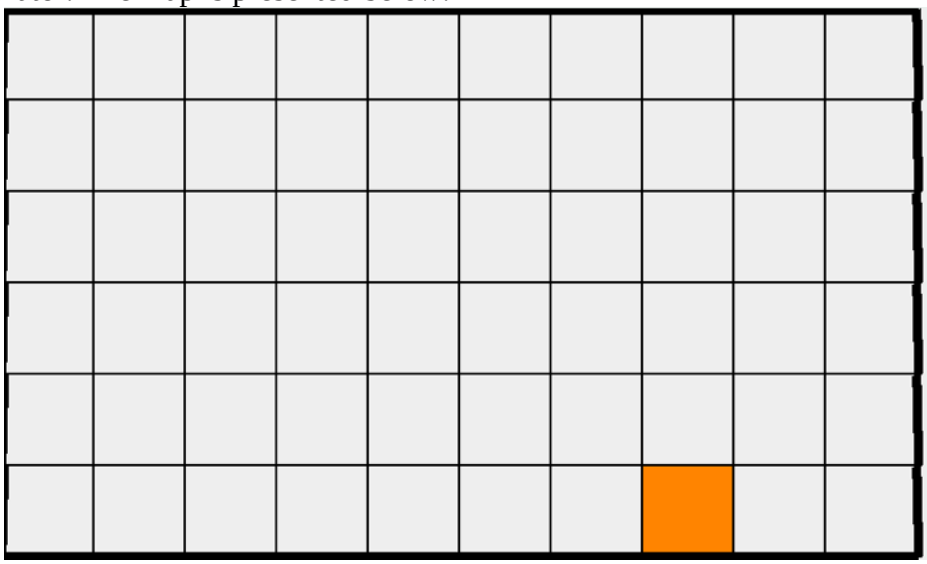
Alan Li
Project 4
CS 4641

Reinforcement Learning Analysis

The MDPs

Electric Fence Gridworld

The "small" state space MDP I made a custom grid world using a reinforcement learning simulator. The map is presented below:

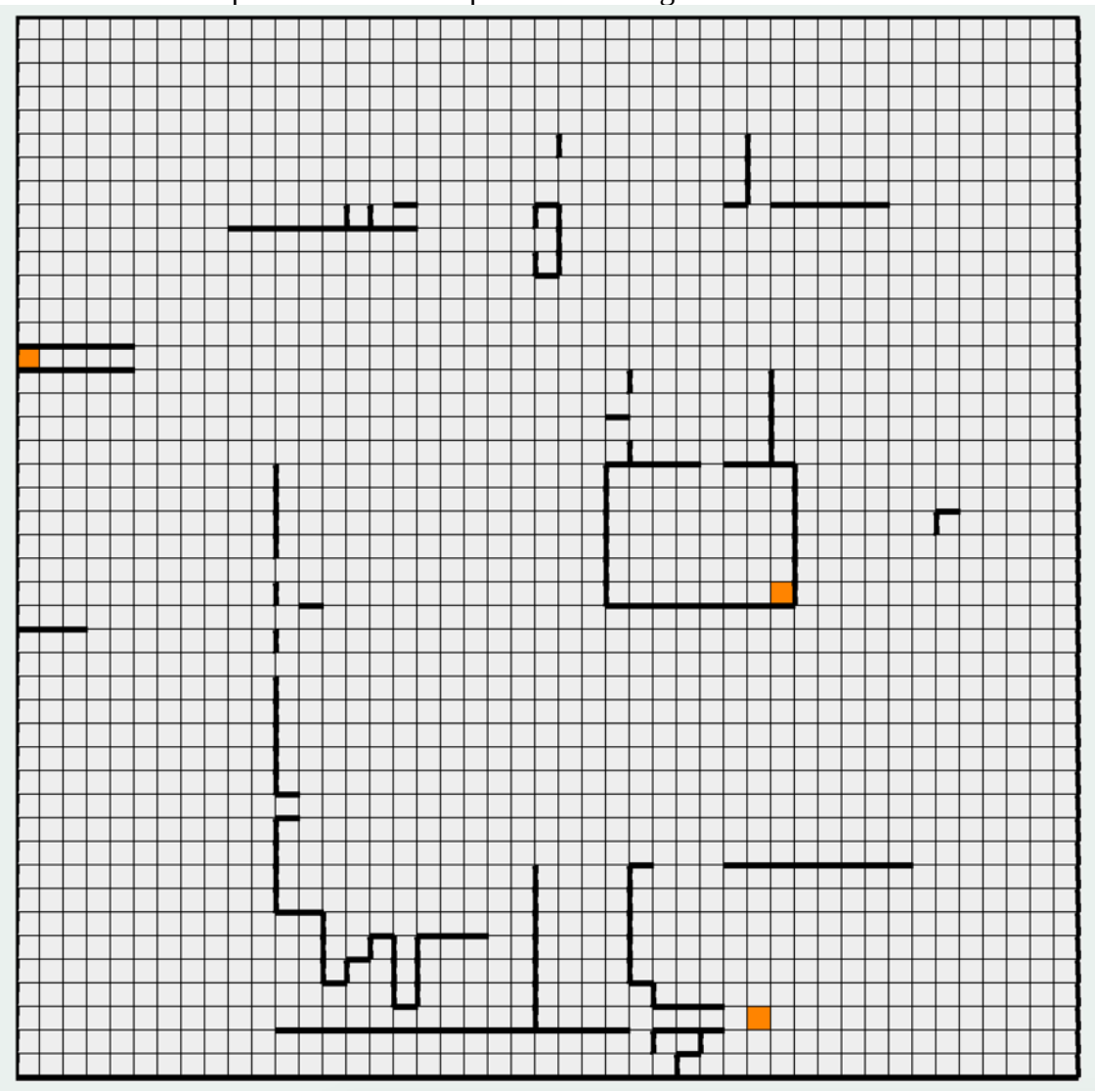


The MDP is a 10x6 gridworld with the starting point being the bottom left corner, and the goal state being the orange square on coordinates (7, 0). On the edges of the map there is an "electric fence," which gives a penalty if the agent bumps into it. There are four possible actions the agent can take: incrementing and decrementing the x or y coordinate. The transition model is represented by a single variable, PJOG, which represents the amount of noise in the environment. Every time an agent takes an action, there is a 1- PJOG chance that the agent will take the intended action, and a PJOG/3 chance that the agent will take any other action. In addition, the transition gives a penalty equivalent to the path cost, meaning the agent has an incentive to reach the goal as fast as possible. This MDP is interesting because it contains a small number of states, yet has a level of depth to it. Clearly, the shortest path to the goal is to walk along the bottom, but there exists a risk of receiving the penalty from the electric fence. Testing this environment with value and policy iteration on different parameters will give insight into how these algorithms deal with risk and reward in contrast to our intuition.

3 Goals Gridworld

The "big" MDP is a 45x45 gridworld with the starting point, parameters and algorithm implementations being the same as the small gridworld. As before, hitting wall will incur a penalty, and transitioning to another state will incur a penalty equal to the length of the path the agent has already travelled. This MDP is interesting because it features 3 possible goal states:

one that is at the end of a wall tunnel, another in a small room, and the last gridworld being on the other side of a partial wall. The representation is given below:

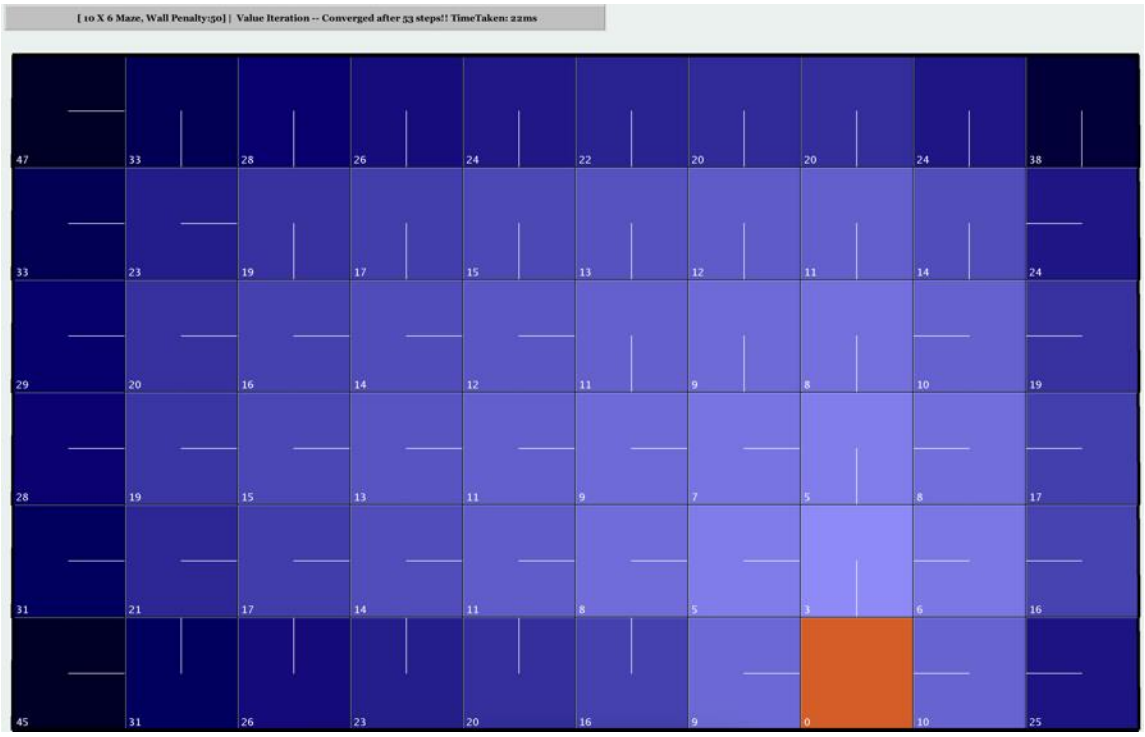


The closest state is the one closer to the right bottom, but to reach it requires the agent to traverse very closely to some walls. The second closest is the one on the left side of the map, but this requires the agent to traverse an extremely narrow tunnel. The third goal is somewhat further away, and requires the agent to enter a narrow door, and then navigate a small room. Lastly, it's important to note that the agent has a relatively safe, but long, path should it choose to go all the way around the wall to the first goal. I chose this MDP because it has the same transition model, state representation, and algorithm implementation as the first MDP, meaning comparing the two will be easier. I think this MDP is interesting because it contains multiple goal states, each, I assume, having their own benefits and drawbacks. This MDP is also more than 30 times larger than the small MDP, having 2025 different states. We can use the experiment results to see how the state space represents the run-time of both algorithms.

Electric Fence

Value Iteration

As a baseline for our analysis, we will run value iteration and policy iteration with the default parameters: PJOG = 0.3, with a wall penalty of 50.



The numbers in the boxes represents the value function of each state, and the lines in each state represent the policy extracted from the values. Surely enough, value iteration recognized the risk by calculating the expected utility through the Bellman equations, and directed the agent away from the fence even though walking along the fence yields the shortest path. However, in the blocks adjacent to the goal state, the policy suggests that the agent takes the risk of hitting the fence. This is probably because the transition cost (path cost) near the goal state is high, and if the agent were to accidentally move away from the goal state, it would require at least 4 more transitions, which is more than the cost of hitting the fence and just trying again. Value iteration converged in 53 iterations, taking 22 milliseconds. This is expected as the number of states is small.

Next, I ran the experiment on varying levels of PJOG:

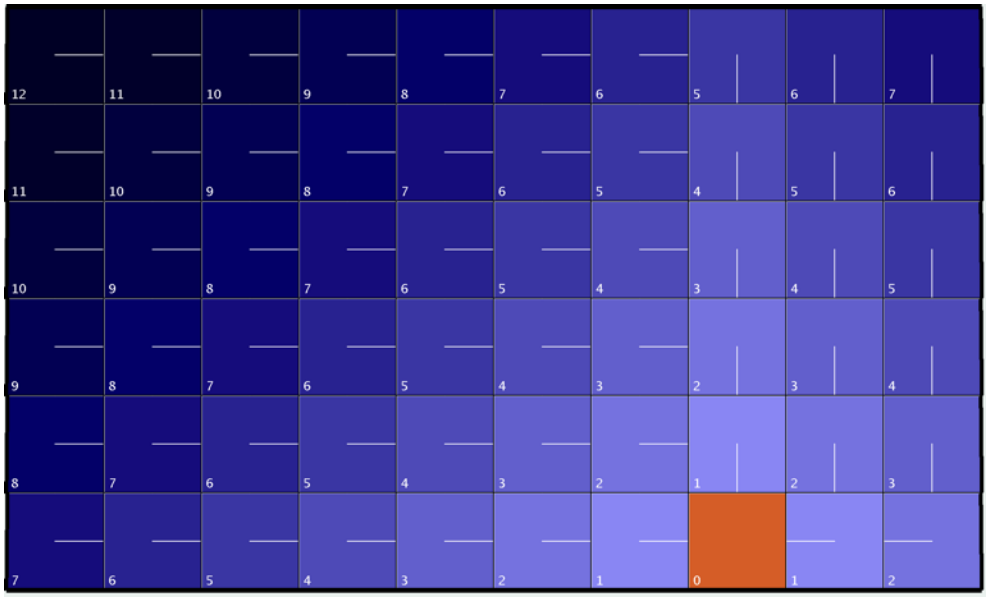
PJOG	Iterations to Converge	Time (ms)
0	13	3
0.2	36	4
0.4	82	12
0.6	268	23
0.8	866	76
1.0	227	21

An observation on the general trend of the experiment is that it takes longer to converge when PJOG increases, yet decreases again when PJOG is 1. An idea as to why this is so is that value iteration takes longer to converge the more uncertain the transition model is. In other words, the

closer the probabilities of taking each action, the longer it will take value iteration to converge. The most uncertain transition model is when PJOG is set to 0.75. In this case, every action is equally likely regardless of intent. Sure enough, this setting takes 1763 iterations to converge, and any policy would technically be "optimal."

