Rapport du projet 13

GESTION D’UN RÉSERVOIR D’EAU PAR Q-LEARNING

Arthur Tornqvist et Quentin Velard

Introduction

SOMMAIRE

- Modélisation du problème de gestion optimale
  - Définition Etats/Actions
  - Définition de la fonction récompense
- Algorithme de Q-learning
  - Politique Politique E-greedy
  - Approximation de Bellman
  - Breakdown de l’algorithme
- Simulation d’un réservoir pour une politique epsilon-greedy
  - Optimisation des hyperparamètres
  - Avec ou sans decay sur les hyperparamètres

MODÉLISATION DU PROBLÈME

Le problème peut être modélisé par un 7-uplet : $\mathcal{M} = (S, A, P, R, \gamma, \alpha, \epsilon)$

- S : Ensemble des états (niveaux d'eau du réservoir) :

$$s(t+1) = s(t) + Q_{\text{in}}(t) - Q_{\text{out}}(t)$$

- A : Ensemble des actions (conserver ou rejeter l'eau):

$$Q_{\text{out}}(t) \in \{0, 1, 2\}$$

- P : Probabilités de transition entre les états (déterminées par les précipitations et les actions):

$$P(s'|s, a)$$

- R : Fonction de récompense (positive si le niveau d'eau est proche du niveau de référence, négative s'il est trop bas ou trop élevé)
- γ : **Facteur d'actualisation** (importance des récompenses futures)

$$\gamma \in [0, 1]$$

- **Le taux d'apprentissage α** qui contrôle la vitesse d'apprentissage

$$\alpha \in [0, 1]$$

- **Le taux d'exploration ε** qui gère le compromis exploration/exploitation

$$\epsilon \in [0, 1]$$

PROBABILITÉ DE TRANSITION

Action (A), pris par l’agent)

État actuel s(t) $\longrightarrow$ Etat futur s(t+1)

$$s_{t+1} = s_t + Q_{in}(t) - Q_{out}(t)$$

où :

- état futur (niveau de l'eau),
- Q_in(t), choisi aléatoirement dans notre modèle,
- Q_out(t), contrôlé par l'agent qui prend des décisions
- $Q_{in}(t), Q_{out}(t) \in [0, 1, 2]$

$$P(s_{t+1}|s_t, Q_{out}(t)) = P(s_t + Q_{in}(t) - Q_{out}t = s_{t+1})$$

FONCTION DE RÉCOMPENSE

$$R(s(t), Q_{\text{out}}(t)) = \begin{cases} +1 & \text{si } |s(t) - s_{\text{ref}}| \leq \delta \\ -1 & \text{si } s(t) \in [s_{\text{min}}, s_{\text{max}}] \text{ et } |s(t) - s_{\text{ref}}| > \delta \\ -10 & \text{si } s(t) = s_{\text{min}} \text{ ou } s(t) = s_{\text{max}} \end{cases}$$

- **Récompense positive (+1) :** Si le niveau d'eau est proche du niveau de référence hrefh_{\text{ref}}href.
- **Pénalité (-1) :** Si le niveau d'eau est trop bas ou trop élevé mais dans la plage acceptable.
- **Pénalité sévère (-10) :** Si le niveau d'eau sort de la plage acceptable (fuite ou débordement).

MODÉLISATION DU PROBLÈME

Le problème d’optimisation est le suivant, on veut maximiser :

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s = s_0, a = a_0\right]$$

On introduit la fonction Q telle que :

$$V^\pi(s) = \max_a Q^\pi(s, a)$$

La politique optimale π* dans ce cadre est alors simplement de choisir l'action qui maximise Q dans chaque état :

$$\pi^*(s) = argmax_a Q(s, a)$$

Q-LEARNING- POLITIQUE ϵ-GREEDY

La politique ε-greedy est une méthode qui favorise les actions optimales tout en permettant une exploration occasionnelle pour éviter les minima locaux.

$$\pi(s) = \begin{cases} \text{argmax}_a Q(s,a) & \text{avec probabilité } 1-\epsilon \quad \text{(exploitation)} \\ \text{action aléatoire} & \text{avec probabilité } \epsilon \quad \text{(exploration)} \end{cases}$$

Une stratégie souvent utilisée est de diminuer ε au fil du temps, par exemple $\epsilon_t = \frac{1}{t}$, afin de privilégier l'exploitation à long terme. C'est le decay

Si nous sélectionnons la méthode gourmande pure (epsilon = 0), nous choisissons toujours la valeur Q la plus élevée parmi toutes les valeurs Q pour un état spécifique. On peut se retrouver bloqué dans un minimum local

Q-LEARNING - MISE A JOUR DE L'ALGORITHME

L'objectif est d'apprendre une fonction Q qui approxime la valeur optimale d'une action dans un état donné.

$$Q \; (s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s = s_0, a = a_0 \right]$$

Le Q-Learning met à jour Q(s, a) en utilisant une approximation itérative :

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \middle| s = s_0, a = a_0 \right]$$

Un alpha plus élevé signifie que vous mettez à jour vos valeurs Q par grandes étapes. Lorsque l'agent est en phase d'apprentissage il convient de réduire cette valeur pour stabiliser la sortie du modèle, qui finit par converger vers une politique optimale.

