Natacha GRUMBACH *Master SIA*

Mathematical Modeling for Intelligent Systems

PROJECT
Reinforcement learning for car-following on a ring road

1 Presentation

The objective of the project is to use the Q-learning algorithm to make a car learn the process of driving. It will learn to drive by following a leader car on a ring route.
The ring has a single lane, cars can not overtake.

For this purpose, I have linked a traffic simulator (SUMO) to my program using the TRACI API to simulate the dynamics of all “autonomous” cars. Only one car, the learning car, will be controlled by the Q-learning algorithm.

The project is based on several files :

- *main.py* : the Python file that contains the Q-learning algorithm and makes it run;
- *display.py* : a Python file that contains the functions to display some charts;
- *ring.sumocfg* : the SUMO configuration file;
- *ring.add* : the additional features on the ring (mainly the rerouter to make cars run on the ring without stopping);
- *ring.rou* : the description of the route and vehicles on the route;
- *ring.settings* : the settings of the simulation frame;
- *ring_network.net* : the SUMO network that measures 637 meters;
- *ring_network_large.net* : a larger SUMO network (about 2 kms).

2 Implementation of the Q-learning algorithm

I have implemented the Q-learning algorithm in Python.

I have defined the controlled car's **state** as :

- its speed
- its relative speed to the leader car
- the space headway, distance between the leader car and the controlled car

2.1 Parameters of the program

The specificity of this program is that it takes a lot of parameters or implentation choices, and that, it is really hard to find the right combinations to make an efficient Q-learning process.

2.1.1 Parameters related to the states

In my program, I specified different parameters to define the different **states**:

- the maximum space headway and the space headway step : the space headway component of the states will have the values from 0 to the maximum space headway, by incrementing of the step.
  For example, if we specify MAX_SPACE_HEADWAY = 50 and STEP_SPACE_HEADWAY = 1, then the space headway will take all the integer values in the range [0,50]. I use a projection to make too large values of space headway be in the range. And I “round” by the step the real distance to find the corresponding state component.
- the maximum speed and the speed step : the speed component of the states will have values from 0 to the maximum speed, by incrementing of the step.
  For example, if we specify MAX_SPEED = 8 and STEP_SPEED = 1, then the speed will take all the integer values in the range [0,8]. I use a projection to make too large values of speed be in the range. And I “round” by the step the real speed value to find the corresponding state component.
- the maximum relative speed and the relative speed step : the relative speed component of the states will have values from – maximum speed to the maximum speed, by incrementing of the step.
  For example, if we specify MAX_RELATIVE_SPEED = 8 and STEP_RELATIVE_SPEED = 2, then the relative speed will take all the even values in the range [0,8]. I use a projection to make too large or too small values of relative speed be in the range. And I “round” by the step the real relative speed value to find the corresponding state component.

Specifying those 6 parameters determines the number of states that will be used in the Q-learning algorithm.

2.1.2 Parameters related to SUMO simulation

Type of vehicles

In the XML file defining the vehicles that will run on my route, I can define different types of vehicles and specify some properties on them, especially their maximum speed :

```
<vType id="learning_car" accel="10" decel="10" sigma="0" length="5" maxSpeed="8"/>
<vType id="leader_car" accel="0.4" decel="5" sigma="0" length="5" maxSpeed="8"/>
```

The maximum speed of the learning car is higher than the one of the leader car, because we want to allow it to catch up with the leader car.
Actually, the accel and decel parameters are not used because we take the full control of the car by calling the *setSpeedMode* TRACI method with 0 as parameter.

For the leader car, I tried to set its maximum speed to different values.

But I has a clear impact on the learning process : if the maximum speed is high, we have 2 possibilities :

- either we will have a lot of states if we set the MAX_SPEED to a value close to the maximum speed of the leader car and STEP_SPEED to a small step : we will have a good precision, but as the number of states will be very large, it will take a lot of time to explore all the states. In a reasonable time, we will have a lot of non explored states, that will reduce the quality of the result of the algorithm.

- Or, if we set the MAX_SPEED to a value close to the maximum speed of the leader car and STEP_SPEED to a large step, we will limit the number of states, but we will loose accuracy because a speed value will be round to the closer "state" value, for example, if we take a step of 5, if the learning car runs at 7.4 m/s, it will be rounded to 5.

Number of vehicles added to the route

In the XML file, I also decide which vehicles must be added on the route.

