Project Report

You will be required to submit a project report along with your modified agent code as part of your submission. As you complete the tasks below, include thorough, detailed answers to each question provided in italics.

Implement a Basic Driving Agent

To begin, your only task is to get the smartcab to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection:

- The next waypoint location relative to its current location and heading.
- The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions.
- The current time left from the allotted deadline.

To complete this task, simply have your driving agent choose a random action from the set of possible actions (None, 'forward', 'left', 'right') at each intersection, disregarding the input information above. Set the simulation deadline enforcement, enforce_deadline to False and observe how it performs.

QUESTION: Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

ANSWER: Yes, the smart cab make it to the destination 70% or 80% of the time. For the simulation of 100 trials, failed trial (agent deadline hit hard time limit) range from 20 to 35. It interests me when the random cab turned left four times in a row. For the environment here, it seems when enforce_deadline was set to false, the learning agent get an extra of 100 steps.

Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the smartcab and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the self.state variable. Continue with the simulation deadline enforcementenforce_deadline being set to False, and observe how your driving agent now reports the change in state as the simulation progresses.

QUESTION: What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem?

ANSWER:
SCab: inputs['light'], inputs['oncoming'], inputs['left'], inputs['right'], next_waypoint.
Env: 'light: red/green', 'oncoming', 'left', 'right'

These states seems appropriate. Since they somewhat all related to rewards, which will affect the Q value. Invalid move's reward is -1. Ex. if car decide to go forward when red light. If car's next action is next_waypoint, reward is 2 otherwise reward is -0.5.
By choosing these states, our learning agent is learning to obey traffic rules and to follow traffic planner. As long as our learning agent learn to follow traffic planner, it should be able to meet deadline. Therefore, deadline is not included.
Below is the note from reviewer, keep them for record.
As if we were to include the deadline into our current state, our state space would blow up, we would suffer from the curse of dimensionality and it would take a long time for the q-matrix to converge. Also note that including the deadline could possibly influence the agent in making illegal moves when the deadline is near.

OPTIONAL: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

ANSWER: In order to get the number of states, we could multiply the states of each variable. Therefore, 2*4*4*4*3 = 384 (No 'None' values in next waypoint, therefore 3 instead of 4)
For a simulation of 100 runs, given each run cover 15 intersections. We would have 1500 data points. When Q values were initialized with a small number, our algorithm may require a large learning rate. Those data points may not seem sufficient. However, consider in this problem we may not need to train all the possible route. Our learning agent may still be able to find the best possible route.

Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by the smartcab will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement enforce_deadline to True. Run the simulation and observe how the smartcab moves about the environment in each trial.
The formulas for updating Q-values can be found in this video.

QUESTION: What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

ANSWER: I notice there are less steps taken and more actions are equal to next waypoint. This behavior is due to the best action derived from Q-Learning algorithm.
The agent is behaving more randomly in the beginning. It run the light, turning right when there is traffic on the left, etc.. Our agent is more likely to violate traffic rules when seeing new state. It makes sense since the agent need to learn before it could make better choices. However, it's not as bad as running in circles.
Let me try to define 'stuck in local minimal', in our case it would mean that our cab always execute according to the learned policy. It never takes risk therefore missing the pot of gold around the corner. This could happen if we set a fix small learning rate. It could be solved using epsilon-decay, allow the agent to explore when knowledge is weak, while exploit more as we gain knowledge.

Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the smartcab is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your smartcab:

- Set the number of trials, n_trials, in the simulation to 100.
- Run the simulation with the deadline enforcement enforce_deadline set to True (you will need to reduce the update delay update_delay and set the display to False).
- Observe the driving agent's learning and smartcab's success rate, particularly during the later trials.
- Adjust one or several of the above parameters and iterate this process.

This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

QUESTION: Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

ANSWER:
Avoid hard coding, keep states as following:
inputs['light'], input['oncoming'], inputs['left'], inputs['right'], self.next_waypoint.

In the beginning, I am exploring fixed alpha, gamma values to solve this problem. For each set of the parameters, I ran 4 rounds of simulation as marked r1, r2, r3, r4. For tuning, I am looking

at total moves taken, penalties, mean penalty rate, failures. Mean P rate is calculated as sum(penalties(ri)/moves(ri))/4 rounds.

