

## Summary:

This project is focused on Markov Decision Processes (MDP) which can be solved with reinforcement learning. All MDP's have four components: A finite set of possible states, a finite set of possible actions, the probability that an action in a certain state at certain time will lead to the next state, and the immediate reward from transitioning states. I will explore three different algorithms to solve an MDP problem known as GridWorld implemented by BURLAP java library.

The three different Algorithms explored are Policy Iteration, Value Iteration and Q-learning. Policy and Value Iteration are well known Dynamic programming solutions to solve MDP's. They are known as Offline planners where the agent has prior knowledge about the effects of its actions on the environment. They are both able learn the optimal policy even within a stochastic environment such as GridWorld.

Value Iteration is a greedy algorithm that tries to find the optimal value function then to extract the optimal policy function. Policy Iteration is similar except that it evaluates the policy and improves the policy until convergence, on average it converges faster than Value iteration if done correctly. Q-learning on the other hand is different in that it is a reinforcement learning technique which doesn't require a model of the environment and learns through experience from trial and error to optimize itself.

Within this project, I will explore the effects of the Initial Reward and Learning rate on the Q-Learning process. Initial rewards are instantiated for each state that has not been explored., this incentivizes exploration as the Q-learning algorithm will seek the states that are unknown in order to properly update them to their optimal value. The Learning Rate influences how likely the Q-learning algorithm will exploit and take advantage of the rewards it knows how to access or how it will explore, and take a risk in the hopes of a future reward at the cost of the current reward.

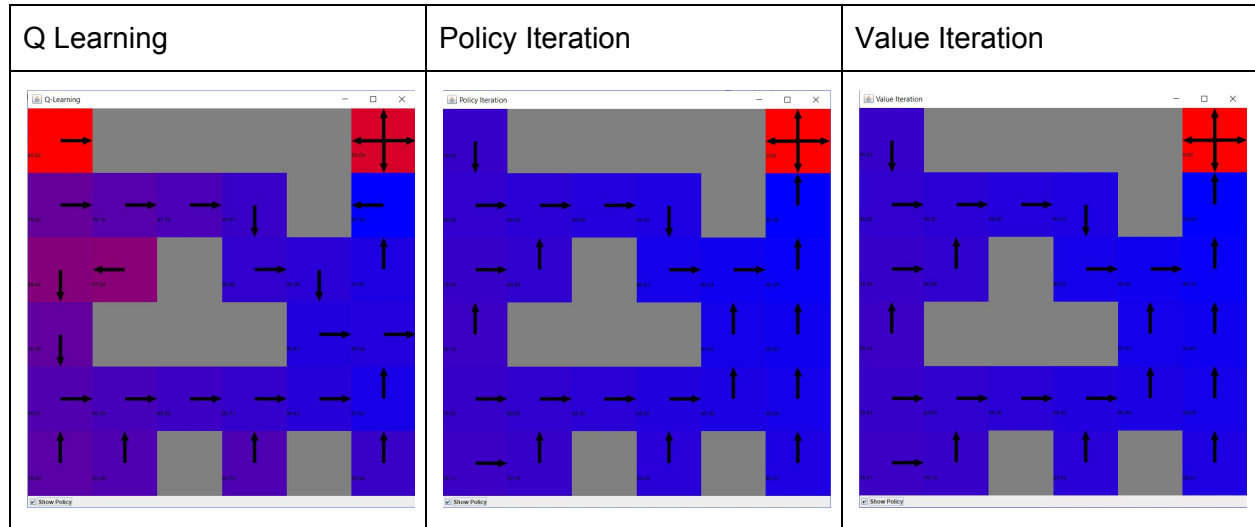
$$Q^{new}(s_t, a_t) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}_{\text{learned value}}$$

## Simple GridWorld Description:

My simple problem was a GridWorld with 25 possible states. GridWorld is set up to be stochastic where each action has an 80% chance of success and a 20% chance of failure to either the left or right of the action. This is interesting because it is stochastic and there's a possibility that a certain action leads to an unintended consequence such as taking a longer route. This is important since Policy Iteration and Value iteration need an environment with a finite number of states to function. This problem is useful because it is a simple model and the

solutions converge relatively fast compared to a larger MDP. Gridworld relates to the real world because is similar to navigating through campus to find the optimal paths between place you want to go. Gridworld is useful to see the differences between Policy and Value iteration from Q learning.

