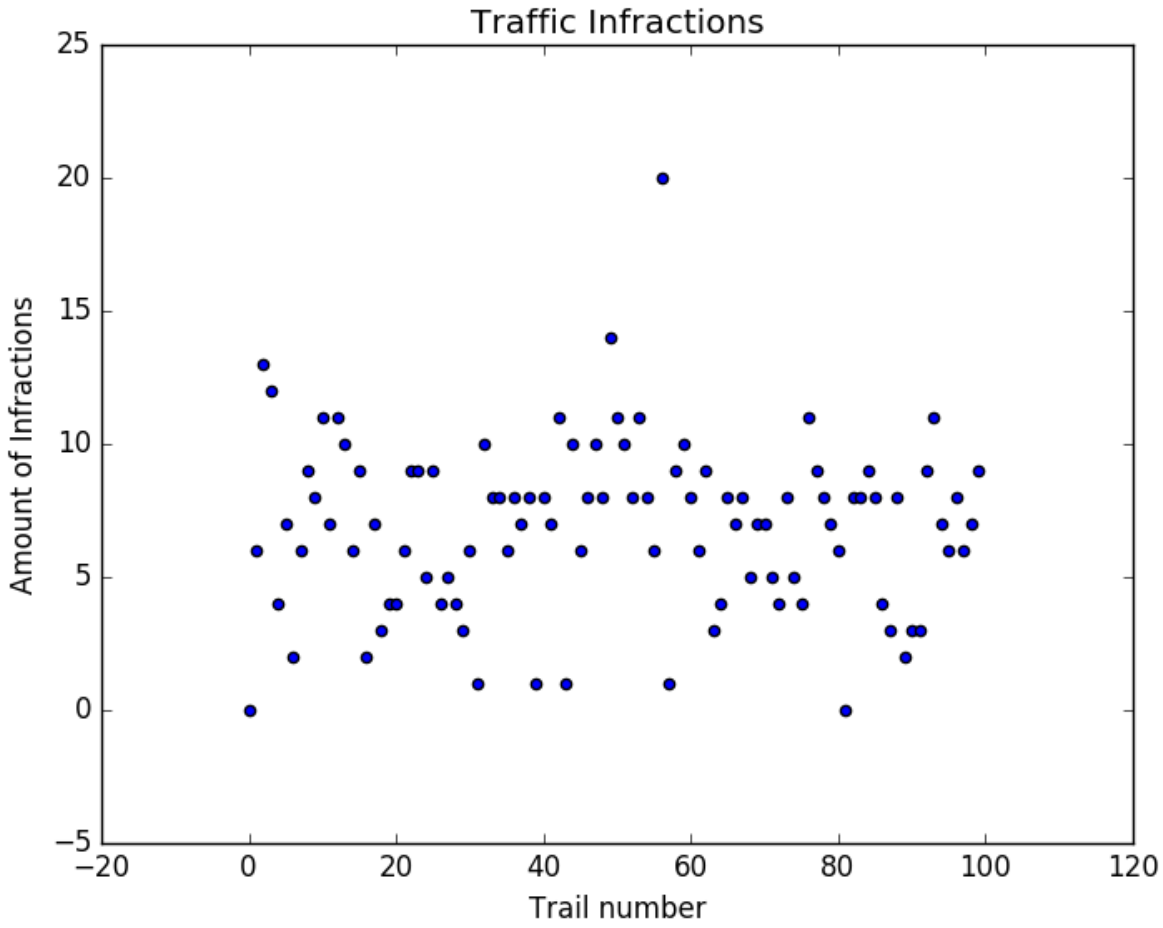
Reinforcement Learning Submission

QUESTION: *Observe what you see with the agent's behavior as it takes random actions. Does the* **smartcab** *eventually make it to the destination? Are there any other interesting observations to note?*

The smart cab, given infinite time, would eventually make it to the goal. But in finite time, since it is taking psuedo-random movements, there is no reason to expect that it would make it to the goal.

