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?*

Yes, the cab makes it to its destination eventually. It can take a really long time though, and it's actually not guaranteed to ever reach the destination (intuition — it is possible for the random sequence of moves to just all be left turns, although this is ridiculously unlikely). This is actually limited to 100 steps past the deadline in `environment.py`. The car makes no attempt to minimize the length of its route, or to avoid other vehicles, it just eventually stumbles onto its destination.

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 appropriate parts of the smartcab's state are `'light'`: the light at the current intersection, `'oncoming'`: the presence of oncoming traffic, `'left'`: the presence of traffic approaching from the left, and `'next_waypoint'`: the current direction of travel. `'light'` tells us whether or not we can proceed in the current direction or turn left, depending on the actions of other cars. `'oncoming'` can restrict whether or not we can turn left given a green light. `'next_waypoint'` differentiates the current direction of our movement. Our current orientation decides where we will end up given an action and our current position.

It's worth noting that `'right'` is not necessary. On a green light, there is no restriction on our movement due to traffic approaching from the right. On a red light, the only move we're allowed to make is a `'left'` turn, but this again is unaffected by approaching traffic from the right.

I have also chosen to keep `'deadline'` out of the state. My reasoning is that whether or not the deadline has actually passed should not affect the optimal strategy for maximizing the agent's reward. In general, the agent is attempting to minimize travel time, the approach of the deadline should not affect this motive. Further, with `enforce_deadline` set to `True` the simulation will end as soon as the deadline has been missed. Including the actual deadline value would increase the size of the state space by a very large amount, especially if the deadline can be arbitrarily large. This would make Q-learning much more difficult. Keeping a binary value that tells us whether or not we have failed to reached the deadline is also not helpful as stated previously; the failure of the trial will give us just as much information as keeping a binary variable would.

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?*

There are 2 states for `'light'`, 4 states each for `'oncoming'` and `'left'`, and 3 states for `'waypoint'`. In total there are $2 * 4 * 4 * 3 = 96$ possible states. There are 4 possible actions, making a total of 384 possible states. This is a relatively small number of states, and should be very amenable to Q-learning.

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 agents movements are much more measured. They are clearly no longer random, as the agent now continues along the same path much more often, and seems to gradually move closer to the target. In general, the agent converges on the destination in much less time than it did with the random action selection.

This is due to the presence of an adaptive policy. The agent will generally act in a manner that has historically maximized utility, while occasionally taking a random action to explore the unknown.

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 ran 1000 trials for each setting of parameters.

Here γ is the discount rate, α is the learning rate and ϵ is the exploration rate (**NOTE:** in my code I used $1-\epsilon$ and called it the exploration rate, it just seems more intuitive to me that way). 'avg. reward' is the mean total reward over all the trials. 'avg. steps' is the mean number of steps taken to get to the destination. 'missed deadline' is the percentage of time that the smartcab does not make it to the destination in time. `'violations'` is the mean of the mean number traffic laws broken for each trial.

γ	α	$1-\epsilon$	avg. reward	avg. steps	missed deadlines	violations
1	1	1	0.269	27.678	80.1%	7.526
1	1	0	0.327	27.648	81.1%	7.454
1	1	0.5	-0.0275	27.929	82.4%	7.532
1	1	0.25	0.1015	27.536	80.6%	7.454
1	1	0.75	0.5515	27.336	79.6%	7.287
1	1	0.125	-0.068	27.951	81.0%	7.568
1	1	0.0625	0.643	28.135	79.9%	7.418
1	1	0.3125	1.0185	27.336	76.3%	7.291
1	0.5	0.03	22.5955	13.042	1.1%	0.163
1	0.25	0.03	22.157	13.108	1.2%	0.184
1	0.75	0.03	21.109	14.532	4.6%	2.582
1	0.125	0.03	22.385	13.274	1.0%	0.193
1	0.0625	0.03	22.4475	13.775	1.0%	0.213
0.5	0.0625	0.03	22.358	13.697	1.6%	0.271
0.25	0.0625	0.03	22.3385	13.387	1.6%	0.269
0.75	0.0625	0.03	22.413	13.522	1.3%	0.218
0.625	0.0625	0.03	22.316	13.289	0.9%	0.248

My final parameter selection is 0.625 for the discount factor, 0.03 explore rate (i.e. 0.97 is ϵ) and 0.0625 learning rate.

I ran a further 10,000 trials with this parameterization, and found an average reward of 22.3349, 13.11 average steps, 0.85% of deadlines missed and 0.1543 violations per trial.

The vehicle makes it to its destination before the deadline about 99.15% of the time. It has a traffic violation in about 2 in 13 trials. The total reward is between 22 and 23, and it takes about 13 steps to reach the destination.

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?*

On our grid, the average distance between 2 points will be 2 in the vertical direction and about 2 in the horizontal direction. That means that if the car going in the right direction, it can get there in 4 moves on average. If going in the wrong direction, the car can turn around by moving in a loop, which requires 4 moves, for a total of about 8 moves. That means that in the average case, the car should be able to reach its destination in about 6 moves. In truth, the lower bound should be slightly higher, as I haven't factored in the possibility of waiting at a red light as part of the optimal strategy (I've made the implicit assumption that at every turn the cab can move 1 intersection closer to the destination).

So with 6 moves established as a theoretical lower bound, how well does our smartcab do? With the parameters I ultimately chose, it makes about 13 moves on average. This is close to 2 times what the car could do if it was omniscient. Our smartcab cannot be omniscient however. Considering that the smartcab is learning its way as it goes along, I think that this performance is great! Recalling that the number of possible state-action pairs is 348, by the time the cab has made ~13 moves, it can't really have full information about the costs and rewards of its actions yet. The fact that it can on average make it to the destination with only about twice as many moves as strictly necessary, having seen only a small fraction of the state-action pairs possible means it is performing very well. Further, the cab only misses it's deadline about 0.85% of the time.

An optimal policy for this problem would always make the right move, the one maximizes utility or expected reward. In this case, the optimal policy is to almost always make a move one step closer to the destination, when such a move is legal. At other times, it may make sense due to traffic conditions to take a longer route, since the goal is not to optimize trip distance, but trip time. Thus the shortest distance does not necessarily correspond to the best path to take.

An optimal agent will certainly not commit any violations. In my code, I could make my agent avoid illegal actions in the update method by checking if the chosen action is illegal and making it choose again until it selects a legal move, but this feels like "cheating" and seems like it would be against the spirit of the assignment. In the real world, the vehicle would not necessarily have *perfect* information about the intersection that it's at, and driving laws and best practices are probably a lot more complex than can be captured in a few if statements. It is still necessary for our agent to attempt *some* illegal moves so that it learns which moves are actually illegal.