Parth Mishra

1. 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?

As expected, choosing random actions causes erratic movements and the car seems to not care about obeying traffic rules and regularly gets into accidents with other cars, runs red lights, etc. This run is still useful because we often want to know how our learned policy performs with respect to a random policy i.e. is our policy significantly better than just choosing an action at random? In order to quantify this question, I calculated the success rate of the car in this random run and it achieved 13% success rate in 100 trials which is really bad. Intuitively speaking, it does not seem difficult for any policy at all to achieve better results than that. However, it must be known that given the circumstances of the situation, we would ideally like to have >95% success rate. If we take into account the fact that “misses” are often due to traffic violations, then it logically follows that the car is more likely to end up in a dangerous situation. We would not want to be in a car that is that likely to put us in a dangerous situation.

2. 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 used 4 states for modeling the smartcab in this environment: Light, Oncoming, Left, and Right. The value of each of these serves to model how the car should behave at any point in time e.g. if the light is red and there is no oncoming cars from the left, then the car can take a right as its next action. A combination of the values of these states is robust enough to cover most relevant traffic situations necessary in order to ensure the car reaches its destination safely and in a timely fashion. The waypoint is also stored due to it being a function of the reward along with the light. Adding in too many parameters for the state can cause calculations to slow down so unimportant ones that do not factor into our reward function can be ignored (deadline) so long as we still maintain enough robustness to model every relevant situation.

3. 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?

Implementing the Q-learning algorithm results in a very noticeable difference in the agent’s behavior compared to the previous random iteration. Empirical observation shows the car obeying traffic rules for the most part, especially as trials go on which shows that it is learning these rules over time via the reward function. Additionally, the car reaches its destination much more frequently since it’s now able to maximize the reward it gets (success rate of ~95% compared to the 13% of the random run). This behavior occurs because of the Q learning algorithm which starts by assigning initial Q-values to state, action pairs. Once initialized, the agent will try actions and update these Q-values for state-action pairs based upon the reward received. This is a very iterative process and as such, the Q-values must be updated frequently. As these Q-values are updated beyond their initial values, the algorithm can then start selecting the maximum Q-value available and take its associated action for the current state. Just doing this however can run into the exploration-exploitation dilemma so the epsilon-greedy method is used which increases the chance of taking a random action in the early trials so that the algorithm can see new paths rather than getting stuck in the same path that it knows can

maximize its reward. By using a decay rate on the epsilon factor, we can reduce the amount of randomness as a function of time (or time_steps in this case) so that as we learn more over time, we have less need to explore and should focus on exploiting the optimal paths that are learned.

