Markov Decision Processes

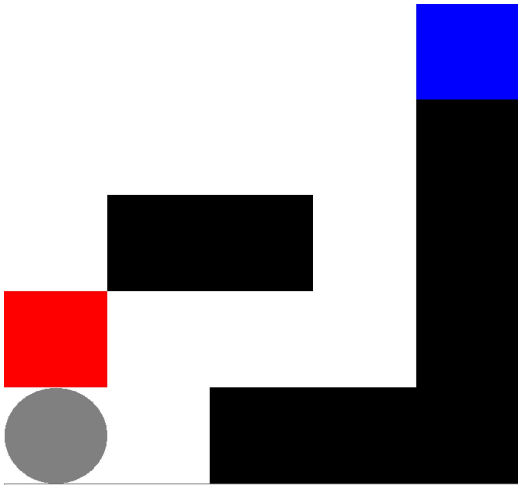
Srinidhi Palwayi (spalwayi3)

Introduction:

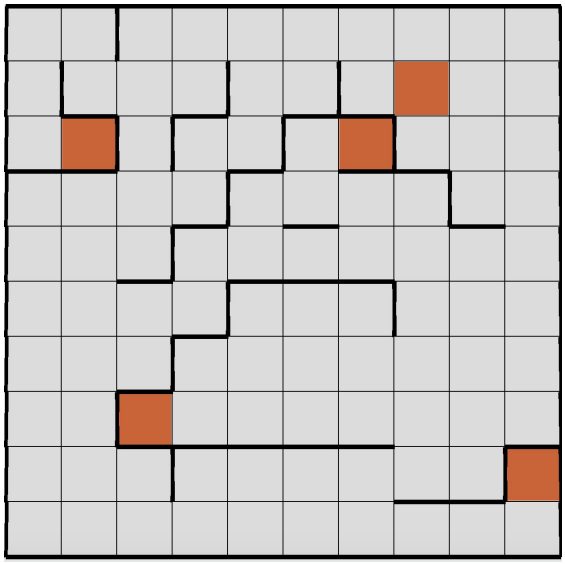
Markov Decision Processes (MDP):

The cornerstone of the markov decision process is that the future only matters on the present state. This is the core principle of a Markov decision process, which models a situation given the states, actions, rewards, and transition model. States are all the possible locations that an agent can be in, actions are the actions that can be executed in a state, and reward is the reward the agent gets for being in state. Lastly, the transition model is the probability of going to a future state given the current state and an action.

Small Number of States MDP:



The figure on the right is the MDP with the smaller number of states. In this figure the red block is the negative reward of 20 points, and the blue block is the positive reward of 100 points and is a terminal state. The black blocks are walls, and the gray circle is the agent. In this MDP I modeled it based on toll roads, the red block is the toll. When driving if you have the option of going through a toll road vs not going through a toll road, the toll road probably will give you more lanes and give you more options to get to the destination faster. This is mimicked here when you can go up, there's more options to get to the destination. Furthermore, the agent wants to get to the destination as quickly as possible because every move is -15 reward. The agent moves with 0.8 certainty in the desired direction, and $0.2 \div 3$ among the other 3 directions. Lastly, this MDP was created using BURLAP.



Large Number of States MDP:

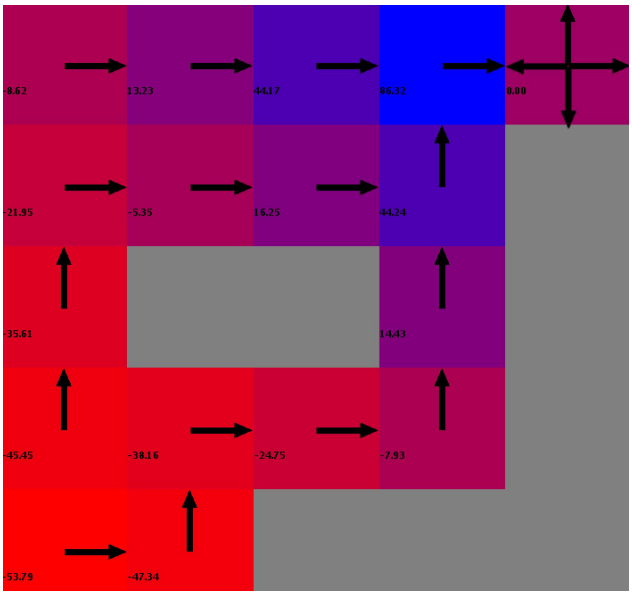
The figure on the right is the larger MDP, which is 10x10 maze. I wanted to explore what happens when there's multiple rewards. Furthermore, these rewards are in room-like situations because the rewards are surrounded by walls. The penalty for the agent hitting the wall is -50. Furthermore, the precision is to check for convergence is set at 0.001. Lastly, this MDP was created using RL Sim.

