RL mini-project : Mountain Car environment

Authors : Rayan Harfouche & Tara [1] Fjellman
Spring 2024

1 Introduction

To be written after the rest of the report is done.

2 Environment

The Mountain Car environment is a RL environment in which the agent is placed in a valley between two hills. The goal is for it to learn to reach the top of the right hill. The agent can only apply a force of -1, 0 or 1 to the car. The state space is continuous and has two dimensions, the position and the velocity of the car. The domain is [-1.2,0.6] for the position and [-0.07,0.07] for the speed. The agent receives a reward of -1 at each time step, and a reward of 0 when it reaches the top of the hill. The episode ends when the agent reaches the top of the hill or after 200 steps.

3 Random agent

We start by running a random agent, i.e. one that selects its action at random. To analyse its performance, we inspect the duration of the episodes. Indeed, if the agent solves the task, the duration of the episodes should be lower than 200[2] (as this is the truncation time). Running the the agent for 1000 episodes, we obtain **??**. Looking at it, it is clear that the agent does not succeed at the task. Actually the agent just oscillates around the minimum.

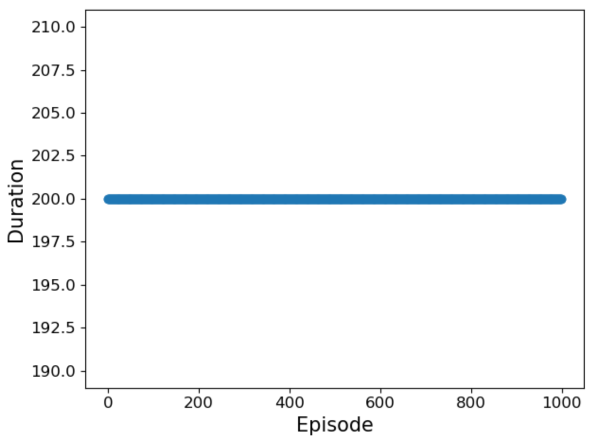


Fig 1: *Episode duration of random agent during training.*

4 DQN

We now turn to the DQN agent. Now the policy is actually learned and should lead to better results.

4.1 Vanilla version

Running 1000 episodes and computing : the duration of the episodes along with the episode-cumulated reward and loss, we get **??**. Looking at it we see that the behaviour is the same as for the random agent, i.e. the agent does not learn the task. This is due to the sparsity of the reward. Indeed, without reward, the agent does have information to adapt.

4.2 Auxiliary reward

To help the agent learn, we now introduce auxiliary rewards that will be added to the environment reward. Theses should promote exploration, contering the sparsity of the

[1]My administrative name is Tobia, so you will find me under that name in EPFL related databases.
[2]... of course except if it solves the task in exactly 200 steps.

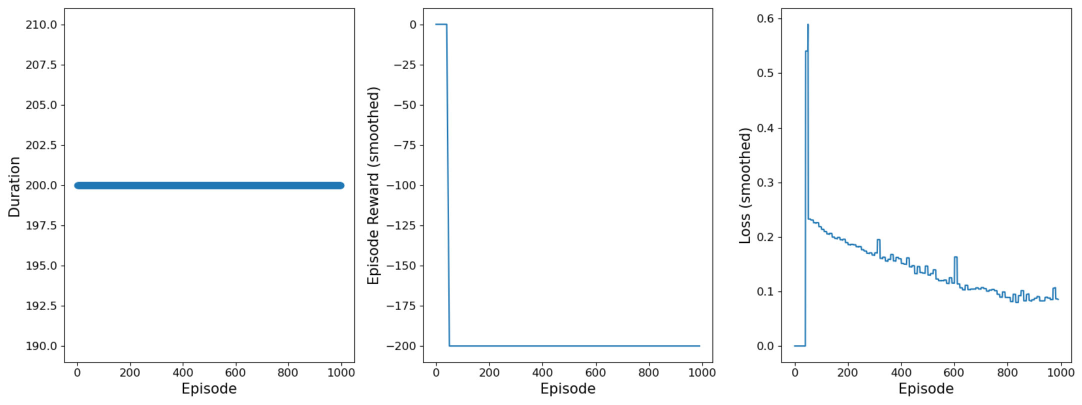


Fig 2: *Results associated to the training of a vanilla DQN agent.*

reward.

