Assignment 4: Markov Decision Process

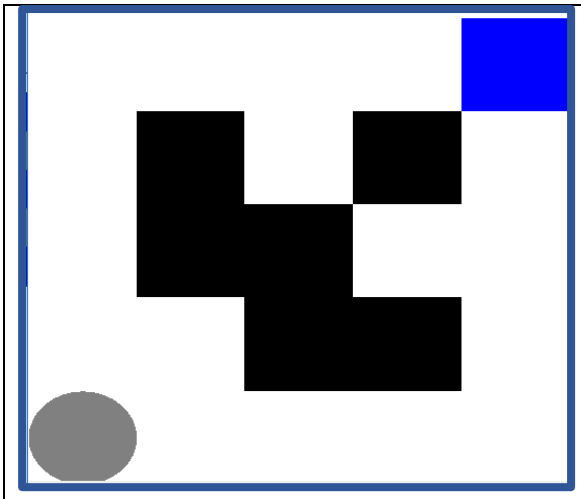
Man Wai Yeung
CS 7641: Machine Learning

Introduction

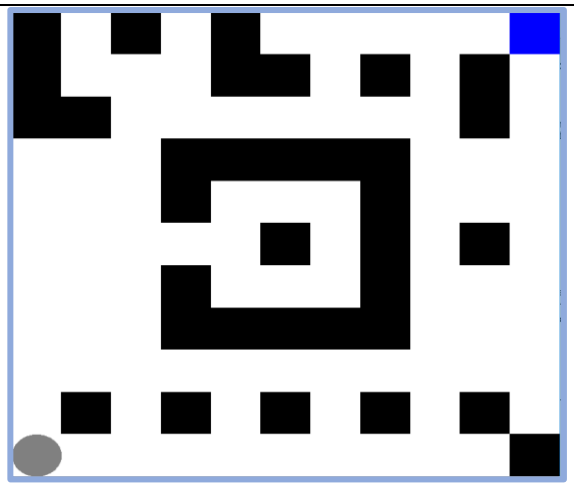
This analysis focuses on how to implement reinforcement learning experiments on two grid world problems and some remarks from experimental designs. The algorithms implemented include two using Markov Chain: value iteration and policy iteration. There is another one using reinforcement learning algorithm: Q-Learning. I will focus on comparing and contrasting the results obtained using the three algorithms and how varying discount rate (gamma), number of iterations, different policies affect the experimental result. I implemented the algorithms using BURLAP package using Java.

Environment

The three algorithms will be used on one small 5x5 grid world and one large 11x11 grid world.



Small Grid World



Large Grid World

At every trial, the grey dot (agent) starts at the bottom left corner and the end state is located at the top right corner (in blue) where the agent will receive 100 points. Every step the agent takes will cost 1 point. Black blocks are walls that the agent cannot step on.

For the small world, there are a total of 19 states. For the large world, there are 87 states.

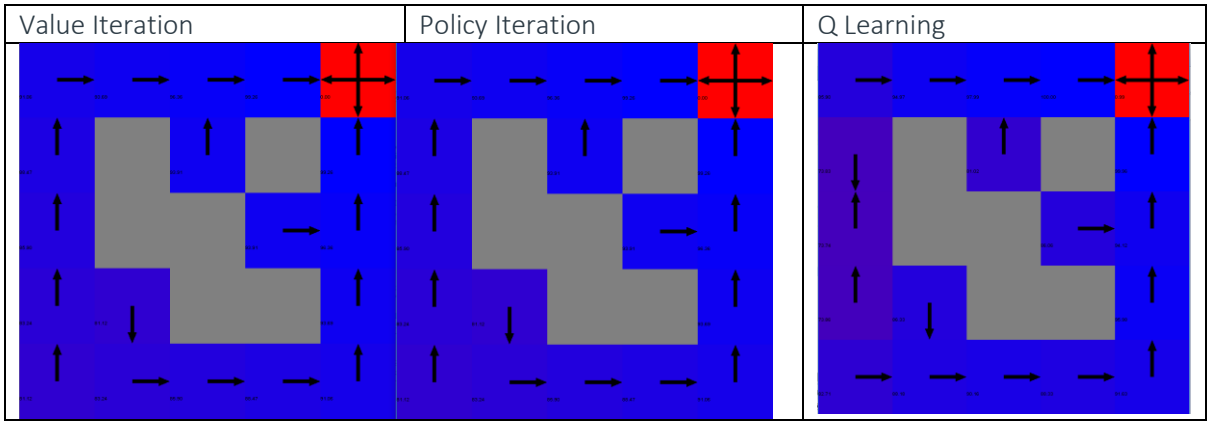
Where are these two grid worlds interesting:

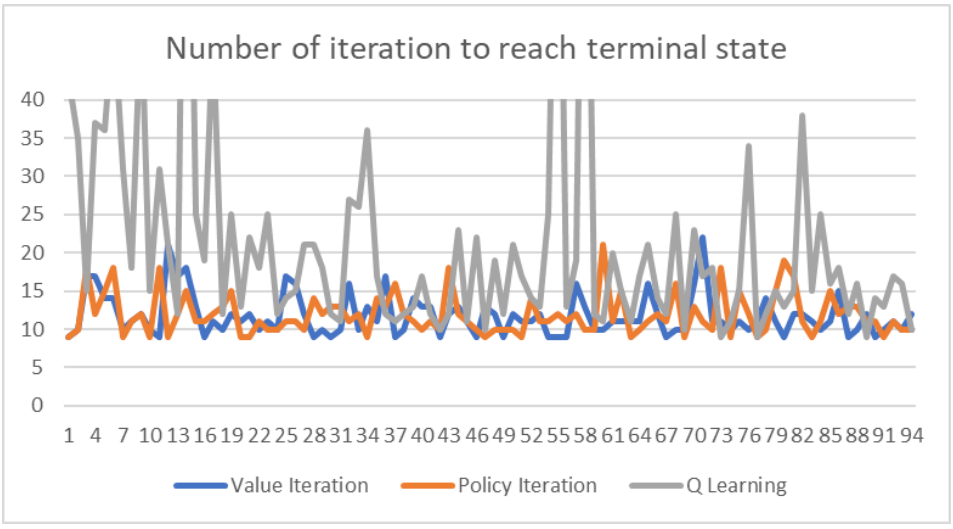
Drawing some inspirations from maze and Pacman, I have designed my grid world to be somewhat challenging. Small grid world contains dead ends and corners that I would like to see how the agent reacts to get out of once it steps into those locations. For the large grid world, I have created a trap in which once the agent steps in there's only one direction that can get out. I want to see how the agent can learn and discover that one-way exit route. Besides the trap, there are several dead ends scattered across the map which makes reaching the destination more difficult than an empty world.

What we learn from Small Grid World

As a baseline experiment, I ran all three algorithms for 100 iterations:

Insight 1: We can see that **both value iteration and policy iteration returns the same policy** while Q-Learning (using a randomized exploration policy) has different policy directions for the leftmost grids with conflicting directions, meaning it has not yet converged, which means that it would require more iterations to converge to a policy similar to value iteration and policy iteration. Theoretically, the result follows the theory we learn in class, since value iteration and policy iteration are markov chain process which has known states, the algorithm already learns a lot about the "grid world" and the transition matrix before it begins to run, while the Q-Learning algorithm learns about the world while it begins iterations, it continues to modify its assumptions about the world. For that reason, it would require more iterations to explore while fully exploit each states.



