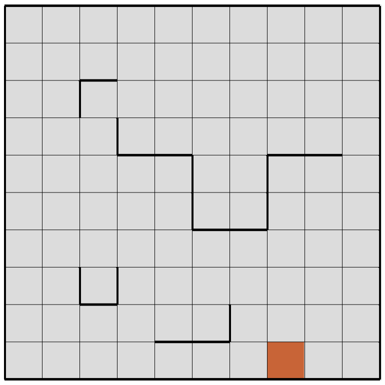
Sagar Laud CS 4641
Assignment 4

Markov Decision Processes

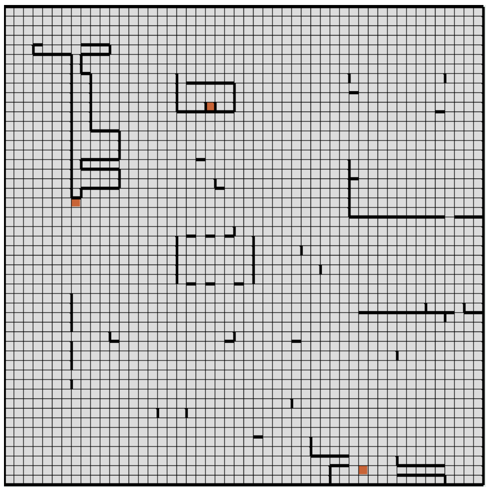
Introduction:

This project focuses on Markov Decision Processes and how they can be solved using multiple algorithms. Value Iteration, Policy Iteration, and Prioritized Sweeping were used to solve both a small and large MDP.

Problems:



The maze above is the "small" MDP which was solved via the aforementioned algorithms. It is a 10x10 maze with one goal state. This maze is interesting because while there is only one goal state, and it is a relatively small maze, there are still traps which would be useless to explore. Specifically, the trap almost directly in the center of the maze turns a problem which is extremely simple into one which will require a little more work to solve. However, this is still an easy problem to solve overall thereby allowing a comparison later with how the algorithms perform on both simple and difficult problems. There are 100 states total and hitting a wall incurs a penalty of 50.



The maze above is the “large” MDP which was used for this project. It is a 50x50 maze with three goal states. This maze is significantly more complex than the smaller one. It has several traps placed in it (with, interestingly enough, a goal in one trap and another just outside one), to cause more difficulty thereby making it more challenging in terms of convergence with all of the algorithms used. This will allow us to compare the strengths of the algorithms in both a large and small state space as well as comparing how they handle more difficult problems. There are 2500 states total and hitting a wall incurs a penalty of 50.

Policy Discovery:

For the purpose of determining the optimal policy, both Value Iteration and Policy Iteration were used. The results were interesting. Both algorithms performed well though as some parameters were modified, the results started to change drastically. The parameter that was modified was the parameter to determine how much noise was in the environment. The parameter which denoted the amount of noise was referred to PJOG with a larger PJOG value simulating an environment with more noise. Specifically, the desired direction was chosen with 1-PJOG probability.

Value Iteration:

The following table illustrates the time and number of steps taken to converge varied with the amount of noise in the environment for the small maze using Value Iteration.

PJOG	Time(seconds)	Steps
0.1	0.016	35
0.2	0.019	59
0.3	0.032	87
0.4	0.051	122
0.5	0.08	188
0.6	0.145	391
0.7	0.552	1424
0.8	0.661	1645
0.9	0.195	634

It is clear that as the amount of noise increases, so too does the amount of time taken to converge. In addition, the number of steps increased quite steadily too. This was expected however as an increase in uncertainty would naturally cause the algorithm to have more trouble and therefore take more iterations and time.

The following table illustrates the results of Value Iteration run on the large maze with the same parameters as above:

PJOG	Time(seconds)	Steps
0.1	2.7	81
0.2	3.48	107
0.3	5.212	159
0.4	7.823	234
0.5	12.9	382
0.6	27.76	828
0.7	142.31	4209
0.8	175.21	5051
0.9	35.61	1036

We see similar results though the numbers are natural given a state space which is 25 times larger than the small problem. The number of steps did not increase linearly however though it certainly did increase considerably. This is again just due to the nature of the problem. However, I believe that having multiple goal states made the problem a little easier than if there had only been one goal state. With only one goal state, the algorithm would have certainly taken much longer.

Policy Iteration:

The following results are from the Policy Iteration algorithm, using the same changes in variables as Value Iteration.