4.2.1 Heuristic reward

4.2.2 Chosing the function

Intuitively, we want this function reward the agent when it makes an effort to reach the top of the hill. We therefore decide to take a heuristic function that is monotonous in the height of the mountain car. The (normalised) height can be deduced from the position as

$$h(x) = \frac{1 - \cos\left(\frac{x - x_0}{x_r - x_0}\pi\right)}{2},$$

where x_0 and x_r respectively are the average-starting and ending positions. A simple and versatile reward function is therefore

$$f(x) = Ah(x)^n - A$$

for $A \geq 0$ some constant we call reward-factor and a given $n \in \mathbb{N}^\star$. The $-A$ offset is there to make extra sure that for all values of the scale factor A, the agent does not get tempted to stay and collect the heuristic reward, we decide to offset the function by its maximal value (i.e. A). This way, the heuristic reward is always negative, and the agent is encouraged to reach the top of the hill. To check that this version of the agent is learning, we plot (for $A = ...$) the loss (see **??**), the duration (see **??**) and the collected reward (see **??**) during learning. Looking first at **??**, we see that the agent solves the task in less than ... episodes. Now turning to **??**, it is interesting to notice that during the time the agent fails at the task (for episodes ≤ 250) the auxiliary reward increases. This means that the agent is exploring heigher and heigher states. After some time, it actually reaches the top of the hill, and so learns that it can increase its environement

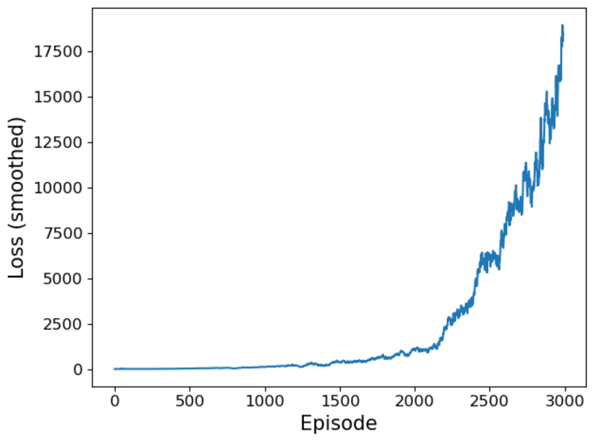


Fig 3: *Evolution of cumulative training loss as a function of the episode for a heuristic reward agent. The reward factor is set to 3.5 and the loss is smoothed with a window of width 10.*

reward by doing so. This is exactly what we wanted to achieve with the heuristic reward ! As for the loss, we see that it is increasing, which might come to a surprise. Indeed in machine learning we are used to seeing training lossess decrease. In this case however, the loss is just an indicator of how well the agent is learning the Q function. The observed increase actually corresponds to the fact that the learning of this function gets harder as its domain increases.

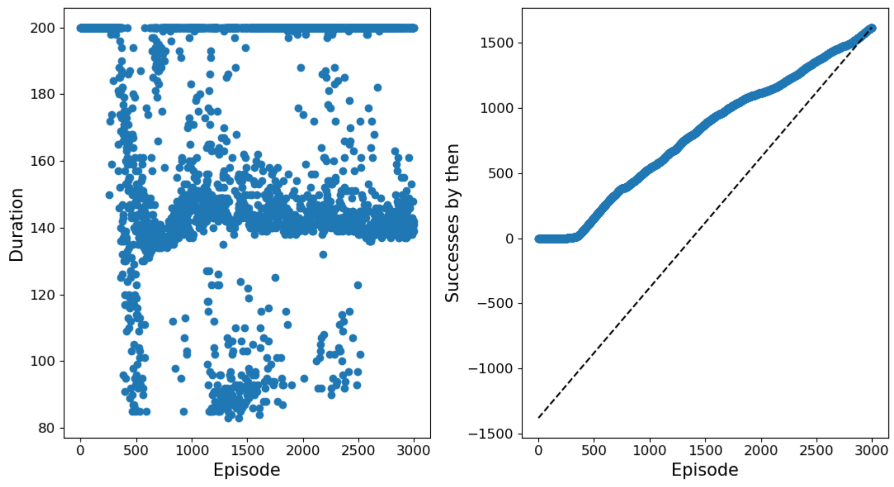


Fig 4: *Evolution of episode duration and cumulative number of successes as a function of the episode for a heuristic reward agent. The reward factor is set to 3.5.*

