Smartcab project report

```
author: miroslav.karpis@gmail.com
project: Udacity Reinforcement Learning project (Machine Learning Engineer Nanodegree)
```

Simulation details

Valid actions

[None, 'forward', 'left', 'right']

Rules

If an agent is at a green light, it should be allowed to:

- Go forward
- Turn right
- Turn left, yielding to any oncoming traffic that is going straight or turning right

At a red light:

- Turn right, yielding to any traffic from the left that is going straight

Rewards and Goal

- The smartcab receives a reward for each successfully completed trip.
- Smaller reward for each action it executes successfully that obeys traffic rules.
- Smartcab receives a small penalty for any incorrect action.
- Larger penalty for any action that violates traffic rules or causes an accident with another vehicle.

Task 1: Implement a Basic Driving Agent

Question 1: Does the smartcab eventually make it to the destination? During my observations (approximately 10 cycles of new target placement), I have observed only 1 case, when the car reached the target (10% success rate).

Question 2: Are there any other interesting observations to note? So far I didn't observe very much of interesting behavior.

Task 2: Inform the Driving Agent

Question 1: What states have you identified that are appropriate for modeling the smartcab and environment?

Currently we have available combinations between following inputs and their states:

- **light**: ['red', 'green']
- **oncoming**: Information whether a car is approaching from oncoming(forward) direction, and the direction where the vehicle is heading. [None, 'right', 'left', 'forward']
- **right**: Information whether a car is approaching from right direction, and the direction where the vehicle is heading. [None, 'right', 'left', 'forward']
- **left**: Information whether a car is approaching from left direction, and the direction where the vehicle is heading. [None, 'right', 'left', 'forward']
- **waypoint**: The next waypoint location relative to its current location and heading. ['right', 'left', 'forward']
- **deadline**: Decrementing iteration counter given to smartcab reach the goal/target.

Because of the simulated model should have a real world like behavior, it is necessary to include all the inputs that a driver would receive in a real world. Our final output action should be based on the combination of current states and rules, it is necessary to include all the inputs that are required for the rules rules.
Below is a list of these inputs:

- light
- oncoming
- right
- left
- waypoint

All the above mentioned inputs contain information which is used in our traffic rules to output the correct action.

Question 2: Why do you believe each of these states to be appropriate for this problem?

Selected inputs and the reason why they are included into our reinforcement learning model:

- **waypoint**: gives the direction where the smartcab should move. This is the most important information about our future goal (recommended future step)

- **light, right, left and oncoming**: All of these inputs are necessary for the prediction, because of they are required in our rules scheme. By correctly following the combination of rules and input states, the agent should be able to suggest appropriate output action.

Not included states:

- **deadline**: Deadline is randomly chosen value from the environment. In some cases the value can be up to 45, what means that to train a Q_table with extra 45 states would require a lot of extra training time.

OPTIONAL 1. *How many states in total exist for the smartcab in this environment?* Below is a list of inputs and their states:

- lights - 2
- oncoming - 4
- right - 4
- left - 4

- next_waypoint - 4

  This makes together a sum of 18 states, and their total combination is 512.

  i. Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? I would say, that yes. It might take longer time to find global optimum, but in real-life situation it is necessary to include all the inputs.
  ii. Why or why not?

Task 3: 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?

Main difference that I observed is that overall agent predictions and arrives to goal are different after several iterations. At the beginning the initial several iterations are relatively similar, because of the Q_table is at initially empty and actions are randomly selected (the same as when we chose random actions). After a while we can see that the behavior of Q-Learning Driving Agent is starting to act according to waypoint (the smartcab is starting to nicely move towards the goal). So after several iterations, we can clearly see the difference between the random and Q-Learning agent, that in the random action scenario the car arrives to the goal only very seldom and in Q-Learning case it arrives nearly every time (after the Q-table values have been taught).

I have also implemented a slow increase logic to our discount factor (gamma) variable. Starting with a lower discount factor and increasing it towards its final value usually yields accelerated learning. Current implementation increases the discount factor by 0.1 for every 10 successful destinations.

Task 4: 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?

