Implement a basic driving agent

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

With the basic implementation of the driving agent that takes a random choice of none, forward, left, or right as available actions, the agent moved haphazardly through the map in a non-optimal way. It was still able to reach the target about 20% of the trials out of sheer randomness. There were many trials where the agent ran out of time.

Identify and update state

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

All the current inputs of light and traffic behaviors as well as the next waypoint information from the route planner were represented. It was very important for the agent to have as much information as possible to learn from. Hence, the lights state allows the learner to understand how to move through the map in an orderly fashion without incurring penalties. The three input of `oncoming`, `left`, and `right` gives the agent a sense of what is happening in the environment with other driving agents as it has to obey the traffic rules. Lastly, the next waypoint enables the agents to learn a sense of direction relative to the destination, which is the most crucial part of being able to reach the target consistently.

Implement Q-Learning

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

After implementing the Q-Learning algorithm, the driving agent became smarter as it started to be aware of its environment as it started to follow the traffic rules and it tries to find the a way to reach the target. The smart driving agent is a result of computed Q values that favour and score higher for appropriate actions that lead to the target destination, and that opposes and scores low for illegal actions. Yet, it still was prone to errors and running out of time. Allowing the agent to learn for 10k+ trials only yielded a 35% success rate. It was better than randomly selecting an action, but improvements needed to be made to the learner.

Enhance the driving agent

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?

There were two changes made. First was on randomizing the selection of the best action to take when there was a tie, and the second was optimizing the learning algorithm to be explicitly greedy.

When selecting the best action to take, the first function employed used a lambda expression to select the action with the highest Q value. But in the event of a tie, the function returned the action at the lowest index. This yielded a low accuracy as the agent wasn't able to explore and learn in these cases. An improved implementation was to generate a list of possible actions to account for ties and then picking a random action from that list. Over time, the driving agent started to recognize the better paths to the targets and there appeared to be less ties for the best actions.

The original Q-Learner was setup as follows:

```
Q(S,a) = alpha*Q_hat(S,a) + (1-alpha)*Q(S,a)
```

But on further investigation and testing, there appeared no need to incorporate the discount factor, gamma, and future action states. This is because the algorithm is inherently greedy as the agent receives a reward for each correct action and a penalty for an incorrect move or a traffic violation. The agent also receives a large reward for reaching the target in time. This is distinctly different finding the shortest or fastest possible route to the target - as long as the agent reaches the target location within the allocated number of moves denoted by the variable `deadline`, it received a large reward of 12.0. Therefore, it was possible to exploit the greediness of the algorithm by maximizing for short-term gains of the next best action instead of the long-term goal of reaching the target the fastest. This was inline with the project goal as well:

> Based on the rewards and penalties it gets, the agent should learn an optimal policy for driving on city roads, obeying traffic rules correctly, and trying to reach the destination within a goal time.

On top of the algorithm changes made, the alpha value of between 0 and 0.5 was tuned for the highest trial success rate over 100,000 trials, respectively at 0.1 increments. It was found that the most optimal alpha value was 0.2. At a higher learning rate, the driving agent attempts more random actions in an attempt to explore but this leads to a slightly reduced success rate. The learning rate of 0.2 offers an optimal balance of exploring and execution. A possible improvement could be decaying the learning rate over time where the agent would have high alpha value at the start that decreases as the agents learns the best actions to take.

alpha	success rate
0.0	20.000%
0.1	99.881%
0.2	99.907%
0.3	99.858%
0.4	99.867%
0.5	99.866%

The agent learns quickly to not perform invalid moves in early trials and optimizes itself in that sense. Interestingly, the agent does not encounter oncoming traffic all too often to learn the best way to navigate through a congested environment. This has caused the agent to incur a negative penalty in some rare instances for not following the traffic rules, such as turning into other driving agents. Perhaps this can be rectified by introducing more dummy driving agents in the environment.

With modifying Q_hat to be the reward value only for the possible action, the Q values promoted the agent to take the best possible action at that point in time in reference to the next way points. While finding the minimum possible route was not promoted, it is likely that in some trials, the agent still takes the shortest way to the target. A fully optimal agent would not perform invalid actions, follow traffic rules, and take the shortest route to the target destination. Still, the current algorithm succeeds at a very high rate of over 99%.