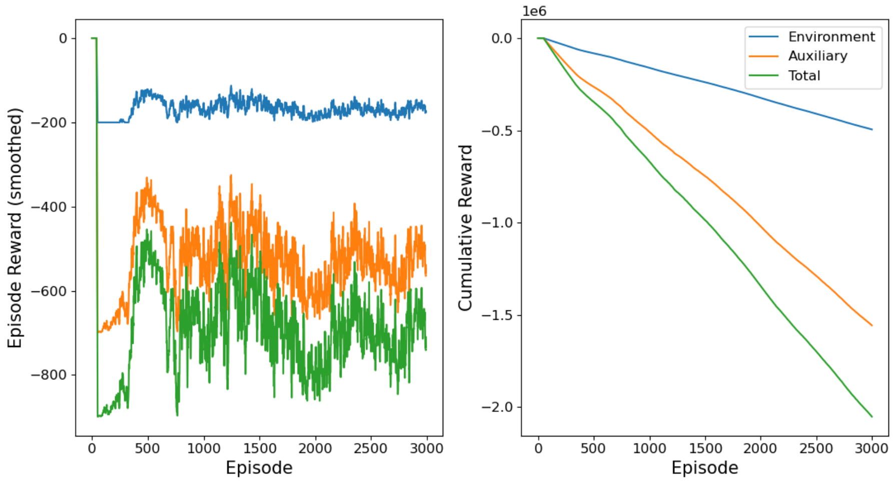


Fig 5: *Evolution of cumulative reward and cumulated cumulative reward (over the different episodes) as a function of the episode for a heuristic reward agent. The reward factor is set to 3.5 and the cumulative reward is smoothed with a window of width 10.*

4.2.3 Auxiliary reward scaling

For the auxiliary reward to be useful, it should intuitively not be too small. Otherwise the situation would be no different than for the vanilla agent. As for the upper bound, we impose that it should not dominate the environment reward. Indeed, otherwise the agent could learn to collect the reward during the full 200 steps instead of reaching the goal. The exact value of this upper bound is not easy to express analytically, but we expect it to be of order 1. This provides us with a crude guess, which we can refine by testing different values. To analyse the effect of the reward factor on the performance, the

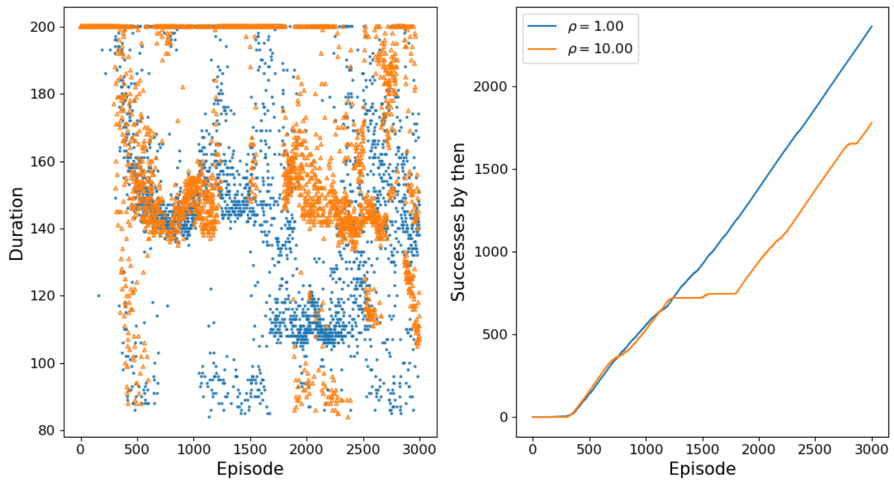


Fig 6: *Evolution of episode duration and cumulative number of successes as a function of the episode for a heuristic reward agent. The different colours correspond to different values of the reward factor*

heuristic agent was run for multiple valuees of the parameter. The corresponding results are shown in **??**. We see that ...

