CS 4641 MDP and Reinforcement Learning Assignment

Jiahao Luo   GT#903103970

1 Introduction

In this research report, it mainly involves two interesting Markov Decision Process (MDP) problems (mazes). For the purposes of the assignment, these two MDP problems will include one that has a "small" number of states, and the other has a "large" number of states. The whole report will be divided to two parts. First, it will solve both MDP problems by using value iteration as well as policy iteration and analyze the results. After that, the Q-learning algorithm will be implemented to both MDP problems and some aspects about this algorithm will be explored. The whole exploration is based on a reinforcement learning simulator with a nice GUI developed by Rohit Kelkar and Vivek Mehta, http://www.cs.cmu.edu/~awm/rlsim/?

2 MDP Problem Details

The two MDP mazes that I selected for analysis are included in the reinforcement learning simulator package which are "1goal-3×6.maze" and "2goal-10×10.maze". Here are some details about them.

Table 1. MDP problem details

Title	1goal-3×6.maze	2goal-10×10.maze
Dimension ($l \times w$)	3×6	10×10
Number of goal	1	2

3 Basic Rule of the Game

In the entire experiment, there are several elements that are involved. Each cell of the maze will represent a state, the bolded border of the cell represents the wall and the orange color state represents the goal state. Imagine an agent can move up, down, left and right to any adjacent cell through the maze. The agent might hit the wall and will get penalty if it happens. Besides, every step it moves can get a movement cost. Arriving to the goal state is the final target of this game where the agent can be grant 0 reinforcement for reaching there.

Also, in order to considering the noise of the environment, it introduces the noise effect by a parameter PJOG which is between 0 ~ 1. The noise can affect the agent's intended movement. For example, we can assume the agent has n possible direction. And if it takes the up action, then the probability of that action will be 1-PJOG, and the probability of other direction will be (PJOG) / (n-1).

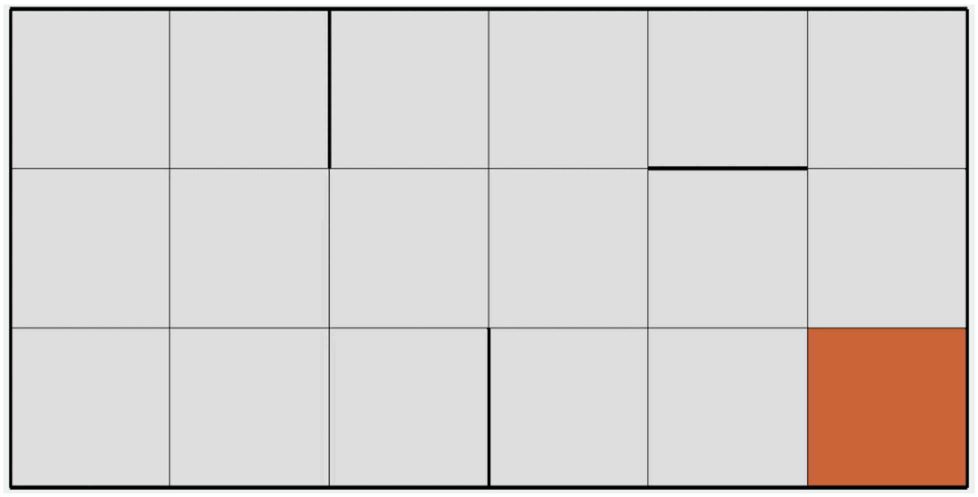
Since there will be only penalty in any action, the optimal strategy is to reach the goal state as soon as possible so that the agent won't get too much positive penalty. And this is the thing that we looking for.

* For the convenience of description, all the graph in the following will obey the (i, j) coordinate which i represents the ith cell along the row, j represents the jth cell along the column. For example, the coordinate of the goal state in Figure 1 is (6, 3).

4 Part I - Solve MDPs with Value/Policy Iteration

4.1 Small Number of States

Figure 1. Small number of states



This figure shows a small number of state (3×6) MDP problem. At first, the precision for checking whether the algorithm converged is set as 0.001, the noise effect PJOG will be set as 30% and the penalty for hitting wall and path cost is 50 units. Initially, the simulator will set the penalty is 0 and the default policy is up for all the states.

