Abhay Dalmia
April 22, 2018

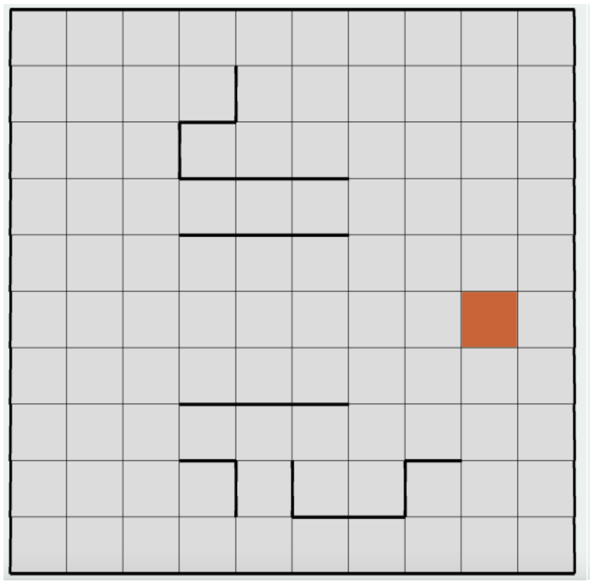
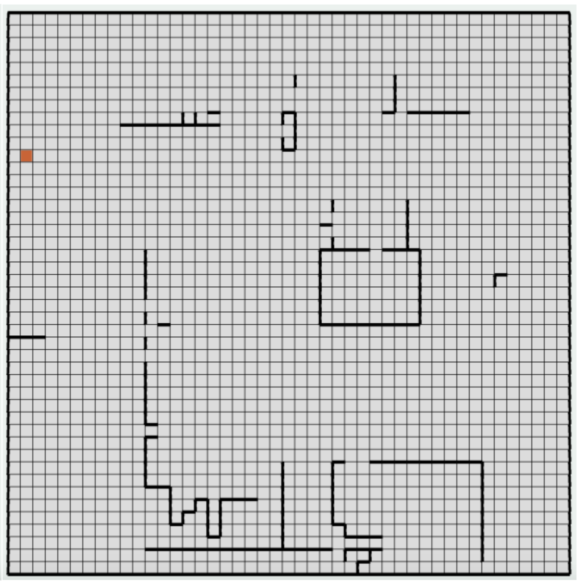
CS 4641 – Project 4 Markov Decision Processes

1. Introduction

I chose to solve the Grid World as the MDP problem. In this problem, there is a grid where one space is the goal state. The search agent mist traverse multiple states while avoiding walls, to reach the goal state. It resembles PAC-MAN closely with the main difference being that there is only one goal state (as opposed to hundreds of foods) and there are no adversarial elements other than the walls. The probability of the agent making the desired move is not 100%. Some amount of randomness is added, that changes this probability of movement.

Although simple, this MDP can be extended to many applications that rely on techniques of solving mazes. An example can be seen with robots that move in factories and warehouses or even an at-home Roomba. The terrain that the vacuum cleaner/robot has to move on is filled with obstacles like walls, tables, chairs etc (similar to the walls in our maze). Additionally, there are several times when the robot may not actually make it's intended move (say moving right instead of front) due to technical glitches. This is represented through the probabilistic action of moving in the maze. Hence the maze problem effectively captures real world problems and allows us to analyze algorithms that potentially apply to these problems.

Autonomous/Self driving cars is another application to which these maze problems can be applied to. In "very simple" words, the goal of an autonomous car is to reach from point A to point B (start to goal state) while avoiding obstacles such as (buildings etc). Though, there is one very important assumption that I am making here: the world I'm talking about is stationary (buildings stay where they are and people are not moving)

To evaluate the performance of policy iteration, value iteration and Q learning (the reinforcement learning algorithm chosen), they will be applied these two mazes. It has a large number of possible states and walls strategically placed to be able to evaluate the three algorithms sufficiently. The larger maze has a very large state space and is more complex than the smaller maze. Hence, it will be a useful and interesting comparison to see how the algorithms perform on these two problems.

