Université Paris Dauphine – PSL

Master IASD

Apprentissage par renforcement sur LunarLander-v3

Rapport de projet en Reinforcement Learning

Participants

Amine ROUIBI
Thomas SINAPI

Année universitaire 2025–2026

15 février 2026

Table des matières

Résumé

Ce projet étudie plusieurs approches d'*apprentissage par renforcement* pour contrôler l'environnement **LunarLander-v3**. L'objectif est d'évaluer, par l'implémentation et l'expérimentation, ce qui permet (ou empêche) l'apprentissage stable d'une politique d'atterrissage. Nous comparons une approche *value-based* (DQN) à deux approches *policy-based / actor-critic* (REINFORCE avec baseline et A2C avec *Generalized Advantage Estimation*). Le rapport se concentre sur (i) les éléments théoriques strictement nécessaires à la compréhension *de chaque méthode* et (ii) l'analyse expérimentale via les courbes d'apprentissage, les scores d'évaluation et la stabilité des entraînements.

1 Introduction

L'environnement **LunarLander-v3** est un problème de contrôle où l'agent pilote des propulseurs pour se poser sur une zone cible, en limitant la vitesse d'impact et l'utilisation de carburant. L'apprentissage est délicat : les épisodes sont longs, les retours sont très variables (crashs vs atterrissages réussis), et de petites erreurs de contrôle peuvent faire basculer l'issue d'un épisode. Dans ce contexte, un objectif pratique est d'atteindre un score moyen supérieur à 200 sur une fenêtre d'épisodes, seuil généralement associé à une résolution satisfaisante.

Le but de ce projet est de comparer, à implémentation et protocole de mesure constants, trois familles d'approches : DQN (value-based), REINFORCE avec baseline (policy gradient) et A2C+GAE (actor-critic). Au-delà du score final, on s'intéresse à la **stabilité** de l'apprentissage : vitesse de progression, variance des retours et sensibilité aux hyperparamètres.

Organisation du rapport. Nous présentons d'abord l'environnement et le protocole expérimental, puis chaque méthode (théorie minimale et choix d'implémentation), avant une comparaison des performances et une discussion des points de stabilité.

2 Environnement et protocole expérimental

2.1 Description de LunarLander-v3

L'état est un vecteur continu de dimension 8, comprenant position/vitesse, angle/vitesse angulaire, et deux indicateurs de contact des jambes au sol ; l'agent choisit parmi 4 actions discrètes (ne rien faire, moteur principal, moteur latéral gauche, moteur latéral droit).

La récompense est *shaped* : elle encourage le rapprochement vers la zone d'atterrissage, pénalise l'éloignement, et ajoute des bonus/malus liés au crash, à l'arrêt stable, au contact des jambes, ainsi qu'à l'utilisation des moteurs.Ce design rend l'apprentissage possible, mais introduit une forte variabilité intra-épisode (pénalités/bonus ponctuels, pénalité de carburant), ce qui complique les méthodes à gradient de politique.

2.2 Mesures rapportées

Nous reportons systématiquement :
(i) la récompense par épisode et sa moyenne mobile (typiquement fenêtre 50 ou 100),
(ii) une mesure de succès (par exemple % d'épisodes avec score ≥ 200 sur une fenêtre glissante),
(iii) des métriques de stabilité propres à certaines méthodes.

2.3 Protocole d'évaluation

Pour comparer les politiques issues des différents entraînements, nous effectuons une évaluation **déterministe** (sélection *greedy*/*argmax* de l'action) sur N épisodes. Nous reportons :

- le score moyen et l'écart-type sur N épisodes ;
- le taux de succès (% d'épisodes avec score ≥ 200).

Dans nos exécutions (logs disponibles dans le dépôt), l'évaluation est faite avec :

- $N = 30$ épisodes pour A2C (évaluations périodiques, sans rendu) ;
- $N = 5$ épisodes pour REINFORCE (script de test) ;
- pour DQN, nous suivons principalement la performance via la moyenne mobile (fenêtre 100) et un indicateur de succès (score ≥ 200) sur les derniers épisodes.

