Observe what you see with the agent's behavior as it takes random actions.

*Does the **smartcab** eventually make it to the destination?*

- Yes, it does!
- Somehow, the smartcab does make it to the destination eventually even though the action selection policy is random.

RANDOM ACTIONS		
Destination reached count	num_trials	Destination reached percentage
20	100	20%
43	200	21.5%
106	500	21.2%

Are there any other interesting observations to note?

- There was nothing else interesting I could observe
- I was expecting some accidents since the action selection policy was random, but I did not notice any. I am guessing other agents might be smart enough to avoid an accident from occurring.
- Also, there was no change in the agent's state and for some reason the agent's action displayed at the top left corner does not match the agent's action displayed by its side.
- I noticed that the world stops the smart cab from actually breaking traffic violations, but just give a negative reward when the smart cab performs actions that could break traffic violations.

*What states have you identified that are appropriate for modeling the **smartcab** and environment?*

- This is a very good question. My answer for this question kept changing as I spent more time modelling the agent. (I wanted to leave my previous answers in here to show the progression of my thinking, but it would too much work for the reviewer).
- Since we are concerned about the time to destination and traffic safety, I decided to go with states that had a bit of both in them.
- Basically, my states were a combination of the direction of the shortest distance to the destination and the traffic light signals (red, green with oncoming vehicles and green with NO oncoming vehicles)

Why do you believe each of these states to be appropriate for this problem?

- Like I noted in my previous answer, having some location information and information for monitoring traffic safety fully describes what could be occurring in the agent's world at each point in time.
- I believe having states that contain both location and traffic information would provide us with enough information at each point in time to make decisions about distance to the destination and safety of the smartcab's driving.

*How many states in total exist for the **smartcab** in this environment?*

- There can be a whole lot of states in the cab environment.
- I went down the path of having a state for each intersection and traffic light and the problem got very complex quickly and the amount of states grew really large!
- With my new state description model, we have a total of 25 states.
- 1 goal state and 24 intermediate states

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?

- Having 25 states is not bad at all.
- Remember, our estimate of the Q value converges to the actual Q value as we visit all the state action pairs up to infinity!
- Infinity is a very long time!
- The more states you have the longer infinity becomes!

What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken?

- The agent's actions were a lot less random after implementing Q Learning
- I could notice the agent getting to the destination faster and incurring the least penalties
- With random actions, it really is just about chance and time.
- The pretty cool thing or frustrating thing is how dumb the agent is at the beginning.
- Initially, it seems random is better since the agent is trying to visit all possible state action pairs to solidify its learning and watching it turn away from the destination broke my heart a couple of times.

Why is this behavior occurring?

- This behavior is occurring because in order for our estimate of the Q value to converge to its actual value, the estimated Q value for all state action pairs needs to be visited and updated infinitely.

- Taking very random actions or non-optimal actions early on, allows our algorithm to recover from local minima issues which I noticed until I added the decaying epsilon and tuned other parameters.

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?

discount_rate (gamma)	learning_rate (alpha)	max_epsilon	epsilon_decay	num_trials	success_rate	rank
0.25	0.85	0.48	0.005	100	29%	5th
0.5	0.5	0.8	0.00085	100	70%	2nd
0.5	0.9	0.9	0.00015	200	36%	4th
0.9	0.5	0.8	0.00025	200	37.5%	3rd
0.9	0.9	0.8	0.00085	500	78.6%	1st

How well does the final driving agent perform?

- The final driving agent performs the best. The last 5 to 10 trials, the agent performs as optimal as it can.

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?

- Yes, I think it does.

How would you describe an optimal policy for this problem?

- I think the optimal policy for this problem would be to do the following:
    - For the most part stop at a red light
    - Avoid turning left when there is an oncoming traffic
    - Follow the shortest distance in terms of direction to the destination

RESPONSE TO FEEDBACK FROM REVIEW

Why did you not use the planner api provided in the code?

- I decided to write some sort of distance function because the planner api just calculates the perpendicular distance.
- This is fine, but I wanted more directions to properly describe my states, so I came up with having 8 directions which is not bad.
- I can optimize the code by using matrices for the calculation and transformation, but I want to focus more on the machine learning algorithm aspect of the exercise.

There seems to be some missing states, and why do you ignore the deadline variable

- I have expanded my states from 25 to 129 to accommodate your relevant suggestions.
- I decided to go with all the possible combinations of directions, traffic lights, and traffic situations at intersections.
- Working on this problem I realized the importance of balancing the number of states you have.
- Like I said in my previous submission, for the estimate Q value to converge to the actual Q value, our algorithm needs to visit all state action pairs infinitely.
- The more states we have, the longer it would take for our estimate Q value to converge to the actual Q value.
- Also having large amounts of states might complicate the problem.
- I can add more states by grouping the deadline values, but I could not justify the benefit of increasing the number of states since I already capture some distance information by using the shortest distance to the destination in calculating my direction.
- Therefor optimally, the algorithm should learn to try to follow the shortest path.
- Drawbacks of having a large state space include longer time to train, longer time to learn, longer time to identify optimal policy, debugging becomes an issue etc.

How quickly does the agent start to follow traffic rules?

- The answer to this question depends on the values of the parameters available for tuning
- While working on this problem, I ran into several weird situations where the agent was constantly following the right on red lights and was receiving a lot of positive feedback from doing that so it got stuck several times in local optima situations.
- Removing the current optimal selection list when selecting a random action helps force the algorithm to try something else and then recover. This helped because now it could explore more states and attach rewards for other actions.
- With most of the parameters, the agent behaves dumb initially and only starts to get smart somewhere between the 30th and 50th trial.
- However, with my best parameter selection I could notice the agent making smart decision around the 20th trial.

How would tuning alpha, gamma and epsilon affect your algorithm?

- the alpha variable controls how much we learn from our old and new Q values
- if alpha is really small, then we learn very little from our new Q values and favor the old ones, which I do not think is ideal.
- the gamma variable is the discount rate we add to our Q value.
- the gamma variable kind of makes us answer the question how much weight do we want to put on the fact that the next optimal state we could get to from this state is awesome.
- I think having high values for both alpha and gamma would help the algorithm perform better overall
- the epsilon value controls the probability of us selecting the optimal action from the policy
- like I mentioned in earlier sections of this document, getting stuck in some local optima is a big issue with our implementation of Q learning.

- Having epsilon decay from a large value (which favors more random actions) to a low value as we do more learning is optimal because it allows us to explore out of and recover from local optima scenarios.
- It also helps us in exploring more state action pairs which improves our actual Q values.

Would it be beneficial to make alpha and gamma time-dependent variables like you did with epsilon?

- I can see some sense in doing the opposite I did for epsilon with alpha and gamma
- That is, increasing the value of alpha and gamma over time as we explore more and a bit more certain that we are getting closer to the optimal policy.
- However, I do not think it would make that much of a difference, I would definitely give it a try sometime later on.

How can we reduce behaviors such as agent's circling around to accumulate more reward?

- The reward system is kind of the foundation and possible the most important part to this algorithm functioning properly.
- Modifying the rewards could discourage some actions and lead to the agent adopting new behaviors
- Adding some sort of punishment for drifting away from the destination could help modify the behavior of the agent or adding a bonus for finishing very early (e.g reward + (deadline * bonus_value)) could also help the agent modify its behavior so it goes to the destination quickly.
- In general though, it is really cool knowing you can train something to perform better in an environment by setting a balanced reward model.