# Project 4: Reinforcement Learning

## Train a Smartcab to Drive

### The setting

In the project “Train a Smartcab to Drive”, a smartcab operating in an idealized grid-like city is given. There are traffic lights at each intersection and other cars present. The smartcab gets a reward for obeying traffic rules and a penalty for not obeying traffic rules or causing an accident. Goal of this project is to implement a learning agent for the smartcab that should learn an optimal policy for driving on city roads, obeying traffic rules correctly, and trying to reach the destination within a goal time based on the rewards and penalties it gets.

### Step 1 - Implement a basic driving agent

In the first step, I let the smart cab take a random action out of the possibilities of:

- doing nothing (state ‘None’)
- driving forward (state ‘forward’)
- turning left (state ‘left’)
- turning right (state ‘right’)

In this step, the smartcab does not learn from the results of it’s actions and has unlimited time to reach the goal.

I put the printed output of my test run into `'output_first_text.txt'`.

In [242]:
```
def read_in_my_text(file_name, print_lines):
    with open (file_name, "r") as myfile:
        output = myfile.read()
    count_reached = output.count('Primary agent has reached destina
tion!')
    count_aborted = output.count('Trial aborted.')
    lines = output.split('\n')
    print "Rate of reaching the destination {}".format(count_reache
d*1.0/(count_reached + count_aborted))
    if print_lines == True:
        print "Last 3 lines:"
        for line in lines[-3:]:
            print line
```

In [243]:
```
read_in_my_text("outputs/out_1_1.txt", False)
```

```
Rate of reaching the destination 0.19801980198
```

In [244]:
```
read_in_my_text("outputs/out_1_2.txt", False)
```

```
Rate of reaching the destination 0.25
```

In [245]:
```
read_in_my_text("outputs/out_1_3.txt", False)
```

```
Rate of reaching the destination 0.2
```

In [246]:
```
read_in_my_text("outputs/out_1_4.txt", False)
```

```
Rate of reaching the destination 0.23
```

In [247]:
```
read_in_my_text("outputs/out_1_5.txt", False)
```

```
Rate of reaching the destination 0.16
```

The smartcab reaches the destination in time in around 20% of the trials. If the smartcab reaches the destination is a question of chance. The actions are chosen randomly, independent of the destination and the received rewards.

## Step 2 - Identify and update state

For the states, I chose a tupel of `'next_waypoint'`, `'light'`, `'oncoming'` and `'left'`.

Without the `'next_waypoint'`, our agent wouldn't know where to go next and reaching it's destination would be a coincedence.

Without `'light'`, the agent would ignore if the light is red or green and therefore not be able to follow the traffic rules.

Without `'oncoming'`, the agent wouldn't be able to differentiate in a situation where the traffic light shows green and he wants to turn left.

Without `'left'`, the agent wouldn't be able to differentiate in a situation where the traffic light shows red and he wants to turn right.

I didn't take `'right'` into account, because no matter what our smartcab is going to do, the traffic from the right side has no influence on it.

I also didn't take the deadline into account. This doesn't help our agent to make the right decisions to follow the traffic rules. For example, if our agent is at an intersection, the light is red and he wants to go forward, the deadline would not give him any information on how to react. It would increase the state space without a good reason.

## Step 3 - Implement Q-Learning

In this step I implemented the q-learning algorithm with a learning rate of 0.5 and a discount factor of 0.5. The next action is always chosen based on the best estemate based on the current state. I didn't use an exploration rate.

In [248]:
```
read_in_my_text("outputs/out_3_1.txt", True)
```

```
Rate of reaching the destination 0.911764705882
Last 3 lines:
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
```

In [249]:
```
read_in_my_text("outputs/out_3_2.txt", True)
```

```
Rate of reaching the destination 0.93137254902
Last 3 lines:
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
```

In [250]:
```
read_in_my_text("outputs/out_3_3.txt", True)
```

```
Rate of reaching the destination 0.06
Last 3 lines:
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
```

In [251]:
```
read_in_my_text("outputs/out_3_4.txt", True)
```

```
Rate of reaching the destination 0.970297029703
Last 3 lines:
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
```

In [252]:
```
read_in_my_text("outputs/out_3_5.txt", True)
```

```
Rate of reaching the destination 0.950980392157
Last 3 lines:
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
Environment.act(): Primary agent has reached destination!
```

Interestingly, the smartcab reaches the destination quite often in time. There seem to be some cases, when the smartcab get's stuck and doesn't reach the destination in time.

