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

Running the driving car simulator with a policy that just picks one of the four possible positions (None, left, right, forward) at random and returns it as an action shows some interesting behavior from the agent. First off, we can see the agent moving around the grid world in a very erratic fashion. Also, we notice the rewards it gets when it performs some correct moves (2 points) at the intersections as well as the penalties it suffers when it violates some of the rules of world, in this case traffic rules (-0.5 points) and crashes (-1 point). However, even moving in a random way, it does, eventually, get to the destination, and when it happens, we can see a larger reward (12).

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

Based on the problem setup, I believe the best way to choose the possible states that the agent can be is by looking at the set of inputs available and make some combinations that model both the agent and the environment with the most description. According to the problem specification, the agent has an egocentric view of the grid intersection it is at being able to see if there are other agents in front of itself, or in one of the both roadways side. The agent can also sense the state of the semaphore, thus, being able to identify it as green or red along with the next waypoint that it should take in order to reach the desired destination. Based on these data, a possible state the agent can be is defined by the combination of these input sensory data. For instance, suppose the agent is at some intersection where the semaphore is **green**, and there are no agents in any of the incoming roadways. This state is represented as:

***light: green, left: None, oncoming: None, nextwaypoint: forward***

As another example, suppose the agent is at an intersection and there is another agent in front of it that wants to turn right, and in this case the semaphore is **red**.

***light: red, left: None, oncoming: right, nextwaypoint: forward***

Note also that in both cases the agent should go forward to comply with the destination goal.

With this structure, we can model the necessary number of states that can differentiate the various states that the agent can be based on the variety of input possibilities that there are. One thing that is worth noting is that there is no indication in this state representation that there could be an agent coming from the road in the right. There is because according to the traffic rules stated for this project, the fact that there is a car coming from the right or not does not affect the current agent at all. In other words, regardless of the semaphore color, the addition of an agent coming from the right does not create a new situation or traffic violation that is not already defined, and that also helps the model to keep a concise Q-Learning table which in turn, benefits the agent’s performance.

Another worth mentioning detail is the fact that to keep the QTable as small and concise as possible, I decided to not hard code all of the possible states at the beginning, but incrementally adding the states to the QTable as it is needed. This is also another strategy that I found would keep the QTable simpler and then more efficient to process.

1. **What changes do you notice in the agent’s behavior?**

After implementing a basic version of Q-Learning and programming the agent to use, instead of a random selected action, the best action from the Q-Learning table, some aspects regarding the behavior of the agent have changed. Firstly and most important this scenario brings to the light the trade-off between exploration and exploitation. It is pretty clear that only choosing the best action from the QTable from the beginning, hurts the model in the sense that depending on what action it will choose to break ties, the result can be very different. Because the QTable starts with 0s all over it at the beginning, for any action the agent asks the QTable for, the result will be the same because of the way the Q-Learning equation works. Because the agent has not seen most of the states at the beginning of the process, it did not learn anything about the environment and therefore, when the Q-Learning tries to pick the best option from the QTable, there is no best option yet, so one of the 4 possible actions must be chosen to break ties. In my experiments, when the None option is the one chosen in this situation, the agent simply does not move and consequently does not learn anything at all. However, when the forward option is chosen, the agent surprisingly acts quite well reaching the destination in about 60% of the hundred trials. I believe that is the core of the exploration exploitation dilemma, in order to use what it knows (exploitation) the agent must first learn the environment (exploration), and it is clear in the first case that when the action is None (for breaking ties) there is no exploration and consequently the agent is not able to learn anything about environment. On the other hand, in the case of the 'forward' action for breaking ties, the agent gets the chance to learn the environment when there is no best option from the QTable yet available, which in turn makes the agent able to fill the table with optimum values that helps in future decisions.

To be more specific in the result, an experiment was done using this basic Q-Learning implementation with no discount factor and a constant learning rate of 0.5. Over the 100 trials, the agent was able to accomplish the goal with an average of 74% accuracy. In this scenario, the forward action is chosen every time there is no best (maximum) value to pick.