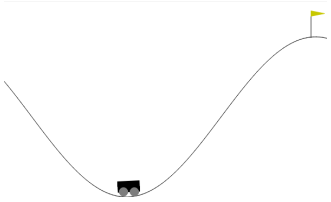
LECOMTE Alexandre GOZLAN Gabriel

Final report
Project: Mountain car (MP1)

Group's members: LECOMTE Alexandre ; GOZLAN Gabriel

The goal of this rapport is to study the performance of different algorithms of reinforcement learning on the well known "Mountain car" problem (an environment for reinforcement learning proposed by OpenAi)

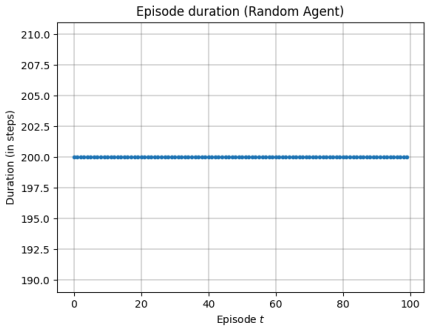


First glance:

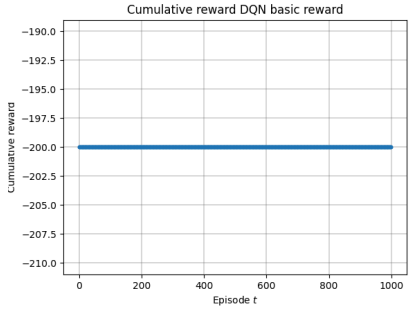
Firstly we want to look at the environment with an agent who acts in a uniformly random manner.

This picture represents the environment of the mountain car problem. The little car has to reach the flag at the top of the mountain. And she can only accelerate to the left and to the right. The rewards = -1 for all action until the car reaches the flag or does 200 actions.

The complexity is that the environment is continuous and that the car must use his inertia to reach the flag (a full right or left is not enough) so there is like a sequence of right and left to reach the top.



Over 100 episodes (1 episode = 1 "try") the full random agent still finishes his try in 200 steps (1 step = 1 action). The main information is that this duration is due to the fact the car never reaches the flag. That is previsible because the strategy to reach the flag is quite complicated, and choosing action randomly will never allow the car to do the right actions.



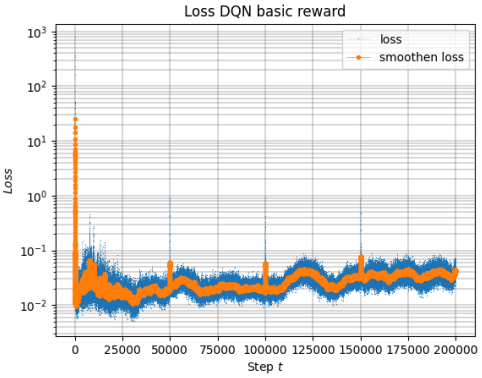
DQN's analysis:

We will now look at an agent trained with the DQN algorithm.
In the following case the environment has not been changed, so the reward is = -1 for each action.

Even after 1000 episodes we can see that the cumulative reward per episode is equal to -200 (morally, the duration of the episode is then = 200 steps) so, over all these episodes the DQN agent does manage to reach the flag once.

Nevertheless, the loss function is converging. The idea is that the Qtarget and the Q used for the policy converge on each other. (We don't reach 0 because of the replaybuffer which is not infinite, coupled with SGD. So there is a little noise, but that does not really impact the performance of the agent.)

