Reinforcement Learning Analysis

Abstract

This paper will examine two Markov Decision Processes (MDPs) to explore two grids created through RL_sim. The first MDP will explore a small state space, while the second will explore a maze with a large state space. Performance analysis of the two MDPs will be provided to study two reinforcement learning (RL) algorithms: Value Iteration and Policy Iteration. Furthermore, we will use the another RL algorithm, Q-learning, to see how the performance compares to the first two with respect to runtime and iterations. Lastly, we will examine the optimal policy for each maze using these three RL algorithms.

Dataset:

Figure 1. Small Maze (0_small.maze), 10x10, 100 States, Wall Penalty = 50, 90

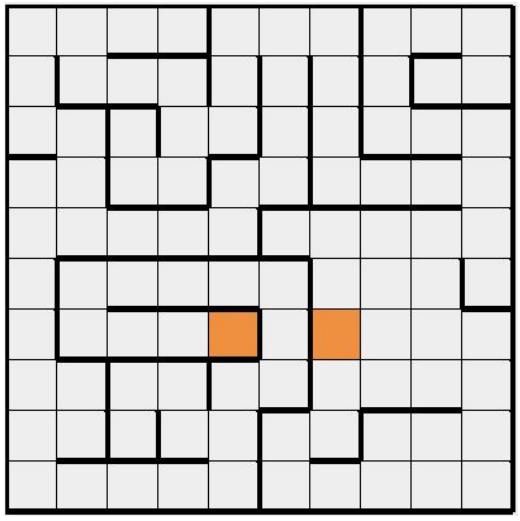
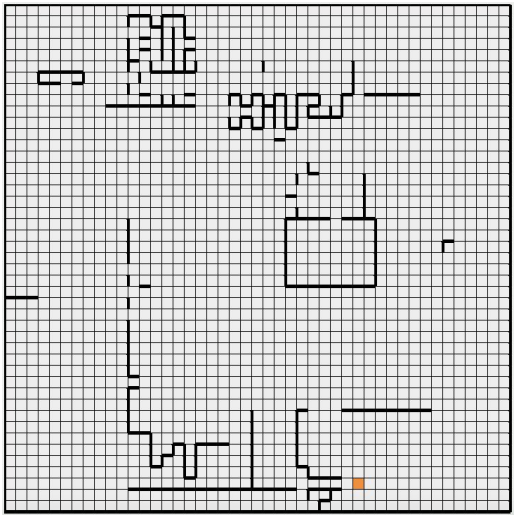


Figure 2. Big Maze, 45x45, 2025 States, Wall Penalty = 50



MDP's create a policy, or a sequence of actions, to navigate the stochastic environment an agent is placed in. They use the assumption that the next state an agent depends on the current state. Using this assumption, MDP's use trial and error, and a reward function to find the best action to take at every state. The best action is one that gives the best utility, which informs the agent how much reward the agent can receive for being in a particular state with respect to neighboring states. Essentially, utility is a heuristic for the agent to navigate the search space.

MDPs are used in this paper to navigate grid spaces to reach a goal state. Two mazes were created using RL-sim, a simulator to run several reinforcement learning algorithms sponsored by Carnegie Mellon. The Small Maze, shown in figure 1, has a smaller state space of 100. The goal of agent placed in the maze is to reach the orange squares which will terminate the game. The agent has a -1 reward for staying alive, and a -50 reward for colliding with the walls. An identical maze was made to have a wall penalty of 90 to see how the agent performs with

higher risk for exploration vs. exploitation since there are more walls. The small maze has multiple goals in different regions. One is closer with more walls surrounding the goal state. The further goal harder to get to with a longer path, but is surrounded by less walls. It will be interesting to see which goal the agents prefer based on conditions like wall penalty and action stochasticity.

To further understand MDPs, the Big Maze shown in figure 2 was created with a state space of 2025 to see how MDPs behave in larger search areas. This maze will show the scalability of RL-algorithms as they behave in the real world where environments are large. Similar to the small maze, the big maze also has multiple goals, with one goal surrounded by more walls. The agent has a -1 reward for staying alive and a -50 reward for colliding with walls. It will be interesting to see how the agent interacts with these conditions using the different learning algorithms and how these results compare to the smaller maze.

