P4: Smartcab

Basic Driver Agent

As expected, the agent moves around randomly. It does make it to the target location, sometimes in a short amount of time, other times in a very large amount of time, after the deadline has passed.

Identify and update state

I chose to keep the following variables in the driving agent's state:

- next_waypoint
- light
- oncoming

The `next_waypoint` variable will help our primary agent to learn to go in the direction necessary to get to the target coordinates. The `light` variable will help our primary agent to obey traffic laws. Oncoming traffic is important when our agent needs to make a left turn, since it should wait if there is oncoming traffic.

I didn't chose left and right because I think that the optimial decisions in this environment doesn't really depend on them. The dangers of running into left and right agents are covered as long as the traffic light is obeyed by all the agents (which is only applicable in this simulated environment, not in real life).

I also didn't choose deadline because of two reasons: First, the speed of the primary agent doesn't change in the simulations, so the only way to get somewhere faster at the same rate is to ignore red lights and make turns when there is oncoming traffic, which I don't want my autonomous agent to learn to do. Second, the agent will only see each state onces, per simulation run, since the deadline decreases by 1 at every turn, so it will never be able to learn from the states.

Implement Q-Learning

I implemented Q-Learning using the update rule from the lecture(https://www.udacity.com/course/viewer#!/c-ud728-nd/l-5446820041/m-634899062) with $\alpha = 0.1$ and $\gamma = 0.1$. At first, I initialized the Q values for all states to 0. The problem with this method was that when it came to selecting the best action, because all the actions had a reward of 0, the argmax just resulted in the first element in the list of valid actions, which was "None". As a result, the primary agent would not move at all.

Then I created random initial Q values in the range $[1.0, 5.0)$ for all the states, and the agent started moving around in the environment. It would at first move around randomly, but as time went on, it started becoming "smarter" and seemed to head towards the destination. As the Q score table became updated with each learning step, the agent got closer to following the optimal policy.

Enhance the driving agent

I optimized Q-Learning by fine-tuning the learning rate α and future reward discount rate γ parameters. The following tables summarize the results of trying different combinations of these two parameters. (Simulators were initialized with `update_delay = 0` in order to expedite processing)

Holding $\gamma = 0.1$ constant

Alpha	Num reached dest. (out of 100)	Runtime
0.1	95	1m9.007s
0.25	97	1m8.659s
0.5	99	1m6.446s
0.66	99	1m4.527s
0.75	99	1m5.649s
1.0	99	1m15.733s

The last four values of alpha produced equal number of destination reached counts, so I chose alpha = 0.66 since it had the lowest runtime. Holding $\alpha = 0.66$ constant (set frame_delay param to 0 so runtime is lower than previous table)

Gamma	Num reached dest. (out of 100)	Runtime
0.001	98	0m51.023s
0.003	100	0m49.877s
0.005	99	0m47.457s
0.007	99	0m50.636s
0.01	99	0m51.540s
0.05	98	0m48.870s
0.1	98	0m49.784s
0.5	35	1m28.908s

The optimal values for the parameters are: $\gamma = 0.66$ and $\gamma = 0.003$. The following table is the result of re-running the simulations 5 times with the optimal parameters found above.

Simulation (100 runs)	Num reached dest. (out of 100)	Num negative reward steps
1	98	17
2	99	19
3	98	18
4	99	20
5	99	14

Negative rewards indicate non-optimal policy. Though in the runs where the agent reached the destination result in net positive rewards, not each step taken produced positive rewards, as can be seen the in last column of the table above. Analyzing the logs of each simulation showed that there were negative reward steps up to run 81. This means that after that, the policy was closer to optimal, but we can't really determine that it's optimal because we don't know the maximum.

Lastly, I re-ran the simulations with 1000 runs, and saw that there were negative reward steps past run 693. This means that run 1000 must be much closer to optimal policy than run 100 in any of the previous 5 simulations.