I add 1 vehicle of type "learning_car" that corresponds to my controlled car.
I add 1 vehicle of type "leader_car" whose name is "leader" that corresponds to my leader car.
Then I can choose to add other vehicles. I tried with different numbers.

Size of the ring

I created 2 rings : one with a small size (about 600 meters of circumference) and one larger (quite 2 kilometers of circumference).

Running the algorithm on one ring or the other gave also different results.

2.1.3 Parameters related to the actions

The algorithm must tell to the controlled car, depending on the values of the Q table for the current state, to take one of those 3 **actions**:

- accelerate
- stabilize its speed
- decelerate

I had to define the coefficients by which the speed of the learning car must be multiplied.
I tried several values. Here is an example of values I have tried.

Id	Action description	Implementation
0	Accelerate	current speed x 1.2
1	Stabilize the speed	current speed x 1
2	Decelerate	current speed x 0.8

2.1.4 Reward policy

As suggested in the subject of the project, the **reward** is defined by:

- the speed of the controlled vehicle
- a large penalty if the controlled car has an accident

I had a lot of difficulties to find a stable solution.
I tried a lot of different parameters; the value of the reward impacts a lot the Q learning algorithm.

Also, I tried to find another strategy to define the reward policy.

In our specific case, it appears that the speed is limited by the leader car speed (no overtaking). Thus, I chose to define the reward by the distance between the controlled car and the leader car.

To optimize this distance, and thus the speed of the controlled car, I compute a safety distance by multiplying the speed in m/s by 3.

Then I authorize a range around this safety distance. If the car is in the range, then it is rewarded. If the car is closer than the range, then it is punished. If the car is further away from the leader, it is punished (less than with a closer distance).

For example, the safety distance of a car running at 3 m/s is 9 meters.

The range in which the controlled car will be rewarded is defined by :
[safety_distance – speed/3 ; safety_distance + speed/3]
For the car running at 3 m/s, it corresponds to: [8 ; 10]

So if the controlled car, running at 3 m/s, is far from the leader car more than 8 meters but not far from more than 10 meters, it is positively rewarded (+10 for example).
If not in this range, the reward becomes negative : high punition if the car is closer to the leader than the safety range (-10 for example), low punition if the car is further away from the leader (-5 for example).

It suggests the controlled car to be as close as possible to the leader car, with a safety distance. Thus, as a consequence, it should also maximize its speed to make it close to the speed of the leader car.

But, I got some side effects with my controlled car that chose, in many cases, to stop itself.

2.1.5 Greedy policy

I have defined a greedy policy based on the ε parameter. The value of ε is higher at the beginning of the learning process. In the first 200 iterations of one simulation, ε = 0.2 (we introduce more randomness). Then, after 200 iterations, ε = 0.1 (less randomness).
Again, this choice can impact the learning process.

2.1.6 Parameters related to the learning process

I used different values for **gamma,** mainly high : from 0.8 to 1.
For the learning rate **alpha**, I tried values in range [0.6 - 0.9].

Another parameter of the learning process is the number of simulations (minimum 3000) that we run and the maximum number of iterations (between 1000 and 2000) in each simulation.

2.2 Other implementation choices

If a state has not been explored yet, I choose to accelerate.

If the learning car has a speed less than 1 and the choosen action is “accelerate”, then I set the speed of the learning car directly to 2.

I chose to initialize the Q table to zero.

I also tried to increase the leader maximum speed step by step, all the 1000 simulations for example. This way, the controlled car learns 1000 simulations with a small speed; then the speed is increased by 1, and so on.
This should allow the Q table to be more explored. But we take the risk that the 1000 simulations are not enough to make the car learn correctly.

3 Results

As explained previously, the multiple parameters and the multiple values for those parameters were a real problem for me.

I tried many and many configurations but I was able to find only a few cases in which I obtained a convergence and an expected result.

