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

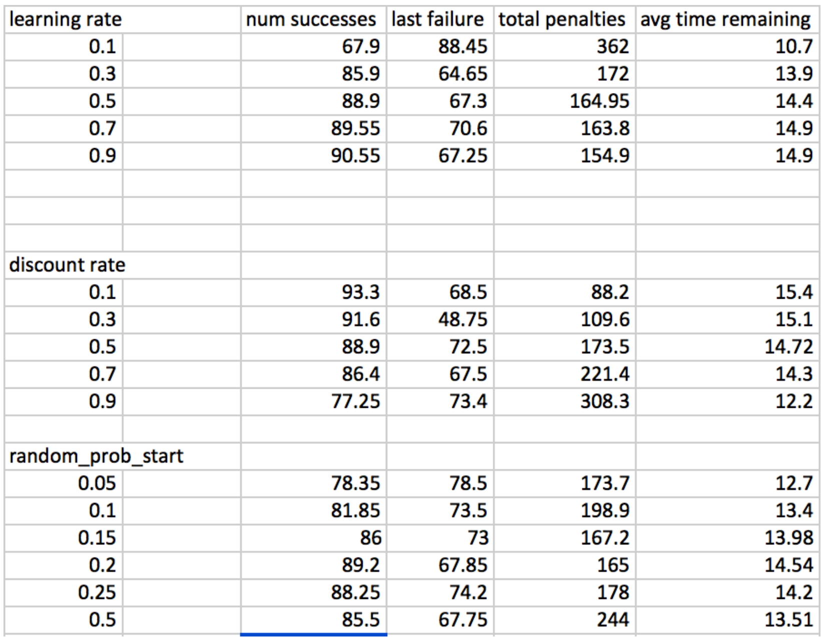
Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

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 20 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.



learning rate		num successes	last failure	total penalties	avg time remaining
0.1		67.9	88.45	362	10.7
0.3		85.9	64.65	172	13.9
0.5		88.9	67.3	164.95	14.4
0.7		89.55	70.6	163.8	14.9
0.9		90.55	67.25	154.9	14.9
discount rate					
0.1		93.3	68.5	88.2	15.4
0.3		91.6	48.75	109.6	15.1
0.5		88.9	72.5	173.5	14.72
0.7		86.4	67.5	221.4	14.3
0.9		77.25	73.4	308.3	12.2
random_prob_start					
0.05		78.35	78.5	173.7	12.7
0.1		81.85	73.5	198.9	13.4
0.15		86	73	167.2	13.98
0.2		89.2	67.85	165	14.54
0.25		88.25	74.2	178	14.2
0.5		85.5	67.75	244	13.51

What I learn from all this is that I need a high learning rate, a low discount rate, and a random probability starting around 20%, which seems about right because it should trigger 4-6 times per trial at the beginning and taper off from there. Much higher and the first trials do too much exploration, and lower and they do very little. It looks like the number of successes and penalties seem to inversely correlated, which is a nice property – we don't seem to have to ignore the laws to get to our destinations.

Setting the (learning rate, discount rate, random_prob) = (0.9, 0.1, 0.2), I get that the number of successes averages 93.6, last failure happens early on average at 51.6, total penalties are low, at 88.9 or an average of less than 1 per trial, and the average remaining time is 15.5 steps. I'd note this is not much different from the statistics shown under discount rate for 0.1, where the learning rate was set to 0.5, so it seems like fixing the discount rate was the most important part of this exercise.

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. Looking through the Q matrix, I notice that states with no oncoming traffic have tended to converge quite well to what we'd expect:

```
('forward', 'red', None, None, None)
Action: forward, Q: -0.517475
Action: right, Q: -0.089555
Action: None, Q: 0.396382
Action: left, Q: -0.594036
```

```
('forward', 'green', None, None, None)
Action: forward, Q: 2.553410
Action: right, Q: -0.029666
Action: None, Q: 1.141784
Action: left, Q: -0.035745

('right', 'green', None, None, None)
Action: forward, Q: 0.636636
Action: right, Q: 3.138987
Action: None, Q: 1.075955
Action: left, Q: 0.629122

('right', 'red', None, None, None)
Action: forward, Q: 0.029726
Action: right, Q: 3.214163
Action: None, Q: 0.261335
Action: left, Q: -0.608863
```

However, in the significantly rarer cases where there **is** oncoming traffic, there may not have been enough experiences with this to learn the right behavior. For example, here, for some reason the agent has learned that the right thing to do when someone is to your right and you want to take a right on a green is to turn left.

```
('right', 'green', None, 'forward', None)
Action: forward, Q: -0.209456
Action: right, Q: 0.011963
Action: None, Q: -0.015366
Action: left, Q: 0.641814
```

So this is obviously not a perfect driver, but in this test environment, this happens so seldom that it doesn't seem to have much impact on our success.