Implement a Basic Driving Agent

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?*

The car hardly made it to the destination.
The car randomly drives. Sometimes it gets positive rewards. But it mostly gets negative rewards.

Inform the Driving Agent

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?*

My state schema: (light, oncoming, right, left, next_waypoint)
It is what I obtain from the "input" variable, which is the only information I can get. I don't think extra encoding/transform of the input would give me any new information.

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?*

2 * 3 * 3 * 3 * 3 = 162 states
162 is a small number. It's easy to store them in a hashtable for look-up/modification later.
Additionally, for each state there are 4 actions to take, therefore we only need 162 * 4 = 648 q states for the q-learning.
Such an encoding schema is easy to implement and logically correct.

Implement a Q-Learning Driving Agent

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?*

The agent acts randomly at first, then it converges to a certain pattern. But it prefers not doing anything for some reason. Pythonically speaking, the action is usually None.

It has something to do with my discount rate, learning rate and the initial random q values.

My learning rate was 0.2 at first, and discount rate was 09 at first. Because I only train the agent with 100 iterations, this rate is too "lazy" for the agent.

Also, the initial q values were random integers from -5 to 5, which made a lot of q states have the same q value. Because the problem gives the "None" action 0 reward, and any wrong actions negative rewards, the agent was encouraged to do nothing to stay "safe".

Improve the Q-Learning Driving Agent

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?*

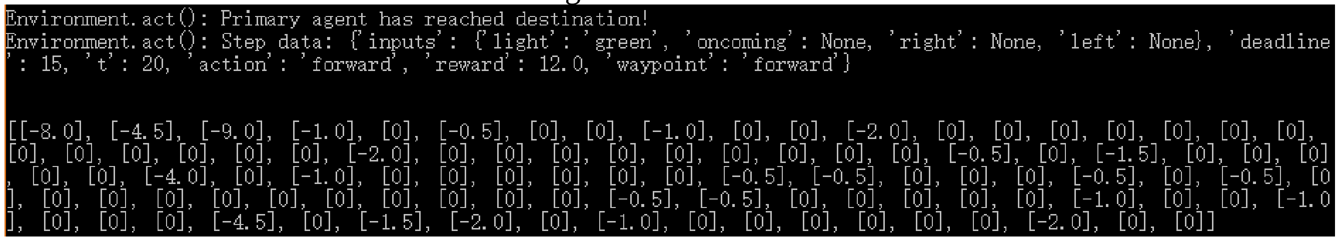
I tried many combinations

Learning Rate: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Discount Rate: 0.25, 0.3, 0.4, 0.5, 0.75, 0.8, 0.85, 0.9, 0.95
Initial Random Q Values: int[-5, 5], int[-10, 10], int[0, 10], float[0, 10], float[0, 3] …

I found that a high learning rate (>=0.5) and a low discount (<=0.5) with a floating point initial q value worked best.

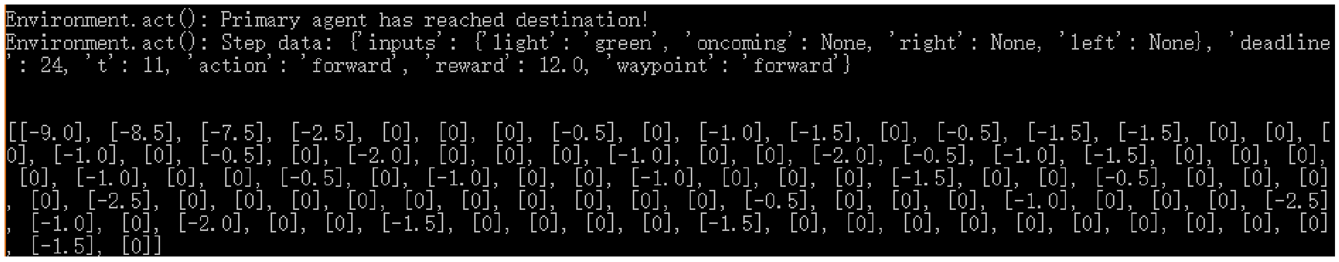
According to the negative rewards record, the agent converges quickly in such a setting.