3.1 First configuration : reward depending on the speed of the learner

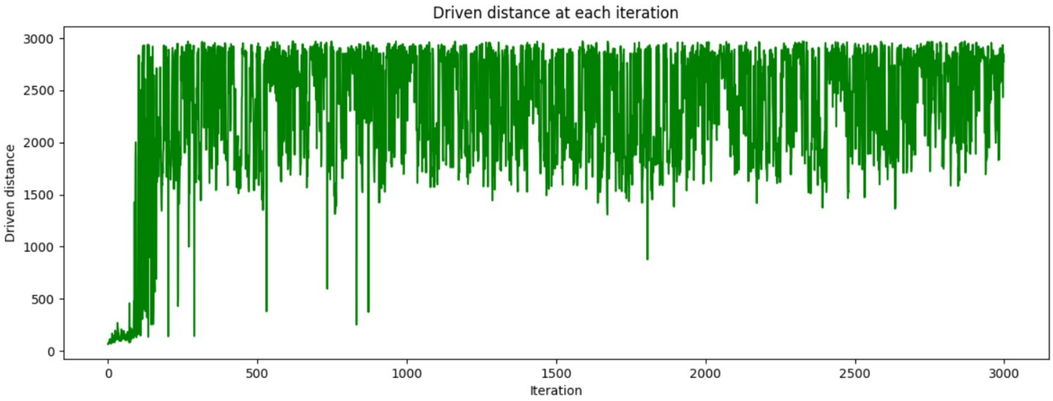
Here are the parameters I have used to have the following results.

Parameter	Value
Max speed learning car	8
Max speed leader	3
MAX_SPACE_HEADWAY	50
MAX_SPEED	5
MAX_RELATIVE_SPEED	4
STEP_SPACE_HEADWAY	1
STEP_SPEED	1
STEP_RELATIVE_SPEED	1
ACTIONS	{0.9 , 1 , 1.1}
Number of other vehicles	30
Reward	-20 if accident otherwise the speed of the learning car
ε	0.2 for 200 first iterations 0.1 after
α	0.9
γ	1
Number of scenarios	3000
Max number of iterations in 1 simulation	1000
Size of the ring	637m

<u>Number of states</u> : 2754

3.1.1 Driven distance chart

The first chart represents the driven distance by the controlled car at each simulation. We see that the driven distance increases simulation after simulation. This means that the car is well learning to avoid collisions.



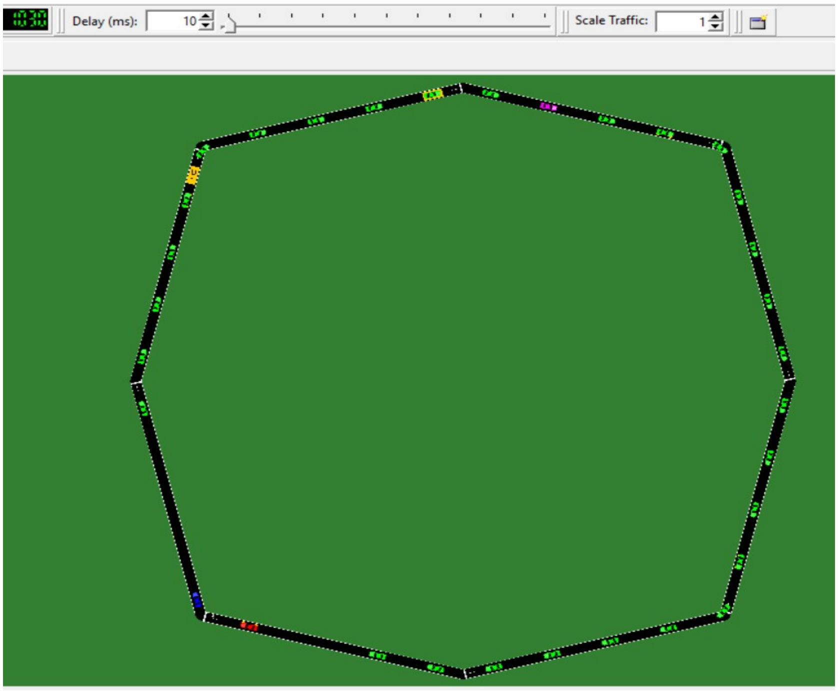
3.1.2 Simulation with the Q table after learning process

After the Q-learning process, I launch again a simulation based on the final Q-table. At each iteration, the algorithm chooses the best action (no randomness any more).

The **red** car is the learning car and the **blue** car is the leader car.