PJOG	Time(seconds)	Steps
0.1	0.084	11
0.2	0.04	8
0.3	0.037	10
0.4	0.059	9
0.5	0.072	9
0.6	0.126	4
0.7	0.044	3
0.8	0.053	3
0.9	0.028	5

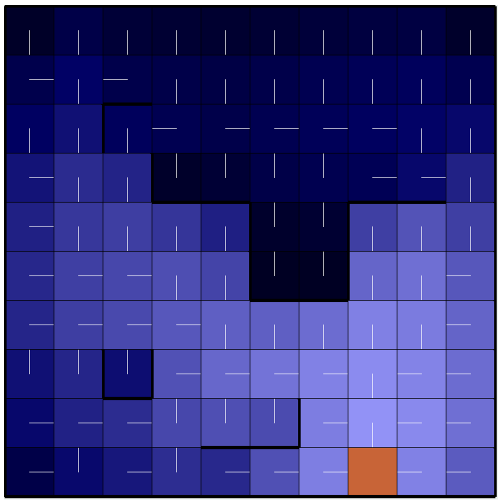
What stands out here is that there is no astronomical jump in time or steps between PJOG values of 0.6 to 0.8. In Value Iteration, we saw a huge jump here, but this is not something that we see here. There is still a jump however, so it is clear that making decisions is still a little more difficult as the level of uncertainty increases. The time increases rather steadily until a very high level of uncertainty.

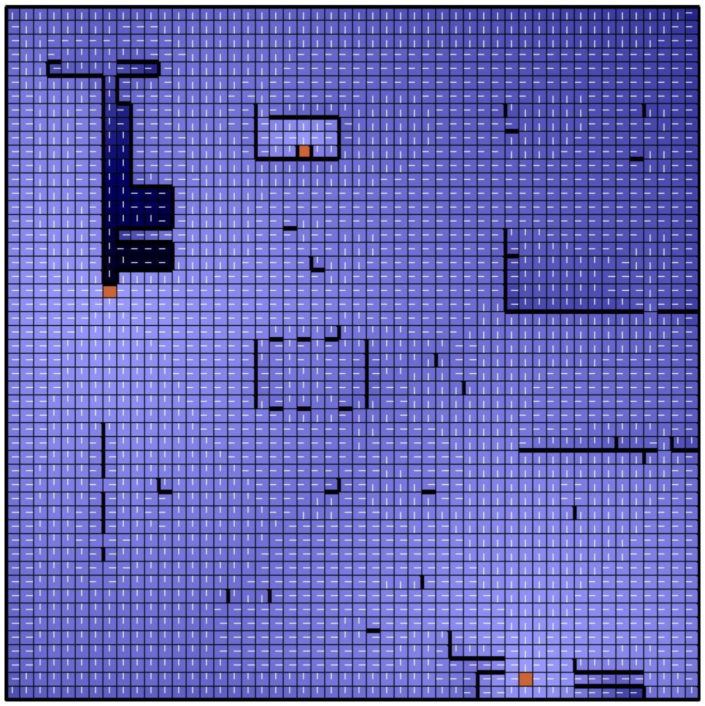
The following table illustrates the results of Policy Iteration run on the large maze with the same parameters as above:

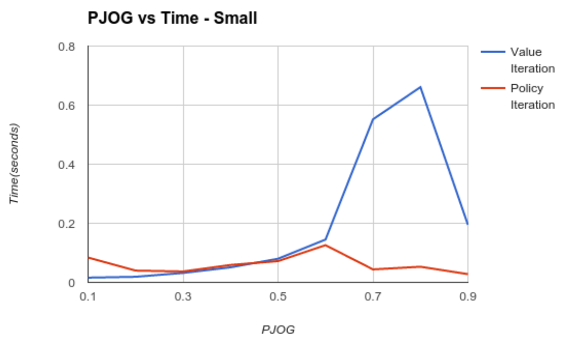
PJOG	Time(seconds)	Steps
0.1	6.39	33
0.2	7.25	34
0.3	9.34	34
0.4	10.41	37
0.5	15.16	49
0.6	20.84	118
0.7	6.01	177
0.8	3.61	77
0.9	9.17	412

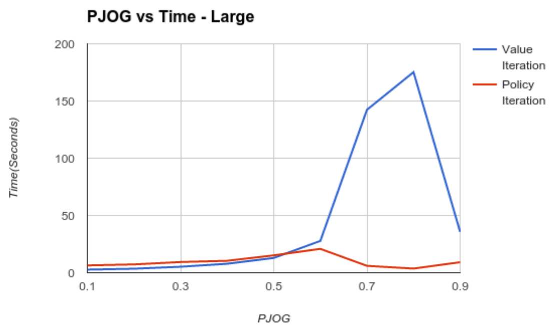
We see the same concept here. There is still an increase in time and in this case, there is an increase in the number of iterations rather steadily as well until we reach a rather high level of uncertainty. From there, the time taken drops drastically. It is worth noting that having fewer goal states would have certainly increased the amount of time taken to converge.