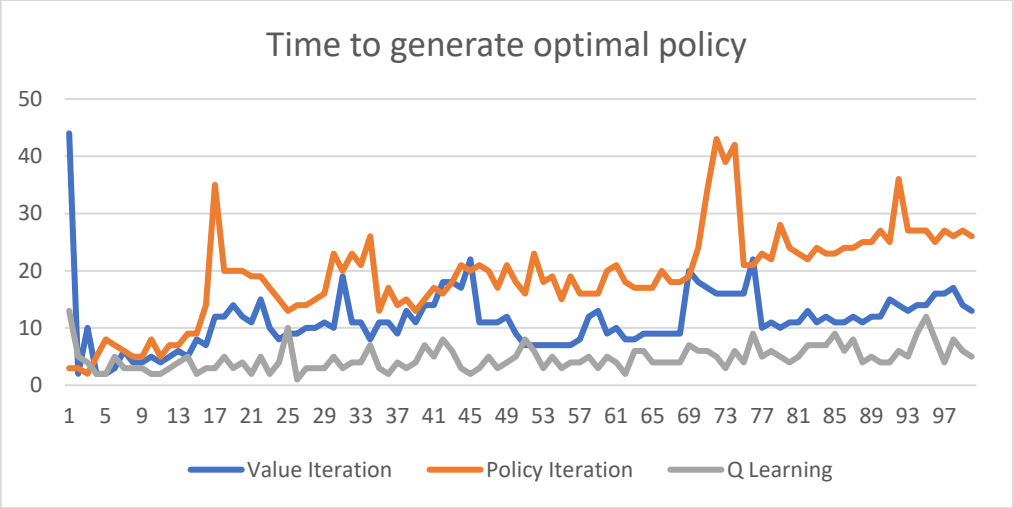


Iterations 1-6 : Iterations required to reach the final state

Iterations	Value Iteration	Policy Iteration	Q Learning
1	1989	37	143
2	23	699	113
3	11	14	35
4	10	16	35
5	12	10	24
6	9	14	68

For ease of visualization, I clipped the chart y-axis at 40 and starting only at iteration 7 since the beginning iterations are funky. Especially for value iteration 1, it takes 1989 iterations before reaching the end state!

Insight 2: **But quickly after the funky experience, the following iteration for value iteration and policy iteration becomes very stable ranging between 10-20 iterations.** I think this is because the value gets updated by each step it takes during the iteration, by taking so many steps early on, it is able to learn a lot of information during those 600/2000 steps in order. So afterwards, it is just the random probability to jump in another distribution that is at work that leads to variation in no. of iterations to reach another state.

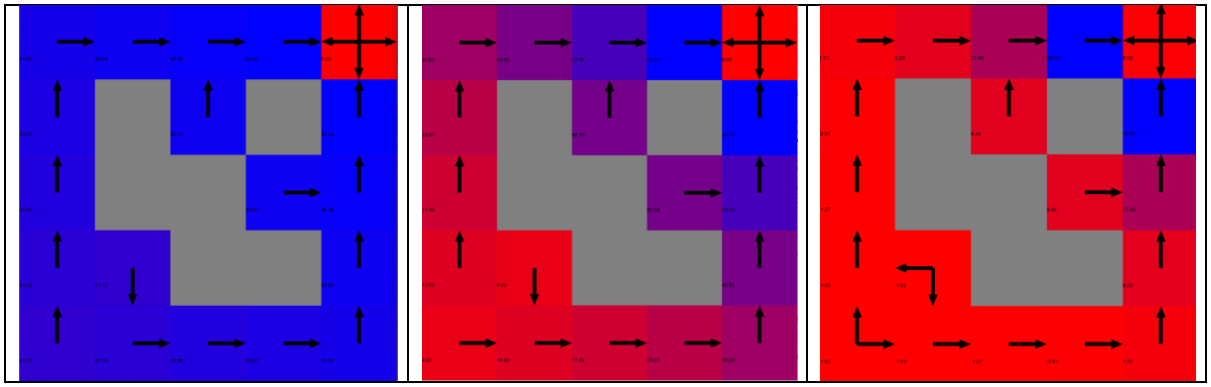


Insight 3: **Compared to value iteration, policy iteration requires more computing time per iteration**. I think it is because in each move, it directly update the policy and recurse backwards where the value iteration updates at the end and generates the policy afterwards. Because of the difference in the approach, policy iteration takes more time to generate optimal policy than value iteration.

Insight 4: **Q-learning takes half the time than policy iteration to generate optimal policy.** I think it is because the optimal policy is well-defined by the equation. So when the Q-function gets updated, the policy is just taking the direction that has highest Q-function value without recalculation.

