1. ***In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location? (basic agent)***

The basic agent acts randomly, making no rational strides toward its goal and making no effort to obey stop lights and avoid traffic. It eventually makes it to its target location if deadlines are note enforced, but does so quite slowly and inefficiently.

2. ***Choose state values: Justify why you picked these set of states, and how they model the agent and its environment.***

I defined my agent's state as [Traffic_Light, Car_Left, Car_Ahead, Car_Right, Waypoint] because these are the inputs that most immediately affect the score of a cab's next maneuver. Knowing these values, the cab should be able to learn to obey the basic rules of the road, obeying stop lights and collisions with nearby vehicles. The key feature that I left out of the cab's state was the deadline. Allowing each eligible deadline value (from 0 to 30) to be a part of the state space would cause the number of possible states to greatly expand without adding significant value to the agent, making it slower to train without adding anything particularly useful to its knowledge base. If the agent is learning the optimal routes to its target, the time remaining should not matter after a short period of training. The end result is a state that represents the spatial environment directly around the cab, where any obstructions might lie, and the waypoint, representing the general direction in which the cab should be moving and where its immediate obstacles lie.

3. ***Implement q-learning: What changes do you notice in the agent's behavior?***

The agent's behavior changes significantly upon implementing Q-learning. The agent begins similarly to the random agent, not appearing to have much reason behind its actions. This changes within a few trials, however, as the benefit of seeing more states can be clearly observed as the agent quickly becomes savvy to traffic rules and begins to reach its target consistently. This improvement is observed because the Q values are stored and updated over time, which means that the agent begins with a series of naive assumptions (i.e. that all states take on a default value), and learns the real values of the states as it perceives them, making the agent become progressively intelligent and successful in its environment.

4. ***Enhance the agent / parameter tune: Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?***

The parameters I sought to tune were learning rate, discount factor, and epsilon. To test these, I cross validated on three folds for each of the parameters, holding the others constant. I then evaluated the performance based on the number of successful trials (all out of 100), observing penalty patterns as well.

For learning rate, I tested the values 0, 0.3, 0.45, 0.6, 0.75, 0.9, and 1.2. 0 and 1.2 were clearly the worst of the set, with the others producing success rates in the 80-90% range. 0.75 proved best, as is shown in the plot below.

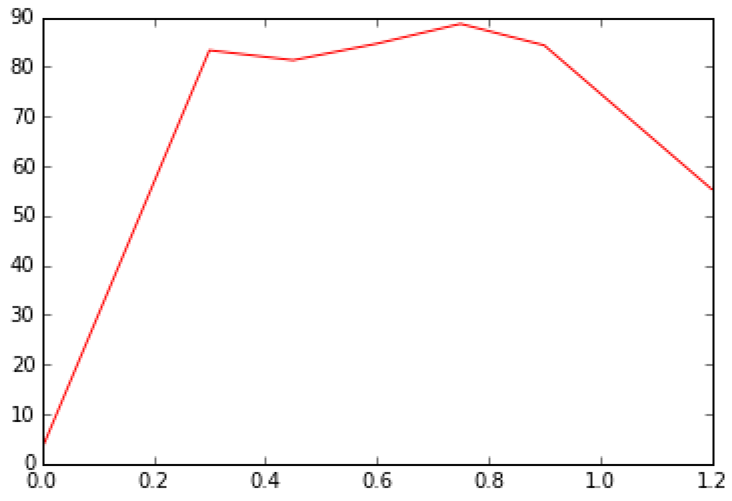

Figure 1: Average number of successful trials by learning rate

For the discount factor, I tested the values 0.1, 0.3, and 0.5. 0.5 was a clear throwaway, and 0.1 and 0.3 produced similar penalty patterns and success rates, with 0.3 succeeding in 90.33% of trials and 0.1 succeeding in 91.33%. A plot of the success rates is shown in figure 2.

