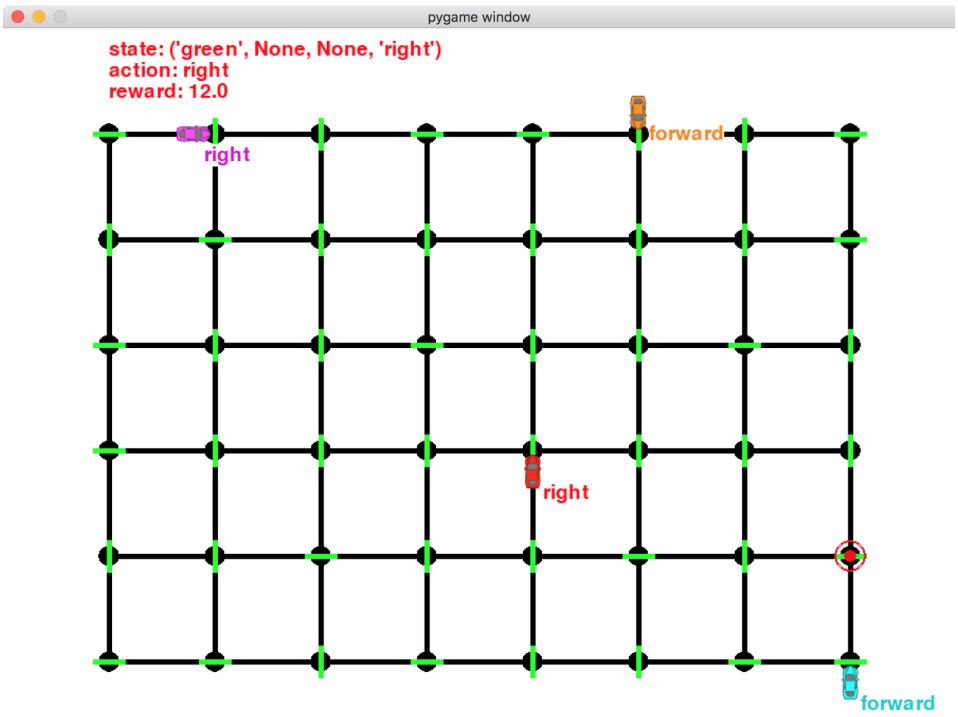
# SMARTCAB PROJECT REPORT

**Yoko Harada**




## IMPLEMENT A BASIC DRIVING AGENT

The first task is to move a red smartcab to directions of None, 'forward', 'left', 'right', randomly. The number of trials to make the destination is unlimited (enforce_deadine=False) unless it hits hard limit.

***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:** To see what the result of each trial easily, the debugging output was commented out. The rest of all printed out lines were saved in the file basic_driving.txt. When the agent successfully reaches the destination the message:

```
Environment.act(): Primary agent has reached destination!
```

shows up. However, when the agent hits -100 rewards and fails, the message below will show up

instead:

```
Environment.step(): Primary agent hit hard time limit (-100)! Trial aborted.
```

After 100 Trials, the result was:

```
$ grep reached basic_driving.txt |wc -l

68

$ grep aborted basic_driving.txt |wc -l

32
```

From the number of lines above, the agent eventually makes it to the destination in about 60-70% success ratio.

For comparison, the trails with enforce_deadline=True option (limits to 5 times of Manhattan distance between start and destination) succeeded 10-15%. Under the constraint of deadline, it's much harder to reach to the destination by randomly choosing next actions.

## INFORM THE DRIVING AGENT

The second task is to add a reporting feature to the agent so that agent keeps meaningful information as well as shows it on the window instead of None. The agent's `self.state` is responsible for the report. Setting some value to this instance variable makes the change. In this task, `enforce_deadline=False` setting doesn't change.

***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?*

**Answer:** The information about light color, oncoming traffic, left side traffic, waypoint, are set to the state. These parameters are added for the reasons below:

- Light color: This decides to choose None/right or forward/left/right as a permitted action. On red, the choices of actions are None/right. On green, those are forward/left/right.
- Oncoming traffic: When the agent is heading to forward/left, this parameter matters. Only when the value is None, forward/left actions are permitted.
- Left side traffic: When the agent is making a right turn on red, this parameter matters. Only when the value is None, right action is permitted.
- Waypoint: This is a suggested next waypoint, the direction to head. The waypoint is based on heading parameter calculated in Environment. The value considers other cars' next waypoints. The value should be weighed.

