Running the agent with random choices

The agent rarely got to its destination, yet it incurred of negative rewards along the way.

Choosing the state of the agent

The variables I chose to represent the state of the smart cab are: incoming, lights, left, right and next_waypoint.
The next_waypoint variable is the only input variable which can provide the agent with information of the position of the destination. As perhaps the most important task of the agent is to reach the destination, and knowing where it is is essential for solving this task, next_waypoint had to be included in the state.

The lights tell us the state of the traffic light which is important that we add to state to ensure that our agent obeys the rule of the traffic light.

The agent needs to know the state of vehicles coming in on the left and on the right to help decide when to give way to traffic.

How Q-Learning was implemented

To understand how Q Learning was implemented in this exercise, we need to explain some concepts

The lookup table(Q-Table).

We used a dictionary whose keys are the states and whose value is another dictionary. This dictionary contains actions and the long term reward Q(s, a) when taking a given you are in state s.
A dictionary was used to model the Q table because it allows lookup to be done in constant time.
A dictionary was used to store the long term results of taking actions because we could easily compute the best action using python's operator keyword. We can also get the maximum Q by taking a max of the long term reward values.

Getting Best action

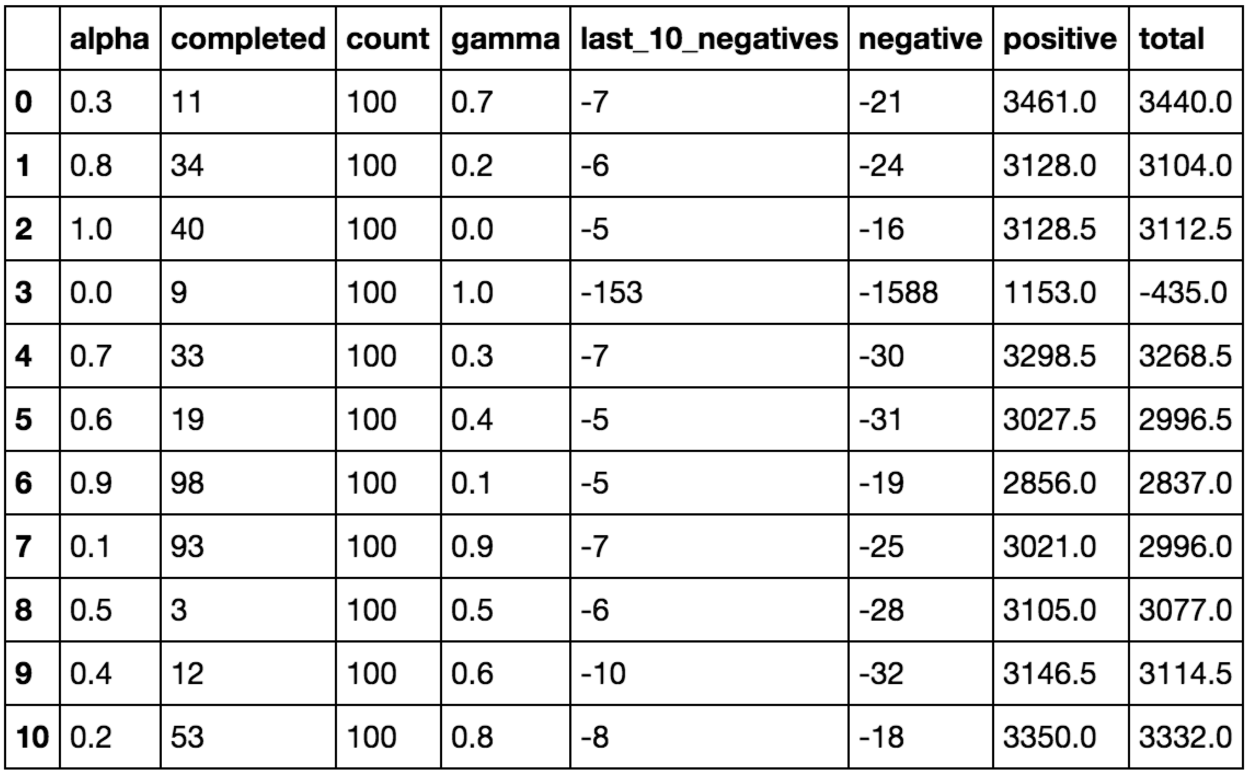
As explained earlier, to choose the best action that can be taken given that you are in a particular state, you get the action dictionary key with the maximum value for Q[key], where key is the state.
Once a state has not been explored, we give a default long term reward of 1 to all actions giving them equal probability to be chosen

Improvement gotten Q Learning.

