1. Implement a basic driving agent

Produces random action which are None, Forward, Left and Right with Random(). which will choose actions evenly.

When agent move by random action, it is likely to go around the map and sometime go and back and forward around the same place. It will eventually go around and will get to target location with some random step sometimes it is fast and sometimes it looks hundreds of steps.

2. Identify and update state

I choose these information to be included in state vector

1. Traffic light, so that agent can learn traffic rule.
2. Other car information from oncoming and left , so agent can learn traffic rule when other car in the street are coming. Traffic from right it in included because US right-of-way*1 rules does not consider right traffic.
3. next way point, agent should know where should it heading to.

If traffic light is omitted, agent can not learn traffic light rule, and if other cars information is omitted it may break traffic rule and crash with other cars. And if next way point is omitted, agent will not have information to decide which way to go to reach destination.

I omit deadline because it does not tell how to go to destination or how it will learn traffic rules. I may help agent to learn to go other way around if it get caught in traffic light when deadline is not so close, but I think that is not necessary because there is not much traffic jam in simulation and adding more state would make longer time to learn all the state and correct Q value.

*1 *US right-of-way rules : 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.*

3. Implement Q-Learning

I set initial Q value to be at 1.0. And choose the highest Q value as action. If they have same Q value then randomly choose one.

After action is chosen, agent get reward and use reward to update Q table below formula

*Q(S,A) = (1-learningRate)*Q(S',A') + learningRate * (reward + discountFactor * Q(S',A'))*

It will pick the highest reward action so after steps it will learn that going against traffic rule will give minus reward and that action will not be picked again.

I ran Q-Learning for 3 times and compare result with random action. For n=100 trials.

	Agent reach destination	Cumulative penalty (times)	Cumulative Number of Moves	Cumulative Penalty Ratio
Random	18	674	2751	0.245
Q-Learning #1	100	14	1255	0.011
Q-Learning #2	36	19	2420	0.008
Q-Learning #3	50	22	2376	0.009

From above table, after apply Q-learning agent will learn about traffic rule and avoid penalty. Random action has 0.245 cumulation penalty ration and Q-learning has average of cumulative penalty ratio of 0.009. And Q-learning agent has reach destination more than random action.

I notice agent in Q-learning that is has very difference behaviour each time it run. At first time agent randomly choose correct policy and continue run the same ways. It learn how to go to destination very quickly. But the second and third runs, agents tend to stay at None action a lot and does not move for several turns. But it finally slowly goes to destination and most of the time is the deadline is over before it reach destination. This is because agent remember method it already learn and stop to go other way which might be better.

4. Enhance the driving agent

4.1 Decay epsilon

Q-Learning in #3 will choose only best action so it is very depends on what action is chosen first and it will try to do the same actions and it does not try to explore others action. I saw agent try to turn right many times and some point it change to None action many times. So eventually agent may choose to stay in None action for long time. To fix this problem I insert epsilon variable and tell it to explore others options.

Epsilon value is set to 1.0 at start and linearly decrease. Where at trial #35 it epsilon becomes 0.0 and stop to explore other actions. And from here agent will choose the best score from Q-table.

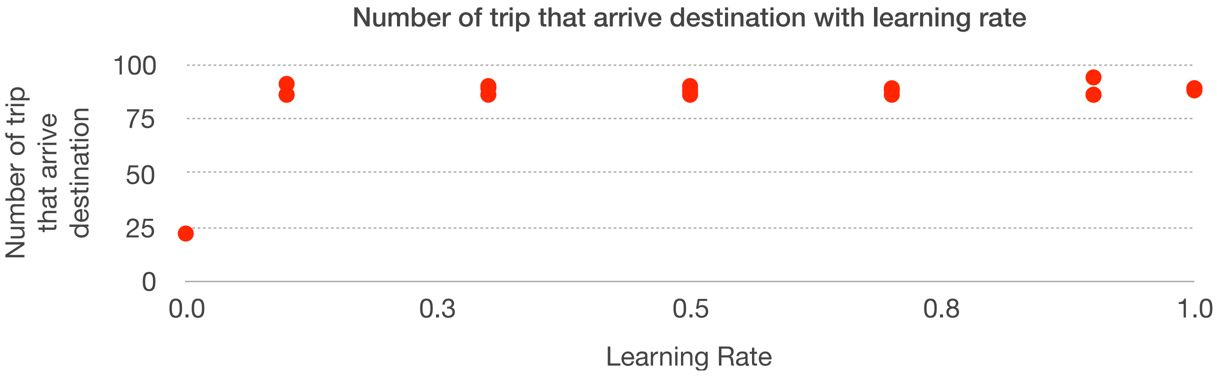
After insert exploration action, agent start to move around randomly at first and break many traffic rule. But after trial#35 where epsilon become 0, agent follow traffic and constantly move toward destination quickly.

