Brian Cleary
SmartCab report

QUESTION 1: *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?*

By design, forcing the agent to perform a drunkards walk resulted in the agent unable to learn from previous actions or use directions provided by the planner. The agent wasted a lot of our customer's time and money because it was only exploring the space, but not exploiting any information to reach the destination. At each intersection, the agent randomly chose an action—none, forward, right or left.

Relaxing the deadline allowed the agent a hard-limit of 100 actions to reach the destination. Out of 100 trials to reach the destination, the agent reached the destination 59 times. When the deadline was not relaxed, the smart cab was only able to reach the destination 19 times before the deadline expired. This makes sense because when the deadline was enforced, the agent had less than 100 actions to reach the destination. This gives credit to the adage, "a broken clock is right twice a day" because even random actions got the agent to the destination 59% of the time.

I could relate to the poor passengers in the smartcab passing the same landmark over and over because it reminded me of a cab ride I took while in Vietnam—it was an interesting experience.

Agent Random Randy					
Softmax	Initialized Q-values	Alpha (high = more learning)	Gamma (low = current reward)	Epsilon (low = following policy	Success Rate (out of 100)
N/A	N/A	N/A	N/A	N/A	59%

QUESTION 2*: 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 chose inputs from the environment file and the next way point from the planner file—light, oncoming, right, left and next waypoint. The *light* parameter can take on "red" or "green" values. The *light* is a necessary parameter to ensure the smartcab can follow the rules of the road based on red/green lights. The *left* parameter is necessary in the state so that the smartcab will learn what direction the car in the left lane is headed. The smartcab needs the *left* parameter to learn to yield to traffic coming from the left. The smartcab needs to the *next waypoint* parameter in the state so that it can learn where the route planner is telling it to go.

The smartcab may not need the *right* parameter—I did not see any restriction on the actions based on the direction of the car in the right lane. However, if I do not include this in the state, the smartcab will not have access to this information. There could be some benefit that I have not thought of—for example, in the most extreme case the agent is traveling East and at the final intersection before the grid ends, while another car is traveling West and at the final intersection before the grid ends. Would the agent in the East quadrant recognize that the other agent to actually next to it not far away—to his/her right? The simulation is presented as 2-D but since the gird-space actually wraps around, the grid-space is more like a sphere. For example, the agent can exit the top of the screen and re-enter the bottom. Since the space wraps around, it is possible that the agent could benefit from the right parameter in unusual ways—so I chose to add it to my states just in case it could be helpful.

I chose not to include the deadline as a parameter in the state—the deadline parameter would increase the state space drastically and could hinder the agent's ability to learn more relevant parameters quickly and efficiently. Provided the agent can learn and meet the requirements without a deadline, deadline will not be included. However, if the agent is not able to reach the destination within the deadline after implementing Q-learning, I will revisit adding this parameter.

***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 agent became a better driver once I allowed it to use Q-Learning and used the softmax function for the random action. The agent drove more purposefully, although not perfectly. I used self.previous_state, self.last_action and self.last_reward to capture the previous state, action and reward for the Q-Learning equation.

I initialized the values in my q-table to zero, then updated them from the reward of a give state using a softmax function to ensure the q-values were between 0-1 and had the characteristics of probabilities.

This Q-Learning algorithm allows me to tune the **alpha** parameter to allow more or less learning—based on the reward, state and action. A zero alpha means that the agent will learn from the current state but not from the next state. A value of one for alpha means the agent will learn only from the next state and not based on the current state.

Gamma is the value of the discount the agent places on future rewards—when equal to zero the agent will only value current reward. When equal to one, the agent places equal value on the future and current rewards.

Epsilon was created to allow the agent to take some random variation from the policy. I intended to allow the agent some deviation from the policy so that he/her did not get stuck in a "local" maximum by taking actions that increase short term rewards but do not maximize long-term rewards. First, allowed the random choices to be purely random, resulting in a success rate of 72%. Then, I used the soft-max action selection policy because I read that this is better than a greedy or soft action selection policy. Softmax assigns a weight to each action, according to the softmax probability—then, the agent chooses a "random" action in regards to these weights. This makes the "random" choice less random because the worst actions are less likely to occur because of the weighting. Constraining the randomness through softmax policy improved this model's success rate to increase to 83%.

Agent Bob: The baseline Q-learner					
Softmax	Initialized Q-values	Alpha (high = more learning)	Gamma (low = current reward)	Epsilon (low = following policy	Success Rate (out of 100)
False	0	0.40	.40	.40	72%
True	0	0.40	0.40	.40	83%

***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?*

I tuned the initialized q-values, alpha, gamma, and epsilon parameters while continuing to use the softmax function. As you can see in the table below, the best success rate that the agent achieved was 99%. I tried many iterations but by no means did I exhaust all permutations. The parameters for the best model are highlighted below, with a few other models for comparison. In comparison to the baseline learner, the best learner's **alpha** parameter increased to 0.50 from 0.40, the **initialized q-values** in the table increased from 0.00 to 0.50. I lowered the initialized **epsilon** parameter to 0.15 then decreased it further as the trials reached the 10 and 20 thresholds. The **Gamma** parameter was lowered from 0.40 to 0.15—resulting in the agent focusing more on the current state rather than the future state.

Brian Cleary
SmartCab report

Agent Bob: The baseline Q-learner					
Softmax	Initialized Q-values	Alpha (high = more learning)	Gamma (low = current reward)	Epsilon (low = following policy	Success Rate (out of 100)
False	0	0.40	.40	.40	72%
True	0	0.40	0.40	.40	83%
True	0	.30	0.05	0.20	96%
True	0.97	.70	0.01	0.15, then .10 after 10 trips, then .05 after 20 trips	96%
True	0.75	.01	.15	0.15, then .10 after 10 trips, then .00 after 20 trips	96%
True	0.50	.05	.01	0.15, then .10 after 10 trips, then .00 after 20 trips	**99%**

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?*

Q-Learning allowed the agent to make significant strides in learning how to reach a destination quickly and efficiently—with a 99% success rate. This policy is not optimal as the agent still received negative rewards on occasion and took a more circuitous route than was optimal. However, I did not include the deadline in the states, so this may have contributed to the more round about driving—even when it reached the destination within the deadline. I watched how it performed when another car was approaching head-on and eventually, made a right turn while my car made a left. My agent seemed to be cautious about waiting until the other vehicle turned right before turning left—this was a good move and in these circumstances, I am glad it was not paying too much attention to the deadline and a preferred the safer route.

The optimal policy would provide all the relevant state parameters that allow it to obey all road rules and take the most efficient route. Given the current reward structure, the optimal policy would ensure the agent followed the planner unless the action would conflict with the road rules, and if there was a conflict it would do nothing. Perhaps if the other vehicles had a simulated GPS which allowed my agent to know where they were at all times it could plan the perfect route and anticipate traffic further away.