Project: *Train a Smartcab to Drive*
Name : *Sri Raghu Malireddi*

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

- The target is to **IMPLEMENT A BASIC DRIVING AGENT.**
- The agent makes a random move/action. The python code for the same is as follows
  - `action = random.choice([None, 'forward', 'left', 'right'])`
- The agent can take either finite time or infinite time to make it to the target location (because of random choice, and no learning is involved).
- The number of steps required for the agent to reach the target, increase exponentially as the distance between the start point and destination increase.

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

- The target is to **IDENTIFY AND UPDATE STATE.**
- The states must be chosen in an effective way, and the number of states should be optimal.
- I used the environment inputs 'left', 'oncoming' and 'light', and a 'recommended State'(next_waypoint) as they contribute to take any decision related to the action of the Smartcab. I am explaining why I chose those parameters:
  - Recommended State: Within the simulation, the agent should head towards its destination. The 'recommended state'(next_waypoint) parameter will help the agent to keep aware of its present position relative to the destination. In a real world scenario, the geographical coordinates of the agent through GPS will replace this parameter.
    - recommendingStates = ["forward", "left", "right"]
  - Light: In the real life, traffic lights play a very important role in controlling the flow of traffic. The awareness of traffic signal will be very essential to prevent any accidents (*ex*: preventing the agent/car from advancing in the case of a red light.)
    - lightStates = ["red", "green"]
  - left, oncoming: These parameters describe the behavior of other agents in the simulation. The traffic that is oncoming ('oncoming'), and the traffic that is heading from left ('left'). Each of these parameters ('left', 'oncoming') will have four values [None, "forward", "left", "right"] which will describe the heading direction of those agents. If no traffic is approaching from 'left' or 'oncoming', the value will be listed as None.
- I didn't use the environment input 'right' and the trial 'deadline' count for reaching the destination and the reasons for the same are stated below.
  - right: This parameter tells us the heading direction of the traffic approaching from right. It will have the same

four values as that of ‘left’ and ‘oncoming’ parameters. The information provided by this parameter is completely unnecessary with the traffic light color. If we assume that the oncoming agent obeys the traffic laws, then there is no situation that can be changed/altered by the presence of another agent approaching from the right. For instance, if my agent’s light is ‘red’, the only possible movement is ‘right’ turn and it will not get affected from the agent approaching from ‘right’. If my agent’s light is ‘green’, the approaching agent will have a ‘red’ light and must give right of way if turning ‘right’. Hence, this parameter will not provide any information but will unnecessarily increase (in an exponential fashion) the number of learning steps for Q-learning.

    - deadline: This parameter gives information regarding how many units of time are left for the agent to reach the destination. Including this parameter makes no sense because, in our state model the agent’s behavior of approaching destination is more influenced by ‘recommended state’ than ‘deadline’. If the parameter ‘deadline’ is included, the agent’s policy of reaching destination might change during the late stages of trial when compared to early stages (ex: The agent might learn it is ok to cross a red light to reach destination within deadline, which is an undesirable action and will end up in penalty in real life.). Also, including the ‘deadline’ will increase the number of training examples drastically for Q-Learning.

- self.state = (self.next_waypoint, inputs['light'], inputs['oncoming'], inputs['left'])

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

- The target is to **IMPLEMENT Q-LEARNING.**
- Initially, the agent starts randomly.
- The Q-Table has random initialization. The learning rate, exploration rate, and discount value have been initialized and set to values within their ranges.
- The Q-Table is updated whenever the agent gets a reward.
- After 20-30 trials, the agent starts taking specific (no more random) actions in particular states and reaches state within the deadline.
- This signifies that the agent is learning! (How quick the learning happens depends on how we set the learning rate, discount value and exploration rate).
- With a learning rate of 0.75, discount value of 0.4 and exploration rate of e**-((3+self.count)/5.0), the agent is able to learn quickly. The exploration rate has been set in such a way that it is a decreasing function and its value decreases for every successful trial (i.e., the agent reaches destination before deadline and with a positive reward).
- I noticed that as the agent gains more experience, the agent is following traffic rules more carefully. We can observe this by seeing the reduction in the number of negative rewards with

every successive trial. So the policy is avoiding traffic violations!

- I also observed that with every successive trial, the ratio of number of steps taken by agent to reach destination to deadline given to the agent at the start of the trial decreased. Through this, we can conclude that the agent is learning to reach destination faster!
- The agent learnt the traffic rules first as we can see that the negative rewards were quickly lowered.

Q) 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?