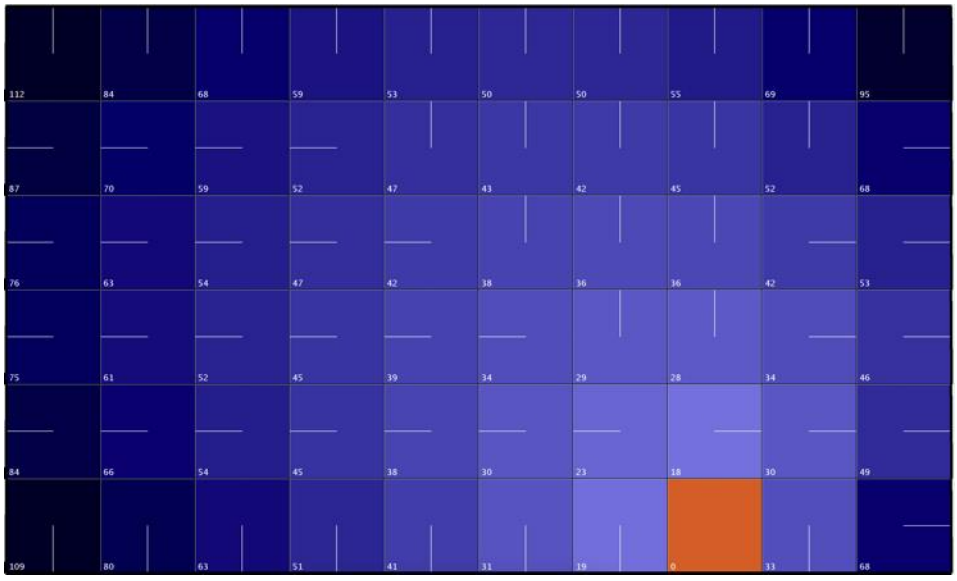
Notable Observations

Below is the result for PJOG = 0:



In a deterministic transition function, the problem becomes trivial. After initialization, every state requires exactly one iteration to converge.

Below is the result for PJOG = 1:



It seems strange that the policy is exactly moving away from the goal and towards the fences. However, looking back at our transition model, if PJOG is 1, then there is a 0 percent chance we

go in the direction we intend and a third chance to take any other action. Thus, value iteration can capture this unintuitive model and give an optimal policy.

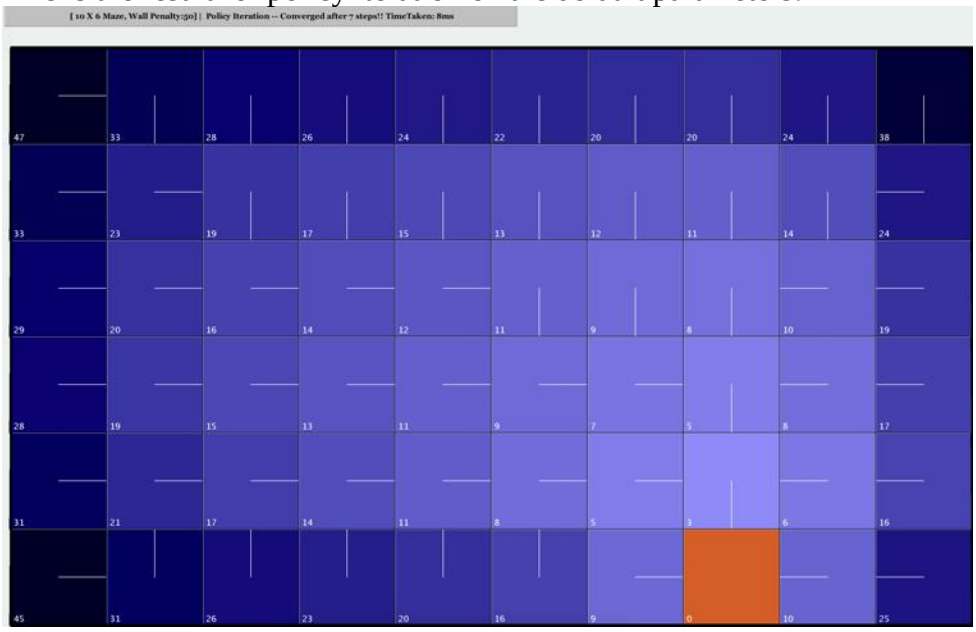
Another parameter we can change is the penalty for hitting a fence. PJOG is set to 0.3.

Penalty	Iterations to Converge	Time (ms)
0	45	3
30	51	5
60	53	3
90	54	6
120	55	7
150	55	7

Other than when the penalty was set to 0, there was no significant change in policy. When the penalty was set to 0, the problem became trivial, and the policy was just to go straight towards the goal. An interesting note on this experiment is that the iterations and run-times do not change much. This is probably because the penalty only functions as a scalar in the value iteration update equations that only matter in certain edge cases, whereas changing the noise in the environment requires value iteration to consider more actions. After toying with more extreme values with the penalty, I realized that the policy would remain largely the same. However, the values that were displayed for each state after iterating were larger the higher the penalty was. Even though the values are different, the policies are the same. This is because the policies are extracted by a comparison of state values rather than the raw values. In other words, the policy for two states with values (0.5,1) is the same if the values were (0.9,1).