5 RND reward

In this section we consider the results associated to the RND auxiliary reward agent.

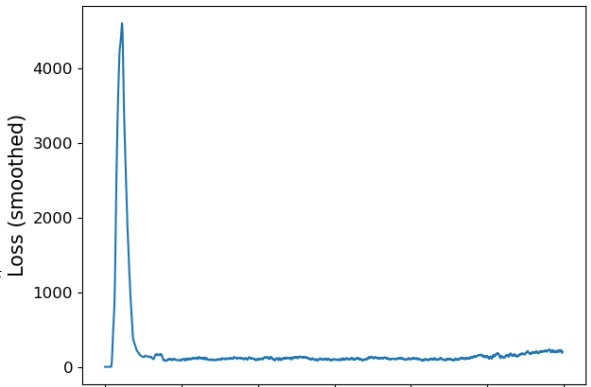
5.1 Normalisation

As recommended by the hint, we standardise the input and output of the RND network. The input standardisation is done since we want the output to vary as a function of how different the observed state is compared to the previously visited ones. Indeed, subtracting the typical state (i.e. the mean one) provides a good idea of how different the new input state is. The normalising by the typical distance from the mean state (i.e. the standard deviation) is done to make sure the input is of order 1, which is a good practice for neural networks to avoid instabilities.

The standardisation of the output is instead done to control the characteristics of the RND reward attributed to the states. Indeed, by subtracting the mean we make sure that typical states are penalised while new ones are rewarded; while normalisation makes it possible to tweak the the order of magnitude of the reward. The clamping is done for this same last reason (as NNs can produce a heavy tail output).

5.2 Reward-factor scale

Mostly the same argument applies as for the heuristic reward. This time however, if the factor is taken too large, we expect the agent to prefer exploring instead of ending the episode by reaching the top of the hill. Indeed, when the agent



will have reached the top of the hill a few times, the states that will have allowed it to do so will have a low RND reward. This is because the RND network will have seen them many times, and so the prediction error will be low. * crude scale : over an episode the rewards should be comparable ??

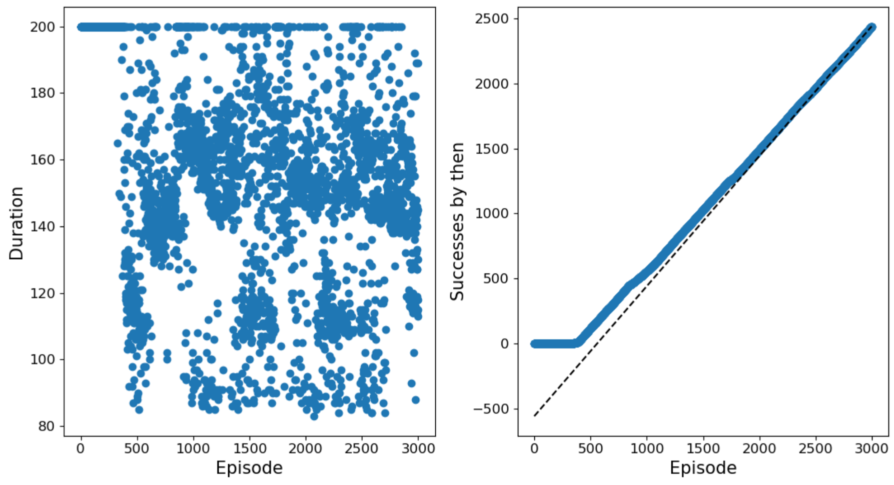


Fig 8: *Evolution of episode duration and cumulative number of successes as a function of the episode for a heuristic reward agent. The reward factor is set to 3.5.*