Over 100 trials, the car's amount of traffic infractions (reward = -1) can be visualized as follows:



It can be seen that the car commits roughly 5-10 errors per trial when it is doing completely random movements. It also rarely reaches the goal.

QUESTION: *What states have you identified that are appropriate for modeling the* **smartcab** *and environment? Why do you believe each of these states to be appropriate for this problem?*

The states given to the agent as inputs are the status of the lights at the intersection, the oncoming traffic status, the right traffic status and the left traffic status. The input is given in dictionary format as shown in the following example: *{'light': 'red', 'oncoming': None, 'right': None, 'left': None}*. I have

chosen to use the ‘light’, ‘oncoming’, and ‘left’ inputs as states to model the smartcab. In addition, I have chosen to use the ‘next_waypoint’ property of the agent as a state to model the smartcab.

The reason that the ‘light’ property is important is that the Agent needs to know the status of the light in front of it in order to make a decision of what to do next. If the light is red, it might be more optimal to go right instead of waiting for the light to turn green. If the agent were to break this rule, it would result in a large negative reward value.

The reason the ‘left’ property (I.e presence of traffic from the left) is important is because if the light is red and there is traffic present on the left, the car must wait for the left present car to pass through before it can safely make a right turn.

The reason the “oncoming” traffic is important is because if our smartcab needs to make a left turn on a green light, it must wait for oncoming traffic to pass before it can safely do so.

Finally, the ‘next_waypoint’ state is needed so the smartcab knows which way to go. Different way points should be associated with different awards, for example a right turn should return a higher reward than a left turn as it is easier and safer to complete.

The ‘right’ property was not chosen as there is no situation where the smartcab really needs to car about the status of the oncoming traffic form the right. When the light is green, the cab has the right of way and go forward, right or left. When it is red, there is no chance of hitting the oncoming traffic on the right as the only option for the smartcab is to go right.

The ‘deadline’ property was not selected as it may influence our agent to make more risky decisions, as time is running down in order to reach the goal. It is more desirable to be a safe driver than to reach where you are going on time. Additionally, it would add a large amount of states to our possible state space.

OPTIONAL: *How many states in total exist for the* **smartcab** *in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?*

The total amount of possible states for the smartcab depends on: lights (2 states), oncoming (4 states), left (4 states), right (4 states) and the next_waypoint (3 states). This is a total of 2*4*4*4*3 = 384 states. For our particular environment, since we eliminated the right states, we have 384/4 = 96 states.

This number seems reasonable, as 96 states are not too much to keep track of even on my laptop. However; it is worth noting that reducing the amount of states as much as possible is desirable since the agent needs to to visit each state many times in order to approximate the bellman equation as well as possible. If there are a high amount of low probability states, the agent will likely need a large amount of trials in order to visit them.

QUESTION: *What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?*

The agents behavior has changed from the random behavior in that it actually attempts to go in the correct direction because of the ‘next_waypoint’ state variable. In the beginning of the trials, it makes a few traffic violations (reward score of -1); however, in later trials it learns to not make as many. This is because it is updating it’s q_table (state/action pairs mapped to q_scores) so that when faced with particular states, it is able to look at it’s own past q history to know how it’s decisions has been rewarded in the past. It can then make an action decision that maximizes it’s expected reward.

