CS 4641

Markov Decision Processes

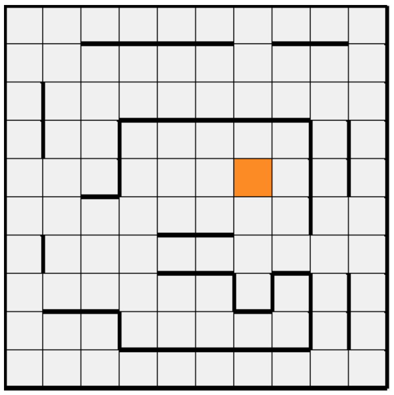
David Gong

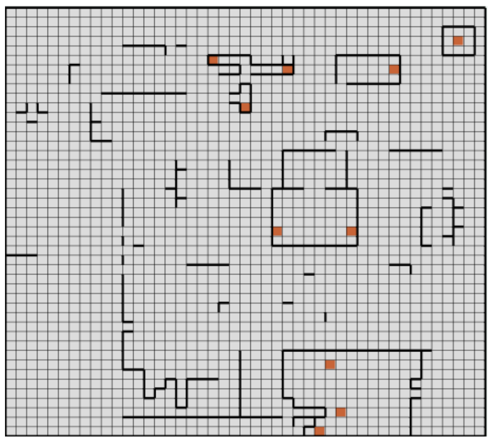
Problem Introduction

Using a reinforcement learning simulator developed by Rohit Kelkar and Vivek Mehta from the Robotics Institute of Carnegie Mellon University, I investigated two MDP problems that were of similar types but different scales in terms of number of states. Policy iteration, value iteration, and Q-learning will be looked at for each problem. For both problems, a -50 points penalty is instated for hitting the wall.

The first MDP problem is a 10x10 maze world, where one goal state is defined. To make the problem more interesting can be thought of as telling a robot in an Amazon warehouse how to navigate to an empty space, where the walls represent how the aisles are designed. As will be observed, the number of walls and states are of much lesser scale than problem 2.

The second MDP problem is of much larger scale, a 45x45 maze world, where multiple goal states are defined. To make the problem more interesting, we can think of the problem as zombie shelters from World War Z – the surrounding buildings within a city are destroyed and the agent must be able to find a way to navigate to a barricaded building/shelter where other survivors are located.


Problem 1 (10x10)


Problem 2 (45x45)

Value Iteration vs. Policy Iteration

Tests for convergence and number of steps for both value iteration and policy iteration were conducted for both problems. We first look at differing values for the "Amazon warehouse problem" (10x10 maze world), and then the "World War Z city shelter problem" (45x45 maze world).

Prior to conducting analysis, one should keep in mind that while somewhat similar, value iteration operates based on finding an optimal value function and one policy extraction, and policy iteration operates based on policy evaluation and improvement, repeated iteratively.

Amazon Warehouse Problem (10 x 10)

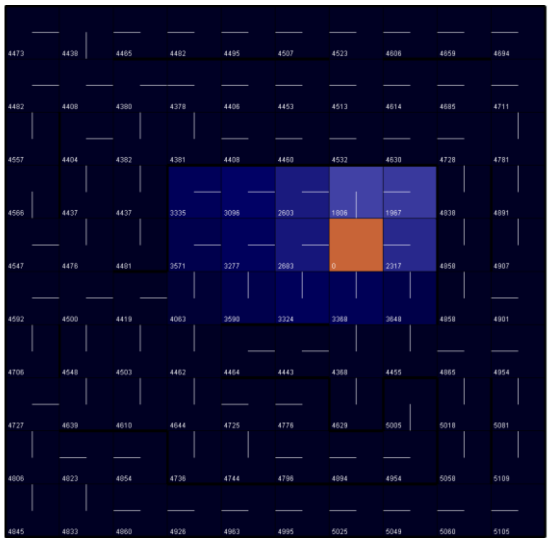
A table is provided, which labels the different metrics for value iteration and policy iteration. The average of around 10 iterations were taken for time, with # of iterations taking the same amount. The key metric that is compared here is PJOG, which in this case represents the probability that the agent moves in an opposing move compared to its intention.

