Basic Driving Agent

When actions are chosen purely by random and the agent is given unlimited time to reach the destination, it does eventually reach it. However, this is purely a matter of chance.

Indentifying and Updating State

Relevant state variables:

- Direction of next waypoint: `[None, 'forward', 'left', 'right']`
- Light status: `['red', 'green']`
- Oncoming traffic: `[None, 'forward', 'left', 'right']`
- Traffic to the left: `[None, 'forward', 'left', 'right']`

Valid actions: `[None, 'forward', 'left', 'right']`

In this simplistic model, the smartcab cares about two things: first, the location of the next waypoint; and second, its ability (right of way) to execute its valid actions.

Of the provided inputs, I've decided to ignore two:

- Deadline: Having the cab's behavior depend on the remaining time seems to me to be undesirable. Also, since there are so many possible values, including this variable would make the state space enormous.
- Traffic to the right: This has no affect on the cab's ability to execute its chosen action:
  - If traffic to the right is empty, there is obviously no impact.
  - If traffic to the right is proceeding straight or turning left, its light must be green and therefore the light for the cab must be red. That means the cab can only turn right, which traffic on the right can not impair.
  - If traffic to the right is turning right, the cab has the right of way to proceed straight. It clearly does not affect the cab's ability to turn left or right.

Implementing Q-Learning

My Q-learning model is initialized uniformly with a value of 1 for all state/action pairs. The learning rate is `1/t` and the discount factor is 0.5. It has a 10% chance to return a random action other than the one with the highest Q-value.

The agent now consistently learns to reach the destination in time. For the first several trials it often fails to do so, but by trial 50 this trend has shifted. By trial 100 it almost always reaches the destination in time and rarely receives any penalties.

The agent was previously choosing random actions at each step. It is now using the reward received after taking an action to update the Q-value of that state/action pair. After enough trials, the Q-values should converge to reveal a reasonable policy. This is consistent with the behavior observed above.

Optimizing the Agent

I identified the following parameters of the agent for tuning:

- The chance of randomly taking an action other than the one with the best Q-value.

- The power of the learning rate (1/t^(power))

The performance of each parameter combination was measured in the last 25 trials of three sets of 100 trials each. The following metrics were gathered:

- Percentage of trials in which the cab reached the destination in time
- Average time to reach destination (of successful trials)
- Average net reward (of successful trials)
- Average penalties received (of successful trials)

The results are shown below:

	20% Randomness				10% Randomness				5% Randomness			
	Success	Time	Rwrd	Pnlts	Success	Time	Rwrd	Pnlts	Success	Time	Rwrd	Pnlts
1/t^(0.6)	81.3%	16.1	30.4	1.2	89.3%	13.6	28.8	0.5	86.7%	14.3	30.4	0.3
1/t^(0.8)	80.0%	16.1	30.0	1.4	96.0%	15.0	31.0	0.5	84.0%	14.8	31.1	0.5
1/t	78.7%	15.1	29.3	1.1	86.7%	14.1	30.0	0.6	50.7%	16.0	32.8	0.3

Based on these result, I believe a randomness of 10% and learning rate of 1/t^0.8 to be the best choice. The cab reaches the destination 96% of the time. It takes slightly longer, on average, than the other learning rates at 10% randomness, but I think consistent on-time delivery is more important. In addition, the cab receives less than one penalty per trip on average, many of which are probably due to the cab taking its random action.

Q-values produced after 100 trials using these parameter values are listed below. The states are listed as tuples of waypoint, light status, oncoming traffic status, left traffic status.

```
(forward, green, forward, none):    forward:  3.00, left:  1,    right: 1,    none: 1
(forward, green, left,    none):    forward:  2.08, left:  1,    right: 1.06, none: 1
(forward, green, right,   none):    forward:  1,    left:  1,    right: 1.64, none: 1
(forward, green, none,    forward): forward:  2.32, left:  1,    right: 1,    none: 1
(forward, green, none,    left):    forward:  1.25, left:  1,    right: 1,    none: 1
(forward, green, none,    right):   forward:  5.13, left:  1,    right: 1,    none: 1
(forward, green, none,    none):    forward:  5.44, left:  2.40, right: 1.92, none: 2.86
(forward, red,   forward, none):    forward:  0.68, left:  0.72, right: 1,    none: 1
(forward, red,   left,    none):    forward:  0.38, left:  0.14, right: 1.94, none: 1
(forward, red,   right,   none):    forward:  0.59, left:  0.64, right: 1,    none: 1
(forward, red,   none,    forward): forward:  0.01, left:  1,    right: 1,    none: 1.16
(forward, red,   none,    left):    forward:  0.68, left:  0.84, right: 1,    none: 1
(forward, red,   none,    right):   forward:  0.84, left:  1,    right: 1,    none: 1
(forward, red,   none,    none):    forward:  0.06, left:  0.02, right: 1.96, none: 2.00

(left,    green, left,    none):    forward:  1.20, left:  1,    right: 1,    none: 1
(left,    green, none,    none):    forward:  1.14, left:  5.63, right: 1.52, none: 1
(left,    red,   forward, none):    forward:  1,    left:  1,    right: 1.15, none: 1
(left,    red,   left,    none):    forward: -0.5,  left:  0.14, right: 1.95, none: 1
(left,    red,   none,    none):    forward: -0.43, left:  0.23, right: 1.83, none: 1.33

(right,   green, left,    forward): forward:  1.16, left:  1,    right: 1,    none: 1
(right,   green, left,    none):    forward:  1,    left:  1.22, right: 1,    none: 1
(right,   green, none,    left):    forward:  1.14, left:  1,    right: 1,    none: 1
(right,   green, none,    right):   forward:  1.11, left:  1,    right: 1,    none: 1
(right,   green, none,    none):    forward:  2.59, left:  1.96, right: 3.47, none: 2.69
(right,   red,   forward, none):    forward: -0.50, left: -0.13, right: 2.97, none: 1
(right,   red,   left,    forward): forward:  0.59, left:  1,    right: 1,    none: 1
(right,   red,   left,    none):    forward:  0.38, left:  0.24, right: 2.38, none: 1
(right,   red,   right,   none):    forward:  0.68, left:  0.51, right: 1,    none: 1
(right,   red,   none,    left):    forward:  1,    left:  1,    right: 1,    none: 1
(right,   red,   none,    right):   forward:  1,    left:  1,    right: 1,    none: 1
(right,   red,   none,    none):    forward:  0.68, left:  0.88, right: 3.31, none: 2.97
```

Looking at the above Q-values, two things stand out to me:

- We are still very far from visiting all state/action pairs, so these Q-values should not be considered to have converged.
- Though the Q-values seem reasonable in many cases, I see several states where the action with the highest Q-value doesn't agree with common sense. For example `('right', 'green', None, 'right')` and `('right', 'green', None, 'left')` both show `forward` as the best action even though the waypoint is to the right and the way is clear to turn right.

In other words, the policy given by these Q-values is far from optimal. However, at 96% on-time delivery, it seems reasonably feasible.