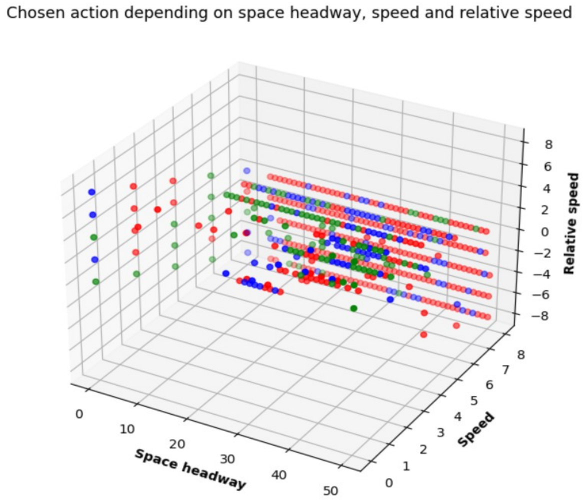
We observe that the controlled car stays at a certain “safety” distance of the leader car and its speed is quite the same than the one of the leader car (the distance between the 2 vehicles remains constant). There is no collision during the simulation time.

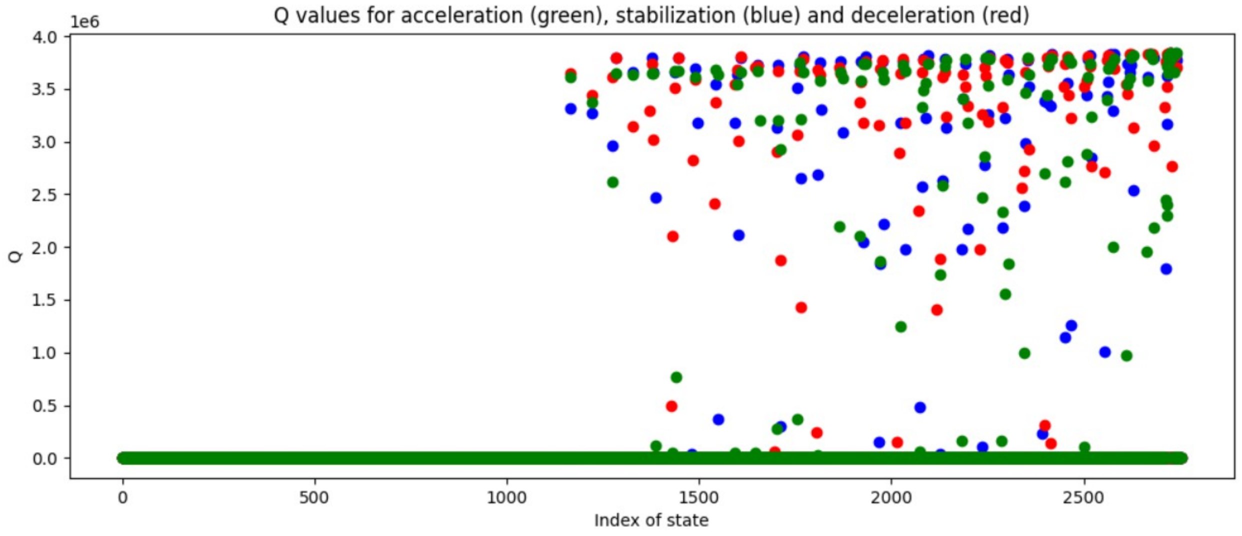
The resulting behaviour of the learning car is thus the one expected.



3.1.3 Q values depending on the states

We can see on those 2 charts the Q values depending on the states.
The color of the points depends on the action :
- the Q value for an acceleration is displayed in **green**;
- the Q value for a stabilization is displayed in **blue**;
- the Q value for a deceleration is displayed in **red**.





We see that there are a lot of states that have not been explored : the green “line” with Q = 0 corresponds to those states; it appears in green because I display the points for the acceleration in last but the red and blue points are under the green points. This can probably be explained by the fact that the speed of the leader becomes constant quickly as the cars controlled by Sumo try to go the fastest by avoiding collisions. On our ring, there are no obstacle, no traffic light, so there is quickly a uniformization of the movement.

3.2 Second configuration : reward depending on the safety distance

Here are the parameters I used to have the following results.

