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

Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

- About 30%, random action smartcab can arrive at destination.
- random action
  - average distance : 4.56253423423
  - average dealine : 29.0
  - average steps : 25.89
  - average total reward : 0.44
  - average penalty : 6.91

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 enforcement `enforce_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?*

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

What states have you identified that are appropriate for modeling the smartcab and environment?

- Available environment values
  - `light` = ['red', 'green']
  - `oncoming` = [None, 'forward', 'left', 'right']
  - `right` = [None, 'forward', 'left', 'right']
  - `left` = [None, 'forward', 'left', 'right']
  - `next_waypoint` = ['forward', 'left', 'right', None]
- All available states count is 2 * 4 * 4 * 4 * 4 = 512
- But we do not need all 512 states. I think following 16 states are appropriate for this problem.
- I don't use `deadline` feature when select state.
  - Because if smartcab has no time for reach to destination within deadline can be ignore rule of the road.
  - So, I ignore deadline feature deliberately.
- CanMove states.
  - `light` = 'green'
    - `GREEN_CAN_LEFT`
      - `None`, `'forward'`, `'right'`, `'left'`
    - `GREEN_CANT_LEFT`
      - `None`, `'forward'`, `'right'`
  - `light` = `red`
    - `RED_CAN_RIGHT`
      - `None`, `'right'`
    - `RED_CANT_RIGHT`
      - `None`
- CanMove states describe our smartcab can go or not to some direction.
  - It contains `light`, `oncoming`, `right`, `left` environment values.
  - So, I think these values must included to available states.
- And append `next_waypoint`.
  - Because `next_waypoint` determine smartcab's next action.
  - ex. If `next_waypoint` is not included to states, we can not determine action correctly.
    - `GREEN_CAN_LEFT` state does not describe about which action should be selected.
    - If next_waypoint is included to states, `GREEN_CAN_LEFT_NEXT_NONE` will be select `None` action.
    - Otherwise, `GREEN_CAN_LEFT_NEXT_FORWARD` should be select `'forward'` action.
- SO, I determine available_states as
  - available_states = [

```
'GREEN_CAN_LEFT_NEXT_NONE'
, 'GREEN_CAN_LEFT_NEXT_FORWARD'
, 'GREEN_CAN_LEFT_NEXT_LEFT'
, 'GREEN_CAN_LEFT_NEXT_RIGHT'

, 'GREEN_CANT_LEFT_NEXT_NONE'
, 'GREEN_CANT_LEFT_NEXT_FORWARD'
, 'GREEN_CANT_LEFT_NEXT_LEFT'
, 'GREEN_CANT_LEFT_NEXT_RIGHT'
```

Why do you believe each of these states to be appropriate for this problem?

- I think above 16 states can represent all possible actions.

How many states in total exist for the smartcab in this environment?

- 512

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?

- No. too many redundant states.

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 (https://classroom.udacity.com/nanodegrees/nd009/parts/0091345409/modules/e64f9a65-fdb5-4e60-81a9-72813beebb7e/lessons/5446820041/concepts/6348990570923) 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

What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken?

- random action
    - average distance : 4.56253423423
    - average dealine : 29.0
    - average steps : 25.89
    - average total reward : 0.44
    - average penalty : 6.91
- initial using q-learning(training by random select by epsilon first 90 times, estimate last 10 times without random select.)
    - average distance : 4.66289817
    - average dealine : 29.9
    - average steps : 13.21
    - average total reward : 22.105
    - average penalty count : 0

- Q-learning smartcab can find the way to destinaiton.
    - Going circles sometimes.
    - Occasionally select random way due to epsilon.
    - Following the rule of road(if turn of random select by epsilon.).
    - It can find the way to destination.

Why is this behavior occurring?

- Smartcab learned by q-learning. And select appropriate way rather than dumb random way.

