Project 4: Reinforcement Learning

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?

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

Commit: https://github.com/lucasdupin/machine-learning/commit/5026353bcc5421a8726b8fc9b223a8ca3232e694

Yes, it will reach the destination but its path is far from optimal.
It will get there by chance, not because we calculated the best strategy.
It also doesn't use any quality function or calculate the utility of any move chosen.

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?

```
# Update state
self.state = {
    #"deadline": deadline,
    "light": inputs["light"],
    "oncoming": inputs["oncoming"],
    "left": inputs["left"],
    "right": inputs["right"],
    "next_waypoint": self.next_waypoint
}
```

Commit: https://github.com/lucasdupin/machine-learning/commit/c45ae8b1d1f730bfc2dc2f0c00529a3cae2820bf

I've identified the following components for a state:
Explanation under each bullet

- light: Semaphore
  Affects if a move is valid or not, you can go forward if it's red.
- oncoming: traffic coming towards you
  You won't be able to turn left if there is oncoming traffic, which makes this key relevant
- left: traffic coming from the left
  affects whether you can turn right or not
- right: traffic coming from the right
  affects whether you can turn left or not

- next_waypoint: the direction that minimizes manhatan distance to the target position
  important since the target point affects the reward

- deadline: How long do the smartcab still have to reach the destination
  In this particular case I decided to **ignore** the deadline because of the curse of dimensionality.
  The great number of possible values for this state would make training much longer than necessary

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?

on init:

```
self.alpha = 0.5
self.possible_actions = ('forward', 'left', 'right', None);
```

on update:

```
# Unique hash to represent this state when learning
state_hash = hash(frozenset(self.state.items()))
# Create dictionary to hold q values
if not state_hash in self.q_table:
    self.q_table[state_hash] = dict((a, 0) for a in self.possible_actions)

# Select action according to your policy
action = random.choice(self.possible_actions)
if (random.random() > 0.1): # Avoid getting locked in local optima
    q_values = [(action, self.q_table[state_hash][action]) for action in self.possible_actions]
    action = q_values[np.argmax(q_values, axis=0)[1]][0]

# Execute action and get reward
reward = self.env.act(self, action)

# Learn policy based on state, action, reward
self.q_table[state_hash][action] = (1.0 - self.alpha) * self.q_table[state_hash][action] + self.alpl
```

Commit: https://github.com/lucasdupin/machine-learning/commit/2a925b7b6f608c4efddb1f342674399686c214d7

It starts moving in an erratic and random manner, like in the first example, but then starts to learn what gives it better rewards, and avoid moves that return negative rewards.

This is accomplished by keeping track of rewards given at each state - in a weighted manner - and applying a learning rate.

I also implemented a feature to avoid getting stuck in local optima, by randomly picking a direction that may not necessarily look optimal at first, but might end up modifying the Q table.

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 parameters used for tuning were: “alpha”, “gamma”, “number of trials” and “random direction rate”.
You can check *some* of the results by looking into these files, each file name contains the values used:

```
100_trials_0.1_alpha_-0.1_gamma.txt
100_trials_0.1_alpha_-0.01_gamma.txt
100_trials_0.1_alpha_-2_gamma.txt
100_trials_0.1_alpha_0_gamma.txt
```

I also tried to incorporate *deadline* into the state but it takes too long to train without any perceived improvement.

Chosen values:

```
alpha: 0.1
gamma: -0.01
trials: 100
random_direction: 0.01
```

Commit: https://github.com/lucasdupin/machine-learning/commit/ef81ffdcdef2264997562f4cf6bb1bae04201ec5

Having **high gammas** make the car move accepting penalties, since staying where you are is as bad as doing something wrong. **High alphas** make it ignore what it had already learned before.
Finally, high **random direction rates** make it look erratic even though it might still be learning. This feature **must** be turned of while executing the model, we don't want cars taking random actions while driving, right?

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?

After a couple iterations it already starts to take fewer penalties. And it will try to keep behaving that way as long as gamma is low.
Take a look at this output, it had only 1 penalty during the last 10 trials:

```
Simulator.run(): Trial 91
Environment.reset(): Trial set up with start = (8, 6), destination = (1, 2), deadline = 55
RoutePlanner.route_to(): destination = (1, 2)
Environment.act(): Primary agent has reached destination! 15.84
Simulator.run(): Trial 92
Environment.reset(): Trial set up with start = (8, 5), destination = (4, 1), deadline = 40
RoutePlanner.route_to(): destination = (4, 1)
Environment.act(): Primary agent has reached destination! 13.83
Simulator.run(): Trial 93
Environment.reset(): Trial set up with start = (6, 5), destination = (4, 1), deadline = 30
RoutePlanner.route_to(): destination = (4, 1)
Environment.act(): Primary agent has reached destination! 9.91
Simulator.run(): Trial 94
Environment.reset(): Trial set up with start = (4, 6), destination = (7, 4), deadline = 25
RoutePlanner.route_to(): destination = (7, 4)
Environment.act(): Primary agent has reached destination! 7.9
Simulator.run(): Trial 95
Environment.reset(): Trial set up with start = (2, 6), destination = (3, 1), deadline = 30
RoutePlanner.route_to(): destination = (3, 1)
Environment.act(): Primary agent has reached destination! 9.84
Simulator.run(): Trial 96
Environment.reset(): Trial set up with start = (6, 2), destination = (2, 2), deadline = 20
RoutePlanner.route_to(): destination = (2, 2)
Environment.act(): Primary agent has reached destination! 5.94
Simulator.run(): Trial 97
Environment.reset(): Trial set up with start = (8, 4), destination = (1, 1), deadline = 50
RoutePlanner.route_to(): destination = (1, 1)
penalty =[
Environment.act(): Primary agent has reached destination! 31.17
Simulator.run(): Trial 98
Environment.reset(): Trial set up with start = (8, 5), destination = (3, 2), deadline = 40
RoutePlanner.route_to(): destination = (3, 2)
Environment.act(): Primary agent has reached destination! 13.79
Simulator.run(): Trial 99
Environment.reset(): Trial set up with start = (3, 2), destination = (8, 6), deadline = 45
RoutePlanner.route_to(): destination = (8, 6)
penalty =[
Environment.act(): Primary agent has reached destination! 14.82
```

This problem requires penalties to be avoided, cars aren't supposed to crash, they must protect their passengers. Even if it means you'll be late to a meeting. this is what I had in mind while tweaking the values, keeping the reward not only positive, but keeping the whole smartcab experience safe.