# P4 SmartCar Report

## 1.Implement a basic driving agent

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

### Q1 Answer: python Q_agent.py in repo(Basic_Agent)

Run in the original codes,we can see the action of car is a random choice of move forward,left,right or none.This policy makes none sense.

So we add some codes to let the car obey the traffic raws.

Determining whether the car can go, acording to traffic rules in US.

```
            if self.next_waypoint == 'right':
                    if inputs['light'] == 'red' and inputs['left'] == 'forward':
                            action_okay = False

             elif self.next_waypoint == 'straight':
                    if inputs['light'] == 'red':
                            action_okay = False

             elif self.next_waypoint == 'left':
                    if inputs['light'] == 'red' or inputs['oncoming'] == 'forward' or inputs['oncoming'] == 'right':
                            action_okay = False
```

## Implement a basic driving agent in 10 turns :

In [22]:
```
import numpy as np
import pandas as pd
```

In [23]:
```
basic_agent_result = pd.read_csv('basic_agent_statics.csv')
```

In [24]:
```
basic_result = basic_agent_result.drop(['Unnamed: 0'], axis = 1)
```

In [25]:
```
basic_result
```

Out[25]:

| | avg_total_reward | reached_times |
|---|---|---|
| **0** | 23.083333 | 24 |
| **1** | 26.394737 | 19 |
| **2** | 24.760870 | 23 |
| **3** | 28.687500 | 24 |
| **4** | 23.175000 | 20 |
| **5** | 26.134615 | 26 |
| **6** | 25.150000 | 20 |
| **7** | 21.181818 | 11 |
| **8** | 25.950000 | 20 |
| **9** | 24.071429 | 14 |

In [26]:
```
basic_result.describe()
```

Out[26]:

| | avg_total_reward | reached_times |
|---|---|---|
| **count** | 10.000000 | 10.000000 |
| **mean** | 24.858930 | 20.100000 |
| **std** | 2.109725 | 4.653553 |
| **min** | 21.181818 | 11.000000 |
| **25%** | 23.399107 | 19.250000 |
| **50%** | 24.955435 | 20.000000 |
| **75%** | 26.088462 | 23.750000 |
| **max** | 28.687500 | 26.000000 |

### As the Describe table shown below,

the average EM_total_reward is 24.86 for each run ,

and the average EM_reached_times is 20,

which means the sucess rate is only 20% based on non_Q_learning algorithm in this case.

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

**Q2 Answer:**or) of inputs as an implicit state. Justify why you picked these set of states, and how they model the agent and its environment. In this case , what available variables the agent need considered are the 5 ones:

1. (the traffic light: light):

   **Reason:** The traffic light is the most important signal for all of cars .

   Only the traffic light is green for the agent to travel forward legally and safely.

2. (next_waypoint):

   **Reason:** It is a correct heading direction based on the destination,the right heading can make the road map always point to the destination.

3. ('oncoming':value):

   **Reason: The value locate in range of (right, left, forward,None )**

   **If the value is not None ,**which means there is other agent on the oncoming position of Q agent and value represents what the other agent will do next step.

4. ('right': :value):

   **Reason: The value locate in range of (right, left, forward,None )**

   **If the value is not None ,**which means there is other agent on the right position of Q agent and value represents what the other agent will do next step.

5. ('left':value):

   **Reason: The value locate in range of (right, left, forward,None )**

   **If the value is not None ,**which means there is other agent on the left position of Q agent and value represents what the other agent will do next step.

**If we introduce to much features what will make a big challengers of distributed data sparsity when the data is not enough or the system is under limited computing resource .Therefore, keeping a balance of feature numbers and data sensity is a critical part of the case.**

**Test1:**

**If deadline is added into the state tuple :**

**self.state['deadline'] = self.env.get_deadline(self)**

**( ( 'light': light ), ( 'oncoming': oncoming ), ( 'left': left ), ( 'right': right ) ,('next_waypoint': next_waypoint) , ( 'deadline': deadline ) )**

**When n_trials of simulator equalts to 100 , the best success rate(reached_times before deadline / 100 trials) is only 45% owing to the distributed data sparsity or something else.**