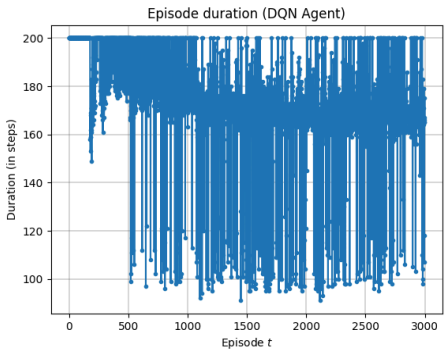
$$L(\theta) = [((r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^{target})) - Q(s, a; \theta^{pred}))^2]$$

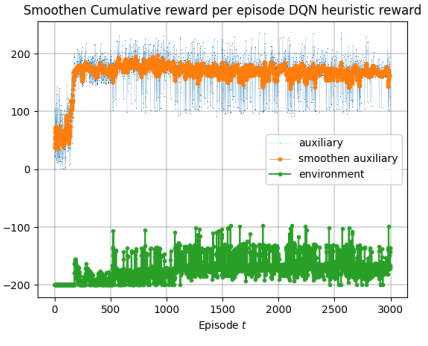


There are some "spikes" in the loss for the episode because of the QTarget refresh. This is normal, the loss changes brutally due to the refreshing. But it can be a problem for the convergence if this happens too much. That is why it is important to have a little QTarget refresh frequency. Too often leads to a noisy loss function which converges with difficulty.
In our case we have chosen ***QTarget_Frenqency = 1 / 50000(steps)***

DQN with heuristic rewards:

The auxiliary reward chosen is increasing with the height (or the distance between the first state and the position x of the car). The idea is that the car is in a bowl-shaped mountain, so to reach the flag it must first reach a higher place. The reward is continuous and exponential as the car goes higher.

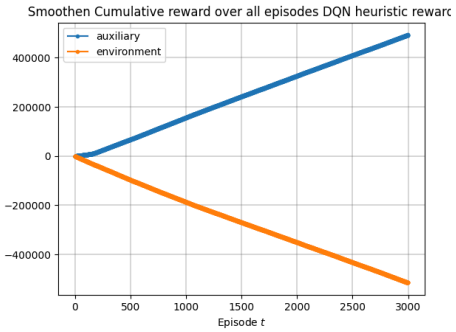


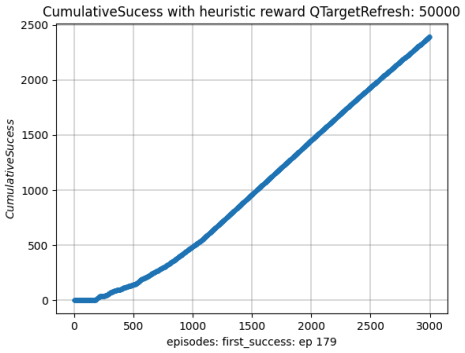


Average norm of auxiliary reward = 0.5

From the duration plot we can see that the agent manages to reach the flag quite often.

LECOMTE Alexandre GOZLAN Gabriel

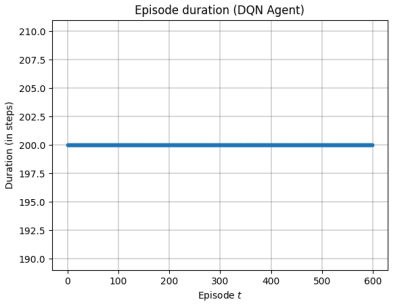


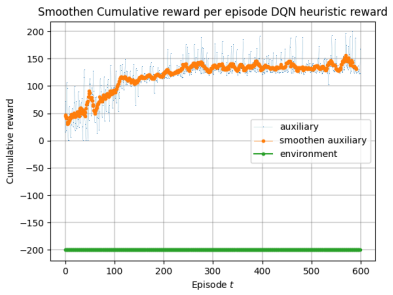


However there is some big variation in the duration per episode. Beside the fact that the car sometimes fails to reach the flag, there are some episodes where the duration is more around 180 steps, than 100 (in the best cases).
On the cumulative reward plot we observe a stability in the rewards per episode. After some training episodes (300), the auxiliary reward is stable around +180 with some noises. Same for the environment reward (after 1000 episodes) which is around -180. We see that in our case the norm of the auxiliary rewards is in the same order as the environment rewards.