When evaluating the algorithms, different values of PJOG were tested. Here PJOG indicates the probability of the agent not moving in the direction of his desired action. So, the probability that the agent will actually take his intended action is 1-PJOG. The number of iterations required for convergence and the time taken to converge was noted for each experiment.

2. Value Iteration and Policy Iteration

Large Maze				
	Value Iteration		**Policy Iteration**	
PJOG	**Iterations**	**time (ms)**	**Iterations**	**time (ms)**
0.01	85	1244	18	12908
0.1	108	1580	14	6509
0.3	181	2586	11	7656
0.6	855	12634	10	13550
0.9	985	14690		

Small Maze				
	Value Iteration		**Policy Iteration**	
PJOG	**Iterations**	**time (ms)**	**Iterations**	**time (ms)**
0.01	18	20	10	46
0.1	28	22	6	38
0.3	61	24	7	25
0.6	304	108	6	21
0.9	480	125	7	17

The value iteration algorithm in a nutshell helps us find the utility of each state where utility is defined as the reward for being in that particular state plus the rewards that we are going to get from that point on (capturing the rewards and utilities for the future states/actions to be taken). While the utility iteration algorithm gives us the optimal utility for each state, the policy iteration algorithm helps us to map a particular state to a desired action.

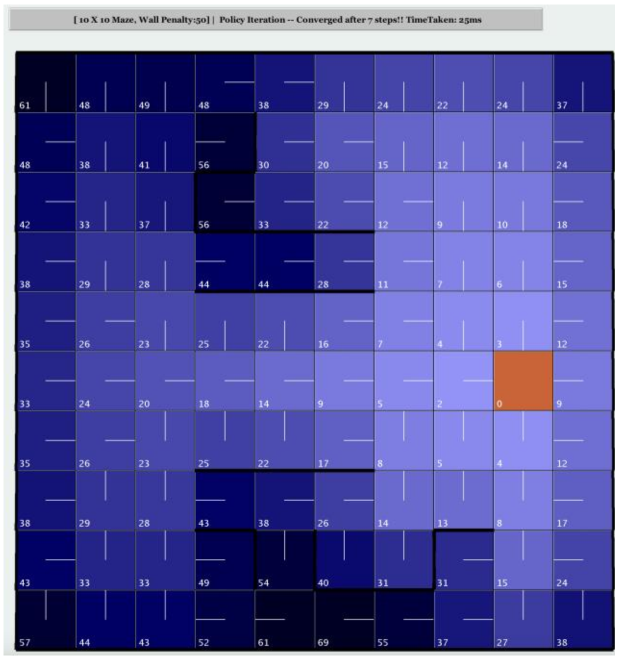
Fist, for value iteration, a conservative value of 0.3 was chosen as the randomness factor (PJOG) and a penalty of 50 was assigned for crashing into the walls. As expected, Value Iteration prefers paths without any walls assigning the states to the north of the goal state much lower penalty values than the states on its west and south. This is most likely due to the algorithm trying to avoid the walls and wall penalty, considering the randomness factor. This case took 2586ms total to run and completed in 181 steps.

When the randomness factor is reduced to 0.01, we see that the algorithm favors the paths through the walls much more than the previous case. This can be attributed to the fact that low randomness almost always guarantees the next state to be the desired one. This reduces the penalty that could have been incurred when you hit the wall and thus Value Iteration is able to give much lower costs to the states that are closer to walls. It is also worth noting that with PJOG values Value Iteration converges much faster probably because the algorithm doesn’t have to spend too much time exploring the non-optimal states. Value Iteration for this case converged in 19 steps and 13ms, considerably faster than first case.