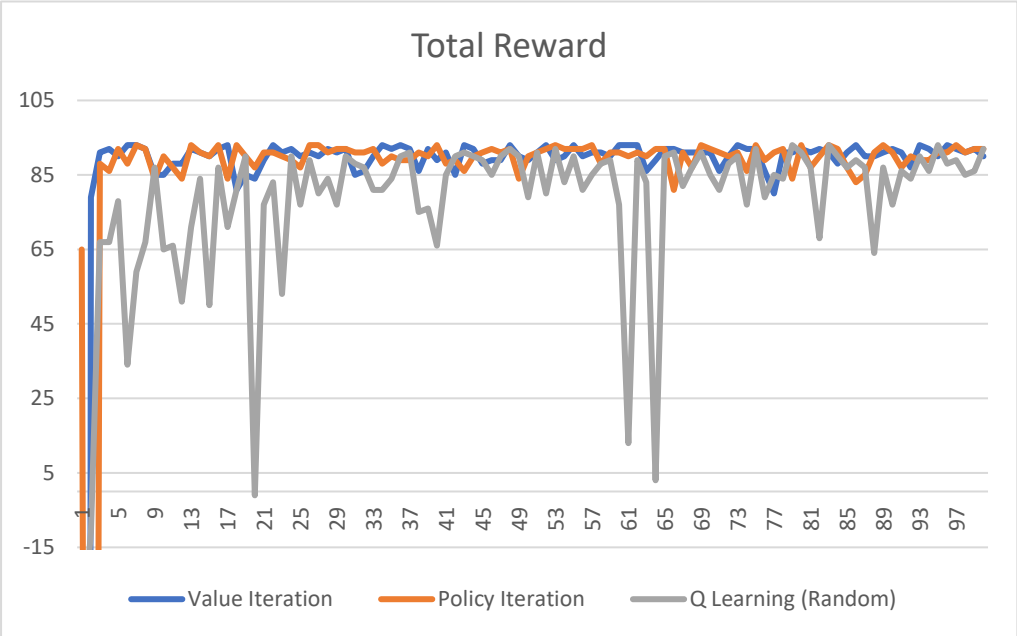
Insight 5: What happens if I vary the discount rate? **The value function of the best direction decreases as discount rate decreases but the policy remains the same.** Comparing to Gamma of 0.99 and 0.8, we can see the beginning state (bottom left) tiles have different direction, which makes sense in terms of the update. Lower gamma means that the function puts more weight on short-term gains. Since the bottom tiles are furthest from the reward state, reward gained at the end discount by a lot and decrease the value function at the tile. When the end state does not matter as much, value functions between one state to another becomes an approximation of nearby state only and ends up being very similar to each other and fail to distinguish the best policy as shown in gamma = 0.99

Gamma = 0.99	Gamma = 0.8	Gamma = 0.3



Insight 6: **Different exploration policy yields different policies using the same number of iterations.** Boltzmann Policy and Epsilon Greedy both are more likely to step into the direction with higher Q-value. Random jump follows a uniform distribution at all directions. With just 200 iterations, we can see that all policies are different. However, Boltzmann Policy yields the best results (most similar to policy from value iteration / policy iteration). I think this is because given the small number of distribution epsilon greedy algorithm have not fully update the probability distribution. As for random jumps, it produces conflicting direction in the policy which doesn't really make sense, which implies the learning is far from converging. Boltzmann Policy seems to explore more and able to update Q-value more quickly to converge to the best policy.