Below are several screen shots of the the negative rewards record



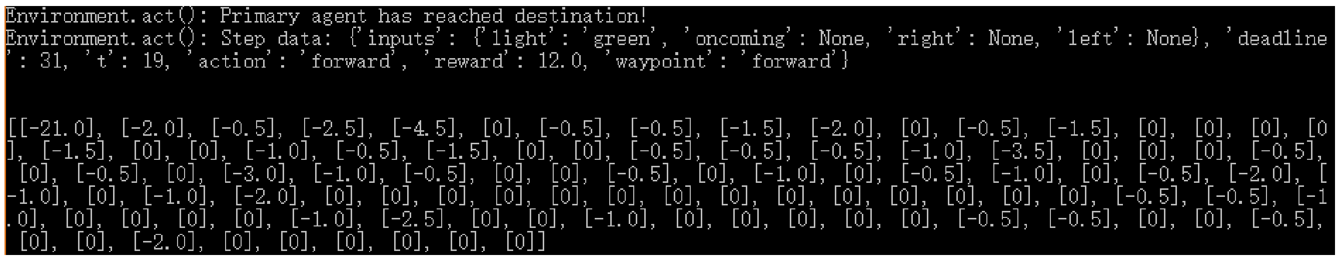
```
Environment.act(): Primary agent has reached destination!
Environment.act(): Step data: {'inputs': {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, 'deadline
': 15, 't': 20, 'action': 'forward', 'reward': 12.0, 'waypoint': 'forward'}


[[-8.0], [-4.5], [-9.0], [-1.0], [0], [-0.5], [0], [0], [-1.0], [0], [0], [-2.0], [0], [0], [0], [0], [0], [0], [0],
[0], [0], [0], [0], [0], [0], [-2.0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [-0.5], [0], [-1.5], [0], [0], [0]
, [0], [0], [-4.0], [0], [-1.0], [0], [0], [0], [0], [0], [0], [-0.5], [-0.5], [0], [0], [0], [-0.5], [0], [-0.5], [0
], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [-0.5], [-0.5], [0], [0], [0], [0], [0], [-1.0], [0], [0], [-1.0
], [0], [0], [0], [-4.5], [0], [-1.5], [-2.0], [0], [-1.0], [0], [0], [0], [0], [0], [0], [-2.0], [0], [0]]
```



```
Environment.act(): Primary agent has reached destination!
Environment.act(): Step data: {'inputs': {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, 'deadline
': 24, 't': 11, 'action': 'forward', 'reward': 12.0, 'waypoint': 'forward'}


[[-9.0], [-8.5], [-7.5], [-2.5], [0], [0], [0], [-0.5], [0], [-1.0], [-1.5], [0], [-0.5], [-1.5], [-1.5], [0], [0], [
0], [-1.0], [0], [-0.5], [0], [-2.0], [0], [0], [0], [-1.0], [0], [0], [-2.0], [-0.5], [-1.0], [-1.5], [0], [0], [0],
 [0], [-1.0], [0], [0], [-0.5], [0], [-1.0], [0], [0], [-1.0], [0], [0], [0], [-1.5], [0], [0], [-0.5], [0], [0], [0]
, [0], [-2.5], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [-0.5], [0], [0], [0], [-1.0], [0], [0], [0], [-2.5]
, [-1.0], [0], [-2.0], [0], [0], [-1.5], [0], [0], [0], [0], [-1.5], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0]
, [-1.5], [0]]
```



```
Environment.act(): Primary agent has reached destination!
Environment.act(): Step data: {'inputs': {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, 'deadline
': 31, 't': 19, 'action': 'forward', 'reward': 12.0, 'waypoint': 'forward'}


[[-21.0], [-2.0], [-0.5], [-2.5], [-4.5], [0], [-0.5], [-0.5], [-1.5], [-2.0], [0], [-0.5], [-1.5], [0], [0], [0], [0
], [-1.5], [0], [0], [-1.0], [-0.5], [-1.5], [0], [0], [-0.5], [-0.5], [-0.5], [-1.0], [-3.5], [0], [0], [0], [-0.5],
 [0], [-0.5], [0], [-3.0], [-1.0], [-0.5], [0], [0], [-0.5], [0], [-1.0], [0], [-0.5], [-1.0], [0], [-0.5], [-2.0], [
-1.0], [0], [-1.0], [-2.0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [-0.5], [-0.5], [-1
.0], [0], [0], [0], [0], [-1.0], [-2.5], [0], [0], [-1.0], [0], [0], [0], [0], [0], [-0.5], [-0.5], [0], [0], [-0.5],
 [0], [0], [-2.0], [0], [0], [0], [0], [0], [0]]
```

The number sequence represents the sum of negative rewards of each iteration(trial).

As you can see, the agent converges to 0 quickly.

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?*

I will see my agent is close to optimal policy but hasn't been not optimal yet. We can still observer some negative rewards from last 10 trials.

I will describe an optimal policy in way metrics:
1. No negative rewards, which means the agent never takes wrong actions nor miss the deadline.
2. Shortest time/Shortest Path, which means the agent don't spend 2 actions if 1 is enough.

However, the second measurement is harder to encode in to q-states and observe.