Throughout the experiment, the agent will have the PJOG parameter adjusted to add more action uncertainty. PJOG is essentially the probability in which the agent does not perform the action it intends. We will use the baseline PJOG of 0.3 to compare the MDPs amongst each other throughout the experiments. This value was chosen to make the agent actions more stochastic and relate to real-world applications of MDPs where robots do not have perfect sensors. It will be interesting to see how the agent makes rational choices under uncertainty using the three reinforcement learning algorithms.

Link to RL_sim software: http://www.cs.cmu.edu/~awm/rlsim/

Value Iteration:

The first RL algorithm we will explore is value iteration. value iteration works by placing the agent in different states and estimating the ‘goodness’ or value of being in that particular state to determine the best policy. Since the algorithm does not know the correct utility value of that state, it uses the immediate reward of the current state, and factors in the reward of neighboring states to influence the agent’s actions at each iteration. The algorithm stops when every state’s value converges, or when the changes are smaller than a threshold. The ‘optimal’ policy adopted by value iteration is to take the path that leads to best values. The full algorithm implemented by RL_sim is found in Appendix A.

Figure 3 shows the effect of increasing the agents PJOG in the small and big maze. As the actions become more nondeterministic, the number of steps taken till convergence increases. This is expected since increasing stochasticity makes it harder for the algorithm to find a solution and converge on a policy. Moreover, the larger maze took more steps to converge in comparison to the smaller maze as there was a larger state space to explore. It is also significant to note that the time taken for the large map was a lot larger at 125,782 ms at PJOG = 0.8 and only 1751 ms for the smaller grid. In general, value iteration was a pretty slow algorithm in both cases when compared to policy iteration which we will discuss further in the paper. The raw data for these experiments can be found the RLSimulation.xlsx file in the main folder. It was surprising to see

that the steps taken to convergence significantly drop for PJOG values of 0.9 and 1. It seems that taking completely random actions actually made the algorithm converge with less steps. However, the actual policy the agent adapts is not very optimal, suggesting that the agent run into the wall at several states.

Figure 3. Small & Big Maze, PJOG v Steps to Converge Graphs

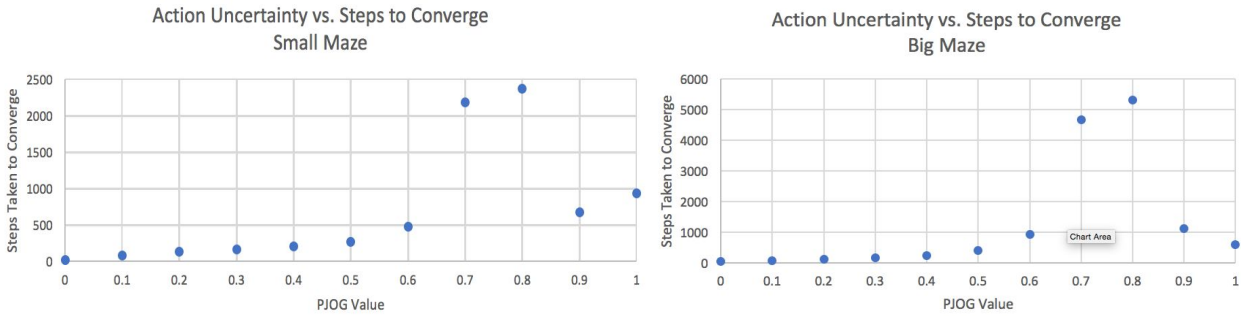
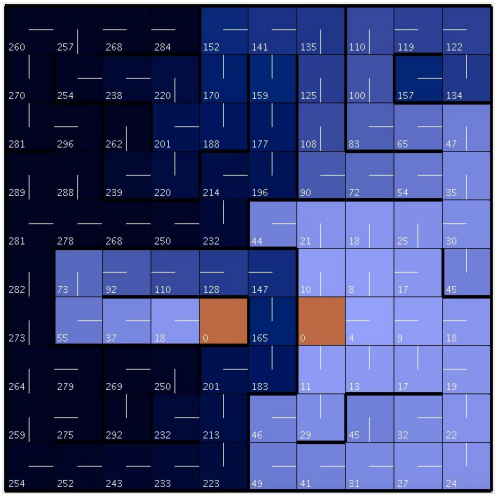


