MLND P4: Train a SmartCab to Drive

Implement a basic driving agent:

The agent chooses actions uniformly at random based on the allowed actions: (None, ‘right’, ‘left’, ‘forward’). The observed behavior of the agent is that it moves around in a random manner. This is the expected behavior. If given long enough, the agent eventually finds the target location.

To run the code in this mode, uncomment line 66:
#epsilon = 1.0

Identify and update state:

The code in environment.py sets up the following reward/penalty system based on states and actions.

Penalty Reward System	
Allowable move reaching waypoint	2
Disallowed move	-1
Allowed move not reaching waypoint	0.5
Allowed move reaching goal	10
No move	1

Allowable moves include None, forward, left and right on green as well as None and right on red. Disallowed moves include forward and left on red. Thus the reward/penalties are based on the state of the light, the location of the next waypoint and the location of the goal. The goal location is not directly available however the waypoints inevitably guide the cab to the goal. Thus the state chosen was the next waypoint and the light status.

self.state_ = (self.next_waypoint, inputs[‘light’])

Traffic:

The project directions talk about US right of way rules for encounters with other agents. There are two approaches to dealing with traffic - the explicit way and the machine learning way. Checks could be implemented to look for traffic and avoid it. This could also be done to allow the agent to reach the goal and is not the purpose of the exercise. I expected a penalty to be assigned to right of way violations (traffic collisions) in environment.py. In this case the state would have been extended to include oncoming and left inputs (traffic) as follows:

self.state_ = (self.next_waypoint, inputs[‘light’], inputs[‘oncoming’], inputs[‘left’])

However there were no penalties there. Furthermore there was the following discussion on the forum. https://discussions.udacity.com/t/right-of-way-does-it-matter/45191

Implement Q-Learning:

The Q-Learning algorithm has been implemented. Instead of choosing a random action, the agent's next action is picked from the best action available from the current state based on Q-values.

The agent's behavior depends on the ordering of the self.potential_actions tuple. If the first action is None, the agent does not move. This is because my algorithm initializes the Q values to zero and the max method returns the first zero value. The None action always incurs a positive reward resulting in always being the maximum value of any state/action pair. Thus the agent doesn't move.

A similar behavior occurs if the first action is 'right', namely 'right' is the only action learned and the agent goes in circles. Again, this is because the algorithm initializes the Q values to zero and the max method returns the first zero value. The 'right' action always incurs a positive reward resulting in always being the maximum value of any state/action pair. Thus the agent goes in 'right' circles.

If the first action is 'left' or 'forward', the agent learns a policy based on two actions and ignores the other two actions. Those two actions are the first action ('forward' or 'left') and the first action that always gives positive reward ('right' or None). The table below summarizes some example policies learned.

Learned Policy Sets	
'foward', 'left', 'right', None	'forward', 'right'
'forward, 'left', 'None', 'right'	'forward', None
'left', 'forward', 'right', None	'left', 'right'
'left', 'forward', None, 'right'	'left', None

In the case that goal is reachable based on a set of actions in the learned policy subset, the agent may achieve the goal in the allotted time. However most often it doesn't.

To run the code in this mode, uncomment line 69:
#epsilon = 0.0

Enhance the driving agent:

The reason the first Q-Learning scheme failed to reliably learn to reach the goal is because it never fully explored the state/action space. In other words it tried to exploit the best action available right away without learning the Q of the other state/actions. To fix this, an epsilon scheme was introduced to the action choice. The best action available was chosen with a probability of epsilon and a random action was chosen with a probability of 1-epsilon. Epsilon was chosen to decay from trial=0 to trial = 99 by the formula epsilon = 1/(trial+1). Thus for the trial = 0 the agent chose actions completely randomly. For trial=1 the agent chose random actions 50% of the time and the best action 50% of the time. By trial=99 the agent chose the best action 99% of the time and a random action 1% of the time. This scheme allowed the agent to learn a policy that gets it to the goal within the deadline every time by trial=99. By trial=99, all rewards gained are positive, thus the net reward is positive.