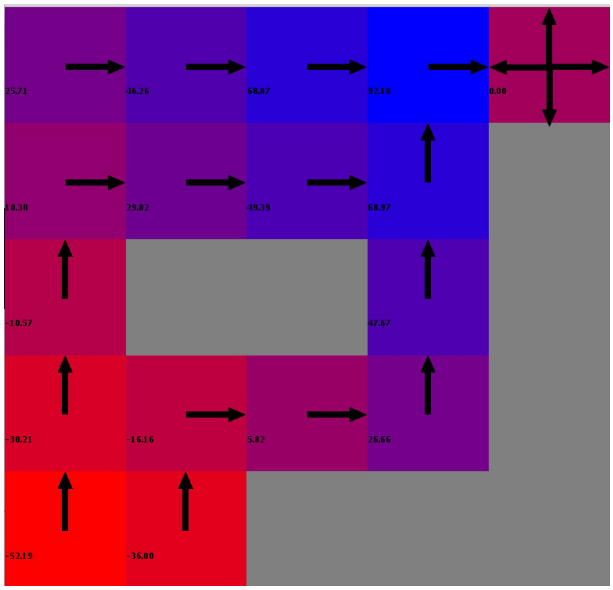
Value Iteration:

The goal of value iteration is to give each state a utility, which is the current reward plus the discounted future reward. The discount rate is represented by gamma. To calculate the utility we use the Bellman Equation: $U(s) = R(s) + \gamma max_a \sum_{s'} T(s,a,s')U(s')$. Since we don't know the utilities of any of the states to begin with we first initialize the utilities of the states to arbitrary values. Next we update each state based on the neighbors it can reach using the Bellman equation. The previous step is repeated until the change between each iteration is miniscule. This algorithm works because even though we start with arbitrary utilities, at each step we also propagate the immediate reward which is R(s).

Small MDP:

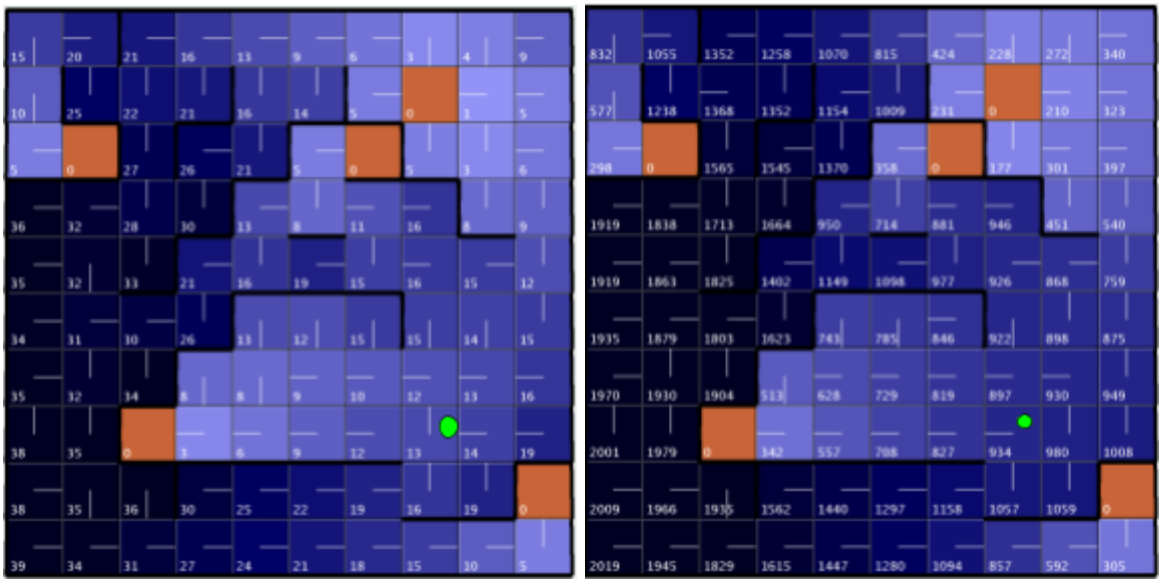
In this experiment, I wanted to see at what values of gamma, would the optimal policy at the starting position be to go into red negative reward block. The figure on the left is when gamma is 0.80, and in this scenario the negative reward block is avoided. In the figure on the right is when gamma is 0.99 and in this scenario the agent goes through the negative reward. The higher the gamma value the more the agent will look at future rewards. Hence, if we look at future rewards more than the agent views going through the red block as a better policy. This makes sense because if we go towards the right instead of up, we avoid current loss in reward, but we are confined to a path that only has one truly beneficial direction at a time to get to a reward. If we have a misturn on this path then we are increasing negative reward because we are not at the terminal goal state.

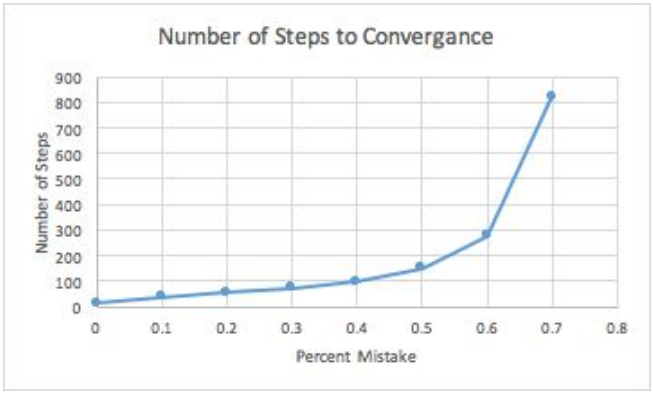


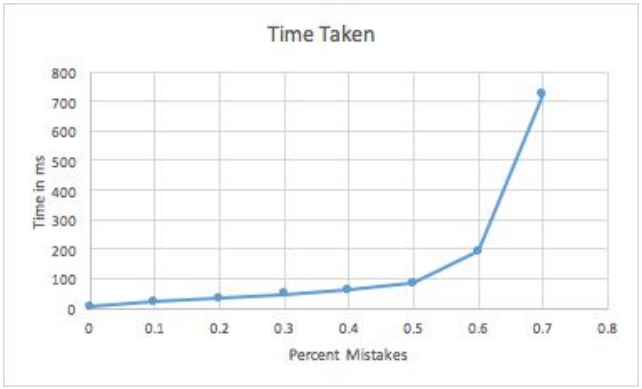


Large MDP:

The figure on the left shows the values and policies when the percent of mistakes made is 10%, and the figure on the right shows the values and policies when the percent of mistakes is 70%. We can see that the color of the boxes are relatively similar, however the cost to be at a state is significantly higher in the 70% mistakes rate. This makes sense because at 0.7 mistake rate, the agent is moving in the intended direction only 30% of the time, which is only slightly better than random. Hence, it's more prone to hitting obstacles. Furthermore, the policy that 0.7 pursues, doesn't actively avoid walls like in the 0.1 scenario. In the 0.1 figure on the state with a green circle, we can see that the optimal policy is to move up, and then head towards the goal state and avoid the chance of hitting a wall even though this is a longer path. In the 0.7 figure, the optimal policy is to directly go towards the goal. This makes sense because it's much more difficult for the agent in the 0.7 figure to end up at the goal, hence it should take the path that's more direct, because even then it's very likely to go off the intended path before reaching the goal state.







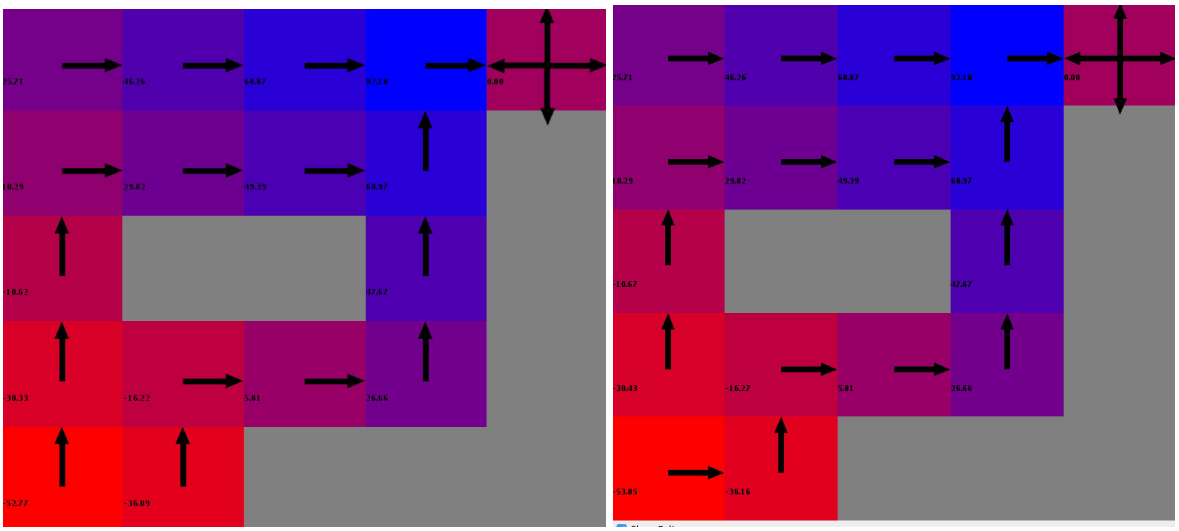
As the percent of time that a step was made in the unintentional direction increased, both the number of steps to convergence and the time taken to convergence increased exponentially. This makes sense because if the percent of times mistakes are made is high, the agent is making numerous suboptimal moves before convergence.

Policy Iteration:

Policy iteration begins by using a guess for the optimal policy. Then the policy is evaluated given the guess. Lastly, the policy is improved using a Bellman equation: $U_t(s) = R(s) + \gamma \sum_{s'} T(s, \pi_t(s), s') U_t(s')$. The last two steps are repeated until the change between each iteration is miniscule.

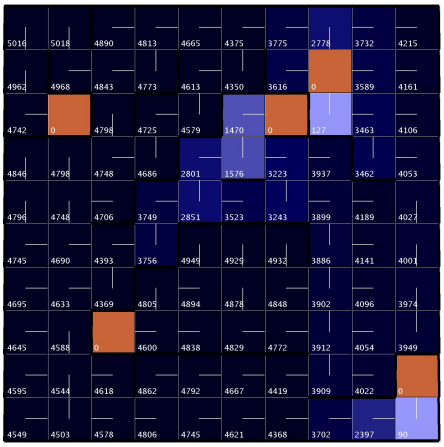
Small MDP:

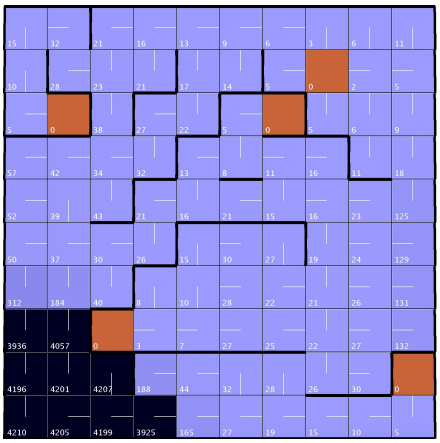
When changing the gamma values, I got the same results for each state as value iteration. Instead I looked at what value of the negative reward, would the starting position choose to go left, instead of up when the the gamma is 0.99. Interestingly, the crossover happens between -20.5 and -21. At -20.5, the agent chooses to go through the toll, but at -21 the agent chooses to avoid the toll. In the images below the reward of -20.5 is shown on the left, and -21 is shown on the right. This makes sense because the agent moves with 0.8 certainty and movement itself is expensive at -15, so if the toll is too costly it is better to circumvent it.