Q-LEARNING - BREAKDOWN DE L'ALGORITHME

1. **Initialisation** :
   - On commence avec une table $Q(s, a)$ contenant des valeurs arbitraires.
   - Chaque état s a une liste d'actions a associées à des valeurs $Q(s, a)$.
2. **Choix de l'action** :
   - On utilise la politique **ε-greedy** pour choisir l'action a :
     - Avec probabilité ϵ, on choisit une action aléatoire.
     - Sinon, on choisit l'action qui maximise $Q(s, a)$.
3. **Exécution et mise à jour** :
   - L'agent exécute l'action, observe la récompense r et le nouvel état s'.
   - La mise à jour se fait avec la formule :

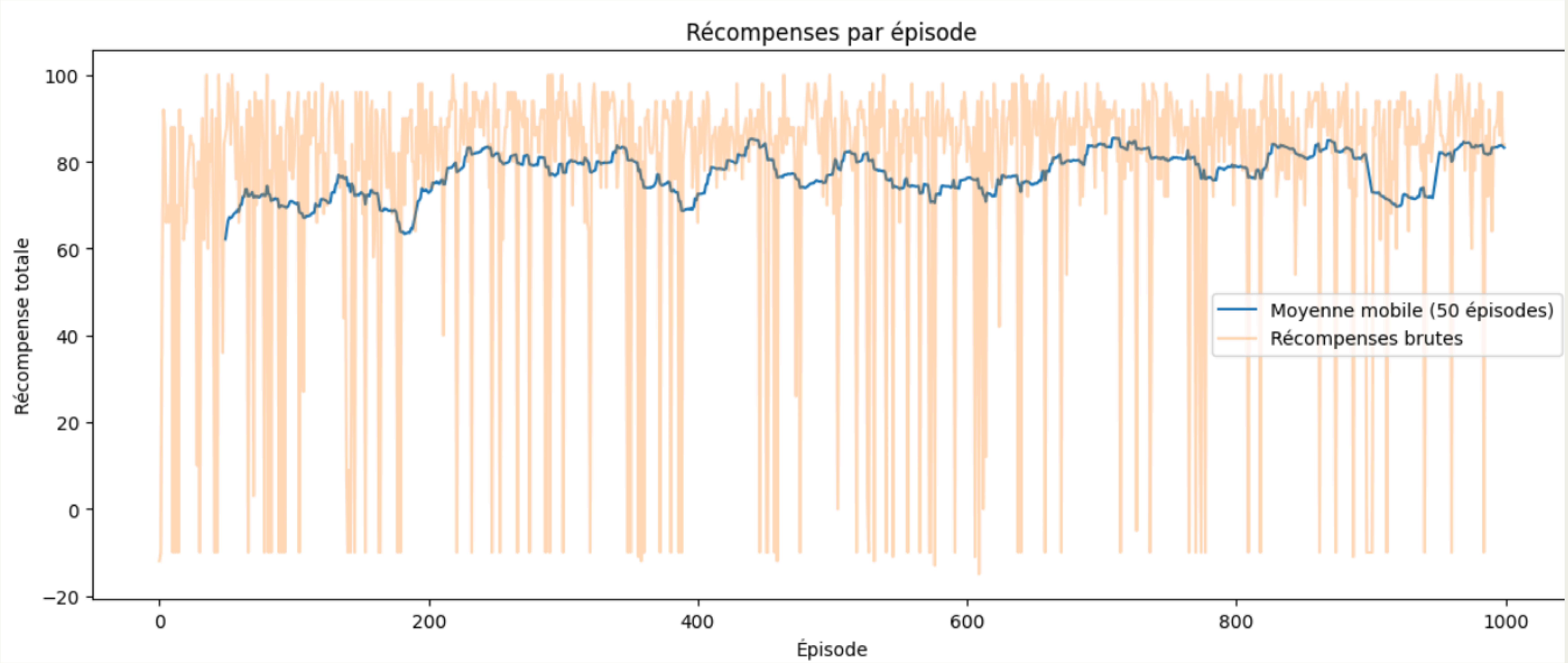
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right]$$

   où :

   - α : Taux d'apprentissage (learning rate, $0 < \alpha \leq 1$).
   - γ : Facteur de réduction des récompenses futures ($0 \leq \gamma \leq 1$).
   - $\max_{a'} Q(s', a')$: Meilleure valeur possible de l'état suivant.
4. **Transition à l'état suivant** :
   - L'état actuel devient s' et l'agent recommence.
5. **Fin d'épisode** :
   - L'agent atteint un état terminal, et l'épisode se termine.
   - L'apprentissage continue sur plusieurs épisodes jusqu'à convergence.

