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?*

The cab just moves in a random manner until it resets or reaches the destination. In simulations, the random agent will reach the destination within the deadline some 20% of the time.

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?*
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?*

The states I have identified as being appropriate for the modeling of the smartcab and environment are the inputs: light, oncoming and left as well as the deadline and next waypoint. I decided not to include 'right' for this simulation because the simulator will not allow traffic to run red lights and otherwise disobey traffic rules. With this assumption, that Dummy Agents will not break traffic laws, the 'right' input provides no value when deciding what action to take next.

Moreover, to show that 'right' is of no value I will show how the other states do provide value when deciding what action to take next. First, all actions we can take are: none, forward, left and right.

If at a green light, going straight or turning right does not require any other input other than the traffic light because we have the 'right-of-way'. If at a green light and wanting to go left, we need to know if there is any oncoming traffic to safely take that action (to obey traffic laws). That covers all possibilities for green lights.

If at a red light, we can only make a right turn when the traffic is clear coming from the left direction. This covers all possible actions and what we need to know to drive the car. Therefore, what the 'right' traffic is doing does not matter to us. However, it's important to note this only applies to this smartcab simulation because in the real world cars can make illegal traffic moves. In the real world, 'right' traffic is very important; such that a car runs a red light. This is why it is always good practice to look both ways before proceeding on a green light.

Further, the deadline is important because the car may choose a different action if the deadline is approaching. For example, it may take a right on red when the next waypoint is to go straight. Given the deadline, it may find that turning right on red is faster. Last, the next waypoint is important because it tells us the route we should take ignoring all traffic rules. In other words, it tells us the best way to get there if we were only looking at distance.

OPTIONAL:

The total number of states that exist for the smartcab in this environment is: 512

With the knowledge I have at this point of the project, since the deadline is a discrete variable I chose to turn it into 4 ordinal values. Deadline is a represented as an integer that decrements by 1 per each time slot. The deadline varies based on how far the smartcab is from its destination. The four ordinal values I chose to represent this variable are represent by how much time is left: 1) 100%, 2) 75%, 3) 50%, 4) 25%. This is something I can tune when building my model, but am going to start here.

The reason I scaled the deadline variable is because of the large dimensionality it creates when looking at each point individually in the range of values. For example, if the deadline is 50, it will have 51 total values (including 0). This example would have 6,528 possible states. However, with my 4 point scale there are only 512 possible states.

The scaled number of possible states does seem reasonable given that Q-learning is going learn and make informed decisions about each state. However, not scaling this does not seem reasonable given it would suffer from the curse of dimensionality and would be very costly in terms of computation.

Would a different scaling be better? How about a percentage of time remaining in groups of 10 versus 4? For example, 100%, 90% time left etc. This is something I will test later. It would give us 1,280 possible states.

	Light	Oncoming	Left	Deadline	Next Waypoint
Total number values	2	4	4	4	4
Possible values	Red, green	None, left, right, forward	None, left, right, forward	1) Not rushed, 2) watching the time, 3) better hurry, 4) YIKES!	None, left, right, forward

QUESTION 3: *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?*

When we implement the Q-learning, the agent takes routes more in the direction of the destination. It is somewhat random and chaotic at first but becomes more organized as the trials continue. This behavior is occurring because it is learning what is 'right' and 'wrong' based on the rewards it gets for doing specific actions in certain states.

QUESTION 4: *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 different parameter values used were (alpha, gamma, epsilon, initial qvalue):

I decided to apply brute force and try all combinations for alpha, gamma and epsilon, where each variable could hold the values of 0-1 in increments of 0.1 while the initial qvalue could be 0-10. This gave 11,000 different possible combinations (why not, right?!).