Figure 4 below shows the results of running value iteration on the small maze with wall penalties of 50 and 90. The value of PJOG was set to 0.3 to solely study the effects of increasing wall penalty. As seen below, there was no significant difference in the policy chosen for each state in the two grids. However, the estimated utility of each state is a lot higher in maze with a wall penalty of 90. This is expected as the non-deterministic nature coupled with higher penalty for collisions makes the agent behave a lot safer and teaches it to avoid areas where there is higher chance of colliding with walls. It seems that the orange goal on the left side of the maze was less prefered when the wall penalty was higher for this reason. Moreover, it was interesting to see that the smaller wall penalty resulted in a smaller number of iterations and larger time. This could be due to the less explorative behavior adapted by the agent in the riskier environment where wall penalties were higher.

Figure 4. Small Maze Value Iteration, PJOG = 0.3

Wall Penalty = 50, 173 iterations, 101ms



Wall Penalty = 90, 83 iterations, 45ms

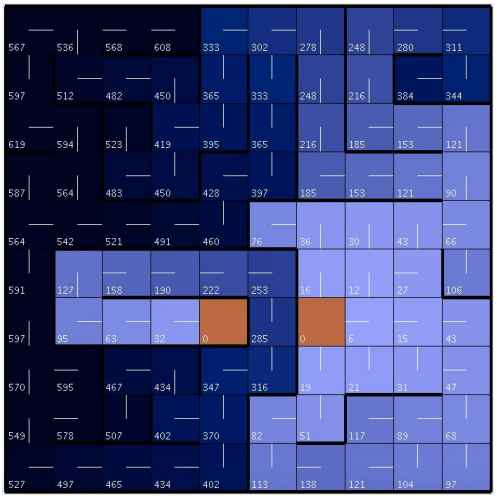


Figure 5 in the next page shows the results of running value iteration on the larger maze. Despite having a larger search space, the maze was able to converge with 163 steps, which was

10 less than the smaller maze. This is probably due to the fact that there are less walls in the larger maze so the agent has more chances for optimal steps at each iteration. It is significant to note, however, that the larger maze took longer with 3843 ms. There was also a clear preference to the orange goal with less walls in this scenario as most of the prefered actions at each square lead to this goal. In both cases, the orange goals with less walls were preferred since this was the safer option. When action certainty was guaranteed (PJOG = 0), both orange goals were equally preferred. This is seen in figure 6. Clearly, as risk increases with action stochasticity, the agent takes the safer route, even if it means traveling the longer distance.

Figure 5. Big Maze Value Iteration, PJOG = 0.3, Wall Penalty = 50, 163 iterations, 3843 ms

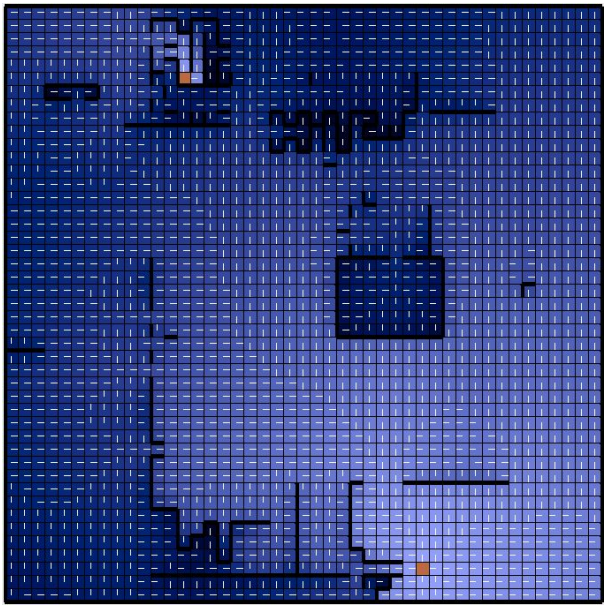
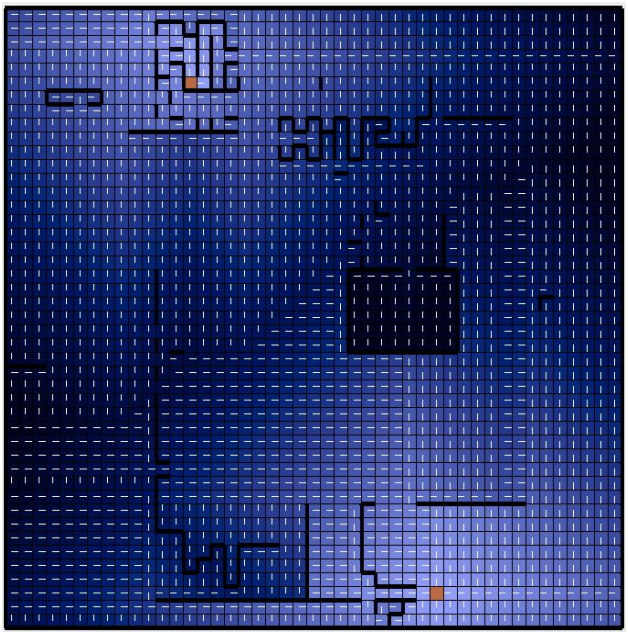