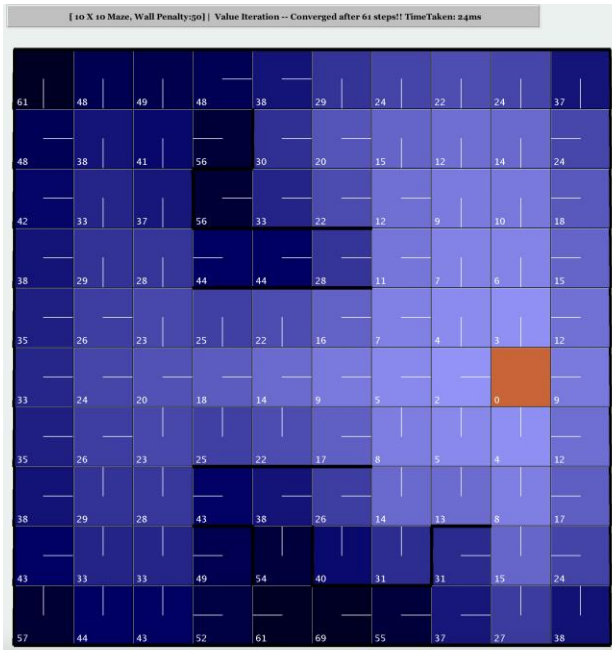
When the PJOG factor is increased to 0.9, which means that 90 percent of the time the next state is not the intended state. In this case we see expected behavior, the algorithm brutally punishes any states that are remotely close to the walls and gives much smaller rewards for being close to the goal state. It is easy to conclude that with high randomness severely affects the algorithm leading to Value Iteration aggressively punishing most states. It is also worth noticing that high variability significantly impacted the runtime of the algorithm as Value Iteration converged after 1220 steps or 665ms, almost 10 times the regular case.

It is evident from the table that policy iteration takes much fewer iterations to converge when compared to value iteration. This is true for both the mazes. This can be explained from the fact that the policy iteration algorithm terminates after an optimal policy has been achieved (i.e further iterations do not change the policies- although the utility values may not yet be optimal). Value iteration on the other hand terminates only when the difference between the utilities of succeeding iterations is below a certain tolerance. Thus, even though the correct ordering of actions may be achieved earlier, the utility of each state could still further be improved. In other words, even though the actions taken by an agent in a particular state may not change, the utility value for that state would continue to improve (until convergence). Thus, more iterations are required for value iterations.

Now although the number of value iterations required to converge are much more than the number of policy iterations, the total time taken by value iterations is much less compared to the time taken by the policy iterations. This shows that the policy iteration update is more expensive than the value iteration update. It can be attributed to the fact that for value iteration we are only updating value/numbers for a particular state (based on reward of current state and utility of neighbor states). But for policy iterations we are actually calculating the utility of a particular state based on a given policy and then using it to improve the current policy/state action (by evaluating the argument max of all the possible actions for a given state that maximizes the expected utility calculated in the previous timestamp). In simpler terms it is doing policy evaluation and improvement.

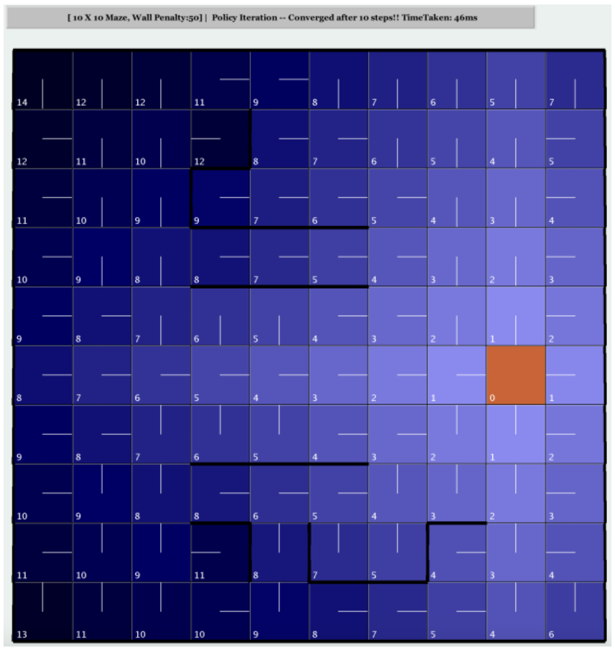

[10 X 10 Maze, Wall Penalty:50] | Policy Iteration -- Converged after 7 steps!! TimeTaken: 25ms