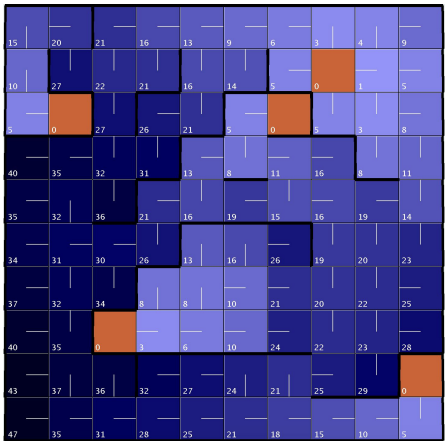


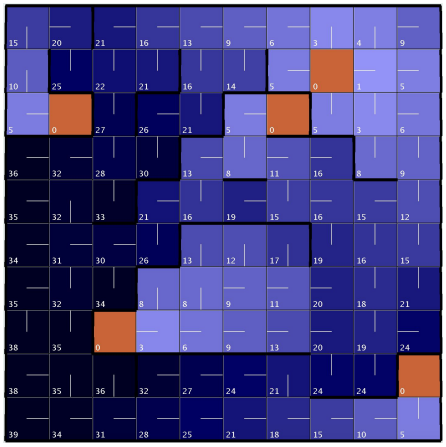
Large MDP:

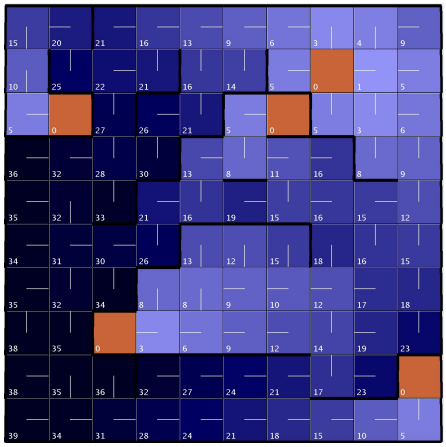
When doing policy iteration, I got the same utility and policy for the states as doing value iteration for each percent error value. However, the number of iterations is significantly fewer in policy iteration. As you can see the in the images below it only takes 7 steps to converge when, when percent error is 0.1. Upon further inspection, it's interesting to note the drastic jump from the first step to the second step, where the majority of values dropped from 4 digit number to 2 digit numbers. The average change in between each step decreases exponentially in subsequent steps.

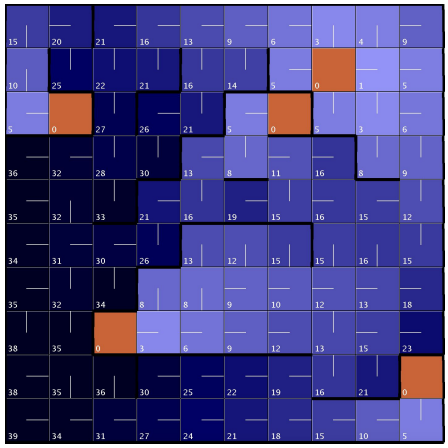


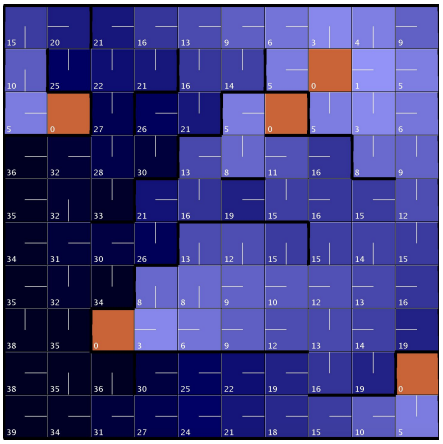


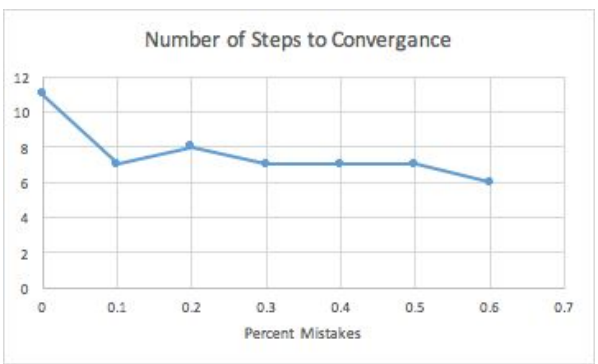











Number of Steps to Convergance
12
10
8
6
4
2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Percent Mistakes


Time Taken
140
120
100
80
60
40
20
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Percent Mistakes

It's interesting to note that the number of steps to converge doesn't change much as the percent of mistaken steps increases. This is very different from value iteration, but in policy iteration each step in policy has a much larger impact. Hence, it takes much fewer iterations to converge than in value iteration. For the time taken, up until 0.2 value iteration converges faster, but after that point policy iteration converges faster. This makes sense because policy iteration take significantly less steps, and even though each step is more costly, unlike value iteration the number of steps isn't growing exponentially.