From these 11,000 I sorted the table as follows:

1. (first sort) descending, on last10_success_rate
    a. This is the success rate for the last 10 trials of the 100 trials performed for the set of parameters, expressed as a value 0-1. 0 meaning the agent did not reach the destination at all in last 10 trials(failed) and 1 = 100% reached destination 10 out of 10 times.
2. (second sort) ascending, last10_num_penalties
    a. This is the count of penalties the agent incurred in the last 10 trials of the round.
3. (third sort) descending, last10_med_time_remaining

a. Simply the median of the time remaining when the agent had a successful trip. For example, if there was 15 of 20 seconds left in the deadline this value would be 0.75. Meaning it finished with 75% of the time still remaining. (high numbers are good)

Table 1 Max values

alpha	gamma	epsilon	qvalue	last10_med_time_remaining	last10_num_penalties	last10_success_rate
0.8	0	0.9	5	0.67	0	1
0.9	0.9	0.9	4	0.665	1	1
0.9	0.7	0.6	9	0.6	1	1
0.7	0.4	0.9	5	0.555	2	1
0.8	0.1	0.7	3	0.6	3	1
0.4	0.8	0.8	2	0.585	3	1
0.6	0	0.4	4	0.67	4	1
0.4	0.6	0.8	4	0.62	4	1
0.8	0.2	0.9	8	0.58	4	1
0.9	0	0.5	6	0.555	4	1