QUESTION: *Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?*

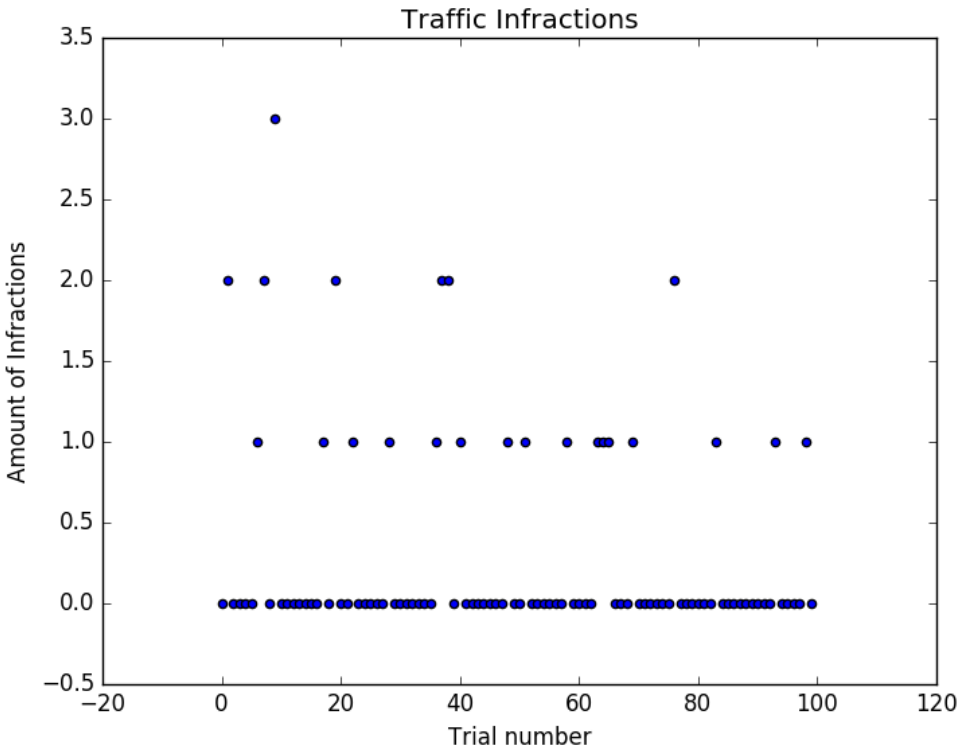
The final parameter used in my RL agent model are as follows:

- Epsilon (exploration variable for simulated annealing algorithm): 0.1
- Epsilon Annealing Rate (degradation rate for epsilon): 0.01
- Alpha (learning rate) : 0.65
- Gamma (discount rate for future q_max value): 0.35

The final performance of the agent using these parameters is as follows:

> **********FINAL REPORT:**********
>
> AMOUNT OF TIMES REACHED GOAL: 98
>
> TRAFFIC INFRACTIONS RECORD: [0, 4, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 3, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0]
>
> AMOUNT OF TIMES GOAL NOT REACHED 2
>
> TOTAL AMOUNT OF TRAFFIC INFRACTIONS: 33

Additionally, the Epsilon value is restarted to 0.05 after each trial. This is because the agent starts in a new located at each trial, and would benefit from additional exploration to try and mitigate the possibility of getting stuck in a local minimum. This results in a higher total infraction rate over the course of all the trials, but allows for the agent to learn more effectively. The agent performs poorly in the beginning of the trials (note the infractions record starting with 4 and 3 infractions) as it has a higher exploration rate and is still developing it’s q_table. This is also where it is more likely to not reach the goal. However, later in the trials, the infraction rate decreases to either 0 or 1 for each trial. The results can be visualized as follows:



As can be seen, the agent commits very few errors after it's initial training period of roughly 40 trials. This can be compared to the first figure showing it's random behavior.

In order to find the optimum parameters, the experiment was conducted in a grid search-like manner, trying many combinations of different parameters. Other parameters tried in the 100 trial experiment are summarized in the table below:

	Alpha	*Gamma*	*Epsilon*	*Epsilon_Annealing*	*Goals Met*	*Total Infractions*
1	0.5	1.0	0.5	0.01	15	55
2	0.5	0.8	0.5	0.01	57	33
3	0.5	0.6	0.5	0.01	82	43
4	0.5	0.4	0.5	0.01	97	32
5	0.4	0.4	0.5	0.01	96	35
6	0.6	0.4	0.5	0.01	94	28
7	0.5	0.4	0.4	0.01	95	27
8	0.5	0.4	0.2	0.01	96	28
9	0.55	0.35	0.1	0.01	97	32
10	0.65	0.35	0.1	0.01	98	33