Q Learning:

Compared to value iteration and policy iteration, reinforcement learning is fundamentally different because it doesn't assume the transition function and the rewards are known. Reinforcement learning is learning based of exploration versus exploitation. On one hand, exploration is the notion that if the agent occasionally takes what seems like a sub-optimal action, the agent will learn more about the environment and might find a path of more reward. On the other hand, exploitation is the notion that agent should use what it believes is true about the environment, and take what it believes is the optimal action. When using the reinforcement learning, the agent attempts to learns the model of its world using exploration and exploitation.

I'm going to use Q-learning to solve my MDPs. The goal of Q-learning is to find the value for being in state, and it does this by assigning each state a Q-value. Q-value is defined as:

$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') max_{a'} Q(s',a')$. This, like the equations used to solve value iteration and policy iteration, is a Bellman equation.

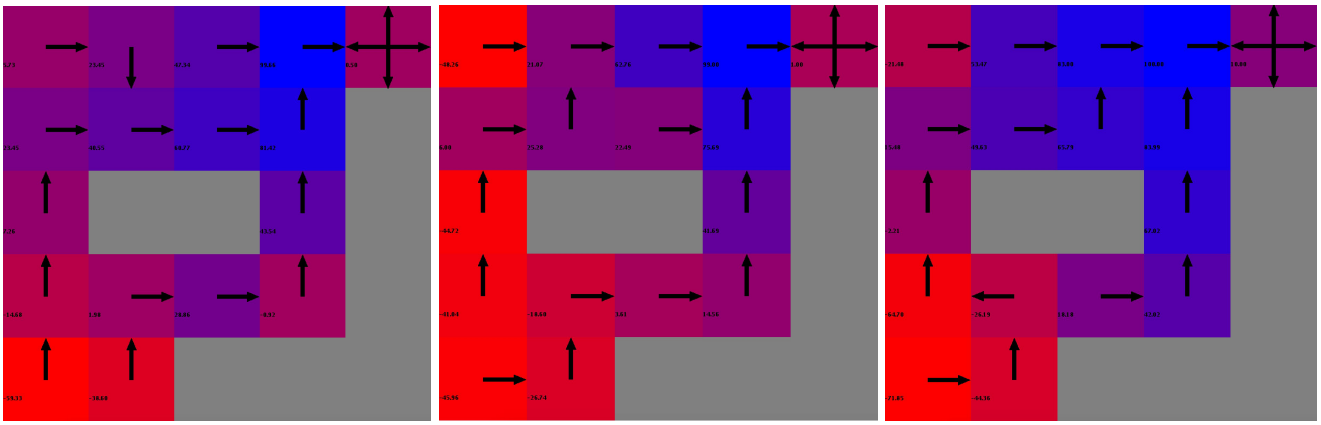
Other values that are crucial to Q-learning are q-initial, learning rate, and number of episodes for planning. Q-initial is the initial q-values for the state action pair. If the q-value starts of high this will motivate exploration. Learning rate determines, what weight to give new values vs old values. If the learning rate is 1 that means the q-value will only consider new information, while if the learning rate is 0 that means that the q-value will only consider old information. If the learning rate is set to decay that means that after every episode the learning rate will become smaller. For the smaller MDP, I didn't decay the learning rate, but for the large MDP I did. The number of episodes, is how many times the agent will go to the terminal state. Lastly, for the large MDP we also have the epsilon term, which aids in exploration. Epsilon is the percent of time that the agent takes an action that doesn't seem optimal in order to explore more states.

Small MDP:

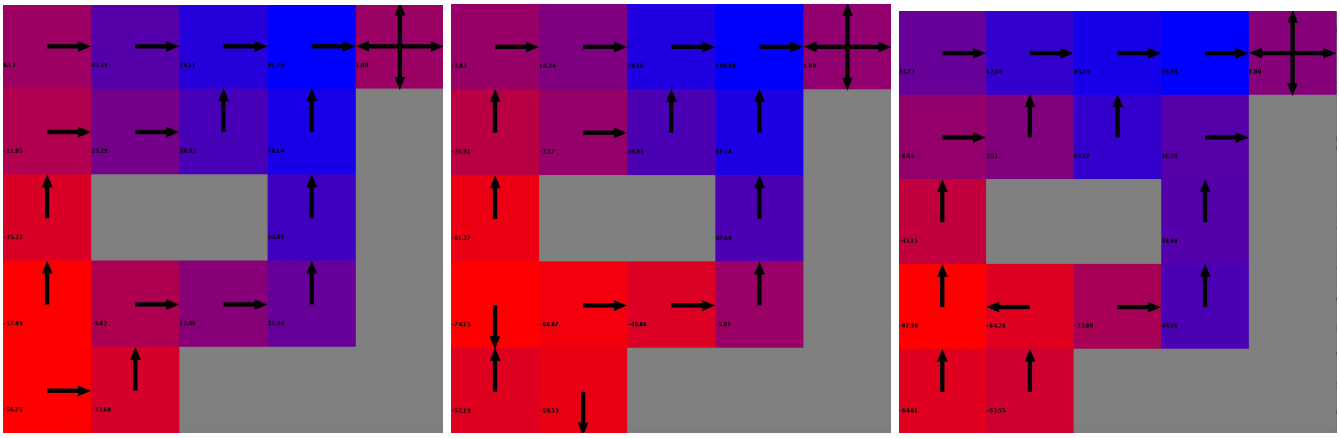
For the experiments below, unless otherwise mentioned, gamma is 0.99, q-initial is 1, learning rate is 0.7, and number of episodes is 10.

