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 `run` function 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.

Question 1

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

Answer:
When letting the agent randomly choosing an action, the agent does not seem to move toward the destination. It walked randomly on the grid. The program was run several times. For trial 1, the agent reached the destination after 53 time steps. For trial 2, the agent reached the destination after 120 time steps. For trial 3, the agent reached the destination after 125 time steps. For trial 4, the agent still had not reached the destination after 200 time steps and the program was terminated. Overall, the agent either did not reach the destination or moved more steps than necessary to reach the destination.

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.

Question 2

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

Answer:
The available environment variables are: `deadline (number of timesteps remaining)`, `traffic light (Green, Red)`, `oncoming car (forward, right, left, None)`, `left-coming car (forward, right, left, None)`, `right-coming car (forward, right, left, None)`, `next waypoint (forward, right, left, None)`. A state is defined as a tuple of `(light, oncoming, left, next_waypoint)`. According to the US right-of-way rule, "On a green light, you can turn left only if there is no oncoming traffic at the intersection coming straight. On a red light, you can turn right if there is no oncoming traffic turning left or traffic from the left going straight." Therefore, the agent does not need to care about the actions of the right-coming car. The information on traffic light, oncoming car and left-coming car is sufficient in describing the agent's egocentric view of the current environment on which the agent can choose actions to avoid breaking the traffic rules. Next waypoint tells the direction heading to the destination hence facilitates the agent to make better decisions on choosing the next direction. The agent is expected to learn the optimal policy within a certain deadline which is considered in implementing Q-Learning with a discount rate and a decreasing leraning rate. Hence the count down to deadline is not recorded in the state.

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.

Question 3

What changes do you notice in the agent’s behavior?

Answer:
The agent acts like it is aware of the surrounding environment and breaks less traffic rules. It is also aware of its location relative to the destination and moves toward the destination in fewer actions. However in approximately half of the trials, the agent still cannot reach the destination within the deadline. The agent also tends to run in circle without breaking traffic rules to gain more positive rewards. To enhance the performance, the discount rate and leraning rate need to be further tuned.

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.

The formulas for updating Q-values can be found in this (https://www.udacity.com/course/viewer#!/c-ud728-nd/l-5446820041/m-634899057) video.

Question 4a

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?

Answer:
In the `def chooseAction` function, the best actions with highest Q values are found and listed. To enhance the agent's performance, `next_waypoint` is set to be chosen if it is among one of the best actions.
Learning rate alpha and exploration probability epsilon are defiend as `1/t` which approaches to 0 as learning proceeds. Discount rate gamma are tuned with three different values: 0.9, 0.5 and 0.2. Both gamma=0.9 and gamma=0.5 have 1 trial failed while gamma=0.2 succeeds with all trials. Hence optimal gamma is chosen as 0.2.
With alpha=1/t, gamma=0.2, epsilon=1/t, the agent reaches the destination within allotted time, with net reward remaining positive.

```
In [1]: import numpy as np
        import pandas as pd
        from IPython.display import display

        # learn with gamma = 0.9
        gamma1 = pd.DataFrame({'Trial': np.array(range(10)),
                               'Reach Destination': np.array(list('F' + 'T'
        * 9)),
                               'Steps Taken': np.array([41, 13, 13, 36, 8,
        7, 13, 20, 14, 16]),
                               'Net Reward': np.array(list('+' * 10))
                              })

        # learn with gamma = 0.5
        gamma2 = pd.DataFrame({'Trial': np.array(range(10)),
                               'Reach Destination': np.array(list('T' * 6 +
        'F' + 'T' * 3)),
                               'Steps Taken': np.array([12, 14, 10, 14, 6,
        7, 31, 8, 8, 6]),
                               'Net Reward': np.array(list('+' * 10))
                              })
        # learn with gamma = 0.2
        gamma3 = pd.DataFrame({'Trial': np.array(range(10)),
                               'Reach Destination': np.array(list('T' * 1
        0)),
                               'Steps Taken': np.array([17, 10, 17,8, 20, 1
        8, 8, 29, 13, 26]),
                               'Net Reward': np.array(list('+' * 10))
                              })
        print 'gamma1 = 0.9'
        display(gamma1)
        print
        print 'gamma2 = 0.5'
        display(gamma2)
        print
        print 'gamma3 = 0.2'
        display(gamma3)
```

```
gamma1 = 0.9
```