Alpha is learning rate (wikipedia):
The learning rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. If the discount factor meets or exceeds 1, the action values may diverge.

Gamma is discount factor (wikipedia):
The discount factor gamma determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward.

	moves(r1/r2/r3/r4)	penalties(r1/r2/r3/r4)	Mean P rate	failures(r1/r2/r3/r4)
Alpha 0.9 gamma 0.1	1297/1340/1319/1358	36/39/41/40	0.0294	2/1/0/1
Alpha 0.9 gamma 0.3	1331/1374/1302/1360	35/36/39/40	0.0280	1/0/1/1
Alpha 0.9 gamma 0.5	1252/1369/1422/1360	32/36/45/40	0.0282	2/0/5/1
Alpha 0.8 gamma 0.1	1302/1359/1222/1225	42/41/32/36	0.0295	2/4/2/2
Alpha 0.8 gamma 0.3	1322/1357/1405/1340	39/39/45/42	0.0304	0/1/1/0
Alpha 0.8 gamma 0.5	1273/1313/1371/1393	41/35/43/38	0.0294	1/2/0/1

The best parameters in this case seems to be: learning rate 0.9, discount_factor 0.3.

It came to me after my second submission, that this question is about ‘Exploitation vs. Exploration’. Now it came back to me that Q learning is about balancing the two. I decide to implement epsilon decay. Where in the beginning, epsilon is close to 1. Our learning cab will make random choices therefore explore more states. As epsilon decays to 0, our learning cab will make best choices according to Q function. Smart agent’s decision is epsilon greedy in this submit now. Consider for 100 trials, there are around 1300 moves, I picked epsilon = 0.05. That

is about 65 random moves. While it allows the smart agent to explore, when we add random choices penalties and aborted trial may increase.
This stack overflow (http://stackoverflow.com/questions/22805872/optimal-epsilon-%CF%B5-greedy-value) suggest setting the decay factor as 1/t. However, I found it decreasing too fast. I took the suggestion from udacity forum and set decay factor to 1/ln(t+5).

Now, tuning gamma only.

	moves(r1/r2/r3/r4)	penalties(r1/r2/r3/r4)	Mean P rate	failures(r1/r2/r3/r4)
gamma= 0.1	1318/1361/1423/1256	42/33/37/38	0.02810	2/2/1/1
gamma=0.2	1256/1424/1514/1345	38/50/51/42	0.03255	1/3/2/2
gamma=0.3	1509/1326/1299/1449	55/58/40/48	0.03598	2/2/1/2
gamma=0.4	1429/1339/**1960**/1456	67/53/**557**/67	**0.4165**	5/0/19/3
gamma=0.5	1442/1416/1351/1382	48/87/69/41	0.0440	2/2/1/2
gamma=0.6	1303/1640/1319/1308	39/87/69/61	0.04545	1/9/2/0
gamma=0.7	1435/1364/1354/1388	56/106/38/42	0.04378	2/1/1/1

It seems here the best gamma value is 0.1. It's worth noting that in one of simulation, our agent went crazy with way more penalties. This is probably due to error propagation as wikipedia states:
Even with a discount factor only slightly lower than 1, the *Q*-function learning leads to propagation of errors and instabilities when the value function is approximated with an artificial neural network.

Below is from the reviewer, keep them here for record.
Another interesting idea would be to implement an e-greedy implementation. Therefore we can randomly explore the with with probability of some random action. As a very simple exploration strategy is e-greedy exploration (generally called "epsilon greedy")

- Select a “small” value for e (perhaps 0.1)
- On each step:
- With probability e select a random action, and
- with probability 1- e select a greedy action

Check out this lecture. A more advanced version of this would be epsilon decay. Therefore we can reduce the chances of random exploration over time, as we can get the best of both exploration vs exploitation. As this typically works very well. We typically see `1 / t` for the decay factor.

The driving agent in later simulation trials either wait or takes the next waypoint as it action. The -0.5 reward never showed up in later trials.

QUESTION: 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? How would you describe an optimal policy for this problem?