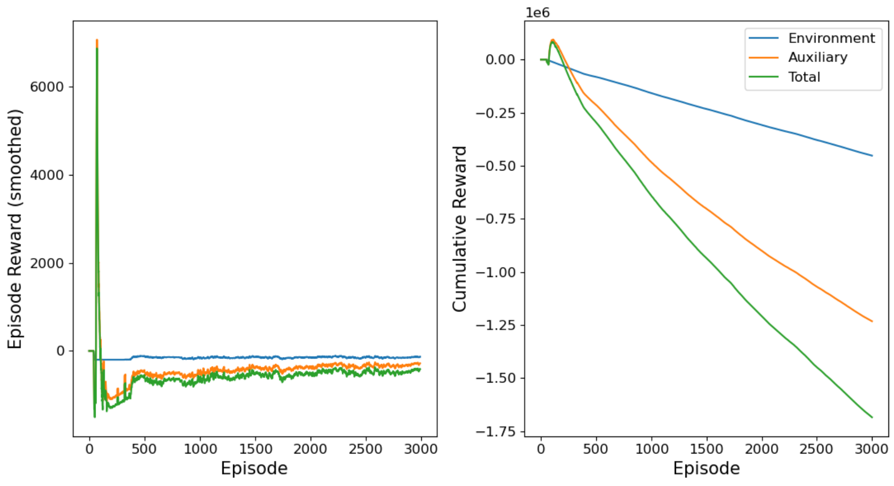


Fig 9: *Evolution of cumulative reward and cumulated cumulative reward (over the different episodes) as a function of the episode for a heuristic reward agent. The reward factor is set to 3.5 and the cumulative reward is smoothed with a window of width 10.*

5.3 Comparison to heuristic reward

* difference in behaviour : we oscillate between the two possible solutions. This can be explained by the fact that the RND

reward penalises the agent for repeating
moves, and therefore leads the agent to switch its method more.

6 Dyna

The Dyna algorithm relies on the discretization of the phase space (space of positions and velocities). The main parameter that will be varied in this section is this section is the size factor α. We take as a reference of the step sizes the vector suggested in the project description $(\Delta x_0, \Delta v_0) = (0.025, 0.005)$. We then test discretizations of the form $(\Delta x, \Delta v) = \alpha\ (\Delta x_0, \Delta v_0)$ for different values of α. Another parameter is the number k of (state,action) pairs randomly sampled among all the visited state-action pairs. We fix it to $k = 3$ and we will provide a note about it at the end of this section.

6.1 Dyna manages to solve the task for some values of α !

When considering the original discrtization ($\alpha = 1$), we observe that the task is solved, in the sense that successes (defined by episodes where the target is reached before 200 steps) are observed. As in the previous sections, we plot the duration of each episode during the training, as well as the number of success with respect to the number of episodes. In Figure ???, we show the result for $\alpha = 1.5$ considered as a value allowing a good training (better than $\alpha = 1$). We also plot results for values $\alpha = 0.55$ and $\alpha = 4.5$.

