Implement a Basic Driving Agent

In this task, I added a new variables list ‘self.actions’ which contains all the possible actions for the agent(None, 'forward', 'left', 'right'). Then I set action = random.choice(self.actions) to choose a random action when we call update().

In this case, half of the time smartcab make it to the destination, but the other half of the time it hits the hard time limit and stops. Most of the time the agent just gives out wrong actions, so the car has to wait for the light to turn green and wondering around. Since we have more than 100 moves in this case and the map is only 8x8, half of the time smartcab will make it to the destination either by miracle or go over most of the map.

Inform the Driving Agent

In this task, I defined 576 possible states for the agent. Each of these states defines smartcab’s current traffic light, traffic information, next waypoint, and deadline. There are 2 traffic lights, red and green. During red light, smartcab only can stay or turn right depending on the traffic on the left and traffic in the other directions doesn’t matter. During green light, smartcab can turn left depending on the traffic oncoming and other actions with any traffic. Thus, traffic on the right doesn’t matter in either case, so I only put the traffic information on the left and forward in each state. Also, the next waypoint helps smartcab to determine the destination, so I include it in each state.

The longest deadline is 70 because the longest distance in an 8x8 map is 14((1,1) to (8,8)) and the deadline is 5 times the distance. Then I divide the deadline into 6 intervals each with the length of 12 because making too many intervals will overfit the data and slow down the learning process. Therefore, the total number of possible states is 576 (2 light signals x 16 possible actions for traffic information on the left and forward x 3 possible waypoints x 6 deadline intervals).

Each of these states is appropriate for this problem since each set of valid actions defines what action is appropriate in that situation. Also, each waypoint gives information for which action will gain a reward. Lastly, each deadline interval suggests how fast should smartcab reach its destination. Deadline intervals are important because smartcab can make more reward by taking the right action toward the waypoint than just go straight to the destination. Thus, without the deadline feature in each state, smartcab will go back and forth to maximize the reward rather than go to the destination. Hence, deadline intervals can alert smartcab to reach the destination before the deadline.

Implement a Q-Learning Driving Agent

In this task, I implemented Q-Learning and update the q-values after taking each action. The agent also picks the action with the largest q-value in a given state. Therefore, the agent will try to maximize rewards after each move. After a few trials, the agent will make valid moves while not always following the waypoint at the beginning. Then, it tries to move toward the waypoint when the deadline is getting close because that gives the maximum reward. This kind of behavior occurs since there’s no penalty for not making to the destination and no reward for finishing early. Therefore, the agent tries to last the trip longer to gain more reward by not the following waypoint when we have a long deadline. However, once the deadline gets close, the agent will try to reach the destination since it wants that 10 rewards at the end.

Improve the Q-Learning Driving Agent

In this task, I tried few different sets of alpha values and the best set of values I got are alpha = 0.1, gamma = 0.7, and epsilon = 0.01. With this set of parameters, the agent can make it to the destination more than 95 percent of the time. I also tried with alpha = [0, 0.5, 0.8]. When alpha is 0, the agent will not improve at all because there’s no learning involves in this case, so the agent only makes it to the destination 50 percent of the time. When alpha is 0.5 or 0.8, the learning rate is too high and the agent pays too much attention to the initial reward and the q-value for next state. As a result, the agent doesn't pay attention to the past experience and try to maximize reward from the next state by making a violated move. Thus, even though the agent can make it to the destination more than 90 percent of the time with alpha 0.5 or 0.8, it violates more moves to make it to the destination, so alpha 0.1 is the best parameter.

My final agent can find a good policy but not the optimal policy. Since there’s no reward for making it to the destination as soon as possible, the agent will try to follow some of the waypoints while not finishing the trip to maximize reward. The optimal policy for this problem should make it to the destination with the shortest path while making valid moves. In order to achieve this, we need to change the environment and give rewards for finishing as soon as possible.