Parameter	Value
Max speed learning car	8
Max speed leader	3
MAX_SPACE_HEADWAY	50
MAX_SPEED	5
MAX_RELATIVE_SPEED	4
STEP_SPACE_HEADWAY	1
STEP_SPEED	1
STEP_RELATIVE_SPEED	1
ACTIONS	{0.9 , 1 , 1.1}
Number of other vehicles	30
Reward	-20 if accident 20 if in the safety range -10 if too close from the leader -5 if too far from the leader
ε	0.2 for 200 first iterations 0.1 after
α	0.9
γ	1
Number of scenarios	3000
Max number of iterations in 1 simulation	1000
Size of the ring	637m

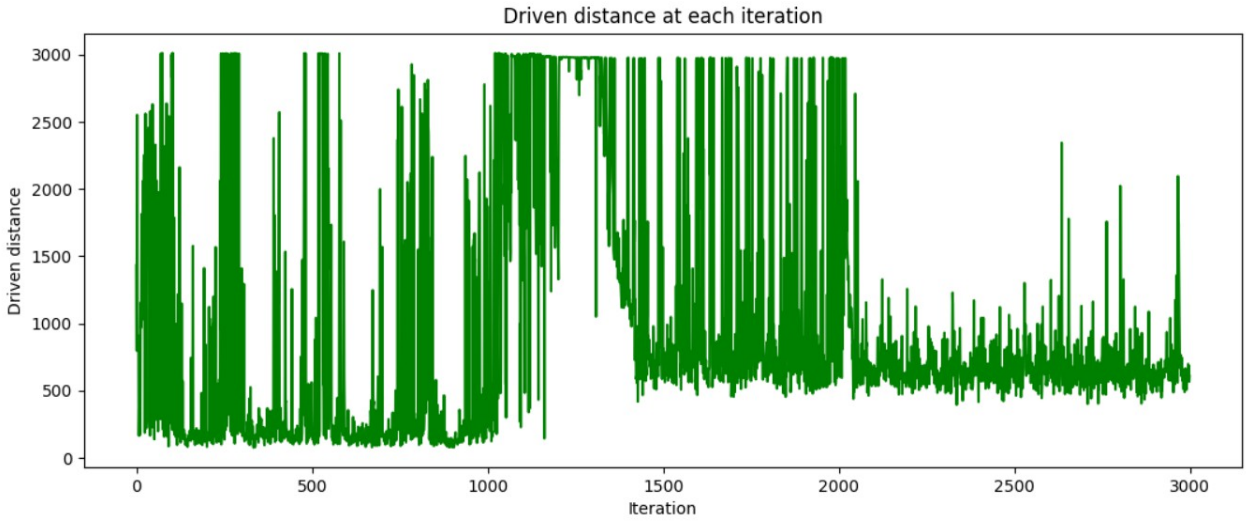
Number of states : 2754

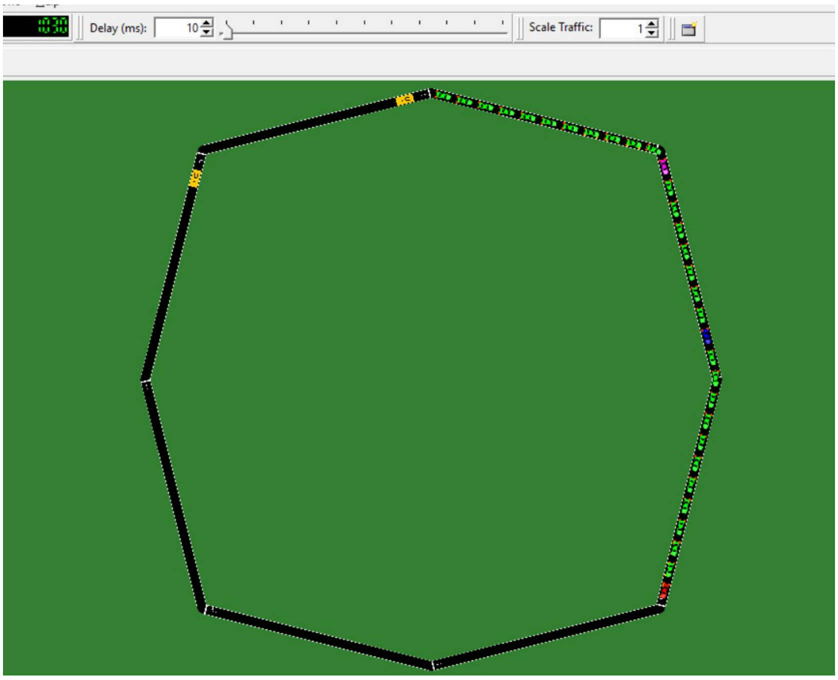
3.2.1 Driven distance chart

The first chart represents the driven distance by the controlled car at each simulation.
We see that the driven distance starts by increasing and reaching high values. But, from simulation number 2000, it stabilizes at about 500 meters, which is of course not optimized.

