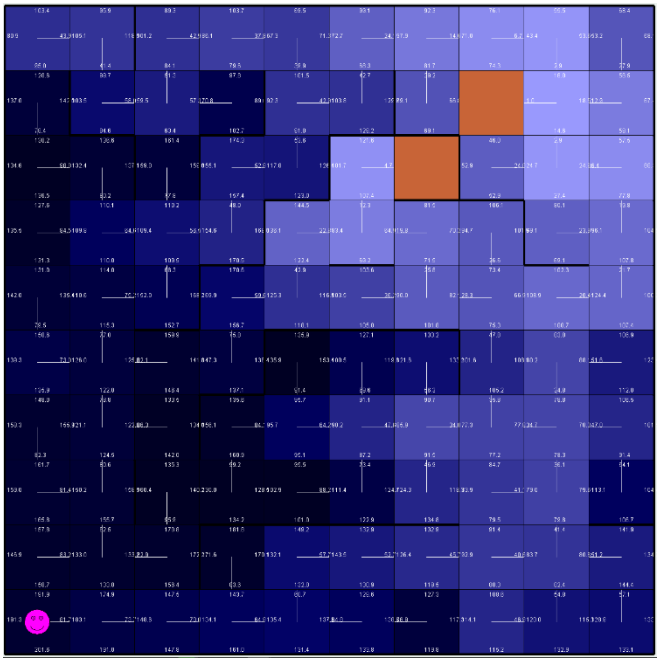

Figure 11. Result of Q-learning run on the big maze with epsilon value of 0. The result converges to the lower goal.

optimal value. One interesting behavior I observed was that for both big and small mazes, the agent converges to the goal with the shortest Manhattan distance if the epsilon value is 0. As in Figure 11, although the optimal solution is to converge to the upper right goal, the agent converges to the lower left goal. We can see from this behavior that the agent is prone to be stuck in local minima if not enough exploration is allowed.

The learning rate represents how much of the newly calculated data will influence the Q-value. A learning rate of zero will not update the Q value at all. It is important to set a balance in the learning rate so that the agent learns about the next calculation to a certain amount, but not too much as to make aggressive decisions and make the model unstable. RL-sim implements a decaying learning rate, which means that the learning rate will decrease, or decay, as the number of episodes increase.

As noted above, I used the epsilon-greedy exploration method for Q-learning because it is the most known strategy. Also, it is obviously more effective than the random restart strategy because while it takes random actions in a probability of epsilon in order to explore, it also takes optimal actions. Epsilon of 0.1, pjog of 0.3 and initial learning rate of 0.7 was used for the mazes.

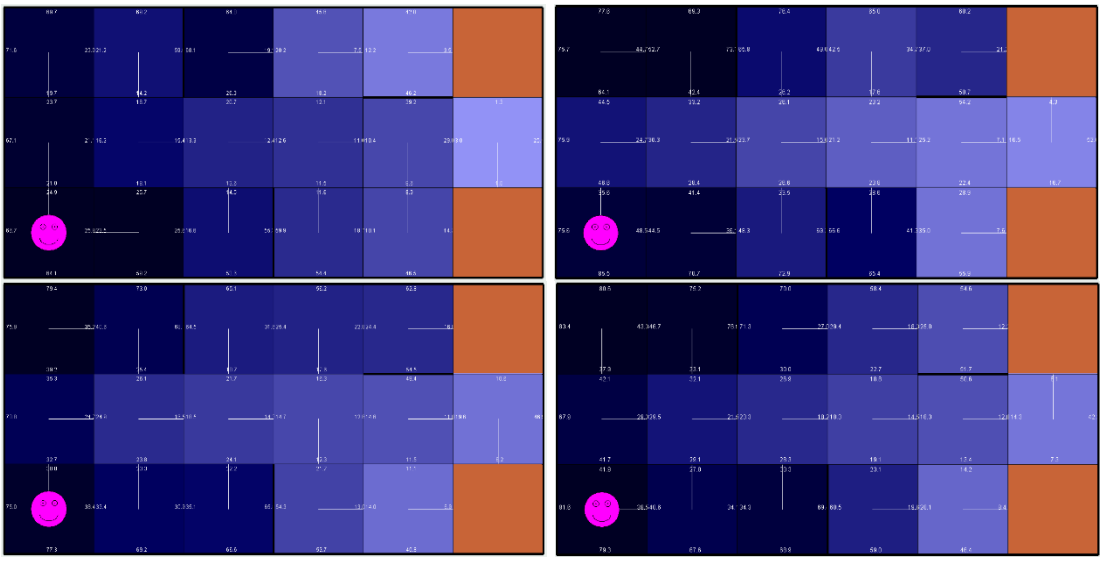

Figure 12. From top left to bottom right: Q-learning run on the small maze with 1000(top left), 5000(top right), 10000(bottom left), and 50000(bottom right) cycles.