Figure 2. Small number of state using value iteration and policy iteration with PJOG=30%

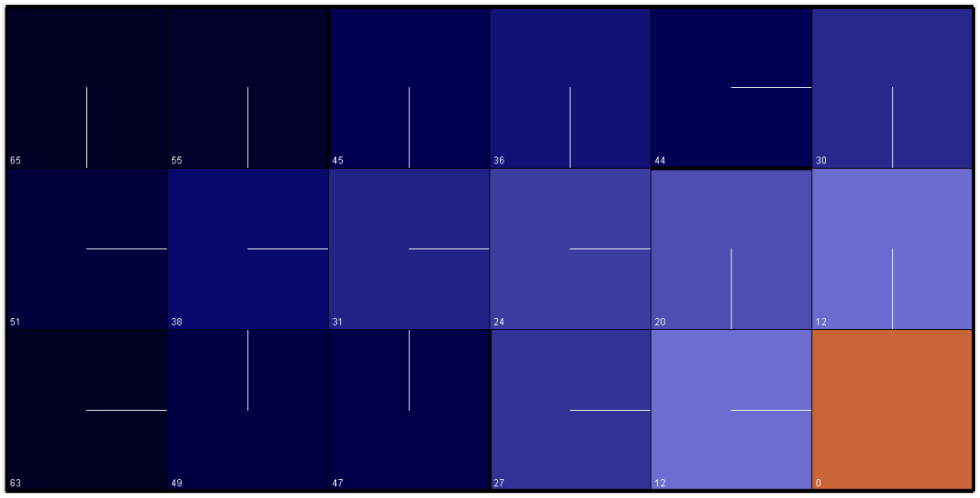
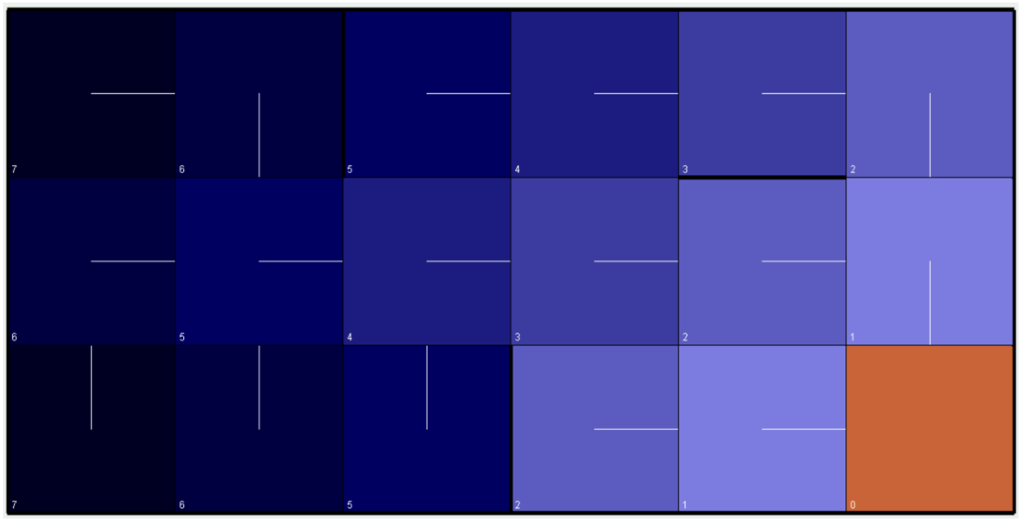


Figure 2 shows the result after 44 iterations of the value iteration algorithm and 4 iterations of policy. As we can see the optimal policy and state values are displayed on the maze. The darker the cell is, the higher the penalty will be. The direction of the white line represents the optimal direction for the agent to take from the center towards adjacent cell. We can imagine these figure as topographic contour, and the water from the top will fall into the valley which is the absorb state and the policy show the flow of the water. Therefore the cell which is closer to the absorb state should have lower penalty. However, there is an exception at the (5, 1) cell which has a higher penalty than the (4, 1) cell or even (2, 2) cell. In my opinion, this happens because we consider the noise effect here. Since there are 30% of the chance that the agent will make a wrong step, when the agent is at (5, 1), it will raise up the chance to hit the wall.

