Implement a basic driving agent

In your report, mention what you see in the agent’s behavior. Does it eventually make it to the target location?

It did run randomly and aimlessly and neglected all the environmental information such as traffic lights and approaching vehicles at each intersection. However, it arrived its destinations eventually after long time. The reason is that random choices will enumerate all possible states which surely includes the target location for each trip.

Identify and update state

Justify why you picked these set of states, and how they model the agent and its environment.

My choices of the set of states are composed of two parts of the information: next waypoint ‘n’ and inputs information ‘s’.

```
if next_waypoint == 'forward':
            n = 0
elif next_waypoint == 'right':
            n = 1
elif next_waypoint == 'left':
            n = 2

if inputs['light'] == 'red' and inputs['left'] == 'forward':
            s = 0
elif inputs['light'] == 'red' and inputs['left'] != 'forward':
            s = 1
elif inputs['light'] == 'green' and (inputs['oncoming'] == 'forward' or inputs['oncoming'] == 'right'):
            s = 2
else:
            s = 3
```

The state I choose can be expressed as a tuple (s,n). The reason I choose implicit states for inputs information is that only four kinds of states based on the environmental information are essential for the smart car to drive. They are:

a. Only to stay.
b. Can turn right and stay.

c. Can go forward, turn right and stay, but not to turn left.
d. Can do all four actions.

Extracting these states from the inputs information can not only reduce the state space dramatically and effectively, but also accelerate the learning rates.

For simplicity, I transform the two-dimension states into one dimension through 3*s + n, which results in 12 states totally.

Implement Q-Learning

What changes do you notice in the agent’s behavior?

I pick up the best action which leads to the maximum Q-values from the current state. However, it did not perform well. It began to choose certain kinds of actions and it failed to reach targets for most of the trials within the deadlines. Also it broke the traffic laws often.

The Q table is initialized as all zeros and updated as the car moving. Since Q value of the state updated at each time step is determined by the chosen action and the Q value increases at the most time. Once one of the Q values is the biggest among that state, the car is prone to choose that action always, and the learning progress stops to update other states.

Enhance the driving agent

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

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?

I made the following two changes to the original RL algorithm,
First, the car chooses random actions at the first 200 steps. This procedure is to let the car move freely and explorer all possible states as many times as possible. Hence the drawback previously mentioned can be avoided.

Second, I introduce another small parameter epsilon, in addition to the learning rate alpha, and the discount gamma. With the possibility epsilon, the car is able to choose accessible actions randomly instead of choosing based on the Q values. In this method, the RL algorithm may escape some local minimums.

To find the optimal values for alpha and gamma, I wrote the assisting loops to search the alpha and gamma in the range of [0.5, 0.9]. The metric is its performances during the 10 testing trips right after the 100 training trips. Only if the car does not incur any penalties and reaches the target every time, I record the total time step it takes during the 10 testing trips. In the method, I found the optimal values are alpha = 0.6, gamma = 0.6. The Q table is recorded as follows for your reference:

```
Minimum TimeStep = 118, Alpha = 0.6, Gamma = 0.6
[[ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 6.41694211  3.18363005  3.12112261  5.50991064]
 [ 8.16433994  4.16887137  4.15527184  7.44322692]
 [ 7.07512766  2.63880854  2.26862031  4.56537066]
 [ 0.          7.72397094  0.          0.        ]
 [ 0.          7.30521538  0.          0.        ]
 [ 5.79208195  0.          0.          0.        ]
 [ 4.7198824   8.07585271  5.81467012  4.68427447]
 [ 8.18891429  6.36572681  4.26441669  6.00124017]
 [ 5.47040409  3.61061383  8.26139944  5.34951225]]
```