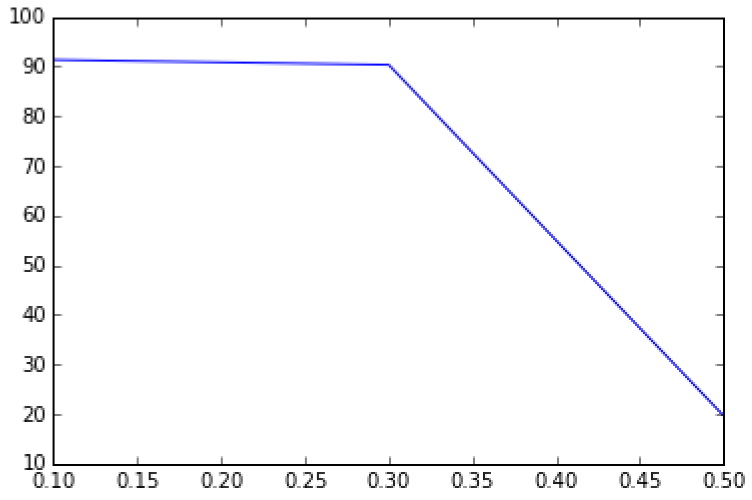

Figure 2: Average number of successful trials by discount rate

The last standalone factor I tuned was epsilon, testing values of 0.05, 0.1, 0.2, 0.3, and 0.4. The penalties of the simulations followed the pattern one would expect, with a higher epsilon leading to more penalties due to random moves being more likely to break traffic rules than learned ones. With the success patterns, 0.1 proved best, with the success rates improving from 0 to 0.1, and then steadily declining thereafter, with the 0.1 success rate being 89.67%. These results are shown in figures 3 and 4.

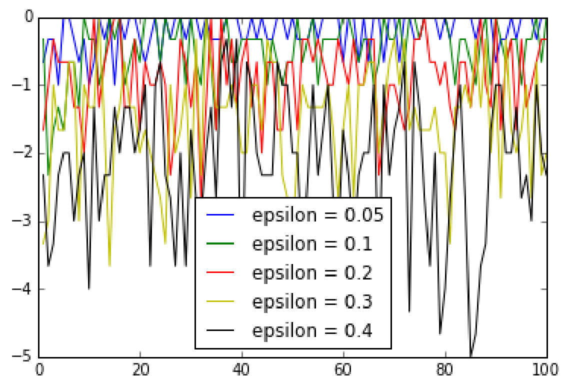

Figure 3: Average penalties by trial

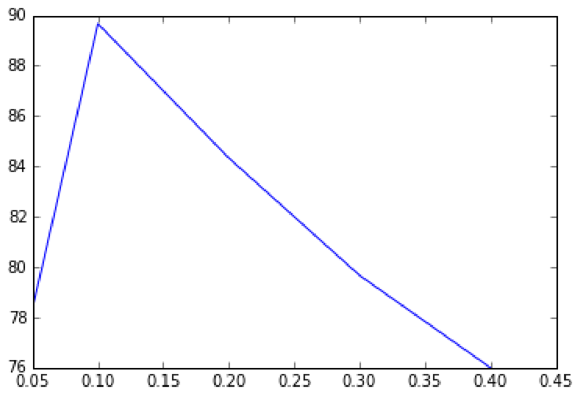

Figure 4: Success rate by epsilon value

So, from this initial stage of parameter tuning, the best setup was learning rate = 0.75, discount rate = 0.1, and epsilon = 0.1.

To tune the model further, I then implemented learning rate and epsilon decay. Learning rate decay is needed so that, as the model learns more, it is able to place more weight on the values it has already learned relative to what it is observing in the present. This allows it to learn faster early-on, and then later rely on this knowledge for an improved level of precision in its decision making. The updating formula I used was ((initial alpha * 10) / (10 + trial number)).

Epsilon decay is needed for a similar reason. The agent needs this element of randomness early on so that it explores the state space and is able to expand its knowledge of the various possible moves it can make and their corresponding rewards. As the agent learns and becomes more reliable, however, the utility of these random moves decreases, and the errors caused by the randomness no longer have the same positive side effect of teaching the agent important new information. Epsilon decay solves this problem by giving the agent a high degree of randomness initially, and then waning this randomness parameter epsilon down as the trials increase, allowing it to take advantage of what it has learned in later trials and perform with low numbers of penalties. This idea comes across well when represented graphically (Figure 5). When the agent uses a constant epsilon, it will always have consistent errors in later stages. When this epsilon decays, however, the agent is able to learn from the randomness in early trials and avoid the corresponding randomness-caused errors in later trials.