Policy Iteration with PJOG = 0.3

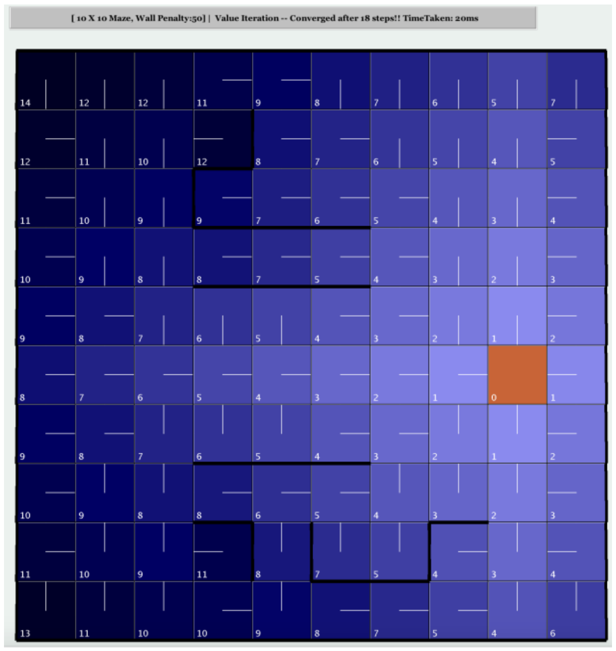

[10 X 10 Maze, Wall Penalty:50] | Value Iteration -- Converged after 61 steps!! TimeTaken: 24ms

Value Iteration with PJOG = 0.3

When comparing policy iteration and value iteration or a conservative PJOG value, the results are quite consistent. the algorithms seems to be a little conservative and prefer the path that goes around the walls even when there is a shorter path that cuts through a walled corridor. This behavior can be attributed to the randomization factor which makes it riskier to traverse through the states closer to the walls as the agent is more likely to lose points there.

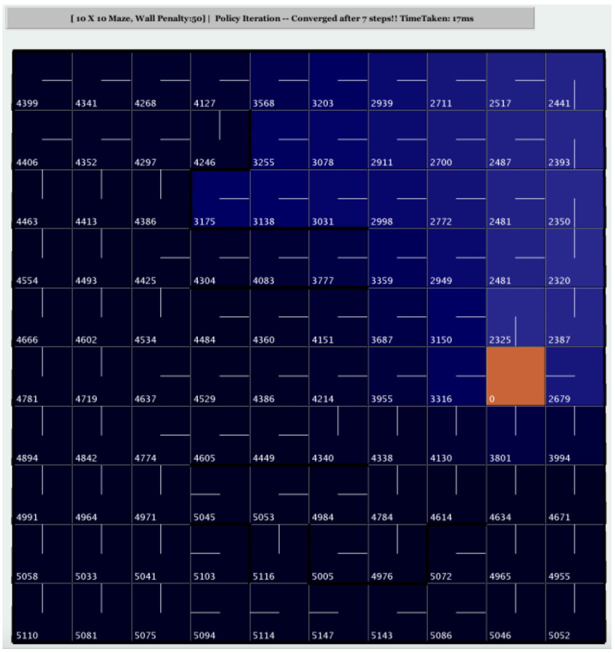

[10 X 10 Maze, Wall Penalty:50] | Policy Iteration -- Converged after 10 steps!! TimeTaken: 46ms

Policy Iteration with PJOG = 0.01


[10 X 10 Maze, Wall Penalty:50] | Value Iteration -- Converged after 18 steps!! TimeTaken: 20ms