Below is a final Q_table after 100 simulations of 100 iterations with inputs **waypoint, light, right, left and oncoming. If take a look a closer look at the Q-table values (for example the first row), we can see that during the training we didn't train every combination of states. I would expect that with extending the training time, we should increase the number of trained state combination, what should reflect in our output (more rewards in a shorter time).

```
{'next_waypoint': 'forward', 'right': 'right', 'light': 'green', 'oncoming': None, 'left': None} [(None, 2.0), ('forward', 2), ('left'
{'light': 'red', 'next_waypoint': None, 'right': None, 'oncoming': None, 'left': 'left'} [(None, 2), ('forward', 2), ('left', 2), ('ri
{'next_waypoint': 'forward', 'right': 'forward', 'light': 'green', 'oncoming': None, 'left': None} [(None, 1.3), ('forward', 3.2649999
{'light': 'green', 'next_waypoint': 'forward', 'right': None, 'oncoming': None, 'left': 'left'} [(None, 2), ('forward', 2), ('left', 2
{'next_waypoint': 'forward', 'right': None, 'light': 'red', 'oncoming': None, 'left': None} [(None, 1.9999999999883582), ('forward', 0
{'next_waypoint': 'forward', 'right': None, 'light': 'green', 'oncoming': 'right', 'left': None} [(None, 2), ('forward', 7.2), ('left'
{'light': 'green', 'next_waypoint': None, 'right': None, 'oncoming': None, 'left': 'left'} [(None, 2), ('forward', 2), ('left', 2), ('
{'next_waypoint': 'left', 'right': None, 'light': 'green', 'oncoming': None, 'left': None} [(None, 1.35), ('forward', 0.575), ('left',
{'light': 'green', 'next_waypoint': 'left', 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2), ('
{'light': 'green', 'next_waypoint': 'right', 'right': None, 'oncoming': None, 'left': 'left'} [(None, 2), ('forward', 2), ('left', 2),
{'light': 'red', 'next_waypoint': 'forward', 'right': 'left', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2),
{'light': 'green', 'next_waypoint': 'forward', 'right': None, 'oncoming': 'forward', 'left': None} [(None, 2), ('forward', 2), ('left'
{'light': 'red', 'next_waypoint': 'forward', 'right': None, 'oncoming': 'forward', 'left': None} [(None, 2), ('forward', 2), ('left', 
{'light': 'green', 'next_waypoint': None, 'right': None, 'oncoming': None, 'left': 'forward'} [(None, 2), ('forward', 2), ('left', 2),
{'next_waypoint': 'forward', 'right': 'left', 'light': 'red', 'oncoming': None, 'left': None} [(None, 1.1), ('forward', 1.5), ('left',
{'next_waypoint': 'right', 'right': None, 'light': 'red', 'oncoming': 'left', 'left': None} [(None, 2), ('forward', 2), ('left', 2), (
{'next_waypoint': 'forward', 'right': None, 'light': 'red', 'oncoming': 'forward', 'left': None} [(None, 1.05), ('forward', 0.8), ('le
{'light': 'green', 'next_waypoint': 'forward', 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2),
{'light': 'red', 'next_waypoint': 'left', 'right': None, 'oncoming': 'left', 'left': None} [(None, 2), ('forward', 2), ('left', 2), ('
{'light': 'red', 'next_waypoint': 'right', 'right': None, 'oncoming': 'forward', 'left': None} [(None, 2), ('forward', 2), ('left', 2)
{'next_waypoint': 'right', 'right': None, 'light': 'green', 'oncoming': None, 'left': 'left'} [(None, 2), ('forward', 2), ('left', 2),
{'next_waypoint': 'right', 'right': None, 'light': 'red', 'oncoming': None, 'left': None} [(None, 2), ('forward', 0.6), ('left', 1.1),
{'light': 'red', 'next_waypoint': 'forward', 'right': 'right', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2)
{'next_waypoint': 'right', 'right': None, 'light': 'green', 'oncoming': None, 'left': None} [(None, 1.3812499999999999), ('forward', 0
{'light': 'red', 'next_waypoint': None, 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2), ('righ
{'light': 'green', 'next_waypoint': 'right', 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2), (
{'light': 'red', 'next_waypoint': 'right', 'right': None, 'oncoming': 'left', 'left': None} [(None, 2), ('forward', 2), ('left', 2), (
{'next_waypoint': 'forward', 'right': 'right', 'light': 'red', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2)
{'next_waypoint': 'right', 'right': None, 'light': 'red', 'oncoming': 'right', 'left': None} [(None, 1.1), ('forward', 2), ('left', 2)
{'light': 'green', 'next_waypoint': None, 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2), ('ri
{'light': 'green', 'next_waypoint': 'right', 'right': 'left', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2),
{'next_waypoint': 'left', 'right': None, 'light': 'green', 'oncoming': None, 'left': 'forward'} [(None, 2), ('forward', 2), ('left', 2
{'next_waypoint': 'forward', 'right': None, 'light': 'green', 'oncoming': None, 'left': 'left'} [(None, 1.4), ('forward', 2.4), ('left
{'light': 'red', 'next_waypoint': 'forward', 'right': None, 'oncoming': None, 'left': 'left'} [(None, 2), ('forward', 2), ('left', 2),
{'next_waypoint': 'forward', 'right': None, 'light': 'green', 'oncoming': 'left', 'left': None} [(None, 1.2), ('forward', 3.9375), ('l