Figure 6. Big Maze Value Iteration, PJOG = 0.0, Wall Penalty = 50, 48 iterations, 1087 ms



Policy Iteration:

Instead of estimating the utility of being in a state to decide on the optimal policy, this RL-algorithm finds the policy directly. It does so by iterating through random policies, and computes the utility of the state from the sequence of actions at each step. The full algorithm implemented by RL_Sim is found in Appendix B. Policy iterations stop when the policy no longer changes. Thus, the algorithm converges with less steps and less time. This is shown in Table 1 where the number of steps taken by the algorithm is shown for each of the three grids. These values are a lot smaller than the number of iterations taken with value iteration from Figure 1. Moreover, the first maze with a smaller wall penalty converged a lot faster as the randomization increased. As expected, the larger maze took more iterations to converge than the small maze, but in general, it took fewer iterations than value iteration. Overall, policy iteration took more time to converge than value iteration. This was probably because the algorithm needed state values to converge and policy to update at each step.

Table 1. PJOG Value and Steps Taken with Policy Iteration on the 3 Mazes

PJOG Value	Steps, Small Maze, 50	Steps, Small Maze, 90	Steps, Big Maze, 50
0	13	18	45
0.1	16	20	54
0.2	16	21	50
0.3	17	6	25
0.4	25	15	33
0.5	4	8	40
0.6	5	7	28
0.7	2	5	24
0.8	2	3	49
0.9	6	4	86
1	12	6	109

Policy iteration was run on the two small mazes with a PJOG value of 0.3. The results are shown in figure 7 on the next page. It was interesting to see that the algorithm needed less iterations to converge when the wall penalty was higher. This could be due to the fact that with value iteration, the algorithm would continue to recalculate utility even when the actions did not change in between the iterations. Policy iterations stops when the actions at each state stop

changing. This allowed the algorithm to converge with less iterations. One interesting thing to see in the right grid is that both orange goals are equally preferred has more light blue tiles indicating that the algorithm was more willing to take risk with more iterations.