From the cumulative reward over all episodes we see that the rewards are stable over the episodes. The performance of the agent is good. From the cumulative success, the probability that the agent reaches the flag under 200 steps is around 80%.

For the loss, there is a pic when the agent firstly reaches the flag. This is because the event has not been seen before, so the neural network Q has not been trained for. *(The drop at the end is do to the smoothening)*

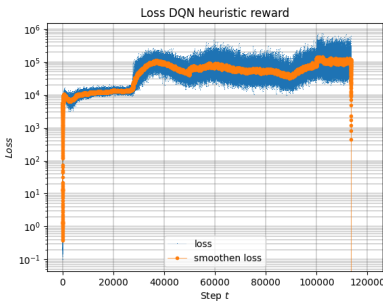


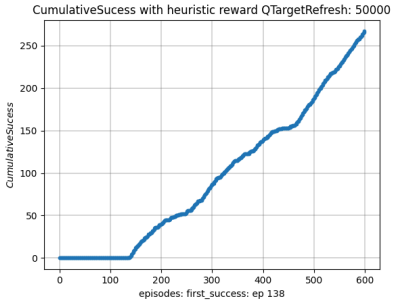


<u>To compare with other norm of auxiliary reward:</u>

Here the norm = 0.05

The agent doesn't win once in 600 episodes. Morally the auxiliary rewards are so weak that we are back to the basic case with only environmental rewards.





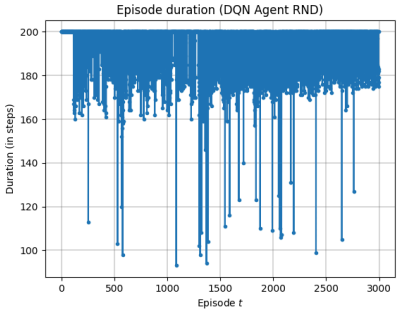
Here the norm = 50

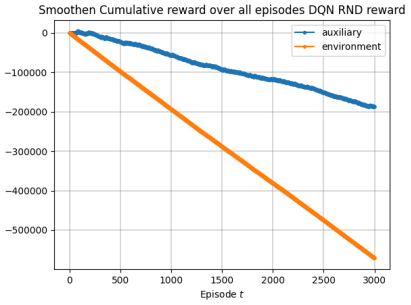
This case is the inverse of the case with norm = 0.05. The agent still manages to reach the flag, but less often than the first case. This is due to the big impact of the auxiliary rewards on the convergence of the loss function. There is no more convergence, the rewards are so big that for each action the loss changes drastically. That leads to a poorer generalization of the DQN agent.

The convergence of the loss function is a good indicator to know if the agent will generalize well or not. But this is not an indicator on the capacity of an agent to "succeed at a game" once. On another hand, in our case big auxiliary rewards make the car reach the flag faster (but with less consistency). There's a balance to be struck.

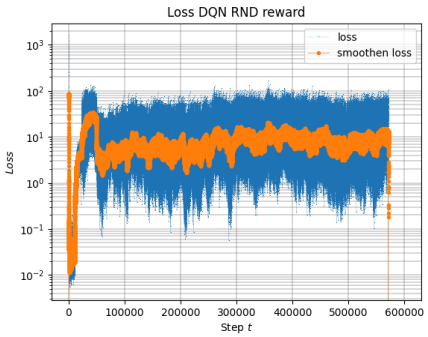
<u>DQN with RND rewards</u> (Reward factor = 2)

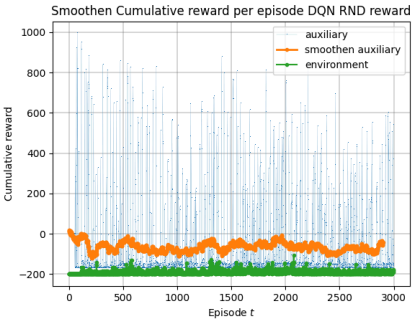
There are 2 main differences with the first case: The instability of the auxiliary rewards, and its norm which can go over 900.