All four parameters contribute to improve learning experience. The current inputs include a right

side traffic information. However, under the assumption that every intersection has a traffic light of red or green, the right side traffic doesn't matter based on the US traffic law. From this reason, the information about right was not added to the state report.

The deadline is another state the agent may use. The value of deadline starts from 5 times of manhattan distance between start and destination ranging 20 to 55, and decrease one by one in every iteration. The deadline is a kind of a time left. If there's a speed factor, the deadline might have been a state to be included, for example, the agent should speed up because it doesn't have much time. However, in this case, the time left doesn't matter to make a decision. For this reason, the deadline was excluded.

The environment has a location by (x, y) coordinate and heading information as well. However, the problem is: given dynamically changing states, an agent should find the best way to proceed. Actual locations on the map doesn't matter in this case.

***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:** Parameters defined in Environment are:

- light: 2 red/green
- oncoming: 4 None/forward/left/right
- left: 4 None/forward/left/right
- right: 4 None/forward/left/right
- waypoint 3 forward/left/right (unless on the destination)

A possible number of states is: $2 \times 4 \times 4 \times 4 \times 3 = 384$. However, the right is not included in the agent's state. Total number of states used to learn is: $2 \times 4 \times 4 \times 3 = 96$. If we think about what we should care at an intersection while driving, the number of states seems reasonable.

## IMPLEMENT A Q-LEARNING DRIVING AGENT

Next task is to implement a Q-Learning algorithm given the state and action of the agent. The state is the agent's state mentioned in the previous section. The actions are None, 'forward', 'left', 'right'. This time, the next action should be the best instead of random. The deadline setting is now changed to, `enforce_deadline=True`.

Q-Learning Algorithm:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max Q(s', a') - Q(s, a)]$$

$Q(s,\ a)$ : *q value at state s and action a*
$\alpha$ : *learning rate*
$r$ : *reward*
$\gamma$ : *discount factor*
$maxQ(s',\ a')$ : *max q value among all possible action at next state*

***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**: Tentatively, parameter settings are fixed to: $\alpha = 0.5,\ \gamma\ =\ 0.8$ Using these fixed values, 100 trials have been done. When the agent successfully reached to the destination, the message is the same as the first section. However, the failing message has been changed to the below since deadlines are active in theses trials.

```
Environment.step(): Primary agent ran out of time! Trial aborted.
```

Like in the first section, counting lines of success and failure, 98 out of 100 reached to the destination with deadline limitation. Absolutely, choosing actions based on Q-learning improved the performance. The randomly chosen actions got a poor success rate with deadline (at most 15%), also took long time to reach to the destination.
In this trial, the failure happened at the first and 38th trials. The first time trial failure makes sense since the learning has just started. The second failure happened before 50% trials finishes. This shows Q-learning worked well.
However, the agent sometime showed not optimal moves. For example, it circled back to the same intersection or headed to the opposite direction. A failure happened when such wrong moves repeated. Luckily, in this environment, other cars' dynamically changing inputs are involved in. Even though the agent circles back to the same intersection and is given the same next waypoint, there's a great possibility that a combination of light color, oncoming and left traffics are not the as the last time. In this 100 trials, agent didn't circle forever. The dynamically changing environment would be the reason the agent made a good success ratio with random $\alpha,\ \gamma$ parameters.

## IMPROVE THE Q-LEARNING DRIVING AGENT

The last task is to improve Q-learning by testing combinations of the learning rate (alpha), the discount factor(gamma), and the exploration rate(epsilon).

***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?*

**Answer**: The results are in the table below. Since the combination of states dynamically changes, the 100 trials of the same parameter set has been tested four times. The table shows learning rate, discount factor, epsilon policy, an average and range of success ratio, cumulated penalty and extra blocks.