**Read 'Deadline_added_grid_version.csv', and then clean Data**

```
df2 = pd.read_csv('Deadline_added_grid_version.csv')
df2_sorted = df2.sort_values(['total_reached_times'], ascending = False)
df2_sorted = df2_sorted.drop(['Unnamed: 0'],axis=1)
df2_sorted['reward_per_pace'] = df2_sorted['avg_total_reward']  / df2_sorted['avg_total_paces']
df2_sorted['success_rate'] = df2_sorted['total_reached_times'] /100
```

```
df2_sorted
```

| | alpha | avg_total_deadline | avg_total_paces | avg_total_penalties | avg_total_reward | epsilon | gamma | total_reached_times | reward_per_pace |
|---|---|---|---|---|---|---|---|---|---|
| **159** | 0.75 | 31.777778 | 17.422222 | 0.933333 | 28.833333 | 0.03 | 0.45 | 45 | 1.654974 |
| **265** | 0.95 | 29.268293 | 16.634146 | 0.634146 | 28.658537 | 0.05 | 0.40 | 41 | 1.722874 |
| **141** | 0.75 | 32.250000 | 18.150000 | 1.200000 | 27.900000 | 0.03 | 0.15 | 40 | 1.537190 |
| **185** | 0.80 | 32.051282 | 20.743590 | 1.051282 | 32.320513 | 0.08 | 0.40 | 39 | 1.558096 |
| **161** | 0.75 | 31.710526 | 19.921053 | 1.368421 | 30.434211 | 0.08 | 0.45 | 38 | 1.527741 |
| **177** | 0.80 | 31.973684 | 19.763158 | 0.894737 | 30.763158 | 0.03 | 0.30 | 38 | 1.556591 |
| **95** | 0.65 | 34.078947 | 18.052632 | 1.184211 | 28.250000 | 0.08 | 0.25 | 38 | 1.564869 |
| **166** | 0.80 | 31.710526 | 19.605263 | 0.973684 | 30.723684 | 0.05 | 0.10 | 38 | 1.567114 |
| **248** | 0.95 | 30.263158 | 16.736842 | 0.868421 | 28.460526 | 0.08 | 0.10 | 38 | 1.700472 |
| **54** | 0.60 | 31.216216 | 20.864865 | 1.054054 | 31.756757 | 0.03 | 0.05 | 37 | 1.522021 |
| **220** | 0.90 | 34.459459 | 19.513514 | 1.405405 | 29.283784 | 0.05 | 0.10 | 37 | 1.500693 |
| **184** | 0.80 | 31.351351 | 17.027027 | 0.675676 | 28.229730 | 0.05 | 0.40 | 37 | 1.657937 |
| **22** | 0.50 | 28.611111 | 15.250000 | 1.000000 | 26.111111 | 0.05 | 0.40 | 36 | 1.712204 |
| **151** | 0.75 | 30.416667 | 17.555556 | 0.944444 | 29.111111 | 0.05 | 0.30 | 36 | 1.658228 |

**So deadline parameter is removed because it is not a good choice as we expected.**

**At last we make the state like below tuple:**

**Agent_State = ( ( 'light': light ), ( 'oncoming': oncoming ), ( 'left': left ),('next_waypoint': next_waypoint) )**

# 3.Implement Q-Learning

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

## Q3 Answer: **The Q3 codes in Repo(Q_learning_Agent)**

According to the random result of Q1, introducing a reinforcement learning method is needed to improve the primary agent driving level.

The Q-learning is the best choice for us.

According to the Q-learning equation:

Q(s,a) := Q(s,a) + alpha * [ r(s,a) + gamma * (argmax(Q(s',a') - Q(s,a) )

We need 3 parameters for the equation.

Alpha: the learning rate , range (0,1)

Gamma: the discount factor, range (0,1)

Epsilon: a factor for the balance of exploration and greedy , range (0,1)

Make a QL_System.py as the Q-learning policy:

Including these functions:

def getQ(…): Get the Q value in the Q dict according to the state to.

def learnQ(…): Initialize the Q value equal to 0 and update the Q value if it is existed.

def bestAction(…): Choose the best action according to the max Q value or select a random choice with a probability of epsion.

Def iterLearn(…): Update the previous Q value when the agent move to the next state.