SIMULATION

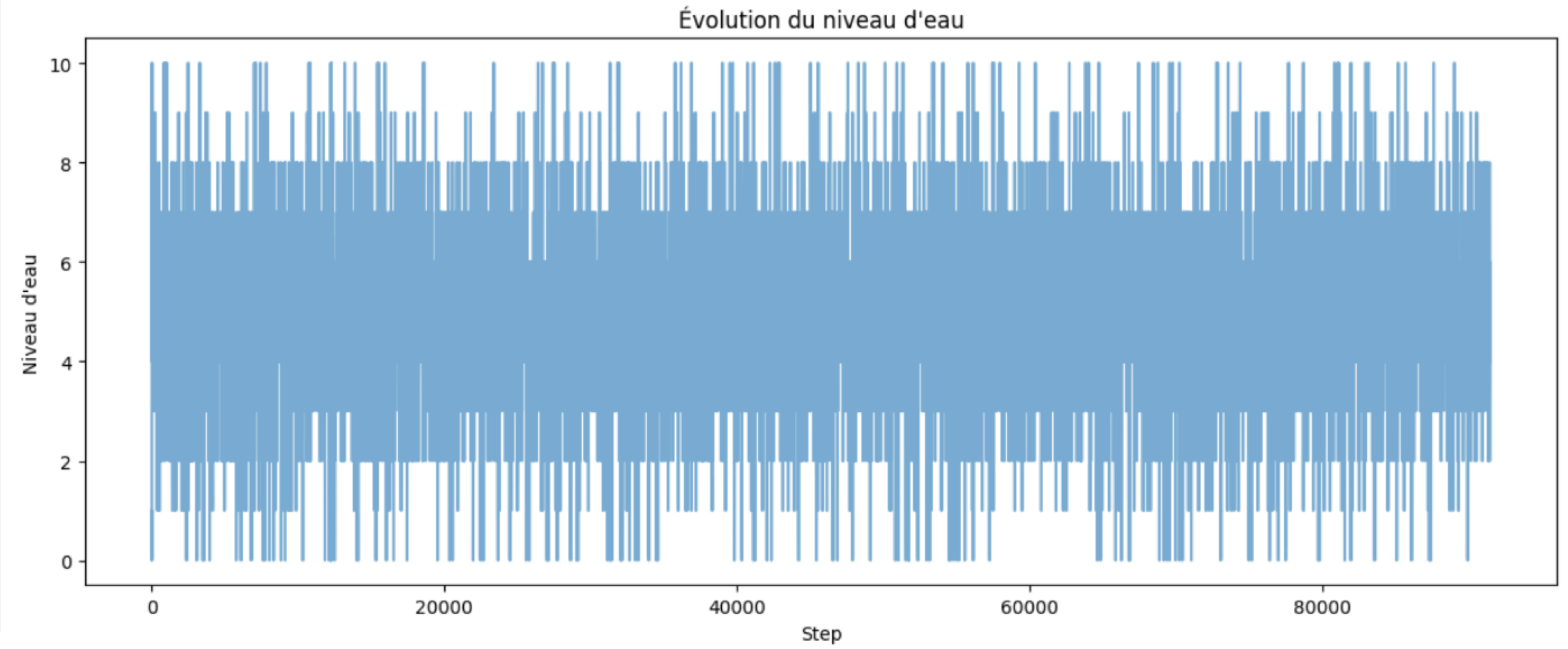
MOYENNE MOBILE ET DONNÉES BRUTES

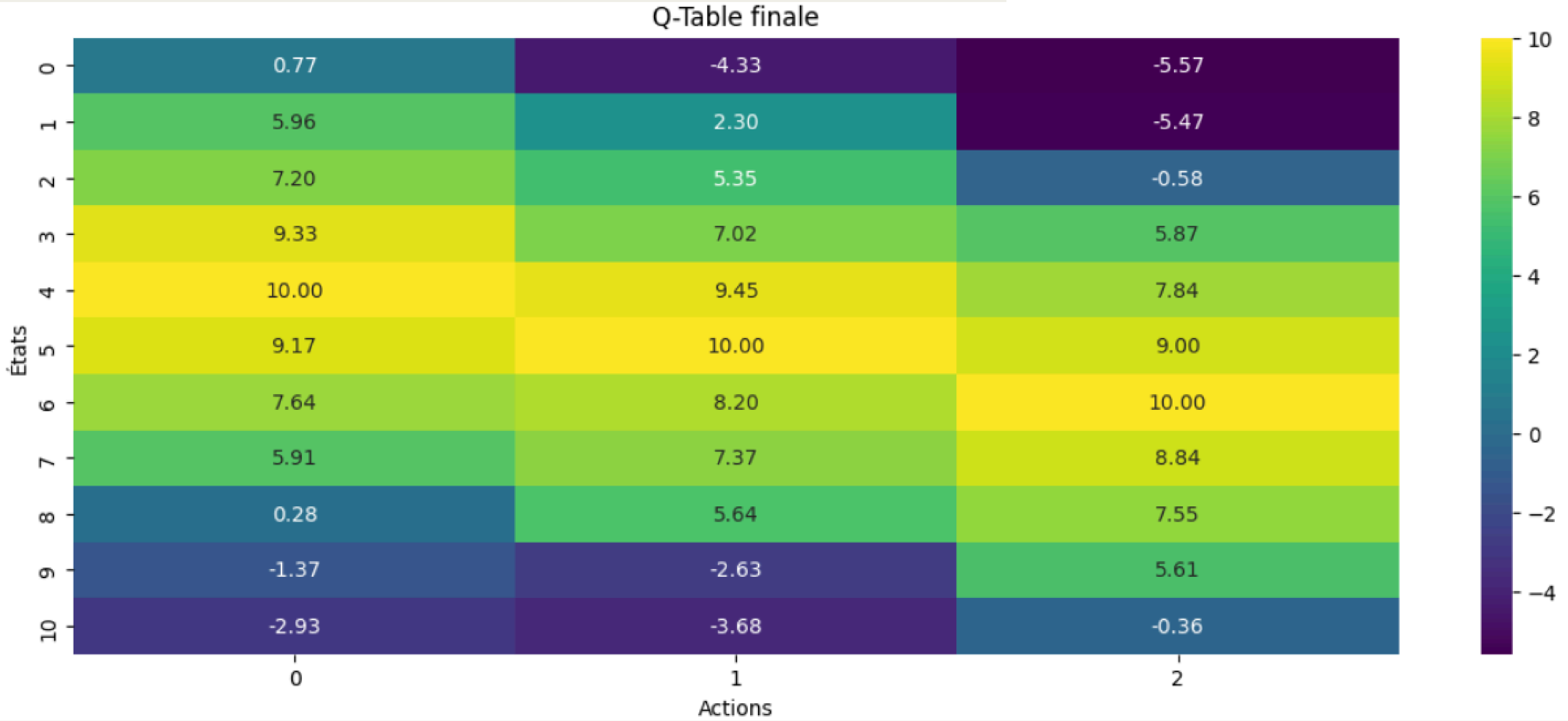


- Moyenne Mobile sur les 50 dernières valeurs, hyperparamètres choisis aléatoirement

