**Implement a basic driving agent**

Implement the basic driving agent, which processes the following inputs at each time step:

And produces some random move/action (None, 'forward', 'left', 'right'). Don’t try to implement the correct strategy! That’s exactly what your agent is supposed to learn.

Run this agent within the simulation environment with enforce\_deadline set to False (see runfunction in agent.py), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

# Introduction

# Initial Behavior

*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?*

Prior to any updates of the agent.py file, the agent (red cab in the pygame window) doesn’t move. Regardless of the time limit, the cab does not make it to the target destination.

**Identify and update state**

***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?*

* Traffic light value
* Car present (‘oncoming’)
* Previous smartcab action
* Pre-previous smartcab action (2 actions previous)

I chose to use the first two elements as they are critical for avoiding penalties for violating traffic rules. The remaining two elements I chose for their value in informing more useful navigation around the grid (e.g. potentially avoiding circular behavior), which can hopefully help the smartcab to reach its destination more rapidly and earn the successfully completed reward.

Initially I included a longer list of variables (including time, number of steps in specific direction and others), but found that the cab found it difficult to learn from a much wider set of states in the q-table, as the cab was less likely to encounter a new state that could be informed by similar previous state, with multiple actions and their corresponding rewards.

The final set of inputs I’ve chosen seems to strike a balance between sufficient

**Implement Q-Learning**

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

After implementing Q-Learning, I noticed that my agent seemed to improve slightly in avoiding penalties for traffic violations, but that it would also often get stuck in repetitive concentrated travel. For example, the cab would often circle in one corner of the traffic grid. Out of the trial runs, the agent started to more often reach the destination.

**Enhance the driving agent**

*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?*

To optimize my Q-Learning algorithm I added 4 key features:

1. Strict randomization in case of best known q-table rewards below a constant threshold (held to be zero initially, but found strangely that values around -0.5 actually produced better end results).
2. Discounting of rewards achieved at higher values of a “time” index - using a ‘LEARNING\_RATE’ (between 0 and 1) that is raised to a power equal to the time unit and multiplied times the reward of an action
3. Addition of action randomization according to a constant RANDOM\_VARIATION\_RATE similar to the epsilon proposed in later Reinforcement Learning Core Lessons
4. Inclusion of ‘last action’ and ‘before last action’ inputs for the q-table

With these updates, for trial sets (or simulations) of around 50 trials my cab was able to fairly consistently achieve average trial scores of over 5.

To definitively understand how variations to parapers

*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?*

References

Q-tables: <http://www.pysnap.com/reinforcement-learning-in-python/>

Key from value: <http://stackoverflow.com/questions/8023306/get-key-by-value-in-dictionary>

Polyfitting: <http://stackoverflow.com/questions/29885408/how-plot-ployfit-n-log-n-with-python>

Matplotlib formatting:

<http://stackoverflow.com/questions/8248467/matplotlib-tight-layout-doesnt-take-into-account-figure-suptitle>

<http://matplotlib.org/users/text_props.html>

<http://stackoverflow.com/questions/14770735/changing-figure-size-with-subplots>