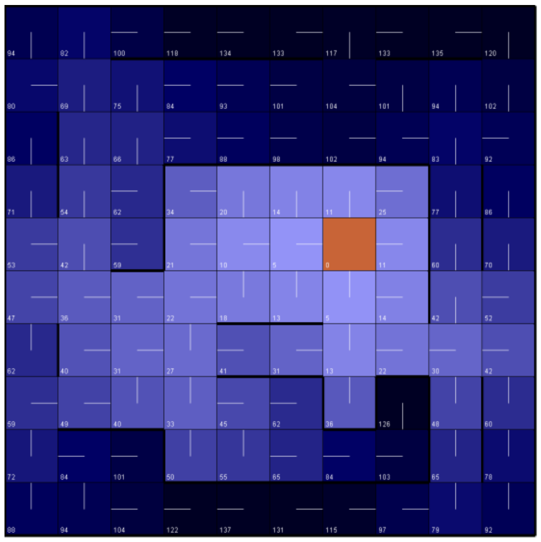
Algorithm	PJOG	Total Time	Iterations
Value Iteration	**PJOG**	**Milliseconds**	**Steps**
	0.3	23ms	65 steps
	0.2	13ms	44 steps
	0.1	7ms	31 steps
Policy Iteration	**PJOG**	**Milliseconds**	**Steps**
	0.3	72ms	7 steps
	0.2	65ms	6 steps
	0.1	57ms	7 steps

The main trend that is investigated here is that as PJOG increases, the number of steps and time it takes to converge also increase on average. This trend was to be expected, primarily because having an agent move against a direction/move that was calculated to be optimal by value or policy iteration in general causes an agent to not be able to reach a target state in the least number of steps possible. This in turn increases the time it takes to reach a goal state. However, I did manage to find one counterexample.

When I changed PJOG to 0.9 for policy iteration, an odd trend I noticed was that the average time it took to converge ended up being 22ms, which goes against the increasing PJOG corresponds to increasing convergence time trend. This basically means that for some of the calculated "optimal" policies were actually suboptimal. For value iteration, however, the trend remained, as the time ended up increasing to 184ms on average with a larger PJOG.

Also, an interesting observation made is how policy iteration technically took longer to converge than value iteration. Typically, this is not the case, as policy iteration usually converges quicker than value iteration. Based on the previous result, however, it was observed that a higher PJOG actually led to quicker run times for policy iteration, and vice versa for value iteration. As a result, it seems that policy iteration seems less sensitive to PJOG changes in contrast to value iteration. The number of steps actually decreased with a higher PJOG for policy iteration, but in general the number of low steps can be attributed to the fact that policy iteration focuses on an iterative approach to policy improvement, as opposed to value iteration which extracts one policy value as a time. This results in more steps, especially with suboptimal pathing.


PJOG: 0.9 (Policy Iteration)


PJOG: 0.3 (Policy Iteration)

Lastly, since the scale for this problem is quite small, we should expect there to be some varying results with some similar trends when conducting the same analysis on our 2nd problem, which has 45x45 states in contrast to 10x10 states.

World War Z City Shelter Problem (45 x 45)

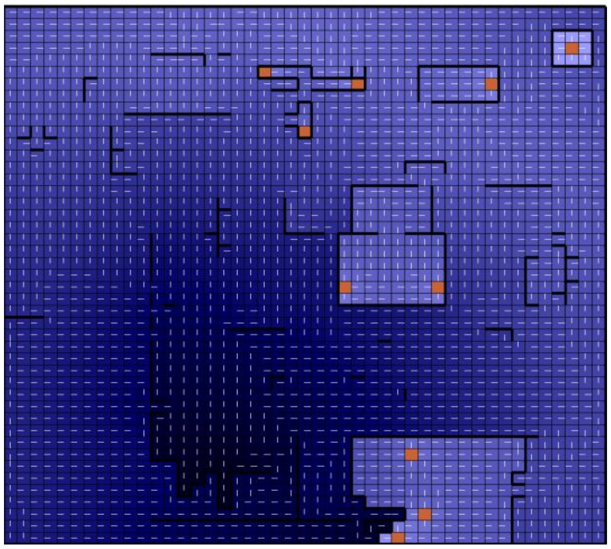
A similar table is provided for analysis on the “World War Z city shelter problem”, which is a 45x45 maze world and has a significantly larger number of states compared to the Amazon warehouse problem. Using the trends and questions from the previous question, an analysis is conducted on the results received.