Figure 3. Small number of state using value iteration and policy iteration with PJOG=0%



Therefore in Figure 3, I change the PJOG to 0% as an extreme point to test, the penalty value decreases along the path to the goal state and the highest penalty would be the cell at (1, 3) or (1, 1) with cost equals to 7 which is the minimal step requirement to reach the goal state. Also, there is another interesting point that if we change to another extreme point when PJOG equals to 75%, then that means the every action that the agent makes would be random with each direction 25% chance, but it will cost an extreme high cost before reaching the goal state. See Figure 4.

Figure 4. Small number of state using value iteration and policy iteration with PJOG=75%

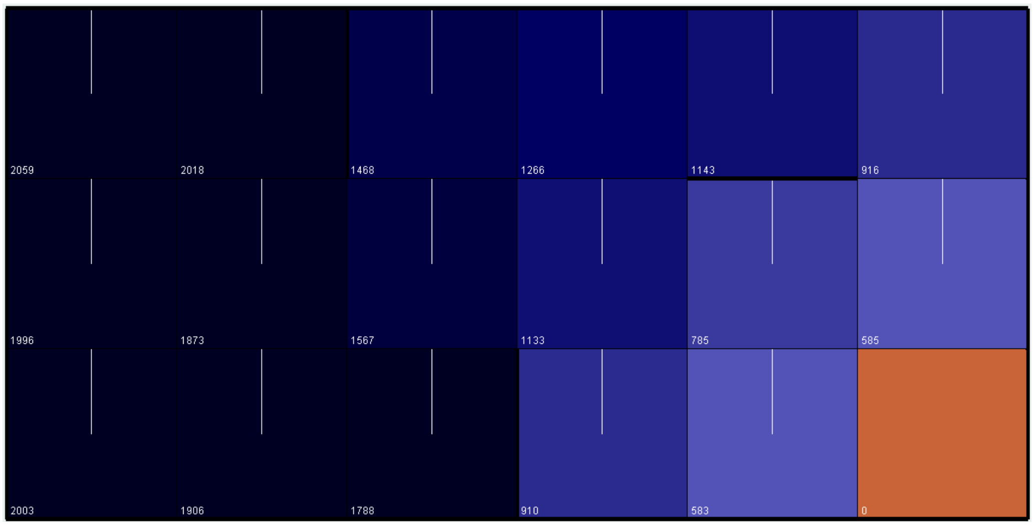


Table 2. Result of Two Iteration Algorithms

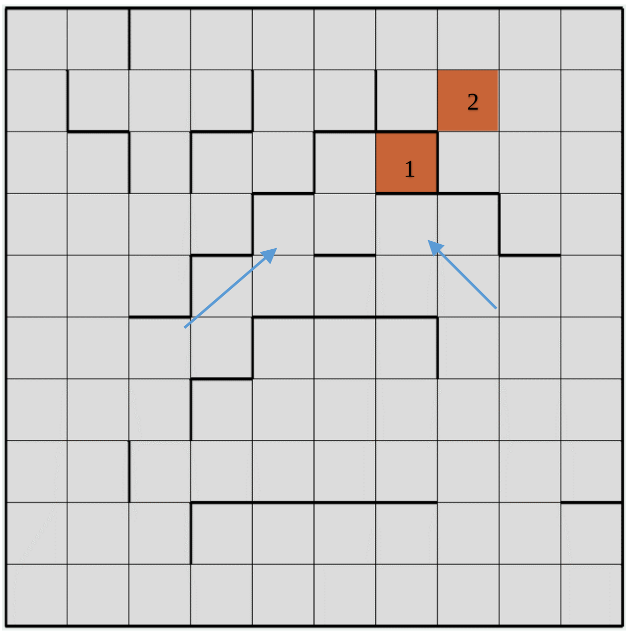
	1goal-3×6.maze		
PJOG	0	0.3	0.75
Precision	0.001	0.001	0.001
Value Limit	5000	5000	5000
Iteration Limit	500	500	500
Policy Iteration	7 steps / 1ms	4 steps / 16ms	2 steps / 11ms
Value Iteration	8 steps / 0ms	44 steps / 4ms	933 steps / 50ms

Comparing to the result from value iteration and policy iteration (Table 2), all the experiments converge to the same answer except PJOG=100%, because the result for policy iteration will exceed the value limit setting which will stop evaluating the current policy. Also we can easily tell that policy iteration takes more time to converge than value iteration in a given maze, however, it takes less number of iteration steps to converge. Each step of policy iteration will take much longer time than value iteration.