## The way to update Q agent.state and QDict

```
def update(self, t):
    # Gather inputs
    self.next_waypoint = self.planner.next_waypoint()  # from route planner, also displayed by simulator
    inputs = self.env.sense(self)
    deadline = self.env.get_deadline(self)

    # TODO: Current state
    self.state = inputs
    self.state['next_waypoint'] = self.next_waypoint
    #self.state_next['deadline'] = self.q_deadline
    self.state = tuple(sorted(self.state.items()))

    # TODO: Select action according to your policy
    self.action = self.car_AI.bestAction(self.state)

    # Execute action and get reward
    #if agent reached, self.done = True ,and then break in env.act().
    reward = self.env.act(self, self.action)
    self.total_reward += reward


    # TODO: Next state
    inputs_next = self.env.sense(self)
    self.state_next = inputs_next
    self.state_next['next_waypoint'] = self.planner.next_waypoint()
    #self.state_next['deadline'] = self.q_deadline
    self.state_next = tuple(sorted(self.state_next.items()))

    # TODO: update Q
    if self.car_AI.qDict.get((self.state, self.action),None) is None :
        self.car_AI.qDict[(self.state, self.action)] = reward
    else:
        self.car_AI.iterLearn(self.state, self.action, reward, self.state_next)

    #update state
    self.total_paces += 1
    #self.action = action
    #self.state = self.state_next
```

**Store the key data for Q agent when it successfully arrived the destination .**

```
if agent is self.primary_agent:
    if state['location'] == state['destination']:

        self.primary_agent.total_paces += 1
        # bonus added
        if state['deadline'] >= 0:
            reward += 10

        self.primary_agent.total_reward += reward

        #store available data for each test
        self.primary_agent.total_reward_list.append(self.primary_agent.total_reward)
        self.primary_agent.penalties_list.append(self.primary_agent.penalties)
        self.primary_agent.total_paces_list.append(self.primary_agent.total_paces)
        self.primary_agent.q_deadline_list.append(self.primary_agent.q_deadline)
        self.primary_agent.reached_times += 1

        self.done = True
        print "Environment.act(): Primary agent has reached destination!"
        print " Total_reward :{} ".format(self.primary_agent.total_reward)
        print " Penalties :  {}" .format(self.primary_agent.penalties)
        print " Total_paces :  {}" .format(self.primary_agent.total_paces)  # [debug]
        print " Deadline :  {}" .format(self.primary_agent.q_deadline)
```

# 4. Enhance the driving agent

a) 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? (Here you should give a detailed list of parameter tested and report the results obtained).

### Q4 Answer to a: ( python Q_agent.py)

After running the simulator for 100 trials, I collect the key data (total_reward, total paces for each turn, total_deadline, total reached_times) and write them into a csv file and make a value sorted according to the success rate .

1. **Make an independent generate function for primary agent**

```
# make a independent generate funtion for primary agent with 3 params
def create_primary_agent(self, agent_class,alpha,gamma,epsilon):
    agent = agent_class(self, alpha,gamma,epsilon)
    self.agent_states[agent] = {'location': random.choice(self.intersections.keys()), 'heading': (0, 1)}
    return agent

def create_agent(self, agent_class, *args, **kwargs):
    agent = agent_class(self, *args, **kwargs)
    self.agent_states[agent] = {'location': random.choice(self.intersections.keys()), 'heading': (0, 1)}
    return agent
```

2. **Set an iteration function to run simulator in different values of (alpha, gamma, epsilon).**

   **At last, make an average statistic for**

**total_reward,penalties,total_paces(how many steps when agent successfully reached destination before deadline) and reached times.**