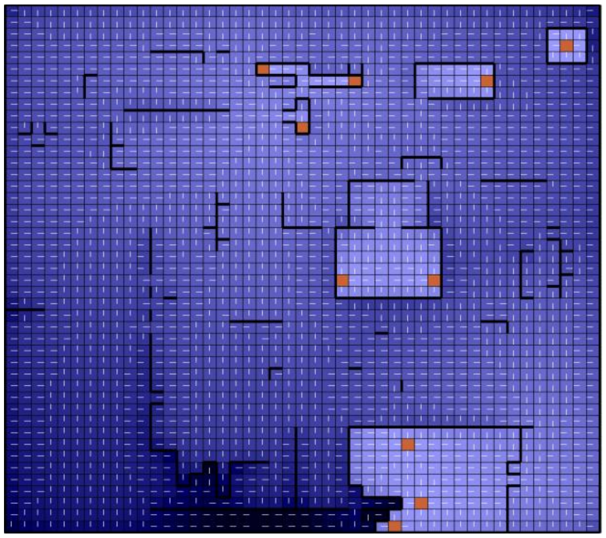
Algorithm	PJOG	Total Time	Iterations
Value Iteration	**PJOG**	**Milliseconds**	**Steps**
	0.3	4164ms	168 steps
	0.2	3035ms	121 steps
	0.1	2142ms	89 steps
Policy Iteration	**PJOG**	**Milliseconds**	**Steps**
	0.3	14802ms	18 steps
	0.2	12363ms	24 steps
	0.1	15452ms	24 steps

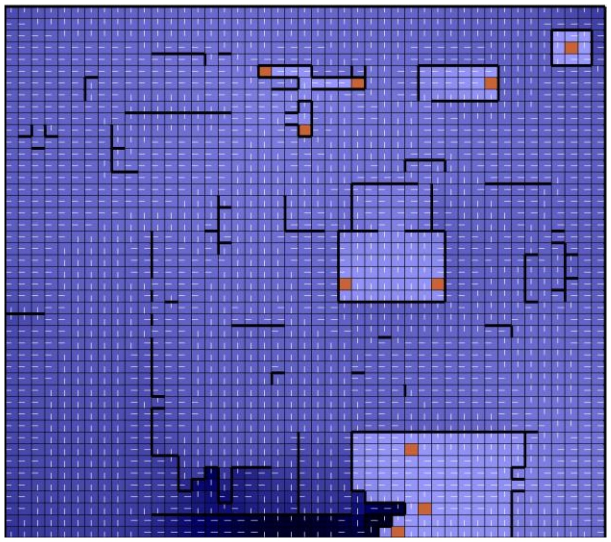
We see here that there is still no obvious trend for policy iteration. The PJOG does not seem to have any correlation with the time it takes for policy iteration to converge, though when PJOG is adjusted to 0.9, it takes only around 3200ms to converge for policy iteration, though it took 181 steps. This is similar to what happened in the previous problem’s counterexample. On the other hand, 0.9 PJOG resulted in a 35000ms to converge, thus supporting the trend that higher PJOG leads to a longer convergence time for value iteration.

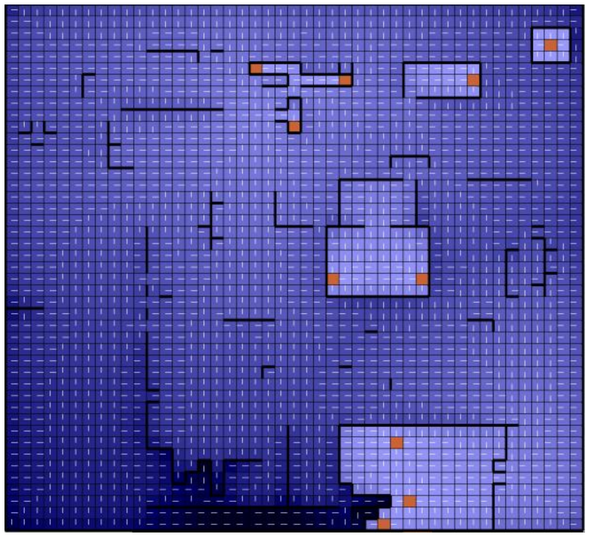
Value iteration, however, stays the same – the higher the PJOG, the longer it takes for convergence and the more steps that are needed.

As expected, an exponential increase in the number of states leads to an exponential increase in the time it takes to converge, even if the number of steps is similar. This is likely because the steps consider more state possibilities, probabilities, and values in the 45x45 problem as opposed to the 10x10 problem.


