A) Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

- Next waypoint location, relative to its current location and heading,
- Intersection state (traffic light and presence of cars), and,
- Current deadline value (time steps remaining),

And produces some random move/action (None, 'forward', 'left', 'right'). Don't try to implement the correct strategy! That's exactly what your agent is supposed to learn.

Run this agent within the simulation environment with enforce_deadline set to False (see runfunction in agent.py), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

Q1 *- In your report, mention what you see in the agent's behaviour. Does it eventually make it to the target location?*

Filename: agent_code_for_**Q1**.py
For this task, the following lines of code were updated/added:
(line 29) options = ['forward', 'right', 'left', None]
(line 30) action = random.choice(options)

Observation and agent's behaviour:

- With the code change, the agent moves through the grid using random actions.
- Multiple rewards are given (-1, +0.5, +1, +2)
    - A negative reward of -1 is given when attempting to move left or forward on a red light
    - A reward of +0.5 is given when turning right on a red or green light
    - A reward of +1 is given when waiting at a green light
    - A reward of +2 is given when moving on a green light (forward, right, left)
- For the moment, it looks like there is no code preventing collisions with other cars.
- With this code, the agent will only haphazardly reach the destination. At the moment, there doesn't seem to be any motivation or reinforcement within the code to move towards the destination.

B) Identify and update state

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

Q2 *- Justify why you picked these set of states, and how they model the agent and its environment.*

Filename: agent_code_for_**Q2**.py
For this task, the following lines of code were updated/added:
(line 26) self.state = (self.next_waypoint, inputs)
(line 27) print "self.state:", self.state

Code Justification:
I chose to process two inputs for updating the current state: "next_waypoint" and "inputs". The "next_waypoint" state provides information on the agent's next step and the "inputs" state provides information on the agent's environment. Equipped with this information, additional code can added to select the most efficient path to the destination while avoiding accident's with other agent's.

I chose to exclude the "deadline" state to avoid rushing towards the destination and harming other agent's in the process.

C) Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that. Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behavior.

Q3 - What changes do you notice in the agent's behaviour?

[to be completed after Q1 & Q2 review]

D) Enhance the driving agent

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

Q4 - 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?

[to be completed after Q1 & Q2 review]

Q5 - 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?
The formulas for updating Q-values can be found in this video.

[to be completed after Q1 & Q2 review]