Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

- Next waypoint location, relative to its current location and heading,
- Intersection state (traffic light and presence of cars), and,
- Current deadline value (time steps remaining),

And produces some random move/action (None, 'forward', 'left', 'right'). Don't try to implement the correct strategy! That's exactly what your agent is supposed to learn.

Run this agent within the simulation environment with enforce_deadline set to False (see run function in agent.py), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

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

- The agent selects action randomly disregarding the conditions of the environment.
- When the light is red:
    - The agent takes action 'None' – the reward is 1.
    - The agent takes any other action than 'None' – the reward is -1.
- When the light is green:
    - The agent takes action 'None' – the reward is 1.
    - For any action other than 'None' – the reward keeps changing to 0.5, 1 and 2
- With enforce_deadline set to False, it explores the whole grid and eventually it makes it to the target location but it takes too long for that.
- With enforce_deadline set to True, the primary agent reaches the destination only sometimes within the set deadline.

Identify and update state

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

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

- First I selected the states next_waypoint and traffic lights.
- I selected these states because these states seem to be the most important states in deciding the action of the agent.
- I implemented the Q-Learning algorithm for these states.
- The agent was performing good.
- But the limitation in these states is the agent doesn't care for traffic, it only takes in consideration the next waypoint and the traffic lights.
- So then I tested the agent by adding more states which are: oncoming, right and left
- Initially I only chose 2 states because I thought the program might not work or work slow or maybe the agent will take more time to learn the policy, given the large size of resulting Q-Table.
- But on trying all the states except deadline, I observed that the program was running smoothly without slowing down, also the agent was learning to follow all the rules applicable on US roads.
- I did not include deadline as a state for my agent because if I include deadline in the agent what will happen is that the agent will try to reach the destination within the deadline not taking into consideration whether it will cause a collision or whether the light is red. This means that the agent will try to reach the destination within the set deadline but it will also break rules or cause collisions in doing so.
- And as I checked the performance of the agent, even when I have not included the deadline as a state, the agent reaches the destination within the given deadline for 999 out of a 1000 trials, so omitting the state deadline is the more preferable option.
- The description of the states are as follows:
    - next_waypoint: Next waypoint to take to reach the destination according to the GPS of the environment. Could be one of forward, left, right.
    - lights: Status of the traffic light. Whether it is green or red.
    - oncoming: If there is any oncoming traffic, it will specify the direction of the oncoming traffic. Could be one of none, forward, left, right.
    - right: If there is any traffic from right, it will specify the direction of the traffic. Could be one of none, forward, left, right.
    - left: If there is any traffic from left, it will specify the direction of the traffic. Could be one of none, forward, left, right.

Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that.

Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behavior.

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

- After implementing the Q-Learning algorithm the agent learns to reach the destination within the deadline.
- Out of 1000 trials, it reaches the destination within the deadline successfully for 999 trials.
- The agent could not reach the destination within the deadline only once during the 157^{th} trial.
- The important thing to notice after implementing the Q-Learning algorithm is that the agent learns to follow the guidelines for driving on US roads.
- Because the length of the Q-Table is 1536, the agent will take much more trials than 1000 to be perfectly able to follow all the rules. So occasionally I notice that there is negative reward when the agent creates a collision by running into oncoming traffic or into traffic coming from the right or left.
- This code can be modified and the agent could be allowed to run until each and every Q-Value in the Q-Table is updated, I believe that the agent could be used successfully in real world with similar scenario.

Enhance the driving agent

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

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?

- First I only used two states to select the best possible action.
- Then I implemented the agent to take into consideration all the possible states.
- I noticed that the agent learns to follow the US rules for driving in road.
- The agent reaches the destination within the deadline successfully for 999 times put of 1000 trials, except the 157^{th} trial.
- Mostly the agent will not cause a collision, but sometimes it will. But once it has learnt what caused the collision, it will not repeat the same action again.
- The collision occurs because the size of the Q-Table is very large and to update all the Q-Values in the Q-Table, we would need to run the agent for much more than 1000 trials.
- After the implementation of Q-Learning was complete, all I needed to do was to find out an optimal value for the Alpha (Learning rate), Gamma (Discount factor) and an optimal Q-Value for initializing the Q-Table.
- The below table shows the agent behavior while using various parameter values.

Test No.	Alpha	Gamma	Q-Value	Total Reward	How many times the agent reached the destination out of 10 trials
1	0.9	0.1	5	296	10
2	0.1	0.9	1	194	2
3	0.5	0.5	5	307.5	5
4	0.1	0.9	10	286.5	0
5	0.9	0.1	10	335.0	9
6	0.6	0.2	10	295	8
7	0.1	0.9	5	196	0

- After trying many different combinations, I predicated the optimal values for Alpha, Gamma and Q-Value to be 0.9, 0.1 and 5 respectively.
- As we can see from the above table, in Test No. 1 and Test No. 5 the agent is performing excellent. But in Test No. 5, the agent is unable to reach the destination in 1 trial although the total reward of the agent is higher than that in Test No. 1. So keeping this in mind I selected the parameters used in Test No. 1 as the optimal parameters for the agent.

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?

- Yes, the agent almost learns the optimal policy. By almost I imply that the Q-Table is very large (1536), and not all the Q-Values in Q-Table can be updated in 1000 trials.
- So occasionally there are some collisions, but from each collision the agent learns and does not create the same mistake again.
- But most of the time the agent successfully reaches the destination without getting into collision.
- After a couple of trials, the agent learns to reach the destination in the minimum possible time while also not breaking any rules.
- It does not take long for the agent to learn to not break the red light.