3.2.2 Simulation with the Q table after learning process

The simulation confirms what we observed in the previous chart. There is no collision, but the controlled car runs in a jerking manner. Thus, it does not optimize its speed and does not optimize the running distance.

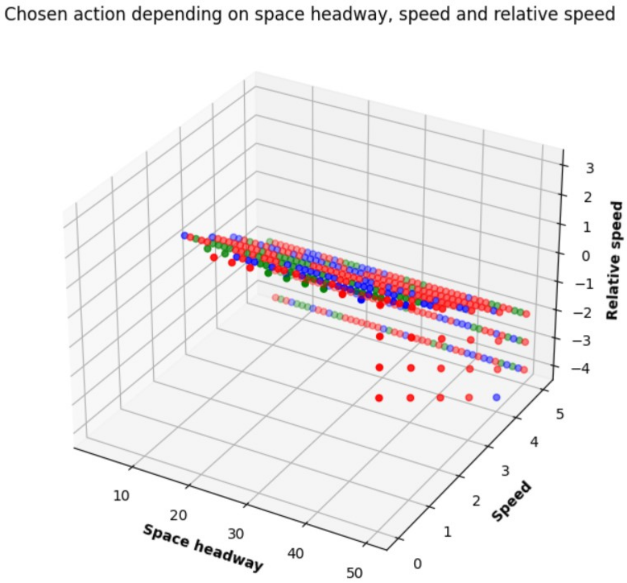


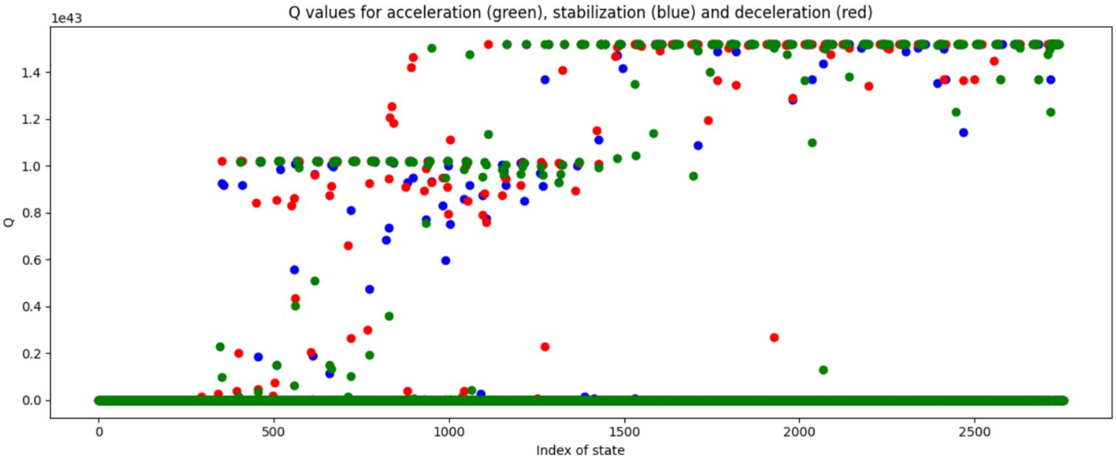

Delay (ms): 10

Scale Traffic: 1

3.2.3 Q values depending on the states

We can see on those 2 charts the Q values depending on the states.
As for the previous configuration, the color of the points depends on the action.
We see again a lot of non explored states (green line at 0 value).





3.3 Other tests

I tried to run a lot of different configurations, by modifying the maximum speed of the leader (trying to increase it), the discretization of the states, the number of iterations, the speed of the other cars, the reward... , but most of the time I got no relevant results: the learning car starts running, then stops itself a few ticks, then runs again, then stops, and so on... It avoids collision, but it does not maximize its speed and the running distance.

4 Conclusion

Thus, I'm really disappointed by these results. I found very few configurations in which I got expected results.

For me, there are 3 possible explanations :

- either I did a mistake in my implementation of the Q-learning, but I checked it several times, step by step, and was not able to find any problem;

- or I did not find the right parameters configuration, whereas I've tried a lot;
- or this problem is not really adapted to be solved by the Q-learning algorithm. Indeed, the problem of learning car works with continuous attributes such as the speed or the distance between 2 vehicles. Thus, in our Q-learning algorithm, we needed to discretize the values. But refining the discretization implies increasing the number of states. And the Q-learning will be less efficient. Maybe the use of the Deep Q-learning with neural networks would have been more adapted and done better results...