Le seuil de succès est fixé à **200** (score ≥ 200) comme dans la documentation de l'environnement et nos scripts.

2.4 Stratégie expérimentale : plusieurs paramétrages par méthode

Plutôt que de s'appuyer sur un seul entraînement par algorithme, nous avons mené des **séries d'essais** (*sweeps*) pour analyser la sensibilité aux hyperparamètres et la stabilité. L'objectif est de rapporter (i) le **meilleur checkpoint** par méthode, mais aussi (ii) **pourquoi** certains réglages convergent mieux.

Chaque run produit : (a) un checkpoint, (b) un log, (c) une figure récapitulative que nous stockons dans les dossiers relatifs à l'entrainement. Pour sélectionner un *best checkpoint*, nous utilisons les critères du protocole d'évaluation.

2.5 Point d'attention : `terminated` vs `truncated`

Les API récentes de *Gym*/*Gymnasium* distinguent : `terminated` (fin réelle MDP) et `truncated` (fin imposée : limite de temps) Cette distinction est critique pour les algorithmes qui **bootstrap** (valeur/critic), car le bootstrap est correct pour une troncature mais *pas* pour une terminaison vraie.

Dans la suite, nous explicitons, pour chaque méthode, comment sont traitées ces fins d'épisodes.

3 Approches évaluées

3.1 REINFORCE : policy gradient (on-policy)

Idée théorique clé

REINFORCE optimise directement une politique stochastique $\pi_\theta(a \mid s)$ en maximisant

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t \gamma^t r_t\right],$$

à l'aide de l'identité du gradient de politique :

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\right].$$

L'estimateur est non biaisé mais de variance élevée, particulièrement lorsque les épisodes sont longs et que le signal de récompense est bruité.

Pour réduire la variance, nous utilisons une **baseline** $b(s_t)$ (estimée par un critic $V_\phi(s_t)$) et un avantage :

$$A_t = G_t - V_\phi(s_t),$$

ce qui diminue la variance sans changer l'espérance du gradient.

Implémentation

La politique est modélisée par un MLP produisant des logits sur les 4 actions ; une baseline V_ϕ est apprise par régression sur les retours. Une régularisation d'entropie est ajoutée pour éviter une convergence prématurée vers une politique déterministe.

Paramétrages testés

Nous avons testé au moins :

- une version REINFORCE ;
- plusieurs tailles de réseau (dans notre dépôt : run final en hidden = 256).

Le run analysé ci-dessous correspond au dashboard `reinforce_maxEpisodes_10000_hiddenSize_256.png`.

FIGURE 1 – REINFORCE sur LunarLander-v3 : évolution du score, entropie, distribution des scores, taux de succès.

Analyse des courbes d'apprentissage. La Figure 1 illustre un apprentissage moins régulier : pendant une grande partie de l'entraînement, la moyenne mobile reste basse puis s'améliore tardivement. C'est cohérent avec une estimation de gradient de politique très bruitée, et un signal de récompense fortement dépendant de quelques épisodes réussis.

Deux points ressortent de cette courbe :

- **Amélioration tardive mais nette.** La moyenne mobile reste longtemps négative, puis progresse fortement sur la fin de l'entraînement, ce qui est typique d'un signal de gradient très bruité.
- **Variance élevée.** La distribution des retours reste large : même lorsque la moyenne augmente, on observe encore des épisodes très négatifs (crashs), signe d'une politique fragile et sensible à l'exploration résiduelle.

Malgré cette variance, l'évaluation déterministe du modèle sauvegardé (Tableau 1) atteint un score moyen élevé sur 5 épisodes. Cela suggère que, lorsqu'on exécute la politique sans bruit additionnel, elle peut être performante, mais que l'entraînement reste moins régulier qu'avec un actor-critic.

TABLE 1 – REINFORCE testé dans `reinforce_test_100ep.log`.