As far as whether or not the agent learned an optimal policy, it is difficult to tell. The agent learns to obey traffic lights and since the other agents are essentially ignored (or could be explicitly accounted for) there are no accidents.

Appendix A shows the Q tables for some gammas and alphas that worked well. The values seem to be converging from trial to trial. However as we saw in the previous section they could be converging to a bad policy. The policy that the agent learns for each column is highlighted in green. These choices seem reasonable from using a human driver heuristic - i.e. that is likely how I would drive.

One state that is interesting is (‘left’, ‘red’) which corresponds to the next waypoint being left while the light is red. Out of the 4 learners shown in Appendix A, 3 learned a policy to turn right while 1 learned None. This is contrary to what I would expect as right seems to take the learner further away from the waypoint than None. However it is reasonable as it doesn’t incur any penalties and allows the possibility of reaching other lights that may be green. Perhaps those agents are simply impatient and would prefer a detour rather than waiting.

Appendix A - ((next_waypoint, light), action) : Q(s,a)

alpha = 0.5, gamma = 0.7		alpha = 0.3, gamma = 0.7	
Trial = 100	**Trial = 99**	**Trial = 100**	**Trial = 99**
(('right', 'red'), 'forward') : 3.72905560238	(('right', 'red'), 'forward') : 3.72905560238	(('right', 'red'), 'forward') : 3.63002208183	(('right', 'red'), 'forward') : 3.63002208183
(('right', 'green'), 'forward') : 5.22353462296	(('right', 'green'), 'forward') : 5.22353462296	(('right', 'green'), 'forward') : 4.83166995006	(('right', 'green'), 'forward') : 4.83166995006
(('left', 'red'), 'left') : 1.75393117115	(('left', 'red'), 'left') : 1.75393117115	(('left', 'red'), 'left') : 1.6605212159	(('left', 'red'), 'left') : 1.6605212159
(('left', 'green'), 'right') : 4.70049239377	(('left', 'green'), 'right') : 4.70049239377	(('left', 'green'), 'right') : 4.74707224138	(('left', 'green'), 'right') : 4.58796379106
(('left', 'green'), 'forward') : 3.27407725913	(('left', 'green'), 'forward') : 3.27407725913	(('left', 'green'), 'forward') : 2.54992776203	(('left', 'green'), 'forward') : 2.54992776203
(('left', 'red'), None) : 4.50228756458	(('left', 'red'), None) : 4.50228756458	(('left', 'red'), None) : 2.15056526862	(('left', 'red'), None) : 2.15056526862
(('right', 'red'), 'right') : 6.54710000674	(('right', 'red'), 'right') : 6.2591996963	(('right', 'red'), 'right') : 5.77860018863	(('right', 'red'), 'right') : 5.79986180334
(('forward', 'green'), 'right') : 4.97076282555	(('forward', 'green'), 'right') : 4.97076282555	(('forward', 'green'), 'right') : 3.64261433437	(('forward', 'green'), 'right') : 3.64261433437
