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?

So far, the cab (as expected) moves about at random. Sometimes it happens upon a destination, and I imagine with sufficient running time and no obstacles it will always *eventually* hit the destination given infinite time, but there's also no assurance that it will do so within any finite timeline no matter how large you make it.

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?

I believe the things that should go in local state for the agent should be bits of data that are useful for deciding what to do next. In addition to the desired direction of travel from the planner, almost every input qualifies with the exception of what any car to the right in the current intersection is planning to do. We need the plan of travel because the direction of the destination informs which way we would prefer to travel; without this bit of information we wouldn't have a reason to turn left instead of going straight, or any reason to travel any particular direction.

We need to know whether the light is green or red because that limits whether we can take our desired action at that moment. We have to know the status of any cars at the intersection oncoming or to the left, because they can interfere with a desired action. If we want to travel right, we can do so on a red light as long as there are no cars from the left traveling through and likewise.

I can't think of a scenario in which we care about cars to the right, though. If we want to go straight, if the light is green we can go and if red we can't. If we want to go right, on green we just go, on red we care what the car from the left wants to do. If we want to go left, on red we stop, and on green we care what the oncoming car wants to do. In no case do we care about the intentions of the car to the right. Caring about deadline is also not an issue, since regardless of how long is left on the clock we still want to take the optimal action at each step. Additionally, there are a lot of possible deadline values, which would increase the number of states we need to account for in the learning matrix.

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?

This seems like a permutation & combination problem. If we just care about the intended direction of travel, then there are 3 states (since as long as we aren't on the destination right now we will want to travel in *some* direction). We also care about the color of the light, which means we can be in 2 light states (red/green) for each direction we will have in total 6 possible states. Now we add in the state of oncoming traffic, for which there are 4 states (often there is no car there resulting in "None"); this is also independent which means 4 options for each of the existing 6 states, or 24 possible states. Then finally we can account for the state of the car to the left, which can be in any of 4 states, once again independent, so multiply by 4 again to result in 96 possible combinations. The number of possible actions is 4 ("None" being an option if the safety circumstances of the current environment prevent taking the desired action immediately), so that means we have 96 states and 4 possible actions, or **384 state/action pairs**.

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 behavior of the agent improves since it states 'Primary agent has reached destination!' with higher and higher frequency from its first run. Initially the agent does make random movements if and before reaching its destination. It's learning to follow the next_waypoints and training laws, although the agent acts slightly sporadically due to the epsilon greedy algorithm included. Furthermore it chooses actions based on an epsilon random chance, else the max argument part of the algorithm. Therefore there's epsilon chance that the agent will always choose an arbitrary action.

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?

The set mentioned in the program performs best, of the sampled parameter combinations, and behaves more rationally than the initial agent when its route is blocked. Once it's gone through around 12 iterations, it begins to behave reliably, and by the time it's gone through the first 30 episodes its traffic infraction rate drops significantly. I suspect over time it performs quite well, so I'm going to run another set of 5,000 trials because I suspect it has converged already and will perform more or less successfully from this point forward.

- Learning Rate: 0.3
- Discount Value: 0.99
- Initial Epsilon Value: -0.008
- Epsilon Degradation Rate: 0.001
- Traffic Infractions: 24

This approach is working ok, but there are some edge cases. When we run into another car at an intersection in the oncoming lane, we tend to lock up. It seems we've learned that it's simply safest to let other cars go, and if both agents decide that then we get stuck. We could up the experimentation rate, but that would result in more infractions and we don't want that. I think the next step might be to have the epsilon value scale out the current q values by random amounts rather than just go away entirely so that if we're stuck for long enough the car will pick another "valid" direction eventually. Another possibility might be to attach a slight negative reward to holding still; less than the penalty of committing an infraction, of course, but high enough to encourage us out of just being parked and waiting. Increasing the discount rate would also help with this. That's probably the easiest to mess with so we'll try that one first.

- Learning Rate: 0.3
- Discount Value: 0.4
- Initial Epsilon Value: 0.1
- Epsilon Degradation Rate: -0.008
- Trials: 100
- Traffic Infractions: 29

This doesn't seem to be helping enough when I scale it up over a large number of trials, we still suffer failure > 10% of the time because of locking up in certain cases. I'm going to revert the discount value and try instead accumulating a small negative reward for staying still.

- Learning Rate: 0.3
- Discount Value: 0.99
- Initial Epsilon Value: 0.1
- Epsilon Degradation Rate: -0.008

- Trials: 100
- Deadlines Missed: 4
- Successful Arrivals: 96
- Traffic Infractions: 28

That seems way better. Let's see how it does scaled up over 500 trials.

- Learning Rate: 0.3
- Discount Value: 0.99
- Initial Epsilon Value: 0.1
- Epsilon Degradation Rate: -0.008
- Trials: 500
- Deadlines Missed: 38
- Successful Arrivals: 462
- Traffic Infractions: 26

We're still missing some deadlines unnecessarily, but fewer, and the infractions are stable. This is probably good enough.

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 it does find a policy that is close to optimal. Because it is aware of obeying intersection rules and going along with direction that minimizes its path distance to destination. The learning is motivated by the reward rule that drive the agent to learn this policy. So the model in the agent is motivated to follow the waypoint suggested by the planner as well as to obey the cross-section rule. Once it follows the planner, it is easy to get to destination within deadline.

I believe this is fairly close, but that an optimum policy would probably make some cleverer decisions when blocked. I think that by re-implementing the epsilon degradation to instead have an action randomly selected from the actions known to not be illegal that could be slightly better. Still, this is fairly close specifically because we care about both not breaking the law/hurting people and getting there on time, but far more about the first than the second thing. If we can get there on time 90% of the time, and never break the law or crash, that's better than a higher time-efficiency rating at the expense of causing traffic accidents occasionally. The most optimal policy (if it exists) would arrive on time 99.9% of the time (when deadline is reasonable) is never crash or break the law.