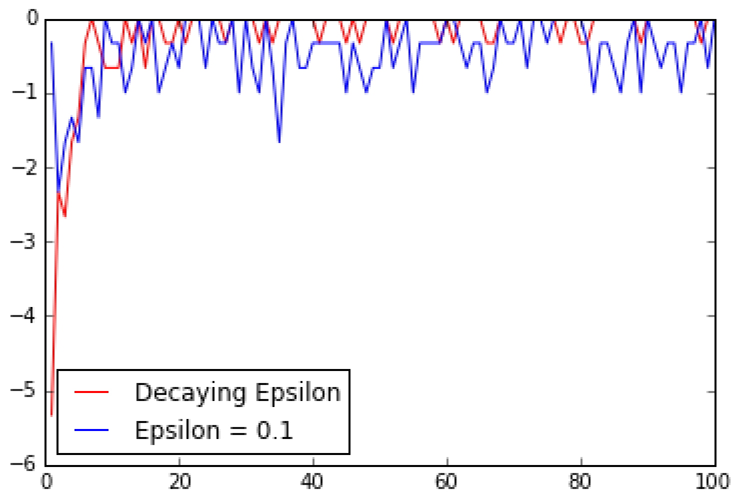

Figure 5: Penalties by Trial with decayed and undecayed epsilon values

As this figure shows, the decayed epsilon displays far better results in the later rounds, making noticeably less errors, resulting in a higher success rate. After applying both epsilon and learning rate decay, the agent's average successes per 100 trials improved from 87 to 93.67. Throughout all of these, net reward remained consistently positive. I did not use this as a significant component of my parameter tuning, however, because its results were noisy and unreliable due to the fact that reaching the target quickly would actually render a lower score than reaching it slowly in some situations.

5. ***Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties?***

While it is imprecise to define an optimal policy in this type of dynamic environment, the quality of the policies learned can be generally assessed by error minimization, success frequency, and time required to reach the target. An optimal policy would make zero penalties and reach the target via the shortest path. I initially thought to measure cumulative rewards by trial as a way of judging the quality of the agent's policies, but then realized that this would actually penalize agents for arriving at the goal faster in certain cases, making it a flawed measure. With this in mind, it appears as if the cab does well in learning a policy that combines obeying the rules of the road (i.e. minimizing penalties) and achieving its goal reliably and quickly. As is seen in Figure 6, the agent's number of moves required to reach its goal follows a pattern that decreases some with time, but reaches a plateau within the 100 trials attempted. Since this number of trials is relatively small, the cab will not see every game state, and will therefore not reach the optimal policy. It does, however, reach a passable one, learning to reach the target with high accuracy (>93%) while racking up very few penalties.

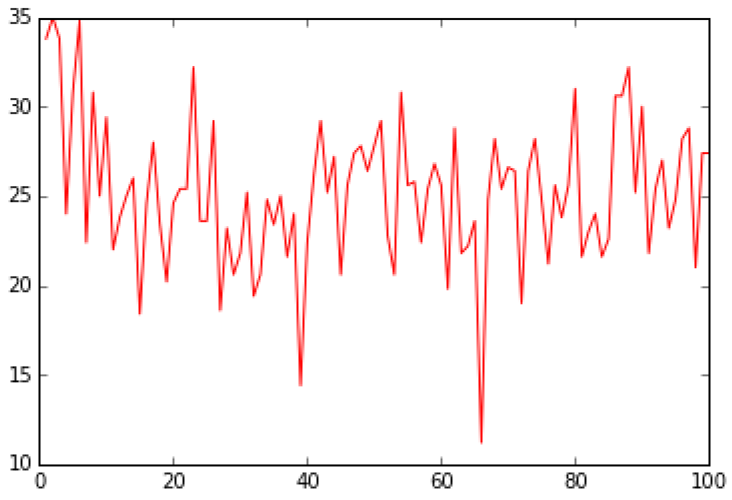

Figure 6: Turns required by agent to reach its goal per round

Figure 6 shows that the agent makes meaningful strides toward an optimal policy by way of finding shorter paths to its target, although it does not appear to attain perfection in this within its 100 trials. Figure 5 shows the significant strides that the agent makes in penalty minimization with the implementation of epsilon decay, with avoiding penalties appearing to be a simpler task for the agent than finding wholly optimal paths.