For my first experiment, I modified the q-initial. From left to right, the q-initial values are 0.5, 1, 10. The values that most closely align with policy iteration and value iteration is when the q-initial value is 0.5. However, both 0.5 and 10, occasionally have policies that take them further away from the goal.

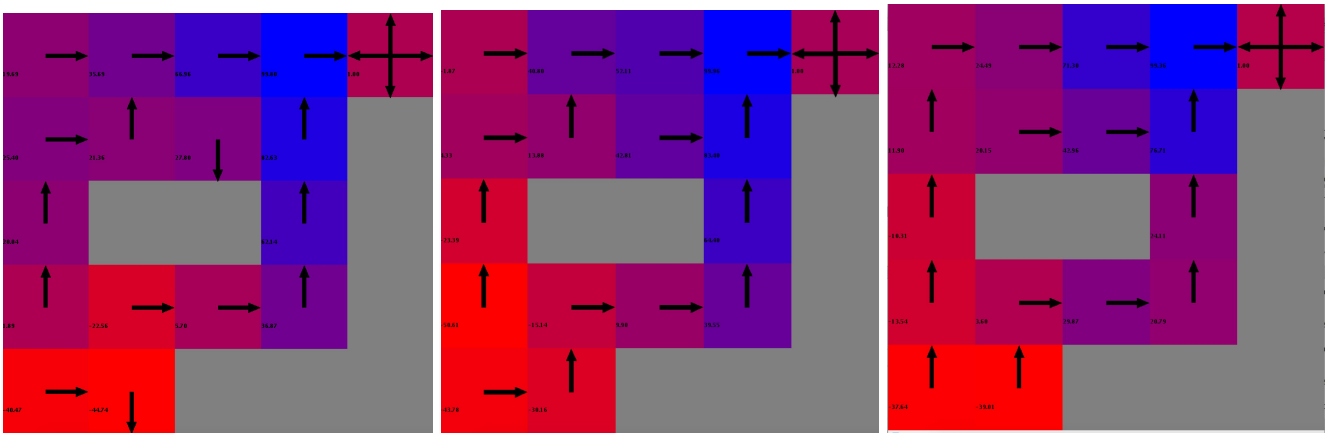
Hence, at 0.5 not enough exploration is done to find the best path to the goal, while at 10 too much exploration is being done, that the agent is not utilizing previous knowledge as much.



For my next experiment, I modified the learning rate. From left to right the images represent a learning rate of 0.2, 0.6, and 0.9. Again, the higher learning rate the more likely the agent is to explore. What seems like the optimal policies, where no state is moving further away from the goal, occurs at 0.2. However, at higher learning rates suboptimal policies occur, where the agent moves further away from the goal, goes into a wall, or even purposely goes through the negative reward after the starting location.



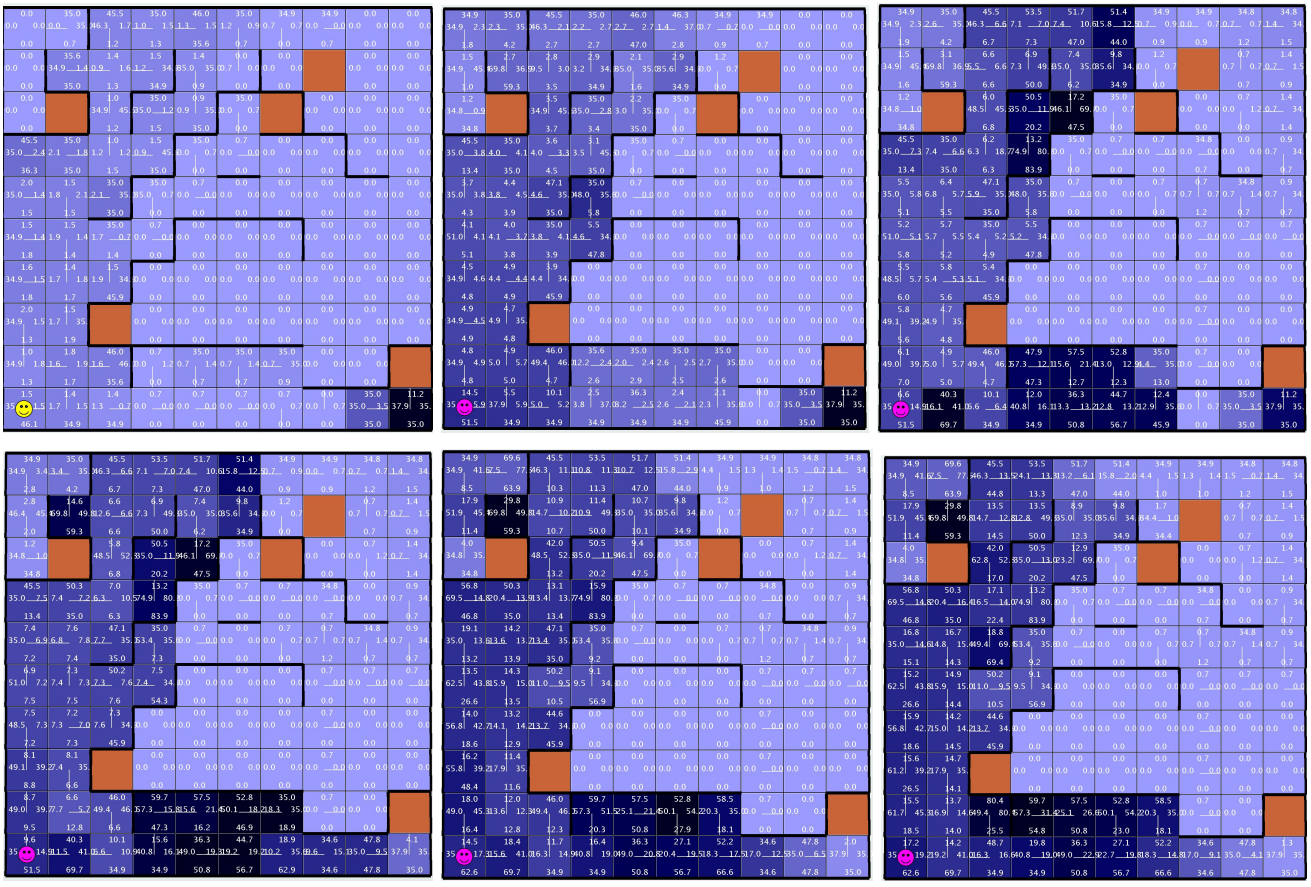
Lastly, I modified the number of episodes. From left to right the images represent number of episodes of 1, 5, 10. When we only had one episode, the agent has policies that are very obviously suboptimal like going into walls. Additionally, each episode makes the agent more confident about its environment, and the ideal number of episodes would be infinite. This is because then each state, action pair can be explored an infinite amount of times.



