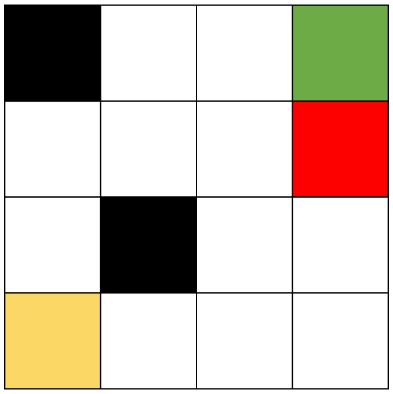
GTID: jliu727

# Assignment 4: Markov Decision Processes

**Description of MDPs**:

**MDP1** (small): This small MDP problem is a small maze similar to the one we learn in the lecture. It is a simple 4x4 grid world and contains four possible types of grid: start grid (yellow), goal grid (green), wall grid (black), danger grid (red), normal grid (white):



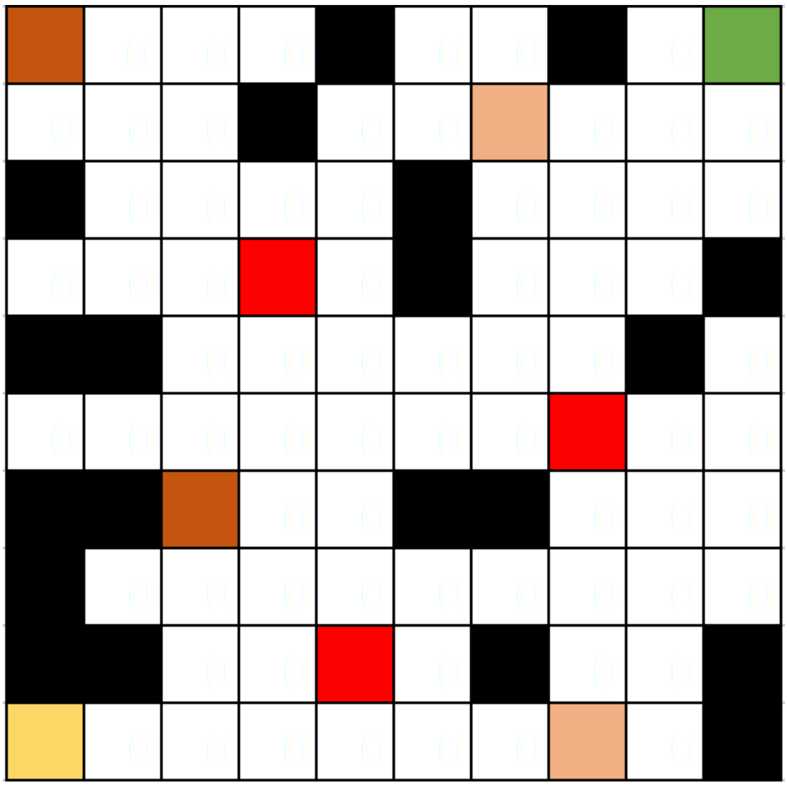
I set the rewards for each type of gird to be the following: Danger gird = -200, Goal grid = 50 and Normal grid = -0.1. We can simply read from the graph the the best situation would be to avoid getting close to the danger grid and take the up-up-right-up-right-right sequence which would be 7 steps including the start position.

This maze is stochastic that at each step there is 80% of chance going to the intended direction and 20% of the chance splitting between other three directions. If the action hits a wall, it would remain in the same grid.

Grid World Problem is interesting in that it is a simplified version of many realistic problems such as finding the shortest road. It has a good visualization of how the steps are taken and decisions are made while still applicable to more complicated problems.

**MDP2** (large): This large MDP problem is a large maze developed from the basis of the small MDP. It is a 15x15 grid world with more types of grid: start grid (yellow), goal grid (green), wall grid (black), large danger grid (red), medium danger (dark orange), small danger (light orange) normal grid (white), we can clearly see from the maze that it is too complicated for us to visually determine the best policy.