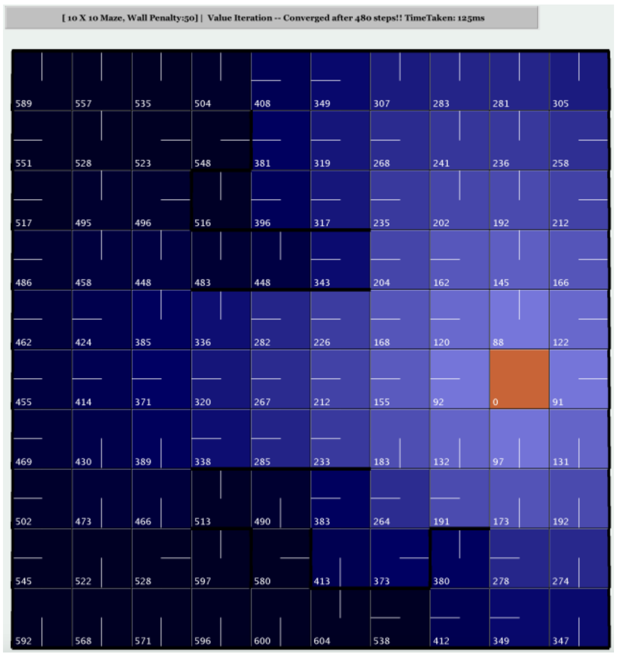
Value Iteration with PJOG = 0.01

With a reduced randomness factor (PJOG = 0.01), we can see the algorithm come up with very different policies as compared to the previous test. The algorithm now punishes paths that are indirect and now encourages direct paths even if they go through the walled corridor. This finding is consistent with the Value Iteration test results and this can also be attributed to the fact that low randomness makes the next optimal state much more reliable enabling algorithm to pick out certain riskier paths.


[10 X 10 Maze, Wall Penalty:50] | Policy Iteration -- Converged after 7 steps!! TimeTaken: 17ms

4399	4341	4268	4127	3568	3203	2939	2711	2517	2441
4406	4352	4297	4246	3255	3078	2911	2700	2487	2393
4463	4413	4386	3175	3138	3031	2998	2772	2481	2350
4554	4493	4425	4304	4083	3777	3359	2949	2481	2320
4666	4602	4534	4484	4360	4151	3687	3150	2325	2387
4781	4719	4637	4529	4386	4214	3955	3316	0	2679
4894	4842	4774	4605	4449	4340	4338	4130	3801	3994
4991	4964	4971	5045	5053	4984	4784	4614	4634	4671
5058	5033	5041	5103	5116	5005	4976	5072	4965	4955
5110	5081	5075	5094	5114	5147	5143	5086	5046	5052

Policy Iteration with PJOG = 0.9


[10 X 10 Maze, Wall Penalty:50] | Value Iteration -- Converged after 480 steps!! TimeTaken: 125ms

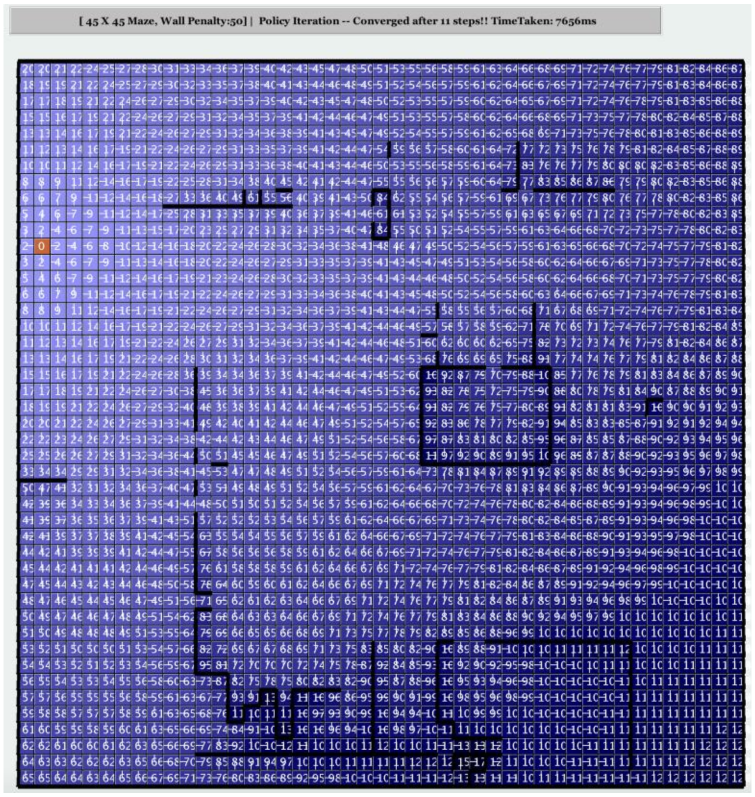
589	557	535	504	408	349	307	283	281	305
551	528	523	548	381	319	268	241	236	258
517	495	496	516	396	317	235	202	192	212
486	458	448	483	448	343	204	162	145	166
462	424	385	336	282	226	168	120	88	122
455	414	371	320	267	212	155	92	0	91
469	430	389	338	285	233	183	132	97	131
502	473	466	513	490	383	264	191	173	192
545	522	528	597	580	413	373	380	278	274
592	568	571	596	600	604	538	412	349	347

