Implement a basic driving agent

What you see in the agent's behavior. Does it eventually make it to the target location?

The agent is acting randomly because we are selecting a random action from valid actions, so its behavior is not predictable and, as a result, it cannot be efficient. It does eventually make it to the target but usually after deadline, because the grid is not infinite.

Identify and update state

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

I picked all of the input values because I think each of them represent some important information from the world. Based on my understanding state should be a set of data that represent the agent's world. Traffic Light is important because it says when we should move and when we should stop and also information about other cars relative to us helps to prevent from an accident and getting better reward. I also added the next waypoint because it makes our states more unique.

Implement Q-Learning

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

After implementing Q_Learning, I realized as time goes on the agent learns more from his mistakes and acts more efficient. It passes traffic light less and gains more reward in comparison with random acting; also it can find its way much faster with higher reward. But it still doesn't choose the best action and the best way; I think there is space to improve.

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?

The first thing that I changed to make my model was **learning-rate,** which can help you to change your current values. It technically determines the learning speed or updating speed that how fast you want to update old information with new experiments, the learning-rate range is between 0 and 1. If you set your learning rate to 0 it means you never learn and in the other hand if you set learning set to 1 it makes the agent deterministic. . I choose 0.78 for Learning-rate.

Second parameter that I changed was **Discount-factor,** based on my research this parameter determines the Importance of future rewards, if we set discount-factor to 0 the agent only consider recent reward or in technical term it makes the agent short-sighted and if we set discount factor to 1 it considers long-term high reward and makes the agent long-sighted. Making a trade off between recent or long high rewards was my second optimization. I choose 0.43 for Discount-factor.

Another optimization that I made was about initialization for new instances (unseen states). I could start with zero but most of my actions became "None", because "None" was first action. I preferred to initialize the unseen states with higher value to do some action until doing noting. Also I found some idea that we can initialize new states with reward. This idea says the first time an action is taken the reward is used to set the value of Q.

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?

After I made some changes in the parameters and a trade of between alpha (Learning rate) and gamma (Discount factor) I think I have better performance and my agent is performing more efficient than before, but I still can do some extra work to make the performance even better. For example we can prevent the agent from moving forward when the light is red or can use grid search to find the best parameters.

Based on the result the agent is reaching its destination in the minimum possible but still there is some space to improve. If I run the engine for more than 50 times after that point success rate would be close to 98% or even better. Penalties and minimum possible are strongly related to trials time. The more we run this engine we get fewer penalties and we reach destination in the minimum possible.