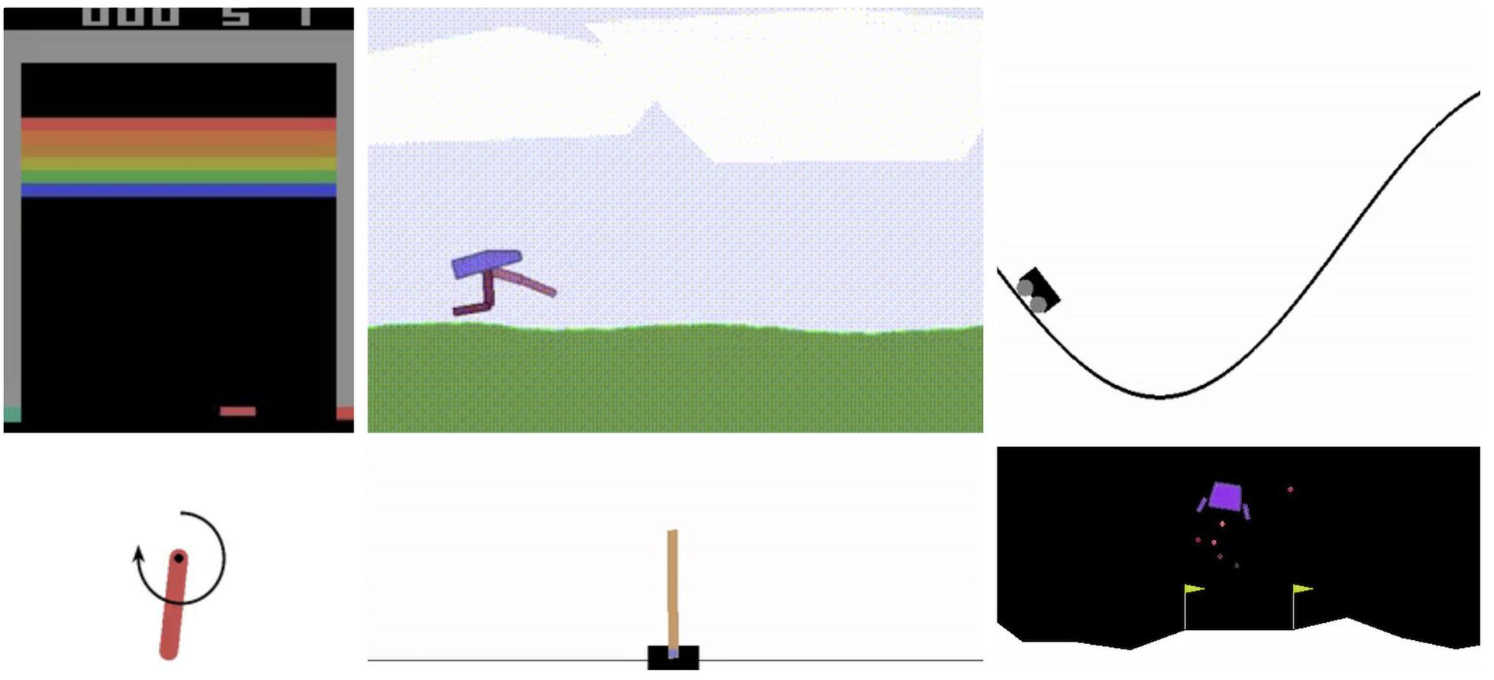
RAPPORT DU PROJET D’APPRENTISSAGE PAR RENFORCEMENT



**LHADJ Yani**

**AHAMDY Mouna**

2025

M2 IASD en Apprentissage

# 

# **Sommaire**

[**Introduction**](#_ohovlxipxo5g) **3**

[**Implémentation des algorithmes DQN**](#_307zp5kqamfq) **4**

[**Analyse des résultats du DQN**](#_2n1a6smpdq0z) **6**

[**Conclusion: Performance finale et choix de la meilleure approche**](#_lbgzc7741jbe) **9**

[**Annexe**](#_6e3ajlo54rgq) **10**

[1. Paramètres d'Entraînement](#_r313ykbbeajg) 10

[2. Figures et animations](#_s0wf82cl31bl) 11

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

# 

# **Introduction**

Dans le cadre de ce projet, nous avons mis en œuvre des algorithmes d'apprentissage par renforcement profond pour résoudre divers problèmes proposés par *Gymnasium*. Initialement, notre choix s'est porté sur un environnement complexe, tel que le jeu de ping-pong (*Pong-v5*). Ce dernier est un environnement multidimensionnel avec des observations de dimensions (210, 160, 3) et un espace d'actions discret composé de six mouvements. Cependant, ses spécificités ont entraîné une consommation excessive de ressources matérielles, nous obligeant à nous tourner vers des environnements plus simples et analytiquement accessibles. Ce rapport présente notre démarche, les solutions adoptées et les résultats obtenus.

