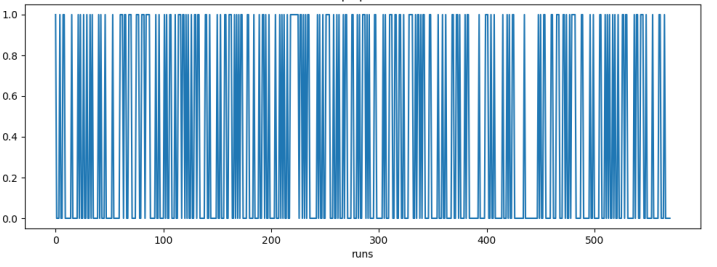
**RESULTS OF MOUNTAIN CAR PROBLEM BUILT ON RL-STUDIO**

**Starting point:**

* Alpha = 0.8
* Gamma = 0.9
* Epsilon = 0.9995
* Reward = 1 if reaches the top, 0 otherwise
* Steps per run = 20
* states = [x\_pos (-10, 10), x\_vel (-100, 100)]
* Actions = [0 (nothing), 1 (going to the left), 2 (going to the right)]

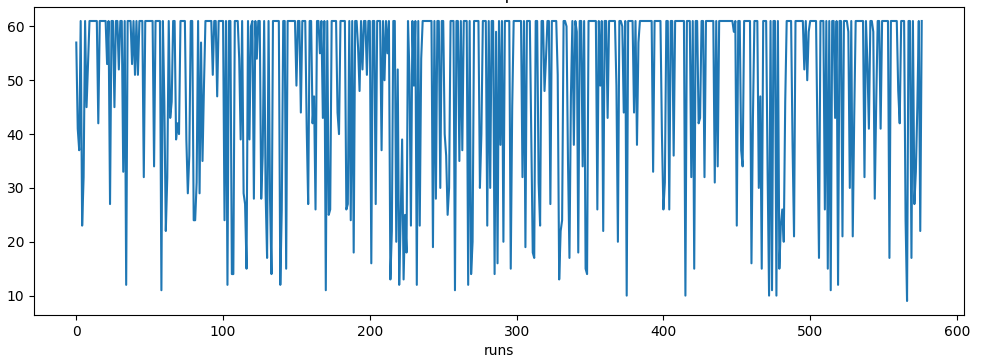
**Process**

* The firstexperimentes were carried out with a bug that avoided tha agent to learn. It was discovered analysing the q\_table progress (almost none).
* After fixing it, a second problem arose, the agent was reaching the top too little, so due to the long trip the car has to make to climb the mountain, it was hard to learn the steps.
* After making a little bit easier to climb (increase the intensity of the actions applied by the agent so it gains speed faster) the mountain and increase the number of steps per run to 60, we discover that the q\_table was not progressing as expected (one more time) and the agent didn´t learn. It was because the discretization of the actions was too broad and it was unlikely that the robot performed two same actions to reach the goal.
* After adjusting the actions so there was not too many, the robot was reaching the goal more often, but it was no clear if it was learning or not because of the graphs we were using to analyze its behavior. Since we were rewarding the agent with 1 each time it reached the top and with 0 otherwise, we see a graph like this:

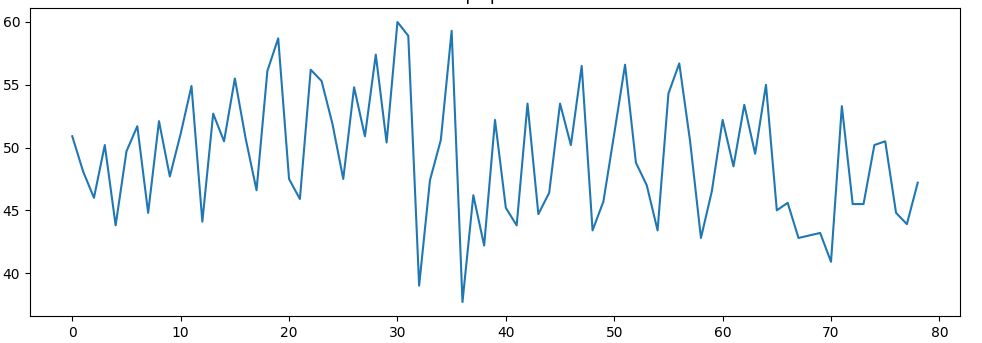


Not pretty clear what is happening, right?

* We replaced then the “rewards\_per\_run” graph by the “steps\_per\_run” graph:



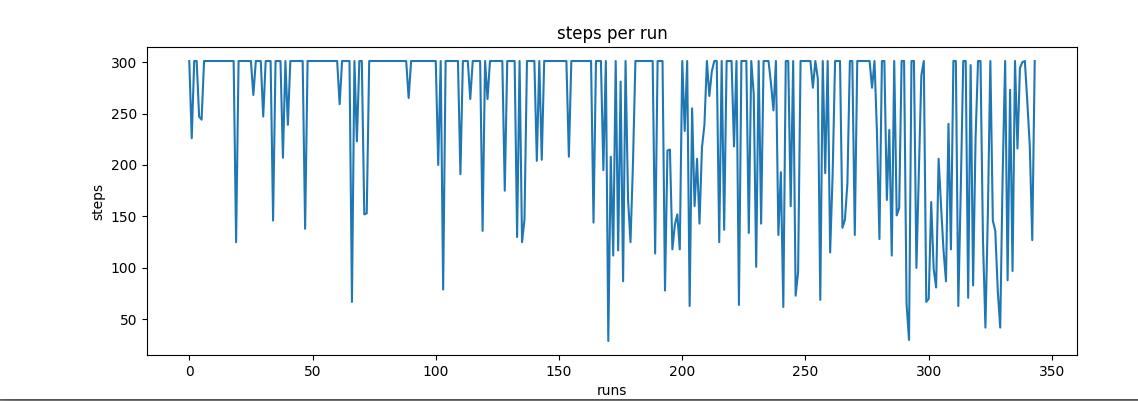
* Promissing but still not pretty clear, so we still plot this graph to provide on-line information, but we started to use the logs stored by the program to better analyze the behavior. What we did was to average the steps per run each 10 runs and plotting this, so we had a more normalized an easier to understand graph:



* Now we have a clearer way to analyze the behavour and the agent is learning, but it is not improving pretty much from the random behavior of the beginning, so the learnings are useless. Then, we decided to perform the following modifications:
  + Increase alpha from 0.8 to 0.9
  + Remain gamma at 0.9
  + Increase epsilon from 0.9995 to 0.99995
  + Increase in a factor of 2 the number of states (we have around 40 “x\_positions” and 50 “x\_velocities” now)
  + Increase the number of episodes per run to 300

With this configuration, we can reduce the risk of creating a dumb agent (two actions applied in the same step may lead to different next states). However, we increased again the number of states, so we must leave the agent learning much more time. Since we are getting older and we don´t want to invest too much time, we increased the experiment velocity reducing the time from action to action and reducing also the intesity of them. In this way, the episodes are shorter but the agent still needs to perform a consecutive not trivial correct actions to reach the goal.

**Final results**



In the graph and the attached logs we can see how the algorithm is progresivelly learning how to reach the goal. To be precise, from the episode 170 onwards, the agent sistematically reach the goal more often and in less steps.