<NEED TO CREATE THE TABLES HERE>

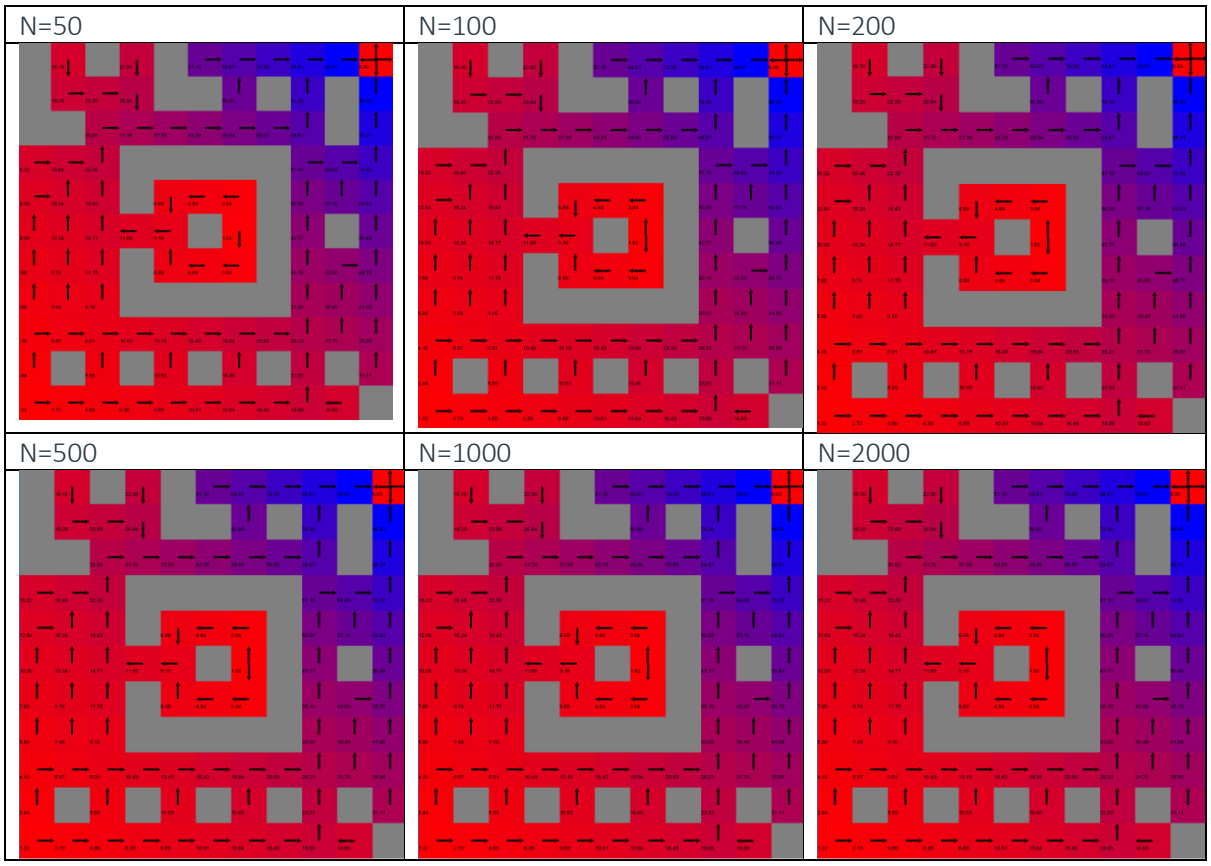


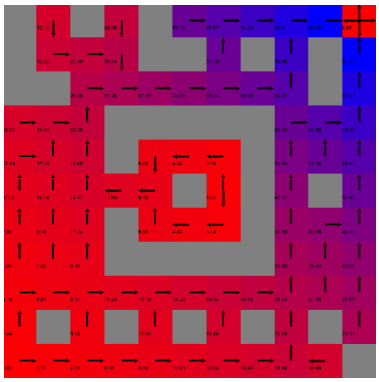
Insight 7: **Q Learning has more fluctuations in the rewards between iterations compared to Value Iteration and Policy Iteration.** I think this is due to the fact that I am using a random jump policy which multiple bad direction would trap the agent in a cycle even if the probability distribution has converged. In comparison, value iteration and policy iteration are able to learn from early iterations and earn rewards of about 90 upon reaching the end state, which means that within 19 available states, it only take 10 steps across these states before reaching the final state.

Large Grid World

Using the large grid world, which means a more complex problem, I conduct another series of experiment to see how algorithms respond.

Insight 1: **For Value Iteration and Policy Iteration, optimal policy converges before 100 iterations**, and they reach the same optimal policy.

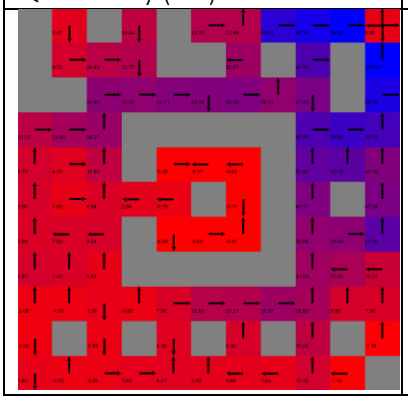
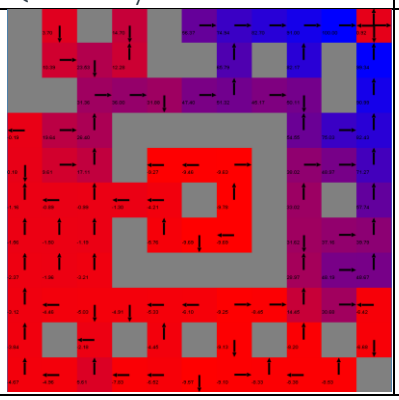
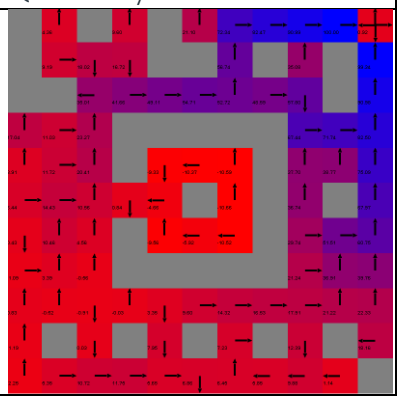