Il est important de noter que face à cette limitation matérielle pour le jeu de ping-pong, nous avons envisagé de segmenter les épisodes en sauvegardant périodiquement le modèle afin de poursuivre l'apprentissage ultérieurement. Cependant, cette approche ne permettait pas de vérifier efficacement si l’agent apprenait réellement, en raison d’une progression trop lente de sa performance.

Nous avons donc opté pour un environnement unidimensionnel, proposé dans la famille *Classical Control*. Ces environnements unidimensionnels présentent l’avantage d’être plus facilement évaluables, car l’atteinte de l’objectif peut être directement observée.

Nous avons fait le choix de tester nos algorithmes sur l’environnement *MountainCar-v0*, un environnement où une voiture doit accumuler de l’énergie cinétique pour atteindre le sommet d'une colline. Cet environnement nous est familier car évoqué dans le cours.

Nous avons opté pour l’algorithme du DQN afin de répondre aux spécificités de l’environnement étudié. En effet, le *Deep Q-Network* (DQN) a été choisi car il s'agit d'une méthode basée sur les valeurs (*value-based*), particulièrement efficace pour des espaces d’états discrets tels que ceux rencontrés dans *MountainCar-v0*. L’objectif serait ici de comparer quatre différentes variantes de DQN sur un même problème afin de comparer leur efficacité.

Nous avons jugé intéressant d’implémenter ces algorithmes, car nous n’en avons pas eu l’occasion dans le cadre du cours. Lors des séances de travaux pratiques, nous avons uniquement exploré les méthodes *Dynamic Programming* (DP), *Monte Carlo* (MC) et *Temporal Difference* (TD). Ce projet nous a donc offert l'opportunité d'explorer de nouvelles approches et de découvrir différentes variantes de *DQN*, telles que *DQN avec Experience Replay*, *Prioritized Experience Replay* et *Double DQN*, afin d’analyser leur impact sur l’apprentissage et la performance des agents.

# **Implémentation des algorithmes DQN**

Notre implémentation des algorithmes DQN est conçue pour fonctionner avec n’importe quel jeu Gymnasium dont l’espace d’état est de dimension 1, ce qui la rend adaptable à divers environnements comme *MountainCar*, et *LunarLander* entre autres. L’architecture du code est modulaire, ce qui facilite la compréhension et la modification des différentes composantes.

L’implémentation repose sur plusieurs fichiers et classes distincts. La gestion de la mémoire d’expérience est assurée par ExperienceReplay pour la version standard et PrioritizedExperienceReplay pour la version avec priorités. Ces classes stockent les expériences sous la forme *(état, action, récompense, nouvel état, indicateur de fin)* et permettent d’échantillonner des mini-lots pour entraîner le réseau de neurones.

Le réseau de neurones utilisé est défini dans la classe qNetwork, qui s’adapte à la nature de l’entrée. Si l’état est un vecteur, un réseau entièrement connecté est utilisé, tandis que si l’entrée est une image, un réseau convolutionnel est appliqué pour extraire les caractéristiques avant de passer aux couches denses. Le modèle implémente aussi des méthodes de sauvegarde et de chargement des poids pour reprendre l’apprentissage à tout moment.

Les hyperparamètres du DQN sont définis de manière flexible pour différents environnements dans un fichier de configuration, où des paramètres comme la taille de la mémoire d’expérience, le taux d’apprentissage, le facteur de discount γ et la stratégie d’exploration ε-greedy sont spécifiés pour chaque jeu. L’exploration est gérée par un facteur ε qui décroît progressivement au fil des épisodes, équilibrant ainsi exploration et exploitation.

La classe principale qui permet d’exécuter les différentes versions de DQN ainsi que le “main” est définie dans le fichier “OneDimensionEnvironment.py”. Ce fichier sert de point d’entrée principal et permet de lancer l’entraînement et l’évaluation des agents sur des environnements de dimension 1, en sélectionnant l’algorithme de DQN souhaité et en paramétrant les options associées. Par ailleurs, le fichier “ThreeDimensionEnvironment.py” propose une implémentation adaptée aux environnements tridimensionnels, permettant d’entraîner et d’exécuter les algorithmes sur des jeux plus complexes comme Pong ou d’autres environnements 3D compatibles avec Gymnasium.