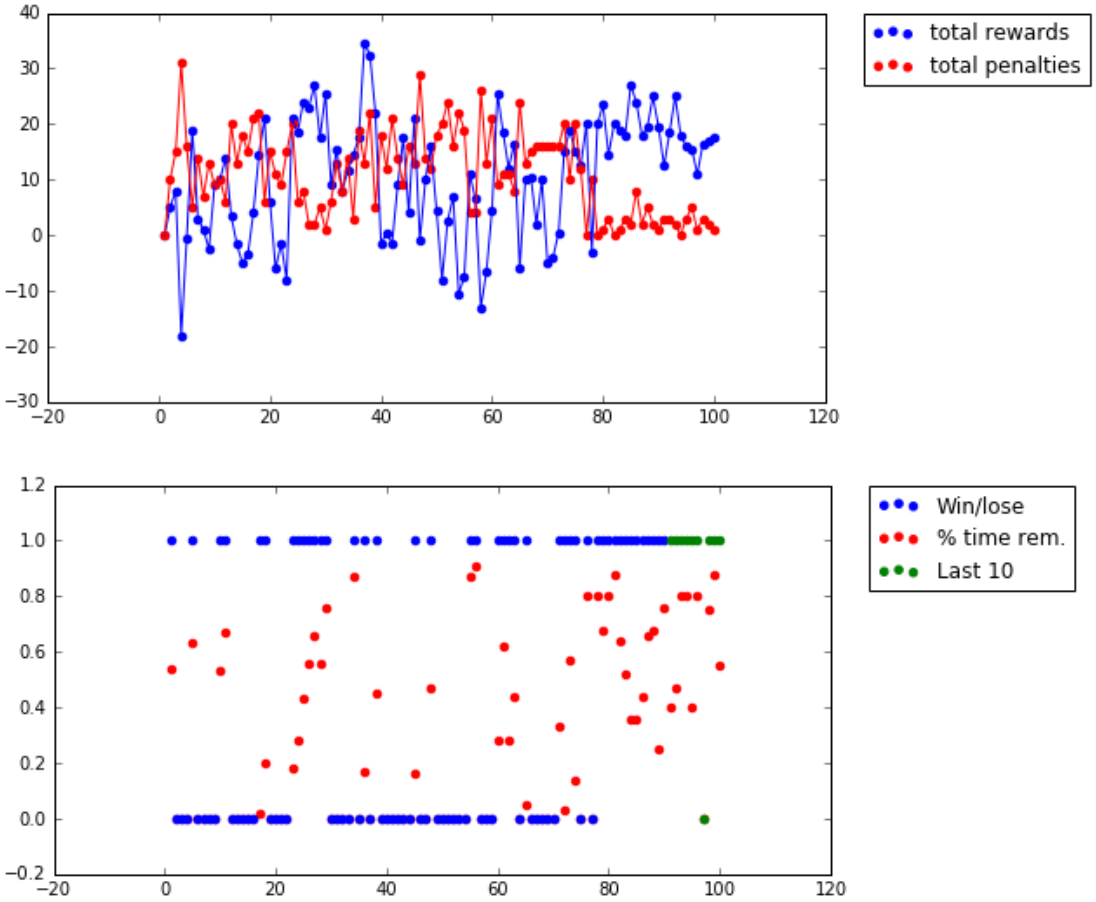
Table 2 Stats of max values

	alpha	gamma	epsilon	qvalue	last10_med_time_remaining	last10_num_penalties	last10_success_rate
count	10	10	10	10	10	10	10
mean	0.72	0.37	0.74	5	0.61	2.6	1
std	0.193218357	0.356058672	0.183787317	2.160246899	0.044845413	1.505545305	0
min	0.4	0	0.4	2	0.555	0	1
25%	0.625	0.025	0.625	4	0.58125	1.25	1
50%	0.8	0.3	0.8	4.5	0.6	3	1
75%	0.875	0.675	0.9	5.75	0.65375	4	1
max	0.9	0.9	0.9	9	0.67	4	1

The (0.8, 0.3, 0.8, 5) performed best. This number was the median of the top 10 best parameters. The final driving agent performs quite well most of the time but has some terrible results sometimes too. I believe this is caused when the agent reaches a state it has not seen before in the last trials of the round. Without being in that state before it has no way to know the optimal action to take, so it will take a random action.

I decided to take the median because the 'optimal' parameters varied some. I ran many of the top 10 results individually and plotted the results to further evaluate them with more rounds in the simulator. They all varied somewhat similarly, again my guess would be due to the agent reaching new states.

Figure 1 Sample plot



I was quite surprised at the result of my top 10 (table 1). From the 11,000 I chose the top 100 and ran them all three times each. If there was consistency in the results I would have seen them show up in the top 10. However, I did not see this. I did see some trends for each of the parameters though.

Alpha ranged 0.4-0.9 with median of 0.8. Gamma had the most variance with a range of 0-0.9, median of 0.3. Epsilon had a range of 0.4-0.9 with median of 0.8. And the qvalue (initial) had range of 2-9 with median of 4.5 (rounded to 5).

More parameters

In addition to tuning the parameters above, I also tried dropping the deadline as part of the state to help with the curse of dimensionality. In this environment, besides a right turn on red, there is not much the smartcab can do to hurry when a deadline is approaching. Even a right-on-red would be gamble to find a way around with all green lights.

I started with 4 possible values for deadline which gave 512 possible states. Dropping the deadline reduced the total number of states to only 128 (from the original 512). With the 4 deadline values I was getting low success rates, approximately under 40%. After dropping the deadline it was closer to 90-95%.

I started out decaying alpha, gamma and epsilon. Unfortunately, at first my code was not correct and it was not decaying correctly. To compound things I also had an initial qvalue of 0. The combination of these two gave very poor results, the agent had a success rate less than 0.2. Once I corrected the qvalue, but before I corrected my decay rate. I tried keeping static alpha, gamma and epsilon rates until the last 10 trials where I would set them all to zero. This also performed quite poorly, which is when I found the error in my original decay rate and ended up with the decay rate = parameter_initial/t, or for example, 0.8/t.

One other thing I would like to try is initializing all the actions when a state is first reached. I think this could provide even better results because if the first action it took in that state was bad, it will now have a negative qvalue and will choose a different action next time (since it is choosing max action). However, my current code just has it choosing a random action until all actions are mapped for that state, then it goes for the max. These are improvements for the future.

QUESTION 5: *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?*

My agent does get close to finding an optimal policy. Towards the end of the trials, it reaches the destination within the time but not the minimum time and does still incur penalites, perhaps due to reaching new states at the end.

An optimal policy for driving this smart cab would be to obey all traffic rules and to reach the destination in the fastest time possible.