The main reason is because the policy evaluation part will cost time to find the exact value of value function until convergence O(s^2). Then the policy is updated from the new utility values. These two steps causes policy iteration to converge over fewer iterations, but takes a longer time to converge especially when the action-to-state ratio is low. Performance of both algorithms depend on the ratio of number of iterations to number of states. In order to improve the performance of policy iteration, given a same size maze, the higher the ratio will be better performance in policy iteration [1].

4.2 Large Number of States

Figure 5. Large number of states



From this MDP problem we can tell the size is larger and the number of goal is two which we can generate an interesting question about the choice of goal state. I am curious which goal state would be more attractive to the agent. The first goal state has only one entry and many walls surrounding it. However, in order to reaching the second goal state, it might need to detour but with 4 entries and less chance to hit the walls. Also, as we can see, if we just consider one goal state, there is a narrow path that direct to the first goal state at (7, 3), also there is another path which is denoted by the arrow on figure 4. Obviously, the narrow path will increase the risk for the agent to hit the wall.

Figure 6. Large number of state using value iteration and policy iteration with PJOG=30%

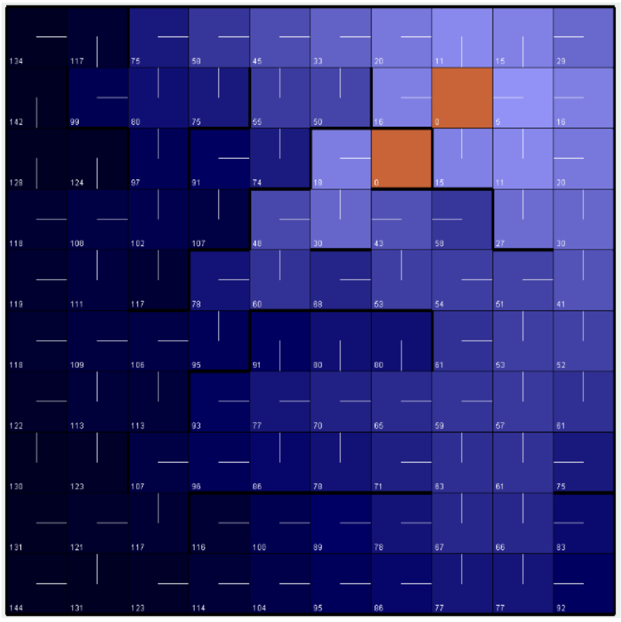
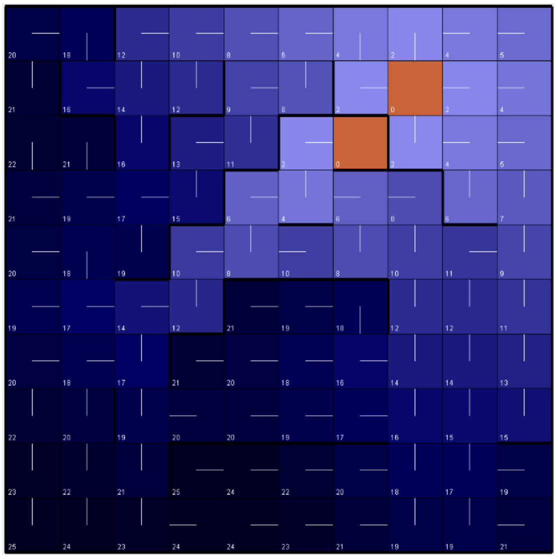


Figure 5 shows the result using both iteration algorithms with the same configuration as the previous experiment (PJOG=30%, precision=0.001, wall penalty=50). Value iteration takes 84 steps and 31ms to converge, while policy iteration takes only 7 steps and 43ms to converge. They both converge to the same answer but policy iteration runs faster than value iteration. According to the darkness of color per cell, the upper corner cells are lighter than the other (lower value). If we assume the agent from the left bottom corner, the policy iteration suggests it to reach the second goal state. The narrow path with too many walls to the first goal state has a relative higher penalty, in the same time the paths to the second goal state looks more attractive to the agent although it may get cost for longer distance.