The epsilon policies are:

- 0 - always choose the best action
- 1 - when an iteration is on 0, 5, 25, 125, 625, 3125 times, choose randomly otherwise the best action
- 2 - only when reward is -1, choose randomly otherwise the best action

The cumulated penalty is a sum of all negative rewards. This parameter shows a degree of bad decision. The extra blocks show the average of blocks traveled minus shortest path (manhattan distance from start to destination). This parameter shows a degree of optimal decision.

| α (learning rate) | γ (discount factor) | epsilon policy | success | cumulated penalty | extra blocks |
|---|---|---|---|---|---|
| 0.5 | 0.01 | 0 | 100.00, 100 ⇔ 100 | -21.00, −19.5 ⇔ −23.0 | 6.36, 6.15 ⇔ 6.58 |
| 0.5 | 0.01 | 1 | 99.00, 100 ⇔ 98 | -22.38 −16.5 ⇔ −28.5 | 6.47, 5.79 ⇔ 7.34 |
| 0.5 | 0.01 | 2 | 98.50, 100 ⇔ 98 | -28.38, −26.5 ⇔ −30.5 | 6.57, 6.36 ⇔ 6.74 |
| 0.1 | 0.01 | 0 | 99.75, 100 ⇔ 99 | -23.50, −21.5 ⇔ −26.0 | 6.56, 6.02 ⇔ 7.06 |
| 0.1 | 0.01 | 1 | 99.00, 100 ⇔ 98 | -27.63, −19.0 ⇔ −33.5 | 6.63, 6.30 ⇔ 7.12 |
| 0.1 | 0.01 | 2 | 97.75, 98 ⇔ 97 | -34.25, −27.5 ⇔ −39.5 | 6.77, 6.50 ⇔ 7.08 |
| 0.01 | 0.1 | 0 | 99.00, 100 ⇔ 98 | -28.13, −21.0 ⇔ −43.0 | 5.97, 5.54 ⇔ 6.43 |
| 0.01 | 0.1 | 1 | 99.25, 100 ⇔ 99 | -22.63, −18.5 ⇔ −26.5 | 6.47, 5.48 ⇔ 6.94 |
| 0.01 | 0.1 | 2 | 99.25, 100 ⇔ 98 | -25.00, −23.0 ⇔ −27.5 | 6.44, 6.02 ⇔ 6.98 |
| 0.5 | 0.5 | 0 | 84.25, 98 ⇔ 65 | -84.85, −23.0 ⇔ −154.5 | 6.71, 6.08 ⇔ 7.20 |
| 0.5 | 0.5 | 1 | 98.50, 99 ⇔ 97 | -35.25, −27.0 ⇔ −47.0 | 7.13, 6.35 ⇔ 8.01 |
| 0.5 | 0.5 | 2 | 98.00, 99 ⇔ 97 | -49.25, −24.0 ⇔ −87.0 | 6.83, 6.62 ⇔ 7.04 |
| 0.01 | 0.5 | 0 | 98.25, 99 ⇔ 97 | -25.13, −22.5 ⇔ −26.5 | 6.75, 6.29 ⇔ 7.48 |
| 0.01 | 0.5 | 1 | 99.50, 100 ⇔ 99 | -25.13, −19.0 ⇔ −28.0 | 6.93, 6.86 ⇔ 7.05 |
| 0.01 | 0.5 | 2 | 98.75, 100 ⇔ 98 | -31.13, −23.5 ⇔ −38.0 | 6.59, 5.81 ⇔ 7.16 |

From the tables above, the best parameters would be:

- $\alpha(learning\ rate)$ : 0.5
- $\gamma(discount\ factor)$ : 0.01
- epsilon policy: 0 - always choose the best action

This combination made stably 100% success ratio. This combination's average and worst cumulated penalty are small, also, the average and worst extra blocks are small. With these parameters the agent worked optimally.