Results:

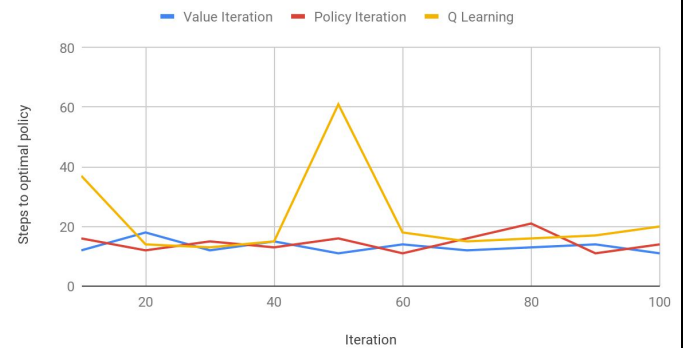


## Summary of result of simple GridWorld :

We can clearly see that Policy Iteration and Value iteration diverged to the same policy whereas Q-Learning was different. The start position is the bottom left of the screen and the goal is the top right and we can see that even the starting move is different. For Value and Policy Iteration the first move was to go right since it prevent the agent from going to the state above where it could end up in suboptimal path. For Q-learning The first move was determined by randomly iterating until it 'hit' the reward. The algorithm only learns based off experience so its first reward skews the algorithm. For the algorithm it isn't as aware of other possible rewards so it colors certain regions "red" to indicate a suboptimal state. Since it relies of trial and error and learns by exploring the state space and all the various combinations of actions so it learned that going in the red territory was really bad. A fundamental trade-off within Q-learning is the trade-off between exploration or exploitation. The agent needs to either take a risk to find new rewards (explore) or capitalize on the rewards it already knows (exploit). If the agent explores too often, It will never gain a lot of reward while if the agent never explores it could never learn a more optimal path. The solution is to start off with a high learning rate encouraging exploration initially and reduce it over time to encourage exploitation.

Here are a few graphs i generated that describe the training process of each of the algorithms over the iterations. All of the algorithms used the same reward function.

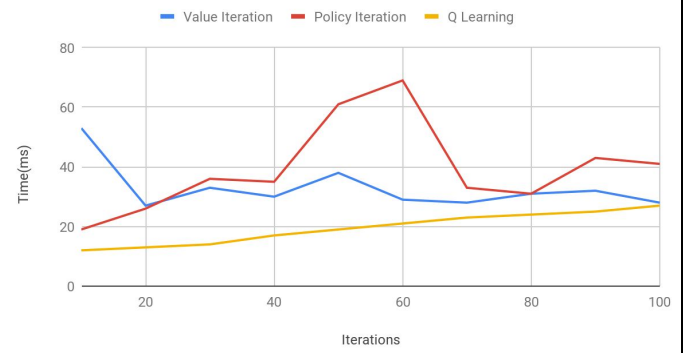
Number of steps agent needed to reach terminal



The optimal policy total reward gain



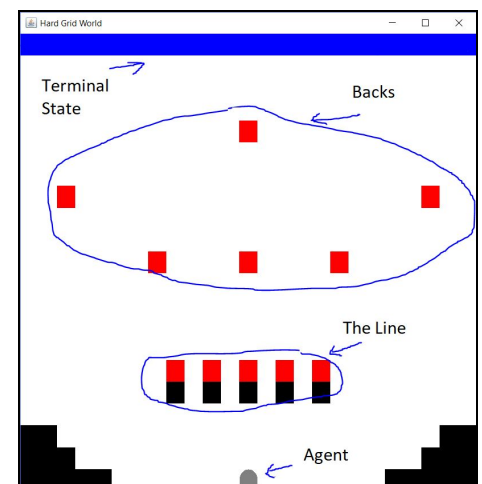
Time in Milliseconds the optimal policy needed



If you take a look at the number of steps the agent took to reach the terminal state, we can see all the algorithms did fairly well and were all very similar. The same is true for the optimal policy total reward gain graph. When we plot the total time to calculate the optimal policy we can see that Policy Iteration took the longest while value Iteration and Q-Learning took a similar amount of time to compute. Q-learning converged as it should however in the Hard Gridworld, we will need to increase the amount of iterations drastically in order for the values converge.

