Project Report

You will be required to submit a project report along with your modified agent code as part of your submission. As you complete the tasks below, include thorough, detailed answers to each question provided in italics.

Implement a Basic Driving Agent

To begin, your only task is to get the smartcab to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection:

- The next waypoint location relative to its current location and heading.
- The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions.
- The current time left from the allotted deadline.

To complete this task, simply have your driving agent choose a random action from the set of possible actions (`None`,`'forward'`,`'left'`, `'right'`) at each intersection, disregarding the input information above. Set the simulation deadline enforcement, `enforce_deadline` to `False` and observe how it performs.

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?

> *ANSWER*:
>
> The smartcab does not make it to the destination. Actions are being taken randomly so there is no learning of the best actions to take and no policy is being applied. Hence there is no need to end up with the route, in average the smartcab gains more if it keeps on moving.

Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the smartcab and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the `self.state` variable. Continue with the simulation deadline enforcement `enforce_deadline` being set to `False`, and observe how your driving agent now reports the change in state as the simulation progresses.

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?

ANSWER:

The current state is defined by the possible combinations of: **light** ('red', 'green'), **oncoming** (None, 'forward', 'left', 'right'), **right** (None, 'forward', 'left', 'right'), **left** (None, 'forward', 'left', 'right') and **next_waypoint** (None, 'forward', 'left', 'right').

In the agent code I generate the state and add it to the list of states that have already occured.

```
this_state= (inputs['light'], inputs['oncoming'], inputs['righ
t'], inputs['left'], self.next_waypoint)
if not this_state in self.list_of_sates:
    self.list_of_states.append(this_state)
self.state.append(self.list_of_states.index(this_state))
```

I believe these states are appropiate for this problem since they can describe all the possible situations occurring at each intersection, taking into account both the input variables (lights and traffic) and the direction in which we hare heading next, which should be the action if not infracting any rule.

I could have also included in the state definition the remaining time to meet the deadline ('deadline = self.env.get_deadline(self)'), but this seems not a good option in terms of both computational time and memory needed, since resultant states would be infinite, as it can be the number of possible deadlines.

The total possible states are 512, and this number is obtained as follows:

```
number_of_states= len([ 'red', 'green'])* len([None, 'forward',
'left', 'right'])* len([None, 'forward', 'left', 'right'])* len([None,
'forward', 'left', 'right'])* len([None, 'forward', 'left',
'right'])=2 *4 *4 *4 *4 = 512
```

Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by the smartcab will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement enforce_deadline to True. Run the simulation and observe how the smartcab moves about the environment in each trial.

The formulas for updating Q-values can be found in this video.

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?

ANSWER:

Random actions gave erratic movements of the car and when a policy is applied these movements are more structured. This is happening because the system is learning and the action is being selected following the policy, which takes into account the reward for the action and the Q.

Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the smartcab is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your smartcab:

- Set the number of trials, n_trials, in the simulation to 100.
- Run the simulation with the deadline enforcement enforce_deadline set to True (you will need to reduce the update delay update_delay and set the display to False).
- Observe the driving agent's learning and smartcab's success rate, particularly during the later trials.
- Adjust one or several of the above parameters and iterate this process.

This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

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?

ANSWER:

I have changed the values of the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) and seen how the smartcab performed, analysing the success rate (how many of the rides ended before deadline) and the number of infractions committed (ilegal movements not following traffic rules).

Simulations are for 100 trials. Success rate is defined as number of success / number of trials.

For the simulations 1, 2 and 3, I only increased the learning rate (alpha), the discount factor (gamma) is set to the a very low value and the exploration rate (epsilon) is 0 (no randomness). It can be seen that the success rate increases as the learning rate does, since the cab learns faster and more rides end before deadline. The number of infractions, is also reduced with faster learning (shorter rides result in lower possibilites of infractions).

For simulations 3, 4 and 5, the discount factor (gamma) was changed while keeping the learning rate (alpha) value to 1 and the exploration rate (epsilon) equal to 0. The discount factor determines how much importance is given to the past experience (Q) in order to take the next action. This means, if the discount factor is bigger, less importance is given to the present reward but more to the previous ones trough the utility. Hence, the number of infractions will be reduced (previous infractions have given negative rewards and we have learned from that!). However, aiming for low infractions and avoiding negative rewards we are following in the same amount the traffic rules (rewards) and learning from past experience (gamma -> Q), which results in lower rate of success.

If we change the exploration rate (simulations 6 and 7), we are adding randomness to the following of the policy. This means, higher exploration rates (epsilon) will be reflected in higher number of infractions, since we will not be following the rules that maximize the reward due to the randomness introduced.

	learning rate (alpha)	discount factor (gamma)	exploration rate (epsilon)	success rate	infractions
1	0.2	0.01	0.0	75%	72
2	0.5	0.01	0.0	78%	60
3	1.0	0.01	0.0	90%	59
4	1.0	0.5	0.0	88%	64
5	1.0	1.0	0.0	11%	38
6	1.0	0.01	0.1	91%	111
7	1.0	0.01	0.5	34%	356

The perfect agent will have the highest learning rate (to learn fast have fewer infractions), low discount factor (to give importance more importance to present reward in order to follow the ruls and but also learn from previous experience (Q)) and low exploration rate. Longer simulations could be launched in order to have more samples and reduce bias.

Given the results I would choose to mostly follow the traffic rules but also learn from past with zero randomness, this could be having $\alpha = 1.0$, $\gamma = 0.001$ and $\epsilon = 0.0$. From table we would have 90% of success with 59 infractions, which is a good and compormise value, when compared to the others.

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?

ANSWER: I think our agent is close to the optimal policy since the success rate is quite high (above 90%). I don't think it is close to reach the point of optimal policy having zero infractions, since sometimes we have a greater reward if we make an infraction but we end in time.

The optimal policy for this problem would be that following reaching earlier while commiting the lowest number of infractions.