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?

The agent moves randomly over the grid. Occasionally it reaches destination within the deadline, but quite rarely as the instruction it is given are random. With enforce_deadline = False, the random agent ends up getting to destination but takes very long because it has no learning from previous trials to leverage on.

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?

The inputs I chose are the following:

- Light
- Oncoming
- Left
- Right
- Next waypoint

I chose these ones as it seems to me that the state is defined by such relevant information as the traffic light, the presence or absence of other cars and the most suitable next position according to our GPS.

The next_waypoint is important as it is the only input variable that can inform the agent about the (relative) position of the destination. inputs['left'] is important because the agent has to yield to the car on the left in certain cases (e.g. when the traffic light is green for the car on the left going straight, and the agent is trying to turn right). A similar argument works for inputs['right'] and inputs[‘oncoming'].

I chose to omit the deadline in that it would increase the size of the possible state space, which turns quite costly in reinforcement learning, hence it needs to be done only when it is necessary.

I believe it is necessary to take into account the traffic light, the presence of other cars on the way and the most suitable next position according to our destination, but I don’t believe it is necessary to take into account the deadline because our GPS will compute the optimal route and do the smallest number of moves needed.

This implies that if such number of moves is bigger than the deadline, the car won’t get to its destination anyway and if the number of moves is smaller than the deadline then it will get to destination but this doesn’t change the optimal route for this game. Therefore, I do not see

much added value by adding the deadline, which makes me to opt for reducing the possible state space by omitting it.

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?

The learned agent gets to destination within the deadline 85% of the time, across all values of alpha and epsilon. It is certainly more often than the random agent.

When enforce_deadline = False and epsilon = 0, sometimes it gets stuck in some point and doesn't move for a very long period, sometimes until I stop by brute force. This is due to the fact that it reaches some local optimum from where it doesn't want to move. Specifying some epsilon parameter such that with probability of epsilon the agent performs some random action possibly gets the agent "unstuck" from such local optimum.

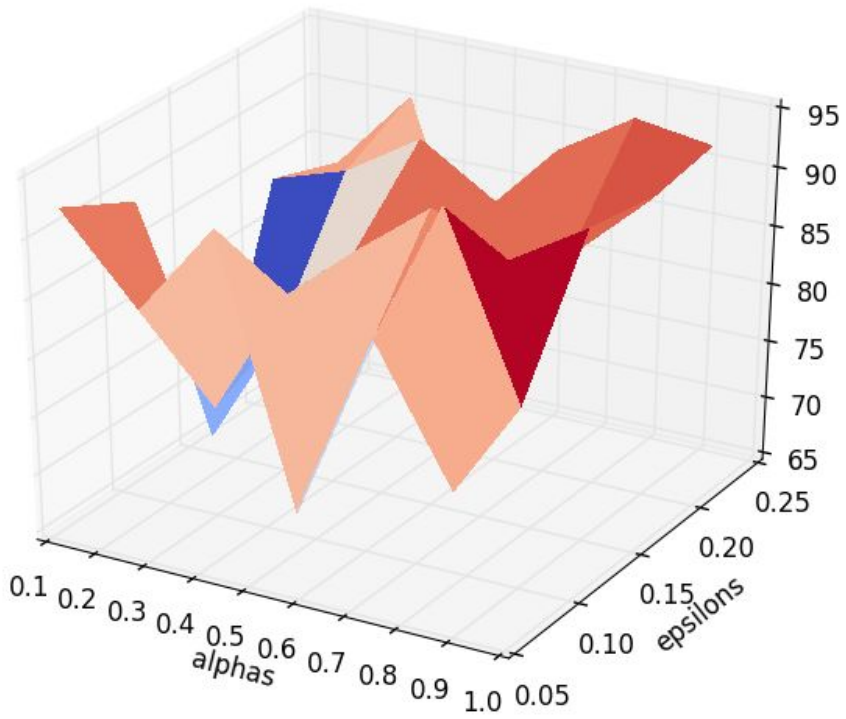
The agent learns from previous trials through the rewards and penalties it obtained through previous observations. At the beginning, the agent leans towards a more exploratory approach in that any unvisited state has reward = 10, thus stimulating the agent to visit new states. In later simulations, when states have been receiving "real-world" rewards, it tends to lean towards exploitation rather than exploration as the initial over-inflated rewards are not there any more, which makes the agent to rely on the "real-world" ones.

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?

The value of alpha and epsilon I tried are the following:

```
alphas = np.array([0.15, 0.30, 0.45, 0.60, 0.75, 0.90])
epsilons = np.array([0.05, 0.10, 0.15, 0.20, 0.25])
```

The set of parameters which works best is alpha = 0.75 and epsilon = 0.1. The final driving agent with such parameters obtains 95% success rate. Please see image below:



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?

An optimal policy would be a policy such that the agent stops when the traffic light is red, drives when it is green, obey traffic rules and takes the actions that leads towards the shortest possible path.

One concrete way to at least get to destination by taking the shortest route is to set action = self.next_waypoint, which I did in the case of the optimal_agent.py. It should be no surprise that it yields 100% success rate irrespective of the alpha learning rate. However, the problem I see with this is that the agent respects the traffic light but doesn't respect traffic, hence it bumps into other agents. An optimal agent would not do that.

My agent does get close to finding an optimal policy but sometimes it wanders around without going straight to destination by taking the shortest route and sometimes it bumps into other cars. Solving this would fill the gap between my agent and the optimal agent.