One more interesting point we can notice is state (8, 5) at the maze. We can suppose that the agent just happens to be in that state which is relatively close to each goal state with a five-step shortest path. However, the optimal action for that state is action to right. That is very interesting because the agent didn’t choose the path to goal state 1, on the contrary, it bypasses the walls and heads to goal state 2.

Since there are more walls and states in the large number of state maze, I have a curiosity to compare the different penalty ways, hitting walls and walking cost. In order to test this, in the next experiment, I set value for wall penalty as 1 which is same as the transition of state, but the agent can’t cross the walls. The following figure is the result.

Figure 7. Large number of state using value iteration and policy iteration with penalty=1



This results will be shown for what happens when the penalty for hitting a wall is 1, while also keeping the probability of any chosen action to be 70% successful, and 10% chance for the agent goes in any other direction. It takes 52 steps for value iteration and 5 steps for policy iteration to converge. By comparing figure 6 and figure 7, the layout of the light color cells are different, the cell's color between those two narrow paths have a lower state value than before. Since the huge decrease of wall penalty (wall penalty equals to transition penalty), the agent would prefer to get into the narrow path which is also the closest path to goal state 1. Although it might be easier to hit the walls, but it is worth to try instead of getting transition penalty along the longer path toward goal state 2.

In the previous example, we have been analyze the state at the left bottom corner and the cross road state at (8, 5). The adjacent states' optimal action around them consider a lot the effect of the wall, so the paths have twists and turns. However, when the wall penalty drops down to 1, the direction of states are pretty clear, they all point to the right upper corner goal state.

Figure 8. Large number of state using value iteration and policy iteration with penalty= -1

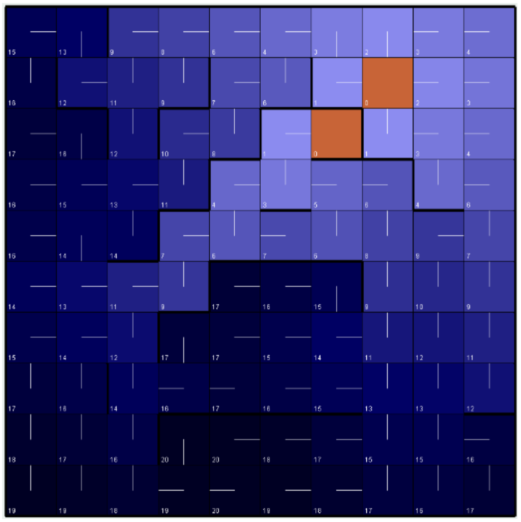


Figure 8 shows the result of both algorithms with wall penalty -1. Both algorithms converge to the same result. The details would be recorded in table 3. When penalty is -1, agent got a one unit deduct from its penalty every time it hits the walls. Thus as we can see in Figure 8, to optimize the final result, it is wise to utilize the wall as a compensation for each transition penalty. If it moves forward one steps and hits the walls in the next step, then the cost would be neutralized. Therefore the optimal policy would be more likely to go along the wall in every action.

As to the comparison between small number of states and large number of states, actually when the number of state is larger, that means the agent would have a larger maze to explore so that the iteration time and steps would be larger than the small number of states. In fact, larger number of states increase the effect of transition penalty.

Table 3. Result of Two Iteration Algorithms

	2goal-10×10.maze		
PJOG	0.3	0.3	0.3
Wall Penalty	50	1	-1
Transition Penalty	1	1	1
Policy Iteration	7 steps / 43ms	5 steps / 59ms	5 steps / 50ms
Value Iteration	84 steps / 31ms	52 steps / 22ms	85 steps / 28ms

5 Part II - Solve MDPs with Q-learning Algorithm

In part II, I will use my favorite reinforcement learning algorithm, Q-learning to solve the two MDPs and compare to the value and policy iteration.

