### Random Driving

*In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?*

The agent moves randomly around the board not caring about the rules of traffic or getting to its destination. Because we're discarding the time limit, it will necessarily eventually make it to the target location, but obviously there's no sense of direction or purpose in a randomly acting agent.

### Identify States

*Justify why you picked these set of states, and how they model the agent and its environment.*

I chose to represent the state as all of the inputs that the car needs to represent its environment. These include the color of the light at the intersection, the state of traffic as defined by the three variables oncoming, left, and right, and the heading, next_waypoint, computed by the planner. These are the only things that should influence the car's behavior at a given time-step.

I considered including deadline as well, so that the car might be able to learn to violate traffic rules if the count was very well (if this proved advantageous) but this increases the number of states by a factor of 25 and would slow learning down by at least that factor. Furthermore, since the car never gets any information about how far away it actually is from the destination, this seems like a pointless thing to do.

## Implement Q-Learning

*What changes do you notice in the agent's behavior?*

I notice that almost immediately, the agent seems to be able to learn how to follow the heading it's supposed to, without breaking traffic rules. With a learning rate of 0.5 and a discount rate of 0.5, and randomly initialized Q-values, it seems like performance is pretty strong.

Unfortunately, when I run this again, the performance seems quite inconsistent, and must depend on the starting state of the random weights. What I see now is that sometimes the car learns to never move. This may be due to the fact that motion in any direction gets associated with negative rewards when it doesn't match the heading.

## Enhance the driving agent

*Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?*

*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?*

I found two ways to overcome the inconsistency problem I noted above. The first felt a little bit like cheating. Since the problem was that the agent sometimes got stuck in a local optimum where no movement meant no penalties, so it never moved, I initialized Q values for actions by assigning uniformly chosen values between 0 and 1 to any action that was not "None" and between -1 and 0 for "None". This worked like a charm, but felt like I was imparting too much of the problem structure to the agent personally.

The other solution I found was to err more on the "exploration" side of the exploitation-exploration tradeoff by defining a certain probability that the agent would ignore everything that it knew and choose a random action to find out what the reward would be. I made this probability diminish over time so that by trial 75 it had gone to zero.

So essentially the final model has three parameters: learning rate, discount rate, and the starting random choice probability. Because I don't want to completely rework the object design of the project, I've assumed that they have independent impacts on the learner's speed and accuracy. So for each of the parameters I've tested a number of different values. For learning and discount rates, I've tried [0.1, 0.3, 0.5, 0.7, 0.9], and for the random parameter [0.05, 0.1, 0.15, 0.2, 0.25, 0.5], varying these while holding the others constant at the values I set at the outset (learning rate = 0.5, discount rate = 0.5, random start = 0.2). For each value I repeat the learning process 10 times and take the average value of the number of successes, total penalties, the latest failure, and the average amount of time remaining when the agent reaches its destination.

What I learn from all this as that as long as my learning rate is not very small (0.1 achieved only an average of 90.6 successes), and the random probability is neither too low (0.05 achieved only 85.4 successes) nor too high, (0.5 got 87.5), I can basically expect the learner to get to the destination 92-95 times out of a hundred with somewhere between 15 and 16 time steps remaining, while incurring a penalty of between 70 and 80 over the whole run – an average of 0.7-0.8 per run, heavily weighted to the first few trials.

I would expect an ideal policy to be one where the learner follows the suggested headings as soon as the traffic rules permit. Given that the model quickly stops

disobeying traffic laws and averages somewhere around 15 time steps remaining, I would say it's achieved behavior fairly close to ideal.