Policy Iteration

This is the result for policy iteration on the default parameters:

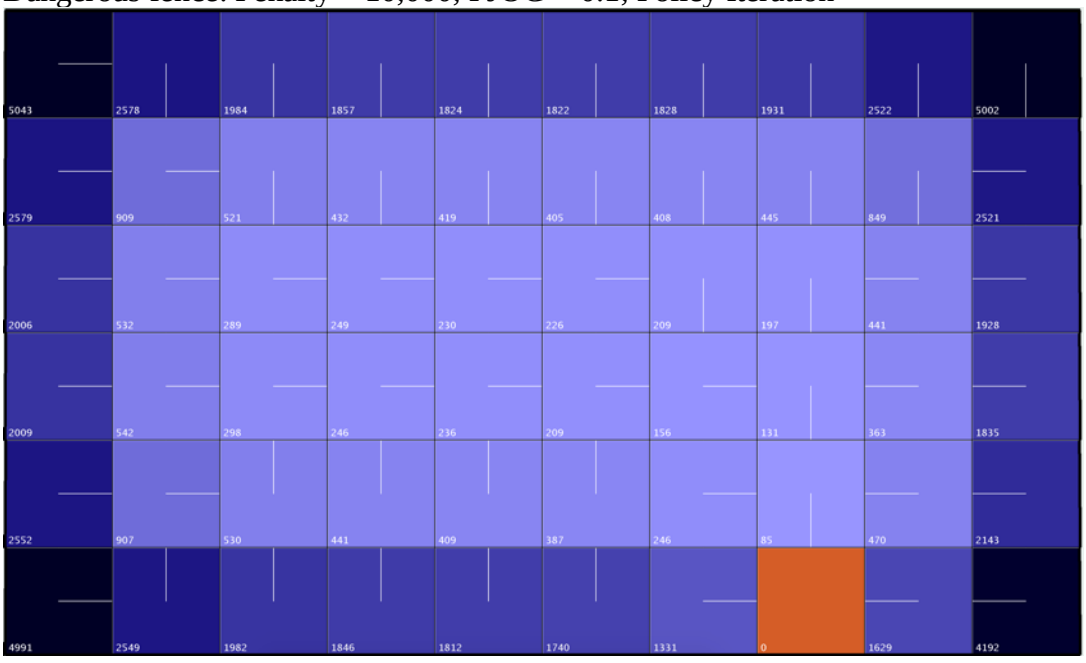


The results are the same as value iteration. The only difference is that policy iteration took fewer iterations to converge. The same results make sense, because both policy iteration and value iteration are guaranteed to converge on the optimal policy. It is not guaranteed, however, that the actual values for each state will be the same. Nevertheless, I will run the same experiments as value iteration to compare the iterations and run-time.

PJOG	Iterations to Converge	Time (ms)
0	3	4
0.2	5	6
0.4	6	12
0.6	4	19
0.8	3	23
1.0	5	15

Here we see policy iteration out-performing value iteration in terms of number of iterations required to converge and the raw time it takes to converge. It may seem counter intuitive because value iteration only extracts the policy from the values once, whereas policy iteration updates after every iteration. However, the equation for value iteration requires us to find the max for all possible actions, whereas the equation for policy iteration does not include such a max. I must also note that for the experiment where PJOG = 1, the policy returned by policy iteration was different from the policy returned by value iteration. This shouldn't happen because both algorithms are supposed to return the optimal policy. However, the PJOG = 1 experiment is a special case because all actions are equally regardless of intent. Therefore, any policy returned is an optimal policy. Because policy iteration will find the same policy as value iteration, and we have already established the differences in run-time between the two, I will not repeat the experiments where I vary the penalty. Instead I will try to think of an interesting situation and see if my intuition lines with the optimal policy.