SIMULATION- NIVEAU D'EAU ET TABLE Q FINALE

- Oscillation autour de la tendance

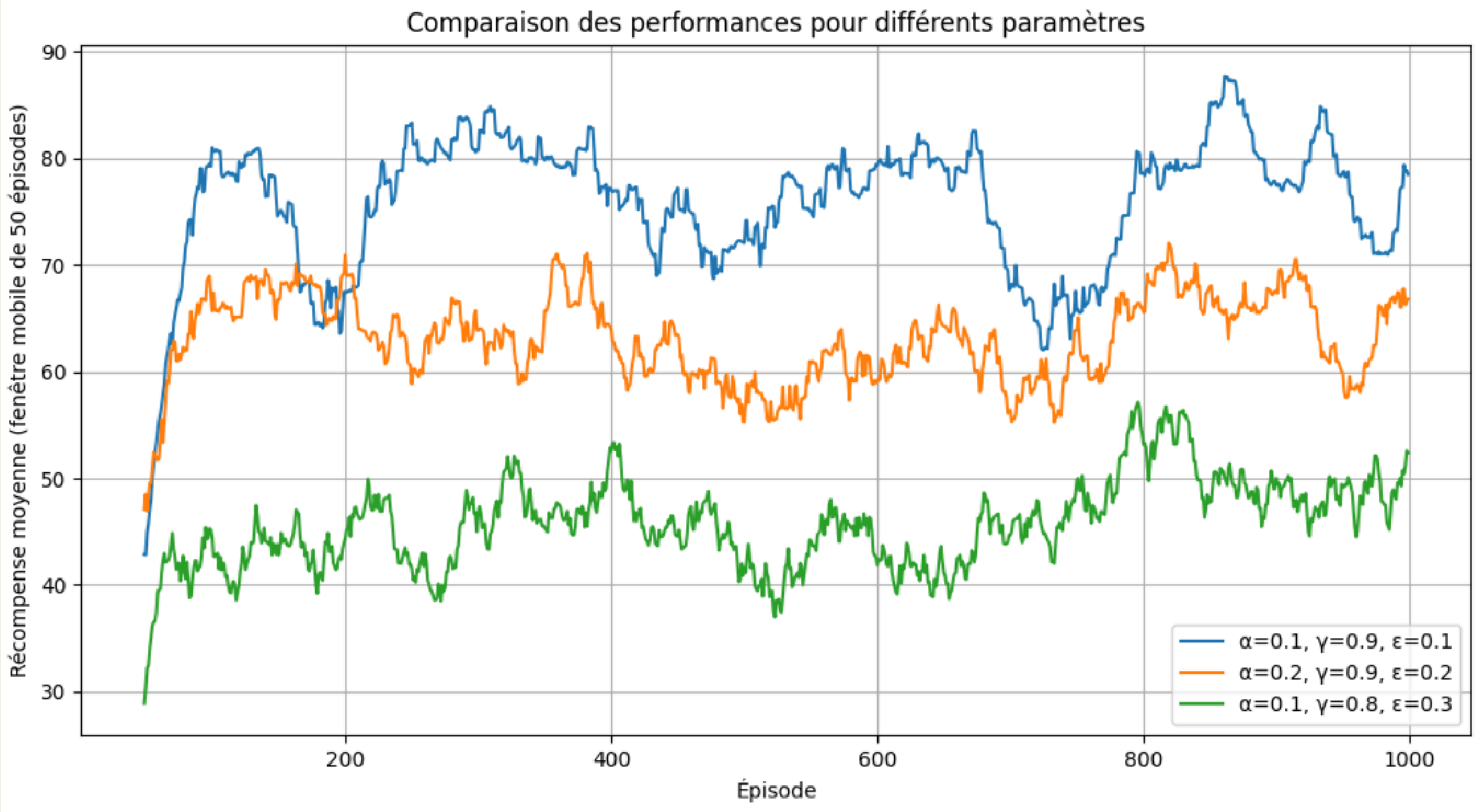




- Heatmap qui donne un paterne de matrice finale

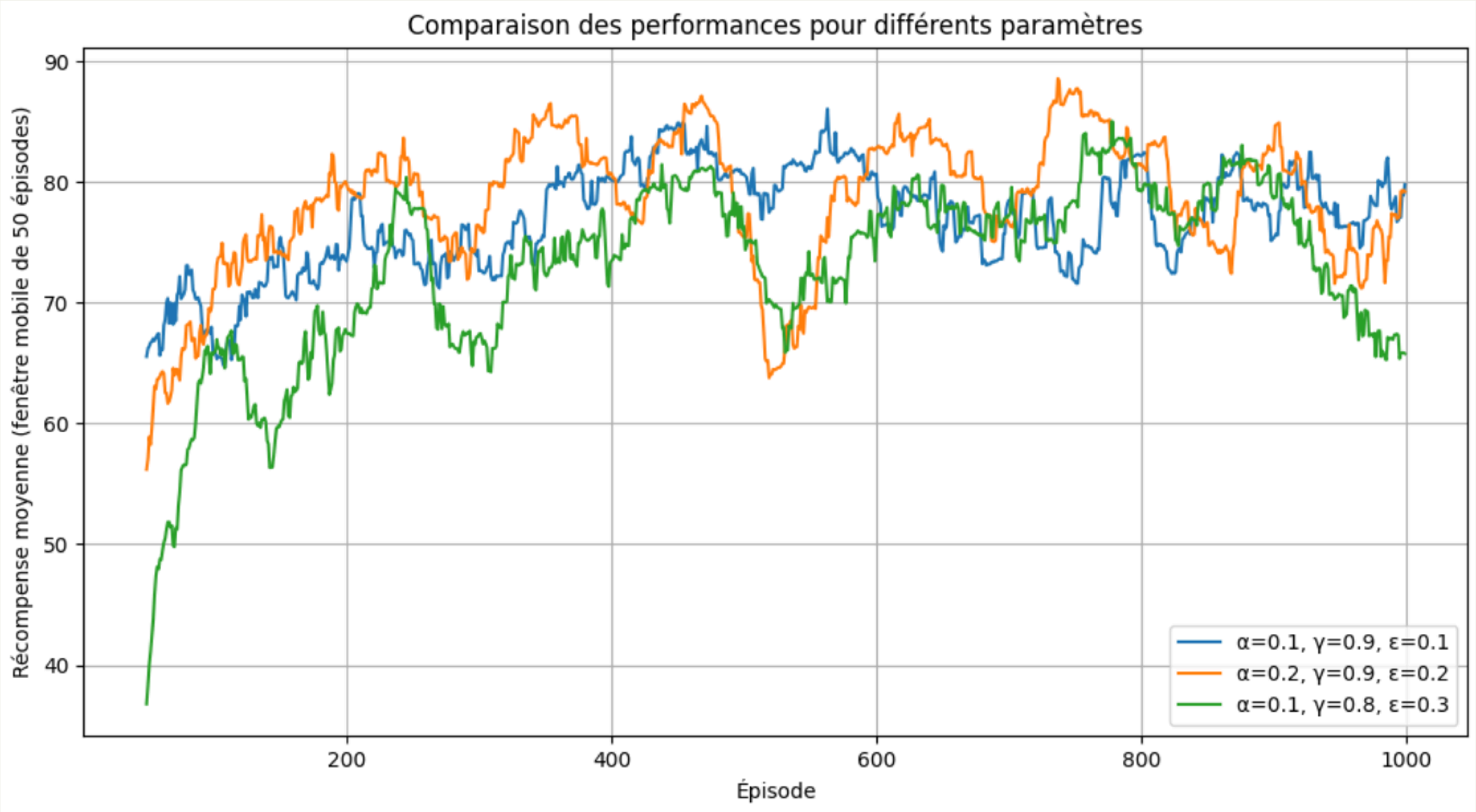
SIMULATION

MOYENNE MOBILE ET DONNÉES BRUTES



SIMULATION - ALPHA DECAY ET EPSILON DECAY

```
# Application de la décroissance de epsilon à la fin de chaque épisode
self.epsilon = max(0.1, self.epsilon * 0.995)  # Décroissance de epsilon ici
# Application de la décroissance de alpha à la fin de chaque épisode
self.alpha = max(0.01, self.alpha * 0.995)  # Décroissance de alpha
```