Here the reward for each type of grid are: Large Danger = -3, medium danger = -2, small danger = -1, goal = 10 and normal grid = -0.1. It is still stochastic with 80% of chance going to the intended direction. This problem is interesting because it is the same type as the small MDP problem we have and could serve as a good contrast to learn methods on different size of MDP problems.



**Methods Used:**

Value Iteration:

In value iteration, we know the reward and model for each state and the goal is to find the utility in optimum policy. The value for each grid is calculated by:

- Start with arbitrary utility
- Update utility based on neighbor utilities
- Repeat until it converges

This would yield the utility in optimum policy and we can derive the policy from there.

Policy Iteration:

In Policy iteration, we also know the reward and model for each state. The difference is that we start from generating a random policy and repeatedly update the policy to eventually find the optimum. It should in theory be faster in total time compared to value iteration as it does not calculate all utilities. It is calculated as following:

- Start from a random policy
- Evaluate the utility given the policy
- Improve the policy based n updated utility
- Repeat until converges

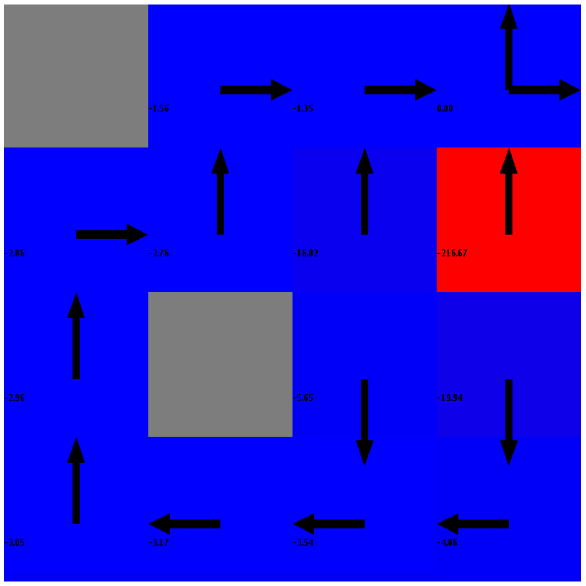
Q-Learning:

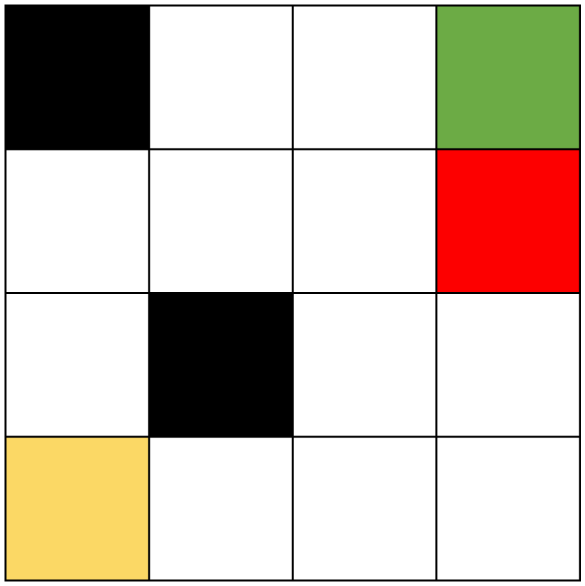
In Q-learning, we do not know the model. It is a reinforcement learning algorithm and it tries to find the optimum policy by finding the Q value for each state. Here, we use epsilon to simulate simulated annealing to avoid falling into global minimum.

**Discussion of Experiments**:

***MDP1***:

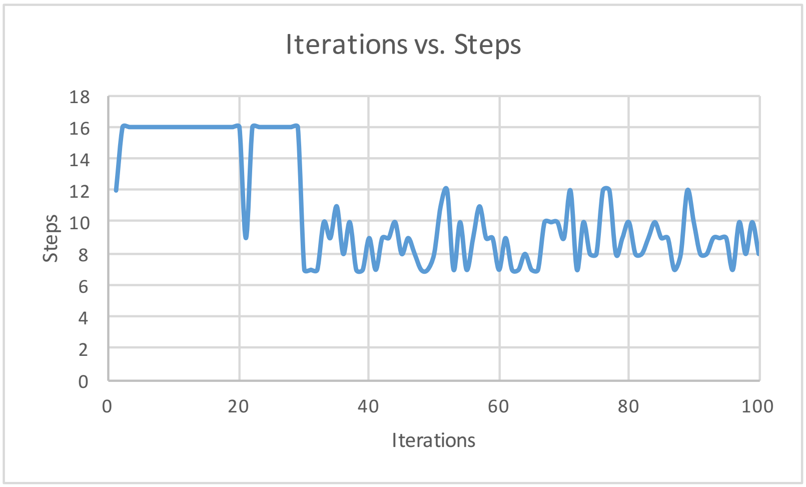
**Value Iteration**:





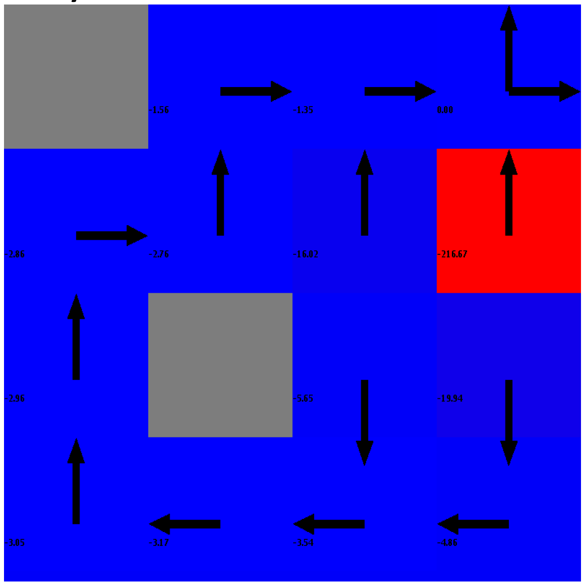
Average Reward: -2.9740000000000015
Average Number of Steps: 10
Minimum Number of Steps: 7
Average Time (in milliseconds): 41

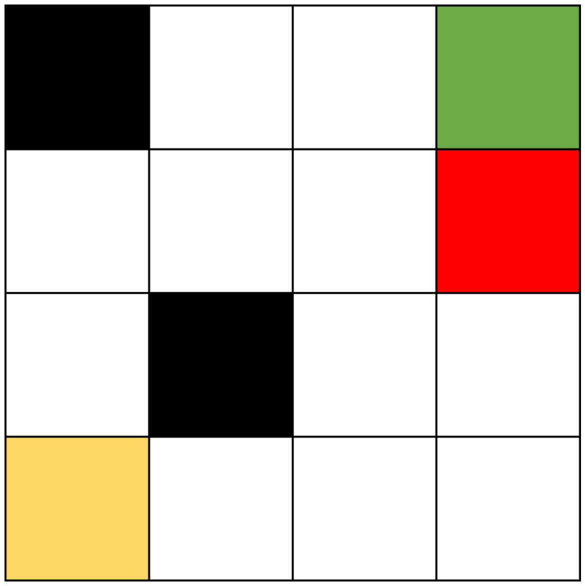
We can see from here that danger grid has much negative utility value than the rest of the grids which makes sense as it deducts large point and the final policy would avoid getting near it.



Here the steps are the steps taken from start to goal. We know from simply looking at the maze that the best route should take 7 steps. We can see here that the method starts to converge around 30 iterations. After that, it flattens.

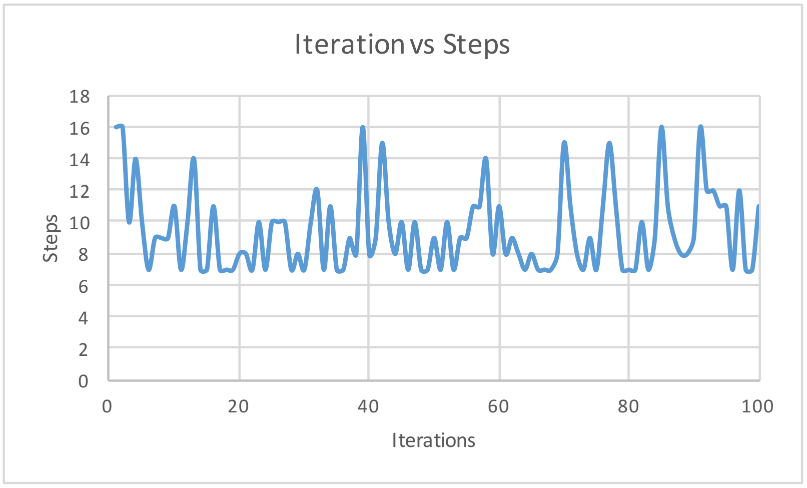
**Policy Iteration**:





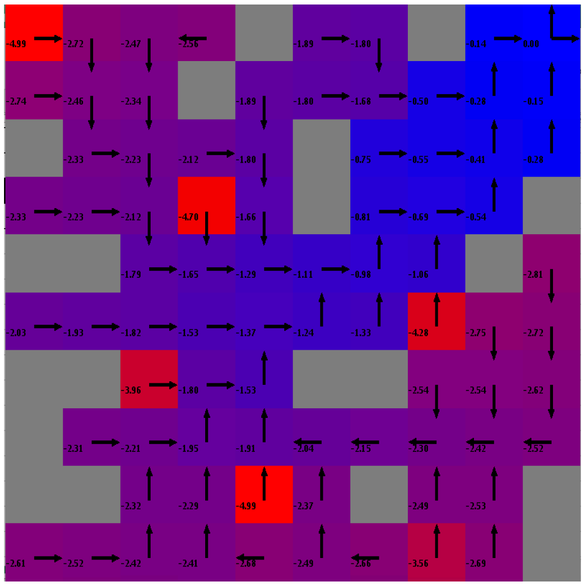
Average Reward: -0.8580000000000002
Average Number of Steps: 9
Minimum Number of Steps: 7
Average Time (in milliseconds): 277

We can see from here that danger grid has much negative utility value than the rest of the grids which makes sense as it deducts large point and the final policy would avoid getting near it.

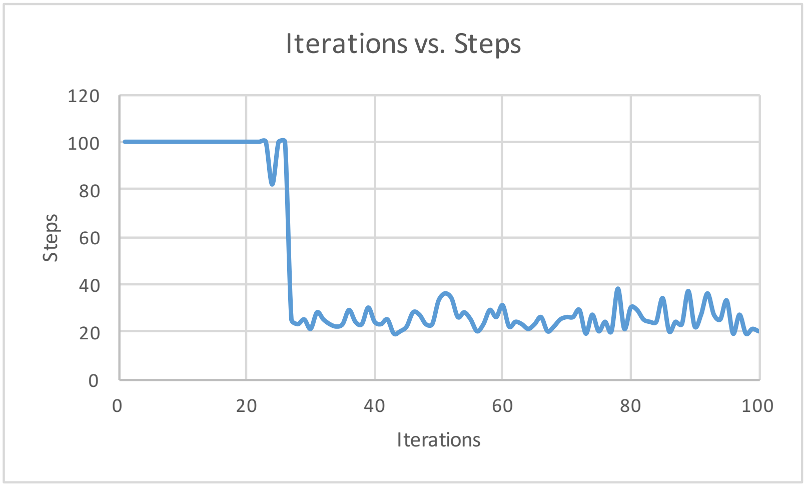


Here we can see that although the steps fluctuate a lot later but the policy iteration method was able to find the optimum rather early before iterations reaching 20. It too finds the minimum number of steps needed to be taken, 7.

**Comparison**:
We can see from the above that both methods find the same optimum route with 7 steps and a reward of -0.6 (not counting the reward of goal in computation). However, we can see that for policy iteration takes much fewer iterations to find the optimum policy than value iterations which makes sense. Because policy iteration only needs to find the best policy, not like value iteration which needs convergence on utility for every state.