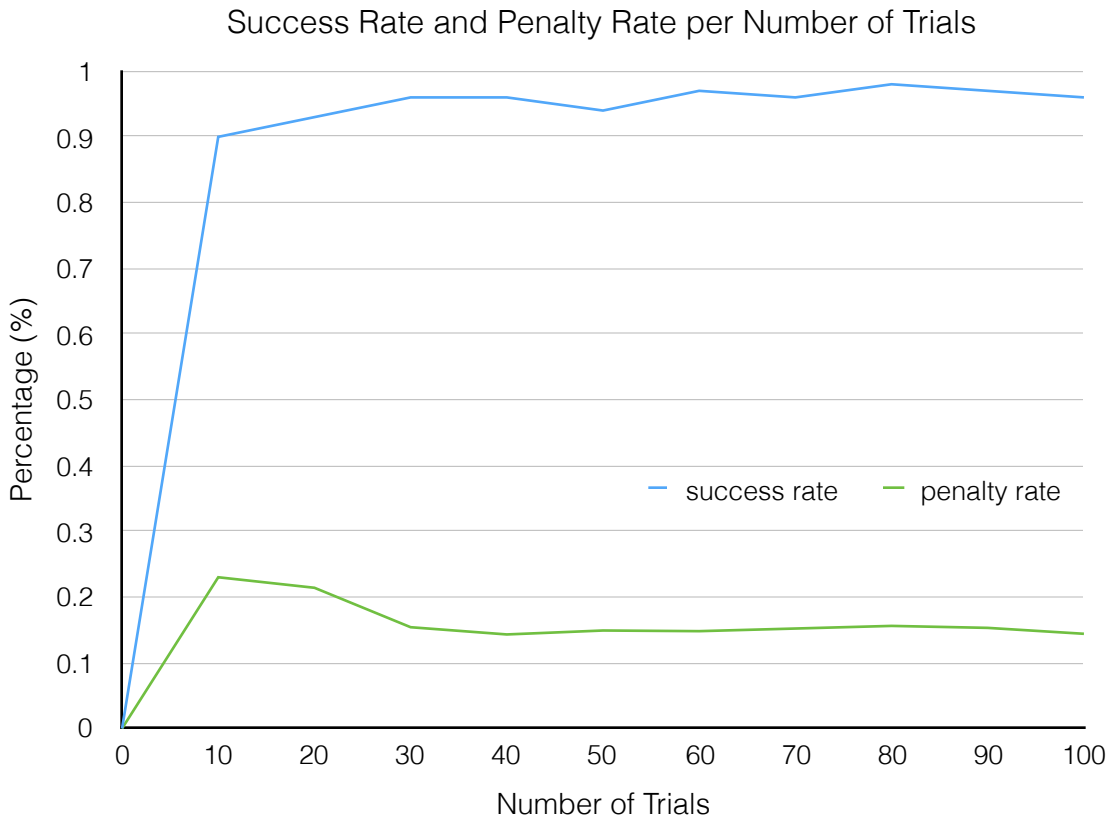
4. Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

alpha	gamma	epsilon	success rate	penalty rate
0	0	0	3.0%	7.5%
0	0	1	0%	8.1%
0	1	0	3.0%	7.5%
0	1	1	1.0%	6.6%
1	0	0	96%	13.9%
1	0	1	95%	14.1%
1	1	0	36%	44%
1	1	1	25%	42.6%
0.25	0.25	0.25	97%	13.7%
0.5	0.5	0.5	92%	22%
0.75	0.75	0.75	88%	26%
0.5	0.1	0.5	98%	13.5%
0.75	0.1	0.75	97%	15.0%

My initial implementation of the Q-learning algorithm already achieved results consistently around 95% success with some default values of alpha, gamma, and epsilon. Slight tweaking led me to have a final value of alpha = 0.5, gamma = 0.10, and epsilon = 0.5 which slight improved the average success rate to ~97%. From the table above, there doesn't seem to be a huge discernible difference in the values alpha, gamma, and epsilon provided they aren't at either end of extreme (0 or 1). From the results of the extreme results, it provided some intuition with which direction to alter the parameters with respect to each other. One obvious drawback of this approach is that this is not an exhaustive search over all permutations but, given the observed patterns it doesn't seem like further tweaking really changes too much to be a significant difference. The algorithm heavily converges towards an optimal policy provided the values of the parameters are in the interval (0,1) In this interval, the penalty rates mostly fluctuate around the 13-15% range while the success rate is mostly between 95-98% Overall,

the results of the algorithm were in line with my initial expectation of an acceptable success rate for this kind of problem. Over time the success rate improves significantly and within a few time steps, the success rate was well into the 90% range which shows how quickly this algorithm can learn.

5. 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?



Like the aforementioned success rate suggests that this policy has converged fairly close to the optimal policy. More trial runs could definitely make it converge slightly further since there were occasional penalties incurred. From the graph above (made after tuning optimal parameters) it's clear that the success rate and penalty rate converge fairly early on by ~30 trials. A theoretical optimal policy would not have any traffic infractions incurred as well as reaching the destination in the shortest amount of time which is not guaranteed by this current policy. Altering the reward numbers to account for this may allow it converge slightly further to the optimal policy but for all intents and purposes, this learned policy seems to be close enough.