# *Project Report*

## Implement a Basic Driving Agent

***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?***

The agent will make silly decision in intersections. Instead of approaching its destination, it is more likely to drive away or bump into other agents. There is few case that the agent makes it to the destination in allotted time, whose probability is negligible.  This agent is vulnerable to traffic jam, when all the other three ways have a cab coming, It will take a long time for it to make a decision, just like what a road novice in real life will do in an intersection.

## Inform the Driving Agent

***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?***

The states I pick are the rights to take right turn, left turn or forward push, the next waypoint that the planner suggests and deadline excess. They are the states directly affects agent's reward in this environment. Below are reasons for each state:

- **The Right to Take Right Turn, Left Turn or Forward Push:** Before trying to make it to the destination, the agent should obey the traffic laws. First, the cases that the agent is able to turn right are: the light is green; the oncoming agent is not trying to take left turn and the one on the left side is not trying to push forward. Second, in order to turn left, the light should be green, and any cab that trying to move from the opposite should only go left. Last but not least, agent can go forward as long as the light is green.

- **The Next Waypoint:** The next waypoint suggests the next way the agent should go in order to get to the destination as soon as possible and avoid risking bumping into other agents. For the agent to learn, the environment should assign penalty if the agent tries to go the way that's not suggested.

- **Deadline Excess:** A cab driver should pick route that similar to what the planner suggests, otherwise, it will waste gasoline and passenger's time which reduce satisfaction or probably, tips. The agent should take penalty if it wanders too much around the town. Deadline in real life is not concrete except the case in which the passenger is going for movie or meeting, etc. So it's better to make it a Boolean value that illustrates whether the agent is late. This is also better than continuous number so that it shrinks the states space in Q learning and increase the learning efficiency in other behaviors.

***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?***

According to the statements in the previous question, whether the agent has the rights to take different actions makes 2*2*2=8 sets of cases, next waypoint makes 3, and deadline excuses makes 2, which together give 48 states in total. And the agent has 4 valid actions so turns out the Q table has 192 entries.

192 is quite reasonable. Supposed we train the cab in a city where we will meet 10 intersections on a trip at average, particularly, in a grid like city with heavy traffic, which provide sufficient information to learn with, the agent should converge quickly given enough trials like 100 which approximately provides 1000 exploration chances.

## Implement a Q-Learning Driving Agent

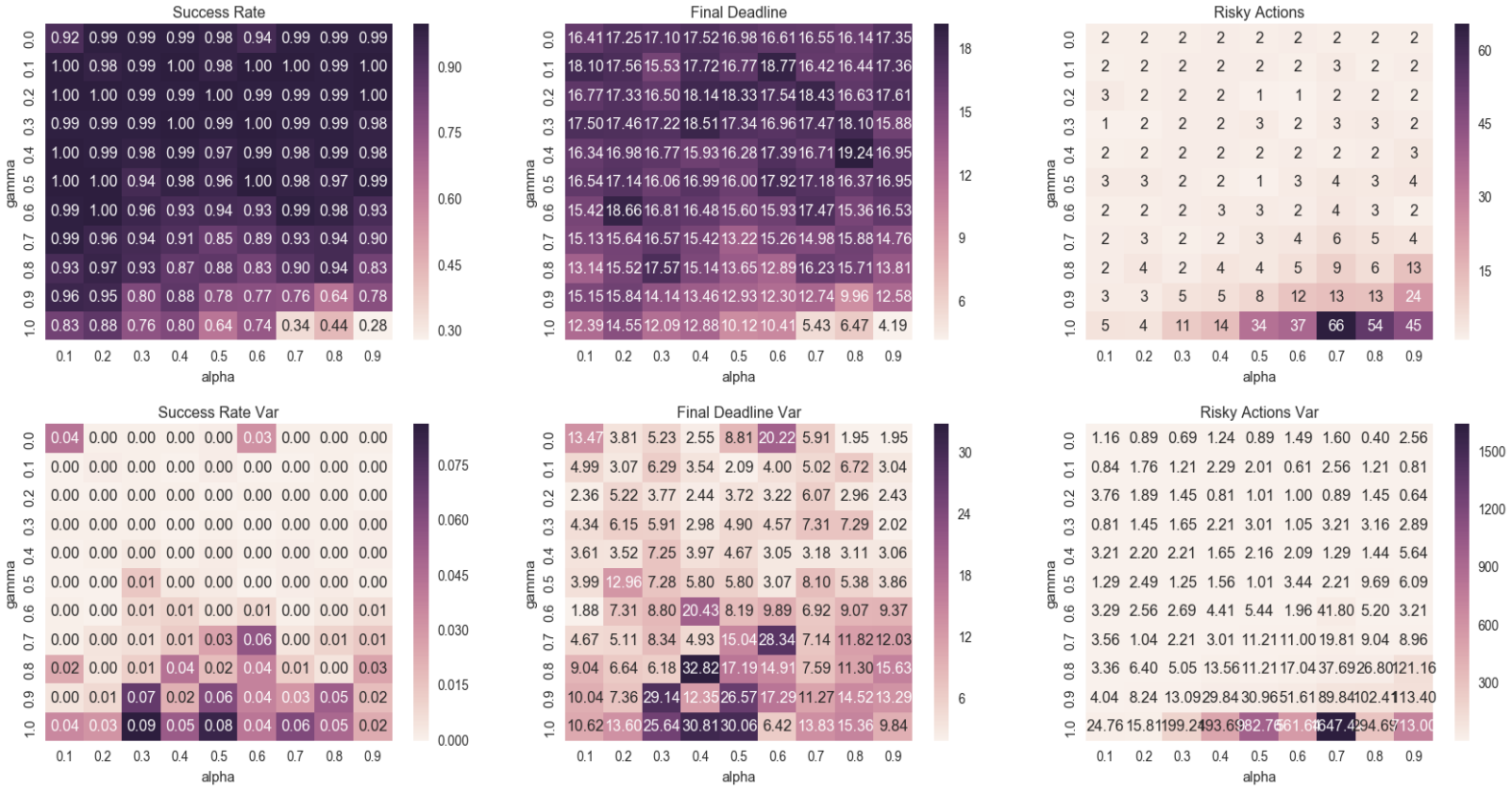
***QUESTION: What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?***