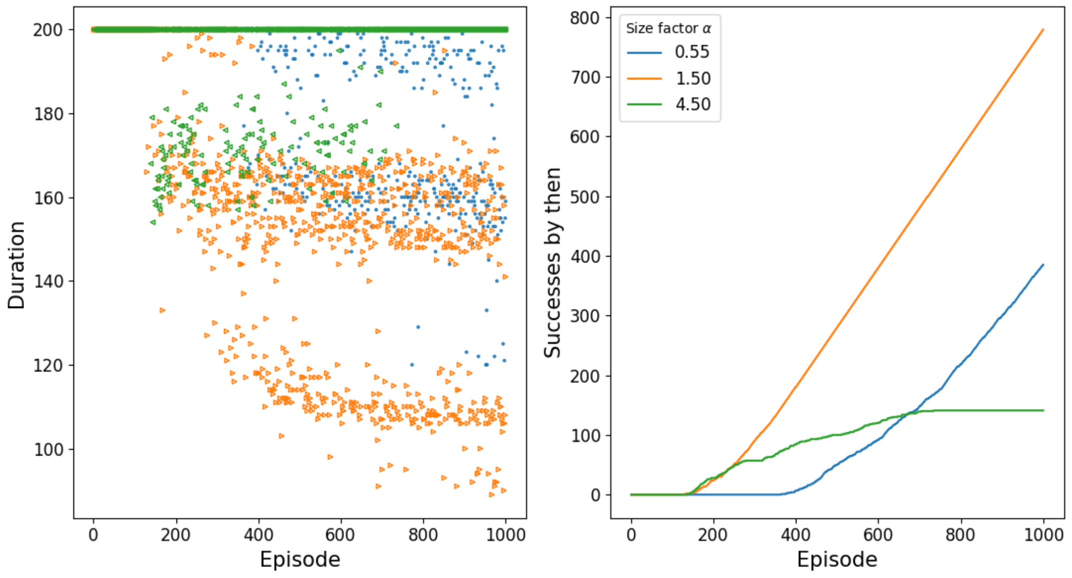


Fig 10

Value $\alpha = 1.5$ shows a good performance as first successes are observed before 200 epsiodes. Then the line of success wrt to the number of epsiodes becomes, for higher number of episodes, a line of slope 1. Interestingly, for that discretuzation size, a pattern (that will be discussed later) appears. For a lower value $\alpha = 0.55$ (fine grained discretization), performance is affected in the following way: the first successes are observed only after around 350 epsiodes, and the curve becomes a line of slope lower than 1 (meaning that even after a high number of iterations, some episodes are failures). Value $\alpha = 4.5$ also shows a low perfomance as the number of successes starts stagnating (less than 150 successes are observed after 1000 episodes). For even too high values like $\alpha = 8$, no success is observed in the window of the 1000 first episodes. This is due to the fact that ???. In the same way, for too low values like $\alpha = 0.3$ no success is observed within the 1000 first episodes. Explanation ???.

At each episode one can compute the euclidean distance between the matrices Q before and after the episode. This will provide a measure of the Q-value update. Figure ??? shows the evolution of that the Q-value update step for a simulation with 10000 episodes, using the most suitable value $\alpha = 1.5$ among the ones that we have tested. As before, we consider $k = 3$. Notice that the y-axis is in log-scale As expected, one can see that the Q-value update step overall decreases with the increase of the number of episodes. This indicates that the update equation: $Q(s,a) \leftarrow \hat{R}(s,a) + \gamma \sum_{s'} \hat{P}_{s,a}(s') \max_{a'} Q(s',a')$ is converging towards a fixed point. The decreasing is however not monotonic. Peaks are observed and correspond to moments at which the estimation of the Q values undergoes a big change, corresponding to the agent learning new strategies (or perfecting). Notice that the decrease of ΔQ does not plateau at the last episodes (setting the x-axis to a log scale allows to see that the decrease continues).

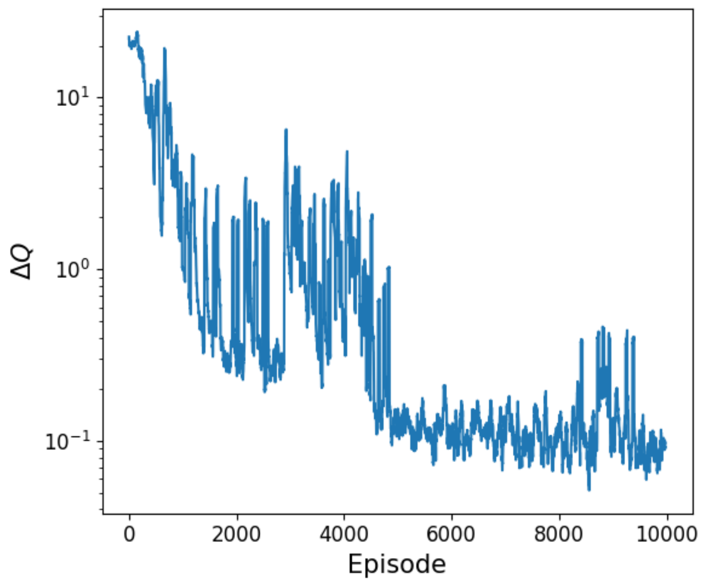


Fig 11

6.2 Intuitive explanation on why it works

Having discretized the space, we dictate to the agent which action to take based on its projection on the grid of states. The main advantage that this allows lies in the fact that

6.3 Q values table and characteristic trajectories

In the following, we show the estimation of the Q values after the learning. In Figure **??**, we represent for each state s (located by a position and a velocity) the value $\max_a Q(s,a)$. On top, we represent three trajectories each corresponding to one the groups of episodes described above. The second subplot indicated the set of states that have been visited (1 if visited and 0 if not). Last part of the figure shows the the number of times each of the states has been visited. Simulation done for 3000 episodes, $k = 3$ and $\alpha = 1.5$. We choose to represent the three trajectories in the same plot in order to save space in the report.