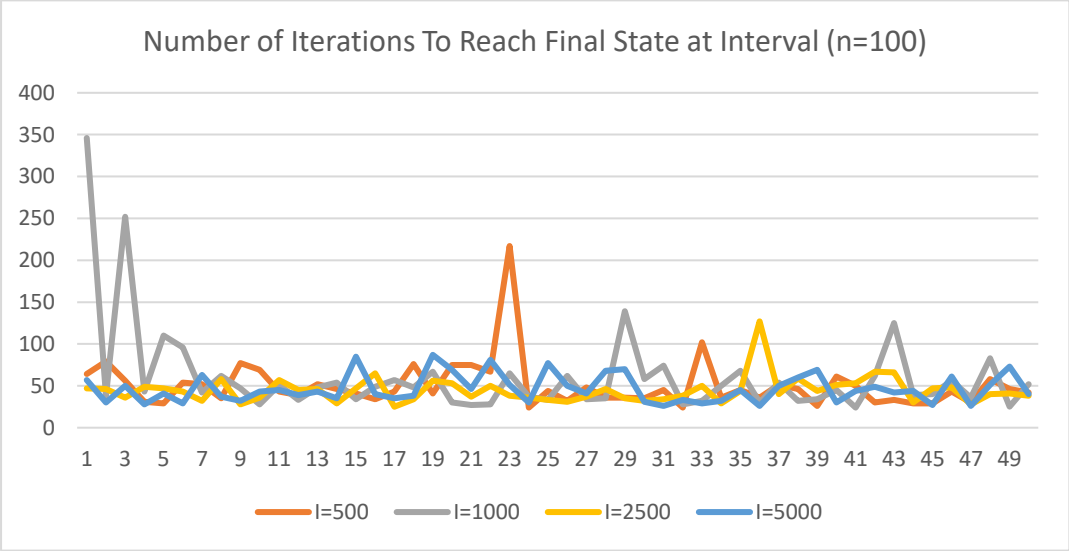
Policy Iteration (100) optimal policy is the same as that of Value Iteration in the above grid

Insight 2: **For more complex problem, Q-Learning takes exponentially more iterations to converge.** In the small grid world, Q-learning takes about 200 iterations to converge to the same optimal policy as MDP algos. In this case, even 1000 iterations or 5000 iterations out, the optimal policy still has same problems in finding the best policy. For example, looking at the bottom row, while the optimal policy is to go to the right (except the grid next to the barrier at the bottom right corner), for Q-Learning, the directions are conflicting with some pointing up and one down, which doesn't make sense intuitively. The reason behind its slow converge is understandable however since there are a lot more directions to explore given the the number of states increase by 8 times. While we allow some degree

of exploration, it still takes a lot of time for algorithm to fully retrieve information about the world to be certain about optimal policy.