PJOG: 0.9 (Policy Iteration)


PJOG: 0.3 (Policy Iteration)


PJOG: 0.9 (Value Iteration)


PJOG: 0.3 (Value Iteration)

Based on the observations made with both problems, we can make a couple of conclusions. The first conclusion is that value iteration is more susceptible to PJOG changes. Value iteration takes an exponentially longer number of steps and time to converge when PJOG is increased. The likely reason for this is because for value iteration, since the optimal value function as well as one policy extraction is done, the policy that typically results from value iteration tends to be optimal, and as a result a higher PJOG impacts value iteration negatively.

On the other hand, a higher PJOG lead to actually lower convergence times for policy iteration, which suggests that increasing PJOG actually causes the agent to move in a path that was actually optimal which was calculated to be suboptimal. The likely explanation for this is that while policy iteration tends to be less computationally intensive, the calculation of general optimal pathings can prove to be somewhat unreliable in these specific cases. One thing to note as well is that it seems that policy iteration tends to be more stable in terms of runtime with regards to changes in PJOG, while convergence time for value iteration seems to maintain a strictly negative exponential correlation with PJOG.

The second conclusion is that there seems to be a positive exponential relationship between the number of states and the time it takes to converge. As can be seen, the 10x10 problem took at most less than 100ms to converge, while the 45x45 problem took around 60-80 times that amount on average to converge.

In addition to conducting tests and analysis with value iteration and policy iteration, additional tests were also conducted with another reinforcement learning algorithm known as Q-learning.

Q-learning

Value and policy iteration assume that the model is generally known - transition states, reward functions, and such. However, if the model is not known for a stochastic environment, per se let us say the stock market, then the agent must learn to be able to interact with the environment to optimize itself.

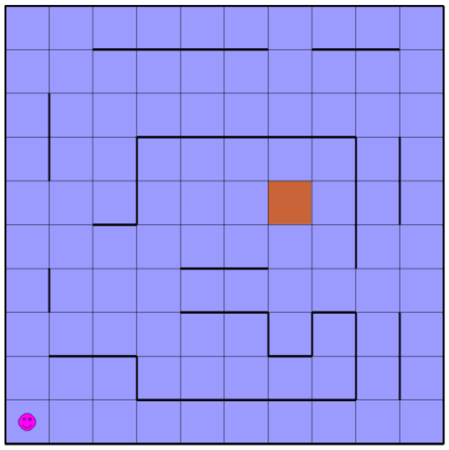
Though the model is more or less known in this case, it is still interesting to see how Q-learning stacks up versus value iteration and policy iteration when the model is in favor of the latter. In addition, the problem of exploration vs. exploitation should be explored. Exploration favors testing out suboptimal values in an attempt to find an even more optimal path after long-term iteration, while exploitation is similar to that of a greedy approach, which is more likely to use the optimal neighbor to find the “optimal” path to the goal state.

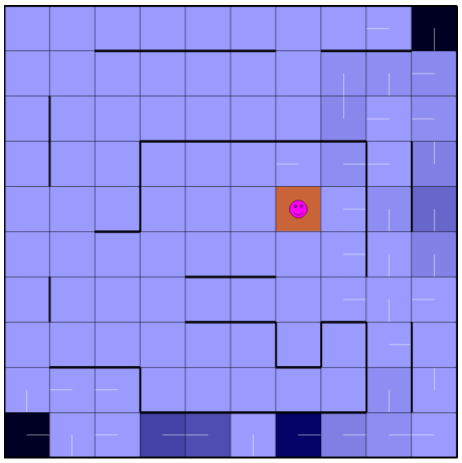
Amazon Warehouse Problem (10 x 10)

To measure the effects of exploration vs. exploitation, we can change the epsilon values. We can also investigate how learning rate affects our agent in this “unknown” environment, and how decaying learning rate affects the overall performance and optimality. We will always keep precision at 0.001 in this case, as done before. We will also maintain 1000 cycles for our Q-learning process. As a base for comparison, we will use:

PJOG = 0.3, Epsilon = 0.3, Learning Rate = 0.7, Decaying LR = True

Not all the possibilities will be visualized, but they will be discussed in detail based on significance. The visualizations also include cycles, which are hard to visualize. Rather, we would like to see how Q-learning performed with regards to finding the optimal path based on changed parameters.



