Implement a Basic Driving Agent

If the actions of the smart cab such as forward, left and right are chosen randomly, a lot of negative reward is given to the smartcab, due the violations against the traffic rules. Sometimes the car does reach the destination with a lot of luck. In most cases the cab will not reach the desired destination in time.
The car has a poor performance if the actions are randomly.

Inform the Driving Agent

If we want to achieve some artificial intelligence for the cab, we need to identify some appropriate states for modelling the cab. There should be some information about the world the states. My approach was to choose an unique permutations of the input values. The inputs which are given by the world to define the states is information about the next destination, the traffic lights as well as the traffic. I defined my states by the following structure in dictionaries:

- Information about the traffic lights which is stored in the variable: *inputs['light']*
- Information about the oncoming traffic which is stored in the variable: *inputs['oncoming']*
- Information about the traffic, which is coming from right: *['right']*
- Information about the traffic, which is coming from left *['left']*
- Information about the next waypoint.

There are two reasons why this actually works:

1. This reflects the same inputs of human drivers.
2. The only information we did not include is the time. This is information is not necessary for this project, because if we would include it, the q learning algorithm would perform really bad. Also it would not be necessary, because it does not reflect the traffic rules.

The following shows the math for calculating the states of my approach:

inputs['light'] = 2 possible values (true/false)
inputs['oncoming'] = 2 possible values (true/false)
inputs['right'] = 2 possible values (true/false)
inputs['left'] = 2 possible values (true/false)
next_waypoint = 3 possible values (forward/right/left)

total number of states = 2 * 2 * 2 * 2 * 3 = 48

Implement a Q-Learning Driving Agent

For the implementation of the q learning driving agent, I used the concept of q-tables. The q-tables are storing the q value of a certain state. This helps for lookup already calculated q values.
The implementation tries to look up the q learning value of the current state and action. If there is no entry in the dictionary, then the algorithm will insert a new combination of the current state and initialises the actions *forward*, *left* and *right* with 1. If the state does already exist, then it will return the action with the highest q learning.
In the next step, we set the next state with the next waypoint and look up the maximum q value for calculating the current q value of the state and action. Then we will get the reward and calculate the q value of the current state and action.

This implementation does a better job than choosing the actions randomly. In the most cases there is no negative reward given. Due this can happen sometimes. Especially when the cab is on the wrong track and the q values of the old destination are still present. The cab does a good job. It drives to the destination very fast. Most of the time we get positives rewards.

This behaviour is occurring because of updating the q table. Often the random actions got a negative reward due to violating the traffic rules. Now the smartcab learns the traffic rules by receiving the feedback.

Improve the Q-Learning Driving Agent

alpha	gamma	epsilon	Reached	Negative	Positive	Total
0.1	0.1	0.2	100	-616	2184	2800

alpha	gamma	epsilon	Reached	Count last 10	Negative	Negative %	Positive	Total
0,3	0,1	0,3	90	9	-406,5	14,6	2369	2775,5
0,2	0,1	0,3	81	8	-489	17,2	2439	2838
0,4	0,1	0,3	91	10	-429	14,5	2522,5	2951,5
0,5	0,1	0,2	32	3	-862,5	31,8	1843	2705,5
0,4	0,1	0,4	96	10	-467,5	15,2	2614	3081,5
0,4	0,2	0,3	91	10	-503	16,1	2605	3108
0,4	**0,2**	**0,3**	**91**	**10**	**-406**	**14,2**	**2452,5**	**2858,5**
0,6	0,1	0,3	92	10	-473,5	15,5	2577	3050,5

I chose 5 criteria to rate the performance of driving agent.
- Reached: The total number of trials, which the agent completed
- Count last 10: The total number of trials, which the agent completed in the 10 final iteration.

- Negative: Total number of received negative rewards
- Positive: Total number of received positive rewards
- Total: Absolute number of negative added to the total number of positive rewards.

The agent performs best, when picking alpha 0,4, gamma 0,2 and epsilon 0,3. The final agent does reach 91 out of 100 trials, which is a very satisfying result. and completes the final 10 steps successfully. The Agent did a good job at learning. Another point I chose this agent is because of his low percentage of receiving negative rewards. This is very good result.

To sum up, with this parameters the agent really performs in a good behaviour. He receives a less negative rewards than the other agents and reach 91 out of 100 trials. In a real life situation, we would chose this agent, because of his low amount of negative rewards. This is the safest agent for the traffic.

The optimal policy is when the agent follows the next waypoint and receives a low/none amount of negative rewards. Instead of picking an action based on its value, the driving agent could follow the next waypoint. However, if the agent would violated against the traffic rules, this would not be a good approach. A combination between choosing to the action for next waypoint and do no action to respect the traffic rules would describe the optimal policy. In this case, the agent would not receive a negative reward because of no violation against the traffic rules. He would only receive a negative reward, if there is too much traffic and waits all the time. The list on the next page shows the difference between the waypoint action and the action of the agent from the last 10 trials. In some cases, the agent could have done a better job. However, there are also cases where the agent did a better job because of a better respect to traffic rules.

Trial	Light	Oncoming	Right	Left	Waypoint	Action
90	red	None	None	None	forward	None
90	green	left	None	None	left	right
90	red	None	None	None	left	forward
90	red	None	None	None	left	right
91	red	None	None	None	forward	right
91	red	None	None	None	forward	left
91	red	None	None	None	forward	right
91	red	None	None	None	left	None
91	red	None	None	None	forward	right
91	red	None	None	None	left	right
91	red	None	None	forward	forward	right
91	red	None	None	None	forward	right
91	red	None	None	None	left	forward
91	red	None	None	None	left	right
91	red	None	None	None	forward	left
91	red	None	None	None	forward	right
91	red	None	None	None	left	right
91	red	None	None	None	forward	None
91	red	None	None	None	forward	left
91	red	None	None	None	forward	None
92	red	None	None	None	forward	None
92	red	None	None	None	forward	left
92	red	None	None	None	forward	None
92	red	None	None	None	forward	right
92	red	None	None	None	left	None
93	red	None	None	None	forward	None
93	red	None	None	None	forward	None
93	red	None	None	None	forward	left
93	red	None	None	None	forward	right
93	red	None	None	None	forward	right
93	red	None	None	None	left	right
93	red	None	None	None	left	forward
93	red	None	None	None	left	None
93	red	None	None	None	left	forward
93	red	None	None	None	left	forward
94	red	None	None	None	forward	left
94	red	None	None	None	forward	right
94	red	right	None	None	right	left
95	red	None	None	None	left	right
96	red	None	None	None	forward	None
96	red	None	None	None	forward	left
96	red	None	None	None	forward	right
96	red	None	None	None	forward	None
96	red	None	None	None	forward	None
96	red	None	None	None	forward	None
96	red	None	None	None	forward	right
96	red	None	None	None	left	forward
97	red	None	None	None	forward	right
97	red	None	None	None	left	right
98	red	None	None	None	forward	right
98	red	None	None	None	forward	right
98	red	None	None	None	left	None
98	red	None	None	None	forward	None
98	red	None	None	None	forward	left
98	red	None	None	None	forward	right
99	red	None	None	None	forward	left
100	green	None	None	forward	right	forward
100	red	None	None	None	forward	None
100	red	None	None	None	forward	left