The first approach before QLearning used random selection of actions. In this mode, the agent rarely got to its destination, let alone reach its destination in time.
With QLearning however, at the beginning, the agent incurred some negative scores, but it quickly improved this number. The agent learnt to reach its destination with net positive rewards and no negative rewards.

Tuning QLearning parameters.

Two parameters that I tuned to get better results was alpha and gamma. Alpha represents the learning rate, while gamma represents the discount factor. From the table below, we can see that the cab has lower negative rewards and higher rates of reaching its destination in time when alpha is 0.9 and gamma is 0.1. With that setting, we recorded 98% completion with 5 negative rewards in its last 10 steps



	alpha	completed	count	gamma	last_10_negatives	negative	positive	total
0	0.3	11	100	0.7	-7	-21	3461.0	3440.0
1	0.8	34	100	0.2	-6	-24	3128.0	3104.0
2	1.0	40	100	0.0	-5	-16	3128.5	3112.5
3	0.0	9	100	1.0	-153	-1588	1153.0	-435.0
4	0.7	33	100	0.3	-7	-30	3298.5	3268.5
5	0.6	19	100	0.4	-5	-31	3027.5	2996.5
6	0.9	98	100	0.1	-5	-19	2856.0	2837.0
7	0.1	93	100	0.9	-7	-25	3021.0	2996.0
8	0.5	3	100	0.5	-6	-28	3105.0	3077.0
9	0.4	12	100	0.6	-10	-32	3146.5	3114.5
10	0.2	53	100	0.8	-8	-18	3350.0	3332.0