***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**: Yes, the agent could find the optimal policy. The first row in the table above shows the highest average success ratio, the minimum average cumulated penalty, and close to minimum average extra blocks. The one of Q-tables of the first row had values below when 100 trials finished:

```
0, (('green', None, None, 'forward'), 'forward'): 7.3380053234
1, (('green', None, None, 'right'), 'right'): 2.18852220014
2, (('red', None, None, 'right'), 'right'): 2.01157683853
3, (('green', None, None, 'left'), 'left'): 2.00832265107
4, (('green', None, 'forward', 'forward'), 'forward'): 1.75260451326
5, (('green', 'left', None, 'forward'), 'forward'): 1.75
6, (('green', 'left', None, 'right'), 'right'): 1.55377473685
7, (('green', None, 'left', 'forward'), 'forward'): 1.53525943949
8, (('green', None, 'forward', 'right'), 'right'): 1.52943780048
9, (('green', None, 'right', 'forward'), 'forward'): 1.5
10, (('green', 'right', None, 'forward'), 'forward'): 1.0
11, (('green', 'forward', None, 'forward'), 'forward'): 1.0
12, (('red', 'left', None, 'left'), None): 0.0
13, (('red', None, 'right', 'left'), None): 0.0
14, (('red', None, None, 'forward'), None): 0.0
15, (('red', 'left', None, 'forward'), None): 0.0
16, (('green', None, 'right', 'forward'), None): 0.0
17, (('red', 'right', None, 'forward'), None): 0.0
18, (('red', 'left', None, 'right'), None): 0.0
19, (('red', None, None, 'right'), None): 0.0
20, (('red', None, 'right', 'forward'), None): 0.0
21, (('green', 'left', None, 'forward'), None): 0.0
22, (('red', None, 'forward', 'forward'), None): 0.0
23, (('red', None, None, 'left'), None): 0.0
24, (('green', 'forward', None, 'forward'), None): 0.0
25, (('green', None, 'left', 'forward'), None): 0.0
26, (('green', None, 'left', 'left'), 'right'): -0.213368364829
```

```
27, (('green', None, 'left', 'left'), 'forward'): -0.23656650862
28, (('red', 'right', None, 'forward'), 'right'): -0.236750360037
29, (('red', 'left', None, 'forward'), 'right'): -0.239638503904
30, (('green', 'left', None, 'forward'), 'left'): -0.239692701479
31, (('green', 'left', None, 'forward'), 'right'): -0.239771662234
32, (('green', None, 'left', 'forward'), 'right'): -0.240007134098
33, (('red', 'left', None, 'left'), 'right'): -0.240016481254
34, (('green', None, 'left', 'right'), 'forward'): -0.25
35, (('green', None, None, 'forward'), 'right'): -0.25
36, (('green', None, None, 'right'), 'left'): -0.25
37, (('green', None, None, 'left'), 'right'): -0.25
38, (('green', None, None, 'forward'), 'left'): -0.25
39, (('red', None, None, 'left'), 'right'): -0.25
40, (('green', None, 'right', 'left'), 'right'): -0.25
41, (('red', None, 'right', 'forward'), 'right'): -0.25
42, (('red', None, None, 'forward'), 'right'): -0.25
43, (('red', None, None, 'forward'), 'left'): -0.5
44, (('red', 'left', None, 'forward'), 'forward'): -0.5
45, (('red', None, 'forward', 'forward'), 'left'): -0.5
46, (('red', None, None, 'forward'), 'forward'): -0.5
47, (('red', None, 'left', 'forward'), 'forward'): -0.5
48, (('red', None, None, 'left'), 'forward'): -0.5
49, (('red', 'left', None, 'forward'), 'left'): -0.5
50, (('red', None, 'right', 'forward'), 'left'): -0.5
51, (('red', None, None, 'left'), 'left'): -0.5
52, (('red', 'forward', None, 'forward'), 'forward'): -0.5
53, (('red', 'left', None, 'right'), 'forward'): -0.5
```

When a next waypoint and action matches, also the action follows the US traffic law, values are positive.

The optimal policy for this problem would be: stably and successfully reaches to the destination with close to the lowest possible penalty and time. Since the average of extra blocks is equivalent to time taken, the table above shows the optimal-ness.

## REFERENCES

These two were extremely helpful to understand how Q-learning works.


1. https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/
2. http://mnemstudio.org/path-finding-q-learning-tutorial.htm