# JHU AAP EN.605.649.83.FA21 Project 5 Report:

# Value Iteration / Q Learning / SARSA

**Mauro Chavez** Mauro.antoine.chavez@gmail.com | MCHAVEZ8@JH.EDU
*AAP M.S. Bioinformatics Student* *(858)354-7465*
*San Diego, CA*

**Abstract**

This document summarizes the fifth project deliverable for Introduction to Machine Learning taken through Johns Hopkins AAP in Fall of 2021. This project involved implementing reinforcement learning algorithms including Value Iteration, Q learning, and SARSA. These reinforcement learning algorithms were applied to a simple model describing a race car trying to go from the starting line to the finish line of a racetrack in as short amount of time as possible. The car could occupy states on the track within its boundaries and take actions in the form of modifying its velocity by increasing or decreasing it by 1 unit. At each time step, the car had the potential to make adjustments to both the x and y components of its velocity independently to change its speed or direction. Training data came in the form of text files that mapped out three tracks in ascii characters. Value Iteration was by far the best algorithm in terms of performance however it requires knowing parameters of the Markov Decision Process that may be unavailable in the real world. SARSA pulled slightly ahead of Q Learning in tests. When crashing meant restarting the race as opposed to stopping, Q Learning required two orders of magnitude more training iterations to come up with a decent policy. Value Iteration on the other hand was able to come up with an optimal policy regardless of crash policy but revealed that the reset policy roughly doubled the time required to traverse a course.

Video demonstration of code can be found here: https://youtu.be/32PslcwZ9XQ

## 1 Introduction

This assignment involved implementing three reinforcement learning algorithms to discover the optimal path for a racecar through a racetrack. This problem can be described as a Markov decision process with states being positions/speeds occupied by the racecar at a given time, actions being adjustments made to a racecar's speed, transition state probabilities describing how likely an adjustment is to be successful, a reward function penalizing each time step spent not at the goal state, and a discount factor to guide learning activities. Value Iteration took a dynamic programming approach to solving this problem by calculating scores for each action at each position on the track per iteration. Q learning and SARSA differed by letting a car take random paths through the track and learning from each attempt to inform future ones. After enough attempts, a good path through the track could be generated from previous experience. Each of these learning algorithms were tested on three sperate tracks (one shaped line an L, one shaped line an O, and one shaped like an R) and experiments were ran to determine the relative performance of each with respect to the number of iterations allowed for training.

### 1.1 Problem Statement

The high-level goals of this assignment were to implement three reinforcement learning algorithms to help a racecar navigate a series of racetracks. The state of the racecar at each time interval could be described by an x coordinate, y coordinate, x velocity, and y velocity. At the changing of each time interval, the x and y velocities would be applied to the racecar's position and it would move x velocity units in the x direction and y velocity units in the y direction. However, prior to each application of the x and y velocities at each time step, the car could alter them by applying an acceleration. The acceleration would come in the form of adding a +1, 0, or -1 velocity delta to each dimension separately. If the car were to run into the wall of a racetrack during its movement, two crash policies were tested. The first sent the racecar back to its last non-wall position while zeroing its velocity. The second would send the racecar back to its starting position. Both these methods were tested to quantify the cost of attempting to learn the more punishing policy on only the R track. Learning the policy of optimal actions to be taken at each potential state occupied by the racecar was the ultimate goal of the application of reinforcement learning algorithms. To complicate matters, there was a 20% chance that each action taken per time step failed to adjust the cars velocity, leaving the x and y components unchanged. This added a small amount of non-determinism to the model. For the purposes of this assignment, the race car always started the race at the same start position per track. To solve this racetrack navigation problem, three reinforcement learning models were implemented and tested for performance; Value Iteration, Q Learning, and SARSA.

### 1.2 Hypothesis

A feature of Value Iteration is that it leverages knowing the structure of the Markov model prior to learning. Q Learning and SARSA do not have the same advantage. Value Iteration knows upfront the transition state probabilities while Q Learning and SARSA must figure those out themselves. Therefore, Value Iteration will likely outperform Q Learning and SARSA across all tracks in terms of requiring less learning time to uncover an optimal policy. Furthermore, the learned Value Iteration policy will be more resilient to non-deterministic actions. Generally speaking, Value Iteration will require iterations through the state space to learn the track proportional to the distance between the start and finish lines on the track. That being said, the upper limit on required Value Iteration passes through the state space to learn the optimal path will be the number of time increments required to traverse the optimal path. In general, it will require many more iterations/attempts for SARSA and Q Learning to learn the optimal path than value iteration. SARSA will beat out Q Learning as it more accurately describes Q(S, A) by picking an action at the resulting state using a method more representative of how the learning process unfolds. The complexity/iterations required to learning each track for each algorithm will be proportional to the size of the track. In regard to testing the crash policy, getting sent back to the start will require an order of magnitude more time to learn an optimal policy.