Dangerous fence: Penalty = 10,000, PJOG = 0.1, Policy Iteration



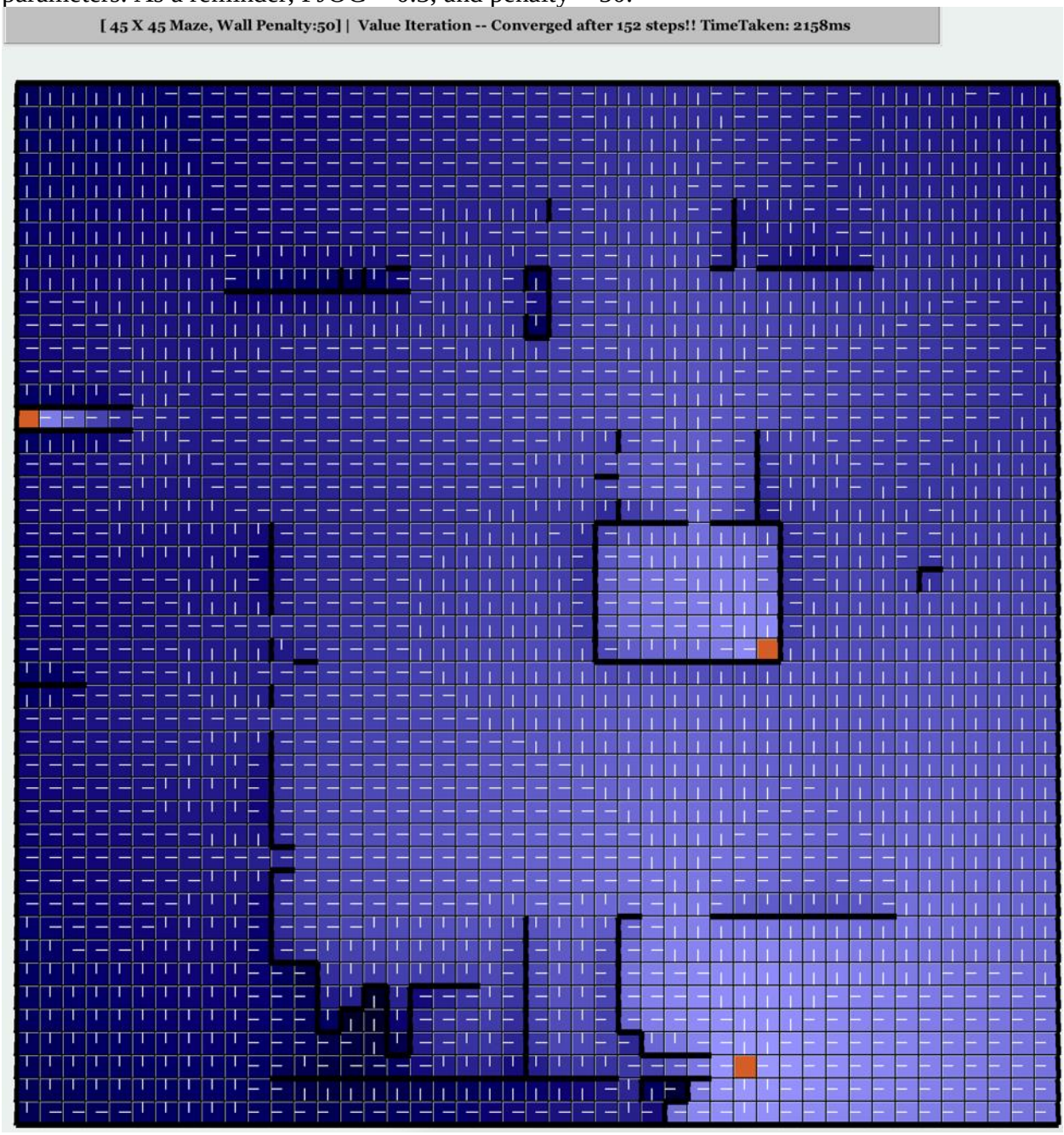
Here, I was expecting that the two policies to the left and right of the goal state would change to the up direction due to the extremely high penalty. However, the policy remains the same in those cases. One post-hoc rationalization is that if the agent tries to move up, there is a 3% chance of hitting a fence, and if the agent moves left, there is also a 3% chance of hitting the fence, but also a 90% chance of hitting the goal state. We also see the policy changing on the second to bottom layer. Instead of going right, the agent chooses to be prudent and go up. Even

though there is a buffer layer before hitting the fence, the penalty for hitting the fence is so high that the agent chooses to have 2 buffer layers on the (0.03) * (0.03) chance of hitting the fence.

Three Goals

Value Iteration

This is the resulting policy extracted from value iteration after running it on the map with default parameters. As a reminder, PJOG = 0.3, and penalty = 50.