Nous allons maintenant comparer les résultats obtenus en exécutant plusieurs variantes du DQN sur *MountainCar* pendant *1000 épisodes*, en utilisant les mêmes hyperparamètres pour garantir une comparaison équitable. Les variantes testées sont :

* DQN standard avec Experience Replay.
* DQN avec Prioritized Experience Replay (PER).
* Double DQN avec Experience Replay.
* Double DQN avec Prioritized Experience Replay (Double DQN + PER).

Les hyperparamètres utilisés sont les suivants :

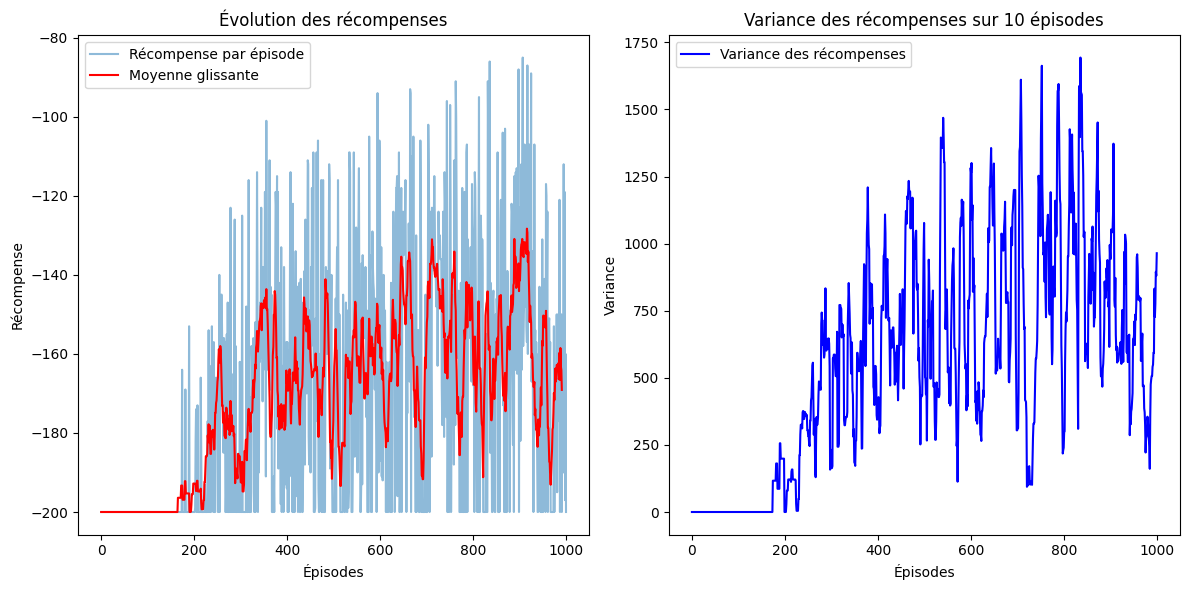
* *replay\_memory\_size* = 10 000 (taille de la mémoire d’expérience)
* *mini\_batch\_size* = 128 (taille des mini-lots pour l’entraînement)
* *epsilon\_init* = 1 (valeur initiale d’exploration)
* *epsilon\_decay* = 0.995 (facteur de décroissance d’ε)
* *epsilon\_min* = 0.01 (valeur minimale d’ε)
* *gamma* = 0.99 (facteur de discount)
* *learning\_rate* = 0.004 (taux d’apprentissage)

Ces hyperparamètres ont été déterminés après avoir testé de nombreuses configurations différentes afin d’optimiser l’apprentissage de l’agent. Plusieurs essais ont été réalisés en ajustant les valeurs du taux d’apprentissage, du facteur de discount et de la stratégie d’exploration (epsilon), mais aussi en modifiant la taille de la mémoire d’expérience et des mini-lots. L’un des paramètres ayant eu le plus d’impact sur l’apprentissage est le facteur de décroissance d’epsilon (epsilon\_decay). En effet, lorsque epsilon\_decay est fixé à 0.9, l’exploration décroît trop rapidement, ce qui empêche l’agent d’explorer suffisamment l’environnement et peut mener à un apprentissage sous-optimal. À l’inverse, lorsque epsilon\_decay est de 0.999, l’agent reste en phase d’exploration trop longtemps, ce qui ralentit considérablement la convergence. Le choix de la valeur 0.995 représente ainsi un bon compromis entre exploration et exploitation.