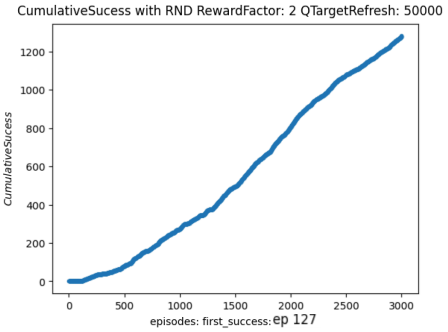




LECOMTE Alexandre GOZLAN Gabriel

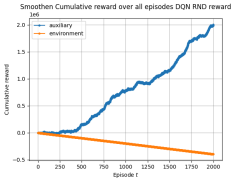


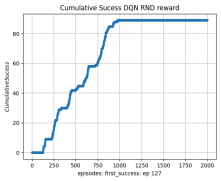




- Duration: The agent manages to reach the flag with some stability around 170 steps
- Cumulative rewards: The cumulative auxiliary reward is noisy, which is a problem because then it is hard to learn a stable policy.
- Loss: As for the heuristic reward, there is a pic for the first win of the car, and another just before because of the auxiliary reward which increases rapidly (note: there are some episodes without auxiliary reward at the beginning to store enough sample for the averaging)
- Reward: There are big variation in the auxiliary reward (between 1000 and -200)
- Success: The agent manages to win, but it is not consistent (probably because of the variations of the auxiliary reward that are too big, so some episodes the agent would prefer to lose with more reward)

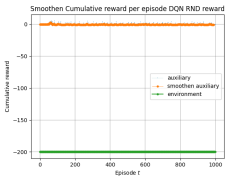
The idea is the same as for the heuristic reward: If the reward_factor is too low then the car never reaches the flag. *In our point of view, a solution should be to reduce the reward_factor over the episode when the car starts to reach the flag.*

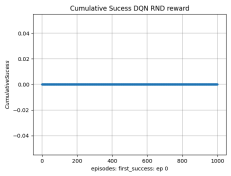




To compare: if the factor_reward is to big (=20)

The cumulative auxiliary reward explodes. Like mentioned before, the car still reaches the flag, but because of the reward that is too big, the car can't generalize (or the policy converges). We can see that the car doesn't reach the flag anymore after 1000 episodes.

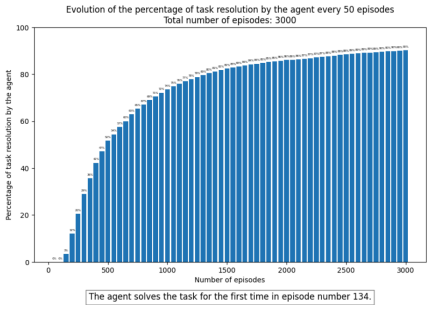




if the reward_factor is little (=0.01)

Obviously in this case the auxiliary reward is so little that we are back to the basic case with only environmental reward.

Dyna's analysis:

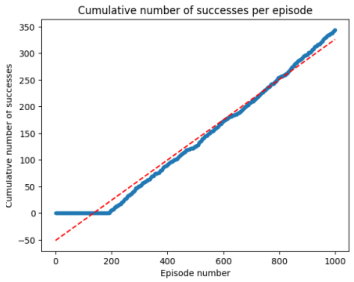


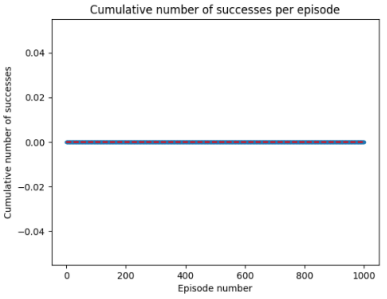
Now let's take a look at the Dyna agent, which is model-based and works by discretizing the environment.