ANSWER:
Yes, my agent is close to finding an optimal policy. The agent follow shortest path to get +2 reward. The agent obey traffic rules to avoid -1 rewards. In the later simulation trials, the smart cab almost always following the planner. Penalties rarely appear.
Looking backwards at the report, from trial 99 - trial 0, here is the summary of this run.
Some observations: (the state below is different from before since it does not include next_waypoint.)

When doing search within all penalties, the state in trial 86&82&68 never shows up before. A -0.5 reward means the agent is not following the route planner. Therefore, the agent may never learned to act in this situation before.
It is interesting in trial 45, the state appeared twice in 45, once in 34, twice in 24. For these 5 penalties. 4 of them are traffic violations (-1 redward) and 1 of them is not following route planner (-0.5). This state says: red light, oncoming traffic is turning left, no traffic on left or right. The correct action should be ‘None’ or ‘right’. Depending on the appearance order of these penalties, I think our agent first learned to follow route planner in this state. Possibly pick up some +2 reward when the route planner’s next waypoint is ‘right’. I assume in the later trials, when the route planner’s next waypoint is ‘forward’ or ‘left’, our agent execute and violated traffic rules. As this bring Q values down, our agent may learn the best action shall be wait in this state when next waypoint is not ‘right’.

total reward is:2300.5; total moves is:1442; total penalties is:35; penalty rate is:0.0242718446602; total fail is:1
Simulator.run(): Trial 86
LearningAgent.update(): deadline = 26, inputs = {'light': 'green', 'oncoming': None, 'right': 'right', 'left': None}, action = right, reward = -0.5

Simulator.run(): Trial 82
LearningAgent.update(): deadline = 15, inputs = {'light': 'green', 'oncoming': None, 'right': 'right', 'left': None}, action = right, reward = -0.5
Simulator.run(): Trial 68
LearningAgent.update(): deadline = 26, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': 'forward'}, action = right, reward = -1.0
Simulator.run(): Trial 65
LearningAgent.update(): deadline = 27, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': 'right'}, action = right, reward = -0.5
Simulator.run(): Trial 54
LearningAgent.update(): deadline = 37, inputs = {'light': 'green', 'oncoming': 'right', 'right': None, 'left': None}, action = left, reward = -1.0
Simulator.run(): Trial 45
LearningAgent.update(): deadline = 38, inputs = {'light': 'red', 'oncoming': 'left', 'right': None, 'left': None}, action = forward, reward = -1.0
LearningAgent.update(): deadline = 37, inputs = {'light': 'red', 'oncoming': 'left', 'right': None, 'left': None}, action = left, reward = -1.0
Simulator.run(): Trial 39
LearningAgent.update(): deadline = 30, inputs = {'light': 'green', 'oncoming': None, 'right': 'forward', 'left': None}, action = left, reward = -0.5
Simulator.run(): Trial 36
LearningAgent.update(): deadline = 9, inputs = {'light': 'red', 'oncoming': 'forward', 'right': None, 'left': None}, action = forward, reward = -1.0
Simulator.run(): Trial 34
LearningAgent.update(): deadline = 43, inputs = {'light': 'red', 'oncoming': 'left', 'right': None, 'left': None}, action = left, reward = -1.0
LearningAgent.update(): deadline = 41, inputs = {'light': 'green', 'oncoming': 'left', 'right': None, 'left': None}, action = left, reward = -0.5
Simulator.run(): Trial 29
LearningAgent.update(): deadline = 30, inputs = {'light': 'red', 'oncoming': 'forward', 'right': None, 'left': None}, action = right, reward = -0.5
Simulator.run(): Trial 24
LearningAgent.update(): deadline = 25, inputs = {'light': 'red', 'oncoming': 'left', 'right': None, 'left': 'forward'}, action = right, reward = -1.0
LearningAgent.update(): deadline = 23, inputs = {'light': 'red', 'oncoming': 'left', 'right': None, 'left': None}, action = forward, reward = -1.0
LearningAgent.update(): deadline = 22, inputs = {'light': 'red', 'oncoming': 'left', 'right': None, 'left': None}, action = right, reward = -0.5
Simulator.run(): Trial 20
LearningAgent.update(): deadline = 45, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': 'right'}, action = forward, reward = -1.0
Simulator.run(): Trial 9
LearningAgent.update(): deadline = 31, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': 'right'}, action = right, reward = -0.5
LearningAgent.update(): deadline = 28, inputs = {'light': 'green', 'oncoming': None, 'right': 'left', 'left': None}, action = left, reward = -0.5
LearningAgent.update(): deadline = 11, inputs = {'light': 'green', 'oncoming': 'right', 'right': None, 'left': None}, action = right, reward = -0.5
LearningAgent.update(): deadline = 0, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': 'left'}, action = right, reward = -0.5
Simulator.run(): Trial 8
LearningAgent.update(): deadline = 24, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': 'left'}, action = left, reward = -1.0
Simulator.run(): Trial 6