Figure 7. Small Maze Policy Iteration, PJOG = 0.3

Wall Penalty = 50, 17 iterations, 46 ms Wall Penalty = 90, 6 iterations, 20ms

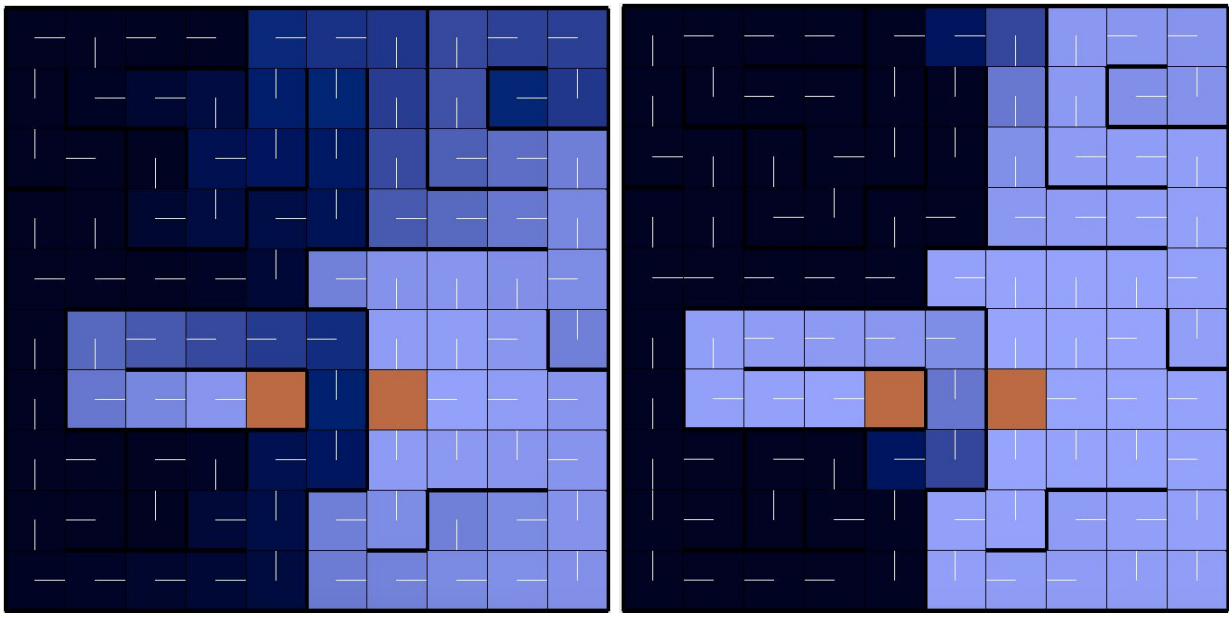


Figure 8. Big Maze Policy Iteration, PJOG = 0.3, Wall Penalty = 50, 25 iterations, 37087 ms

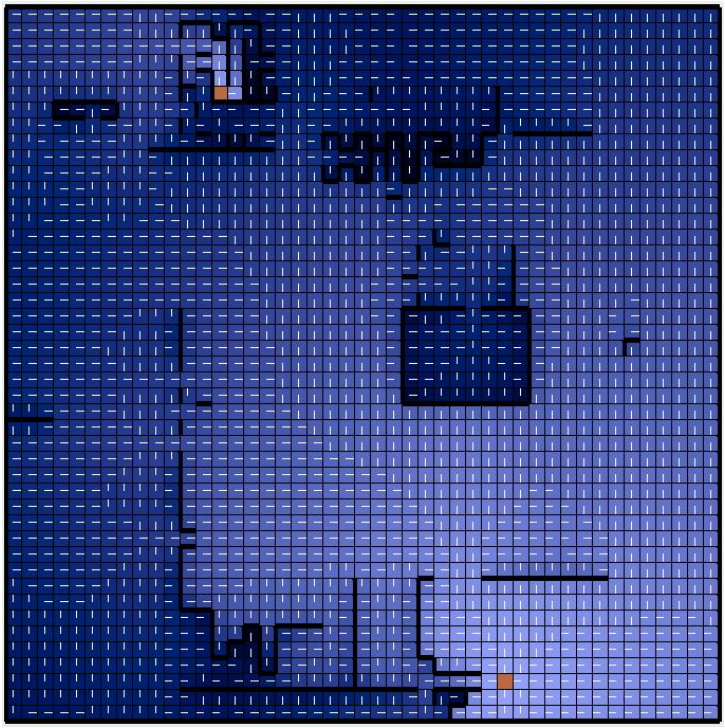


Figure 8 above shows the results of running policy iteration on the larger map. We again see a drop in the number of iterations needed for convergence when compared to value iteration. The

number of steps was halved in policy iteration. This suggests that for larger state spaces, it may be worthwhile to use policy iterations since the algorithm was able to find a policy that was similar to value iteration. Still, we cannot conclude if the policy found by the algorithm is the absolute optimal policy, but we can say that the agent learns to make reasonable choices given the search space. Overall, there was not too much of a difference in the actual policies reported by value iteration and policy iteration. This is probably due to the fact that both algorithms use domain knowledge of what they have learned at each step to find a solution.

Q-learning:

We will now look at the reinforcement learning algorithm of Q-learning. Q-learning uses priori knowledge to explore the space as opposed to domain knowledge used in value and policy iteration. The agent knows the state it is in and the actions it can perform. The algorithm assigns each state a Q-value that estimates the value of being in that state based on the reward the agent receives at each visit. Q-learning is a form of online learning, where the agent takes a lot of risk initially and explores the search states that it would not take if it solely were based on old policies from previous iterations. This risk factor, epsilon, is decayed with some schedule so that the agent eventually converges to choosing the policy with better Q-values. There is also a learning rate that impacts how much the agent learns at past iterations effects new updates.