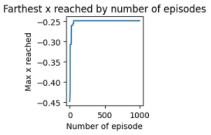
The figure opposite gives an overview of the agent's task resolution: 1st success between 100 and 200 episodes, and the frequency of successes increases logically with the number of episodes, ending up with 90% success, i.e. around 2,700 successes for 3,000 episodes (even if there is no auxiliary reward).

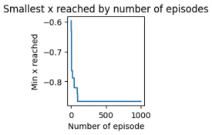
Intuitively, if we use too large bins, we don't give the agent enough feedback: this lack of precision doesn't allow the agent to penalize bad paths sufficiently, and as a result he won't be able to reach his goal.

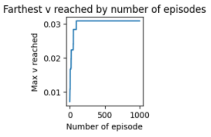
On the other hand, with bins that are too

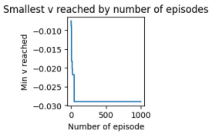


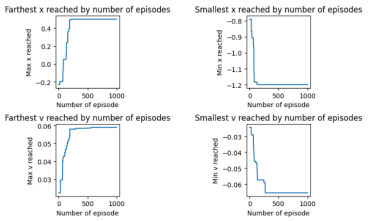




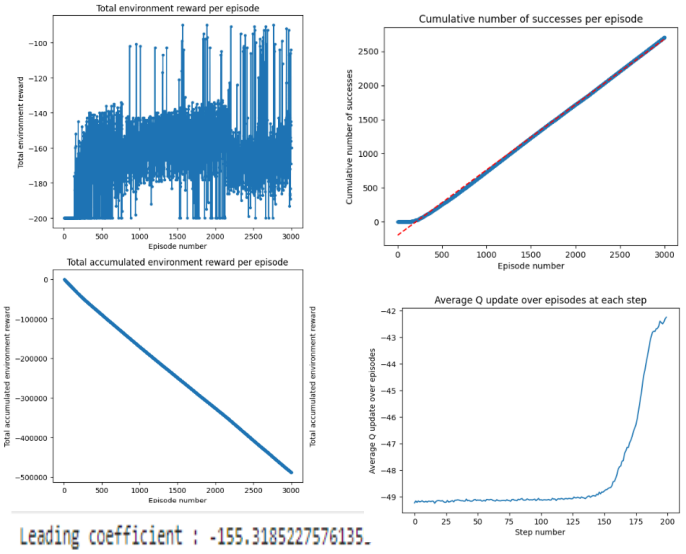






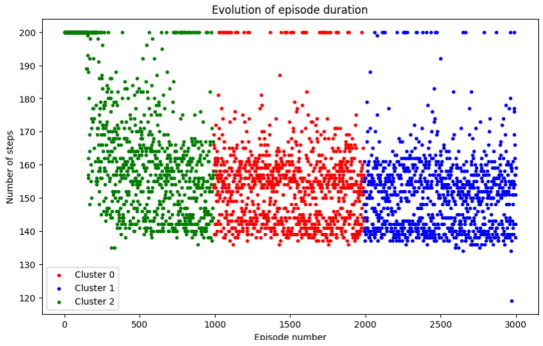


small, the agent will have enough feedback to find the right path, but the limit of 200 possible actions per episode will maybe prevent him from reaching the end (with “normal” bins, the agent very rarely finishes below 100 steps; if we double the number of bins, it's not even certain that the agent can reach the goal even if he makes the best choice every time).
On the right, we can see that experimentally, bins that are too large (by halving the number of bins) prevent the agent from succeeding in the task even after 1000 episodes, while with twice as many bins (on the left), the agent succeeds. the task but is not a perfect agent.