Variante	Checkpoint	Mean eval	Std	Succès (%)
REINFORCE	`reinforce_256.pt`	243.54	53.73	89.0

Résultats des variantes REINFORCE. En pratique, REINFORCE reste néanmoins plus coûteux en échantillons et plus difficile à stabiliser : (i) la variance du gradient demeure élevée, (ii) l'algorithme est *on-policy* : chaque amélioration repose sur des trajectoires fraîches, ce qui rend l'apprentissage plus lent qu'une méthode avec bootstrap. Ces constats motivent l'utilisation d'un actor-critic (A2C/GAE), qui exploite un critique $V(s)$ et un estimateur d'avantage moins bruité.

3.2 A2C avec *GAE* : actor-critic (on-policy, bootstrap)

Idée théorique clé

Les méthodes *actor-critic* apprennent simultanément :

— un **acteur** $\pi_\theta(a \mid s)$, optimisé par gradient de politique ;

— un **critique** $V_\phi(s)$, entraîné pour approximer la valeur et réduire la variance du gradient.

L'acteur est mis à jour avec un avantage $\hat{A}_t$ au lieu du retour brut G_t :

$$\mathcal{L}_{\text{policy}}(\theta) = -\mathbb{E}\left[\log \pi_\theta(a_t \mid s_t)\, \hat{A}_t\right].$$

Pour estimer $\hat{A}_t$ de façon plus stable, on utilise **Generalized Advantage Estimation (GAE)** :

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l \geq 0} (\gamma\lambda)^l\, \delta_{t+l}, \qquad \delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t),$$

où λ contrôle le compromis biais/variance.

Implémentation

Dans notre implémentation, l'entraînement alterne (i) collecte de rollouts, (ii) calcul de GAE, (iii) mise à jour conjointe acteur/critique. Nous utilisons également :

— **annealing d'entropie** (coefficient diminué au cours du temps) pour passer d'une exploration forte à une politique plus déterministe ;

— **gradient clipping** pour limiter les explosions de gradient ;

— **gestion correcte de** `terminated`/`truncated` afin de booter correctement lorsque l'épisode est tronqué.