Bien que ces paramètres soient loin d’être optimaux, l’objectif est de comparer la performance de chaque approche sur le même environnement et durant la même période d’entraînement. Cela permettra d’évaluer l’impact de l’Experience Replay, du Prioritized Experience Replay et de l’algorithme *Double DQN* sur la convergence du modèle et la stabilité de l’apprentissage.

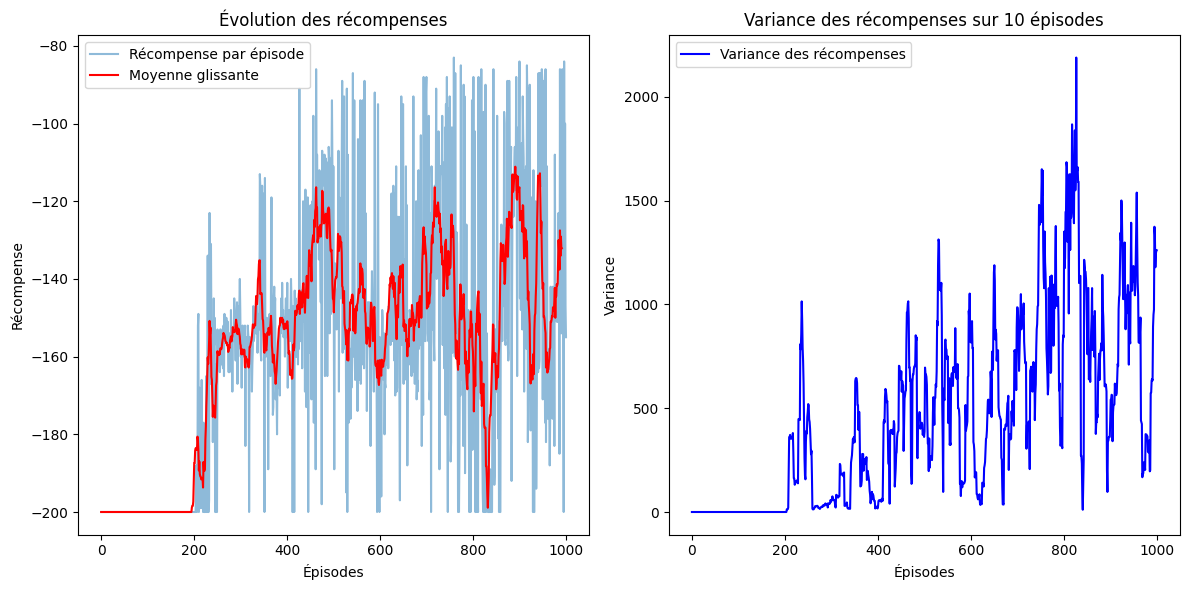
# **Analyse des résultats du DQN**

Pour analyser la performance de notre code, nous avons fait le choix d’étudier l’évolution des récompenses, ainsi que la variance de ces dernières pour l’apprentissage du problème Mountain Car avec plusieurs variantes de l’algorithme Deep Q-Network (DQN). L’étude de l’évolution et de la variance permet de comparer l’efficacité en termes de rapidité de convergence, de stabilité et de performance finale.

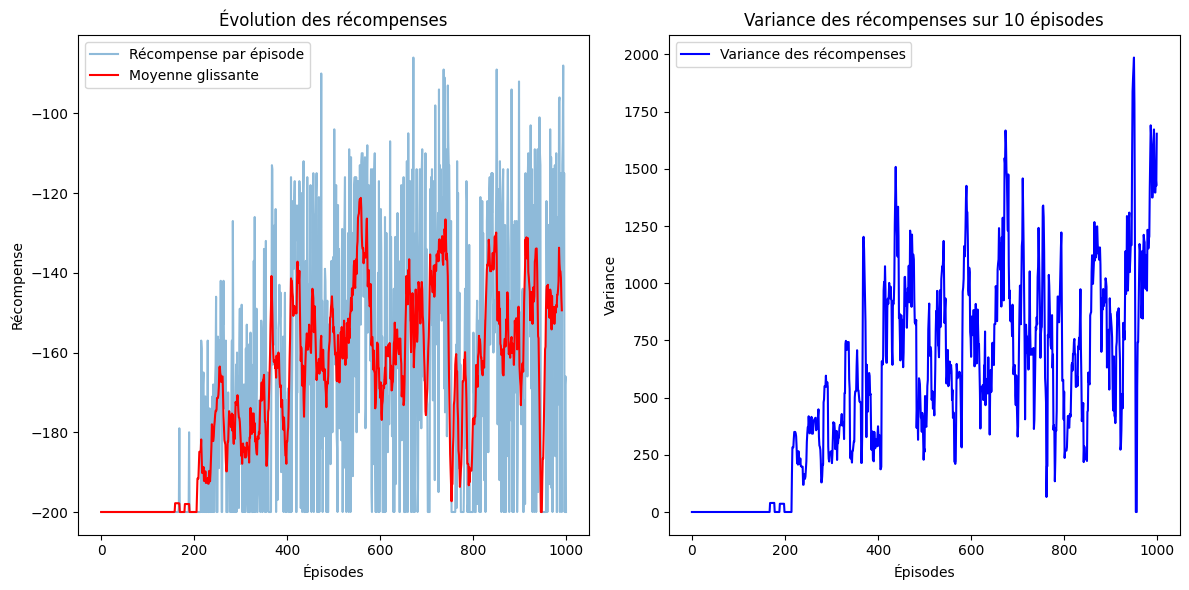
En observant les courbes de récompenses, on remarque que le DQN classique présente une convergence plus lente et une fluctuation importante des récompenses. Cette instabilité est typique du DQN de base, qui souffre du phénomène de surestimation des valeurs d’action, ce qui peut mener à un apprentissage inefficace et une convergence irrégulière.