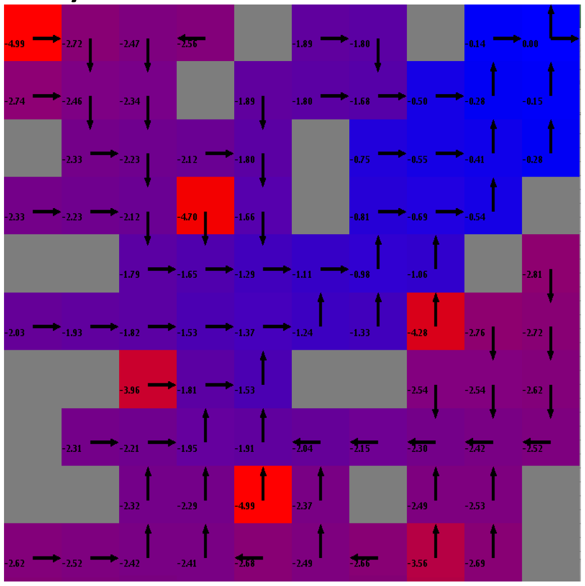
***MDP2***:
**Value Iteration**:


Average Reward: -4.7419999999999994
Average Number of Steps: 44
Minimum Number of Steps: 19
Average Time (in milliseconds): 167

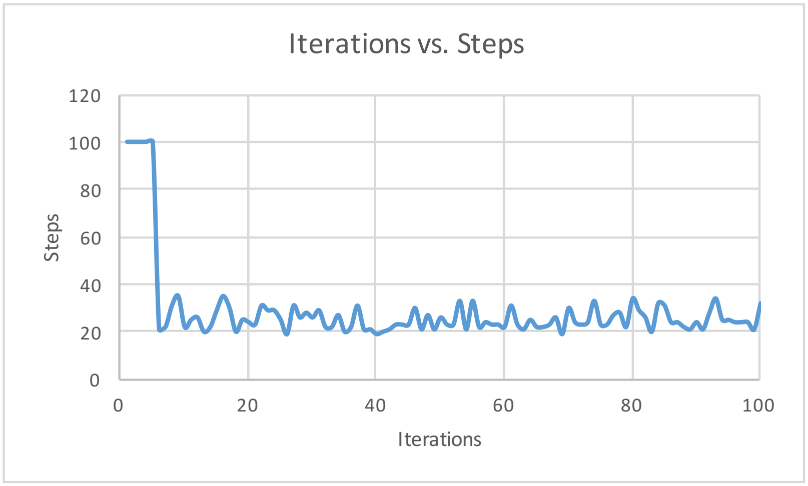


We can see from here that the value iteration takes a little more than 40 iterations to converge to the optimum with 19 steps. The danger (large, medium and small) grids are also correctly marked with very negative utility value compared to other normal grids.

**Policy Iteration:**



Average Reward: -3.2419999999999998
Average Number of Steps: 28
Minimum Number of Steps: 19
Average Time (in milliseconds): 3786



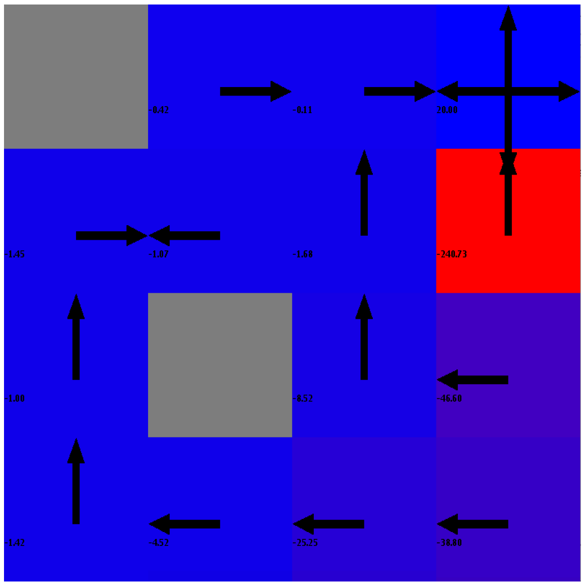
We can see from the graph that it converged around iterations = 26. It also correctly marked the danger grids.