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?*

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?*

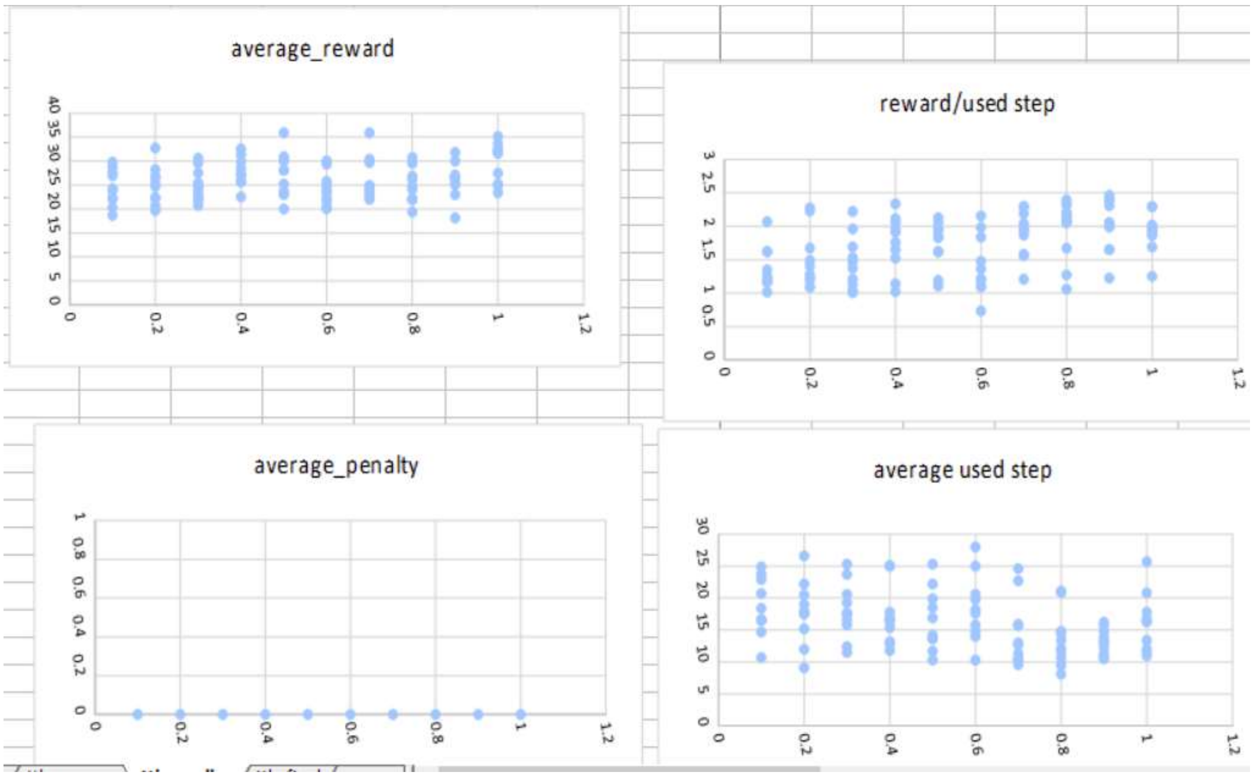
ANSWER

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?