Interestingly, the plot of Q-values shows a spiral starting at the bottom of the mountain and velocity 0. The spiraling behavior is followed by the trajectories represented. One can notice that in the lower part (positive velocities), when approaching to goal position, the $\max_a Q(s,a)$ values start approchaing zero values. This means that being present in those states, one would not expect to recieve a lot of negative reward before the end of the episode, in other words, the state easily leads to the goal. Matrix of counts illustrates the fact that after 3000 epsiodes, most occupied states are disposed in spirals. Interestingly,

`../runs/dyna/dyna-k=3-ss_coef=1.5_eps=3000/figs/Q_final_matrix.png`

Fig 12: *Qmatrixcounts*

a peak of count can be seen at the extreme left position and zero velocity, corresponding to cases where the car hits the wall.

6.4 Bonus

6.5 Note on varying k

7 Comparison of the agents

In this section we compare heuristic and RND DQN agents to the Dyna agent.

We first do so for the training phase by training each of these agents for 3000 episodes, and then plotting their associated enverionement reward evolution. The results are shown in ref.... We can see that ....

Now we compare the performances by running each of the trained agents for 1000 additional episodes in 'evalutation mode', i.e. using a greedy policy. The plots of the corresponding episode durations are shown in ref.... To reduce stochastic fluctuations the results are made on the same environement seeds across the agents. We can see that ....

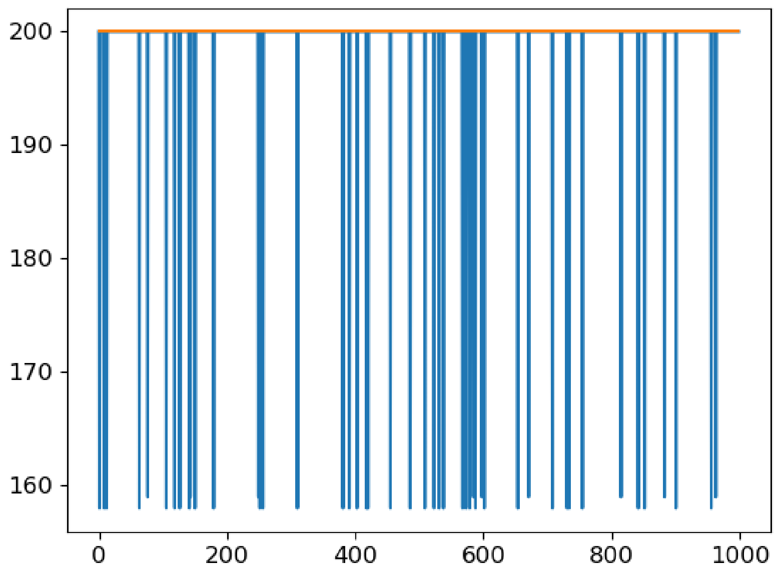


Fig 13: *Episode durations of heuristic and RND DQN agents and Dyna agent.*

8 Conclusion

8.1 Two types of solutions ?

From the plot of the episode durations, we can see that the agent seems to have two distinct behaviours. Indeed, we can see that the density of points is larger for $90 < d < 120$ and $150 < d < 180$ than in between of these ranges. Investigating this more closely (inspecting the trajectories associated to these performances), one can trace this difference back to the initial condition.

If we start on the right of the minimum, we are able to reach the goal fast by only 2 large swings : the first to the left and the second to the right. If we start on the left of the minimum, the solution takes longer. Indeed, we need to go right, then left, then right again. This is due to the fact that the car has to gain enough speed to reach the top of the hill. Notice that the distinction between these two solutions is not that sharp. This is due to the fact that the initial condition is uniform over $[-0.6, -0.4]$, and so the agent sometimes starts close to the minimum, and therefore is not exactly in any of the two cases.

9 Appendix