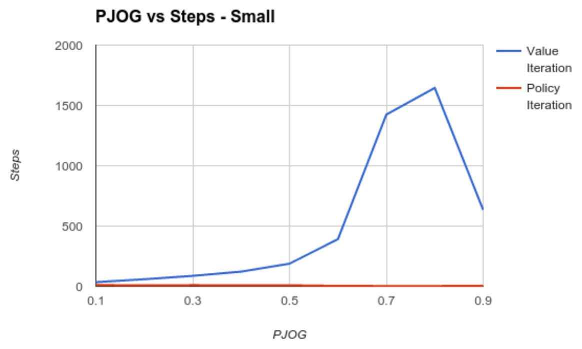
Value Iteration vs. Policy Iteration:

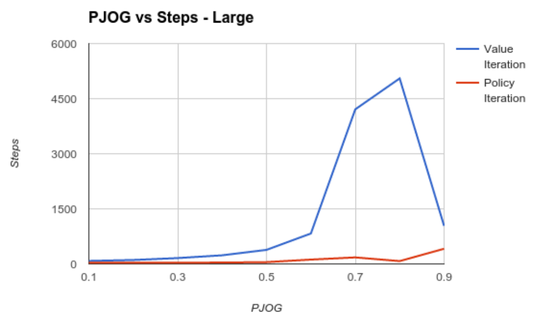












Now that we have looked at how value and policy iteration perform as the size of the space increases and as uncertainty increased, a comparison between the two algorithms is certainly warranted. It is worth noting that both algorithms generated the same policies as each other regardless of what the PJOG value was. The policies generated for a PJOG value of 0.3 are shown above for both the small and large problems. In addition, the figures above show how the two algorithms compare to each other as the level of noise increases in two separate problems. It is interesting to look at them side by side as the curves look pretty much identical in both problems (though the second problems numbers are larger by a substantial magnitude). However, to see how well these algorithms did, we should compare how much the computation time increased by compared to the size of the state space. We could compare the change in the number of steps though this is not necessarily a good metric. The steps in Value Iteration run more quickly than those in Policy Iteration though Policy Iteration is good at accomplishing more with each iteration. Naturally because there are multiple goals in the large problem, the results are far better than they could have been, however, since the problems were the same when it came to being solved by the algorithms, a comparison of this sort is valid. So, let us see what happens when the state space increases:

Algorithm	PJOG	Time Taken(small)	Time Taken(large)	Percent Increase
Value Iteration	0.3	0.032	5.21	16181.25%
Policy Iteration	0.3	0.037	9.34	25143.24%