## 2 Data Pre-Processing

Racetracks were provided as text files. These files all followed the same format where the first line contained a comma separated string of two integers. The first integer reported the height of the track and the second reported the width. All other lines following the first mapped out the shape of the track in a grid. " # " characters corresponded to illegal states on the track while " . " characters designated legal ones. For the purposes of this project, " # " characters represented walls that could be ran into by the racecar. " S " characters signified start positions and " F " characters signified goal/end positions. For this implementation, racetracks were objects in memory and the course itself was stored as an instance variable of the racetrack in the form of a list of lists. Racetrack objects themselves contained methods for printing the track, generating lists of legal/end/start states, and validating moves across the track. Each row in the track file following the first became a list where each item in the list stored a single character in the row. The rows were then stored in a list to represent the track matrix data structure. Due to this implementation, indexing an (x, y) coordinate required interfacing with the track matrix in format [y][x]. Moving from left to right increased the x coordinate while moving from top to bottom increased the y coordinate. In other words, position (0, 0) was the top left location in the track and position (track_length)(track_height) was the bottom right of the track. A positive x velocity meant moving to the right and a positive y velocity meant moving down.

### 2.1 L track

This track was 11 x 37 units large and all walls in the track ran parallel to the x axis or parallel to the y axis. It was the simplest track in terms of layout and also the smallest in terms of size with only 160 legal coordinates for the racecar to occupy. Here the stretches of track were the widest with walls spaced the farthest distance apart as compared to other tracks.

### 2.2 O track

This track was 25 x 25 units and mapped out the shape of an O. It was the second largest track in the data set with 220 legal coordinates for the racecar to occupy. Here the track was generally much narrower than the L track with legal coordinates being much closer to walls on average.

### 2.3 R Track

This track was 28 x 30 units large and roughly takes the form of an R. It was the largest track with 293 legal coordinates for the racecar to occupy. Corridors along this track were generally as wide as those in the O track except they almost never went in a straight line for too many units. This track curves and bends such that going in a straight line perpendicular to the x or y axis was not possible for many consecutive steps.

## 3 Experimental Approach

### 3.1 Reinforcement Learning Algorithms Applied to the Racetrack Problem

The reinforcement learning algorithms implemented for this assignment included Value Iteration, Q Learning, and SARSA. For each model, the reward for taking any action at any state that did not land at the finish line came with a cost of -1. An action from a state that resulted in landing on the finish line had a cost of 0, making each position on the finish line absorbing states. This structure had the effect of tasking each model with learning to complete the course in as few steps as possible. Value Iteration follows a dynamic programming approach, repeatedly iterating through every state on the racetrack and slowly back filling the optimal move starting from the finish line back to the start line. Roughly speaking, on iteration i, it would have calculated the optimal paths i steps from the finish line ignoring non-deterministic actions failing to take the optimal policy for a given state. For the purposes of this assignment, Value Iteration was implemented to undergo a predefined number of iterates through the state space. Once all iterations were complete, the policy would be tested. Value Iteration is a special case of reinforcement learning as it requires knowing the Markov Decision Process model ahead of time and can leverage knowing transition state probabilities. Q Learning and SARSA on the other hand are different reinforcement algorithms that learn the state space through trial and error. They both let the racecar begin by taking random paths through the track and slowly learn from each attempt. The goal here as learning progresses is to transition from taking mostly random paths to mostly learned optimal paths. For the purposes of this assignment, this transition in action selection was accomplished using epsilon greedy search to pick which action to perform at a given state. Each time an action was to be selected, with probability epsilon, a random action would be chosen. With probability 1-epsilon, the optimal action that maximized Q(S, A) would be chosen. Epsilon was initialized as 0.95 and for each attempt to traverse the track was multiplied by a scalar defined as ((total_allowed_attempts – current_attempt) / total_allowed_attempts). This had the effect of going from picking more random actions during early attempts to navigate the track to picking more optimal ones in later attempts. The Q Learning algorithm differed from SARSA in how it calculated Q(s, a) for each given state along a track traversal attempt. When taking action *a* from state *s*, Value Iteration uses the maximum valued action at the resulting state to update Q(*s, a*). When doing the same calculation, SARSA picks the action at the resulting state in the same way that it picked the action at the current state – using the epsilon greedy selection policy described above.