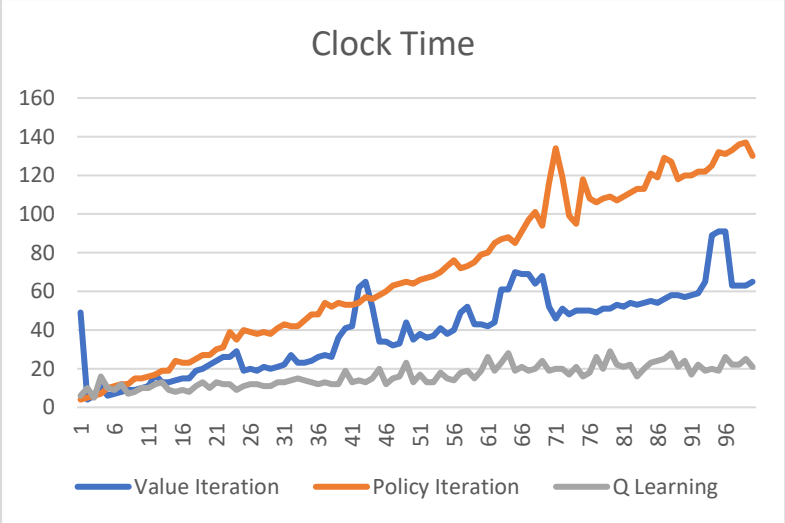
Q-E-Greedy (0.3)-500	Q-E-Greedy-1000	Q-E-Greedy-5000
		

Insight 3: I compare the number of iterations to reach final state at 100 intervals compared across Q-learning at different intervals in the line charts.

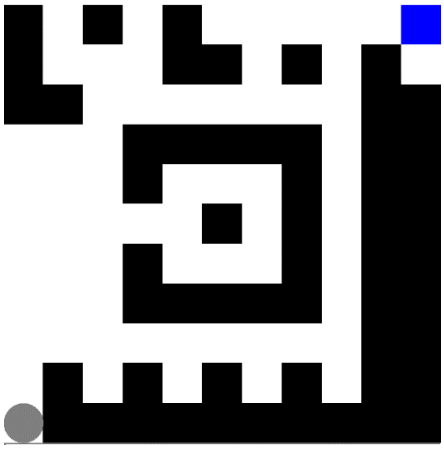


I was expecting to see that the number of iterations would be lowest with 5000 iterations. However, the result looks quite similar across all four iterations. I think the reason is because my epsilon-greedy strategy with epsilon of 0.3 still contains a bit of exploration so while the transition matrix is not fully converged, the agent behaves like quasi-random jumps still, similar to what algo with smaller iterations. It is interesting to see that in the first 150 iterations for 1000 iterations run, some iterations takes up to 350 iterations to reach final state. Albeit such long tour, the algo was able to explore a lot of information and bring down the number of iterations needed down quickly.

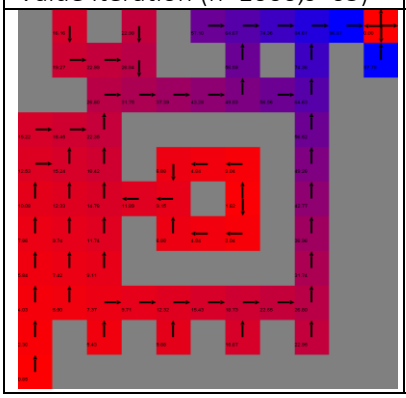
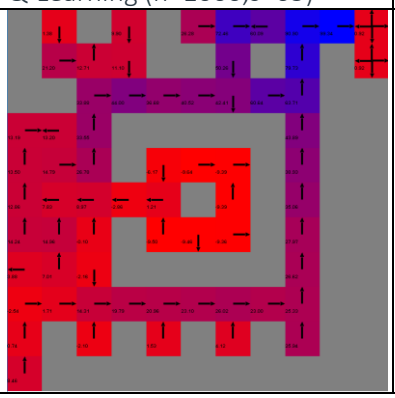
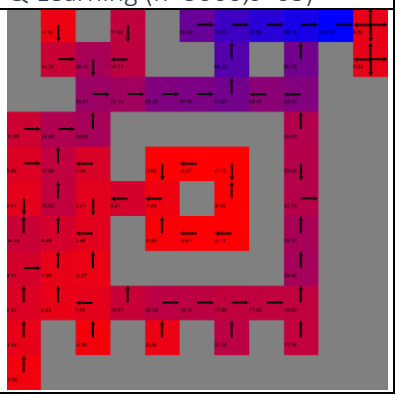
Insights 3: In terms of computational time, Q-learning again is the most efficient, taking only half of the time that of value iterations and 1/6th of the time of policy iteration, which supports my previous analysis in simple grid world.

