Implement a Basic Driving Agent

QUESTION: *Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?*

In this case scenario the car is moving randomly around the grid. It obeys to the rules related to the traffic lights, although it does not avoid incoming traffic and sometimes runs into the other vehicles.
I noticed that a few times it reached destination, but it was a pure case as most of the time the agent was driving around even further away from the goal.
The action taken by the car was consistent with the action reported on the upper left corner of the screen, but it was not matching with the label next to the car. The reason is that the label next to the car is reflecting the "next_waypoint" and, looking at the code in the "planner.py" file I can see that this value is set based on the current position of the car and the location of the destination. Therefore this value is not random (as instead is the action that the car takes) and it sorts of identifies which way to go for planning the the trip and reach the destination. I believe that if the action taken by the car and the "value" "suggested" by the next_waypoint are matching, then the agent is rewarded.

Inform the Driving Agent

QUESTION: *What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem?*

The states that I have considered are the following:
- traffic light
- oncoming traffic
- left traffic
- next_waypoint

I have not included the right incoming traffic because the agent is allowed to move in the opposite direction.
Another variable that is not included in the state is the deadline (deadline = self.env.get_deadline(self)). This variable could be represented in ranges or at each single timeframe. Both representations would dramatically (exponentially!) increase the size of the state. It's always good to keep in mind the "curse of dimensionality" when considering the appropriate size.

I believe these states appropriately describe the environment as they take into consideration the traffic regulation and the planned path for reaching the destination (identified by "next_waypoint").

OPTIONAL*: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?*

These are the total "states" I can count:

- traffic light (2 options: “red”, “green”)
- oncoming traffic (4 options: “none”, “left”, “right”, “forward”)
- right traffic (4 options: “none”, “left”, “right”, “forward”)
- left traffic (4 options: “none”, “left”, “right”, “forward”)
- next_waypoint (3 options: “left”, “right”, “forward”)

So, in total I would have 2*4*4*4*3 = 384 states.
Although, if I had to be consistent with the assumption that we are in a US traffic regulated environment, then the “right traffic” should not be considered, therefore the total number of states would be 96.

The q-Learning goes to convergence when it visits infinitely often each state/action value pair, so the more states we have the more iterations the algorithm has to make and the slower it reaches a final value.
The training time would be exponential to the state size. (similar to the curse of dimensionality we saw in previous topics).

Implement a Q-Learning Driving Agent

QUESTION: *What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?*

At first, the agent takes obviously a random action, since the initial value for the Qtable are set to zero and the first action is a random one.
Later on, at every stage the Qtable gets updated and the Q values increase, optimizing therefore the selection of the best action to choose at each stage.
In fact, the agent now, as the steps increase, makes more “accurate” decisions and while initially it takes random loops, then it refines its path choosing the action that takes it closer to the destination.
As the steps increase it reaches the destination more quickly.
This is due to the fact that, for choosing the best action it looks at the Q value. This value takes into consideration the utility of the current state and the utility of the “next” state. The higher the value, the higher the long term expected reward. This defines the best action to take.

The problem now is that if the agent ever happens upon a decent option, it will always choose that one in the future, even if there is a better option available. To overcome this, we need to introduce exploration. This will take us to the “exploration-exploitation” dilemma, which I’m going to address next.

Improve the Q-Learning Driving Agent

QUESTION: *Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?*

I initially set alpha=0.8 and gamma=0.9 and the accuracy rate was around 60/65%

In the next iteration I updated to:
alpha=0.8 and gamma=0.1
and the success rate slightly improved.
Finally, I lowered the value of alpha as well, ending up with the following values:
alpha=0.2 and gamma=0.1 and the accuracy rate improved a little bit to around 72%.
The improvement that was most successful though was changing the value of epsilon, therefore tweaking the exploration/exploitation "ratio".
I allowed the epsilon value to gradually decay (the GLIE method, or greedy-limit and infinite exploration) which improved the accuracy to around 94%.
Additionally, after I tweaked the alpha value that gave the best results, I have use the "alpha decayed" and reduced the learning rate over time. Thus as time elapses, the agent becomes more confident with what it's learned in addition to being less persuaded by the information.

QUESTION: *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? How would you describe an optimal policy for this problem?*

The agent does reach the destination many times, although it does incur in few penalties from time to time (see below better clarification and evaluation).
The policy for this problem does look for the best possible route that follows the "planned" path (self.planner.next_waypoint()).
It does not look for ways to reach the destination in the minimum possible time.
The "state" in fact does not include the deadline as one of the factors.

A perfect policy would be in this case the one that follows the optimal path with no penalties. The accuracy alone in this case won't help us identify if this policy is as such.

Another good measure that we could use would be to consider the number of times the agent incurs into penalties along the way.
I calculated the percentage of penalties within the total number of steps taken. On average I get a result of 15% penalties incurred. It's interesting to see how the number of penalties, correctly, decreases over time as well.

Additionally, I compared the "inputs" vs the "next_waypoint" vs the "action" taken in order to see where and when the agent was making a mistake:

I noticed that the agent was taking the wrong action, therefore incurring into a negative reward in the beginning of the iterations, when the agent was still learning.

i.e. (step # 5)

inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}
`next_waypoint: right`
`action: forward`

On the other hand, in the final steps, it was taking a different action vs the way_point route, because it was also taking into consideration the traffic rules. This shows how well the agent has learned over time.

i.e. (step # 1,507)

inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}
next_waypoint: left
action: None