**alpha_list = np.arange(0.5,1,0.05)**

**gamma_list = np.arange(0.05,0.5,0.05)**

**epsilon_list = [0.03,0.05,0.08]**

```
def grid_best_params():

    alpha_list = np.arange(0.5,1,0.05)
    gamma_list = np.arange(0.05,0.5,0.05)
    epsilon_list = [0.03,0.05,0.08]

    alphas = []
    gammas = []
    epsilons = []

    avg_total_reward = []
    avg_total_penalties = []
    avg_total_paces = []
    avg_total_deadline = []
    total_reached_times = []

    for alpha in alpha_list:
        for gamma in gamma_list:
            for epsilon in epsilon_list:

                alphas.append(alpha)
                gammas.append(gamma)
                epsilons.append(epsilon)

                a = setup_simulator(alpha, gamma, epsilon)

                # statistic the average data of Q agent in each pair of (alpha, gamma)
                avg_total_reward.append( np.average(a.total_reward_list) )
                avg_total_penalties.append( np.average(a.penalties_list) )
                avg_total_paces.append( np.average(a.total_paces_list) )
                avg_total_deadline.append(np.average(a.q_deadline_list))
                total_reached_times.append(a.reached_times)


    pd.DataFrame({'alpha': alphas,
        'gamma': gammas,
        'epsilon': epsilons,
        'avg_total_reward': avg_total_reward,
        'avg_total_penalties': avg_total_penalties,
        'avg_total_paces': avg_total_paces,
        'avg_total_deadline': avg_total_deadline,
        'total_reached_times': total_reached_times,
        }).to_csv('Gridsearchn100_version.csv')
```

### 3. Clean data in the statistics_file.csv

```
def data_clean_smartcab(file_name):

    df = pd.read_csv(file_name)
    df_sorted = df.sort_values(['total_reached_times'], ascending=False)
    df_sorted['reward_per_pace'] = df['avg_total_reward']  / df['avg_total_paces']
    df_sorted['success_rate'] = df['total_reached_times'] /100
    df_sorted.to_csv('Gridsearchn100_version.csv')
```

4. **Tuned different combinations of (alpha,gamma, epsilon),**

   **The top 20 performances dataset : success rate range in 95%-99%**

**Read 'Gridsearchn100_version.csv', and then clean Data**

```
df4 = pd.read_csv('Gridsearchn100_version.csv')
```

```
df4_sorted = df4.sort_values(['total_reached_times'], ascending = False)
df4_sorted = df4_sorted.drop(['Unnamed: 0'],axis=1)
df4_sorted['reward_per_pace'] = df4_sorted['avg_total_reward']  / df4_sorted['avg_total_paces']
df4_sorted['success_rate'] = df4_sorted['total_reached_times'] /100
```

```
df4_sorted
```

| | alpha | avg_total_deadline | avg_total_paces | avg_total_penalties | avg_total_reward | epsilon | gamma | total_reached_times | reward_per_pace |
|---|---|---|---|---|---|---|---|---|---|
| **63** | 0.60 | 30.808081 | 14.333333 | 0.161616 | 30.707071 | 0.03 | 0.20 | 99 | 2.142354 |
| **91** | 0.65 | 29.191919 | 13.929293 | 0.232323 | 30.000000 | 0.05 | 0.20 | 99 | 2.153735 |
| **168** | 0.80 | 30.555556 | 14.888889 | 0.141414 | 31.065657 | 0.03 | 0.15 | 99 | 2.086499 |
| **253** | 0.95 | 29.646465 | 13.494949 | 0.343434 | 29.262626 | 0.05 | 0.20 | 99 | 2.168413 |
| **228** | 0.90 | 29.387755 | 14.071429 | 0.204082 | 30.188776 | 0.03 | 0.25 | 98 | 2.145395 |
| **22** | 0.50 | 29.285714 | 13.132653 | 0.244898 | 28.765306 | 0.05 | 0.40 | 98 | 2.190365 |
| **70** | 0.60 | 30.721649 | 14.082474 | 0.226804 | 29.984536 | 0.05 | 0.30 | 97 | 2.129209 |
| **230** | 0.90 | 28.865979 | 13.474227 | 0.402062 | 28.711340 | 0.08 | 0.25 | 97 | 2.130834 |
| **56** | 0.60 | 31.649485 | 15.432990 | 0.350515 | 31.536082 | 0.08 | 0.05 | 97 | 2.043420 |
| **238** | 0.90 | 29.690722 | 14.670103 | 0.237113 | 30.469072 | 0.05 | 0.40 | 97 | 2.076950 |
| **119** | 0.70 | 30.000000 | 15.082474 | 0.474227 | 31.298969 | 0.08 | 0.20 | 97 | 2.075188 |
| **251** | 0.95 | 29.536082 | 14.463918 | 0.453608 | 29.938144 | 0.08 | 0.15 | 97 | 2.069850 |
| **116** | 0.70 | 29.895833 | 14.375000 | 0.447917 | 30.447917 | 0.08 | 0.15 | 96 | 2.118116 |
| **160** | 0.75 | 29.427083 | 15.052083 | 0.333333 | 30.812500 | 0.05 | 0.45 | 96 | 2.047059 |
| **115** | 0.70 | 29.010417 | 13.104167 | 0.229167 | 28.640625 | 0.05 | 0.15 | 96 | 2.185612 |
| **50** | 0.55 | 29.736842 | 14.442105 | 0.431579 | 29.636842 | 0.08 | 0.40 | 95 | 2.052114 |
| **44** | 0.55 | 30.052632 | 15.231579 | 0.410526 | 31.268421 | 0.08 | 0.30 | 95 | 2.052868 |
| **233** | 0.90 | 29.526316 | 14.389474 | 0.389474 | 30.089474 | 0.08 | 0.30 | 95 | 2.091075 |
| **122** | 0.70 | 28.473684 | 13.105263 | 0.410526 | 28.236842 | 0.08 | 0.25 | 95 | 2.154618 |
| **158** | 0.75 | 31.052632 | 15.010526 | 0.305263 | 31.047368 | 0.08 | 0.40 | 95 | 2.068373 |