## Hard GridWorld Description:

While I was watching the Alabama Vs. Georgia football game, I noticed that we as humans have figured out optimal running paths to get the ball from behind the line to get a touchdown. I have created a big version of GridWorld where the agent has to find its way through multiple defenders who are trying to take down the agent. GridWorld is set up to be stochastic where each action has an 80% chance of



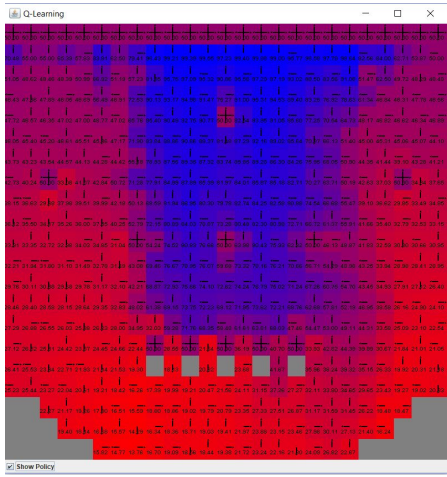
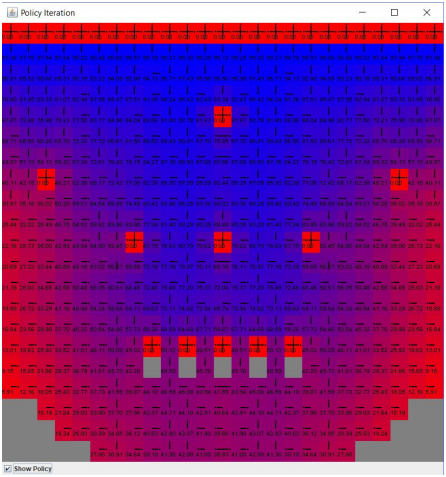
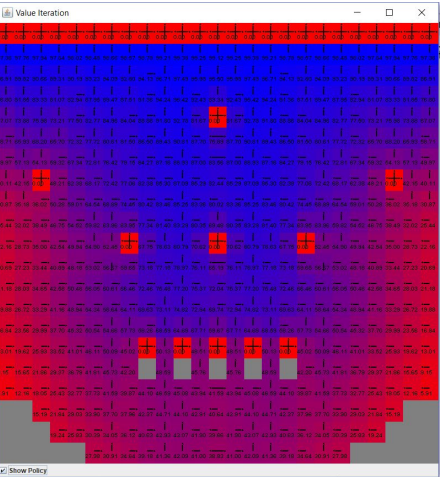
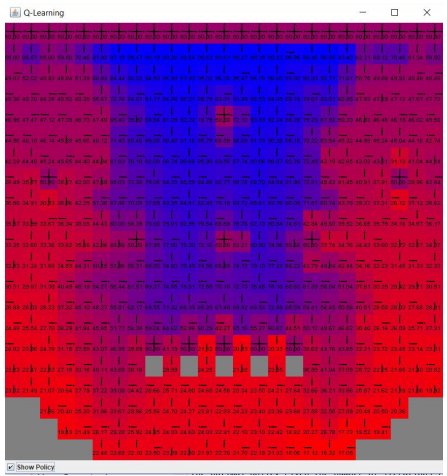
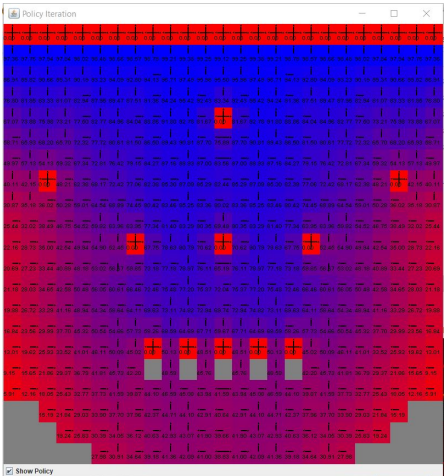
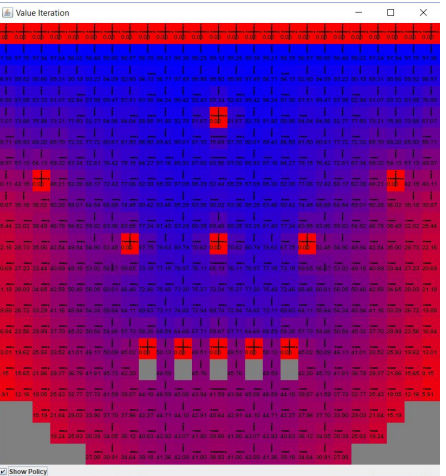
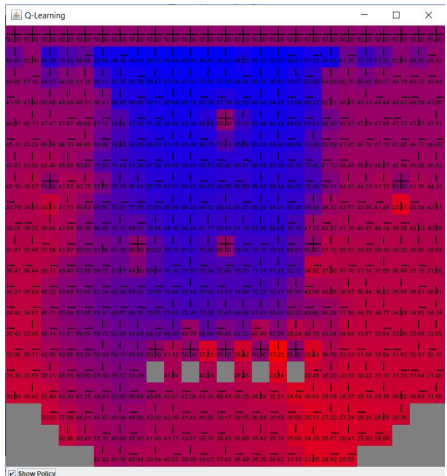
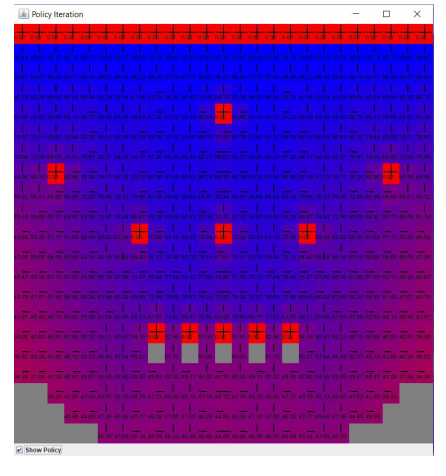
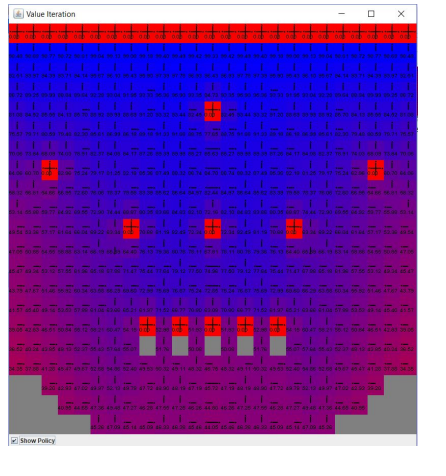
success and a 20% chance of failure to either the left or right of the action. When the Agent was taken down by a defender, I assigned it negative reward (-40) while if the Agent successfully reached the top of the screen, I gave it a good reward (+100). Each defender (shown in red) is a terminal state and the whole top of the world is also a terminal state. I tested these algorithms on multiple weights. Once the agent is passed the line, it also wants to avoid the Backs which are trying to eliminate the agent. Each red dot while represents a defender is a terminal state that deals negative reward to the agent, so in theory the