For Value Iteration, the discount rate was fixed at 0.80. For Q Learning and SARSA, the discount rate was increased to 0.90 and they operated learning rate of 0.25 initially. As noted earlier, the search policy followed epsilon greedy selection where for each attempt through the state space, epsilon was scaled by a factor of (current attempt / maximum allowed attempts).

To quantify how well an algorithm was able to define an optimal policy for a track after learning, 10 attempts through the track were "timed" and averaged. A max time allowance was set so that if a car was still navigating the track after 400 time steps it would terminate.

## 3.2 Comparing Stop Policies on R track

Value Iteration was tested on the R track by applying the algorithm under both crash policies for seven different max iteration parameters. These were; 10, 20, 28, 50, 60, 100, 200. As noted earlier, the max iteration parameter would define how many passes value iteration would make through the state space before the policy was tested. When studying Q Learning's ability to learn the R track, an attempt would end once a crash into a wall occurred and the car was sent back to its start position. Q learning was tested under both crash policies for 18 different max allowed attempts parameters. This was necessary as the parameters used to learn the stop policy had little relevance to the parameters required to learn the reset policy. Due to time constraints, SARSA was not tested under both crash policies.

## 3.3 Comparing Algorithm Performance

Each Algorithm was tested on each track using the stop crash policy. For both Q Learning and SARSA, the tested variable would be the allowed attempts through the state space before timing the efficiency of the learned policy. An attempt would end once the finish line was crossed. This would be compared to the number of iterations through the state space allowed for Value Iteration to learn the optimal policy.

**Experimental Results**

## 3.4 Stop Policy Comparison on R Track

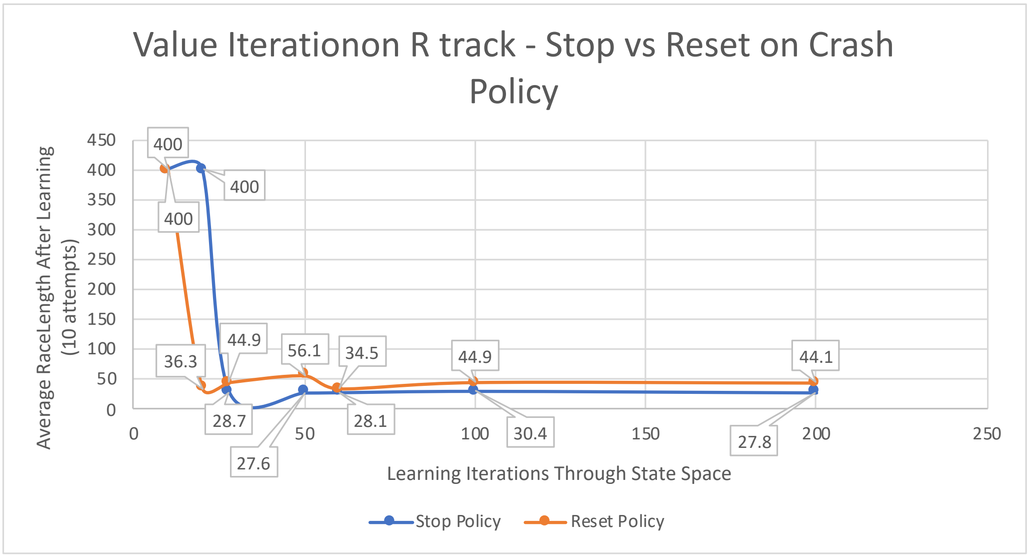

*Figure 1: Value Iteration on the R track under both crash policies testing different values of how many iterations through the state space would occur before the learned policy was tested.*