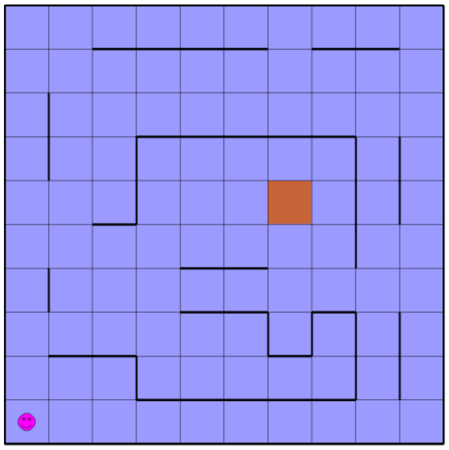


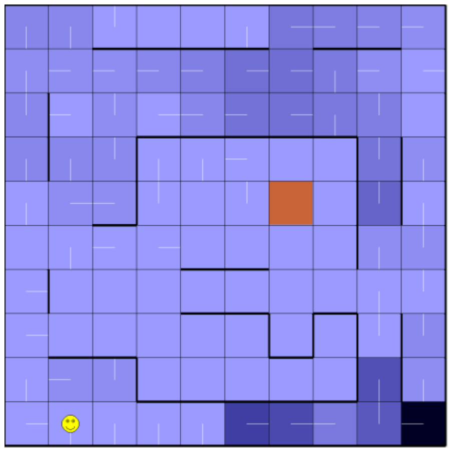
PJOG = 0.3

Epsilon = 0.3

Learning Rate = 0.7

Decaying LR = True





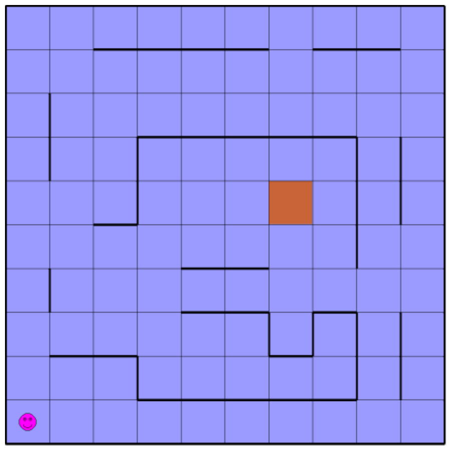
PJOG = 0.3

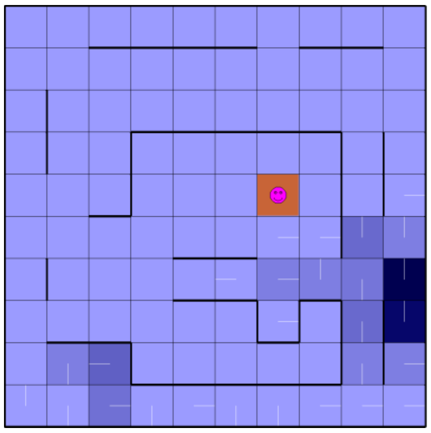
Epsilon = 0.3

Learning Rate = 0.7

Decaying LR = False

We see here that a decaying LR seems to be much beneficial for the agent, as calculations do not have as much bearing on adjacent states with a decaying LR. The agent often travelled around the same area for quite some time before moving onto a new set of states, and even looped around the entire maze before travelling to the goal state.