Value Iteration with PJOG = 0.9

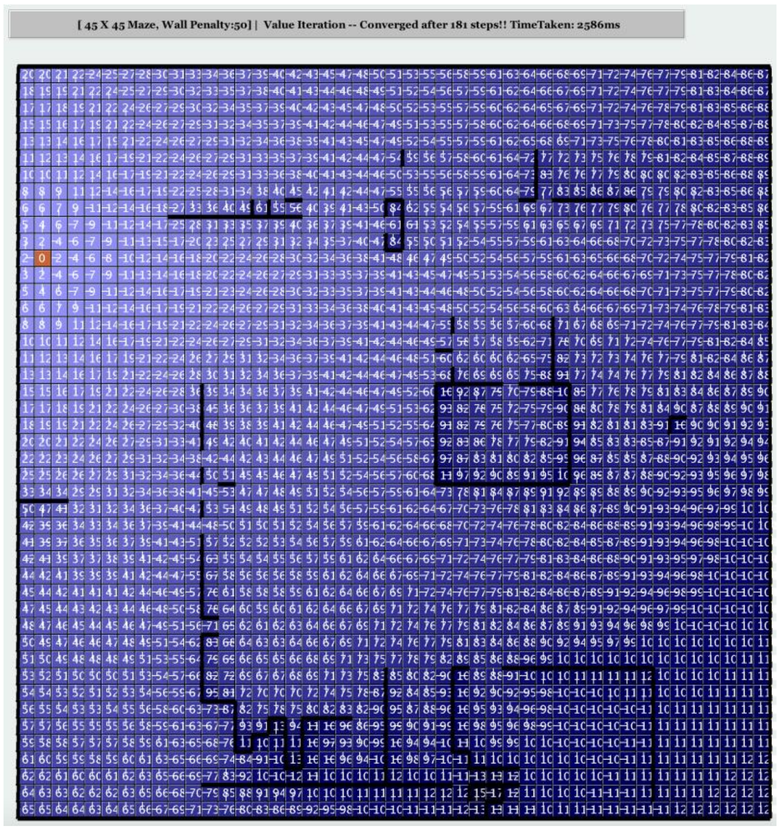
With a very high randomness value (PJOG = 0.9), Policy iteration attaches very high costs with all the states that are 2 steps away from the goal state. This finding is also consistent with the previous Values Iteration finding and intuitively makes sense. A PJOG = 0.9 value means that the next state is almost guaranteed to be suboptimal and thus makes the algorithm perform poorly. Interestingly, with a high randomness factor Policy iteration converged in 10 steps with results that are similar to the Value Iteration. Thus, it can be concluded that for relatively noisy environments it would more advantageous to use Policy Iteration to get a faster result in less steps.

We can also see from these side-by-side comparisons, that while the results and very similar for the PJOG = 0.3 case, for both algorithms, the results vary more for the PJOG = 0.9 case. (can be seen by the difference in color gradients/concentrations).

This analysis holds true for the larger maze sizes. It is just harder to analyze due to the size and complexity.

