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?

When it's totally random, it hits the hard time limit of -100 quite a few times. As it's entirely random, it does not care about traffic or the lights, making it an incredibly dangerous driver. Unless every other driver on the road is incredibly skilled, this car is not likely to make it to its destination in one piece! It only has around 20% success rate on average. See tally for details.

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?

Almost all of the input is logically necessary for the state to function, except for the deadline variable and the right traffic. Because the deadline changes depending on the number of moves made, it makes for a poor teacher as far as state goes. The learner is very likely to encounter the same combination of the other variables (oncoming, left, and light), but rarely will it encounter the same combination of the others *and* the deadline, making for an exceedingly poor learner. Kind of an equivalent to overfitting. However, adding on one more variable, `next_waypoint`, should be a good thing, because we want the qlearner to associate good things with getting closer to the target position (which is the only place that `next_waypoint` will be equal to `None`, so we can safely leave that out). The one other variable we can probably get rid of is actually the right traffic input. According to the specifications of the project, we never have to worry about them.

QUESTION: 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 four variables in the state. Two of them have four possible values, one has three, and the last has only two. `Oncoming` and `left` both have four possible values, `None`, `forward`, `left`, and `right`. `next_waypoint`, while it technically has four, we can safely ignore one of its values completely, `None`. Because `None` only shows up when the destination has been reached, we can safely use only the last three variables. Finally, there is the `light` variable, which has only two possible values: `red` and `green`.

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?

I notice that it fails a lot less, firstly. On top of that, it gets better over time; in the beginning it takes many random actions, but as it goes on, it improves until it's near perfect. This behavior is occurring because it's learning the appropriate action for every possible scenario, and takes into account the `next_waypoint`, which always points to the best way to get to the destination.

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?

See the tally folder for more details, but the best values I found were only marginally separated:

- gamma = 0.2, alpha = 0.4, epsilon = 1.1, success = 93.890625%
- gamma = 0.5, alpha = 0.5, epsilon = 1.5, success = 93.515625%

In the tally folder, you'll find output from 64 runs of each set of parameters. The exact command used for each variable modification was:

```
for f in {1..64}; do python smartcab/agent.py; done | grep 'Primary agent has
    reached destination' | wc -l >> tally
```

I incremented the number at the end of tally for every unique parameter combination, and manually modified the values at the beginning.

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?

An optimal policy for this problem is one that follows *all* traffic laws and *always* takes the most efficient route to the destination within those constraints. I created a test to see if it was creating the optimal policy by testing my qlearner against a 'perfect' actor (one that follows all rules of the road and proceeds as efficiently as possible to the destination). I found that with the first `gamma`, `alpha`, and `epsilon` values listed above, it chooses the *perfect* action 94% of the time (assuming that you only measure the last ten runs, as that's when it should most follow policy). This means that the qlearner, after 90 runs and above, will follow traffic laws or do the most optimal thing 94% of the time. Please note it's not necessarily breaking the law, it could also just be that it does not take the fastest route sometimes. You can test this on a large number of runs (similar to my variable modification above) by running:

```
for f in {1..64}; do python smartcab/agent.py; done > out \
grep 'p.act' out | wc -l \
grep 'q.act' out | wc -l
```

That will give you the raw number of perfect acts (p.act) and the number of qlearner acts (q.act). Divide them by each other to find the exact percentage for that run.

A perfect policy is implemented in the code under the perfect actor comment. You can test it by acting on `paction` instead of `action`. It is defined as one that follows all traffic laws and gets to the destination as soon as possible.