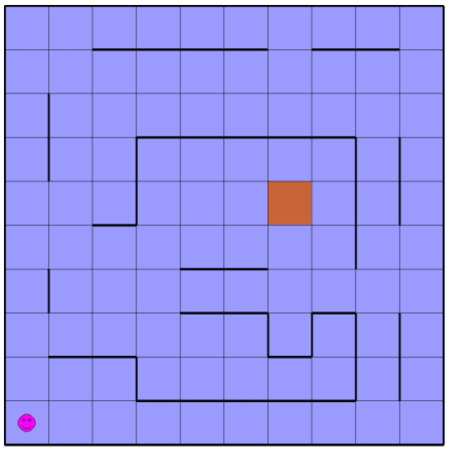


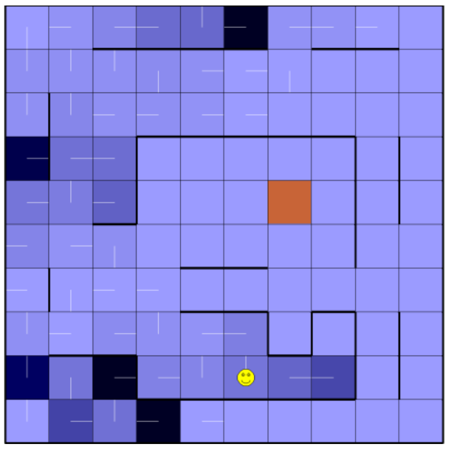
PJOG = 0.5

Epsilon = 0.3

Learning Rate = 0.7

Decaying LR = True





PJOG = 0.5

Epsilon = 0.3

Learning Rate = 0.7

Decaying LR = False

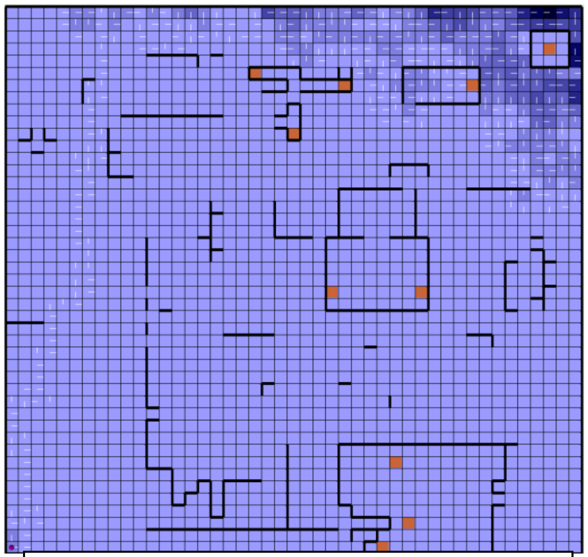
We can once again see that decaying LR often causes the agent to stick around a certain region and overfit and converge to a local optima. An increase in PJOG, however, surprisingly made the agent move in a manner that made it converge to the goal state faster. However, this can be explained by the fact that the agent is making suboptimal moves anyway, so increasing PJOG can actually make it move optimally more times than it does in a very suboptimal manner. It depends on the model overall, however.

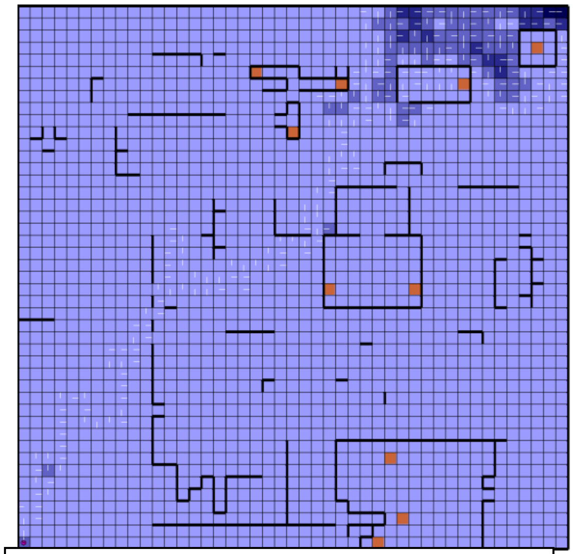
Not visualized in the above section is how Epsilon and Learning Rate affects the Q-learning agent. The summary of the results are as follows – Epsilon seems to offer only a bit of improvement in terms of change overall after multiple cycles are performed, but this could be because of the lack of the number of states, and a higher Learning Rate results in something similar to what one would expect from an increase in Epsilon, which is more of an inclination towards exploitation and travelling the path more travelled. Overall, however, we can conclude from the Amazon Warehouse problem that decaying LR seems to have possible implications, and adjustments for PJOG can also have a substantial impact. We lastly investigate how changing these parameters and variables can produce an effect on a much larger scale maze world.

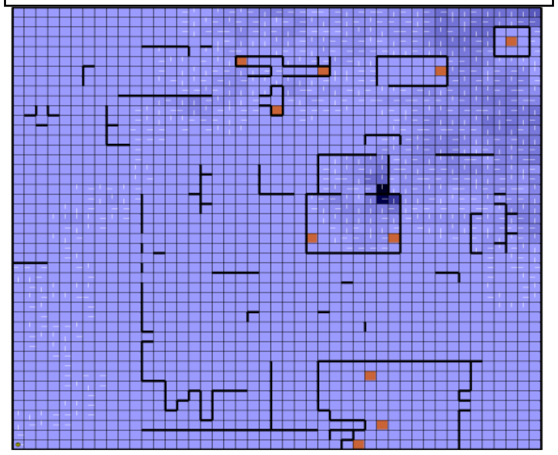
World War Z Problem (45 x 45)