I pick the next waypoint the planner offers as the exploration source in the training. Therefore, at the beginning, the agent acts the silly way the random one did, and it still makes a lot of troubles to the traffic. After a couple of trials, the agent started to make less mistake in the intersections. More trials give better decisions. The agent makes a lot of mistakes at the beginning because the Q table is not quite initialized yet, the agent still being ignorant about what is good or bad. As the training goes on and the agent gets penalty for mistakes and reward for right moves, the Q table is gradually collecting information for later use, and according to the building Q table, the agent can choose the action that could give a best Q value, i.e. the agent will make better decision if the situation has been seen before.

## Improve the Q-Learning Driving Agent

***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 parameter I pick are `gamma = 0.2` , `alpha = 0.5` , `n_trials = 100` . And decaying `epsilon` is used starting with 0.1, the best out of 1, 0.1, 0.01 via testing. Implementing the `GridSearch()` method in `agent.py` will give 6 heat maps by iterating grid searching for `n_times` times. For every *gamma-alpha* pairs, the success rate, final deadline and times that the agent takes risky actions are shown as follow: the figures at the top show average value for each criterion in the last 10 trials out of 100, while those at the bottom show the variance respectively. Ideally, the best parameters should maximize success rate and final deadline while minimize the risky actions in the last 10 trials. First of all, safety is the most important in traffic. Then we should look at the success rate and final deadline. So `gamma = 0.2` , `alpha = 0.5` give low number of risky actions and continuous 100% success rate with sufficient deadline left. Of course, the choices of optimal parameters depend on how much the client cares about a criterion over one another, but no matter what, the heat maps should help.



***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?***

For the smartcab problem, an optimal policy should make the trip safe first, and simultaneously tries to approach the destination and reach there as soon as possible, just like what we do in real life. More strictly, the agent should be smart enough to make some decisions like avoiding going to the blocks with busy traffic or switch to a better plan when it saw heavy traffic.

Based on the performance on the last 10 trials of 100, the agent approximately following an optimal policy:

- The grid search heat maps in the last question suggest that the policy is good at guiding the agent to the destination.
- On-time arrival is not difficult for the agent according to the deadline heat maps. I also introduce a simple check that record the times that -0.5 penalty, which means the action is legal but `action != next_waypoint`, occurs in the last 10 trials and print the average to see if the agent will take useless moves that might not help to approach the destination. This result turns out 0.0 so far most of the tests.
- The agent occasionally avoids all the risky moves (those give -1.0 penalty, check using the same manner for silly actions), while sometimes it takes 1 or 2 risky moves in every trials. This can be improved if we just take next waypoint as best action instead of random choice when a state is not yet seen, but it will sacrifice more chances to explore the Q table entries.
- One important thing that the agent is not capable to do now is to switch route when next waypoint is not available. To do this, the planner should also return a spare next waypoint if it exists.