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?*

By choosing random action from 4 viable actions (None, left, right, forward), the agent moved around in the environment. Since agent didn't take into account any of the traffic rules, it received negative reward whenever the random action resulted in traffic violation reported by environment. Also, agent didn't follow any strategy to reach destination so random action at each intersection resulted agent moving randomly in the environment ignoring goal to reach the target.

By NOT enforcing simulation deadline agent many times hit hard time limit (100 iterations) and aborting simulator run. Sometimes agent did make to destination but way past deadline but again there is no pattern here.

It is now clear that agent needs a strategy or policy to reach the destination guiding to choose right action at each interaction. This policy will need to take into account on how to reach the destination optimally given the traffic laws (avoiding collisions) , learning reward influence to take proper action at each intersection.

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?*

***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?*

I have defined agent state variables based on following possible inputs at any intersection:

Next_waypoint = {None, forward, left, right}

Light = {green, red}

Oncoming Traffic = {None, forward, left, right}

Right Traffic = { None, forward, left, right}

Left Traffic = {None, forward, left, right}

Although we can eliminate some of the values I have choose to model states with all possible inputs in mind. In this case,we can state uniquely represent any combination of above possible values.

For example, at one intersection at any some specific time, we may have Agent heading ‘forward’ (based on router planner), with a ’red’ light, ‘none’ oncoming traffic, ‘forward’ right traffic and ‘none’ left traffic. So we can represent this state as combination of these values (Next_Waypoint + Light + Oncoming + Right + Left) : ‘ForwardRedNoneForwardNone’ or more simply ‘FRNFN’ or ’frnfn’.

With above reasoning we can have total of 512 different states (4 (waypoints) X 2(lights) X 4(oncoming) X 4(right) X 4(left) = 512) for this kind of grid of any size with given this set of input conditions. Yes, this seems like reasonable number given the state machine can be applied to any grid size. With Q-learning we wanted to learn best possible (state, action) pair for any given state and this state will capture all unique state representing grid intersections.

Also, by choosing the state based on above combinations, I don’t to worry about changed traffic rules (as the state capture fully all possible values) in simulation with given set of possible values in each traffic. This provides nice residence and extensibility at the cost increased possible state values.

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?*

The basic approach for implementation is

1. Initialize the Q-values table (Q_learn) for each possible (state, action) observed : Q(s, a). I have chosen 0. has initial value
2. Observe the current state, s.
3. Choose an action, a, for that state based on one of the action selection policies (example: Greedy-Epsilon).
4. Take the action, and observe the reward, r, as well as the new state, s'.
5. Update the Q-value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula outlined and parameters described above.
6. Set the state to the new state, and repeat outlined steps from 1 to 6.

The step5, following equation summarizes the process of Learning
Q(s, a) = Q(s,a) + alpha [r(s,a) + gamma (argmax(Q(s', a') - Q(s,a)]
Where,
alpha : learning rate: determines to what extent the newly acquired information will override the old information
gamma (discount factor): the weight agent should attribute to future reward vs current reward.
argmax(Q(s', a') : Q-values from potential future actions

In step 3, I have chose Greedy-Epsilon policy for action selection: In this policy, the action is selected greedily with respect to the Q-value estimates a fraction (1−є) of the time (where є is a fraction between 0 and 1), and randomly selected among all actions a fraction ϵ (epsilon - exploration rate) of the time. Policy that has some randomness promotes exploration of the state space.

With this implementation, i have run the agent with few scenarios with learning and policy parameters set to their extremes to understand agent behavior,

1) With with basic set of parameters set to values that with no learning or no exploration to observe the behavior(Exploration rate to 0, future discounts to 0, learning rate to 0). As suspected, agent didn't reach destination most of the time before the deadline, confirming that agent policy is choosing maximum explored action but with no chance to explore the new states (local maximum problem)

2) With exploration rate to 1, future discounts to 1, learning rate 1 (always learning but always exploring without using learning data). In this case, as expected agent didn't reach destination most of the time within deadline but did better than previous scenario. In this case, agent policy is always taking random actions without looking into learned rewards.

3) With exploration rate to 0 (which means greedy all the time), future discounts 1, learning 1 agent did poorly again. As expected, agent policy is choosing maximum of explored (state, action) pair values runs into (local maximum) problem. And didn't converge to destination.

With this observation, found the need to fine tune learning rate and policy choices (parameters: alpha, gamma, epsilon) to be able to reach destination optimally.

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?*

***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?*

Following are some of runs of parameter tuning i have tried with to understand agent learning behavior. My observation is that with best combination of parameters tuned agent was able to reach the destination consistency after about ~ 16 to 19 trails (converged fully). Here are some of the different parameters parameters i have tried guiding me to successful convergence.

epsilon 0.1, alpha 1, gamma 1 - did very poorly. only reached destination 26 trails out of 100 trails. Never fully converged.
epsilon 0.1, alpha 0.1, gamma 1 ~ didn't converge only reached destination ~60 trails out of 100 trails. Never fully converged

This tells me that gamma (discount factor - fully favoring future rewards) close 1 is producing unwanted results. So modified strategy keep tune gamma close to zero.

epsilon 0.1, alpha 0.1, gamma 0.1 ~ reached destination in all 91 trails out of 100 trails. Not fully converged (8 failures are distributed)
epsilon 0.1, alpha 1, gamma 0.1 ~ reached destination in all 91 trails with 9 failures. Not fully converged
epsilon 0.1, alpha 1, gamma 0 ~ reached destination in all 90 trails with 10 failures. Not fully converged

From the above, concluded that alpha doesn't seem to factor much in our scenario. Since our world is deterministic, we can keep learning rate constant 1 optimal. however, it can be at constant 0.1 also. However, above results are much encouraging.

epsilon 0.2, alpha 1, gamma 0.01 ~ Agent reached destination in 84

trails out of 100 trails
epsilon 0.05, alpha 1, gamma 0.1 ~   Agent reached destination in about ~80 trails out of 100 trails

Epsilon set 0.1 yielded very good results.

Given the above results, agent did perform well for exploration rate set to 0.1 (90% greedy selection among all possible actions, 10% of the time random action choice), alpha set close to 1 (always learning) , gamma set close to zero agent reached destination in about ~15 trails and with success rate close to 90%. During learning agent did incur penalties (reward = negative values) sometimes but able to reach the destination on time (with in deadline).

Given the grid model, and router planner strategy, the best policy would be follow the planner guidance (way_point) as 'action' at each intersection with 'None' action if there is chance of collision or violating traffic rules. This would avoid any penalties and reach destination optimally.

To avoid penalties and converge quickly, I have also explore by setting initial Q_value to be high. But I didn't notice any noticeable change.

intialQValue = 12, epsilon 0.1, alpha 1, gamma 0.1 ~ reached destination in about 94 trails with 6 failures. failures are intermittent. Did n't converge fully