**Figure 1 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour DQN basique.**

Lorsque le mécanisme de Prioritized Experience Replay est introduit dans le DQN, la convergence devient plus rapide. Cet effet s’explique par le fait que la stratégie de relecture prioritaire favorise l’apprentissage des transitions les plus importantes, accélérant ainsi l’amélioration de la politique. Contrairement à l’expérience replay standard qui sélectionne des échantillons de manière aléatoire.

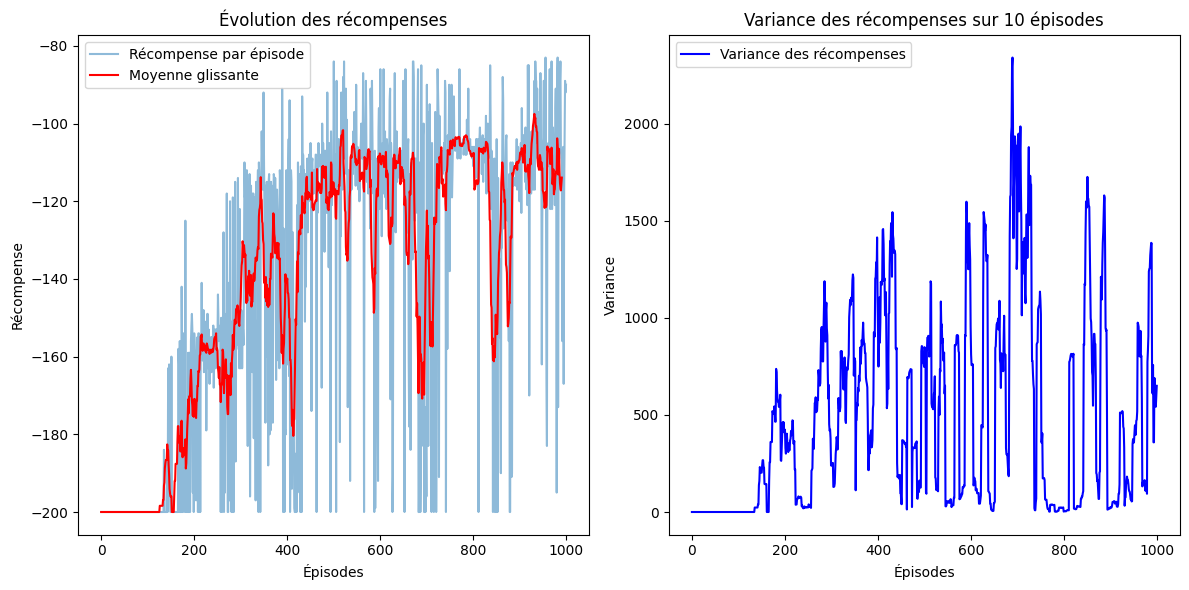


**Figure 2 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour DQN avec Prioritized Experience Replay.**

L’utilisation du Double DQN, qui corrige la surestimation des valeurs d’action en utilisant un réseau distinct pour sélectionner les actions et évaluer leur valeur, contribue également à une amélioration de la convergence. On observe une diminution des fluctuations des récompenses et une meilleure stabilité de l’apprentissage comparé au DQN classique.  


