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?

Answer: By picking an action at random from the list of actions None, 'forward', 'left' and 'right' it can be observed that the agent very rarely reaches the destination when enforce_deadline is set to True. When the the enforce_deadline is set to false the agent take a very long time to reach the destination.

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.

Answer: The states that were used was the direction(None, 'forward', 'left', 'right') and the traffic lights('red' and 'green'). The direction and the traffic light input take together are used to make decision for the agent to follow. There are other input signals like oncoming, left and right which tell us if there is traffic coming from front, right or left of out position. However, these inputs(oncoming, right and left) do not award or deduct points. Therefore, these inputs were ignored. To make the agent comply to complete set of traffic rules there should be some negative point given for running into oncoming traffic and that must be taken into account while implementing the Q-Learning algorithm so that the agent learns to not run into traffic.

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?

Answer: After the use of the Q-Learning algorithm it can be observed that the agent reaches the destination almost every time within the alloted time. It learns to wait at red traffic light to avoid getting a negative reward.

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?

Answer: If we initialize the Q-value to 0 it can be observed that for any value of alpha(learning rate) and gamma(discount rate) the agent stays in the same position till the time runs out. For an initial Q-value of 1 and higher the agent moves from one location to another. Then for alpha value of 1 it can be observed that the agent reaches the destination only sometimes. The agent sometimes also remains in the same position and does not make any move even when the light is green to reach the destination. For lower values of alpha like 0.1 it can be seen that the agent takes a while to learn the policy and avoid negative reward. Similarly, for gamma(discount rate) it can be observed that the agent performs well at lower values than when gamma is set at higher values. After trying many combinations of alpha and gamma, the optimal values were found to be 0.5 and 0.2 for alpha and gamma respectively.

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?

Answer: The agent learns to avoid negative rewards and always reaches the destination. It is close to finding an optimal policy. However, it sometimes does not take the shortest path.