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?

The agent wanders around the grid with no apparent direction due to the fact that its actions are selected at random. There is no guarantee that the agent will ever make it to the target location. All directions are uniformly chosen at random, so there is no way of telling how likely it is that the agent will ever reach its destination.

Identify and update state

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

These are the states that can be immediately observed by the learning agent. The agent should pay attention to its future location, whether the light is green and allowing it to move, and if there are cars oncoming from any direction. Some states were unnecessary, however. The oncoming traffic from the right is nonessential if you pay attention to the color of the light. On the other hand, after reading some discussion and looking over the code, it appears that the *oncoming* and *left oncoming* traffic also do not matter because the agent is already programmed to acknowledge *right-of-way.* This is helpful because it severely reduces the input space that the agent needs to learn. If this were not the case, then it could still be argued that the only other input dimension to add to the state would be *oncoming* traffic. That would make sure that the agent learned not to make left turns when there is *oncoming* traffic. Adding the *deadline* to the state would add far too many discrete states to explore in the limited amount of time there is to arrive at the destination. It is possible to decrease the number of states in the deadline by referencing fractions of the *deadline* time left as discrete variables.

Implement Q-Learning

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

The agent starts to take the None action more as it observes a +1 for every time it does not run a red light. It is not initially encouraged to find the goal because it receives rewards for following correct rules of the road. It does not receive a huge negative reward for not finding the destination, so it makes sense that it would try to receive multiple small rewards instead of searching and making a mistake. After receiving a few +2s for following through with the *next-waypoint* it starts to take more and more *lefts, rights, and forwards.* Since these waypoints are indicated by the planner, the agent works towards the eventual goal and eventually receives a large reward that will end up contributing to its tendency to follow the *next-waypoint*.

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?

I tried implementing the ego-centric and allo-centric models in the tutorial provided at this link:

https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/

This was to encourage more exploration without encountering the issue of the agent choosing the None option because it gives short-term returns. This did not have much of an effect, because the agent quickly learned and relied on the optimal policy. The basic implementation of Q-learning with limited and carefully selected states learns an optimal policy very quickly. All other aspects of Q-learning, like the learning rate and gamma, were selected after reading many tutorials. They are not scientifically verified, as each problem will have its own optimal values, they are simply suggested values to start near.

I tried adding a fractional representation of the deadline to see if the agent would learn how to use this as an excuse to gain more rewards before getting to the exit, but did not see much improvement. Instead, I used this representation to determine the random action rate, so the agent would be encouraged to explore until the last fractions of the deadline of the trial.

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?

Due to randomness, it is hard to strictly state the performance of the learning agent. Every so often, the agent is programmed to take a random action that might lead to a negative reward. This can help in the learning, but hurts the overall performance. The agent quickly finds a good policy that achieves cumulative rewards per round near 30. This stays pretty stable no matter the changes made to the learning agent.

It also reaches the goal in around 80% or more of the trials. In the last 50, 25, and 10 trials it tends to do better than the overall average and does better as the trials approach the last trial.
Once again, due to random actions within trials, minimum possible time does not make for a good metric to measure the efficiency of the learning agent.
All analysis was done using the output_analysis.py file.