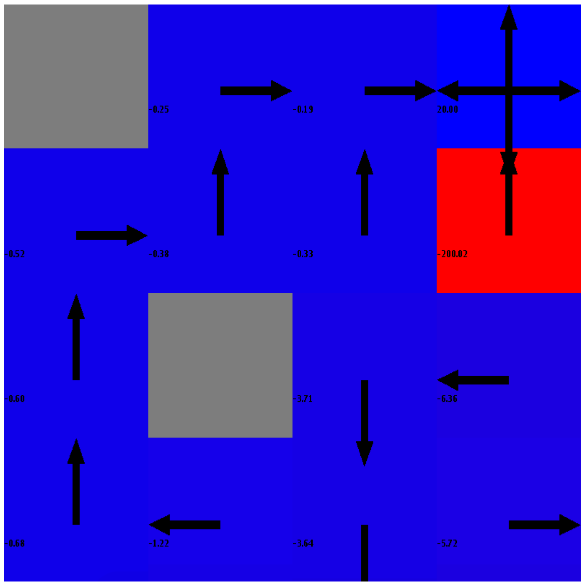
**Comparison**:
We can see that similar to the small MDP, policy iteration is taking much fewer iterations to converge than value iterations for the same reason. Also, they converge to the same reason.

***Overall Comparison between MDP1 and MDP2:***
We can see that for all methods, it takes longer to converge for MDP2 than MDP1 which makes perfect sense as there are many more states to consider in a large maze. It would take much more time to go through one iteration and it takes more iterations to converge.

**Q-Learning:**
**Performance**:

***MDP1***:



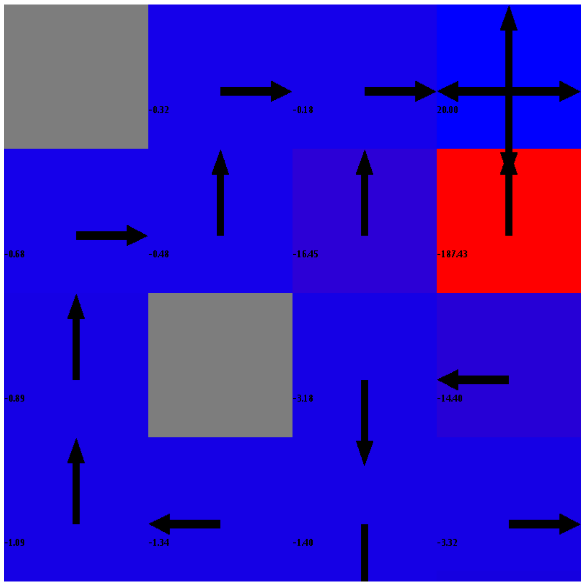


Learning Rate: 0.9
Initial Q-value: 20
Epsilon: 0.1

Average Reward: -3.914199999999999
Average Number of Steps: 12
Minimum Number of Steps: 7
Average Time (in milliseconds): 59

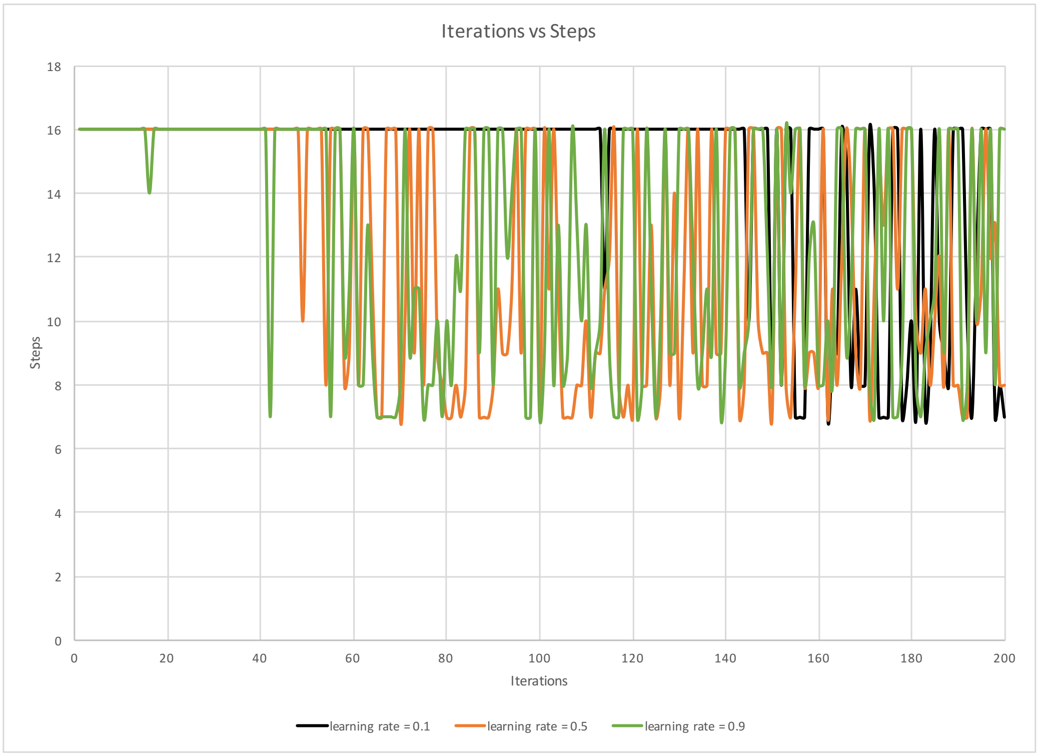
Learning Rate: 0.5
Initial Q-value: 20
Epsilon: 0.1