- The target is to **ENHANCE THE DRIVING AGENT.**
- Exploration rate: To ensure that the agent is not under time pressure, the exploration rate is fixed in such a way that agent covers more states in the Q-table. We have to use a decreasing function for the exploration rate.
    - I first started with a logarithmic function that is dependent on the deadline, log(deadline+0.0001)*0.0015. Though it is a decreasing function and decreases with every successive step in a trial, when the agent reaches destination the deadline starts to countdown again. So, it is not an appropriate way.
    - Now, I moved from logarithmic function to exponential function, e**-((3+self.count)/5.0), where I am using number of successful runs (self.count) as a variable. Though the agent was performing well in both the above cases, this second case is the more appropriate approach. The values 3 and 5.0 in the above function are set by tweaking (by trial and error) and by observing the behavior of exponential function's range and average rewards per action of the agent.
- Q-values: I tried initializing the Q-values with zeros. Though the agent is reaching destination, it is taking unnecessary steps/continuous cycles/sometimes stuck while reaching destination. So, I initialized the Q-values with random numbers. Now, the agent is taking specific actions in particular states after about 20-30 trials. The agent not only learnt traffic rules, but also learnt to reach destination faster (without unnecessary steps/continuous cycles).
- Q-Table size: I reduced the size of Q-table by using a simple logic (the right-of-way of the traffic lights). This resulted in a removal of a number of states from Q-table.
- Discount and Learning rates: The discount rate is set to 0.4 and the learning rate is set to 0.75. The values are set by trial and error method and by observing agent's behavior in terms of average rewards per action and average number of successful trails. The following table gives some insight about how the agent reaches destination w.r.t different values of Discount value and Learning rate.

Discount Value	Learning Rate	Dest. Reached/#Trials
0.10	0.10	91/100
0.25	0.25	90/100
0.40	0.25	92/100
0.60	0.25	88/100
0.80	0.25	87/100
0.25	0.50	95/100
0.25	0.75	97/100
0.40	0.75	98/100

- Also with lower learning rates, the agent took too much time to understand the traffic rules. I understood this by checking the average rewards w.r.t number of trials. For a learning rate of 0.1, it took around 30 trials to reach to an average reward of 1.2, but with learning rate of 0.4 it took 15 trials to reach average reward of 1.2 and with learning rate of 0.75 it took just 5-6 trials to reach the same level (1.2). Here, I tried to explain how quickly the agent is learning the optimum policy with the variation in learning rate.

Q) 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?

- The target is to **ENHANCE THE DRIVING AGENT.**
- Yes, my agent got close to finding an optimal policy.
- The optimal policy for the agent would be as follows:
  - Obey traffic rules
  - Move in the direction of recommended state
  - If the traffic rules occlude the recommended state, the agent should not move.
- My algorithm generally took 15-20 trials to converge to the optimal policy.
- The Q-Table that is generated after training the agent for 100 trials was used to the test the trained agent. The trained agent was put under a 100 trial test to check the effectiveness of the Q-Table. The results are shown in the table below.

Discount Value	Learning Rate	Dest. Reached/ #Trials	Actions (Taken/ Available)	Penalties (Zero rewards)
0.25	0.25	100/100	1052/2986	0
0.50	0.25	100/100	1350/2008	0
0.25	0.50	100/100	1120/2610	0
0.40	0.60	100/100	1378/3160	0
0.40	0.75	100/100	1252/3076	0

- From the above cases, we can observe the behavior of the trained agent with different learning and discount rates. The optimal policy is the one with the discount value of 0.40 and the learning rate of 0.75, in which the trained agent took only 1252 actions out of the 3076 allowed actions (which leaves a 59.3% breathing space!). And the trained agent has zero penalties! (which means no violation of traffic rules).

- Some observations from the Q-Table with Discount rate = 0.4 and Learning rate = 0.75.
- My Q-Table keys are in the following format – (recommendedState, lightState, oncomingState, leftState, actionState)

```
>>> QTable[('right','red', None, None, 'right')]
3.3333400222586773
>>> QTable[('left','red', None, None, 'left')]
-0.9672618131279759
```

The above lines explain a lot, when there is ‘red’ light and if my agent wants to take a ‘right’ direction, the higher Q-value suggests that it can take the right direction, but my agent can’t take a left direction.

```
>>> QTable[('right','green', None, None, 'left')]
0.7918757202597836
>>> QTable[('right','green', None, None, 'right')]
3.3333462549199515
>>> QTable[('right','green', None, None, 'forward')]
0.7962913594374947
```

When there is no traffic, and if there is a green light the agent has to follow the recommended state. The higher Q-value for the action state of ‘right’ and lower values for other states signifies that our Q-Table is trained properly!

Some other examples which doesn’t need an explanation but can be well understood by their Q-values!

```
>>> QTable[('right','green', None, 'forward', 'right')]
3.2903060400128243
>>> QTable[('left', 'green', None, None, 'right')]
-0.324563160082
```

This following line shows the key that has the highest Q-value in the whole Q-table which looks quite obvious by observing the key.

```
>>> QTable[('forward', 'green', None, None, 'forward')]
10.839883347109732
```

Conclusion: The agent has learnt the traffic rules on its own, it is taking less number of actions from the total available actions to reach the destination. While testing the trained agent over 100 trials, it has shown zero penalties. Overall, the agent has converged to the optimal policy! For much challenging real-life problems, the dynamic learning rate (for better convergence) has to be employed instead of a fixed value.