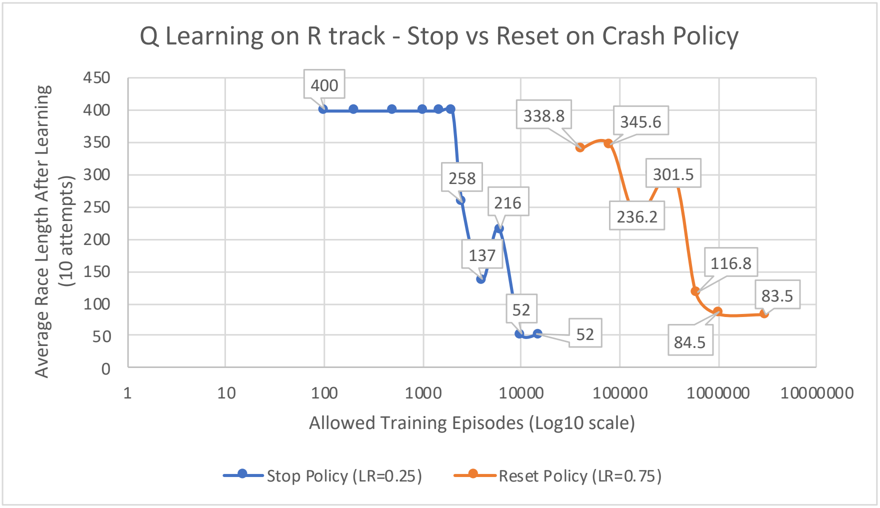

*Figure 2: Q Learning on R track under both crash policies testing different values of how many learning attempts through the state space would occur before the learned policy was tested.*

## 3.5 Algorithm Comparison

The general story behind the comparison of each reinforcement learning algorithm is that Value Iteration was able to learn an optimal policy after far fewer training iterations and the resulting policy was much more resilient to non-deterministic actions steering the racecar off course. Q Learning and SARSA would converge to learning good policies (comparable to the performance of Value Iteration) but the average time to complete the course would be hurt by situations where non-deterministic actions pushed the racecar into states that the algorithms had not learned. In these situations, the car would get stuck and report the max allowed time trial value (400). In figure 3, SARSA averaged a time of 133.1 after 10,000 allowed training iterations to learn the L track. However, when looking into the set of 10 time trials averaged to get this score, they were; 16, 400, 17, 400, 22, 16, 400, 16, 23, and 21. Most of these scores get close to competing with the performance of Value Iteration. However; on three of these attempts the car ventured into states where SARSA hadn't learned an optimal policy. The car got "stuck" and failed to complete the race after 400 time steps. This is a general trend for Q Learning and SARSA across algorithm comparison on each track. Once these algorithms were able to start reporting times that competed with value iteration, their resulting average time is more of a reflection of how many times a car got stuck in a time trial. In other words, after a certain point when Q Learning and SARSA converged to generally having a good policy for traversing the track, their average race length was more a reflection of how resilient the policy is to non-deterministic actions.

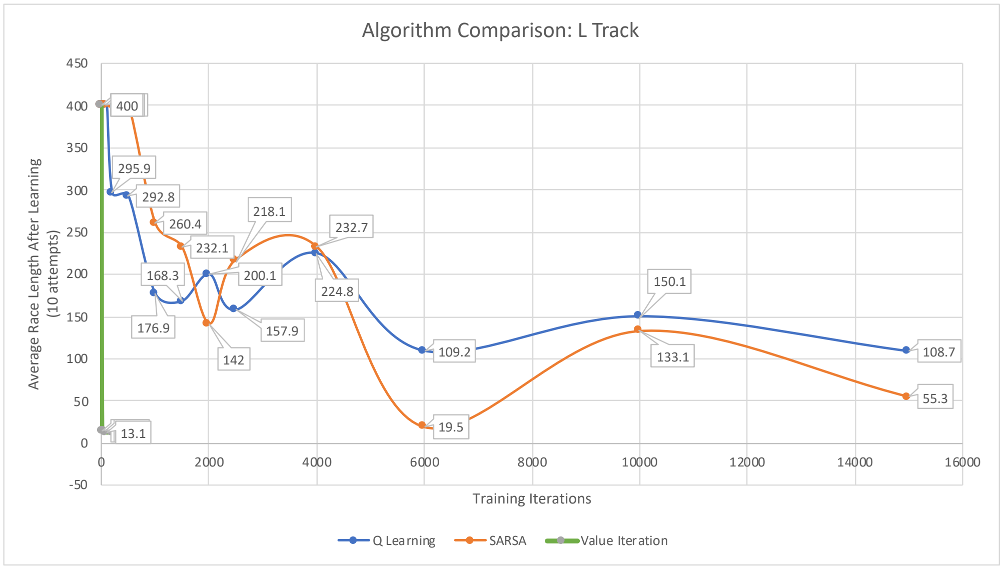

*Figure 3: Algorithm comparison on the L track following the "stop" crash policy. Value Iteration converged to an optimal policy with average race time between 13 and 14 after 20 allowed training iterations where more iterations did not improve performance.*