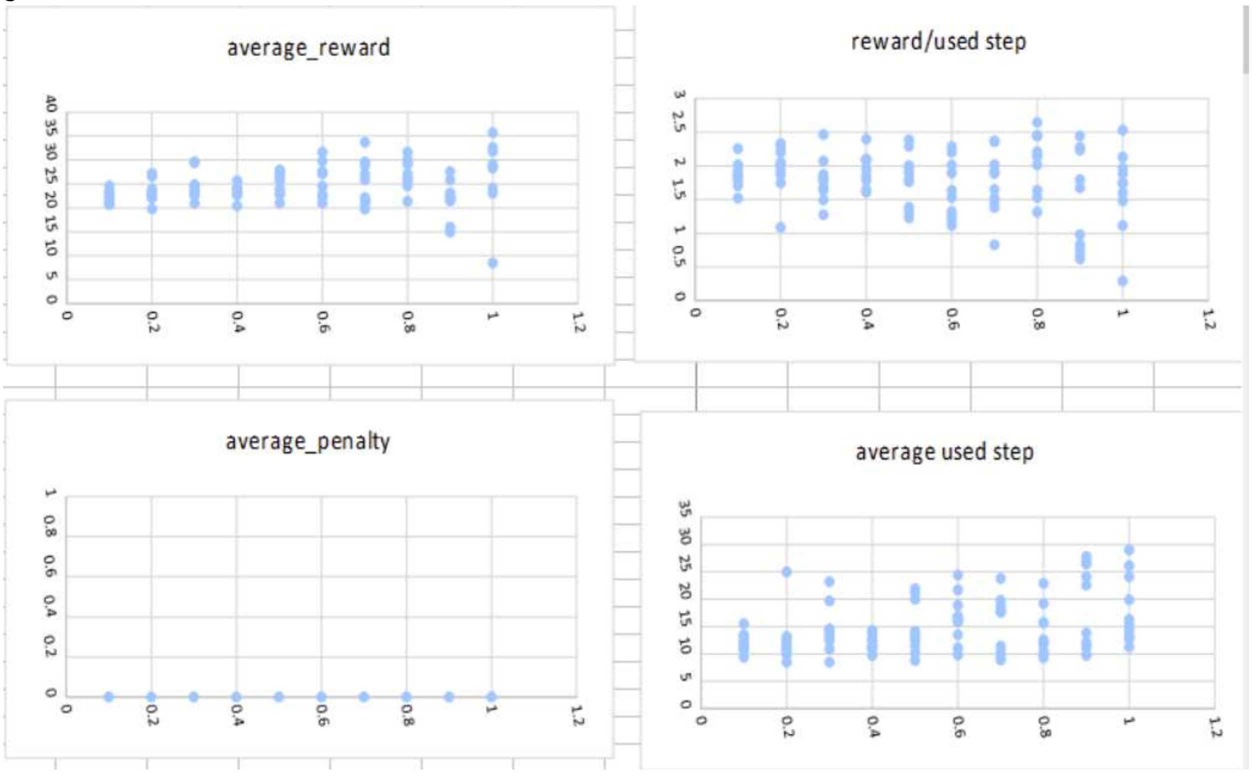
- Initialize q-values.
    - default all q-values are zero.
    - big penalty when ignore rule of road.

```
# big penalty when ignore rule of the road
big_penalty_value = -100
self.set_q_value('GREEN_CANT_LEFT_NEXT_NONE', 'left', big_penalty_value)
self.set_q_value('GREEN_CANT_LEFT_NEXT_FORWARD', 'left', big_penalty_value)
self.set_q_value('GREEN_CANT_LEFT_NEXT_LEFT', 'left', big_penalty_value)
self.set_q_value('GREEN_CANT_LEFT_NEXT_RIGHT', 'left', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_NONE', 'forward', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_NONE', 'left', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_FORWARD', 'forward', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_FORWARD', 'left', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_LEFT', 'forward', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_LEFT', 'left', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_RIGHT', 'forward', big_penalty_value)
self.set_q_value('RED_CAN_RIGHT_NEXT_RIGHT', 'left', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_NONE', 'forward', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_NONE', 'left', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_NONE', 'right', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_FORWARD', 'forward', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_FORWARD', 'left', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_FORWARD', 'right', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_LEFT', 'forward', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_LEFT', 'left', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_LEFT', 'right', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_RIGHT', 'forward', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_RIGHT', 'left', big_penalty_value)
self.set_q_value('RED_CANT_RIGHT_NEXT_RIGHT', 'right', big_penalty_value)
```

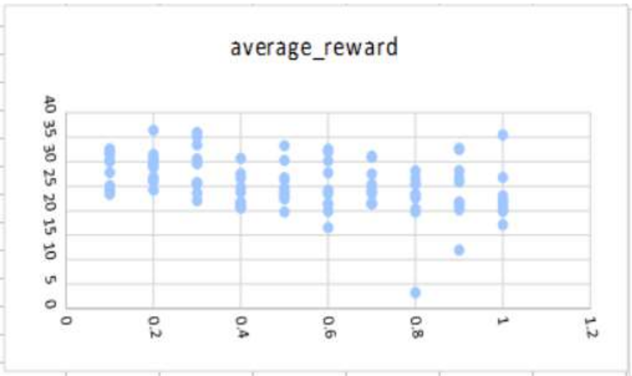
- I justify q-learning function like this.
    - alpha = 1 / (learning_delta * n)
    - used step = initial deadline - current deadline
    - reward'
        - if reward < -0.5 :
            - reward' = (reward *100) / max(used step, 1)* 10
        - else:
            - reward' = reward / max(used step, 1) * 10
    - Q' = (1 - alpha) * Q + alpha * (reward' + gamma * max_q_value_in_state)
- I use reward' instead of reward. Because i want force more penalty when ignore rule of the roads. And add some penalty on used step. As result, our smartcab gave less rewad when used step is large.
- And find learning_delta, gamma, epsilon values which maximize average total reward.
- test result <u>result xlsx file (20160819_smartcab_result.xlsx)</u>
- epsilon : 0.9 is best.