{'light': 'green', 'next_waypoint': 'forward', 'right': None, 'oncoming': None, 'left': 'forward'} [(None, 2), ('forward', 2), ('left'
{'next_waypoint': 'left', 'right': None, 'light': 'red', 'oncoming': None, 'left': None} [(None, 1.9999996066086165), ('forward', 0.04
{'light': 'red', 'next_waypoint': 'right', 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2), ('r
{'light': 'red', 'next_waypoint': 'forward', 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2), (
{'light': 'green', 'next_waypoint': 'forward', 'right': 'left', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2
{'next_waypoint': 'right', 'right': None, 'light': 'red', 'oncoming': None, 'left': 'forward'} [(None, 2), ('forward', 2), ('left', 0.
{'next_waypoint': 'forward', 'right': None, 'light': 'green', 'oncoming': None, 'left': None} [(None, 1.32421875), ('forward', 4.62656
{'light': 'green', 'next_waypoint': 'right', 'right': 'forward', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 
{'light': 'red', 'next_waypoint': 'right', 'right': None, 'oncoming': 'right', 'left': None} [(None, 2), ('forward', 2), ('left', 2), 
{'light': 'green', 'next_waypoint': None, 'right': 'forward', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2),
{'next_waypoint': 'forward', 'right': None, 'light': 'red', 'oncoming': None, 'left': 'forward'} [(None, 1.2), ('forward', 0.8), ('lef
{'light': 'red', 'next_waypoint': 'right', 'right': None, 'oncoming': None, 'left': 'forward'} [(None, 2), ('forward', 2), ('left', 2)
{'light': 'red', 'next_waypoint': 'forward', 'right': None, 'oncoming': 'right', 'left': None} [(None, 2), ('forward', 2), ('left', 2)
{'light': 'green', 'next_waypoint': 'forward', 'right': 'forward', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left'
{'light': 'green', 'next_waypoint': 'forward', 'right': None, 'oncoming': 'left', 'left': None} [(None, 2), ('forward', 2), ('left', 2
{'next_waypoint': 'forward', 'right': None, 'light': 'red', 'oncoming': 'left', 'left': None} [(None, 1.9406249999999998), ('forward',
{'light': 'green', 'next_waypoint': 'left', 'right': None, 'oncoming': None, 'left': 'forward'} [(None, 2), ('forward', 2), ('left', 2
{'next_waypoint': 'left', 'right': 'forward', 'light': 'green', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 7
{'light': 'red', 'next_waypoint': None, 'right': None, 'oncoming': 'forward', 'left': None} [(None, 2), ('forward', 2), ('left', 2), (
{'light': 'red', 'next_waypoint': 'left', 'right': None, 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 2), ('ri
{'light': 'green', 'next_waypoint': 'left', 'right': None, 'oncoming': 'forward', 'left': None} [(None, 2), ('forward', 2), ('left', 2
{'next_waypoint': 'left', 'right': None, 'light': 'red', 'oncoming': 'left', 'left': None} [(None, 1.3), ('forward', 0.8), ('left', 0.
{'next_waypoint': 'forward', 'right': None, 'light': 'red', 'oncoming': None, 'left': 'left'} [(None, 1.7999999999999998), ('forward',
{'light': 'red', 'next_waypoint': 'left', 'right': None, 'oncoming': 'forward', 'left': None} [(None, 2), ('forward', 2), ('left', 2),
{'light': 'red', 'next_waypoint': 'forward', 'right': 'forward', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 
{'next_waypoint': 'forward', 'right': 'left', 'light': 'green', 'oncoming': None, 'left': None} [(None, 1.5), ('forward', 3.375), ('le
{'next_waypoint': 'forward', 'right': None, 'light': 'red', 'oncoming': 'right', 'left': None} [(None, 1.1), ('forward', 0.8), ('left'
{'next_waypoint': 'forward', 'right': None, 'light': 'green', 'oncoming': 'forward', 'left': None} [(None, 2), ('forward', 2), ('left'
{'light': 'red', 'next_waypoint': 'forward', 'right': None, 'oncoming': 'left', 'left': None} [(None, 2), ('forward', 2), ('left', 2),
{'next_waypoint': 'right', 'right': 'forward', 'light': 'green', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 
{'next_waypoint': 'forward', 'right': 'forward', 'light': 'red', 'oncoming': None, 'left': None} [(None, 1.6), ('forward', 0.7), ('lef
{'next_waypoint': 'left', 'right': None, 'light': 'green', 'oncoming': 'right', 'left': 'right'} [(None, 1.5), ('forward', 2), ('left'
{'light': 'green', 'next_waypoint': 'right', 'right': None, 'oncoming': None, 'left': 'right'} [(None, 2), ('forward', 2), ('left', 2)
{'next_waypoint': 'forward', 'right': None, 'light': 'green', 'oncoming': None, 'left': 'forward'} [(None, 1.1), ('forward', 3.5862499
{'light': 'green', 'next_waypoint': 'forward', 'right': 'right', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 
{'light': 'red', 'next_waypoint': 'forward', 'right': None, 'oncoming': None, 'left': 'forward'} [(None, 2), ('forward', 2), ('left', 
{'next_waypoint': 'right', 'right': 'left', 'light': 'green', 'oncoming': None, 'left': None} [(None, 2), ('forward', 2), ('left', 1.5
{'next_waypoint': 'right', 'right': None, 'light': 'red', 'oncoming': None, 'left': 'left'} [(None, 2), ('forward', 2), ('left', 2), (
{'light': 'green', 'next_waypoint': 'left', 'right': None, 'oncoming': 'right', 'left': 'right'} [(None, 2), ('forward', 2), ('left', 
```