	Net Reward	Reach Destination	Steps Taken	Trial
0	+	F	41	0
1	+	T	13	1
2	+	T	13	2
3	+	T	36	3
4	+	T	8	4
5	+	T	7	5
6	+	T	13	6
7	+	T	20	7
8	+	T	14	8
9	+	T	16	9

```
gamma2 = 0.5
```

	Net Reward	Reach Destination	Steps Taken	Trial
0	+	T	12	0
1	+	T	14	1
2	+	T	10	2
3	+	T	14	3
4	+	T	6	4
5	+	T	7	5
6	+	F	31	6
7	+	T	8	7
8	+	T	8	8
9	+	T	6	9

```
gamma3 = 0.2
```

	Net Reward	Reach Destination	Steps Taken	Trial
0	+	T	17	0
1	+	T	10	1
2	+	T	17	2
3	+	T	8	3
4	+	T	20	4
5	+	T	18	5
6	+	T	8	6
7	+	T	29	7
8	+	T	13	8
9	+	T	26	9

Question 4b

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?

Answer:
Run 30 trials, the average steps for the agent to get to the destination is 13.06; the average net penalty is -2.03.
The agent tends to take best actions whenever possiple, therefore sometimes breaking traffic rules but always avoiding taking less optimal actions and running in circle. The agent can always make to the destination within alloted time but with an average penalty of -2.03. One possible explanation might be the absolute value of a reward given to the agent when it's approaching the destination is always greater than the absolute value of a penalty given to the agent when it breaks a traffic rule. Therefore, the agent always prioritize moving toward the destination rather than moving without breaking traffic rules.

```
In [2]: learning = pd.DataFrame({'Trial': np.array(range(30)),
                                 'Reach Destination': np.array(list('T' * 3
        0)),
                                 'Steps Taken': np.array([11, 19, 11, 9, 6,
        11, 19, 12, 13, 18,
                                                          16, 4, 6, 14, 23,
        26, 12, 11, 10, 12,
                                                          29, 10, 5, 12, 8,
        19, 7, 16, 12, 11]),
                                 'Net Reward': np.array(list('+' * 30)),
                                 'Net Penalty': np.array([-2, -3, -2, -2, -
        1, -2, -1, -2, -2, -4,
                                                          -2, 0, -2, -1, -
        2, -3, -1, -3, -1, -3,
                                                          -2, -3, -1, -2, -
        1, -4, 0, -2, -3, -4])
                                 })
        print 'Steps and Net Penalty for 30 trials:'
        display(learning.drop('Trial', axis=1).describe())
        print '30 Trials:'
        display(learning)
```

```
Steps and Net Penalty for 30 trials:
```

	Net Penalty	Steps Taken
count	30.000000	30.000000
mean	-2.033333	13.066667
std	1.066200	5.988111
min	-4.000000	4.000000
25%	-3.000000	10.000000
50%	-2.000000	12.000000
75%	-1.000000	16.000000
max	0.000000	29.000000

```
30 Trials:
```

	Net Penalty	Net Reward	Reach Destination	Steps Taken	Trial
0	-2	+	T	11	0
1	-3	+	T	19	1
2	-2	+	T	11	2
3	-2	+	T	9	3
4	-1	+	T	6	4
5	-2	+	T	11	5
6	-1	+	T	19	6
7	-2	+	T	12	7
8	-2	+	T	13	8
9	-4	+	T	18	9
10	-2	+	T	16	10
11	0	+	T	4	11
12	-2	+	T	6	12
13	-1	+	T	14	13
14	-2	+	T	23	14
15	-3	+	T	26	15
16	-1	+	T	12	16
17	-3	+	T	11	17
18	-1	+	T	10	18
19	-3	+	T	12	19
20	-2	+	T	29	20
21	-3	+	T	10	21
22	-1	+	T	5	22
23	-2	+	T	12	23
24	-1	+	T	8	24
25	-4	+	T	19	25
26	0	+	T	7	26
27	-2	+	T	16	27
28	-3	+	T	12	28
29	-4	+	T	11	29