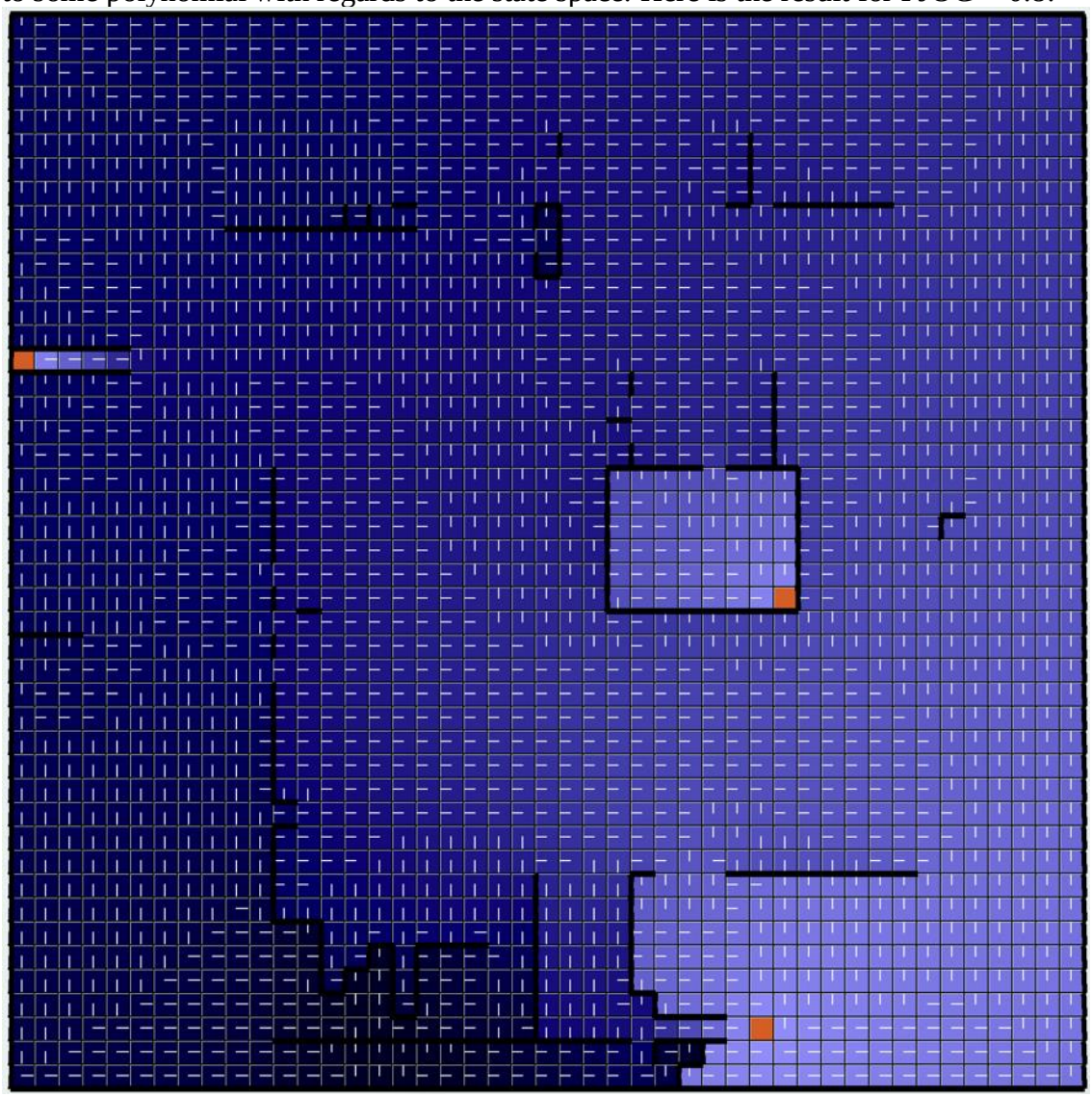


If we follow the optimal policy starting from the beginning, we see the agent cross the wall through one of the narrow openings and make its way to the goal at the bottom. Even though there was a shorter path to this goal along the bottom of the map, it is extremely close to the walls, so it makes sense that the agent took the next closest path. The run-time for this algorithm

was huge, taking almost 300 times longer than the longest run-time of any value-iteration experiment from the small MDP. Now I will vary the PJOG and re-run the experiments:

PJOG	Iterations to Converge	Time (ms)
0	43	566
0.2	106	1392
0.4	245	3397
0.6	926	12708
0.8	5232	71039
1.0	599	8117

An observation on the general trend of the experiments is that the run-times are significantly higher than the in the small MDP. While this is expected because there are more states to iterate, the state space difference is by a factor of about 30, but the time differs by a factor of more than 900 for some experiments. This suggests that the run-time for value iteration increases according to some polynomial with regards to the state space. Here is the result for PJOG = 0.8:



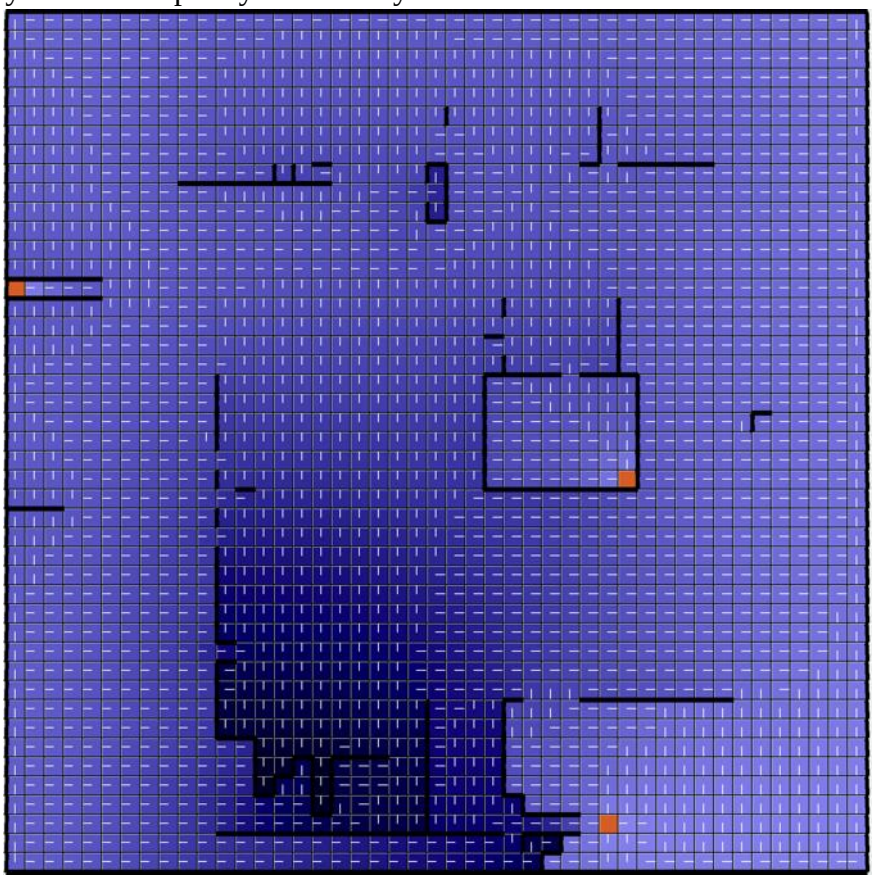
What is so interesting about this policy is that it is very unintuitive. Following the policy from the start leads straight into a wall near the bottom. Yet we know that this mathematically the optimal policy. This experiment reveals that the algorithm can solve problems that are well beyond human intuition. Something else I realized is that setting PJOG = 0 will give a policy pointing to the closest goal, meaning that value iteration can be used to find the closest goal state from any point. This is analogous to creating K clusters, using Euclidean distance as a distance function, where k is the number of goal states.

Policy Iteration

Penalty = 50

PJOG	Iterations to Converge	Time (ms)
0	42	19683
0.2	18	10260
0.4	16	13316
0.6	13	21787
0.8	45	1810
1.0	107	1248

Here we see policy iteration maintaining its superiority over value iteration in that it takes fewer iterations to converge. However, in this MDP policy iteration sometimes takes much longer than value iteration in terms of raw run-time. One idea for why this might happen is if updating the policy becomes expensive. Value iteration only extracts the policy once at the end, whereas policy iteration must update the policy after every iteration. My guess is that updating the policy for large state spaces takes a long time, so even though policy iteration takes fewer iterations to converge, each iteration takes a longer time to complete. Policy iteration also has the tendency to slow down near local optima. It may have been the case that the map contains many of such local optima. Another strange thing I noticed was that the policies returned by policy iteration did not always match the policy returned by value iteration.



Above is the policy returned for PJOG = 0.8 using policy iteration. A brief comparison to the policy returned by value iteration reveals that these policies differ slightly. This is not like the case where PJOG = 1, in which any policy is optimal. However, it may be the case that there are multiple optimal policies, and that policy iteration just happened to find one different from value iteration. Another possibility is that the policies are different, but by a small enough margin that the algorithm implementation does not discern between them. One parameter that I did not mention is precision. Both algorithms terminate once updating the value of each state does not change by more than the value set by precision. The default value is 0.001, and it may have been that policy iteration got a different policy due to this precision.

Another interesting observation is that in both algorithms, not matter what PJOG was set to, the policy would (if it was human readable) go towards the same goal: the one towards the bottom of the map. I had edited the map to try and make it ambiguous as to which goal was the best goal to reach, but it seems as if I made a map that only had one truly best goal. It may also have been the case that the policies that are not human readable tried to reach another one of the goals, but if they had I do not know because the state space was too large to get a grasp on what the policy was trying to do. One possible way to get the agent to go towards another goal would be to alter the penalties. Maybe if hitting a wall was not bad, the agent would try to go up to the goal in the tunnel. If hitting the wall would be extremely bad, then maybe the agent would take a safe route all the way over the wall and to the first goal again. The reason I did not conduct penalty experiments on this map is because 1) there is not much space left in this paper and 2) the software used to run these experiments would require me to remake the map from scratch just to make the walls have different penalties, and placing walls in the editor is a huge pain.

Conclusion

It seems that both algorithms have no problems finding the optimal policies, but policy iteration seems to run faster on smaller MDP's. In the case of larger MDP's, sometimes value iteration runs faster due to not having to update the policy as much, whereas policy iteration must update the policy after every iteration and can also slow down near local optima. In any case, policy iteration takes fewer iteration to converge, according to my definition of an iteration, because it seems to be doing multiple passes of value iteration before updating the policy. It quickly becomes apparent that for non-gridworld MDP's, such as continuous problems or even more complex state representations, both algorithms become unpractical. This is because they must iterate through the entire state space. 2025 states (the size of the larger MDP) is not even that large, especially when compared to complex games such as checkers and chess. In this case, using Q-learning would make more sense, because Q-learning does not require us to visit all the state spaces. However, there is no guarantee that Q-learning will allow us to converge on the optimal policy. Therefore, each algorithm has its own drawbacks and situations in which they should be used.