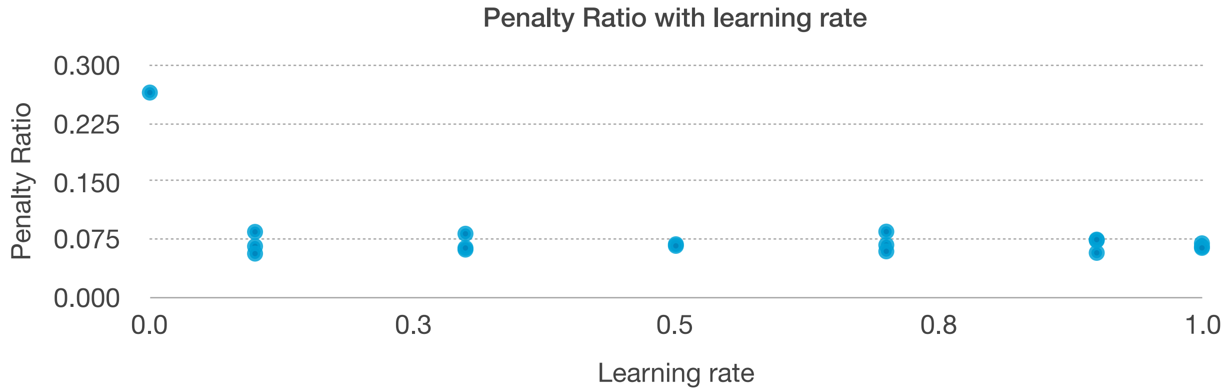
With	learning rate	0.5
	epsilon	1.0 : decay at -0.03 per trial
	discount factor	0.2

Result : Agent arrive destination within deadline 90 out of 100 times. And cumulative time of penalty is 104 out of 1508 or cumulation penalty ratio is 0.069 which is higher than before but still less than random action.

4.2 Tuning learning rate

Below graph is a result for changing learning rate from 0.0 to 1.0. Plot with number of trip that agent arrive destination in 100 trips and penalty ratio.





When learning rate is 0, it will not update Q-table value and continue to use initial value as formula below. And when learning rate increase it will also increase to update Q value to new reward. If learning rate is 1.0 agent will update all of the reward from current information.

*Q(S,A) = (1-learningRate)*Q(S',A') + learningRate * (reward + discountFactor * Q(S',A'))*