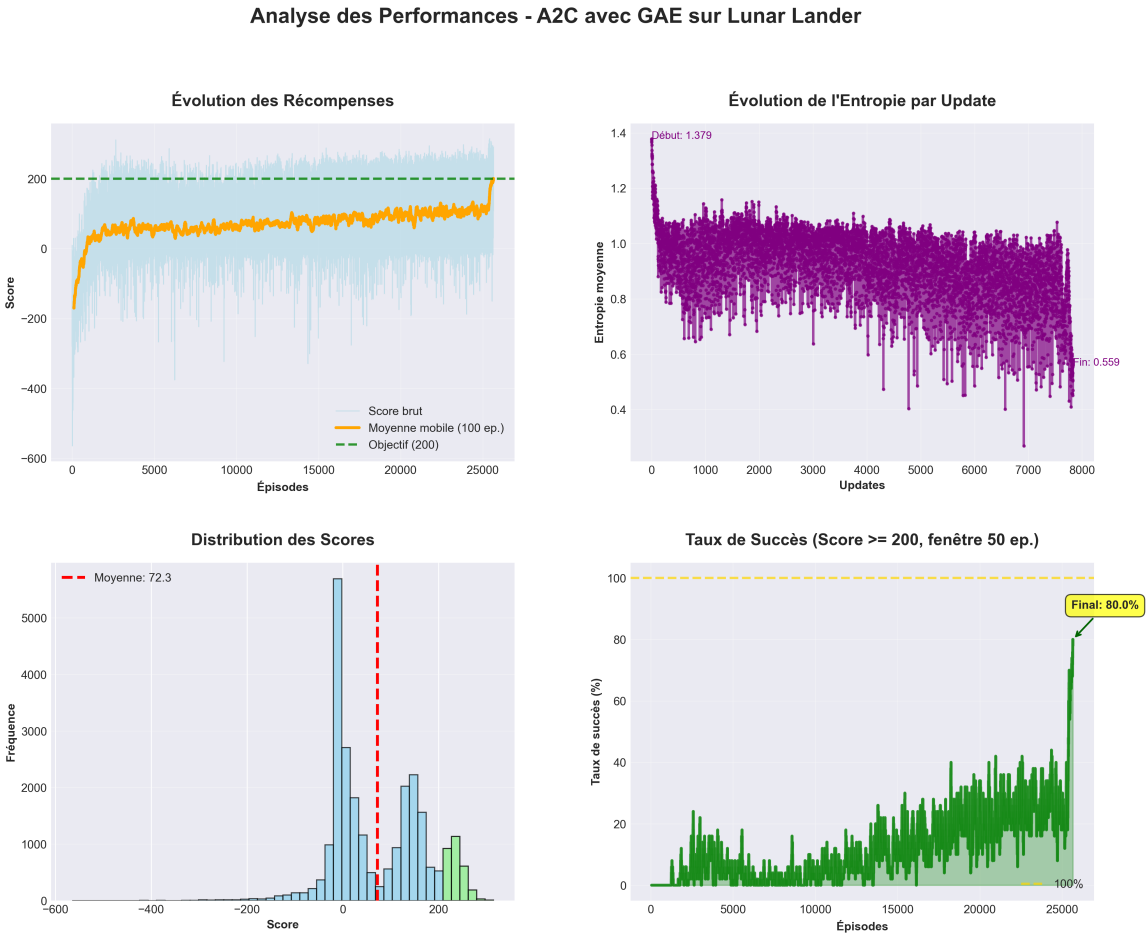

FIGURE 2 – A2C (hidden=384) : progression des scores, entropie, distribution et taux de succès.

Analyse des courbes d'apprentissage. La Figure 2 met en évidence une progression plus régulière que REINFORCE. L'utilisation d'un critique $V(s)$ et de GAE réduit la variance de l'estimateur d'avantage, ce qui se traduit par des améliorations plus continues et une politique finale globalement plus stable.

Paramétrages testés

Nous avons testé plusieurs variantes d'A2C :

— différentes tailles de réseau (hidden = 256 / 384 / 512) ;
— différentes stratégies d'entropie (constante vs annealing) ;

La variante retenue comme meilleure dans nos essais est la version **A2C (hidden=384)**, qui obtient le meilleur `Best eval reward` dans les logs (Tableau 2).

TABLE 2 – Comparaison de quelques variantes A2C.

Variante	Checkpoint	Mean eval	Std	Succès (%)
A2C (hidden=256)	`a2c_256.pt`	200.84	91	71
A2C (hidden=384)	`a2c_384.pt`		200.2 ± 98.6	

Résultats agrégés A2C (à compléter).

Ce qui marche / ce qui marche moins (analyse)

Les courbes (Figure 2) montrent une dynamique d'apprentissage plus robuste que REINFORCE : le score moyen augmente régulièrement, l'entropie décroît (politique moins aléatoire), et le taux de succès progresse jusqu'à des valeurs élevées. En pratique, deux facteurs ont eu un impact majeur sur la réussite :

- **Bootstrap correct en fin d'épisode** : traiter toutes les fins comme terminales (c.-à-d. $V(s_{t+1}) = 0$) en cas de troncature casse l'estimation de valeur, ce qui dégrade fortement l'update acteur/critique ; la correction `terminated` vs `truncated` rétablit un signal d'avantage cohérent.
- **Compromis exploration/convergence** : l'annealing d'entropie évite de rester bloqué dans une exploration permanente tout en limitant la convergence trop rapide vers une politique fragile.

Cette méthode est toutefois sensible aux hyperparamètres (taux d'apprentissage policy/value, coefficients de perte, λ de GAE) : des choix agressifs peuvent entraîner des oscillations de performance et nécessitent typiquement un mécanisme de sauvegarde du *best checkpoint*.

3.3 DQN : approximation de valeur (off-policy)

Idée théorique clé

DQN approxime une fonction $Q_\theta(s, a)$ avec un réseau de neurones, puis apprend en minimisant une erreur de type *Bellman* :

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a'), \qquad \mathcal{L}(\theta) = \mathbb{E}\left[(Q_\theta(s, a) - y)^2\right].$$

L'utilisation d'un *target network* Q_{θ^-} (paramètres figés puis recopiés périodiquement) réduit le problème de *cible mouvante* et stabilise l'entraînement.