LearningAgent.update(): deadline = 30, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': 'left'}, action = left, reward = -0.5
LearningAgent.update(): deadline = 27, inputs = {'light': 'red', 'oncoming': None, 'right': 'right', 'left': None}, action = right, reward = -0.5
LearningAgent.update(): deadline = 21, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = left, reward = -1.0
Simulator.run(): Trial 5
LearningAgent.update(): deadline = 27, inputs = {'light': 'green', 'oncoming': None, 'right': 'left', 'left': None}, action = right, reward = -0.5
LearningAgent.update(): deadline = 24, inputs = {'light': 'green', 'oncoming': None, 'right': 'left', 'left': None}, action = forward, reward = -0.5
Simulator.run(): Trial 2
LearningAgent.update(): deadline = 12, inputs = {'light': 'red', 'oncoming': 'forward', 'right': 'right', 'left': None}, action = forward, reward = -1.0
LearningAgent.update(): deadline = 11, inputs = {'light': 'red', 'oncoming': 'forward', 'right': None, 'left': None}, action = left, reward = -1.0
Simulator.run(): Trial 1
LearningAgent.update(): deadline = 24, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = -0.5
LearningAgent.update(): deadline = 23, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = -1.0
LearningAgent.update(): deadline = 17, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = -1.0
Simulator.run(): Trial 0
LearningAgent.update(): deadline = 35, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = -0.5
LearningAgent.update(): deadline = 33, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = -0.5
LearningAgent.update(): deadline = 32, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = -0.5
LearningAgent.update(): deadline = 29, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = left, reward = -1.0

For this problem, the optimal policy is using least steps to get from start to destination without violating traffic rules. Follow planner's next waypoint to get maximum rewards. The following values from Q learning table shows this policy.
64 out of 384 states were visit, therefore its corresponding Q value was calculated. From this number it doesn't seem we are close to the optimum policy. Even though the smart agent was able to find the destination in last 15 or so trials without encountering penalties.

I categorize the Q table in terms of states and numbered each category. I could see for some categories, those only have on state in it, the agent has not learned enough. Some of them might be all right, since the Q value is positive. For those have negative values, it maybe possible a initial Q value of 0 is determined to be a better choice, which ends up a penalty. Eg, in category 3, 11, 17, 18 etc.

1('green', None, None, 'forward', 'forward'), None): 0.0,
(('green', None, None, 'forward', 'forward'), 'forward'): 0.32456330758907215,
(('green', None, None, 'forward', 'forward'), 'left'): -0.044151280474657974