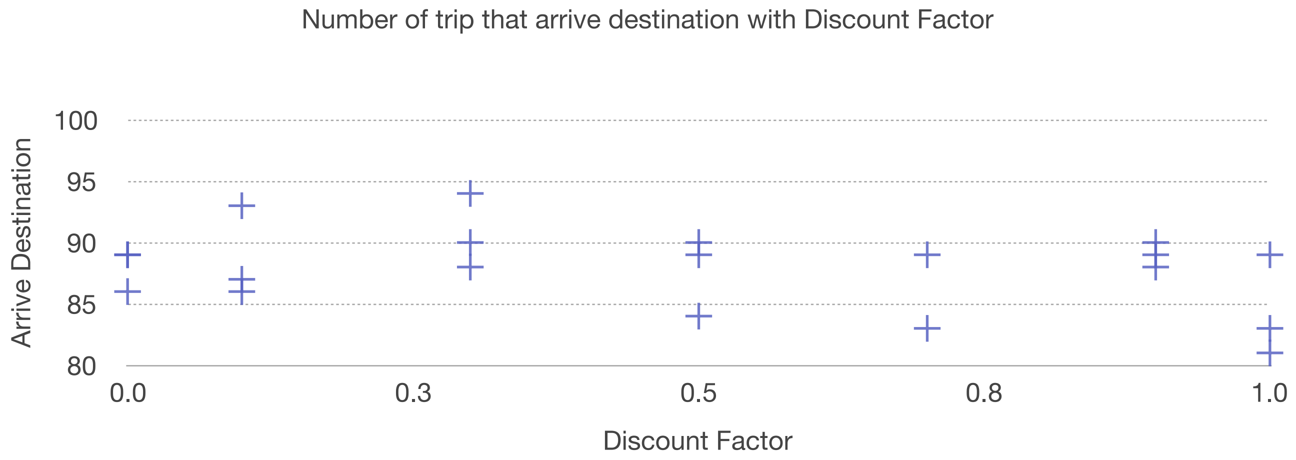
From the result, learning rate 0.0 give a penalty ratio and number of trip that arrive destination result as the same as random action. Since agent cannot learn anything. And for learning rate 0.1 - 1.0, agent can arrive destination are more that 85%. with penalty rate 0.07 and different learning rate seems not to change much. I think this is because this problem is a fully deterministic environment. So tuning learning rate does not change much result and that agent perform same result as learning rate is 1.0.

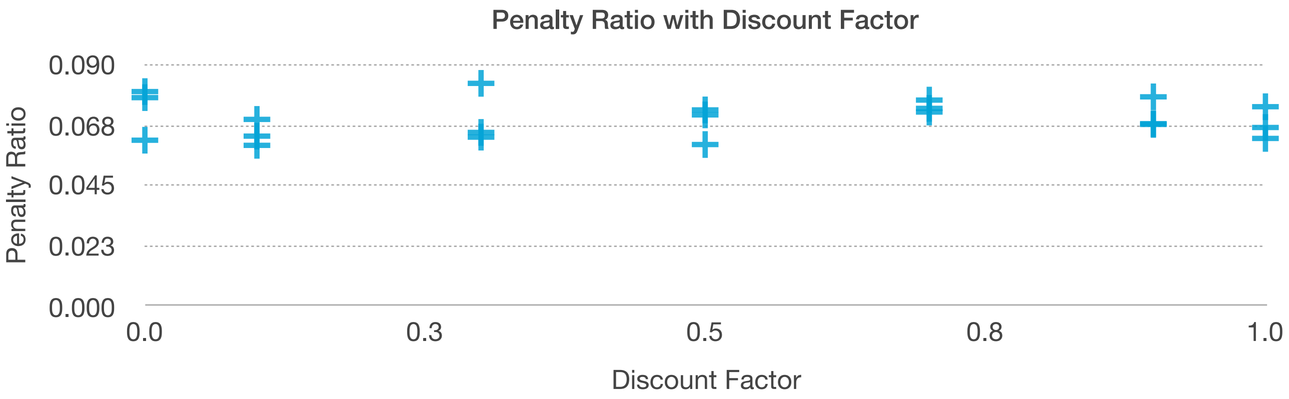
So I choose to set learning rate at 1.0.

4.3 Tuning discount factor

Discount factor is a parameter that give an importance of estimate reward Q(S',A'). If it is near 0 agent will learn from only current reward and if it is near 1.0 it will strive for a long term high reward.

In this experiment, i set learning rate at 1.0 and change discount factor from 0.0 to 1.0. Result is plot in below graphs





From result, discount factor = 0.3 has highest number of trip that arrive destination. Penalty Ratio for discount factor = 0.1 is lowest, but I decide to use 0.3 because I don't see much different for range of penalty ratio (0.06 - 0.08)

4.4 Final result and optimal policy

As final result, tuned parameter are

learning rate	1.0
epsilon	1.0 : decay at -0.03 per trial
discount factor	0.3

With this parameter, agent can reach destination at average of 90.67 time in 100 runs. Penalty ratio is 0.07.

Agent will go toward directly destination direction no driving around, but if that direction is against the traffic rule agent will stop and wait until that direction is clear to go.