**Figure 3 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour Double DQN.**

Enfin, la combinaison du Double DQN avec le Prioritized Experience Replay représente la variante la plus performante. Cette approche bénéficie à la fois de la réduction de la surestimation des valeurs d’action et d’un échantillonnage plus intelligent des expériences de l’agent. Les courbes montrent une montée plus rapide des récompenses et une stabilisation précoce, ce qui indique que l’agent atteint une bonne politique en un nombre d’épisodes réduit.



**Figure 4 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour Double DQN avec Prioritized Experience Replay.**

Bien sûr, il est important d’étudier la question de la stabilité des agents. La variance des récompenses au cours de l’apprentissage est un indicateur important pour comparer les méthodes DQN et conclure sur leur stabilité. Une variance élevée signifie que l’agent alterne entre de bonnes et de mauvaises décisions de manière irrégulière, ce qui peut être un signe d’instabilité dans l’apprentissage.

Le DQN classique présente une variance significative, ce qui montre que l’agent a des difficultés à stabiliser sa politique.

Avec l’introduction du Prioritized Experience Replay, la variance diminue, ce qui suggère une stabilisation plus rapide du comportement de l’agent. L’amélioration est encore plus marquée avec le Double DQN, car la correction de la surestimation empêche l’agent de se baser sur des évaluations incorrectes des actions, ce qui réduit les fluctuations excessives.

La combinaison du Double DQN et du Prioritized Experience Replay montre la variance la plus basse, indiquant un apprentissage plus fluide et une meilleure robustesse de l’agent face aux variations de l’environnement. Cette combinaison permet donc d’obtenir une politique efficace et stable.

# **Conclusion: Performance finale et choix de la meilleure approche**

L’analyse des courbes d’évolution et de variance des récompenses met en évidence que l'algorithme basique du DQN est la méthode la moins efficace en raison de sa lenteur de convergence et de son instabilité. Il atteint une performance acceptable après un grand nombre d’épisodes, mais de manière irrégulière. L’ajout du Prioritized Experience Replay améliore considérablement la rapidité d’apprentissage en donnant plus de poids aux transitions importantes, ce qui permet d’accélérer la progression de l’agent. Cependant, cette approche seule ne corrige pas totalement la surestimation des valeurs d’action, ce qui peut encore entraîner des choix sous-optimaux. Le Double DQN, en revanche, réduit la surestimation et offre un apprentissage plus stable, bien qu’il ne soit pas nécessairement plus rapide que le DQN standard dans ses premières phases d’apprentissage. Enfin, la combinaison du Double DQN et du Prioritized Experience Replay représente la meilleure solution parmi les variantes testées. Elle permet d’obtenir une convergence rapide, une stabilité améliorée et une meilleure politique finale. Cette approche est donc la plus adaptée parmi les quatre pour résoudre efficacement le problème du Mountain Car.

On en conclut que le Double DQN avec Prioritized Experience Replay est l’approche à privilégier dans l’environnement *MountainCar-v0* de *Gym*. Bien que ce dernier a de meilleures performances par rapport aux autres variantes de DQN, elle reste sensible aux choix des hyperparamètres et nécessite un nombre d’épisodes conséquent pour atteindre une politique optimale.

En guise d’ouverture, une alternative intéressante repose sur les méthodes basées sur les politiques, telles que le Proximal Policy Optimization (PPO). Contrairement aux approches basées sur les valeurs, PPO ajuste directement la politique de l’agent avec des mises à jour plus stables et efficaces, ce qui pourrait offrir une alternative prometteuse pour résoudre le problème du *MountainCar* de manière plus robuste et rapide.

# **Annexe**

## **1. Paramètres d'Entraînement**

| **Paramètre** | **Valeur** |
| --- | --- |
| Taille de la mémoire d'expérience | 10 000 |
| Taille des mini-lots | 128 |
| ε initial (exploration) | 1.0 |
| Facteur de décroissance ε | 0.995 |
| ε minimum | 0.01 |
| γ (discount factor) | 0.99 |
| Taux d'apprentissage | 0.004 |

