# 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. This can be:

Left, right, forward, none

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

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

Consideration was given to the duration (e.g. the time left) but this has been discounted as we do not want the cab to take more risky action in order to get a higher score based on a minimal length of time. This could lead to crashes which as a cab passenger we would want to avoid! It also increases the number of states dramatically making learning harder.

However, this still leaves a potential 512 states which is too many for a potential 40-50 step journey. Therefore, the following have been identified as the key states that need modelling (this could potentially be reduced to a smaller set of states but this reflects the driving rules defined for the game in a meaningful way):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Inputs |  |  |  |  |  |
| State ID | Light | Oncoming | Left | Right |  | Planner\_action |
|  |  |  |  |  |  |  |
| 0 | Green | None | None | None |  | Any |
| 1 | Green | Forward | None | None |  | Left |
| 2 | Red | None | None | None |  | None |
| 3 | Red | None | None | None |  | Right |
| 4 | Red | Forward | None | None |  | Left |
| 5 | Red | Left | None | None |  | Right |
| 6 | Red | None | Forward | None |  | Any |
| 7 | Red | None | None | Forward |  | Any |

The following functions have been defined in agent.py:

1. setup\_states() creates a default state table for each of the above state ids
2. determine\_state\_id(input, planner\_action) determines an ID from the table above

As a Pandas DataFrame is used to hold the states, I have also included agent\_action and reward as columns in the DataFrame. This is so that the agent can determine the possible actions and the possible rewards for any state.

The state is updated by the following line in the update() function in agents.py:

self.state = (self.next\_waypoint, inputs)

OR…

If the light is green and everything is None than that’s state 0 (and the reward is the same as following the planner)

It only really matters then if there’s another car coming, then there are different states and the rewards are based on what the planner wants to do and what the other cars are doing (and then I guess some don’t matter??)

Or if the light is red and again it’s dependent on what the other cars are doing…

So define what these states are…

Get the cab to define and learn them as it goes along…

If light = green then state =0 and reward = what the environment (?) gives back… (it doesn’t matter what the other cars are doing??) actually it does if you want to turn left and oncoming = forward

if light = red then different states… see description… but go with the rewards from the environment to define what they are perhaps???