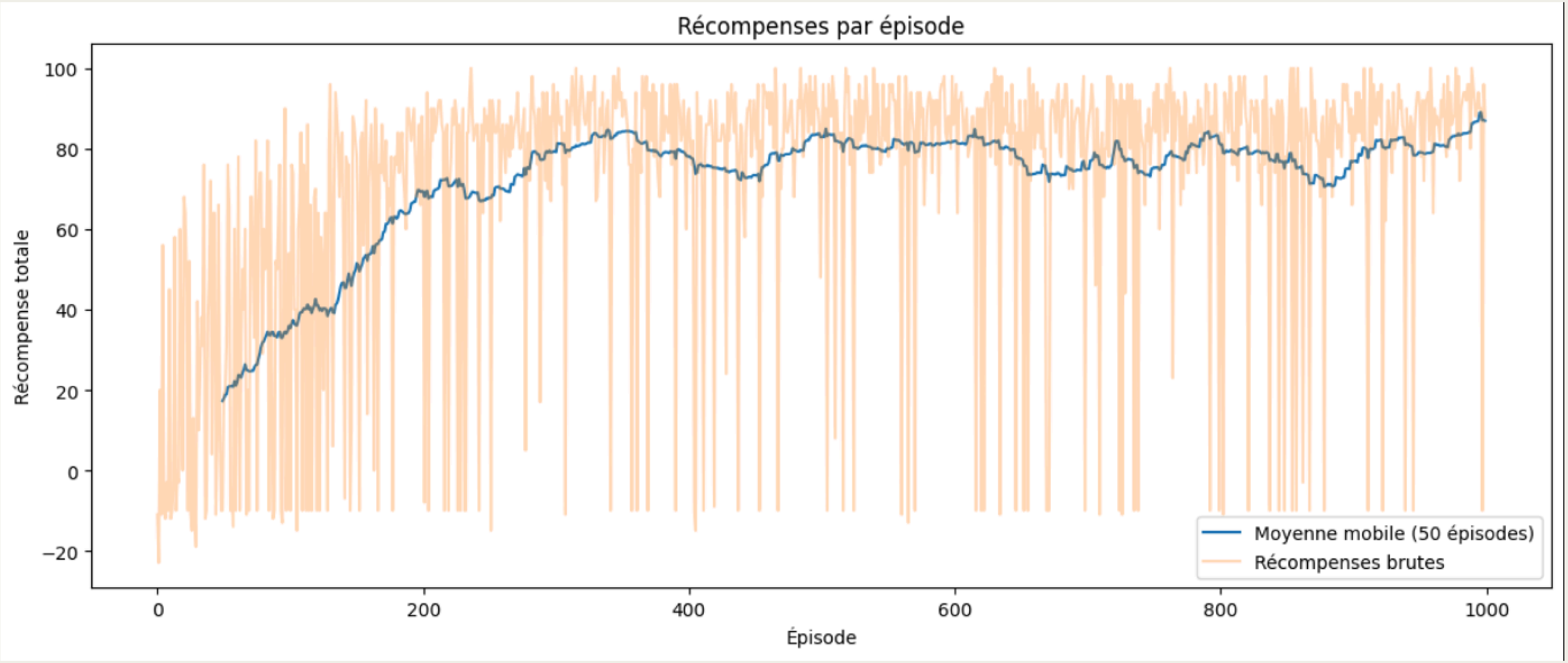


SIMULATION

OPTIMISATION DES HYPERPARAMÈTRES

L'objectif est d'optimiser les hyperparamètres pour maximiser la récompense moyenne sur **les 50 derniers épisodes**.

Meilleure combinaison d'hyperparamètres: {'alpha': 0.88, 'epsilon': 0.44, 'gamma': 0.86}