QUESTION: *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? How would you describe an optimal policy for this problem?*

The agent is close to the ideal policy. It reaches the goal roughly 98% of the time on a trial of 100. An ideal training strategy would be if the agent were trained on a large amount of trials, then have it's reset 'epsilon' turned off (time to stop 'messing around'). In order to this described training methodology, an experiment was set up that had 1000 trials, while reseting the exploration variable epsilon for the first 200. The results are as follows:

AMOUNT OF TIMES REACHED GOAL: 994

TRAFFIC INFRACTIONS RECORD: [0, 5, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,

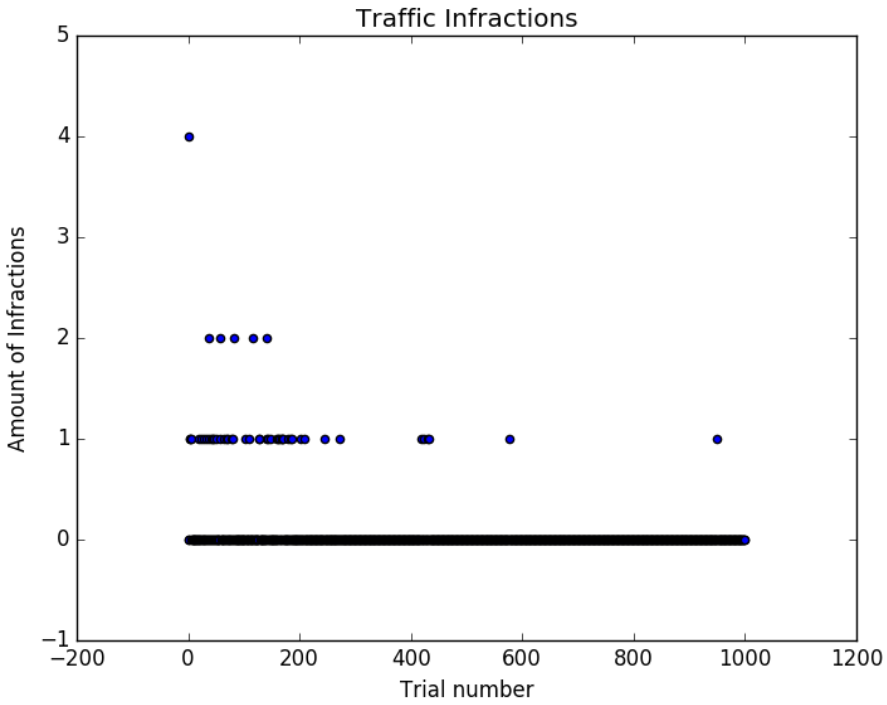
…

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

AMOUNT OF TIMES GOAL NOT REACHED 6

TOTAL AMOUNT OF TRAFFIC INFRACTIONS: 52

Our agent performs better then expected, reaching a 99.4% goal rate, while committing 0 traffic infractions in the majority of trials. This can be visualized in the following plot:



As can be seen in the plot, the agent has a small period in the first 200 trials where it makes a few mistakes. However, after trial 200, when the epsilon reset rate is turned off, it makes very few errors, and most of the time completes the trial without a single infraction.

For this problem I would describe an optimum policy as making it to the goal 100% of the time with 0 infractions. However, due to the mechanics of reinforcement learning this is not possible. You need to make mistakes in order to learn. Perhaps this is a philosophical lesson for life as well.

In that light maybe it would be better to describe an optimum policy as one that makes 100% of the trips with zero infractions after a certain threshold allowed for it to learn. In the case of the optimum model that we trained, there are only 6 infractions made after trial 200 and 100% success rate of reaching the goal.