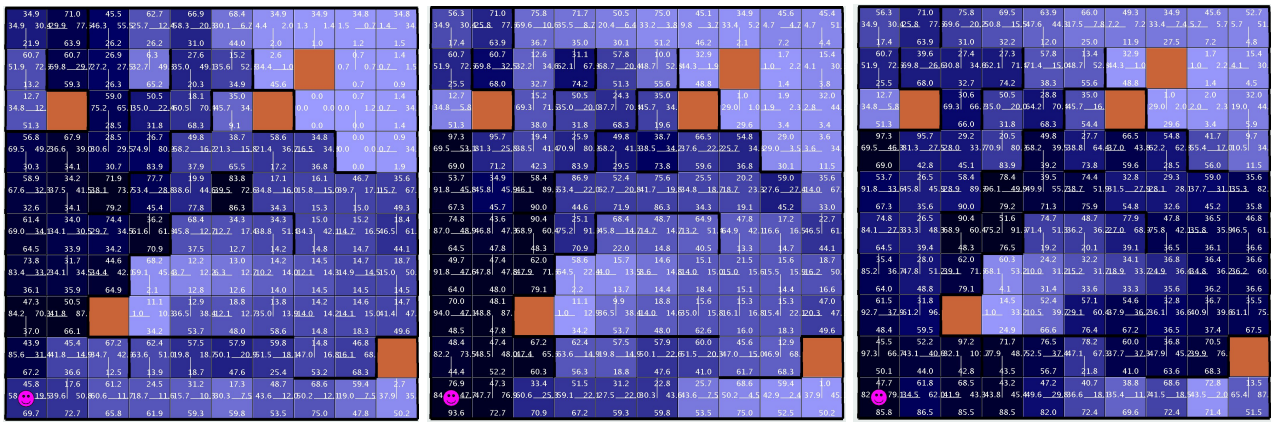
Large MDP:

For the experiments below the percent mistake and epsilon is 0.1, learning rate is 0.7, precision is 0.1, and number of episodes is 50, unless otherwise noted.

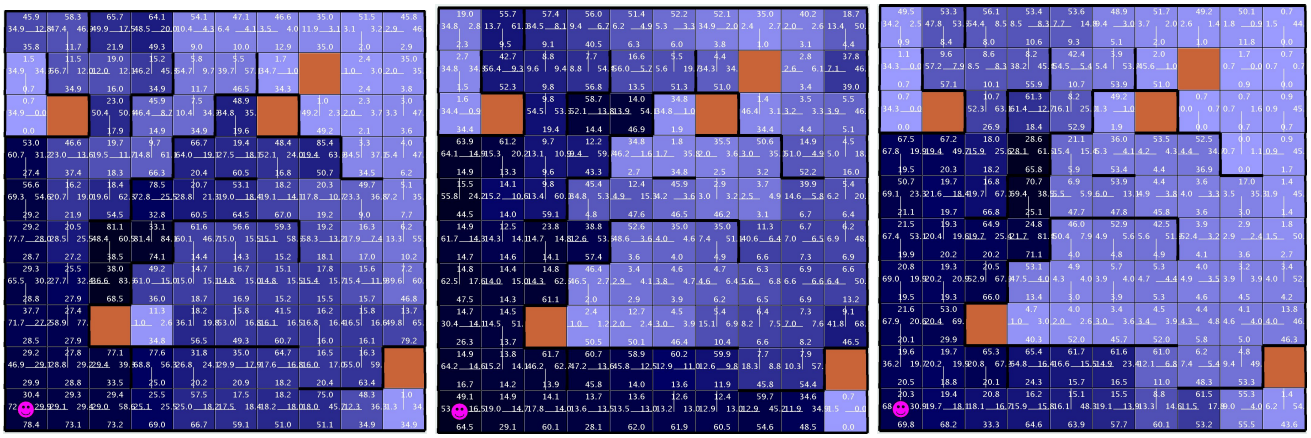
For my first experiment, I wanted to see how many much the q-values changed every 3 episodes. It's interesting to note that only 2 goals have been explored in the first 3 episodes, which is likely due to the low epsilon. Additionally, the goal state that is closest to the bottom left doesn't get explored even after 18 episodes, and the area only gets explored after 24 episodes.



