SmartCab Project Report

*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 random smartcab arrives at the destination in many cases, though it can take quite a while! This reminds me of the book, *A Random Walk on Wall Street*. Randomness will be key to learning.

The (x,y) coordinate system of the game area starts in the upper left at (1,1) and increases to the right and to the bottom. The axes wrap around in both the x and y directions. Left, right and forward steps incur a small negative reward (-0.5) if we move further from our destination. The penalty doubles if the move is not allowed. This occurs when

- The stoplight is red and we want to go left, right or forward.
- We want to turn left, but the oncoming car is turning to its right
- We want to turn right, but the car on our left is going straight

We are allowed to travel forward with a green light at all times. It doesn't matter what the car on the right is doing, as they must give us right of way. We earn 2 points if we successfully travel the same direction as the planner suggests, which the code labels as our "waypoint." If we meet our time limit we earn an additional 10 points.

The planner first aligns in the proper x direction or east/west, then moves in the y direction or north/south to the final destination. All motions are unit movements up, down, left or right. Manhattan distance seems more important than Euclidean distance. The planner seems to ignore the ability to wrap around the grid.

*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?*

I track the following:

- The state of all cars (none if a lane is free)
- The recommended waypoint
- The state of the light

The waypoint indicates which way we should travel to approach our destination, a nice abstraction of the difference between the current location and the destination. This is the essential information for figuring out which way to drive.

The state of the cars and the light affect our motion. We need to learn traffic patterns based on reward. We get negative reward when we attempt to violate a rule of the road. This is sufficient information to learn driving in the US on the right, or driving in the UK and Australia on the left. Later we can grow the learning space if we fail to meet deadlines. That wasn't necessary.

*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?*

We have the following variations in our state:

- 4 possible values for each of 3 cars at an intersection
- 2 possible values for a stoplight
- 4 possible values for the waypoint

This yields 256 possible states (4*4*4*2*4) in this environment.

We've reduced the dimensionality of the problem. Initially, we have the current location (48), the destination (48), 4 directions for each of the 3 cars in the intersection (4*4*4 or 64 total), the current time (up to 100), the recommended waypoint (4), and the state of the stoplight (2). That would be 48*48*64*100*5*2*48 or 147.5M states. We can approach a billion states by including the remaining, allotted time. Our reduction from 147.5M to 256 makes the problem tractable and reasonable.

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 agent prefers to sit and do nothing as it learns. When it attempts an improper move, the action is penalized. Revisiting the same state then prefers sitting still, which does not incur a penalty

We need a way to introduce more experimentation. I accomplished this with two modifications. First, I added Gaussian noise to the initial Q values. This effectively introduced randomness in newly encountered states. Next, I tweaked the "max" algorithm to randomly choose between actions that yield the same Q values. This worked so well, in fact, that I didn't bother with epsilon!

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?

I tested the algorithm by varying alpha and gamma from 0.2 to 0.8 then ran 500 trials with the same random seed. From here I calculated the mean score and the mean number of steps in all successful trials, then divided score by steps to achieve a mean value per step (note: these charts can be produced by running the code):

Mean Value Per Step by Varying alpha, gamma in our Q-Learning Agent

	δ = 0.2	***δ = 0.4***	***δ = 0.6***	***δ = 0.8***
α = 0.2	*1.806*	*1.841*	*1.870*	*1.799*
α = 0.4	*1.829*	*1.881*	*1.857*	*1.848*
α = 0.6	*1.791*	*1.842*	*1.881*	*1.853*
α = 0.8	***1.913***	*1.791*	*1.829*	*1.857*

The best settings had the higher overall mean score, which would choose alpha=0.8 and gamma=0.2. As you can see below, the win rates for these settings are quite close, hence the need to differentiate on just how well they executed each step!

Win Rate by varying alpha, gamma in our Q-Learning Agent

	δ = 0.2	***δ = 0.4***	***δ = 0.6***	***δ = 0.8***
α = 0.2	*0.990*	*0.998*	*0.996*	*0.988*
α = 0.4	*0.994*	*0.988*	*0.990*	*0.990*
α = 0.6	*0.998*	*0.998*	*0.998*	*0.996*
α = 0.8	***0.998***	*0.998*	*0.994*	*0.990*

The first reviewer also asked how well we did on the first 100 trials, which we've shown below. This also seems to indicate that our chosen values are good.

Win Rate by varying alpha, gamma in our Q-Learning Agent

	δ = 0.2	***δ = 0.4***	***δ = 0.6***	***δ = 0.8***
α = 0.2	*0.96*	*0.99*	*0.98*	*0.94*
α = 0.4	*0.99*	*0.97*	*0.97*	*0.98*
α = 0.6	*0.99*	*0.99*	*0.97*	*0.99*
α = 0.8	***0.99***	*0.99*	*0.99*	*0.97*

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?

The agent approaches an optimal policy with a mean value per step of 1.913. Ideally, we'd generate 2 points per every step, following the suggested plan, and not encounter any red lights or traffic. Waiting for an obstruction reduces the mean value per step.

An optimal policy would wait at every obstruction, incurring no penalty, then proceeding according to plan at all other times. The optimal policy would predict this correctly for all 256 states with 0 penalty. Our learner achieved a -0.107 penalty and a 99.8% completion rate.

The learner encountered less than 25% of all possible states, seeing between 53 and 57 states 95% of the time. Penalties usually occur in new states where we don't yet have experience with the result of our actions.

A detour from plan only makes sense when a plan cannot be completed in time. However, we are given 5x the plan length as a step limit. We would have to encounter an obstruction at each step that would induce at least 2-5 step delay on average (as we could at best drive around in 3 steps). A 5-step delay requires a double traffic light, or a traffic light with 2-3 cars in succession driving the right way, among 48 possible intersections, 8-10 times in a row. The probability is negligible.