2(('green', 'forward', None, 'left', 'forward'), 'forward'): 0.29508767864618357,
3(('red', 'forward', None, 'right', 'forward'), 'forward'): -0.265872585674852,
4(('red', 'left', None, None, 'forward'), 'left'): -0.16000063380066787,
(('red', 'left', None, None, 'forward'), 'right'): -0.08314327949131106,
(('red', 'left', None, None, 'forward'), None): 0.0,
(('red', 'left', None, None, 'forward'), 'forward'): -0.16635427622614699,
5(('green', None, 'forward', None, 'forward'), 'forward'): 2.2908980484137254,
6(('green', None, 'right', None, 'forward'), 'right'): -0.05928212471595255,
(('green', None, 'right', None, 'forward'), None): 0.0,
7(('green', None, None, 'left', 'forward'), None): 0.0,
(('green', None, None, 'left', 'forward'), 'right'): -0.0857803878761108,
8(('green', 'left', None, None, 'forward'), 'left'): -0.04320785929589858,
(('green', 'left', None, None, 'forward'), 'forward'): 0.34172850406095034,
9(('red', None, None, None, 'forward'), None): 0.0,
(('red', None, None, None, 'forward'), 'right'): -0.16690410034766703,
(('red', None, None, None, 'forward'), 'left'): -0.40242960438184466,
(('red', None, None, None, 'forward'), 'forward'): -0.30341307554227914,
10(('green', 'left', None, None, 'right'), 'right'): 0.7399211590329509,
11(('red', None, None, 'right', 'forward'), 'right'): -0.1111158108079052,
12(('red', 'right', None, None, 'right'), 'right'): 0.29382255586391987,
13(('red', 'left', None, None, 'right'), 'forward'): -0.15339429465719767,
(('red', 'left', None, None, 'right'), None): 0.0,
(('red', 'left', None, None, 'right'), 'left'): -0.15335962332999362,
14(('green', None, None, 'forward', 'left'), 'left'): 2.0792033989410363,
15(('green', None, None, None, 'right'), 'right'): 2.260645190720249,
(('green', None, None, None, 'right'), None): 0.0,
16(('green', None, None, None, 'left'), 'forward'): -0.2790553132756236,
(('green', None, None, None, 'left'), 'left'): 2.5158533717795897,
17(('green', 'right', None, None, 'forward'), 'right'): -0.057956961812039506,
18(('red', 'left', 'forward', None, 'forward'), 'right'): -0.16649037667746924,
19(('red', None, 'left', None, 'right'), 'right'): 0.3510347683064014,
(('red', None, 'left', None, 'right'), 'left'): -0.21075117050092493,
20(('red', None, 'left', None, 'forward'), 'right'): -0.05823786391622861,
21(('green', None, None, None, 'forward'), 'right'): -0.24044917348149392,
(('green', None, None, None, 'forward'), 'forward'): 4.364905565608436,
22(('red', None, 'right', None, 'left'), None): 0.0,
23(('red', None, None, None, 'left'), 'left'): -0.21908933281411486,
(('red', None, None, None, 'left'), 'right'): -0.22755980665670933,
(('red', None, None, None, 'left'), None): 0.0,
(('red', None, None, None, 'left'), 'forward'): -0.3284587387530511,
24(('red', None, None, 'forward', 'forward'), None): 0.0,
25(('red', None, None, None, 'right'), 'right'): 2.9412461758039083,
26(('green', None, None, 'right', 'forward'), 'right'): -0.06484815447237502,
27(('green', None, None, 'right', 'left'), None): 0.0,
28(('green', None, 'left', None, 'forward'), None): 0.0,
(('green', None, 'left', None, 'forward'), 'forward'): 0.6065702349709454,
29(('green', None, 'left', None, 'right'), 'left'): -0.07946500764193362,
(('green', None, 'left', None, 'right'), 'right'): 0.6655553298769166,
30(('green', None, None, 'left', 'right'), 'left'): -0.06353844891304551,
(('green', None, None, 'left', 'right'), 'forward'): -0.1134630939582562,
31(('green', 'right', None, None, 'left'), None): 0.0,
(('green', 'right', None, None, 'left'), 'left'): -0.14985094399051974,
32(('red', 'forward', None, None, 'forward'), 'left'): -0.26425736993644544,
(('red', 'forward', None, None, 'forward'), 'forward'): -0.15811879210971763,

(('red', 'forward', None, None, 'forward'), 'right'): -0.0122050581127148,
(('red', 'forward', None, None, 'forward'), None): 0.0
33(('red', None, 'right', None, 'forward'), 'right'): -0.036886799219313524,
(('red', None, 'right', None, 'forward'), 'forward'): -0.1712140649371239,
34(('red', 'left', None, None, 'left'), None): 0.0,
35(('green', 'forward', None, None, 'forward'), 'forward'): 2.4329995043860198,
36(('red', None, 'forward', None, 'right'), 'right'): -0.1447438839476537,