# Smartcab Project

## Implement a Basic Driving Agent

**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 driving agent moves around at random, regardless of traffic lights, oncoming traffic or its ultimate destination. If given an unlimited amount of time, then the smartcab will eventually reach its destination. However, its course is naïve and random, so it tends to take a very long time before reaching its destination.

## Inform the Driving Agent

***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?*

In order for the smartcab to make informed decisions and to properly associate actions with their associated rewards, the smartcab must have a state that incorporates the next waypoint, the traffic light, and the direction of traffic in the left, right and oncoming directions. In total, this represents a potential of 384 possible states (2 lights * 3 waypoints * 4 right * 4 left * 4 oncoming). This is a large number of possible states to be explored; however, we have a significant number of trials to learn from. Assuming that the average deadline is 30 actions, then we would have the opportunity (but not guarantee) to explore all states within 15 trials. Considering we are running a 100 trial training simulation, it seems reasonable that by the end of the simulation the LearningAgent will be well informed. However, this approach may take significant training before it converges to an optimal policy for all states, and requires we store a large number of (State, Action) pairs in our Q-table, which could present issues with scalability in the future.

In order to increase the LearningAgent's scalability and reduce training times, we could simplify the states space by incorporating specific traffic situations into the LearningAgent. For example, we can simplify the above 384 possible states into the following 6 super states:

- State #1: Stop On Red – light is red and waypoint is not right

- State #2: Right on Red – light is red, waypoint is right and left traffic is not forward
- State #3: Yield Left Turn – light is green, waypoint is left, oncoming is right or forward (*Note: it appears that this traffic rule is not implemented in the simulator*)
- State #4: Left – does not match yield criteria from States 1-3 and waypoint is left
- State #5: Right – does not match yield criteria from States 1-3 and waypoint is right
- State #6: Forward – does not match yield criteria from States 1-3 and waypoint is forward

By simplifying the number of states, we dramatically reduce the number of (State, Action) pairs that we need to map in the Q-table: from (384, 4) to (6, 4). However, we should note that this requires us to embed domain knowledge into the LearningAgent itself, rather than solely embedding the domain knowledge within the environment via the rewards system.

## Implement a Q-Learning Driving Agent

**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 informed smartcab very quickly stops making mistakes and starts efficiently following traffic rules; whereas the random car does not learn from its experiences and continues to move around randomly. The informed smartcab is able to learn from its past actions because we are updating the results of those actions (the rewards) into the Q-table. Once the Q-table has been updated, either positively or negatively, the smartcab can then poll the table once it enters a new state and pick the action that will provide the highest expected reward. Over iterations, this pushes the smartcab to choose actions with the highest positive rewards for any given state.

## Improve the Q-Learning Driving Agent

***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?*

In order to identify optimal parameter settings for alpha, epsilon and gamma, we ran a grid search over values from 0 to 1 incrementing by 0.2 for each parameter and measured performance based on the mean number of steps remaining until the deadline when the destination was reached (or zero in the event the destination was not reached). The results of those simulations revealed that the LearningAgent performed significantly better with an epsilon value of 0, but was relatively insensitive to the values of alpha and gamma. This conclusion seems reasonable for epsilon, as it represents a tradeoff between exploration and exploitation. However, our environment is essentially deterministic; meaning that there is only one clear optimal action for each state, which once discovered should be tightly adhered to. Similarly, the insensitivity to alpha and gamma can be explained because the system never presents the smartcab with a situation in which it would have to break the traffic laws in order to successfully complete its mission. In such a case, the smartcab would have to learn whether to trade a short term negative reward for the long term bonus reward; a decision that would be calibrated by alpha and gamma.

***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?*

Yes, the agent does a very good job of finding an optimal policy. In this case, the optimal policy is most often to simply obey the traffic laws and otherwise move towards the next waypoint. Other than the binary award for reaching the destination, there is no immediate penalty (or reward) for timeliness, which means that the smartcab does not get rewarded for reaching the destination in a faster time – only that it reaches the destination within the deadline. Additionally, given that the simulator is programmed to set the deadline to five times the L1 distance (minimum number of actions between start and destination), the smartcab will almost never encounter a scenario in which it obeys the traffic rules, moves legally along the waypoints and does not reach the destination in time – the smartcab would have to spend four out of very five actions at a Stop on Red in order for this to happen. Therefore, the smartcab views the optimal policy to wait on hard red lights (0.0 reward), rather than to ever break a traffic law to reach the destination faster (-1.0 reward) because it never truly faces a tradeoff between breaking the traffic law and reaching the destination on time.