## 

## 

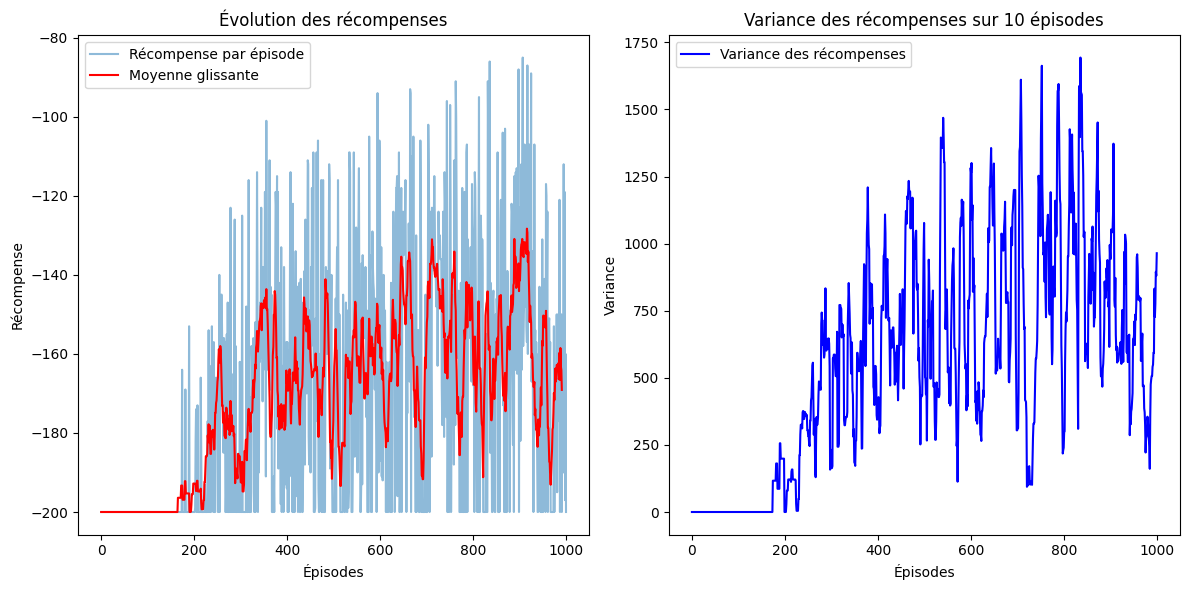
## 

## 

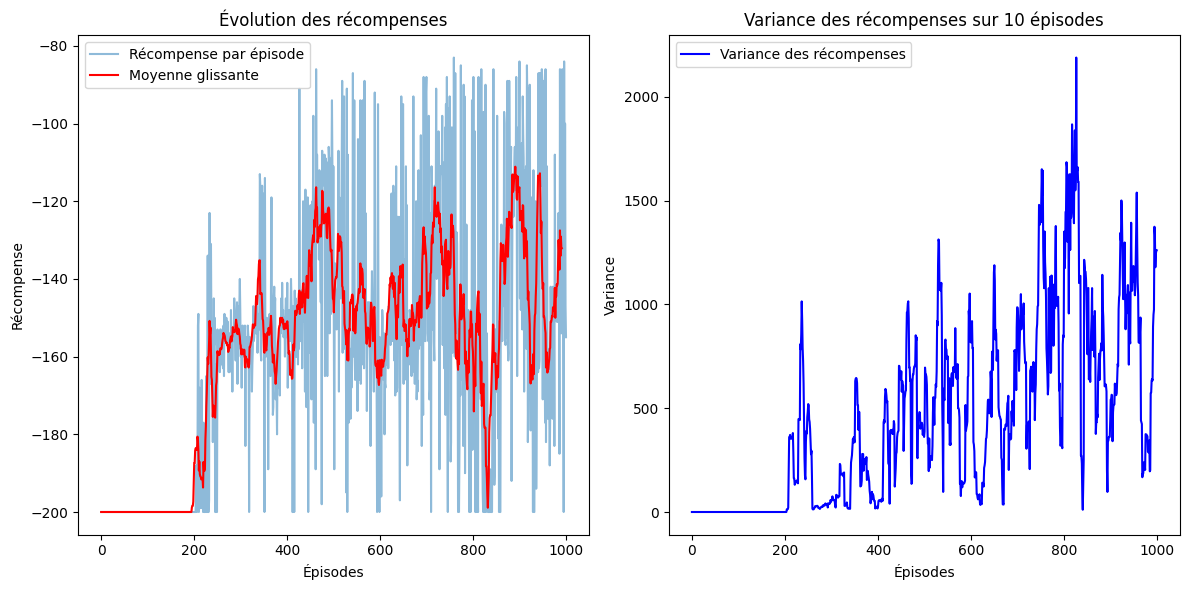
## 

## **2. Figures et animations**

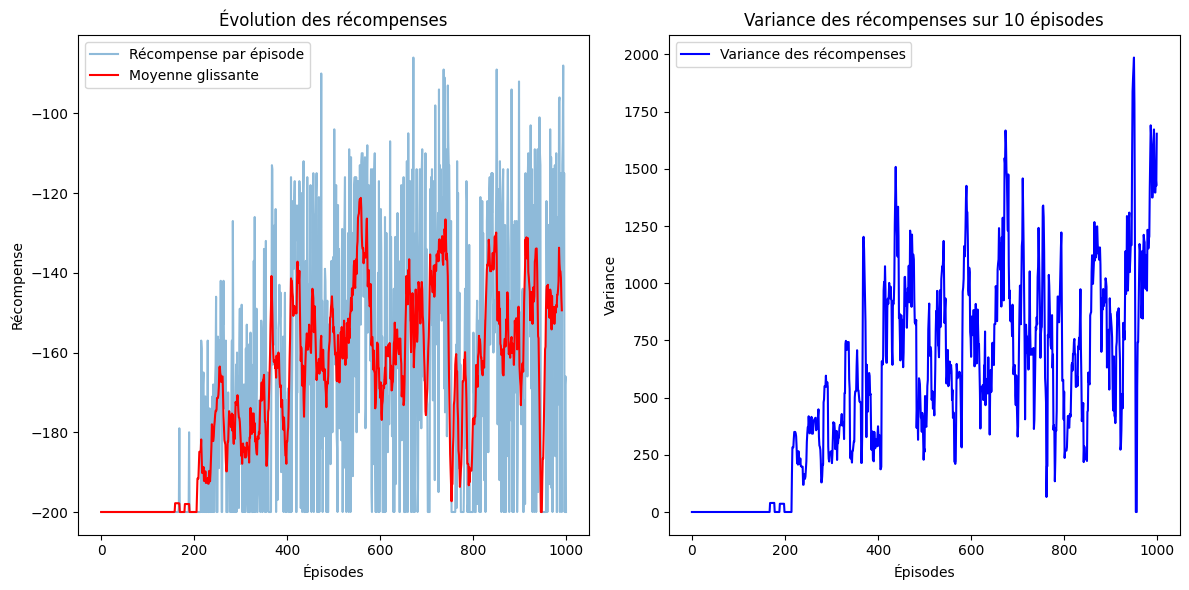