agent should try to avoid them while also trying to get to the top of the graph as quickly as possible since each step the agent takes will have a penalty. The following is my HardRewardFunction.java found in the util file. The reward once the agent reaches the top of the screen or “field” is 100. Each action it takes that is not a terminal state, the agent receives a penalty of 0.1 to force it to get to the reward as quickly as possible. The “yardMultiplier” parameter weighted how much to penalize movements in the x direction away from the midline. This should in theory cause the agent to prefer policies that either go through the “line” or the to go around the “line.” The “tackleMultiplier” increases the penalty that the agent takes from a tackle.

```
@Override
public double reward(State s, GroundedAction a, State sprime) {
    // get location of agent in next state
    ObjectInstance agent = sprime.getFirstObjectOfClass(BasicGridWorld.CLASSAGENT);
    int ax = agent.getIntValForAttribute(BasicGridWorld.ATTX);
    int ay = agent.getIntValForAttribute(BasicGridWorld.ATTY);
    // are they at goal location?
    if (ay == this.goalY) {
        return 100;
    }
    //yardMultiplier penalizes deviations from the middle in the x direction
    //tackleMultiplier increases the penalty from the terminal state
    return -1*(Math.abs(ax - this.goalX/2)^2)*this.yardMultiplier - .1 + map[ax][ay]*this.tackleMultiplier;
}
```

