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 basic algorithm reaches to the destination but agents made a lot of wrong decisions such as moving forward when the light is red etc.

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.

- I have picked light condition, left and oncoming traffic condition, in addition to planner next way decision. These are illustrated in Figure 1 and 2. As I've explained in the following part, agent does learn what to do on red traffic sign, or green light, as well as when there is a traffic from left side or oncoming.

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?

- I have implemented the Learning algorithm and have seen that agent reaches the destination for most of the trials. It learns that taking no action or going right when the light is red is the correct behavior as you can see from the table below. Each column represents a state variable such as light condition, oncoming traffic planner action etc. Value is the learning parameter, higher values are more appropriate action state pairs.

Index	light	oncoming	left	planner_actior	learner_action	value
0	red	none	none	none	none	0.13
1	red	none	none	none	forward	0.0519
2	red	none	none	none	left	0.0776
3	red	none	none	none	right	0.184
4	red	none	none	forward	none	2.62
5	red	none	none	forward	forward	0.756
6	red	none	none	forward	left	0.708
7	red	none	none	forward	right	2.2
8	red	none	none	left	none	2.69
9	red	none	none	left	forward	0.643
10	red	none	none	left	left	0.676
11	red	none	none	left	right	2.18
12	red	none	none	right	none	2.73
13	red	none	none	right	forward	0.656
14	red	none	none	right	left	0.644
15	red	none	none	right	right	3.63

Figure 1

- It learns that following the planner actions does help it to achieve more rewards.

Index	light	oncoming	left	planner_actior	learner_action	value
261	green	none	none	forward	forward	8.74
262	green	none	none	forward	left	2.23
263	green	none	none	forward	right	2.17
264	green	none	none	left	none	2.83
265	green	none	none	left	forward	2.18
266	green	none	none	left	left	4.31
267	green	none	none	left	right	3.38
268	green	none	none	right	none	2.83
269	green	none	none	right	forward	2.28
270	green	none	none	right	left	2.21
271	green	none	none	right	right	7.29
272	green	none	forward	none	none	0.146
273	green	none	forward	none	forward	0.156
274	green	none	forward	none	left	0.1
275	green	none	forward	none	right	0.058
276	green	none	forward	forward	none	1.1
277	green	none	forward	forward	forward	7.7
278	green	none	forward	forward	left	1.51
279	green	none	forward	forward	right	2.02
280	green	none	forward	left	none	1.51
281	green	none	forward	left	forward	1.25
282	green	none	forward	left	left	3.39
283	green	none	forward	left	right	1.21
284	green	none	forward	right	none	2.79
285	green	none	forward	right	forward	1.79
286	green	none	forward	right	left	1.85
287	green	none	forward	right	right	0.0981

Figure 2

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?

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?

- I have changed the alpha (learning rate) and gamma (discount factor) factors and e-greedy algorithm threshold value to change the probability of random actions in the learning algorithm. I have reduced the threshold in order to make agent use the learned actions more often. This has improved agent to reach the destination quicker. Basically, I have changed the ratio of random actions (exploration) and exploitation.

- I have also started with very small alpha factor (learning rate~0.1), which makes learning algorithm to converge at longer time. Then I have increased he alpha value to increase the speed of convergence.

- I have also reduced the discount factor (where I have started with 0.95) which also improved the convergence by relatively increasing the current reward factor in update equation.

- I always allow agent to take random actions which results in penalties but this is part of the learning (exploration)