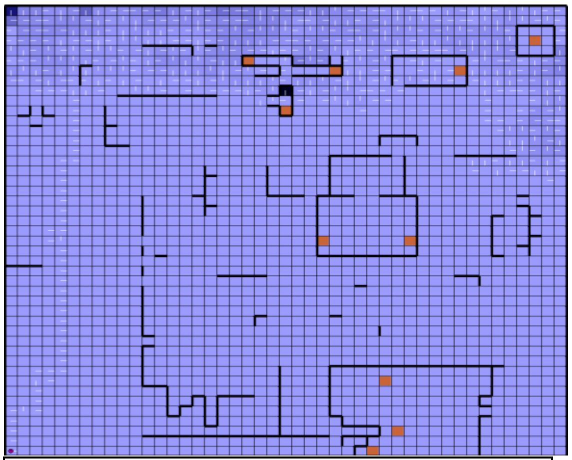
The same base for comparison is used as last time. This time, we visualize the episode results for the variables that were not visualized in the previous problems, namely epsilon and learning rate.

In addition, we would also like to most importantly make a distinction between exploration vs. exploitation for this problem, as a definite conclusion was not able to be made with the Amazon warehouse problem.


PJOG = 0.3, Epsilon = 0.3, Learning Rate = 0.7, Decaying LR = True


PJOG = 0.3, Epsilon = 0.3, Learning Rate = 0.7, Decaying LR = False


PJOG = 0.3, Epsilon = 0.6, Learning Rate = 0.7, Decaying LR = True


PJOG = 0.3, Epsilon = 0.1, Learning Rate = 0.7, Decaying LR = True

In this case, decaying LR actually seemed to have a positive impact on the convergence to a possible goal state. Another interesting observation is how the Q-learning agent tried to reach the top right goal state, which is blocked off by walls and inaccessible, but the agent insists on trying to converge to that specific goal state first.

Due to some page constraints, additional visualizations will not be provided, but an explanation of the results will be done. With the top right goal state, no matter how I changed my parameters, the agent always attempted to reach that goal state. This is likely explained by the fact that there are less reward penalties for moving upwards in the beginning, and as a result, the agent attempts to move in the way that is least susceptible to wall encounters. This results in the agent moving upwards, encountering some upward walls, then moves to the right, then upwards again, and lastly it stays at the top right corner, where in normal circumstances it would converge to a goal state. However, I blocked the wall on purpose, because I was curious to see how the agent would react when it triggers the penalties for hitting the wall. Surprisingly, the agent often stayed in that area for quite some time.

An important note, however, is the exploration vs. exploitation result. When epsilon was increased, ultimately only a few solutions were encountered. However, when epsilon was decreased in favor of exploration, multiple solutions were able to be encountered which were otherwise not accessible without performing exploration. The middle and second-most upper right goal states were reached for the first time with exploration. However, the tradeoff lies at the time it takes to converge – it ultimately takes much longer for the agent to converge.

PJOG can break the trend mentioned in the paragraph previous to the previous. This is because it is more likely to move in the suboptimal manner when increased. However, once again the agent always seems to consistently somehow make it to the top right-hand corner, which suggests that the reward penalty for hitting a wall is quite large. If reduced, some other possible avenues of exploration could be performed.

Conclusion

Through conducting analysis of reinforcement learning algorithms, namely value iteration, policy iteration, and Q-learning, we find that there are scenarios in which one algorithm is better than the other. Value iteration and policy iteration are ideal for use on known models, while Q-learning is best used on an unknown model.

We first found that value iteration actually converged faster than policy iteration in terms of time, though it required many more iterative steps than policy iteration. We also learned that value iteration typically returns a true optimal policy, while this is not necessarily the case for policy iteration. However, policy iteration in general is still faster and more stable, while if PJOG was greater than expected, value iteration performs poorly. The most surprising revelation overall is the fact that value iteration was faster than policy iteration in both problems, when in general it is expected that policy iteration converges faster than value iteration.

The second part of the assignment involved using another reinforcement learning algorithm to perform exploration strategies, namely Q-learning. The main issue to consider with regards to Q-learning is Exploration vs. Exploitation. Through changing the epsilon values and learning rates, we were able to adjust for both cases. Ultimately, it was found that no one method is better than the other – rather the best method would be to incorporate a balance between the two strategies.