Implementing a basic driving agent.

When implementing the basic driving agent by initially defining action as:

```
action = random.choice((None, 'forward', 'left', 'right'))
```

The agent would sometimes find its destination and occasionally not (within an extended period of time before terminating the program) find its destination. Furthermore, using simple random choices, the agent would not find its destination within the deadline. The action definition was updated as suggested in a number of forum posts:

```
action = self.next_waypoint
```

With this definition, the agent always finds its destination. The agent seems to find the destination within the deadline and almost always with a positive reward at the end of each trial (this is based on observations of 30 trials.)

Definition of state

The state is defined as the inputs. The state consists of 'light', 'oncoming', 'right', and 'left'. These are the necessary inputs for the agent to make a decision on what action to take. Location was not included as the actual location is not relevant to what action to take since the actions are solely governed by what color the light is, whether there is oncoming traffic, and if it is clear to turn left or right. Finally, deadline is not included in state since the deadline is not *necessarily* required for the agent to take an action.

Implement Q-Learning

When q-learning was initially implemented, the agent took an extremely long time to reach its destination. The agent would repeatedly get stuck in suboptimal actions since it was looking for the max of similar states.

When state was updated to include next_waypoint the agent began to find reach its destination much more quickly. However, the agent did encounter an issue with red lights. The agent, after next_waypoint was included in state, repeatedly chose to go forward even if the light was red.

Enhancing the Driving Agent

In an effort to improve the driving agent I updated gamma in an effort to minimize it's effect on the agents actions. Since over time the optimal action in each state will be known, I want to minimize gamma in order to have the agent use this known optimal action. In order to do so, gamma is initialized at 1 and over time the number of trials decreases gamma so its impact on the agents' actions are minimized. Gamma is decreased by the following definition:

```
gamma = 1 / len(self.q_table)
```

This was inspired by a number of discussions in the forums.[1]

Based on the same reasoning as gamma above, I then updated the definition of alpha in order to minimize the impact of the learning rate, as the optimal actions become known over time. I gave alpha the same definition as gamma above:

```
alpha = 1 / len(self.q_table)
```

With the updated alpha & gamma it seems as though the agent experiences a bit of an improvement over time. The agent seems to take less time to reach the destination. However, it is not entirely clear whether or not the agent is finding the optimal action is certain states (primarily when the light is red and there is no oncoming traffic or other cars to the right or left.) That being said, the agent does seem to become more efficient over time.

[1] https://discussions.udacity.com/t/question-on-the-maximization-step/170555/5