**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

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

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**

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.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

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

* Traffic light value
* Car present
* Time steps remaining
* Number of steps moved forward in single direction
* Reward

I chose to use the first two elements as they are critical for avoiding penalties for violating traffic rules, and the remaining three elements, as they could potentially be useful in allowing the smartcab to reach the destination rapidly enough to 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.

**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 3 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
3. Addition of action randomization according to a constant RANDOM\_VARIATION\_RATE similar to the epsilon proposed in later Reinforcement Learning Core Lessons

With these updates, my cab was able to fairly consistently achieve end-of-trial total scores of around

*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>