Theoretically, the estimated Q function $\hat{Q}(s, a)$ converges to the actual $Q(s, a)$ if each state and actions are visited infinitely often. Q-learning run on the two mazes supported this theory. Figure 12 are the results of Q-learning performed on the small maze. Since Q-learning updates its Q values by exploring the states until it reaches the goal, more episodes lead to more optimal Q value estimates. Obviously, since the agent has no knowledge of the model, it takes much more time than value or policy iteration to converge. However, with enough iterations, Q-learning achieved quite similar results to the preceding algorithms. 1000 iterations on the small maze does not converge to the optimal policies and goals. However, as the number of cycles increase, the policies become more similar to the optimal solution. In fact, Q-learning achieves the same policies and similar Q values to value iteration after 50000 cycles.

The big maze shows similar behavior. Figure 13 shows Q-learning run on the big maze with 1000 cycles, and after 100000 cycles in which the policies matched that of policy and value iteration.

However, there are some points to note. Although 50000 iterations on the small maze and 100000 iterations on the big maze happened to return the same policies as value and policy iteration, performing more iterations often returned different policies for some states. In other words, there was still no convergence after a massive number of cycles. This agrees to the theory that although Q-learning will learn more of the optimal policy as the number of iterations increase, it ultimately needs an infinite number of iterations in order to converge.

Obviously, because Q-learning has no knowledge of the model such as rewards and penalties, it performs poorly compared to value iteration and policy iteration. In fact, Q-learning will have to take almost an infinite number of iterations in order to converge. Nevertheless, I was impressed that Q-learning estimated the optimal policies quite accurately without any knowledge of the model.

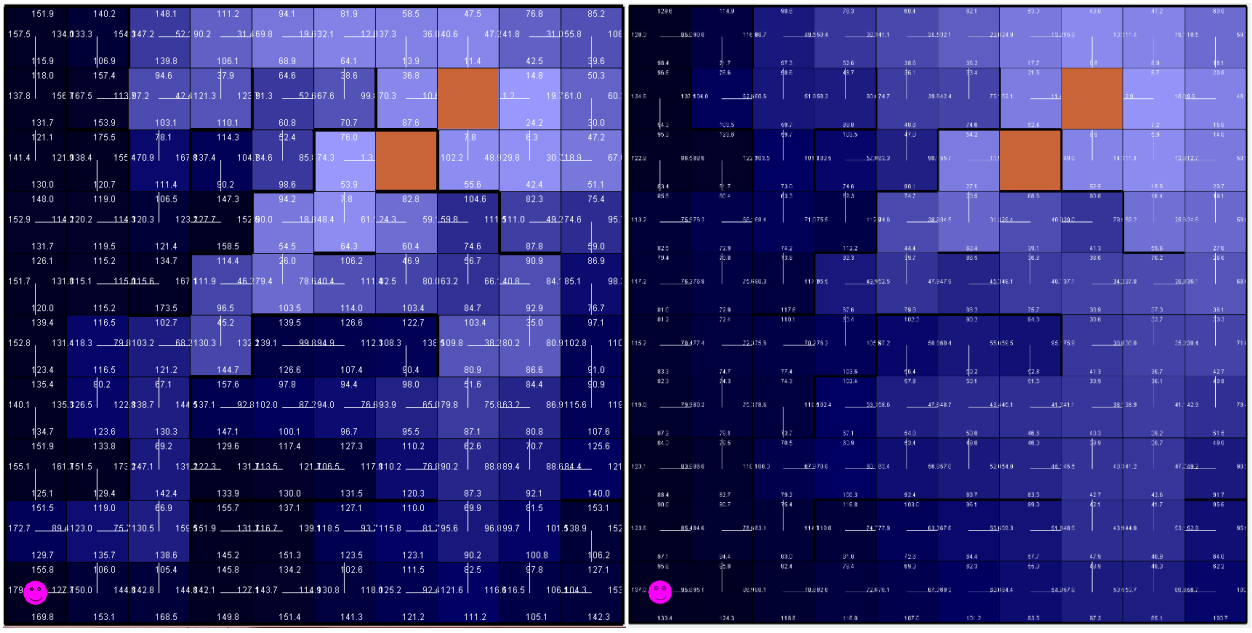

Figure 13. From left to right: Q-learning run on the big maze with 1000(left) and 100000(right) cycles.