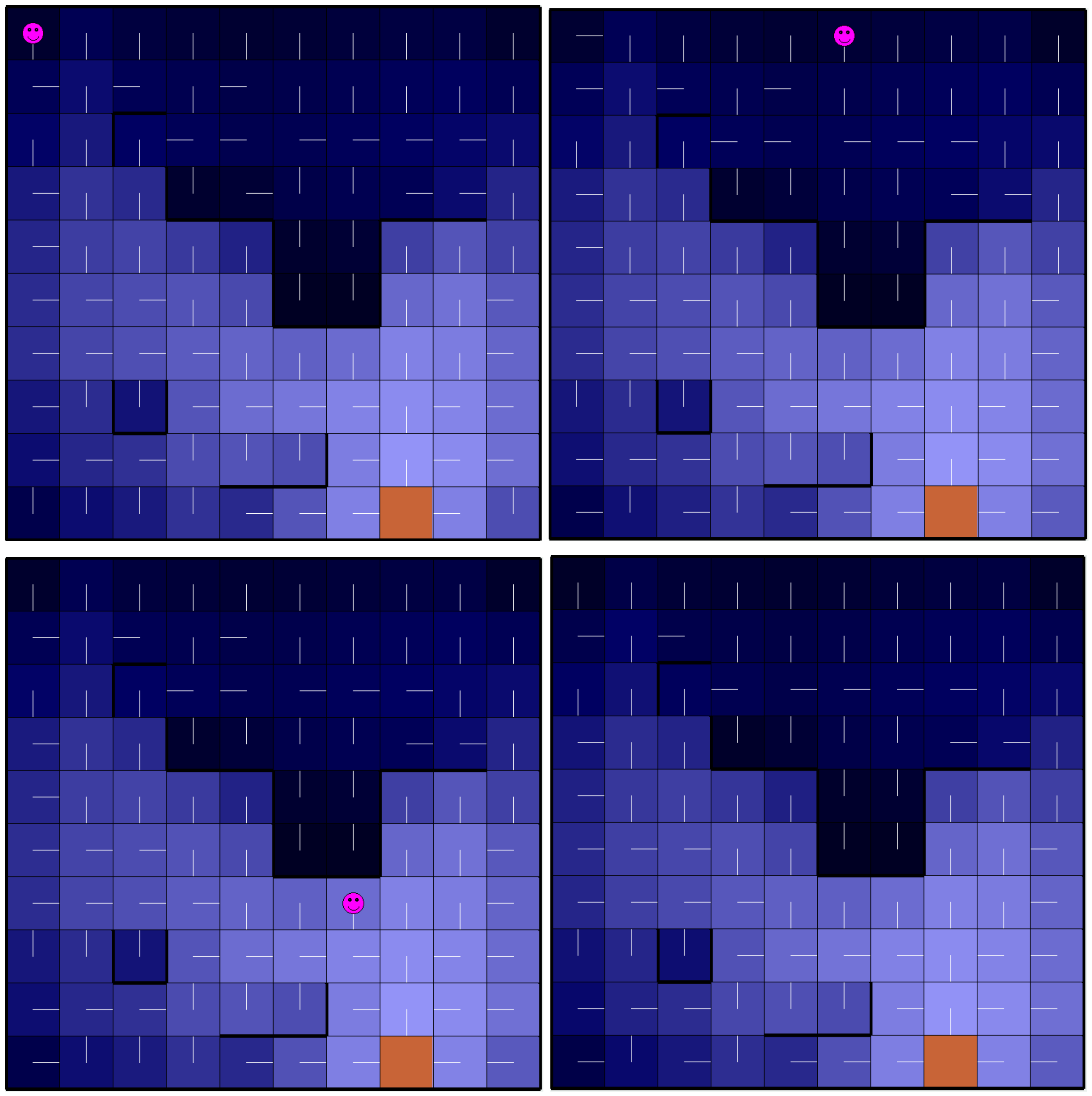
That datapoint was chosen for PJOG because the initial values were very similar. The difference in between the two percentages is quite large. However, the overall number of iterations taken by Policy Iteration is always significantly less than that of Value Iteration. The reasoning for this is because Policy Iteration does more work during each iteration and as a result could end up taking more time. The Policy Evaluation step specifically is what takes up a considerable amount of time. However, what it does do well is cut down the action space. This is the strength of Policy Iteration. The strength of value iteration on the other hand is handling larger problems as long as the amount of noise isn't too high. As long as the PJOG value isn't greater than 0.5, Value Iteration outperforms Policy Iteration in both problems. So, without loss of generality, Policy Iteration is more suited to problems with larger action spaces while Value Iteration is better suited to handle problems with larger state spaces.

Some analysis is warranted however in regards to how both algorithms deal with increased amounts of noise. Value Iteration does not handle a large amount of noise well. For Value Iteration, there was a significant jump between the time taken for PJOG values of 0.6 and 0.7. This jump was huge both in time and steps. I believe this to be because of how decisions are made and what the MDP can gather from the information. When things go the way that we want them to with at least 40% certainty, we can still glean some information and go and logically go towards a decision that makes sense. However when only 20% or 30% of our decisions go the way we want them to, and with a wall penalty of 50 which is rather high, it is tough for the algorithm to make a good decision as with a reasonable probability, one could hit the wall but since with an 70% or 80% probability, we may go in any other random direction, there isn't enough certainty in failure to make a good decision. We do see that enough confidence in failure actually yields better results than when we expect to fail 70% or 80% of the time. In fact, the results when we have the greatest amount of noise are actually better than when we can only trust our decisions 20 or 30 percent of the time. The reasoning for this is that at this point, we are confident in our ability to fail. As a result, it is best to try to walk into the wall because it is 9x more likely that you won't actually hit it if you try to. This is the reasoning for this result.