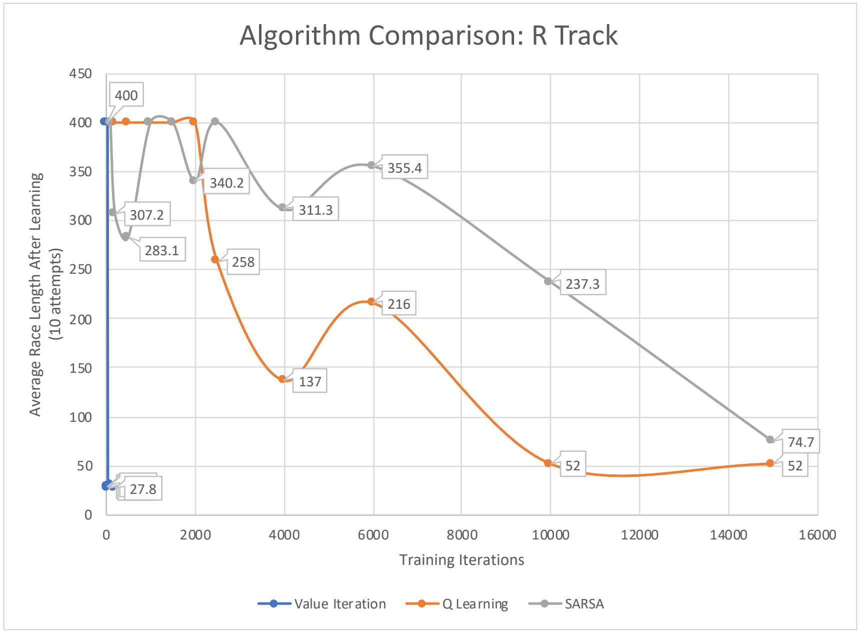

*Figure 4: Algorithm comparison on the R track following the "stop" crash policy. Value Iteration converged to an optimal policy with average race time between 27 and 30 after 28 allowed training iterations where more iterations did not improve performance.*

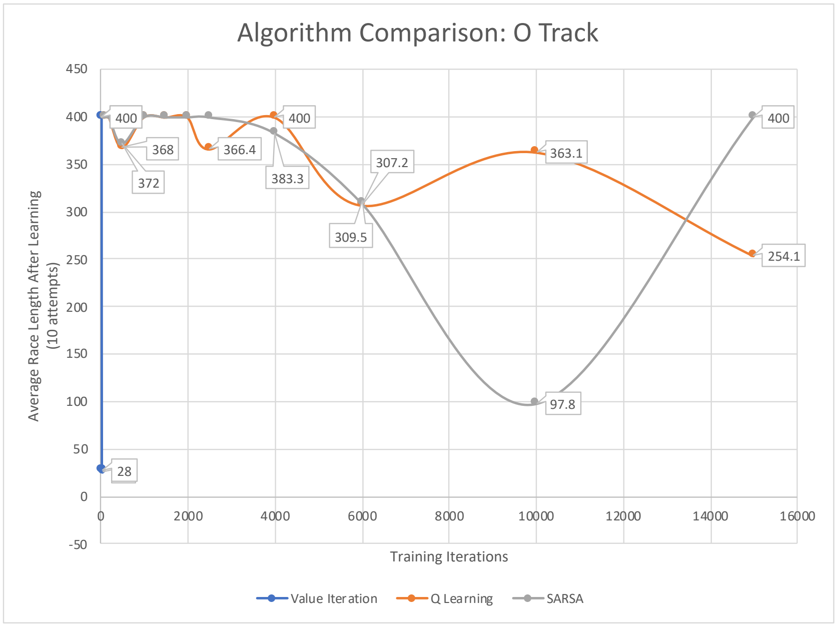

*Figure 5: Algorithm comparison on the O track following the "stop" crash policy. Value Iteration converged to an optimal policy with average race time between 26 and 28 after 28 allowed training iterations where more iterations did not improve performance.*

