# P4 – SmartCab Project Write-up

## Implement a basic driving agent

## Identify and update state

State could be defined as the inputs into the agent at this time. An example of this is:

State = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}

We also have an input which is the direction that the Planner is trying to send the Smart Cab for the next waypoint. This can be:

Left, right, forward, none

A combination of these could be used to create the state for the cab, for example:

State = {'light': 'green', 'oncoming': None, 'right': None, 'left': None, ‘waypoint’: ‘Left’}

However, in order to work well with Python dictionaries, the format of this state has been modified use tuples, such as:

State = (('lights', 'red'), ('oncoming', None), ('right', None), ('left', None), ('waypoint', 'forward'))

There are other possible environmental variables the could be included in the state, but these have been discounted for the following reasons:

Deadline: This is the time to the destination. This has not been used in the state because:

1. we do not wish for the agent to become reckless as the deadline approaches, for example running red lights
2. it significantly increases the number of possible states making it difficult for the agent to learn the rules of the game in the typical duration of a game
3. The rules of the road, or the rules by which the agent has to drive, do not vary based on proximity to the deadline
4. It could be possible to incentivise the agent to run a red light if the deadline is approaching, but this will likely increase the chance of an accident (see point i).

For the smartcab, I have allowed the state model to emerge over time. I have not set up an initial state space complete with default values as many of these may never happen (for example, we are unlikely in this simulation to see 3 cars at a particular junction).

## Implement Q-Learning

The learnt reward for each state are held in a Python dictionary called ‘q’. This has a key based on the state\_id from the above table.

Each entry in the q dictionary contains 4 values based on the possible actions that the smart cab can take:

{'forward': -1.0, 'right': -0.5, None: 0.0, 'left': -1.0}

The function choose\_action() determines if the state that the cab is in is one that has been seen before. If it isn’t, then a random action is picked from the possible actions (None, forward, left, right), and if it is, then a choice is made from the actions with the highest q value. For example, if left and forward both have a q value of 2, then the agent will pick one of those at random.

What is obvious from implementing this functionality is that the cab reaches the destination much quicker and more purposefully than before. Its direction is generally towards the destination, whereas previous to this functionality, it would wander aimlessly around the environment.

This change in behaviour is due to learning about the rules of the environment and picking the action with the best reward. As the environment rewards for following the direction to the next waypoint, and provides a negative “reward” for not following the rules of the road, the cab learns that the best possible action is to follow the direction of the next waypoint while avoiding traffic accidents.