## Hard GridWorld Results:

Below I showed the solutions that each algorithm produced given a reward function with two parameters. The first parameter multiplies the penalty of movements in the x direction away from the midline to incentivize being closer to defenders. and the second parameter multiplies the penalty of being terminated by defender in the hopes of being less ‘risky’ and being near defenders.



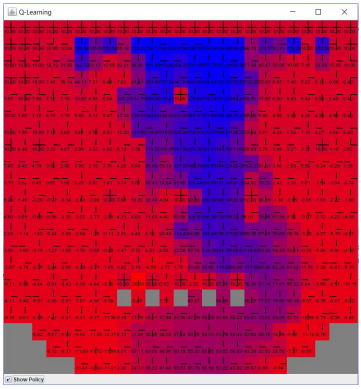
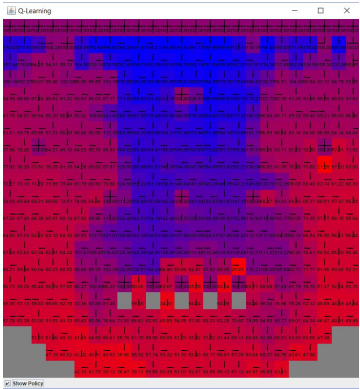
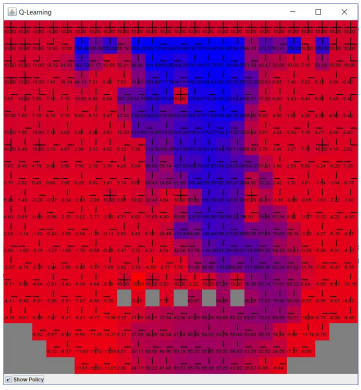
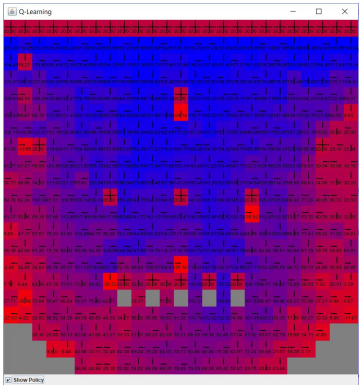
params	Q Learning (Qinit = 50, learningRate = .5)	Policy Iteration	Value Iteration
0, .5			
.5, .5			
.5, .25			

## Policy Iteration and Value Iteration:

For the most part policy iteration and Value iteration converged to similar answers. I used the same termination reward. As the “yardMultiplier” increased, both policies learned to avoid the sides and try to run through the middle and to avoid the defenders by running around them.

## Q-Learning:

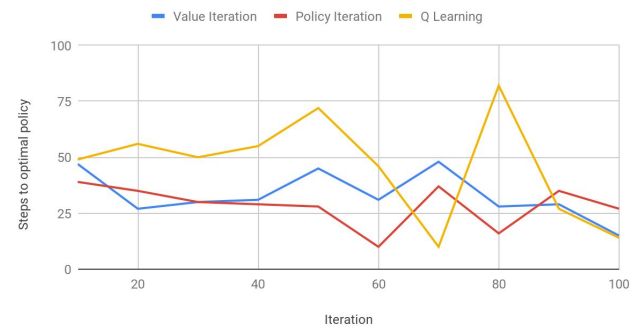
Q-learning converged to a different result from Policy and Value iteration. Q-learning is often used to solve much more complex problems where the environment is unknown. Q learning appears to “hold on” to the first solutions that it finds and diverges from there. We can clearly see that the Q-values propagation is not uniform and veers toward a certain a path that it knows leads to reward. I iterated Q learning 1000 times to ensure that it converged as it wasn’t converging with the same 100 iterations that I gave Policy and Value iterations. I also played around with the learning rate and Initial values for the Q-learner to see how the effects on the algorithm and what it learned.

	10 initial Q, .5 learning	100 initial Q, .5 learning
The effect of changing the initial-Q value of every state is dramatic. Here we see the increasing the Initial Q value, as expected encourages more exploration which explains the larger “blue” region indicating states with high Q-values as a result of exploration.		
	10 reward, .5 learning	10 reward, .1 learning
The effect of changing the learning rate is also dramatic. Here we see the increasing the learning rate encourages more exploration which explains the larger “blue” region indicating states with high Q-values as a result of exploration.		

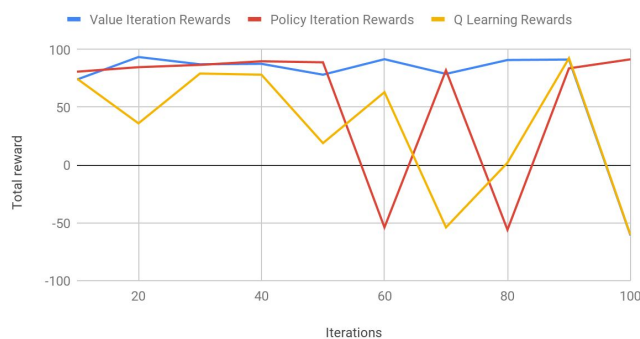


Here are a few graphs i generated that describe the training process of each of the algorithms over the iterations. I've attached the performance graphs as a result of the YardMultiplier parameter at 0.5 and the tackleMultiplier parameter at 0.5. This is only 1 of 3 different combinations of parameters for the Reward function. If you would like to see all the graphs, click on the link of the Aggregated Data below. All the algorithms used the same reward function and iterations of Q-learning was 1000 times that of Value/Policy Iterations to ensure convergence. For example iteration 10 on the graph to my right represents iteration 10 of value/policy iteration but iteration 10000 for Q-learning.