Mécanismes de stabilisation utilisés

Deux mécanismes sont déterminants en pratique :

- **Experience Replay** : les transitions (s, a, r, s') sont stockées dans un buffer et rééchantillonnées aléatoirement afin de casser les corrélations temporelles, d'améliorer l'efficacité échantillonnale et de rendre la descente de gradient plus stable.
- **Target Network** : la cible y est calculée avec un réseau retardé, mis à jour périodiquement.

Implémentation (résumé)

Le réseau Q est un MLP simple ($8 \to 128 \to 128 \to 4$) ; l'exploration suit une stratégie ϵ-greedy avec décroissance.

Paramétrages testés

Nous avons testé plusieurs réglages en ne modifiant qu'un hyperparamètre à la fois (sweep) :

- baseline ;
- décroissance de ϵ plus lente ;
- replay buffer plus grand ;
- mise à jour du target network plus fréquente ;
- learning rate plus faible.

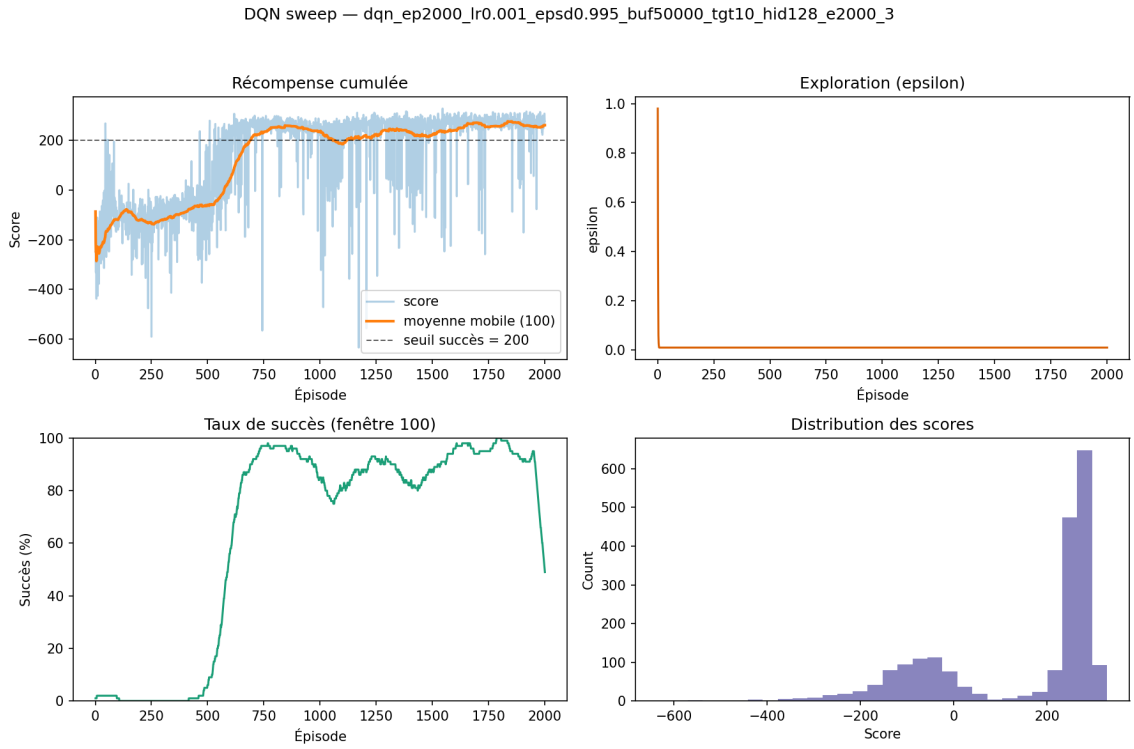

FIGURE 3 – DQN sur LunarLander-v3 : score par épisode, moyenne mobile, décroissance de ϵ, taux de succès et distribution des scores (dashboard).

