## Basic Driving Agent

The first thing we did was to implement a basic driving agent. This agent selects a random move from `[None, 'forward', 'right', 'left']` at each iteration, regardless of the input variables, and deadline. We observed that the agent would occasionally reach the destination, although typically well after the deadline had passed. Sometimes the hard deadline of -100 would prevent the agent from proceeding.

## Identifying States

There are several states which we deem important in deciding on an action.

- 'next_waypoint' is important, as it is our desired direction.
- 'light' is important, as the agent is penalized when going 'forward' or 'left' on a red light.
- 'oncoming' is important as the agent is penalized when going left with oncoming traffic.
- 'left' is important as we can only turn right if there is no traffic coming from the left, or oncoming traffic turning left.

States which are not important

- 'duration': we don't care about the duration, as when the duration ends a new trip will start, but the agent is not penalized for this. Also, incorperating duration would make an impractical amount of states, requiring a huge amount of trails in order for the agent to learn.
- 'right': we don't care about any cars to the right, as they do not affect our ability to go 'forward', 'right', or 'left'. (If the agent is at a red light, it can't forward or left regardless of the agent to the right.)

## Q-Learning

At first the agent meanders around randomly, but after that it slowly learns how to drive! Due to this initial meandering, many of the rewards we see are negative. In order to encourage the agent to seek out the best action, we choose a random move with probability (epsilon / trail number). This prevents the agent from becoming stuck in local maxima. Eventually, as the agent tries all combination of moves this random restarting is no longer necessary. The default initial value of `0.3` was chosen for epsilon, as this gives a high enough probability to explore the state space initially, but tapers off as the trail number increases.

## Enhanced Driving

The initial Q-learning implementation worked, but applying a learning rate of `0.56`, and a discount factor of `0.44` caused the learning agent to prioritize new rewards, and therefore learn slightly faster. This was determined experimentally in a "grid-search" style, and choosing the agent which recieves the highest average reward over the last 10 runs. Within 100 trails, the agent finishes nearly every task on time, and with a positive reward.

```
Best Rates:
    discount  mean_reward      rate  score
45  0.555556     2.412980  0.444444  100.0
40  0.000000     2.390202  0.444444  100.0
13  0.333333     2.379997  0.111111  100.0
63  0.333333     2.342778  0.666667  100.0
64  0.444444     2.339992  0.666667  100.0
```

Something interesting to note is that the agent initially learned to turn right at red lights, instead of waiting. This is due to a reward of `0.0` being given for waiting at a red light. Since waiting at a red light can sometimes be the correct action (for instance, if the next_waypoint is 'forward', or 'left'), the agent should possibly be given a positive reward in these cases. As an alternative approach, I set the default Q value for each Transition to 1, instead of 0. This resulted eliminated some of the tendency to get stuck in a local maximum at a red light.

## Exploring the State Space

To see whether we have learned an optimal policy, we shed some light on the degree to which the state space is explored by the smart cab.

- next_waypoint: ['left', 'forward', 'right']
- light: ['green', 'red']
- oncoming: ['None', 'left', 'forward', 'right']
- left: ['None', 'left', 'forward', 'right']

This makes 3 **2** 4 * 4 = 96 states.

### Percent success over the last 10 trails of each run.

```
Scores: [100.0, 100.0, 100.0, 100.0, 100.0, 90.0, 100.0, 100.0, 100.0, 100.0, 100.0
, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.
0, 90.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.
0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100
.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 10
0.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 1
00.0, 100.0, 100.0, 90.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 1
00.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0,
100.0, 100.0, 100.0, 100.0, 100.0, 100.0]
Mean Score: 99.7
Mean States visited: 23.82
```

We can see that over 100 runs, there are about 3 runs where the agent does not reach the

destination 10/10 times in its last 10 runs.

On average (over 100 runs), the agent will visit ~24 states. This is only ~25% of all possible states. However, some of the states are unlikly to be reached. Even after a run with 10000 trails, the agent only explores ~53 states. This is possibly because with only 3 other agents in the game, combinations of oncoming and left traffic (with all permutations of over next_waypoint and light) are rare. These total 54 states (3 2 3 * 3, where both oncoming and leftward traffic are not None.), which the agent will not encounter very frequently. We tested this hypothesis by increasing the number of "dummy" agents, which resulted in the agent learning an average of 36.3 states with 10 agents, and 47.6 states with 20 dummy agents, over 100 runs, instead of 24 states with the default 3 dummy agents. Therefore we conclude that with more training, and by training under a variety of conditions we can sufficiently explore the feature space. This illustrates part of the labor of reinforcement learning, if we have no way merging states. Each situation is treated different, requiring an extraordiny amount of training as the amount of features increases. Furthermore, if a rare situation is not encountered during training, it could be encountered in production with unfavorable consequences.

## Is our policy optimal?

Given that 54 states will be encountered with relatively low probability (with 3 dummy agents), the agent learns 24 / 42 of the most frequently occuring states, which is about 57%. This is not close to an optimal policy, but from our average rewards and success rate, we see that we do not need to learn the optimal policy to reach our destination consitently, and with a net positive reward.