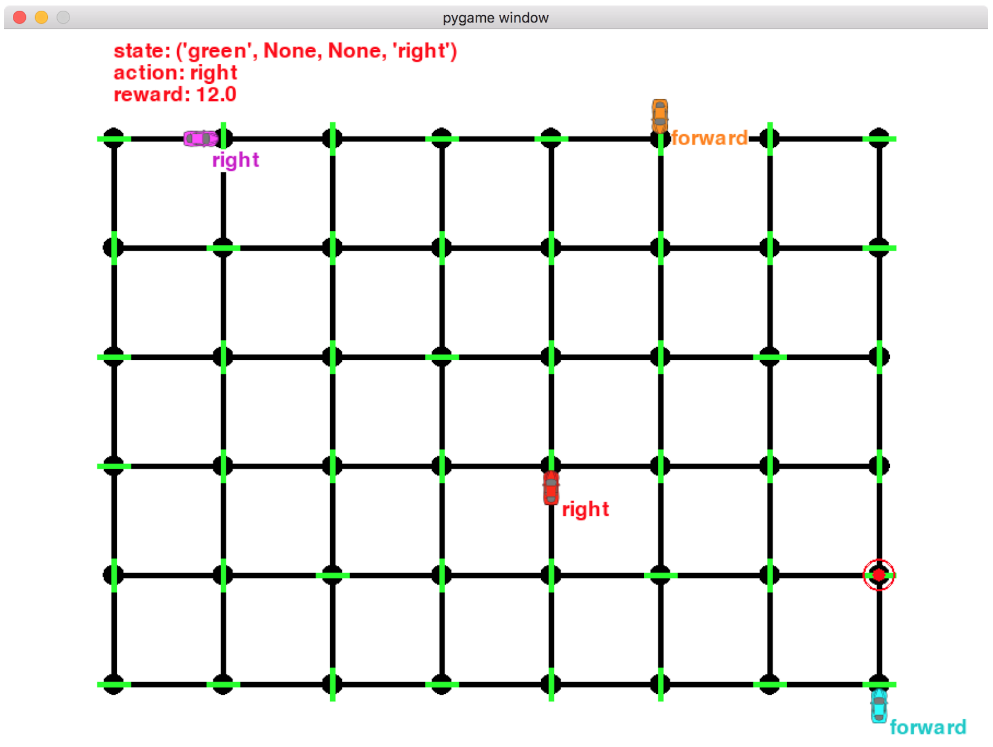
SMARTCAB PROJECT REPORT

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. From this reason, the information about right was not added to the state report.

The environment has location by (x, y) coordinate and heading information as well. However, the problem is: given dynamically changing states, 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, some can be eliminated. For example, right doesn't need to consider. When the light color is red, oncoming doesn't affect. Compared to the all possible states, permitted combinations are less. Given that, 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:

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

$Q(s,\ a)$: *q value at state s and action a*
α : *learning rate*
r : *reward*
γ : *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 smartcar 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 were reached to the destination with deadline limitation. Absolutely, choosing actions based on Q-learning improved the performance. The randomly chosen actions took long time to reach to the destination when it could. Whereas, Q-learning based action selections took much shorter, which was within 5 times of manhattan distance between start and destination, ranging 25 - 55.

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).

The parameters used are:

$\alpha(\textit{learning rate})$: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

$\gamma(\textit{discount factor})$: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

epsilon policy:

- 0 - always choose the best action
- 1 - once every 5 times, choose randomly otherwise the best action
- 2 - only when reward is -1, choose randomly otherwise the best action

The 100 trials has done on every combination above.

***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: Tables below are the best 10 and worst 10 based on the success times out of 100 trials. 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.

The 36 combinations reported 100% success ratio. The best 10 in the table were selected based on the cumulated penalty.

- Best 10

α (learning rate)	γ (discount factor)	epsilon policy	success	cumulated penalty	extra blocks
0.7	0.2	2	100	-18.5	6.05
0.4	0.7	0	100	-18.5	6.54
0.1	0.7	2	100	-20.0	5.92
0.5	0.1	0	100	-22.0	5.39
0.1	0.9	0	100	-23.0	6.88
0.9	0.3	0	100	-23.5	6.07
0.2	0.1	0	100	-23.5	6.47
0.7	0.2	0	100	-24.0	5.9
0.5	0.2	0	100	-24.0	6.15

0.6	0.1	0	100	-25.0	6.11

For comparison, below are the worst 5 combinations.

- Worst 5

α(learning rate)	γ(discount factor)	epsilon policy	success ratio	cumulated penalty	extra blocks
0.4	0.7	2	42	-362.0	6.76
0.7	0.9	1	49	-585.5	10.14
0.9	0.9	1	55	-551.5	12.24
0.4	0.9	2	57	-257.0	8.88
0.5	0.9	1	58	-503.5	9.45

From the tables above, the best parameters would be:

- $\alpha(learning\ rate)$: 0.5
- $\gamma(discount\ factor)$: 0.1
- epsilon policy: 0 - always choose the best action

This combination made 100% success ratio. Although the cumulated penalty is the third, the average of extra blocks is the 4th among all (minimum: 4.97). The agent acted optimally. The first entry in the best 10 table looks optimal; however, the average of extra blocks, 6.05, is the 44th.

***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. After the 100 trials, the agent could find the optimal policy. There was no parameter combination that made the minimum penalty and minimum extra blocks. Some were close but success ratios weren't 100%.
The optimal policy for this problem would be: successfully reaches to the destination with the lowest possible penalty and time. Since the average 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