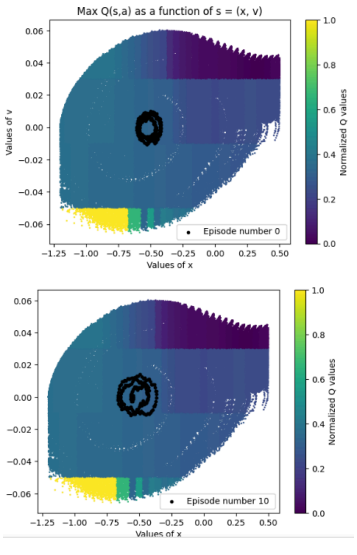
The average value of the Q update for the first 100 steps is almost constant, as the agent almost never solves the task, and increases exponentially from 150 steps onwards, as the agent almost always finishes the task within 200 steps.

For episodes, from the first success onwards, the agent almost always solves the task, since the graph of cumulative successes shows almost a straight line, i.e. as soon as the agent solves the problem once, it is then almost perfect.
This is also visible on the rewards graph, in particular with this horizontal segment of -200 at the beginning, and also the leading coefficient of the total accumulated reward plot indicates the average number of steps per episode, which is around 155.

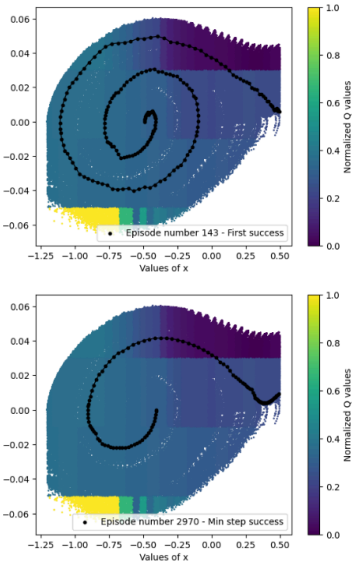


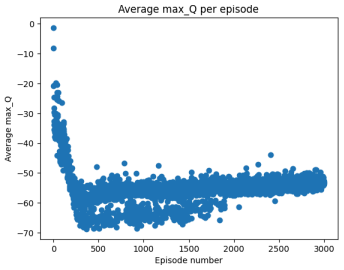
The Dyna agent manages to solve the task by taking into account the possible future gain for the next state from the state in which it is, which allows it to take into account the implication of its actions, and therefore that sometimes it is necessary to go back to go further. In addition, the agent manages to precisely predict the results of its actions because it is model based, and therefore converges in a more optimal way.

In the episode duration plot, we can see a pattern: apart from the times when the agent doesn't solve the task, particularly at the beginning, the number of steps required remains almost the same, and oscillates between 170 and 140 steps, with a “hole” in the middle of the plot: when the agent succeeds in the task, it generally does so in two different ways, with one a little longer than the other.



We have represented the max Q normalized as a function of the state, and we notice that the values are low when the speed becomes important (because then we are on the right track to solve the task), and the same when the x becomes important (a “diagonal” decrease). The paths taken by the agent are reminiscent of the phase trajectory of a damped mechanical oscillator, but with a clockwise path: this is logical because the agent functions in the same way: it understands that it it is necessary to increase the speed when the maximum position reached is not sufficient by going backwards, and when the speed is high and the position small, it understands that it must move forward, like a sort of opposite to the friction of an oscillator.
We added trajectories of the agent (in black). We can see that at the beginning (on the left), the agent

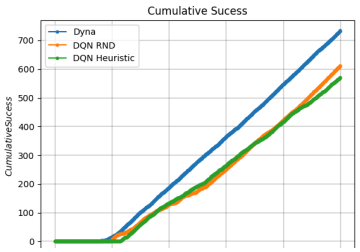


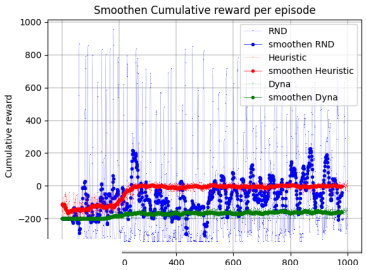


oscillates in a small area, because it discovers the environment and does not stretch enough. On the right we have kept two other important episodes: the first success and the quickest success. The link with the plot of the duration of the episodes then appears naturally: the agent actually succeeds in the task in two ways: either by starting directly on the left to gain acceleration and thus succeed more quickly (around 140 steps), or he loses a little time by first going right (around 170 steps).