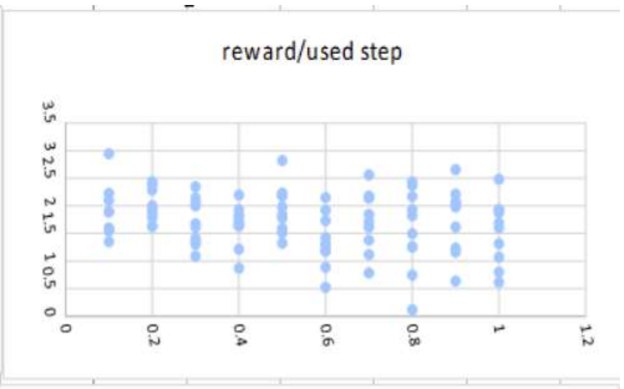


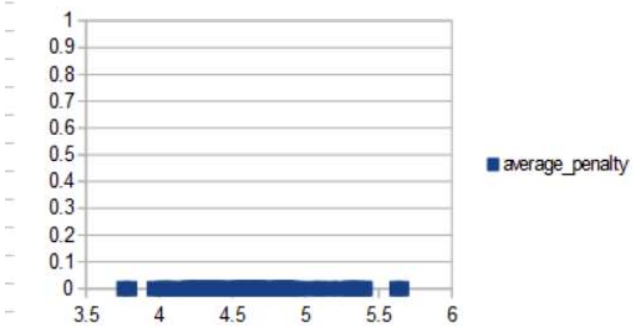
- gamma : 0.8 is best.

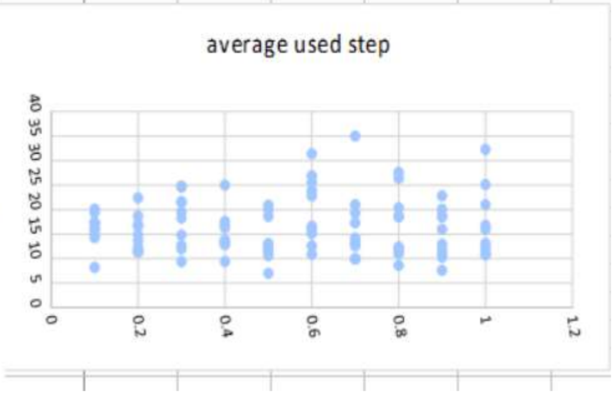


- learning_delta : 0.2 is best.









- best parameter set is
  - alpha : 1/(0.2 * n)
  - gamma : 0.8
  - epsilon : 0.9

How well does the final driving agent perform?