Average Reward: -9.135499999999999
Average Number of Steps: 12
Minimum Number of Steps: 7
Average Time (in milliseconds): 38



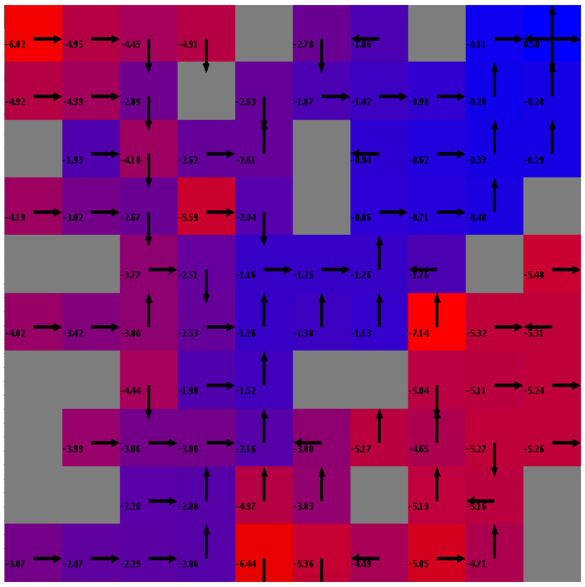
Learning Rate: 0.1
Initial Q-value: 20
Epsilon: 0.1

Average Reward: -2.3745000000000007
Average Number of Steps: 14
Minimum Number of Steps: 7
Average Time (in milliseconds): 79



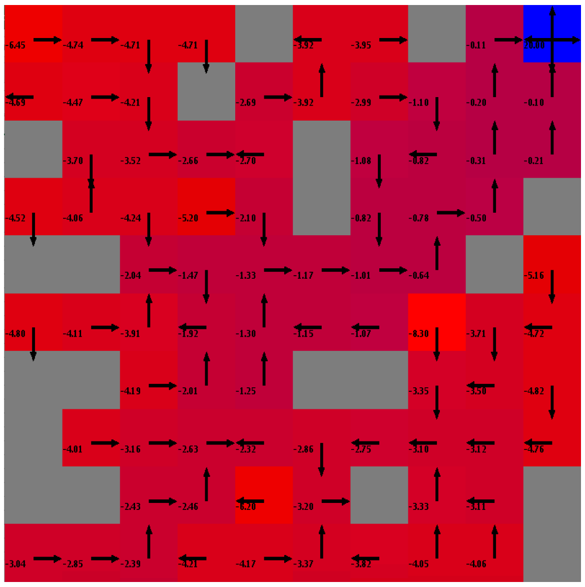
We can see clearly that with the increase in learning rate, it converges to the optimum with fewer iterations which makes sense as it learns from its neighbors more aggressively.