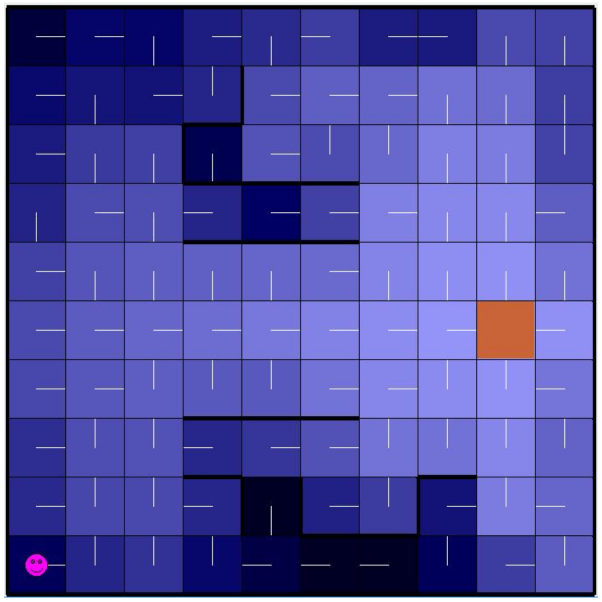

[45 X 45 Maze, Wall Penalty:50] | Policy Iteration -- Converged after 11 steps!! TimeTaken: 7656ms

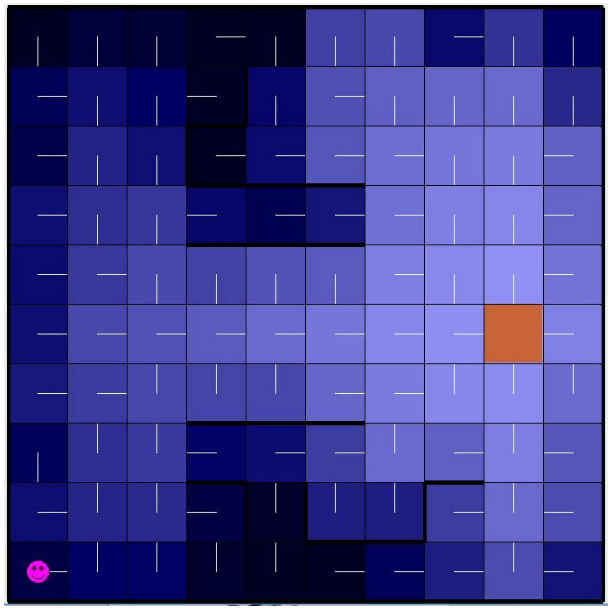
Policy Iteration with PJOG = 0.3

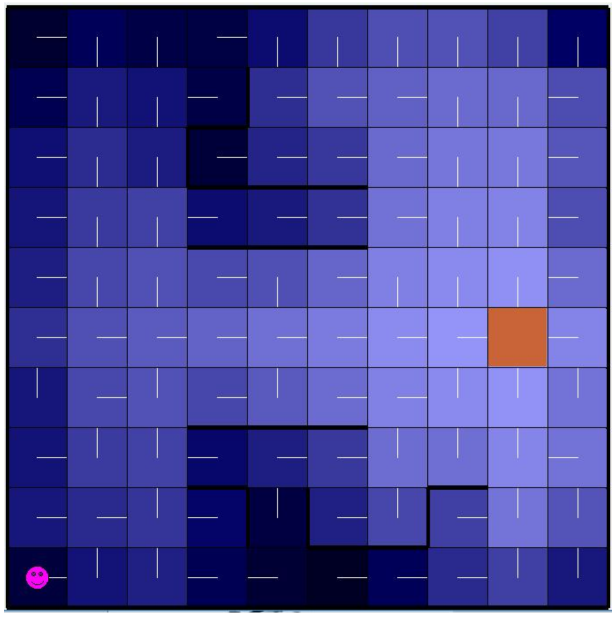

[45 X 45 Maze, Wall Penalty:50] | Value Iteration -- Converged after 181 steps!! TimeTaken: 2586ms