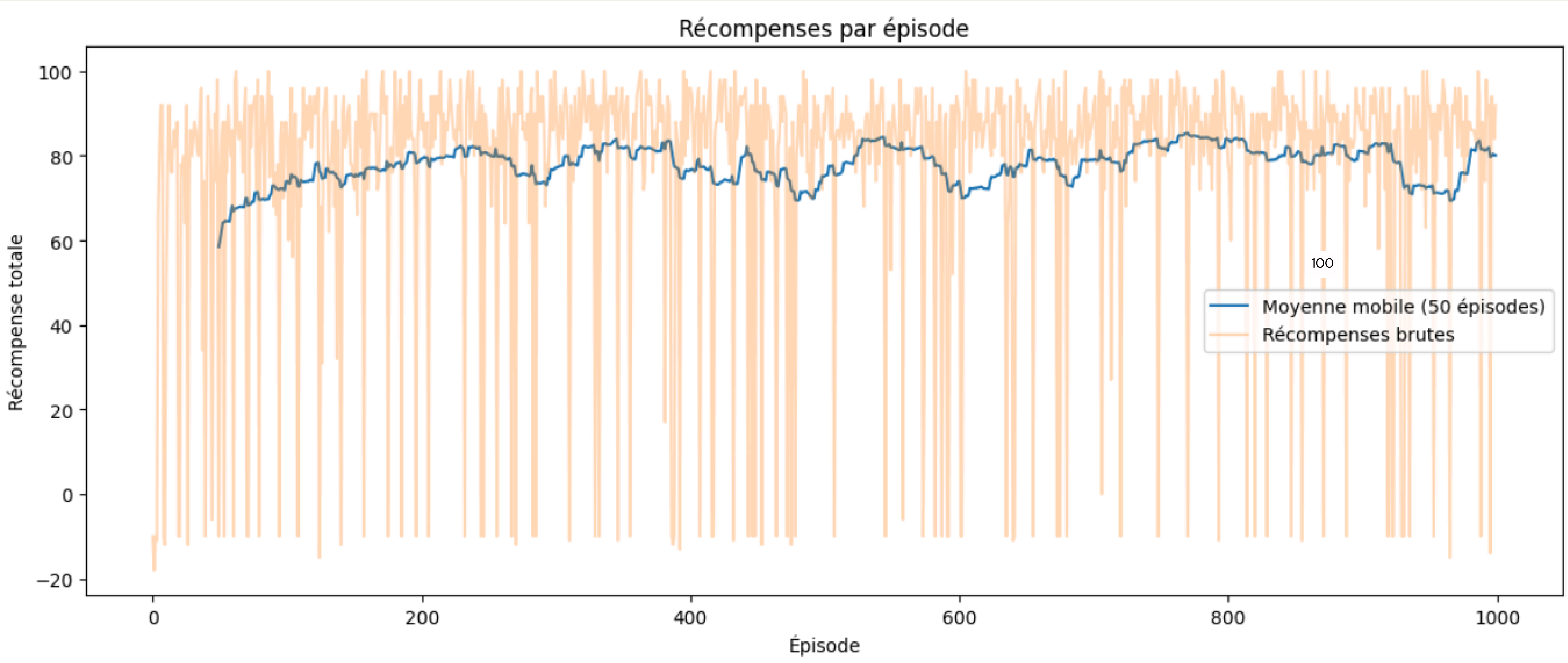


- On optimise le choix des hyperparamètres avec une méthode d’optimisation bayésienne
- La récompense moyenne sur les 50 derniers épisodes frôlent les 95

SIMULATION

OPTIMISATION DES HYPERPARAMÈTRES

- L'objectif maximiser la moyenne mobile des récompenses sur l'ensemble des épisodes d'entraînement.
- Meilleure combinaison d'hyperparamètres: {'alpha': 0.67, 'epsilon': 0.14, 'gamma': 0.94}



- On optimise le choix des hyperparamètres avec une méthode d'optimisation bayésienne
- La récompense moyenne est rapidement haute et stable, on a aussi minimisé la variance