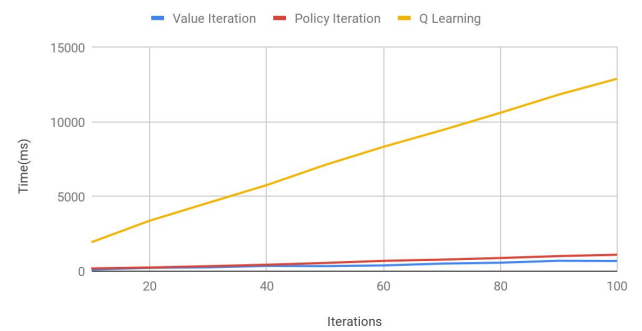
Number of steps agent needed to reach optimal policy



The optimal policy total reward gain



Time in Milliseconds the optimal policy needed



From the figures above we can observe similar patterns noticed previously. The steps the agent needed to reach the terminal state for all iterations was erratic from the stochasticity in the environment. Q-learning was the most erratic in terms of how many steps it required since it took 'exploratory' actions to learn more about the environment. If we look at the optimal policy total reward gain graph, the maximum reward gain is pretty similar between all the algorithms however all the algorithms appear to take penalties as a result of running into defenders from the stochasticity of the actions. In terms of training time, Q-learning took the longest time to train followed by policy iteration and value iteration. Policy Iteration on average takes the shortest time if implemented correctly since Policy Iteration requires the solution to a linear system, something a software package like MatLab is optimized for however I used BURLAP, which is implemented in Java, which may explain why policy iteration within this implementation took longer than value iteration.

## Conclusion:

I learned a lot about reinforcement learning and ways to solve Markov Decision Processes from this project. Overall for my two finite MDP problems, Value Iteration and Policy Iteration outperformed Q-learning. Q-learning of course would outperform an MDP problem where the probabilities or rewards were unknown since Value Iteration and Policy Iteration cannot solve those problems since they both require known rewards and probabilities. This demonstrates the “No Free Lunch” theorem where certain algorithms outperform others in one domain of problems yet fail in others. Q-learning was an active learner throughout the whole project and was constantly trying to obtain domain knowledge about the environment whereas Value Iteration and Policy Iteration were offline planners and did not need to learn from trial and error the optimal policy.

Increasing the number of states increased the total time required to train as well

Q-learning is much more applicable because it does not have the constraints that value iteration and Policy Iteration have. For example, in a problem like trying to train a self-driving vehicle, let's say we're trying to find the optimal path from point a and point b. If we were to try and solve this problem using Policy or Value Iteration, we would need to know the exact probability of traffic and cost of taking a certain road for and so much more information. In the real world, we don't have a highly accurate model of things like traffic which may change from day to day, year to year. With something like Q-learning it is much more practical since it will automatically adapt and learn from trial and error.

Link to View aggregated Data.:

<https://docs.google.com/spreadsheets/d/1g1NYKKtSrEkqXpxwjQ0FsYIRN8NpgFF6627coga3LAM/edit?usp=sharing>

Google Docs Link:

[https://docs.google.com/document/d/1yk7oPdIvWv2uJpqYkKAdQ8NUDADpPGK\\_sRa-di3Gww/edit?usp=sharing](https://docs.google.com/document/d/1yk7oPdIvWv2uJpqYkKAdQ8NUDADpPGK_sRa-di3Gww/edit?usp=sharing)

## Resources:

Alzantot, Moustafa. “Deep Reinforcement Learning Demystified (Episode 2) - Policy Iteration, Value Iteration and...” *Medium.com*, Medium, 9 July 2017, [medium.com/@m.alzantot/deep-reinforcement-learning-demystified-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa](https://medium.com/@m.alzantot/deep-reinforcement-learning-demystified-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa).

Zettlemoyer, Luke. *Markov Decision Processes (MDPs)*. [courses.cs.washington.edu/courses/cse473/11au/slides/cse473au11-mdps.pdf](https://courses.cs.washington.edu/courses/cse473/11au/slides/cse473au11-mdps.pdf).