There are two parameters that are pretty important for Q-learning algorithm. The first one is learning rate (or decaying learning rate) which is between 0 and 1. Learning rate denotes the probability that the agent will take an action into a lower Q-value state in order to explore the map, and when the agent has run for many episodes, it is learning rate will decay instead of keep exploring. The second one is episode (or cycles). Each episode equals to a training session. During a training session, the agent will explore the environment and receive the penalty until it reaches the goal state.

The exploration policy that I choose is ε-greedy exploration policy. It will select the best action with probability 1- ε and choose random action with probability ε.

5.1 Small Number of States

Figure 9. Small number of state using Q-learning algorithm with PJOG=30%

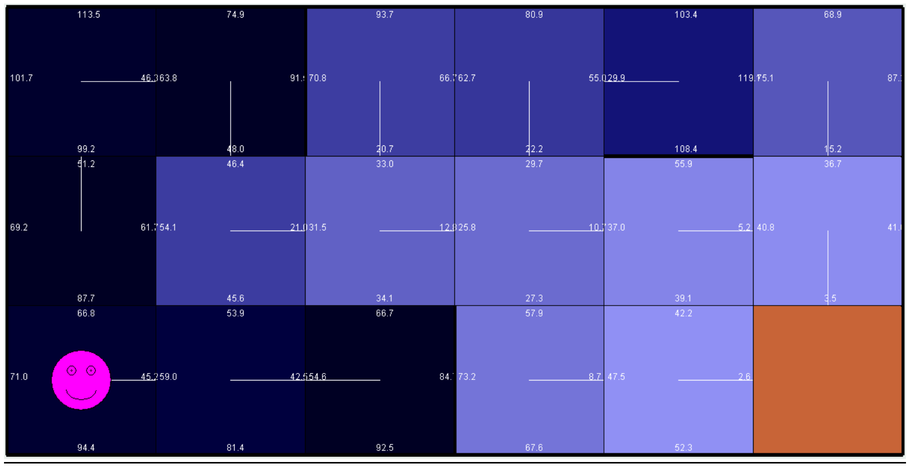


Figure 9 shows the result of small number of states using Q-learning algorithm. In this experiment, the PJOG will be set as 0.3 by default, epsilon=0.1 (For the epsilon greedy policy for better exploration.), precision=0.001, learning rate=0.7 with "Decaying learning rate" checked and 1000 cycles.

After running the Q-learning algorithm, the most obvious drawback for it is time consumption. In comparison to the cases above where they knew the model, rewards and so on, Q-learning is much slower than both value iteration and policy iteration. It took more than 10 minutes for it to converge. The reason for this is that Q-learning knows nothing about the model, it needs to spend time on searching good policies and values for each state. Moreover, Q-learning calculates the Q-value for each state-action pair which is different from value and policy iteration. They just display the value and optimal policy for each related state.

Comparing to figure 2 and figure 9, Q-learning algorithm calculates all the state-action pair for each state, and we can tell from the figures that more than 60% of the optimal policy is similar. Also the penalty value is much higher than value and policy iteration, because the Q-learning is keep exploring and even get into the lower Q value. The difference may cause by the unknown of the model. By default, I set the episode cycle as 500 times. And result above is based on those 500 times learning. The purpose of learning is to reinforcement the "brain" of the agent. Thus, episode is a very import parameter for Q-learning. The more episode it runs, the more optimal the result will be.