However, Policy Iteration handles noise very well. There is a slight spike in time taken as noise increases but it is nothing compared to that of Value Iteration. I believe this to be because of how Policy Iteration actually works. It does not focus as much on individual states in its iterations but rather focuses on policies. Since it is dealing in the realm of policies, and because Policy Iteration focuses on taking larger steps, when it comes to noise, the problem does not become significantly more difficult. Because it can make a significant change to the policy and try out an entirely new one, it does not have to worry so much about smaller issues and since the policy improves with every step, it will not take an incredible amount of time to converge. In this type of problem, it is easier to change the entire policy rather than change individual values because the actions are not reliable. In addition, the set of policies is finite while the set of values is theoretically infinite. As a result, convergence will take much less time for Policy Iteration vs. Value Iteration when there is a large amount of noise present.

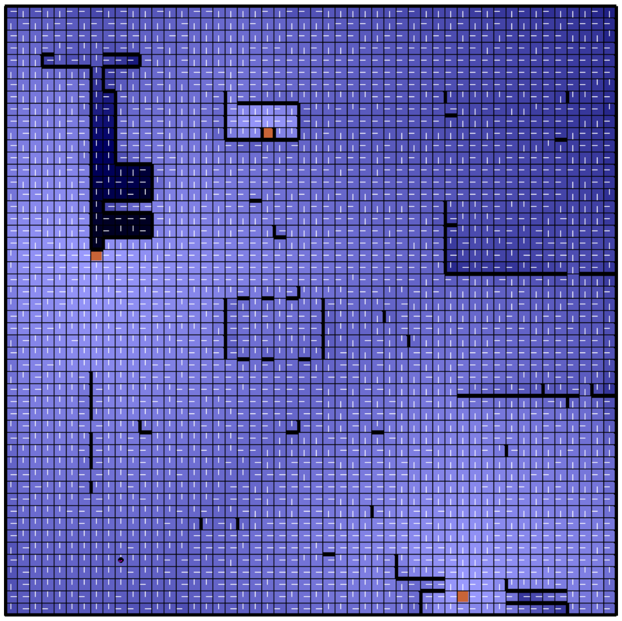
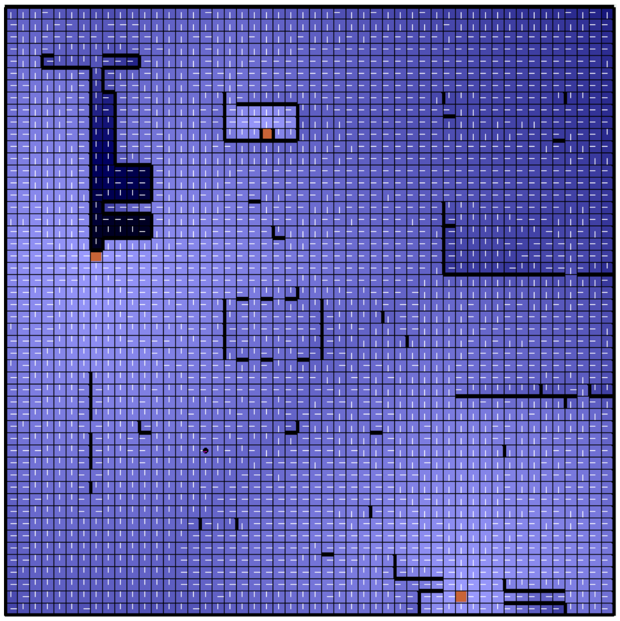
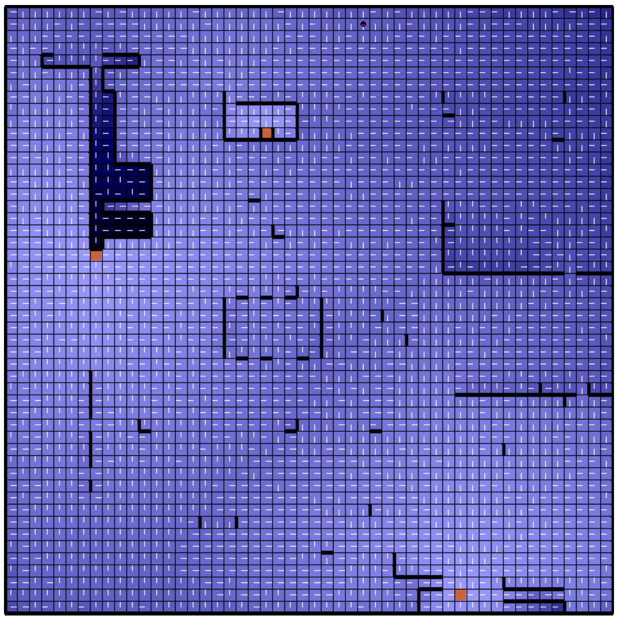
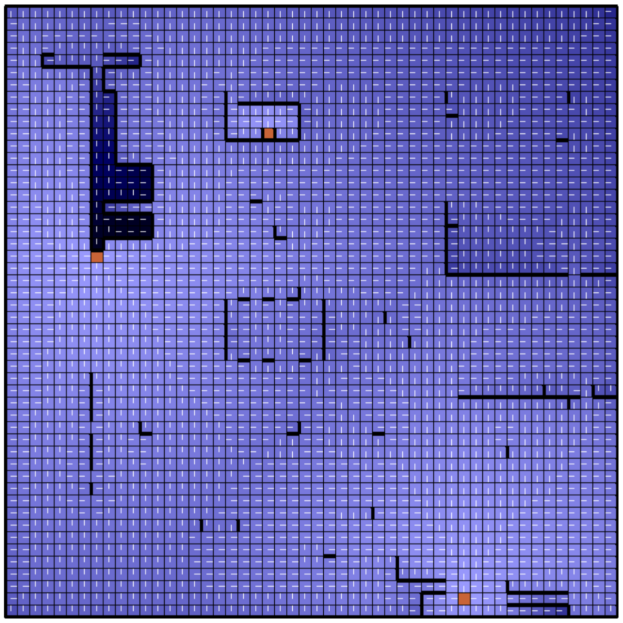
Prioritized Sweeping:

The following is the result of Prioritized Sweeping on the small maze with an epsilon value of 0.1 (top left), 0.5 (top right), and 0.9 (bottom left). The larger the epsilon value, the higher the probability of exploring. The bottom right corner is the optimal policy determined by Value and Policy Iteration:



Looking at the policies that were determined, we see that choosing whether to explore or exploit does not make a significant difference in a maze of this size. With only 100 possible states, this dilemma did not greatly affect the results. Of all three policies which were produced however, the one where we essentially only exploited was the most wrong (though not by much). The policies produced by choosing between exploring and exploiting with equal probability and choosing to explore most of the time yielded virtually identical results (they only differ in one square) and are both very close to the optimal policy. So the conclusion that we can make is that when the state space is small enough, the decision that we make when facing the exploration vs exploitation dilemma does not make a huge difference. Let us see if this is the case for larger state spaces as well.

The following is the result of Prioritized Sweeping on the large maze with an epsilon value of 0.1 (top left), 0.5 (top right), and 0.9 (bottom left). The larger the epsilon value, the higher the probability of exploring. The bottom right corner is the optimal policy determined by Value and Policy Iteration:

Unlike the problem with the small maze, the larger maze has a significantly larger state space. As a result, the decision of whether to explore or exploit did make a substantial difference here. In general, the more the algorithm chose to explore, the better it did as long as it exploited a reasonable amount as well. When the algorithm chose almost exclusively to exploit, it did not perform very well in general. The reasoning for this is simple. Because it exploited exclusively, it did not really learn anything. This caused an erratic policy throughout the maze. This is in a way, analogous to overfitting in supervised learning where we believe our data a little bit too much and don’t explore the possibilities. However, when the algorithm chose almost exclusively to explore, the results were not great either. Close to the goal states, the policy matched the optimal one. However, the further and further we got from the goal state, the more and more erratic the policy was. The reasoning for this is that when we choose to explore exclusively, we may stray from the goal state because we are trying to learn as much as possible and are not actually using what we already know. The results show pretty clearly that in a larger state space, the exploration vs exploitation dilemma truly is something to think about. The best result was produced when exploring and exploiting had equal probabilities. The policy produced here was substantially better because the algorithm was willing to not only learn more but also use the information learned effectively. It is clear that when we explore or exploit too much, our policy will likely not be the best. However, if we find a reasonable middle ground between both, we can find a very good policy, perhaps even the optimal one given enough time.