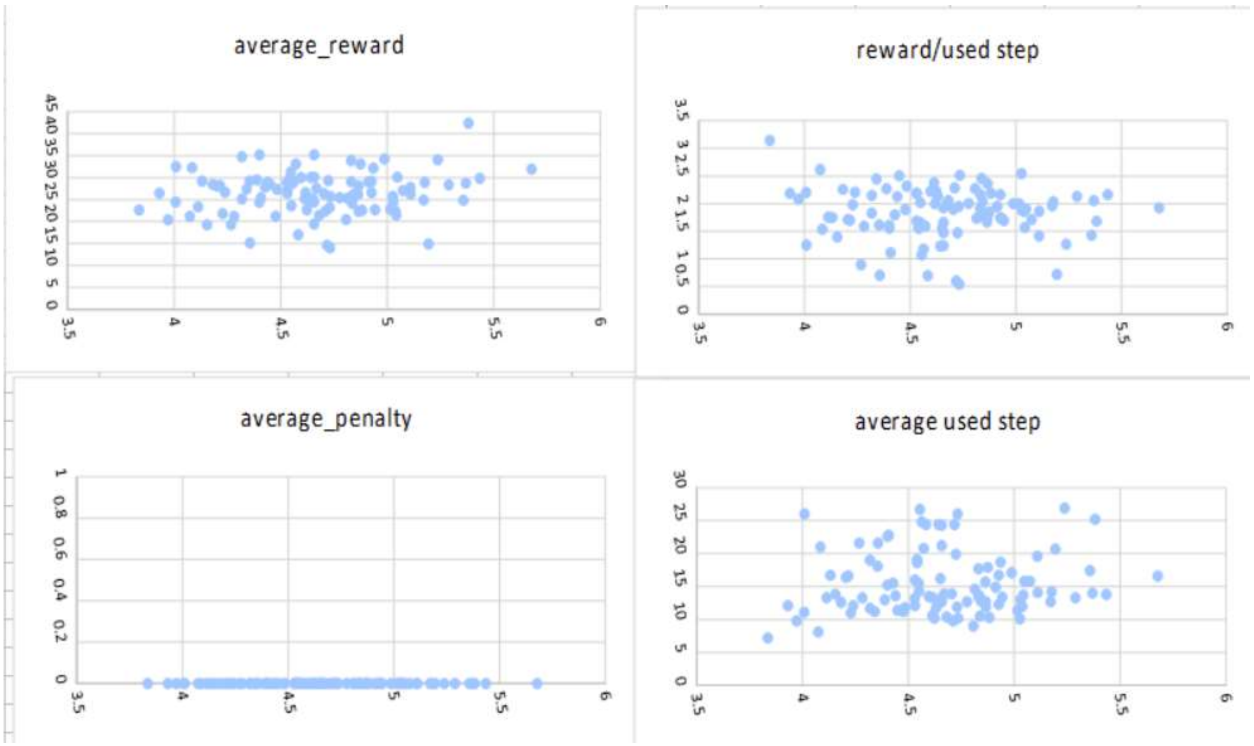
- Maximum average total reward point : 42.4
- Average average total reward point : 26.411

How would you describe an optimal policy for this problem?

- Optimal policy should be like this.
  - Not incur any penalites.
  - Average total reward is high.

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?

- Result is like this.



- Q-value's table is like this.

state	action : None	action : forward	action : left	action : right
GREEN_CAN_LEFT_NEXT_NONE	0.000000	0.000000	0.000000	0.000000
GREEN_CAN_LEFT_NEXT_FORWARD	0.934968	3.606986	2.379418	0.825169
GREEN_CAN_LEFT_NEXT_LEFT	2.002181	0.163618	5.018174	2.144785
GREEN_CAN_LEFT_NEXT_RIGHT	1.944834	4.670036	1.786558	4.495921
GREEN_CANT_LEFT_NEXT_NONE	0.000000	0.000000	-100.000000	0.000000
GREEN_CANT_LEFT_NEXT_FORWARD	0.000000	0.000000	-100.000000	0.002396
GREEN_CANT_LEFT_NEXT_LEFT	0.000000	0.000000	-100.000000	0.000000
GREEN_CANT_LEFT_NEXT_RIGHT	0.000000	0.000000	-99.451756	0.116174
RED_CAN_RIGHT_NEXT_NONE	0.000000	-100.000000	-100.000000	0.000000
RED_CAN_RIGHT_NEXT_FORWARD	0.000000	-17.628662	-39.224885	-0.580188
RED_CAN_RIGHT_NEXT_LEFT	0.377460	-55.390485	-25.718232	2.041497
RED_CAN_RIGHT_NEXT_RIGHT	5.776657	-10.242352	3.092213	10.482273
RED_CANT_RIGHT_NEXT_NONE	0.000000	-100.000000	-100.000000	-100.000000
RED_CANT_RIGHT_NEXT_FORWARD	0.000000	-100.000000	-100.000000	-95.797101
RED_CANT_RIGHT_NEXT_LEFT	0.000000	-100.000000	-100.000000	-100.000000
RED_CANT_RIGHT_NEXT_RIGHT	0.000000	-100.000000	-100.000000	-99.744473

- Our smartcab always select action that not incur any penalties.
- Average total reward is relatively high(compared to random selection).
- random action
  - average distance : 4.56253423423
  - average dealine : 29.0

  - average steps : 25.89
  - average total reward : 0.44
  - average penalty : 6.91
- initial using q-learning
  - average distance : 4.66289817
  - average dealine : 29.9
  - average steps : 13.21
  - average total reward : 22.105
  - average penalty count : 0
- initial using q-learning
  - average distance : 4.662993318
  - average dealine : 29.895
  - average steps : 15.448
  - average total reward : 26.411
  - average penalty count : 0

In [ ]:

```

```