***MDP2:***



Learning Rate: 0.9
Initial Q-value: 0.5
Epsilon: 0.1

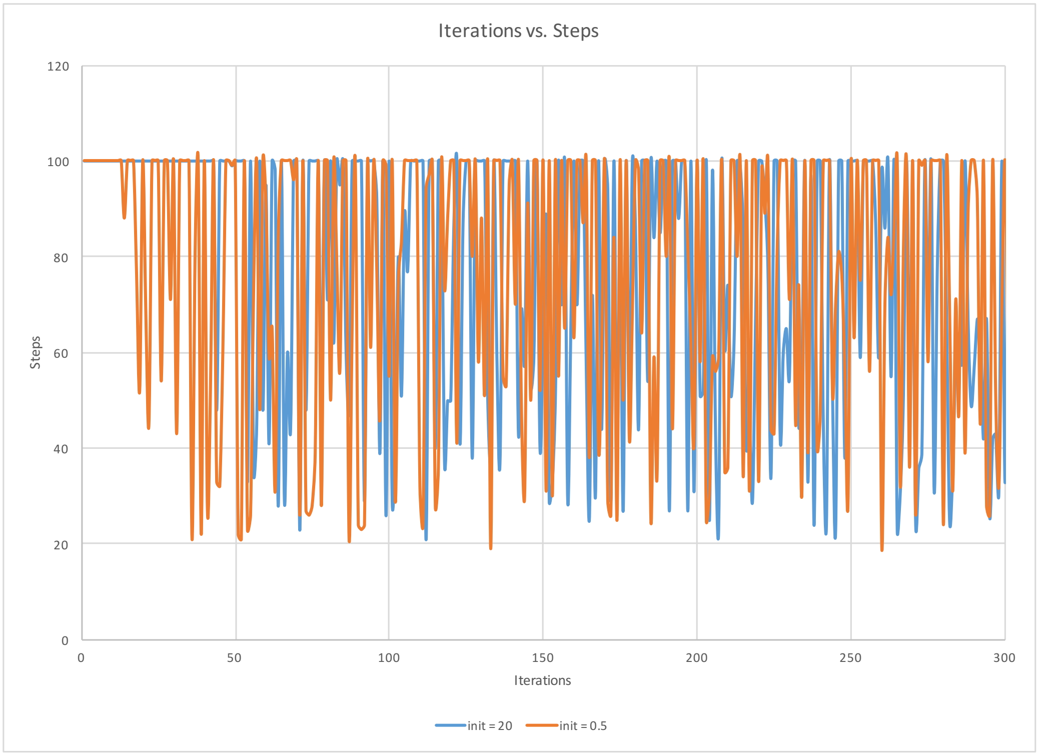
```
Average Reward: -9.686699999999993
Average Number of Steps: 73
Minimum Number of Steps: 19
Average Time (in milliseconds): 174
```



Learning Rate: 0.9
Initial Q-value: 20
Epsilon: 0.1

```
Average Reward:-9.894599999999997
Average Number of Steps: 72
Minimum Number of Steps: 19
Average Time (in ms): 254
```

**Compare with value/policy iteration:**
When comparing with value iteration and policy iteration, we can see that q-learning is generally slower which makes sense as it does not have the model and rewards and consequently needs more iterations. It also does not do as a good job as marking out the danger grids as value/policy iterations which could be because it does not know the rewards as the other two methods do. However, with proper setting of initial value, learning rate and epsilon, we can still learn the optimum policy with q-learning fairly well.



We can see from here that by changing the initial Q value, we are seeing a faster convergence with smaller positive Q value than larger positive Q value. It could be because the smaller positive initial Q value is more close to the optimum Q value, therefore, with the same learning rate, it starts converge faster.

**Exploration strategies:**
I used two exploration strategies, the first is 'simulated annealing' by adding an epsilon and the second is to give Q a large initial value. By adding the epsilon to allow policy to randomly jump with a small percentage (10%), I was able to reach optimum policy every time without stuck in any local minimum. I also tested using large initial value (assigning a positive initial Q value) and small initial value. It turns out a too large initial value would prevent q-learning from falling into local minimum but it would also slow down the process which makes sense as it takes longer to converge to the optimum value from a too large value with the same learning rate.

**Code Reference**:
Code copied from https://github.com/svpino/cs7641-assignment4 with modification on MDP problems and different parameter settings.