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?*

I set the enforce deadline to False and my trial period is 100 times. With 100 times of trials, some trials the smart cab eventually arrive at the target location, despite the fact that all cab took different steps to arrive at the destination. For instance, in one trial it only takes 8 steps for the cab to arrive at the destination.

Under basic driving scenario, the cab simply picks one of the actions as the input and sometimes it gets lucky to arrive at the destination. Therefore, whether it receives positive or negative rewards does not affect its decision making at all, since some cabs that arrive the destination with negative rewards.

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 picked are Light, Oncoming Traffic, and the Traffic coming from the Left.

On a green light, a left turn is permitted if there is no oncoming traffic making a right turn or coming straight through the intersection. On a red light, a right turn is permitted if no oncoming traffic is approaching from the left through the intersection.

In addition, I also picked the next waypoint as one of the states since taking action or not is one of the inputs that is required in this smart cab project. It can determine the state of the traffic light for its direction of movement, and whether there is a vehicle at the intersection for each of the oncoming directions. For each action, the smart cab may either idle at the intersection, or drive to the next intersection to the left, right, or ahead of it.

In addition to selecting these states, Traffic from the Right and Deadline are two states being left out in Q-Learning factors. Since the light would control the traffic from the right, the factor should not be considered in the states as it would be conflicting with the

light. Whether the traffic from the right would obey or break the traffic rules and clash with the cab is another factor that should be included in the discussion here.

Potential flaws of adding the deadline to the states is that deadline is generated by environment, which is deterministic. Compared to other states and actions that are stochastic, adding deadline as a factor would be conflicting in finding optimal Q-Learning policy.

The smart cab now considers the output from the environment. Because in this evaluation, we have not enforced the deadline, thus all trials that successfully arrived at the destination exceeded the deadline being set at the start.

These states are appropriate to solve the problem since how much reward the cab could get through the actions is entirely based on these states after receiving inputs from the environment.

The smart cab receives a reward for each successfully completed trip (+2.0), and also receives a smaller reward for each action it executes successfully that obeys traffic rules. The smart cab receives a small penalty (-0.5) for any incorrect action, and a larger penalty for any action that violates traffic rules or causes an accident with another vehicle (-1.0).

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?*

Total number of states should be four, and the states are Lights, Next Waypoint, Oncoming Traffic, and Traffic from the Left. However, the combination of all actions should be 2 (lights: Green/ Red) x 3 (next waypoint: Forward/Left/ Right) x 3 (oncoming: Left/ Right/ None) x 3 (left: Forward/Right/None), despite some combinations won't appear due to the nature of traffic rules. Given the restriction of the environment and the route planner, I would argue this number of states does not seem reasonable for the Q-Learning agent to make informed decision. Often times, the agent does not have enough time to explore all the possible actions under the states before the designated deadline assigned by route planner. The deadline is automatically generated by the environment and the timeframe is justified based on the formula of [(vertical distance + horizontal distance) x 5]. Under this constraint, the agent does not have an extended period to explore all the rewards it might get and thus suffering from the problem not making the best informed decisions.

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?*

Compared to the basic driving agent that moves randomly and receive rewards arbitrarily, the Q-Learning agent would rather not move randomly to avoid receiving negative rewards. The downside of this is that sometimes the agent would stay at the same location exploring all the possible exhaustively without actually moving towards the final destination.

As I set the alpha/learning rate equals 0.8 and gamma/discount factor equals 0.9, the smart cab would care about negative reward, and sometimes rather taking no action, as opposed to in previous attempts without applying Q-learning agent. In the trials earlier, one could argue whether a smart cab arrives at destination is purely by luck since the cab chooses its action randomly.

The goal of the agent is to maximize its total reward. Therefore, the agent has taken all the possible combination of reward into consideration to avoid negative reward in total. However, we can find the agent would tend to do nothing to avoid penalties that would add up to negative reward, and this action directly leads to inactive agent movement, which is not optimized since the ultimate goal is to reach the destination.

In addition, after several rounds of observation, whether the smart cab successfully moves to the destination has a lot of to do with the route being assigned by the route planner. For example, if the destination is rather close to where the smart cab initiates, and if the route is rather simple and straight forward, chances are that the cab might arrive at the destination within the deadline.

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 chose the different parameters shown as below. Among all sets of different combination, I found the set of Alpha=0.5 with Gamma=0.75 is the agent with best performance. The agent has a successful rate of 76% in 100 simulations and is marginally better than the other tuning parameters tested.

		Alpha (reducing time to learn, can achieve goal within time frame)		
		0.5	0.75	1
Gamma	0.5	33/100	34/100	65/100
	0.75	76/100	68/100	28/100
	1	35/100	42/100	30/100

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?*

After selecting the value of Alpha and Gamma, the successful rate is around 6~7 times per 10 trials. Thus I set the initial Q-values of forward, left, and right to 4, and by increasing the initial Q-values, the net reward would be amplified and therefore explicitly prompt the agent to explore more states and learn more in the ending. By tuning this parameter, the success rate of the agent is now increased to 7~8 times per 10 trials. Moreover, it increased the frequency of agents getting reward of 2.0 and -0.5 as well as reducing the number of steps required getting to the destination.

Alpha, the learning rate, determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent learn nothing at all, while a factor of 1 would make the agent consider the most recent information only. In essence, it is a tradeoff between learning from historical data and making decision based on new entries.

The Gamma is the discount factor used to progressively reduce the value of future rewards. If a Gamma parameter is closer to 0, the agent will consider only immediate rewards, while a Gamma parameter is closer to 1, the agent will consider future rewards with greater weight and delay the reward.

Other than the best policy I found in the above questions, I also tried several sets of different of tuning. For instance, I have also tested when Alpha=0.5 and Gamma = 0.9, but nonetheless its successful rate is not as high the one Alpha=0.5 and Gamma = 0.75. Ultimately, an optimal and ideal policy would be one that reaches destination at the minimum time possible without incurring any penalties. However, several successful attempts in 100 simulations that arrive at the destination did receive penalties of -0.5 for incorrect actions. An interpretation of this is that incorrect action might seem detrimental at the moment, but in long term the agent might be able to achieve the goal faster. That is to say, the agent is looking at a bigger picture. Therefore, I would describe an optimal

policy for the problem is to arrive at the destination with the least time possible, and to allow negative reward due to incorrect actions in order to achieve the greater goal. After tuning the initial Q-value, I would argue the Q-learning agent is now acting optimally and always trying to go the shortest path possible, which also represents the least time required. An instance shown below demonstrates that the agent is willing to take minor penalties, even at the very first few steps, in order to reach the goal faster.

Examples extracted from one of the successful simulations:

Environment.reset(): Trial set up with start = (8, 6), destination = (8, 1), deadline = 25
RoutePlanner.route_to(): destination = (8, 1)
The current state is: ('red', 'forward', None, None)
t:0
LearningAgent.update(): deadline = 25, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = -0.5
The current state is: ('red', 'forward', None, None)
t:1
LearningAgent.update(): deadline = 24, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = -0.5
The current state is: ('red', 'left', None, None)
t:2
Environment.act(): Primary agent has reached destination!
LearningAgent.update(): deadline = 23, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 9.5