Finally, in this bonus section, we can also see these two existing paths separated by a blank: the Qs at the top are larger and are associated with a shorter duration (as Q is always negative), while the Qs at the bottom of the separation are associated with a longer duration (start from the right).

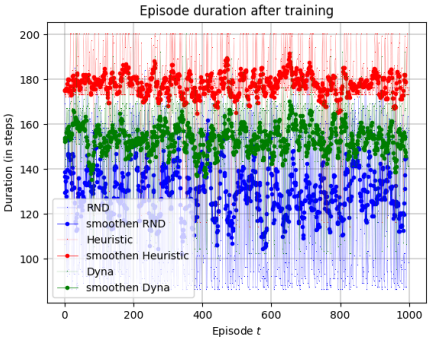
Results during the training:

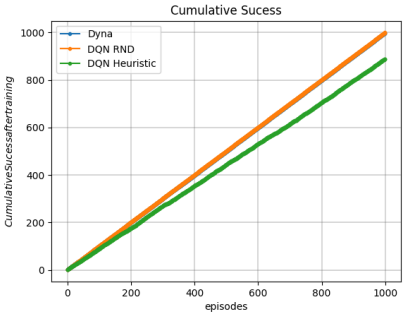


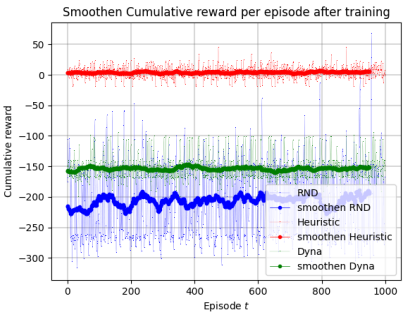


We observe a trend for the DQN and another for Dyna. The cumulative success of Dyna is really stable compared to DQN.
For the cumulative reward, DQN with heuristic reward and Dyna are both stable, while the DQN with RND is really noisy. (As expected)

Results after the training (on same seeds):







As explained for the results during the training, the DQN agents have some instability, so the results after the training are a little impacted (only for the cumulative success).

The episode duration plot is interesting: There is a clear hierarchy between the models. The DQN with RND has the best results (it's the fastest), with just behind the Dyna, and after the DQN with heuristic rewards.

For the cumulative success, the fact that the DQN with RND has the same performance than the Dyna is somehow "lucky". If we look at the cumulative success during the training we see that the performance of the DQN depends on when we stop the training (it's the same for the DQN with heuristic rewards). So on average, the DQN with RND has the same performance as the DQN with heuristic reward, and the Dyna does at least better than both.

The cumulative reward underlines well the difference between the algorithms. Dyna has only environmental rewards, DQN with RND has big variation due to the "novelty" behavior of RND, and DQN with heuristic rewards has bigger and stable rewards.

Conclusion:

In conclusion, DQN and Dyna have both pros and cons.

The DQN with heuristic reward is not really competitive with the RND one and Dyna in general. However in our "simple" problem, because it's easy to find good heuristic rewards, the heuristic version can be an alternative to the RND because the performances on the cumulative success are similar and seems to be more stable on the long run (for a training over a high number of episodes). However the RND version outperforms the heuristic version on the duration of episodes. And Dyna outperforms the heuristic version in both fields.

Lets compare Dyna and DQN with RND: Dyna outperforms DQN on the stability of the training and so on the cumulative success. This is previsible because Dyna is model-based so a little variation in the environment does not impact the agent (compared to DQN). However the DQN outperforms Dyna on the episode durations. It's also previsible because the main idea behind the RND is to improve the strategy of exploration, so the agent can easily find a better policy (we will not just be focused on reaching the flag).