## Step 4 - Enhance the driving agent

I played around with the learning rate and calculated it using the time step. I stayed with a discount factor of 0.1. I tried several exploration rates, but that lead to not reaching the goal more often.

In [253]:
```
import pandas as pd
from IPython.display import display

import warnings
warnings.filterwarnings('ignore')
```

In [254]:
```
def get_results(file_name):
    results = pd.read_csv(file_name, header = None, names = ["Learn
ing Rate", "Discount Factor", "Exploration Rate", "Initial Deadline
", "Steps Needed", "Total Reward", "Trial Reward"])
    # delete first row, no values
    results = results[1:]
    #display(results[-10:])
    #display(results)
    return results

def get_info(input, trial_number):
    reached = 0
    for i in range(len(input)-10,len(input)):
        if input["Steps Needed"][i] <= input["Initial Deadline"][i]
:
            reached += 1

    print "Results for last 10 Trials:"
    print "Mean Reward: {}".format(input["Trial Reward"][-10:].mean
())
    print "Times Destination reached: {}".format(reached)
```

For finding the best learning rate, I started with a value of 1.0.

In [255]:
```
def read_in_attempts(number):
    for i in range (1,6):
        result = get_results('learning_rate_{}_{}.csv'.format(numbe
r, i))
        get_info(result, i)
```

In [256]:
```
first_attempt = read_in_attempts(2)
```

```
Results for last 10 Trials:
Mean Reward: 20.0
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 20.3
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 2.75
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 22.2
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 14.3
Times Destination reached: 10
```

I was pretty happy with my results, but played with other values, too. For exmple, for a learning rate of 0.75 I got the following results.

In [257]:
```
second_attempt = read_in_attempts(3)
```

```
Results for last 10 Trials:
Mean Reward: 22.15
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 20.75
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 22.4
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 19.95
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 23.75
Times Destination reached: 10
```

This was not better in reaching the goals, but with the mean reward of the last 10 trials. I tried using the total steps the agent did. I wanted to the learning rate to decrease with increasing total steps. The results of just implementing a learning rate of (1.0 / (t + 5)) + 0.75 which will decrease with increasing t are as follows. I needed the (t + 5) to not get over 1.0.

In [258]:
```
third_attempt = read_in_attempts(1)
```

```
Results for last 10 Trials:
Mean Reward: 20.9
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 22.35
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 19.2
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 22.05
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 23.05
Times Destination reached: 10
```

The results are very good, but I thought what might happen if I have a learning rate getting over 1.0 in the beginning. A learning rate of (1.0 / t) + 0.75. That would mean the agent would consider only most recent information in the first four steps, because then the learning rate is 1.0 or above.

In [259]:
```
forth_attempt = read_in_attempts(4)
```

```
Results for last 10 Trials:
Mean Reward: 24.15
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 23.35
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 23.55
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 24.3
Times Destination reached: 10
Results for last 10 Trials:
Mean Reward: 19.75
Times Destination reached: 10
```

This gave me even higher mean rewards in the last 10 trials.

In addition to the changes I made to the learning rate, I changed the initialization from my q table from initializing it with zeros to using random values between 0 and 4. With my first attempt the smartcab got stuck most of the time.

My final agent finds a close to optimal policy. As seen above it usually reaches the destination in time in the last 10 trials and it usually only gets penalties for reacting wrong when trying to go forward. This happens with green or red lights, but the green case dominates. Other rules based on more inputs than the traffic light and the direction the smartcab needs to go seem to be learned faster.