**As the table shown, 4 groups get the best reached_times 99 in 100 trials .**

1. **(alpha:0.60,gamma:0.20,epsilon:0.03)**

   **Get 99% success rate,**

   **avg_total_penalties=0.16paces/100trials**

   **Reached the destination only cost average 14.33paces in average deadline 30.80 paces.**

   **And Q agent get 2.14rewards/per paces with EM_total_rewards = 30.71 points.**

2. **(alpha:0.65,gamma:0.20,epsilon:0.05)**

   **Get 99% success rate,**

   **avg_total_penalties=0.23paces/100trials**

   **Reached the destination only cost average 13.93paces in average deadline 29.19 paces.**

   **And Q agent get 2.15rewards/per paces with EM_total_rewards = 30.00 points.**

3. **(alpha:0.80,gamma:0.15,epsilon:0.03)**

   **Get 99% success rate,**

   **avg_total_penalties=0.14paces/100trials**

   **Reached the destination only cost average 14.88paces in average deadline 30.55paces.**

   **And Q agent get 2.08rewards/per paces with EM_total_rewards = 31.07points.**

4. **(alpha:0.95,gamma:0.20,epsilon:0.05)**

   **Get 99% success rate,**

   **avg_total_penalties=0.34paces/100trials**

   **Reached the destination only cost average 13.49paces in average deadline 29.38paces.**

   **And Q agent get 2.16rewards/per paces with EM_total_rewards = 29.26points.**

Q4 b) 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? (Here you should observe and analysis the behavior of the agent, does it always go in an optimal path and follow traffic laws and if not under what specific circumstances does it not follow the optimal policies).

## Q4 Answer to b:

The best marks of success rate is 99% in the specified parameters (alpha= 0.60, gamma= 0.20, epsilon = 0.03 ) ，(alpha= 0.65, gamma= 0.20, epsilon = 0.05 ) ，(alpha= 0.80, gamma= 0.15 epsilon = 0.03 ), (alpha= 0.95, gamma= 0.20, epsilon = 0.05 )which is much higher than 20% the random in basic agent method.In addition, the Q agent cost only 14.33 total paces while the average deadline is 30.81.It is meaning the Q agent is high efficiency to reach the destination .

As we can see in the top20 data table, different parameters values combined as a group also make good marks in this case,which means the parameters is not as expected important as we think before.Furthermore, a good Q-learning policy in agent.update really play the key role in it.

The Q agent got 99% success rate meaning only 0.14-0.34 penalties got in 100 trials while only cost less than 50% steps of deadline for a trip.

So I think the optimal policy based on Q-learning has been found.In addition, we can find this method got much less average paces ,penalites and higher total rewards to reach the destination than other schemes in different parameters.

It means the Q_agent not only has learned how to reach the destination correctly but also knowed the best road plan in each location.