First, I examined how learning rate affects the learning agent by setting it to 0.1, 0.3, 0.7, and 1.0. Higher learning rates took a lot more time to converge. This suggests that future rewards and new information may not always indicate better policy. It might actually be better to use the immediate reward as an indicator for estimating the Q-value. A similar experiment was done setting by learning rate to 0.5 and varying epsilon. It was found that higher epsilon values make the algorithm converge a lot faster. This is probably due to the fact that the agent is able to explore more areas by taking more risks, so it is able to find better policies faster. The actual algorithm implemented by RL_Sim can be found in Appendix C.

To compare Q-learning to the other learning algorithms, PJOG was set to 0.3. We use a decaying learning rate of 0.2 and an epsilon value of 0.4. We vary the number of cycles from 1000, 5000, and 10,000 to see how the algorithm performs in the small maze with a wall penalty of 50 in Figure 9. As there are more cycles, the algorithm improves in finding the better actions that lead to rewards. This demonstrates the trade-off in Q-learning. Initially, the algorithm is able to explore more areas at the cost of a more expensive learning time since the agent will fail often. But as time goes on, the algorithm is able to learn more and improve its overall performance. The time spent exploring is costly, and the algorithm may get stuck more often when choosing actions. Q-learning is not as useful in this scenario and performed slower with worse policies when compared to value and policy iteration.

Figure 9. Q-Learning on Small Maze, 1000 Cycles (L), 5000 Cycles (R), 10000 Cycles (Bottom)

