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?

To make the agent perform random actions:

action = random.choice(Environment.valid_actions)

Run for **n_trials = 100** and **enforce_deadline = True**, this is done so it can be compared in the same terms with the later iterations of the agent.

Statistics of the random run agent saved in the file **random_run.txt**

- total reward accumulated = 118**(1)**
- % of success = 27/100 = 27%
- # of actions taken = 2750
    - -0.5 pts. actions taken = 798
    - -1.0 pts. actions taken = 747
    - 0.0 pts. actions taken = 708
    - 2.0 pts. actions taken = 497
    - 10 pts. rewards received = 27
- # of state-action pairs discovered = 101
- % of state-action pairs discovered = 101/(2*4^3*3*4) = 6.57%
- Time:
    -

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
20.00	25.00	30.00	30.85	36.25	55.00

- Distance:
    -

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
4.00	5.00	6.00	6.17	7.25	11.00

- Environment statistics:
    -

s.light	s.left	s.oncoming	s.right	s.next	action
green:1259	None :2729	None :2700	None :2717	forward: 732	None :708
red :1491	forward: 8	forward: 13	forward: 11	left : 692	forward:683
	left : 8	left : 31	left : 17	right :1326	left :670
	right : 5	right : 6	right : 5		right :689

(1) The file failed to account for the rewards of achieving the goal and the sum of rewards for the 100th iteration, these were added here manually.

The smartcab does reach the destination by chance in this particular run 27% of the time. Worth of attention is how the reward system works, that is, what state-action pairs result in positive reward and which in negative ones.

The actions as expected are somehow evenly distributed as are the green/red lights. The streets are fortunately for the smartcab not heavily transited.

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?

The identified states are the following:

- Traffic light: [red,green] (2)
- Left street: [left,right,forward,none] (4)
- Right street: [left,right,forward,none] (4)
- Oncoming street: [left,right,forward,none] (4)
- Next waypoint: [left,right,forward] (3)

This is the information the agent can perceive as it travels the streets; with this information it will learn and decide the optimal actions to take to arrive to the goal in an optimal way.

I believe this information is enough to model the states for the problem, so much in fact that it can be modeled as a deterministic environment (the agent will always know exactly what to expect of a given state-action if the state-action has been already visited).

OPTIONAL: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

The number of states-actions is the multiplication of the number of options in each of the variables that create the states and the possible actions we can take in each of these states.

- **# of <s,a>** = (lights*left*right*oncoming*waypoint)*action = (2*4*4*4*3)*4 = **1,536**

It seems reasonable, the states can be added as they are seen, so the agent can start with an empty table and start populating and updating it as it explores the world. If all the <s,a> are visited, then it will need a 1538*(6+1) table, adding also the rewards for each <s,a>. From there when exploiting the world, it can just look up for the specific <s> it is in and select the <a> that has the biggest <r>.

But in practice, there are <s> that are rarely seen. The random run only visited 101 different states in 2750 turns.

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?

The logic behind the Qlearning algorithm implemented is the following:

- Detect current state, if not in the table add it.
    - Choose random action and record Qvalue of action-reward.
- If state is found, check ε vs. random variable:
    - If ε greater, choose randomly from the actions.
        - It will randomly choose first from untaken actions if any, if not it will choose randomly from all actions.
    - If ε lesser choose the best action from the Qtable.
        - If there is a tie choose randomly from them.
        - Unvisited actions within a state have a high start Qvalue; this is to encourage the learner to explore (optimist).

Thus, it incorporates random learning when at a state it hasn't been before or with actions it hasn't done before. If state-action is completely know, it chooses the highest recorded reward. ε is calculated as follows, the value of timestep can be fiddled with to calibrate the rate of change of ε.

$$\varepsilon = \frac{1}{1 + timestep}$$

This way it reduces its value as time passes, going from exploration to exploitation.