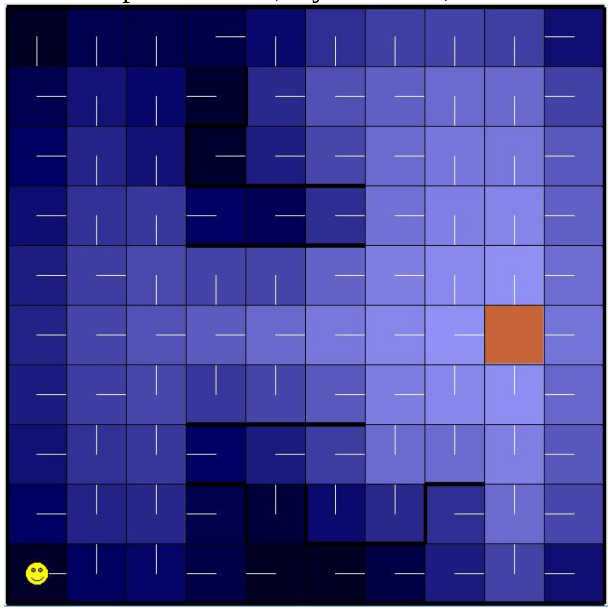
Value Iteration with PJOG = 0.3

2.3 Q Learning


Epsilon = 0.1, Cycles = 25,000


Epsilon = 0.4, Cycles = 25,000


Epsilon = 0.8, Cycles = 25,000


Epsilon = 0.8, Cycles = 250,000

For this part of the assignment we are comparing a reinforcement learning algorithm to the previously used value and policy iteration algorithms. The reinforcement learning algorithm that I chose was the Q-learning algorithm. While performing the value and policy iteration algorithms we assumed that we had access to the transition model and rewards for a particular state. However, in the real world we may not have to such information. Reinforcement algorithms like Q-learning make use of this available information to come up with solutions which may or may not be optimal but more accurately represent real word situations.

Learning rate and epsilon are two important parameters that are used in the Q-learning algorithm. Learning rate represents how much of the old data we use along with the new data to estimate the “Q-value” for a particular state. We start with a high learning rate (learning more from new data) and then gradually decrease the learning rate over time (relying/believing more in past data). Epsilon could be viewed as the measurement of randomness (similar to simulated annealing) where we tend to use the more optimal solution most times (with a probability of 1-epsilon) but other times (probability epsilon) we may use a suboptimal solution. This is done to avoid situations like being stuck in some local minima. It also brings forth the concept of exploration vs exploitation which is an important dilemma for Q-learning and most other reinforcement algorithms. While performing the Q-learning algorithm experiments different values of epsilon were used to compare and contrast results. The learning rate was 0.7 and the PJOG value was kept at 0.3 to keep an even basis for comparison with previous results.

It is interesting to note the results obtained after running the Q-learning algorithm on the problem sets above. The first thing to note is that none of the results (for both problems) matched that of value or policy iteration. This is because we have limited the number of iterations that are performed for Q-Learning. For Q-learning to return an optimal solution the states have to be visited an infinite number of times. It should be noted that the results for the smaller maze are however more similar to the value/policy iteration results than the Big maze results. This can be attributed to the fact that the smaller maze is much simpler (fewer states and less obstacles), and thus is easier for the Q-Learning algorithm to map the given transitions to policies. Another observation, was that for both the small and the big maze the Q-Learning algorithm took much longer to execute than the value and policy iteration. This follows from the fact that we are not given a model and the reward for each state but only transitions to work with. We need to visit each state multiple times (well infinite for optimal) to obtain a good expected Q value for each state and then chose the appropriate action for each state based on these Q values.

Q Learning also took the most time out of all three algorithms, and never managed to converge to the optimal policy (although it did come close). The 1000 iterations run took about 2 sec, which puts its time at about 2ms per cycle. The time taken each step is significantly lower because in this implementation each step is the time taken by the agent to reach destination state from the starting state. Although Q Learning takes the most iterations it does enable the agent to truly learn from its surroundings and past mistakes and eventually reach the optimal policy. Which is a very impressive feat given that this algorithm requires no domain knowledge.