(('left', 'green'), 'left') : 5.60466891831	(('left', 'green'), 'left') : 5.69939230042	(('left', 'green'), 'left') : 4.24257810337	(('left', 'green'), 'left') : 4.24257810337
(('right', 'red'), 'left') : 3.61049564369	(('right', 'red'), 'left') : 3.61049564369	(('right', 'red'), 'left') : 3.31935755816	(('right', 'red'), 'left') : 3.31935755816
(('left', 'red'), 'right') : 4.82104813242	(('left', 'red'), 'right') : 5.321949319	(('left', 'red'), 'right') : 4.15290291393	(('left', 'red'), 'right') : 4.55695826612
(('right', 'green'), 'left') : 4.60502320954	(('right', 'green'), 'left') : 4.60502320954	(('right', 'green'), 'left') : 4.54293835317	(('right', 'green'), 'left') : 4.54293835317
(('forward', 'red'), 'left') : 2.37060491765	(('forward', 'red'), 'left') : 2.37060491765	(('forward', 'red'), 'left') : 1.92647353396	(('forward', 'red'), 'left') : 1.92647353396
(('forward', 'red'), 'right') : 4.19955730206	(('forward', 'red'), 'right') : 4.81492438318	(('forward', 'red'), 'right') : 4.23358343231	(('forward', 'red'), 'right') : 3.9401559487
(('forward', 'green'), None) : 5.13561504653	(('forward', 'green'), None) : 5.13561504653	(('forward', 'green'), None) : 5.24250370322	(('forward', 'green'), None) : 5.24250370322
(('forward', 'red'), None) : 3.97576045157	(('forward', 'red'), None) : 4.08912994303	(('forward', 'red'), None) : 3.7282046026	(('forward', 'red'), None) : 3.7282046026
(('forward', 'green'), 'left') : 4.94861788146	(('forward', 'green'), 'left') : 4.94861788146	(('forward', 'green'), 'left') : 4.53594336475	(('forward', 'green'), 'left') : 4.53594336475
(('left', 'red'), 'forward') : 2.45365441385	(('left', 'red'), 'forward') : 2.45365441385	(('left', 'red'), 'forward') : 1.00746878991	(('left', 'red'), 'forward') : 1.00746878991
(('forward', 'red'), 'forward') : 2.75549849989	(('forward', 'red'), 'forward') : 2.75549849989	(('forward', 'red'), 'forward') : 1.65134553685	(('forward', 'red'), 'forward') : 1.65134553685
(('right', 'red'), None) : 5.66436973277	(('right', 'red'), None) : 5.66436973277	(('right', 'red'), None) : 5.2291970819	(('right', 'red'), None) : 5.2291970819
(('left', 'green'), None) : 4.78922744407	(('left', 'green'), None) : 4.78922744407	(('left', 'green'), None) : 4.08280508599	(('left', 'green'), None) : 4.08280508599
(('right', 'green'), None) : 5.22985121729	(('right', 'green'), None) : 5.22985121729	(('right', 'green'), None) : 5.32540568419	(('right', 'green'), None) : 5.32540568419
(('forward', 'green'), 'forward') : 8.94617344748	(('forward', 'green'), 'forward') : 8.78349652193	(('forward', 'green'), 'forward') : 8.16994646623	(('forward', 'green'), 'forward') : 7.1690244129
(('right', 'green'), 'right') : 5.88023275525	(('right', 'green'), 'right') : 6.0865386143	(('right', 'green'), 'right') : 6.18154110408	(('right', 'green'), 'right') : 5.91461642624

