Project 4 of Machine Learning Nanodegree: Smartcab

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

At this point in the project, the update function in agent.py looks like this:

```
def update(self, t):
    # Gather inputs
    self.next_waypoint = self.planner.next_waypoint()  # from route planner, also displayed by simulator
    inputs = self.env.sense(self)
    deadline = self.env.get_deadline(self)

    # TODO: Update state
    self.state = inputs

    # TODO: Select action according to your policy
    action = random.choice(self.env.valid_actions)

    # Execute action and get reward
    reward = self.env.act(self, action)

    # TODO: Learn policy based on state, action, reward

    print "LearningAgent.update(): deadline = {}, inputs = {}, action = {}, reward = {}".format(deadline, inputs, action, reward)  # [debug]
```

As you can see from the code, the action variable is simply a random choice between the valid inputs: forward, left, right, none. Inside the simulator, the agent moves randomly according to this action variable around the map. The agent eventually makes it to the target location, but it can take a long time to get there due to the random nature of the action.

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

I chose to pick states based on the inputs for "light", "oncoming", "left" and the next waypoint as defined by the route planner. I believe that the oncoming and left traffic are the only other cars that the learning agent has to worry about since these are states in which there is potential to break a traffic law by making a bad left or right turn. If oncoming and left traffic is

clear, then the only other things that the agent needs to worry about is if the light is red or green in order to pass through the intersection and where the next waypoint is in order to get to the destination as quickly as possible.

Implement Q-learning. What changes do you notice in the agent's behavior?

I began by implementing a simple version of Q-learning in which the agent does not take future reward into account. Q-values are updated as:

$$Q(s,a) = Q(s,a) * (1 - learning\_rate) + reward(s,a) * learning\ rate$$

The change in the agent is immediately apparent. After not reaching the destination three or four times in the early trials, the learning agent performs surprisingly well and almost always reaches the destination within the time limit. In fact, the learning agent looks like it almost always takes the shortest path, sometimes preferring to take a few right turns in order to get around red lights.

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties?

In the end, I chose to keep this implementation of the Q-learning algorithm with a learning rate of 0.8 and a discount rate of 0. This implementation weights the agent's decisions more towards what it has most recently learned and less towards the current situation at hand. This fact that the discount rate is zero also makes the agent myopic, or that the agent is short-sighted, only considering current rewards rather than long-term rewards.

Below are the results of testing different values for learning and discount rates.

Learning Rate	Discount Rate	Success Rate	Total Reward
0.5	0	95%	2815
0.8	0	97%	2951
0.8	0.1	96%	3042.5
0.8	0.5	95%	2890.5
0.5	0.1	93%	2843
0.5	0.5	92%	2837
0.2	0	84%	2892.5

By looking at the table, one can see that the Q-learning implementations that had the highest success rates are those with a high learning rate and zero discount rate.

I would define an optimum policy as one in which the agent takes the the route with the shortest distance to the destination while avoiding penalties. The final implementation with a learning rate of 0.8 and a zero discount rate comes extremely close to this policy and almost always heads directly towards the destination with minimal deviation after the first couple of trials. Other implementations of the algorithm, while successful, stray from this optimal policy in that they sometimes come very close to the destination, only to circle the block one or two times before deciding to go to the destination in order to get rewarded.