Courbes d'apprentissage DQN La Figure 3 montre un apprentissage rapide (la moyenne mobile progresse fortement), mais une variance qui reste visible : même après convergence apparente, certains épisodes restent très négatifs. Cette instabilité s'explique par la stratégie d'exploration ϵ-greedy et par la nature off-policy de DQN. Même après convergence apparente, certaines actions exploratoires peuvent provoquer des échecs sévères.

Résultats agrégés du sweep DQN. Les résultats ci-dessous proviennent d'une **évaluation déterministe sur 100 épisodes** pour chaque checkpoint (moyenne, écart-type et taux de succès : score ≥ 200). Les fichiers de log sont disponibles dans `2.DQN/logs/`.

TABLE 3 – Comparaison des variantes DQN (2000 épisodes) : évaluation déterministe sur 100 épisodes.

Variante	Checkpoint / log	Mean eval (100 ep)	Succès (%)
Baseline (buf=10k, ϵ-decay=0.995, tgt=10)	`dqn_..._1.pth`	-431.0 ± 291.9	10
ϵ-decay plus lent (0.997)	`dqn_..._2.pth`	132.0 ± 253.1	63
Replay buffer plus grand (50k)	`dqn_..._3.pth`	268.3 ± 31.7	95
Target update plus fréquent (tgt=5)	`dqn_..._4.pth`	257.0 ± 56.4	90
Learning rate plus faible (lr=$5 \cdot 10^{-4}$)	`dqn_..._5.pth`	97.2 ± 105.7	26
Exploration "visible" (eps min=0.05)	`dqn_explo.pth`	261.7 ± 66.5	94

Analyse des résultats DQN. Le Tableau 3 met en évidence trois conclusions simples :

— **Le replay buffer plus grand est le meilleur choix.** Le run buf=50k (#3) donne la meilleure robustesse : moyenne élevée (268.3) et variance faible, avec **95%** de succès.

— **Le target update plus fréquent (tgt=5) est une bonne alternative.** Il reste très performant (90% de succès), mais avec plus de variance (donc plus d'épisodes ratés).
— **Certains réglages peuvent échouer complètement.** La baseline (#1) est négative en moyenne : cela illustre la sensibilité de DQN aux données disponibles dans le replay et à la stabilité de la cible TD.

Le run *exploration visible* obtient aussi d'excellents résultats (94% de succès), mais reste un peu plus variable que le meilleur sweep. Pour visualiser explicitement l'évolution de ϵ sur ce run (et donc la différence avec les runs où ϵ tombe très vite à zéro), nous reportons son dashboard complet en Annexe A.1.

4 Analyse comparative

Comparaison des meilleurs modèles

Pour rester lisible, on compare uniquement le **meilleur modèle** obtenu pour chaque méthode :

Table 4 – Comparaison des meilleurs modèles par méthode (évaluation déterministe).

Méthode	Variante retenue	Mean eval	Succès (%)	Référence
DQN	replay buffer 50k	268.3 ± 31.7 (100 ep)	95	Tab. 3
REINFORCE	hidden=256	243.5 ± 53.7	89	Tab. 1
A2C+GAE	hidden=512)	–	–	Tab. 2

Ce qui fonctionne pour chaque méthode (et pourquoi).
— **DQN** : fonctionne le mieux quand la *diversité des transitions* est suffisante (grand replay buffer) et que la cible TD est stabilisée (target network). On obtient une politique très robuste en fin d'entraînement.
— **REINFORCE** : peut atteindre un bon score, mais il reste plus variable car l'update vient de retours Monte-Carlo bruités. La baseline et l'entropie aident, sans supprimer complètement la variance.
— **A2C + GAE** : entraîne plus régulièrement grâce au critic et à GAE (avantage moins bruité). En échange, il faut régler finement le couple actor/critic (lr, entropie, λ) et gérer correctement `terminated`/`truncated`.

Pourquoi DQN est souvent meilleur à la fin ? Dans ce projet, DQN finit souvent avec le meilleur score moyen car il est **off-policy** : les mêmes transitions sont réutilisées de nombreuses fois via le replay buffer. Cette réutilisation améliore l'efficacité en données et consolide une bonne politique une fois que l'exploration est maîtrisée.