In order to update the reward, a learning rate α is used.

$$Qvalue_{t+1} = Qvalue_t * (1 - \alpha) + \alpha * r$$

For this deterministic environment, the choice of α has no impact as the same <s,a> always results in the same <r>. Still α is calculated as follows:

$$\alpha = \frac{1}{\#\ of\ visits\ to\ this < s, a >}$$

Statistics of the Qlearner agent saved in the file **smart_run4.txt**

- total reward accumulated = 2180.5 **(2)**
- % of success = 99/100 = 99%
- # of actions taken = 1363
  - -0.5 pts. actions taken = 33
  - -1.0 pts. actions taken = 23
  - 0.0 pts. actions taken = 692
  - 2.0 pts. actions taken = 615
  - 10 pts. rewards received = 99
- # of state-action pairs discovered = 86
- % of state-action pairs discovered = 86/(2*4^3*3*4) = 5.59%
- Time:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
20.0	20.0	25.0	28.3	35.0	55.0

- Distance:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
4.00	4.00	5.00	5.66	7.00	11.00

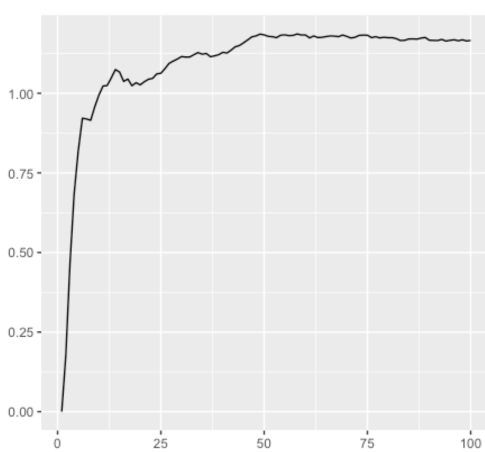
- Environment Statistics:

s.light	s.left	s.oncoming	s.right	s.next	action
green:589	None :1343	None :1338	None :1344	forward:944	None :692
red :774	forward: 7	forward: 7	forward: 9	left :268	forward:415
	left : 9	left : 11	left : 3	right :151	left :111
	right : 4	right : 7	right : 7		right :145

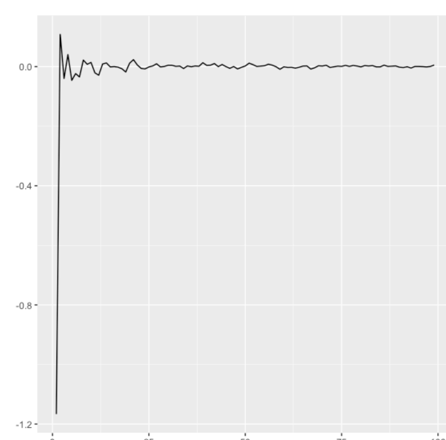
(2) The file failed to account for the sum of rewards for the 100th iteration; these were added here manually.

Many things become evident now, time, distance and the inputs of the environment are very similar between the 2 runs, as they are randomly generated and the agent has no impact on them.

The real differences appear on what the agent is doing and learning. In the runs it can be appreciated that most of the negative actions at the start which is to be expected as it is learning; after that rarely negative actions are seen (though in the last run there is one for a previously unseen state), whereas in the random runs they are uniformly distributed.



Cumulative average for Qlearn agent.



Cumulative average for Random Agent.

The previous plots show the cumulative mean reward calculated with the following equation, it is calculated 100 times, 1 for each run of the algorithm.

$$Mean\ Reward_t = \frac{Reward\ CumSum_t}{\#\ of\ timesteps_t}$$

It can be appreciated a convergence of the algorithms, at about:

- 1.17 pts/timestep with the Qlearner
- 0.005 pts/timestep with the random agent.

Worth noticing as well is the success rate of the agents. From 27% to 99%, and the fact that the Qlearner almost never incurs in traffic violations or suboptimal decisions.

Although two runs are presented and compared here, each agent performed several runs and are overall consistent with the results presented.

Before continuing, there are some aspects of the planner, environment and overall functions of problem worth highlighting.

