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

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.

Tuning

Using the above described testing method of 100 trials, limited by deadlines, I tuned the model by altering the weighting factor and learning rate. The results of those tests are included below.

Table 1 Adjusting Learning rate with fixed Discount Factor

Discount Factor (gamma)	Learning Rate (alpha)	Average # of Actions to Goal	Success Rate
0.5	0.5	14	79%
0.5	0.3	14	55%
0.5	0.7	14	52%

Table 2 Adjust Discount factor with fixed learning rate

Discount Factor (gamma)	Learning Rate (alpha)	Average # of Actions to Goal	Success Rate
0.3	0.5	15	69%
0.7	0.5	15	57%

After running these tuning trials, it appears that optimal performance is achieved when learning rate and the discount factor are set to 0.5. The second set of tuning trials were to decide on a "random restart" percentage and a decay factor for that percentage.

Optimal Policy Discussion

With the experiment setup in this way, there are 192 possible state-action pairs. (2 possible light states, 2 left, right and oncoming vehicle states, three possible waypoints, and four actions). The theoretical optimal policy would have the agent explore all 192 and correctly assign a high Q-value to 48 possible state-action pairs, these would represent he optimal action at each possible state, and a lesser value to the remaining 144 possibilities, indicating a sub-optimal action.

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. This seems like a small percentage of the explorable space being explored, but in a space where the agent is never more than 12 moves away from the goal (moving from opposite corners) the rest of the state action pairs appear to be unnecessary to achieving the goal of reaching the destination within a specified number of actions with positive reward. If the space were larger, the specified number of actions fewer, or the reward structure more negative to sub-optimal actions, this policy would most likely not provide good results.

Methods of Improvement

These are simply thoughts on how the agent could be improved and not necessarily items that were implemented in my agent. As the goal of this project is to implement the agent in an effective way, that has been achieved, for further improvements, I've discussed them below.

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 in two ways.

1. I believe the current reward system could be tuned to sum to a negative reward if non-advancing moves are repeatedly taken. As it is now, the reward system only gives negative reward when a dangerous move is taken such as running a red light. This could technically sum to large positive reward just for circling the block "legally". This system offers no reward to progress to the goal. A better reward system would be a slightly negative reward for a non-progressing but "legal" action and a heavily negative reward for "illegal" moves. This would generate a safe driver with incentive to move toward the goal.
2. This could also take the form of a non-linear reward system. As the deadline approaches, the reward structure would be more skewed toward taking an action, preferably toward the goal. This would reduce the positive 1 reward for a "None" action as the deadline approached.

Both of these would require deeper edits than the "agent" file and I think are beyond the scope and intent of this project.

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