5 Conclusion

Sur LunarLander-v3, nos essais confirment que la performance ne dépend pas seulement de l'algorithme, mais aussi de la stabilité de l'entraînement. DQN atteint les meilleures moyennes mobiles avec un replay buffer plus grand (run buffer=50k), mais reste sensible à l'exploration résiduelle. REINFORCE avec baseline peut produire un modèle performant en test déterministe, au prix d'une trajectoire d'entraînement plus irrégulière. Enfin, A2C+GAE offre le compromis le plus robuste dans nos runs : les métriques d'évaluation et la moyenne glissante dépassent le seuil de résolution, à condition de traiter correctement les fins d'épisodes (`terminated`/`truncated`) et de contrôler l'exploration via l'entropie.

Références

[1] Gym Documentation (gymlibrary). *Lunar Lander (Box2D) : reward shaping et seuil "Solved"*. Documentation technique, consultée en 2026.

A Annexes

A.1 DQN : run "exploration visible"

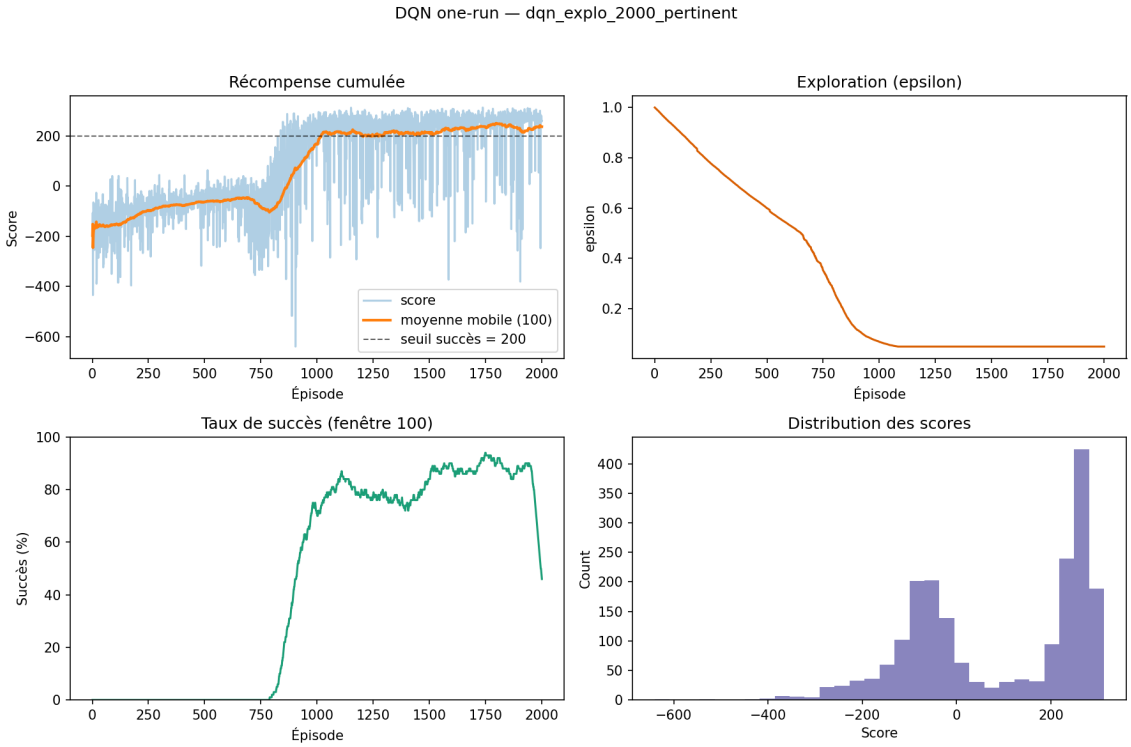

FIGURE 4 – Dashboard du run DQN "exploration visible" (progression des scores, moyenne mobile, ϵ, taux de succès, distribution des retours).