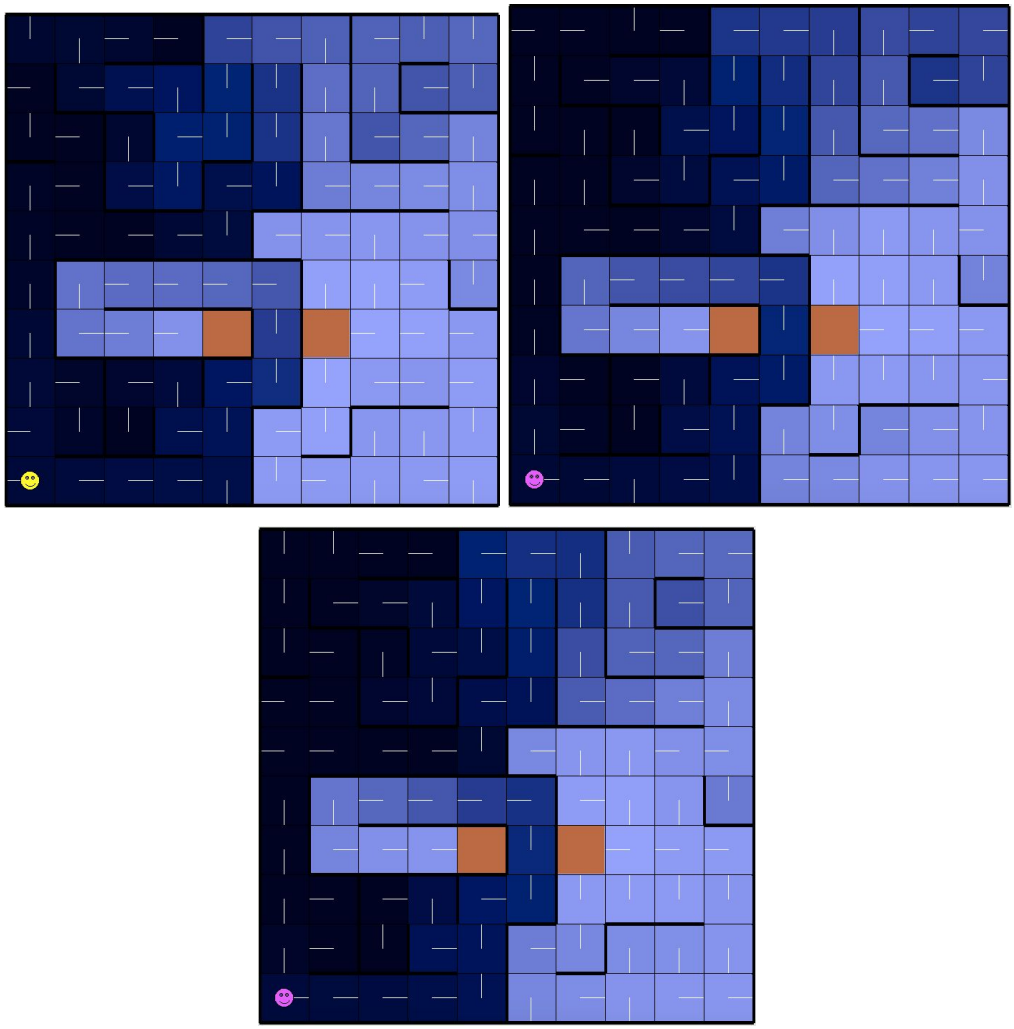
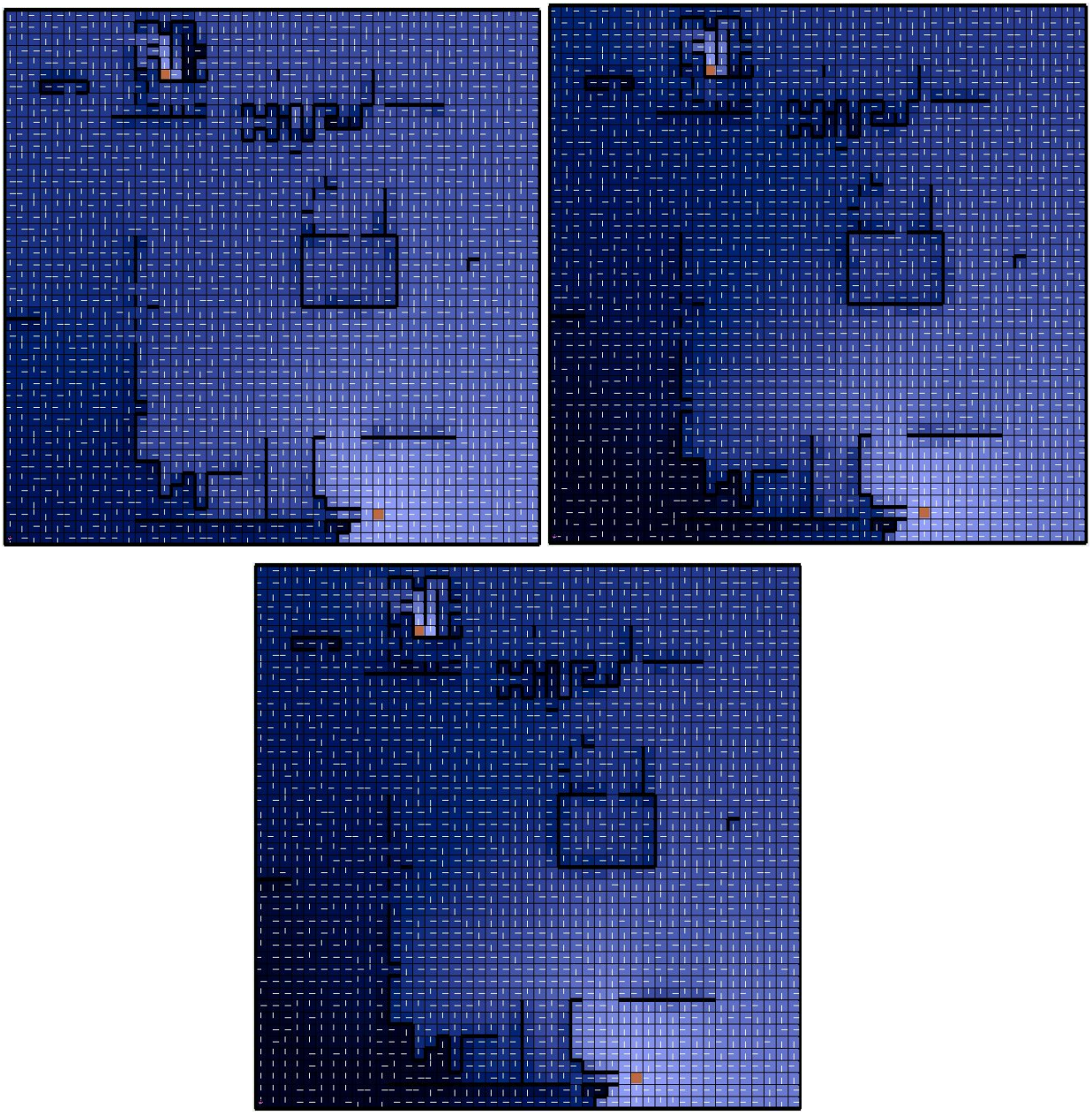


