P4: Train a Smartcab to Drive

Section 1 – Implement a basic driving agent

The agents.py file was altered to include a randomized selection of action for the smartcab. On each iteration, None, forward, left, or right were randomly selected. This was run 10 times with the following results.

Iteration	*Number of Steps to Goal*
1	10
2	90
3	22
4	172
5	52
6	33
7	192
8	7
9	40
10	105
Average	72.3

There appears to be no improvement in the driving agent behavior, which is to be expected if we are picking random actions. On average, it took the driving agent 72.3 moves to make it to the destination.

Section 2 – Identify and Update States

When run out of the box, the code returns a state list that includes

- Light
- Oncoming
- Right
- Left

For the purpose of the Q learning algorithm, it is necessary to include a general “ideal” direction to move the agent closer to the destination, so for my algorithm the basic state list is rewritten as:

- Light
- Oncoming
- Right
- Left
- Next Waypoint

Optionally, there could also be a deadline state to incentivize the agent to move faster to the goal as the deadline approaches. In my first iteration of this algorithm I left this term out as I found it complicated the Q table and did not improve results.

Section 3 and 4 – Implement Q-Learning and Enhancing the Driving Agent

I implemented my Q-Learning using the GLIE algorithm with a tunable decay of the "random restart" factor by a percentage on each trial. I found that when the deadline is enforced, the driving agent reaches the destination approximately 80% of the time in a 100 trial run and arrives, on average, in 14 actions. This is slightly skewed as it generally takes 15-20 trials for the driving agent to begin formulating a successful policy.

If the driving agent is allowed to make unlimited actions per trial, the performance was generally much poorer. I believe this could be improved by a more sophisticated reward structure. This result is reinforced by a run of the agent with the deadline limit enforced using 500 trials. Performance above approximately 225 trials generally begins to deteriorate as the Q table becomes more complex. This leads me to believe that, for this algorithm, implemented in this manner, the optimal policy is achieved between 100-180 trials.

Within 100 trials, this algorithm returns approximately 70 unique state-action pairs. Once the algorithm discovers these 70 pairs, it appears to perform very well, with a success rate in excess of 90% of reaching the destination with a positive reward and within the deadline. Running the algorithm with 500 trials, it discovers approximately 30 additional state-action pairs, indicating that within 100 trials, approximately 70% of this state space has been explored and iterated upon.

Methods of Improvement:
There is a lack of urgency for the agent to follow the next directional waypoint regardless of consequences as the deadline nears. As a result, the agent sometimes gets stuck making cumulative decisions that sum to a positive reward, but do not advance the agent towards the destination. This could be improved with some form of weighted reward system as the deadline nears. I experimented with such a system but was unable to find the performance I was looking for and subsequently commented out portions of code related to that effort.

Notes and console output is provided in the agent.py file for the rest of this section.