SIMULATION

OPTIMISATION DES HYPERPARAMÈTRES ET DES DECAYS

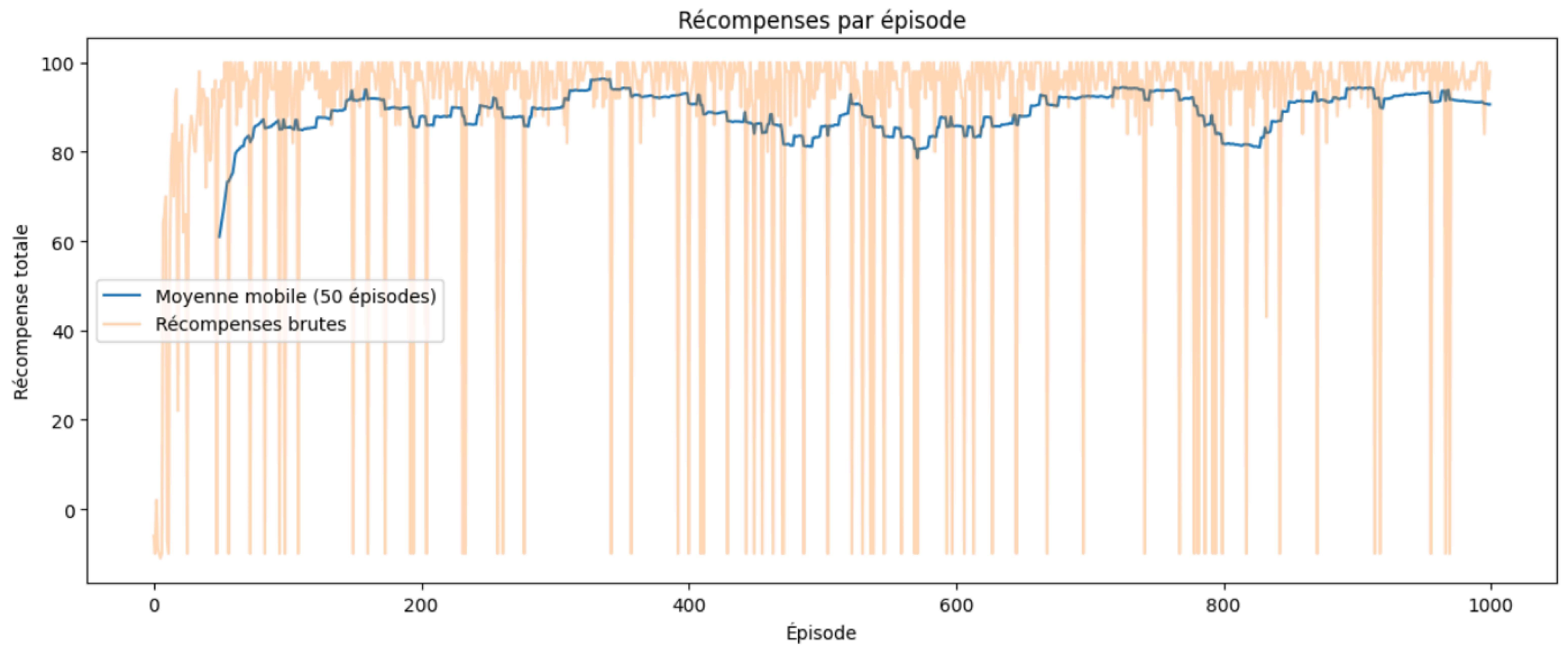
```
# Appliquer le decay après chaque épisode
self.epsilon = max(self.min_epsilon, self.epsilon * self.decay_epsilon)
self.alpha = max(self.min_alpha, self.alpha * self.decay_alpha)
```

- On optimise le choix des hyperparamètres avec une méthode d’optimisation bayésienne
- La récompense moyenne finale est haute et stable
- Meilleure combinaison d'hyperparamètres:{'alpha': 0.02, 'decay_alpha': 0.98, 'decay_epsilon': 0.92, 'epsilon': 0.16, 'gamma': 0.73}

SIMULATION

OPTIMISATION DES HYPERPARAMÈTRES ET DES DECAYS

```
# Appliquer le decay après chaque épisode
self.epsilon = max(self.min_epsilon, self.epsilon * self.decay_epsilon)
self.alpha = max(self.min_alpha, self.alpha * self.decay_alpha)
```



- On optimise le choix des hyperparamètres avec une méthode d’optimisation bayésienne
- La récompense moyenne est toujours haute et stable
- Meilleure combinaison d'hyperparamètres:{'alpha': 0.60, 'decay_alpha': 0.98, 'decay_epsilon': 0.95, 'epsilon': 0.49, 'gamma': 0.75}

CONCLUSION

- La **politique ε-greedy** équilibre exploration et exploitation.
- Le **Q-Learning est une méthode hors-politique** qui trouve la politique optimale en mettant à jour $Q(s, a)$.
- **La mise à jour suit une approximation de Bellman**, garantissant la convergence vers la solution optimale sous certaines conditions.
- **Alpha :** Affecte la vitesse d'apprentissage (comment les nouvelles informations influencent les décisions de l'agent).
- **Gamma :** Influence l'importance des récompenses futures (récompenses immédiates contre à long terme).
- **Epsilon :** Détermine l'exploration de l'agent (plus epsilon est élevé, plus l'agent explore, ce qui peut affecter l'efficacité de l'apprentissage).
- On a testé **différents critères de performance**(moyenne mobile sur les 50 derniers épisodes ou maximisation des moyennes mobiles) pour avoir **des hyperparamètres et des decays** (sur alpha et epsilon) optimisés.

ANNEXE - SIMULATION
DOUBLE Q-LEARNING

Le Double Q-Learning est une variante du Q-Learning qui vise à réduire l'optimisme biaisé dans l'estimation des valeurs d'action.

🛠️ Principe du Double Q-Learning

Dans Double Q-Learning, **deux tables de Q-valeurs** sont utilisées (Q_1 et Q_2) pour découpler la sélection et l'évaluation de l'action :

1. **Mise à jour de Q_1** :

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha \left[r + \gamma Q_2(s', \arg\max_{a'} Q_1(s',a')) - Q_1(s,a) \right]$$

2. **Mise à jour de Q_2** (symétriquement) :

$$Q_2(s,a) \leftarrow Q_2(s,a) + \alpha \left[r + \gamma Q_1(s', \arg\max_{a'} Q_2(s',a')) - Q_2(s,a) \right]$$

L'alternance entre les mises à jour de Q_1 et Q_2 permet d'obtenir une **meilleure stabilité et une estimation moins biaisée.**

ANNEXE - SIMULATION

DOUBLE Q-LEARNING