Mention what you see in the agent's behavior. Does it eventually make it to the target location?

Eventually, after some very random and dangerous behavior, the agent did make it to the target.

```
Environment.act(): Primary agent has reached destination!
```

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

I picked ("wp"),("light"),("oncoming"),("right"),("left"),("deadline") with a reason for each:

- *("wp")*: The waypoint is by far the most important part of the state, as it indicates where our target is.
- *("light")*: There is a large penalty for not respecting traffic lights, without this part of the state the car would not be able to learn that a red light means stop.
- *("oncoming", right, left)*: These are needed to avoid colliding with other agents, 3 agents is not a lot, but if we increase it to say 10 agents, we see this becomes very important.
- *("deadline")*: We incorporate the deadline into the state in order to give the car a notion of urgency, if we don't add this we risk the car just turning in order to accumulate points.

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

For a long time I could not figure out why the car was not learning to go towards the WP. I discovered it was not being rewarded for it because I stopped storing the waypoint on the agent. Which is where the environment reads it in order to give a good reward for heading towards the waypoint.

```
Bug cause by me: State1: (('light', 'green'), ('wp', 'right')), Action:right,
State2:(('light', 'red'), ('wp', 'forward')), Reward:0.5 [should be 2]
```

Once I solved this I was able to observe just how powerful Q-learning is.

On the first runs, the car does not know much about the world, but as the rewards and punishments start rolling in we can literally see how the agent starts to behave in accordance to the rules of the world. In order to understand what was happening better I decided to print the Q matrix as the agent learns. Its very clear when you see the updates happening in real time: It runs a red light and get punished, so it decreases the value of moving when the light is red. When it moves towards the waypoint one can see the matrix being updated, moving towards the destination was a good choice. After about 30-50 runs, the "basic" policy seems to be found and the car can reliably get to the destination.

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?

After seeing how it took less than 100 trails to get a good sense of the environment, I went on to analyse the explored states. I found many states that are not very explored as the agent is biased towards what it knows. My main change here was to add the desire to experiment randomly in order to explore more states during the learning phase. I also added a "train"/"test" mode to the agent (similar to cross validation). On the train mode, the agent will learn and is more willing to explore. Once the test begins there is no more learning and no more experimentation.

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?

My agent was configured via 3 parameters: learning rate, future discount on rewards and an exploration rate. In order to find the appropriate values, I decided to borrow an idea from SKLearn and implement a GridSearch on many possible values. I had to go into the code and disable the rendering part of the environment in order to speed up the trails. I ran 8,000 different combinations for 2 hours in order to arrive to the following values:

Parameters	Success%	Av Mistakes	Av Run Score
lr=0.35 fr=0.3 er=0.95	100%	1.04	25.85
lr=0.5 fr=0.05 er=0.85	100%	1.06	30.24

I was surprised by how much experimentation was used on these results, but it aligned with my initial observation that not many trails are needed to get a general understanding. A further improvement would be to make the experimentation not fixed but dynamic, based on how confident the agent is, how much learning time is available, etc.

So in the end, the agent is able to reach the destination with a 98%-100% success rate (on 50 trails) averaging just 1 mistake per trail.

I am very amazed that such a simple algorithm paired with a simulator can achieve such impressive results, Q-Learning it is a very powerful tool.

References

https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/

https://github.com/NathanEpstein/reinforce

http://arxiv.org/pdf/1311.6054.pdf