Figure 10. Small number of state using Q-learning algorithm with PJOG=0%

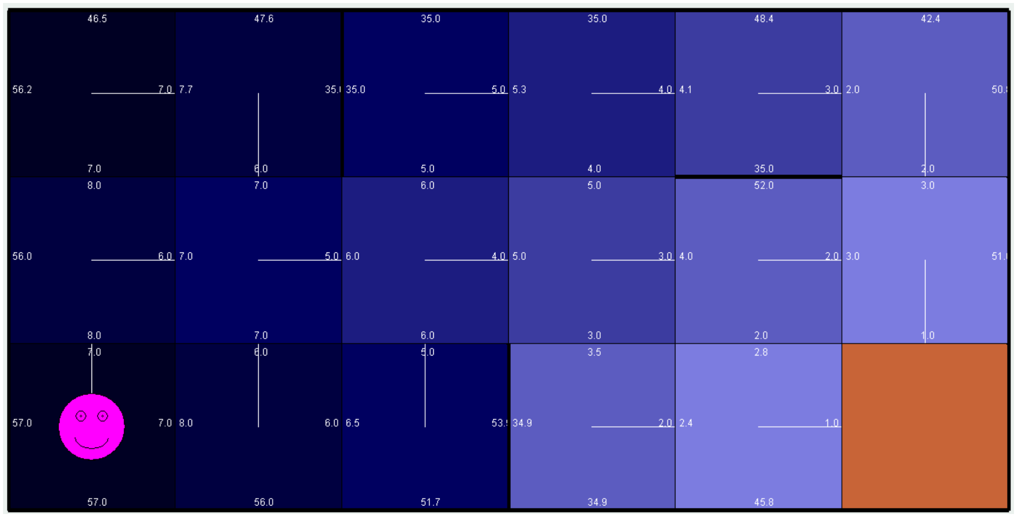


Figure 10 shows a much clearer path for the agent, as well as precise numbers which is similar to value and policy iteration. In fact, Q-learning achieved the exact same policies and values as policy and value iteration did. This shows that Q-learning can achieve very accurate results when the number of cycles (500) are sufficient for a certain scenario.

As a model-free learning algorithm, the disadvantage to Q-learning is that there is no domain knowledge of the transition model and all the states' costs. The agent will learn policy without learning the model. Thus, the agent will take an action to explore the state and assign a value of state-action pair Q (s, a).

However, there is some advantage to use Q-learning algorithm. First, Q-learning algorithm is simple and easy to implement. Second, since Q-learning is a model-free algorithm, so it doesn't need to search the whole map before it runs. Thus if it finds a new state, it can directly add it to the Q matrix. Thus, Q-learning would be useful when the transition model is hidden.