## 4 Discussion

From the results above, Value Iteration clearly comes out on top with respect to quickly learning an optimal policy that is robust to correcting for not deterministic actions steering a racecar off course. Not only did Value Iteration converge to optimal policies quickly for each track, but once it did, there were no 400 time attempts. Furthermore, Value Iteration would often be completely unable to perform until it hit a specific allowed iterations value. Once hitting this value, convergence to an optimal policy would be almost immediate where further increases to allowed iterations would not improve performance. This optimal allowed iterations threshold was proportional to the size of the track being learned. Changes in performance after the optimal allowed iterations threshold was reached are attributable to non-determinism in the racetrack model. With respect to the stop vs reset policy on the R track, it takes almost twice as long to traverse a racetrack following the reset policy as it does to traverse a racetrack using the stop policy. From earlier testing, the learned stop policy strategy seemed to go at max speed across the track and crash into the wall. Then the car would start again going towards the finish line. Following the reset policy the model couldn't use crashing into a wall as a tool to quickly change the direction of its x and y velocities. This likely resulted in much more careful behavior that ended up slowing the car down. In general, the performance of Value Iteration makes sense as it leverages knowing features of the Markov Decision Process prior to learning that are not available to Q Learning and SARSA.

With respect to Q Learning following the stop vs reset policy on the L track, the difference is much greater than anticipated. When following the stop policy, Q Learning converged after 15,000 training iterations. The rate at which performance improved slowed down dramatically after 10,000 but testing larger values may have slowly converged into the optimal number of steps reported by Value Iteration. When following the reset policy, Q Learning did not converge until 1,000,000 training iterations. Even so, once the rate at which performance improved slowed down, it was still pretty far from the optimal number of steps reported by Value Iteration for the stop policy. Here it could have taken upwards of 10,000,000 training iterations for Q Learning to get the same performance as Value Iteration. As discussed in the results, the performance of Q Learning and SARSA once they began to converge is less so an artifact of slower policies and more so attributable to the car getting stuck on some time trials. The car would navigate into a position with poorly optimized policy and end up unable to finish the race. This would result in Q Learning or SARSA reporting ~7-8 good time trials nearing Value Iteration performance where 2-3 failed attempts would skew the average race time by adding scores of 400. To account for this, the algorithms could be allowed to learn more or the method of quantifying performance could be improved.

SARSA was only able to outperform Q Learning on the L and O tracks. The two algorithms are the same with the exclusion of how they update Q(s, a) as discussed in section 3.1. It is interesting that Q Learning was able to beat out SARSA on the R track, but repeated tests may end up revealing the opposite. Performance of these algorithms was very dependent on how many non-deterministic actions took place during time trials. To help mitigate this and collect more accurate results, it would make sense to do 100 time trials instead of 10 when quantifying policy performance. Surprisingly, the O track was the hardest track to learn for these two algorithms. SARSA seemed close to getting a good policy at one point but it may have just gotten lucky. Running more tests would be required to tease out why exactly Q Learning and SARSA had such difficulty with this track. It was unique in that it had four 90 degree turns which requires very careful tweaking of the car's velocity. Further testing could take the form of increasing the allowed training attempts and tweaking learning parameters such as discount rate and learning rate.

## 5 Conclusion

Value Iteration's performance relative to Q Learning and SARSA is important to note as Value Iteration seems to have less utility. The Value Iteration algorithm requires knowing the Markov Decision Process prior to learning which is likely a very challenging requirement to meet when working with real world examples. It may be possible to experimentally map out transition probabilities for a learning scenario, but then it may very well make more sense to just start with Q Learning and SARSA. From this analysis, it is clear how difficult it is to learn these transition probabilities from scratch. It makes me appreciate the unknowns involved in reinforcement learning and the computational work required to elucidate them. Switching from the reset crash policy to the stop policy was a critical component to the feasibility of this analysis. Q Learning required 2 orders of magnitude more training iterations to learn a decent policy following reset vs stop crash protocols. The reported performance of Q Learning and SARSA reveals a trend but testing more parameters would have been interesting. I had initially set a learning rate and began to collect data but once realizing it was underperforming changed it from 0.25 to 0.75. Tweaking learning rate and discount rate could reveal better performance even for the same number of allowed iterations. No variable learning rate was used as adjusted epsilon greedy search provided Q Learning and SARSA with the ability to transition from taking more random paths to taking more learned paths. However, there may be gains in performance to be made from adding in a variable learning rate on top of that.