alpha = 0.5, gamma = 0.9		alpha = 0.3, gamma = 0.9	
Trial = 100	**Trial = 99**	**Trial = 100**	**Trial = 99**
(('right', 'red'), 'forward') : 7.63323892136	(('right', 'red'), 'forward') : 7.63323892136	(('right', 'red'), 'forward') : 4.93404024103	(('right', 'red'), 'forward') : 4.93404024103
(('right', 'green'), 'forward') : 11.117183466	(('right', 'green'), 'forward') : 11.117183466	(('right', 'green'), 'forward') : 2.12559227575	(('right', 'green'), 'forward') : 2.12559227575
(('left', 'red'), 'left') : -0.5	(('left', 'red'), 'left') : -0.5	(('left', 'red'), 'left') : -0.313483680909	(('left', 'red'), 'left') : -0.313483680909
(('left', 'green'), 'forward') : 4.72787432226	(('left', 'green'), 'forward') : 4.72787432226	(('left', 'green'), 'right') : 0.39562154151	(('left', 'green'), 'right') : 0.39562154151
(('left', 'red'), None) : 1.8549375	(('left', 'red'), None) : 1.8549375	(('left', 'green'), 'forward') : 3.55010071923	(('left', 'green'), 'forward') : 3.55010071923
(('right', 'red'), 'right') : 12.524201235	(('right', 'red'), 'right') : 11.0984244198	(('left', 'red'), None) : 9.3932642956	(('left', 'red'), None) : 9.3932642956
(('forward', 'green'), 'right') : 7.08392803322	(('forward', 'green'), 'right') : 7.08392803322	(('right', 'red'), 'right') : 11.6207335546	(('right', 'red'), 'right') : 11.8867626206
(('left', 'green'), 'left') : 12.8911191039	(('left', 'green'), 'left') : 12.8911191039	(('forward', 'green'), 'right') : 4.61127850281	(('forward', 'green'), 'right') : 4.61127850281
(('right', 'red'), 'left') : 7.96745931895	(('right', 'red'), 'left') : 7.96745931895	(('left', 'green'), 'left') : 11.9517401565	(('left', 'green'), 'left') : 11.9517401565
(('left', 'red'), 'right') : 11.0781844069	(('left', 'red'), 'right') : 11.0781844069	(('right', 'red'), 'left') : 1.50177930293	(('right', 'red'), 'left') : 1.50177930293
(('right', 'green'), 'left') : 9.06560410889	(('right', 'green'), 'left') : 9.06560410889	(('left', 'red'), 'right') : 2.56161350136	(('left', 'red'), 'right') : 2.56161350136
(('forward', 'red'), 'left') : 2.105197572	(('forward', 'red'), 'left') : 2.105197572	(('right', 'green'), 'left') : 5.76877487689	(('right', 'green'), 'left') : 5.76877487689
(('forward', 'red'), 'right') : 3.38807761876	(('forward', 'red'), 'right') : 3.38807761876	(('forward', 'red'), 'left') : 5.06650679476	(('forward', 'red'), 'left') : 5.06650679476
(('forward', 'green'), None) : 5.73928858111	(('forward', 'green'), None) : 5.73928858111	(('forward', 'red'), 'right') : 5.86991086034	(('forward', 'red'), 'right') : 5.86991086034
(('forward', 'red'), None) : 9.99999901969	(('forward', 'red'), None) : 9.99999885661	(('forward', 'green'), None) : 8.02441040453	(('forward', 'green'), None) : 8.02441040453
(('forward', 'green'), 'left') : 2.37129663495	(('forward', 'green'), 'left') : 2.37129663495	(('forward', 'red'), None) : 9.99999916271	(('forward', 'red'), None) : 9.99999896372
(('left', 'red'), 'forward') : 3.78471213221	(('left', 'red'), 'forward') : 3.78471213221	(('forward', 'green'), 'left') : 4.51812423396	(('forward', 'green'), 'left') : 4.51812423396
(('forward', 'red'), 'forward') : 4.75011131117	(('forward', 'red'), 'forward') : 4.75011131117	(('left', 'red'), 'forward') : 0.157772175792	(('left', 'red'), 'forward') : 0.157772175792
(('right', 'red'), None) : 10.0	(('right', 'red'), None) : 10.0	(('forward', 'red'), 'forward') : 7.08055387388	(('forward', 'red'), 'forward') : 7.08055387388
(('right', 'green'), None) : 10.0	(('right', 'green'), None) : 10.0	(('right', 'red'), None) : 5.84220959962	(('right', 'red'), None) : 5.84220959962
(('forward', 'green'), 'forward') : 11.8799324486	(('forward', 'green'), 'forward') : 11.9182706851	(('left', 'green'), None) : 0.87327	(('left', 'green'), None) : 0.87327
(('right', 'green'), 'right') : 12.4861987478	(('right', 'green'), 'right') : 12.4861987478	(('right', 'green'), None) : 2.59197150273	(('right', 'green'), None) : 2.59197150273
(('right', 'red'), 'forward') : 7.63323892136	(('right', 'red'), 'forward') : 7.63323892136	(('forward', 'green'), 'forward') : 12.5252469024	(('forward', 'green'), 'forward') : 12.4524441639
(('right', 'green'), 'forward') : 11.117183466	(('right', 'green'), 'forward') : 11.117183466	(('right', 'green'), 'right') : 12.2622738871	(('right', 'green'), 'right') : 12.2622738871