5.2 Large Number of States

Figure 11. Large number of state using Q-learning algorithm with PJOG=30%

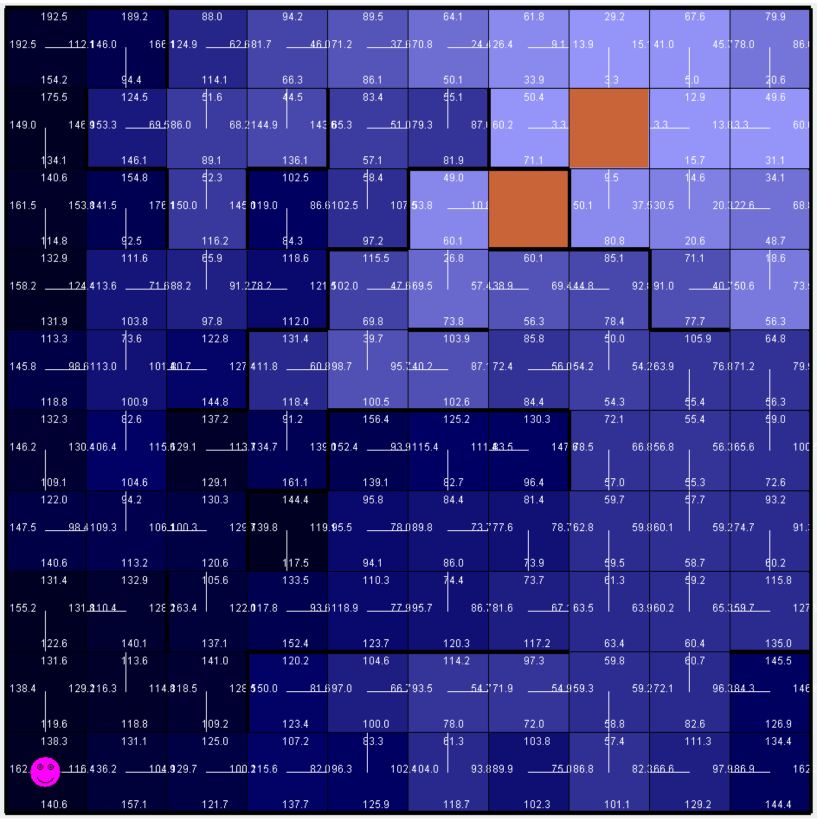


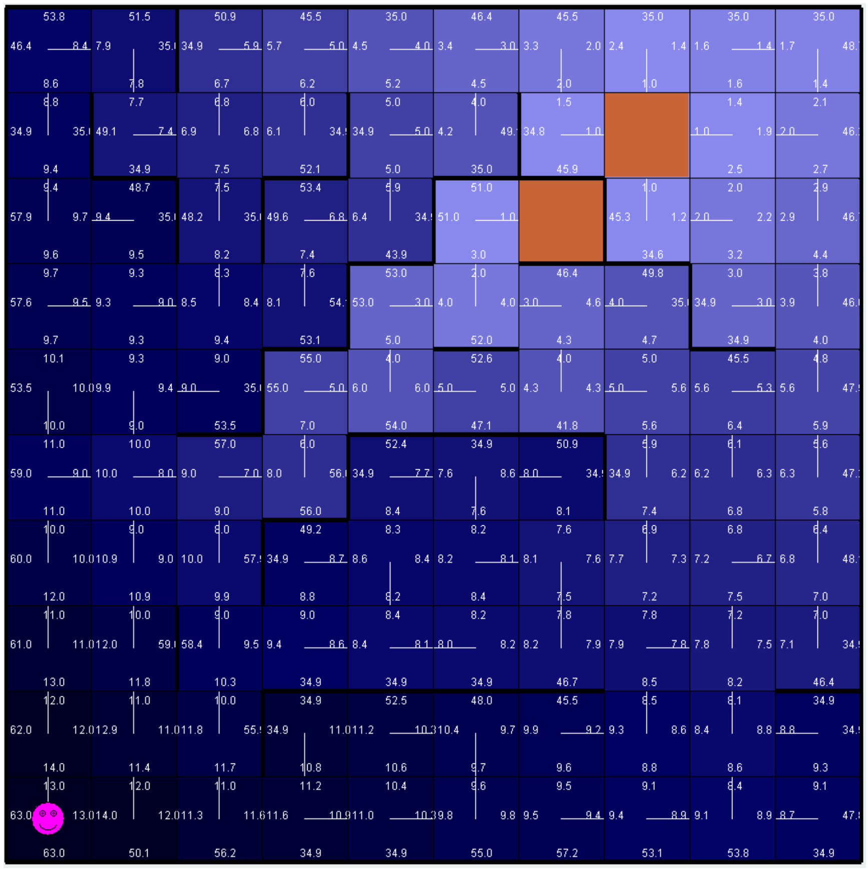
Figure 11 shows the result of small number of states using Q-learning algorithm. In this experiment, the PJOG will be set as 0.3 by default, epsilon=0.1 (For the epsilon greedy policy for better exploration.), precision=0.001, learning rate=0.7 with “Decaying learning rate” checked and 1000 cycles.

Obviously, the major difference between Q-learning on small number of states and large number of states is the time consumption. The larger number of states takes more time for the agent to explore in a larger states in each cycle, thus each cycle takes longer to complete. However, if the size of maze is much bigger than the larger one, then the time consumption would be a serious problem.

As to the left bottom starting point and state at (8, 5) mentioned before, those two states have some interesting results showing up. At first, for the starter point, it is some kind of weird for the agent move right. Because this direction is not towards to both goal state, besides this direction will lead it to a path which is surrounded by walls. Also at state (8, 5), the optimal policy for the agent is downward which doesn’t make sense at all. Because no matter what direction that the agent chooses, the overall direction for it to choose is towards to goal state as accurate as possible.

I think the reason for these interesting things happening. It is because the decision that Q-learning made is based on what it learned through the limited execution. In fact, the number of execution cycles would be as much as possible. However, the Q-learning needs to comprise to the time consumption problem such that the results appear to be less accurate than the policy/value iteration results since the values for each state are drastically different. Therefore, it is very important to make a tradeoff between accuracy and number of cycles.

Figure 12. Large number of state using Q-learning algorithm with PJOG=0%



In figure 12, the optimal policy generated by Q-learning is much more logical this times comparing to the previous experiments. Both starter state and state at (8, 5) make sense for their policy. In fact, Q-learning achieved the exact same policies and values as policy and value iteration did. This shows that Q-learning can achieve very accurate results when the number of cycles (500) are sufficient for a certain scenario.