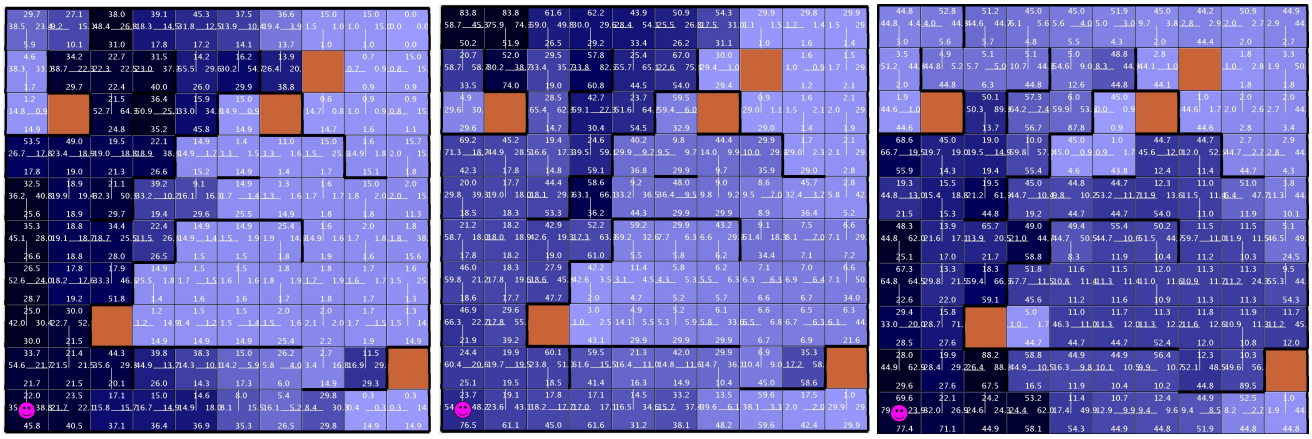
In the images below from left to right, I let 124, 224, and 624 episodes pass. At 124 episodes, states beneath the upper right hand most goal the state have yet to be explored. At 224 and 624 optimal policies were not seen around some of the goal states because the best policy was not to go into the goal, as it should have been. This can be seen in the goal state in the upper right hand, where the state directly above the goal chooses to take a longer path to the goal. This probably occurred due to the low epsilon rate and the decaying learning rate, which didn't encourage much exploring after a certain number of episodes.



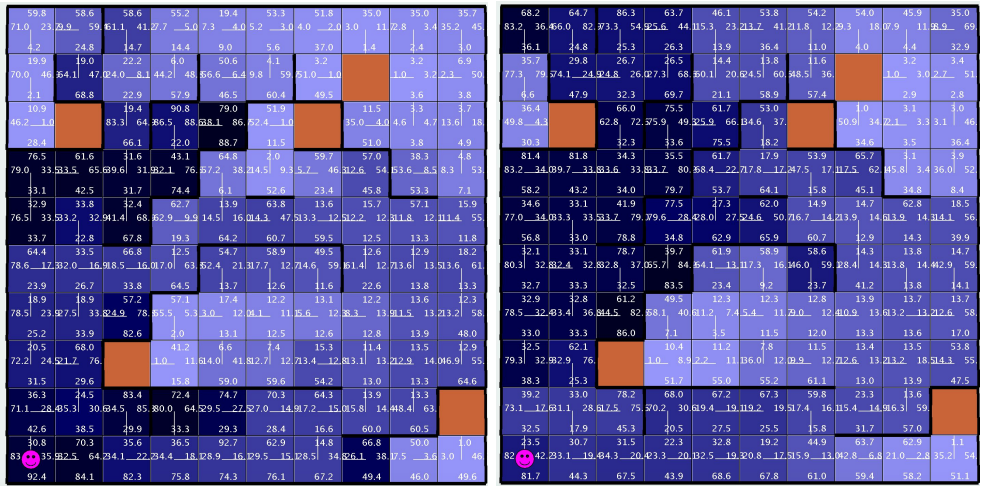
In my next experiment, I changed the epsilon rate and for the images below from left to right the rates are 0.2, 0.4, 0.6. It's interesting to note that 0.2 has darker purples around the middle goal state, even though it should be lighter since we are close to the goal. Additionally, with an epsilon value of 0.2 the best believed policy doesn't reach the middle goal. With an epsilon of 0.4, the policy around more goal states became more optimal. However, at a policy of 0.6 too much exploration was done and the policies around goal states was less optimal than 0.4.



In addition, I modified learning rates. From left to right the learning rates are 0.3, 0.6, and 0.9. A learning rate of 0.3 relied too heavily on previous information, and many states on the right hand side of the board are explored very little. However, at learning rate of 0.9 the middle goal state, doesn't have a perceived best policy leading to the goal state.



Lastly, I compared decaying learning rates to constant learning rates. For the images below I used 200 episodes, learning rate of 0.7, and an epsilon of 0.3, and the left-hand image has a constant learning rate, while the right-hand image has a decaying learning rate. Although the results are comparable, there are fewer suboptimal policies with the decaying learning rate. This makes sense because when we have a decaying learning rate, we start to rely more heavily on previously known information in later episodes.



References:

“Intro to Machine Learning” Udacity, n.d. Web.

https://github.com/juanjose49/omscs-cs7641-machine-learning-assignment-4 use this wrapper for BURLAP