## Basic Driving Agent

The basic driving agent picks a random action from all possible actions available. It does not seem to reach the goal in a consistent manner. In 100 trials the basic driving agent was able to reach destination 17% of the time and received negative reward 26.5% of the time

## Choice of States

I chose my state as a tuple with information about light, and next_waypoint. I chose these states because state needs to contain all the information necessary to predict the effects of an action and to determine if it is a goal state. The reward of smart car is a function of light and next_waypoint. So having these two attributes in state helps the smart car decide on the action that would maximize reward. Other attributes like oncoming, left, right and deadline are not considered in calculating reward for an action, so they don't need to be included in state. Increasing state space will lead to curse of dimensionality problem. Having more attributes in state means the smart car will have more number of states corresponding to different combinations of each attribute in state and hence will require more trials to learn reward of each combination of the state attributes. Also adding an attribute to state without considering it in reward function will make it more difficult for the smart car to relate its action to reward.

## Q-Learning

Using q-learning the smart car is able to consistently reach goal and reduce the time to reach goal with each trial, whereas the basic agent was randomly able to reach goal sometimes and sometimes was taking a lot of time without reaching goal. Using q-learning the smart car is obeying traffic law, but the basic smart car does not follow the traffic law. Using q-learning smart car is observing the current state with light red or green and next_waypoint and then taking the action that maximizes reward. With every trial the smart car seems to be putting more value to action that obey the traffic law like going forward, left or right on green and putting less value to actions against the traffic law like left on red. In fact, actions that violate traffic law have negative reward.

Using q-learning with alpha 0.8, Gamma 0.2 and epsilon 0.1, in 100 trials the smart cab is able to reach destination 64% of time and received negative reward 14.74% of the time

After implementing decay, the smart cab is able to reach destination 79% of the time and received negative reward 13.8% of the time

It seems implementing decay, the smart cab is able to remember better from previous trials.

The table below shows percentage of time the smart car reached goal and percentage of time it received negative reward for 100 trials using different values of alpha, gamma and epsilon with decay

| Alpha | Gamma | Epsilon | %reached destination | %received negative reward |
|---|---|---|---|---|
| 0.8 | 0.2 | 0.1 | 79 | 13.8 |
| 0.9 | 0.2 | 0.1 | 70 | 13.75 |
| 0.8 | 0.1 | 0.1 | 67 | 14.38 |
| 0.8 | 0.3 | 0.1 | 71 | 13.8 |
| 0.8 | 0.2 | 0.0 | 64 | 14.18 |

## Improving Q-Learning

Smart cab using q-learning is able to reach the goal significantly more frequently than the basic driving agent. However, the smart cab is still not able to reach goal 100% of the time and it still violates traffic laws. In the last 10 trials while running for 100 trials, it was going forward and left on red light. Another problem with the smart cab is, it does not take the shortest possible path to reach the goal.

Some improvements to the policy can be made to improve performance. For example, traffic laws were removed from reward function for simplicity. Adding traffic laws to reward function would help the smart cab learn traffic laws over few trials. Also the rewards were not weighted according to distance from the goal. Putting higher weight on the action that leads to shortest path to reach goal, would help the smart cab learn to drive in the direction of shortest path.

Below is a print of q values after 100 trials of the smart car with decay

{(None, None): 0.45552395554538194,

(State(light='green', next_waypoint='forward'), None): 1.456072893031634,

(State(light='green', next_waypoint='forward'), 'forward'): 2.4065142799701533,

(State(light='green', next_waypoint='forward'), 'left'): 0.9832341881646278,

(State(light='green', next_waypoint='forward'), 'right'): 0.9157339396332008,

(State(light='green', next_waypoint='left'), None): 1.4368720782749913,

(State(light='green', next_waypoint='left'), 'forward'): 0.8792546346216589,

(State(light='green', next_waypoint='left'), 'left'): 2.347548842664804,

(State(light='green', next_waypoint='left'), 'right'): 0.9697295192160741,

(State(light='green', next_waypoint='right'), None): 1.4345318196927233,

(State(light='green', next_waypoint='right'), 'forward'): 0.9677382978172998,

(State(light='green', next_waypoint='right'), 'left'): 0.9502586050742365,

(State(light='green', next_waypoint='right'), 'right'): 2.4412577343027193,

(State(light='red', next_waypoint='forward'), None): 1.340113547491348,

(State(light='red', next_waypoint='forward'), 'forward'): -0.6491928110179035,

(State(light='red', next_waypoint='forward'), 'left'): -0.638927349504578,

(State(light='red', next_waypoint='forward'), 'right'): 0.9359763870235336,

(State(light='red', next_waypoint='left'), None): 1.3657216451373306,

(State(light='red', next_waypoint='left'), 'forward'): -0.6838133588731777,

(State(light='red', next_waypoint='left'), 'left'): -0.6728088218975086,

(State(light='red', next_waypoint='left'), 'right'): 0.8313890022940555,

(State(light='red', next_waypoint='right'), None): 1.4809654693426537,

(State(light='red', next_waypoint='right'), 'forward'): -0.5094778868790683,

(State(light='red', next_waypoint='right'), 'left'): -0.5123296497639546,

(State(light='red', next_waypoint='right'), 'right'): 2.4071973640663984}

Looking at the q values, one improvement I would make is reduce reward for states with action None. That way the smart car will stop less and reach destination faster.