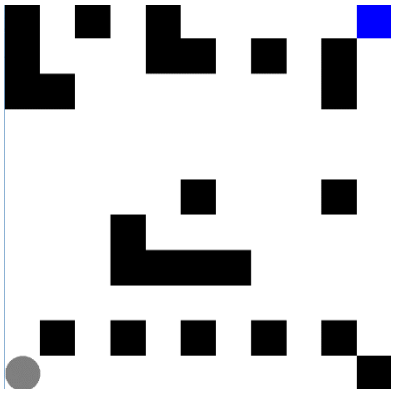


The last thing I want to explore is how does the number of state affects the speed of convergence:

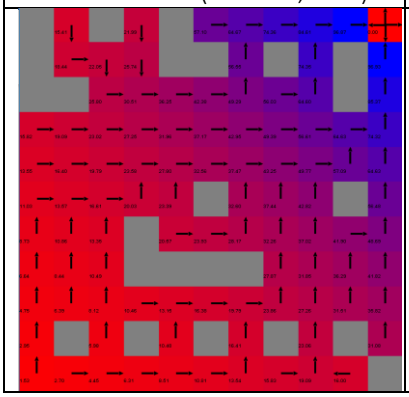
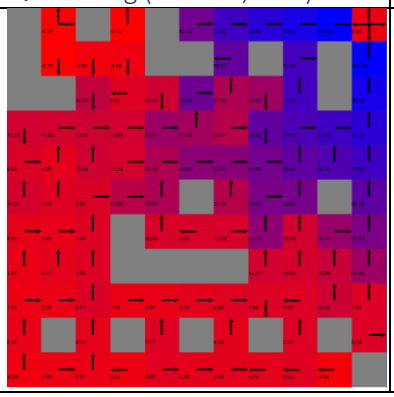
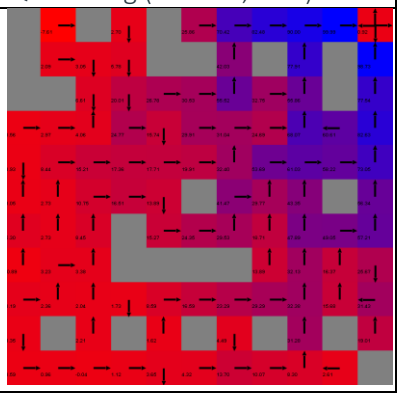


I decreased the number of states from 87 to 65 by blocking the bottom row and the column to the right. By blocking of some barriers, this problem is an easier path to get to the agent.

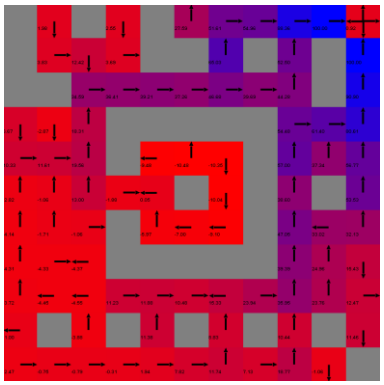
Value Iteration (n=1000,s=65)	Q-Learning (n=1000,s=65)	Q-Learning (n=5000,s=65)
		



To the other direction, I increased the number of states from 87 to 97 by removing the death trap in the middle of the grid world. Since the middle trap is harder to learn, this is also an easier problem.

Value Iteration (n=1000,s=97)	Q-Learning (n=1000,s=97)	Q-Learning (n=5000,s=97)
		

Comparing with the original problem, it seems that with more or less states, the Q-Learning algorithm has not converged to the same policy as value iteration. Even though the general direction looks correct, some arrows still points to the border or to the wall. **It seems that the complexity for the optimal policy has not moved its needle much by increasing or decreasing the number of states by 15 states.** I think it is because computationally it is similarly complex with a lot of information unlearnt. The only way for the optimal policy to converge is to increase the iteration to sufficiently large.



Q-Learning with 50000 iterations for the original grid world problem, even then the arrows furthest away from the final state can give the wrong direction.

All in all, by experimenting different policies, different discount rates, varying number of iterations, it helps me to understand how reinforcement learning and MDP algorithm updates to find the optimal solution. I think Q-Learning is a very powerful tool despite not knowing much initially, by setting the right balance between exploration and exploitation, it can successfully know enough of the world and identify the best actions. I would like to see how Q-learning can be applied into multi-agent scenario or dynamically changing environment where updating the actions will be more complex.