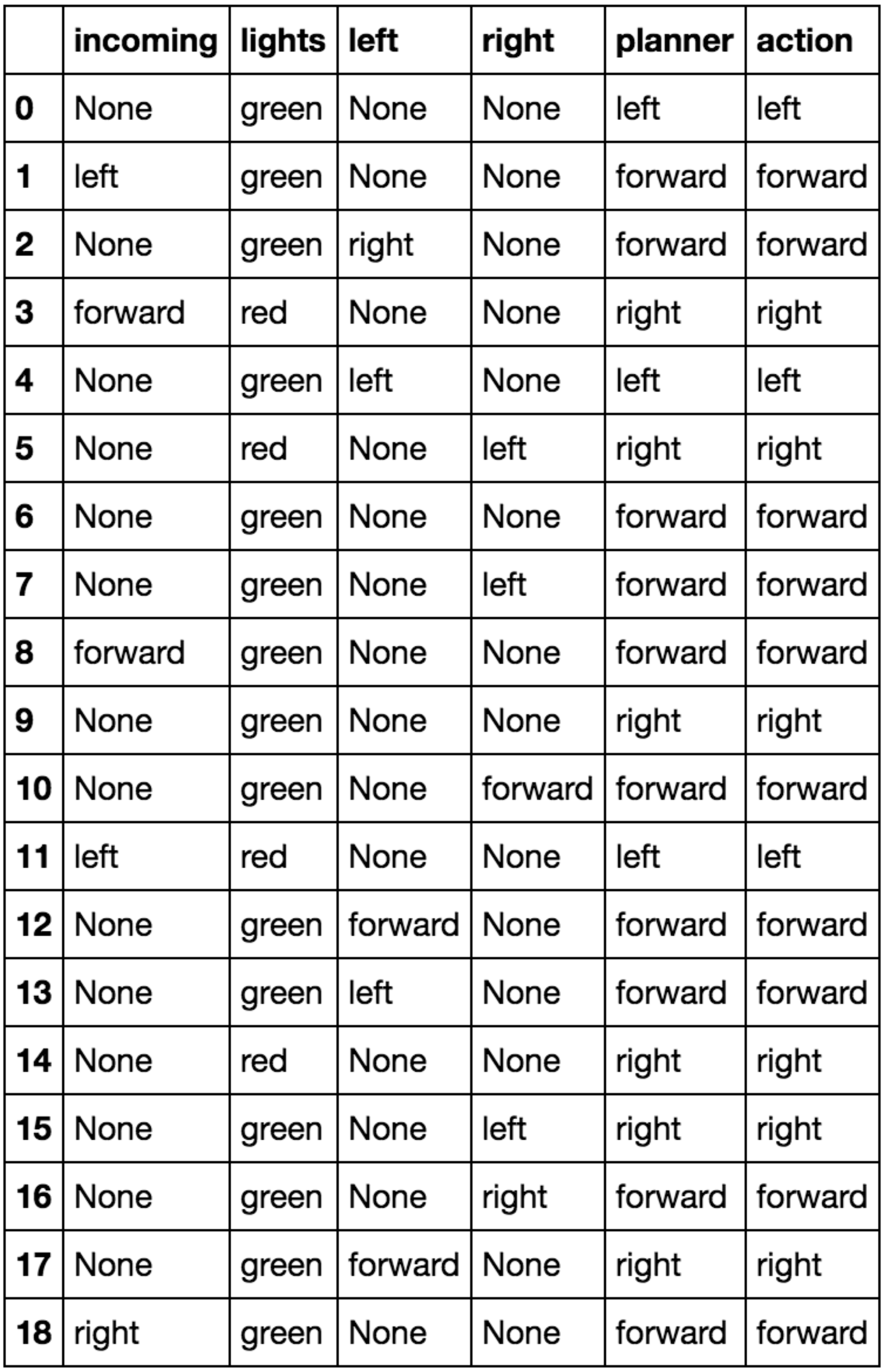
Discussing an Ideal Policy

The ideal policy for the model will have the following characteristics

- The agent accumulates as much rewards as possible while still reaching its destination in time
- The agent accumulates as little negative rewards as possible – we would want the car to be safe
- The agent obeys traffic rules so it doesn't rake up fines for its owner

- An ideal agent should always follow the planner most of the time

We noticed that the agent however, did not follow the planer's suggested directions all the time. In some cases, it did and at other cases, it does not. These two table show instances where the agent both follows the planers suggestions where it doesn't



	incoming	lights	left	right	planner	action
0	None	green	None	None	left	left
1	left	green	None	None	forward	forward
2	None	green	right	None	forward	forward
3	forward	red	None	None	right	right
4	None	green	left	None	left	left
5	None	red	None	left	right	right
6	None	green	None	None	forward	forward
7	None	green	None	left	forward	forward
8	forward	green	None	None	forward	forward
9	None	green	None	None	right	right
10	None	green	None	forward	forward	forward
11	left	red	None	None	left	left
12	None	green	forward	None	forward	forward
13	None	green	left	None	forward	forward
14	None	red	None	None	right	right
15	None	green	None	left	right	right
16	None	green	None	right	forward	forward
17	None	green	forward	None	right	right
18	right	green	None	None	forward	forward

1	None	green	None	left	left	right
2	None	green	None	forward	left	right
3	None	red	None	None	None	forward
4	left	red	None	None	right	forward
5	left	green	None	None	right	forward
6	None	red	None	forward	left	forward
7	None	red	None	None	left	None
8	None	green	forward	None	left	right
9	None	red	right	None	forward	None
10	None	red	forward	None	forward	None
11	forward	red	None	left	right	forward
12	right	red	None	None	forward	right
13	None	red	None	right	right	forward
14	None	red	None	forward	forward	right
15	left	red	None	None	None	forward
16	None	red	left	None	left	forward
17	left	red	None	None	forward	None
18	left	green	None	None	None	forward
19	None	green	None	left	None	forward
20	None	green	None	None	None	forward
21	forward	green	None	None	left	forward
22	None	red	None	right	forward	right
23	None	red	left	None	forward	right
24	None	red	None	None	forward	None
25	None	red	None	left	forward	right