*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?*

1. **Prior** to implementing the random choice among possible actions `(None, 'forward', 'left', 'right')`, the red car stayed in place. This makes sense, because previously the action for each iteration was always `None`.
2. **After** implementing the random choice among possible actions, as expected, the car began to move, and yes, it eventually makes it to its destination (but usually not before the deadline expires)
3. After listening to the lectures on Q-Learning, I admit I **expected** for the car to reach its destination "better" (e.g. more directly) on subsequent iterations, but then I noticed `# TODO: Learn policy based on state, action, reward`, and realized that learning will come later.
4. It's a little hard to tell, but it also appears like the agent is, at this point, doing wrong things (e.g. disobeying traffic laws in ways that are likely to bring harm to itself and others). From what I could tell in `environment.py`, such harmful actions are classified as `Invalid Moves`, and result in a `reward` of `-1.0`. There were definitely a few `rewards` of `-1.0` in the simulation.
5. So, at this point, it seems like we've successfully given movement to an actor which is likely to (unintentionally) inflict harm. Yikes!

*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?*

Every state in `inputs` (that is, in `self.env.sense(self)`) is important for modeling the **smartcab** and its environment.

1. The state of the light (e.g. `red, green`) is required to know whether the desired next action is presently allowed.
2. Similarly, the location and heading of other cars is important for knowing whether or not our `agent` needs to adjust its desired heading. Without knowing this information, our `agent` will be unable to perform **collision avoidance**.

*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?*

1. From mere permutations of the `inputs` vector, there are 128 distinct states. This is because there are two states for `light`, and four for each of `oncoming`, `right`, and `left` (specifically, `None`, `forward`, `right`, and `left`).

```
{'green', 'oncoming': None, 'right': None, 'left': None}
{'red', 'oncoming': None, 'right': None, 'left': 'left'}
{'red', 'oncoming': 'right', 'right': None, 'left': 'forward'}
```

2. This high number of states seems correct for the `agent` to have a full understanding of

2. This high number of states seems correct for the agent to have a full understanding of the intersection and its possible next actions. Without an understanding this full, it seems unlikely that a **smartcab** would be able to explore alternative routes in case the chosen one is blocked.

    - For instance, consider a smartcab approaching a green light, correctly intending to go straight through the intersection. If another car approaches the intersection from the smartcab's right, intends to go forward, and should stop at the red light but doesn't, the smartcab must correctly ascertain that it has to stop, or else it will contribute to an accident it could otherwise have prevented, even though it's not the case that it caused the accident all by itself.
    - Even though there are 128 distinct possible states, many of these states are invalid, since they would cause an accident even in the absence of the smartcab. However, even strange accidents happen occasionally, so while the likelihood of many of the potential states is low, it's not zero, so the smartcab can't assume they'll never occur.

***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?*

1. Previously, the smartcab was only taking random actions, and unfortunately no mechanism was in place to choose better on subsequent trials. Now, however, although the smartcab begins by taking random actions (many of which are invalid), by the third or fourth trial, the number of invalid actions approaches 0, and the smartcab appears to take a fairly direct route to the destination.
2. Interestingly, by the 7th or eighth trial (though perhaps sooner), the smartcab develops a tendency to loop, and never remains at an intersection. While comical, this is probably also not an ideal policy in the abstract, since it's wasteful of gas, increases wear on the vehicle, and greatly confuses the passengers. However, perhaps that behavior can be eliminated by tweaking variables in the Q-Learning algorithm.

***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?*

1. `epsilon`, interestingly enough, was not used (discussion follows).
2. `alpha` was dynamic in this algorithm, always being the inverse of the number of times the Q-Learning algorithm had encountered the current state-action pair. Thus, initially, alpha is 1, then 0.5, then 0.33, etc. for a particular state in which that action has already been chosen. This has the effect of replacing `epsilon`'s simulated annealing effect, as over time, the learning rate "cools" just the same, though by a different calculation.
3. `gamma` was `0.03`. This is primarily an attempt to suppress the "looping" tendency identified in the question above. Interestingly, a `gamma` this low still allowed the smartcab to "learn" a good driving policy within a few trials.
    1. At a gamma of 1, the car remains stationary always.
    2. At a gamma of 0.9, the car very quickly favors looping.
    3. Even at a gamma of 0.33, the car begins to enter a looping behavior within 5 or 6 total trials.

    4. At a gamma of 0.1, the tendency to loop is diminished, but looping occurs eventually after the ~12th trial.
    5. At a gamma of 0.03, it appears that the looping behaviour does not reappear, even up to 100 trials.
4. The resulting driving agent prefers to identify a straight-line (Manhattan) path and follow it to the destination. This agent does not opportunistically divert to side streets when it finds itself at a red light (the "looping" behaviour discussed aboved). This agent obeys traffic signals, but occasionally acts in a way that would **cause a collision** with other vehicles on the roadway (yikes!). The success rate of this agent rapidly approaches 1.0 over 100 trials.
5. The occasional collisions, I assert, are because of a low number of trials relative to the probability of encountering any given other-vehicle configuration. Since encountering other vehicles is relatively rare, the smartcab doesn't have time in only 100 trials to learn not to ram into them.
6. By increasing the number of trials to 1000, the smartcab eventually encounters enough other-vehicle state-action pairs to "learn" the correct actions to take in those cases.

***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?*

1. The resulting agent quickly approaches an optimal policy.
    1. After the first few trials, the agent generally attempts to head toward the destination along the straight-line Manhattan path.
    2. For the next hundred or so trials, the agent will occasionally encounter other vehicles. As it encounters other vehicles rarely, it is uncertain of the action it should take in these cases. This usually results in collisions.
    3. After the next few hundred trials, the agent has learned enough about other vehicles to avoid running into them.
2. To me, an optimal policy would be able to discern between otherwise equivalent paths, and to take the path which allows forward movement at the present time. This, however, I think is limited by the waypoints which are provided to the agent by the RoutePlanner.
3. Additionally, an optimal policy would be able to identify slowdowns ahead and to take an alternate route which, while not usually better, would be better whenever the original route has "traffic."
4. Finally, an optimal policy would be able to exploit inherent regularity in traffic signalling to choose a path while taking into account the amount of time which will be spent at red lights. This could simply be using existing "green wave" traffic signal design, or it could be switching to an alternate route when the agent realizes that the lights on the current path will waste a lot of its time.