Below are some results from different input combinations:

waypoint + light + left + right + oncoming

- overall_iterations: 1403
- overall_simulations: 100
- total_sucess: 98
- self.gamma: 1.0
- sucess_rate: 98.0 %

waypoint + light + left + right + oncoming + deadline

- overall_iterations: 1780
- overall_simulations: 100
- total_sucess: 80
- self.gamma: 0.9
- sucess_rate: 80.0 %

waypoint + light

- overall_iterations: 1382
- overall_simulations: 100
- total_sucess: 95
- self.gamma: 1.0
- sucess_rate: 95.0 %

From the above results we can see that the best results of 98%, we have got with the waypoint, light, right, left and oncoming input combination. We can see also interesting results in the case where we have included also deadline input. The input to the model contains most information, but the output results are the worst. The reason is that the deadline input/information does not relate to our rules - is not needed for our action prediction and the total number of states is random/different every iteration.

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?

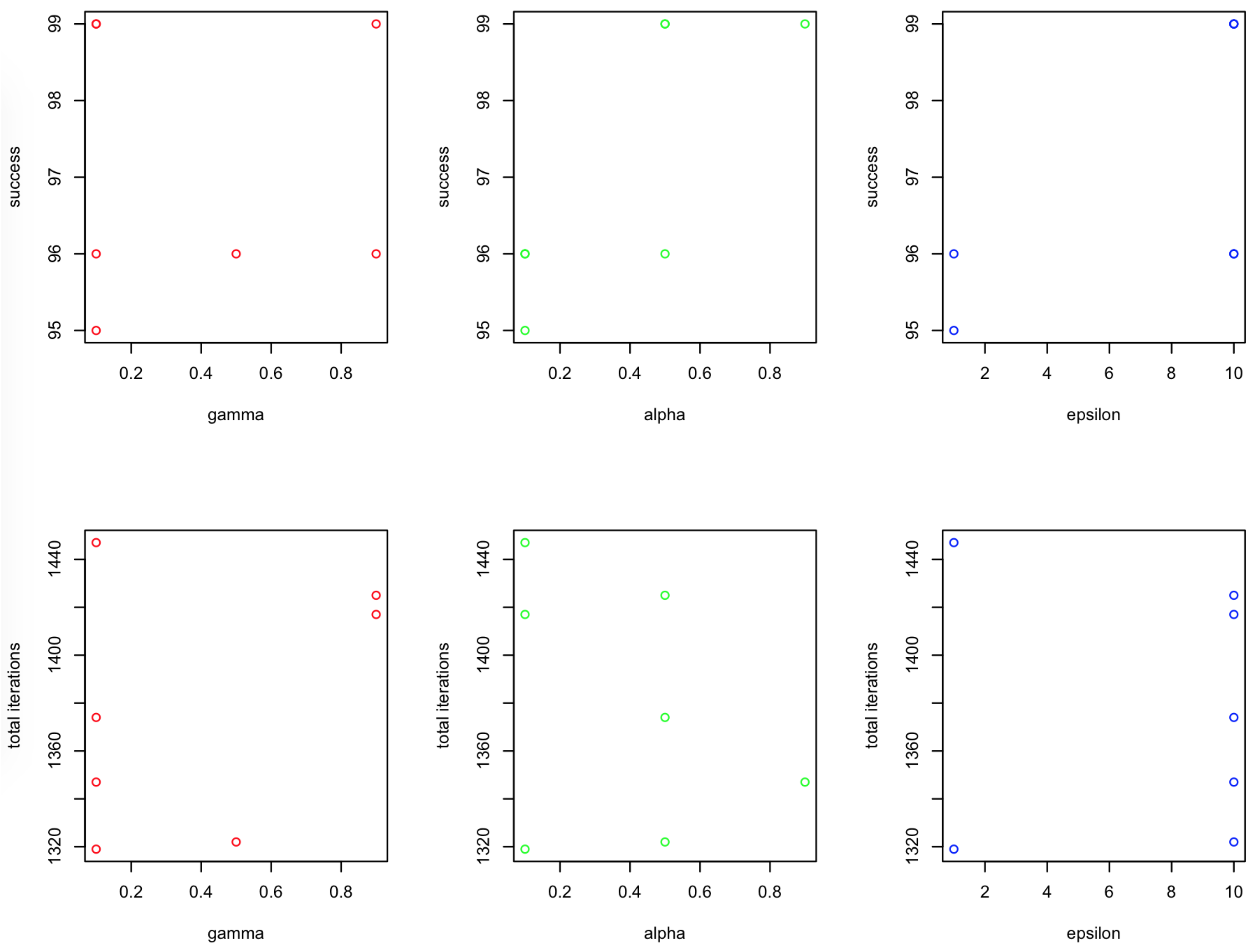
In the table below are results from different combinations of gamma (discount factor), alpha (learning rate) and epsilon (exploration rate). We can see that the best results of success rate and number of total iterations is in simulation ID 4, where we the model achieved 99% of success percentual rate in 1347 iterations. The combination of 0.1 for discount factor (gamma), 0.9 for learning-rate (alpha) and 10 for exploration rate (epsilon) was the most successful one. We need to mention that values of exploration rate and discount factor and changing during the simulation. Exploration rate is slowly decreasing and gamma value is slowly increasing. Reason why we are getting the best results from above mentioned combination might be following:

- High learning rate alpha (0.9 and increasing) - will make the agent override old information with the new one at higher rate. Reason why the results are good might be because of our agent follows waypoint input (short-time goal), which should give us the shortest path to our target. And since new waypoint is our input during every iteration, we do not need pay very attention to our old information.

- Low discount factor gamma (0.1) - Similar reason as described above explains good results from 0.1 gamma value. With low gamma value, the agent is "myopic" (or short-sighted) and it only considers current rewards.

Sim id	gamma	alpha	epsilon	total iterations	success rate
1	0.1	0.1	1	1447	95.0 %
2	0.1	0.1	10	1319	96.0 %
3	0.1	0.5	10	1374	99.0 %
4	**0.1**	**0.9**	**10**	**1347**	**99.0 %**
5	0.5	0.5	10	1322	96.0 %
6	0.9	0.5	10	1425	99.0 %
7	0.9	0.1	10	1417	96.0 %

Final implementation results plot:



self.gamma = 0.9 # discount factor self.alpha = 0.1 # learning rate self.epsilon = 10 # exploration rate

The chosen input waypoint, light, right, left and oncoming gives the best success rate, but it reaches the destination in an average possible time. As mentioned above, by increasing the number of training iterations the general number of steps should increase. Reason is that some states have still been in their initial value 2.

A very useful control of our trained model is our Q-table, where we can see what actions will our model choose with specific sates combination. By observing the final Q-table we can see that all the actions are properly taught if we consider only the included states (light and next_waypoint).