- The position of the smartcab and the destination change each run (as one would expect of a realistic simulation for a cab).
- In the current configuration were the smartcab follows the planner function, the best policy is to follow the planner whenever that doesn't lead to incurring in a traffic violation.
- If the planner is optimal, and the Qlearner is optimal, the smartcab should reach the destination in the minimum time without incurring in traffic violations.
- An optimal planner would need to receive feedback (ex. destination NE, cab facing North with 'red light', planner proposes 'forward' feedback 'red light', then proposes right turn and assigns a reward for following new direction), but since that isn't the case, the best policy is to follow the planner whenever valid.
- In the current configuration, rewards can mess up the learner at the beginning of the training, suboptimal action could reach the goal thus receiving a high reward and reinforcing this suboptimal behavior.

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 Qlearner was initially configured with epsilon = 1, and an alpha = 1 for each <s,a,r>. They change as mentioned earlier, epsilon by 1/(1+0.1*t) and alpha (1/#visits) to <s,a,r>. This really has no impact in the calculation of the reward (an average, or substituting the old value with the new reward work as well) since the reward received doesn't change (deterministic).

Gamma "γ" is set to 0, the reason of this is that the agent searches for optimal behavior in the present, the states are unrecognizable by the position, and by being in a state and performing an action we can arrive to any other state without foresight or intuition. The rewards in current time are the drivers.

Said this, Gamma was not used, the value of Alpha has little to none impact as the agent was envisioned and that leaves epsilon.

Since the new unseen actions are initialized with a high reward (optimistic with the unknown), the algorithm always will sweep all unvisited actions. Also since the rewards do not change, if the agent is in "exploration mode" but all the states have been visited previously, then it goes automatically to "exploitation mode" selecting the max from the Qtable. The decreasing of epsilon was fiddled with, increasing or decreasing the change rate without noticeable impact. The agent has to explore in the unknown whether by choice of by circumstance.

Finally, although the state space for this problem is big (1536), there the majority of the theoretical states rarely if ever appear, ex. Traffic coming from all 3 other streets, thus learning the states that appear frequently is very fast and the agent can start exploiting early on.

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?

I believe it does, taking a look at the last run:

```
Simulator.run(): Trial 99
Environment.reset(): Trial set up with start = (2, 3), destination = (8, 6), deadline = 45
RoutePlanner.route_to(): destination = (8, 6)
total reward: 2178.0, epsilon: 0.0
        'light','left,'oncoming','right','waypoint','action','reward','state visits'
         red,None,None,None,right,right,2.0,60
         red,None,None,None,forward,None,0.0,567
         red,None,None,None,forward,None,0.0,568
         red,None,None,None,forward,None,0.0,569
         red,None,None,None,forward,None,0.0,570
         green,None,None,None,forward,forward,2.0,388
         red,None,None,None,forward,None,0.0,571
         red,None,None,None,forward,None,0.0,572
         red,None,None,None,forward,None,0.0,573
         green,None,None,None,forward,forward,2.0,389
         red,None,None,None,forward,None,0.0,574
         red,None,None,None,forward,None,0.0,575
         red,None,None,None,forward,None,0.0,576
         red,None,None,None,forward,None,0.0,577
         red,None,None,None,forward,None,0.0,578
         green,None,None,None,forward,forward,2.0,390
         red,None,None,None,forward,None,0.0,579
         red,None,None,None,forward,None,0.0,580
         red,None,None,None,forward,None,0.0,581
         red,None,None,None,forward,None,0.0,582
         green,None,None,None,forward,forward,2.0,391
         green,None,None,left,forward,forward,2.0,3
         red,None,None,None,right,right,2.0,61
         green,None,None,None,forward,forward,2.0,392
         red,None,None,None,forward,None,0.0,583
         red,None,None,None,forward,None,0.0,584
         red,None,None,None,forward,None,0.0,585
         red,None,None,None,forward,None,0.0,586
        Environment.act(): Primary agent has reached destination!
         green,None,None,None,forward,forward,2.0,393
```

It doesn’t incur in any penalty and follows the planner. It consistently reached the destination (save one time that was a road blockage or something).

The optimal policy would be to follow the planner whenever that doesn’t incur in a penalty. And it does that.