Figure 10 on the next page shows the same experiments run on the larger grid. The larger maze converged slower than the smaller maze due to the larger search space. But overall, the time taken was not too significant but was longer since the algorithm preferred to explore to some extent. There is more variation in the policy chosen in the maps created by Q-learning. In certain areas, the agent wants to collide with the wall even though this is not the optimal solution. Clearly, Q-learning algorithm converged on subpar solutions given the same amount of stochastic actions. This shortcoming seems to indicate that domain knowledge may be the best heuristic if you can figure out the transition and reward functions of the given environment.

Figure 10. Q-Learning on Big Maze, 1000 Cycles (L), 5000 Cycles (R), 10000 Cycles (Bottom)



Conclusion:

There are a few clear distinctions to be made with the three reinforcement learning algorithms. First, value iteration takes a long time to converge. It was able to find good solutions but took longer since the algorithm focused on converging on the value of state even though the actual recommended action to take on the state might not be changing.

This brings us to the next fact that policy iteration was able to find relatively optimal solutions with fewer iterations. This comes with the drawback that it may actually require more time to find these solutions as the policy is recalculated at each step. This number increases as the state space grows. Clearly, both algorithms have a trade-off that needs to considered for the

MDP problem at hand. If the search space is small, it might be alright to choose policy iteration. But for larger state spaces, value iteration is the way to go.

There is a stark limitation with value iteration and policy iteration that Q-learning does not exhibit. Value and policy iteration depend on earlier domain knowledge to converge with each iteration. So if the algorithm learns earlier on that the states are risky, it will prefer not to take this path. But if the reward at the end of the riskier path were high, these algorithms would never know since they learn early on that there is a high chance to fail.

Q-learning is different in that the agent is taught to take more risk initially which it decreases by some schedule. This makes the agent wander the space more, allowing it to consider more options when it find a policy. Depending on the risk, this may not be the best algorithm to execute in the real world where learning errors can actually break robots , which can be very expensive. But for a simple maze or AI program in a virtual reality where learning costs are low, it may be worth to invest in this higher risk learning time to find the best policy in the end. It also converges on similar solutions to value iteration and policy iteration given that it does not use any domain knowledge. This makes Q-learning a lot more applicable in a real setting.

Appendices

A. Value Iteration Algorithm

initialise $V(s)$ arbitararily
loop until policy good enough
  loop for $s \in S$
    $V_{t+1}(s) = \min_{a \in A}\{1 + \sum_{s'=S}(P(s'|s,a) \cdot x_{ss'})\}$
    $\pi_{t+1}(s) = \arg\min_{a \in A}\{\sum_{s'=S}(P(s'|s,a) \cdot x_{ss'})\}$
    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

  end loop
end loop

B. Policy Iteration Algorithm

Policy Iteration

initialise $\pi(s)$ arbitararily
loop until policy good enough
  loop until $V(s)$ has converged
    loop for $s \in S$
      $V_{t+1}(s) = Path\_Cost + \sum_{s' \in S}(P(s'|s,\pi(s)) \cdot x_{ss'})$
    end loop
  end loop
  loop for $s \in S$
    $\pi_{t+1}(s) = \arg\min_{a \in A}\{\sum_{s' \in S}(P(s'|s,a) \cdot x_{ss'})\}$
  end loop
  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
end loop

C. Q-Learning Algorithm

Q Learning

Initialize $Q(s,a)$ *arbitrarily*
Repeat for each episode
Initialize s
  Loop until s is goal
  Choose best action a *from* $Actions(s)$
  Perform a *and observe reward* r *and next state* s'

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[x + \min_{a'} Q(s',a')]$$

  $s = s'$
  End Loop

Where, x = 1, *if transition was safe*
  = *penalty, if transition is unsafe*
α = *learning rate*