**Figure 1 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour DQN standard.**



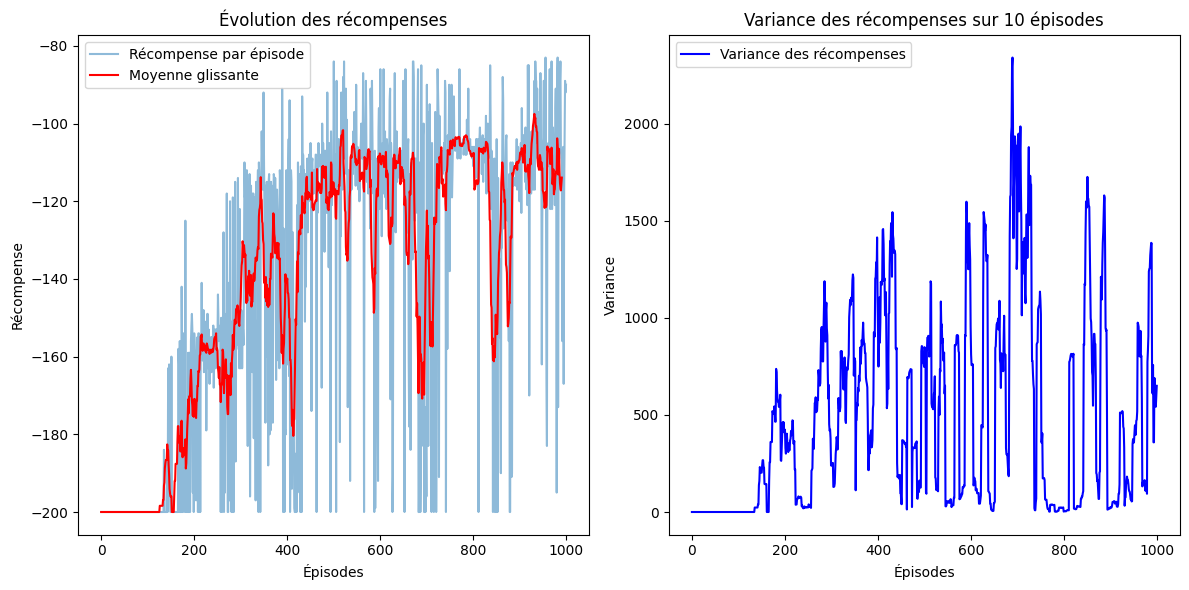
**Figure 2 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour DQN avec Prioritized Experience Replay.**



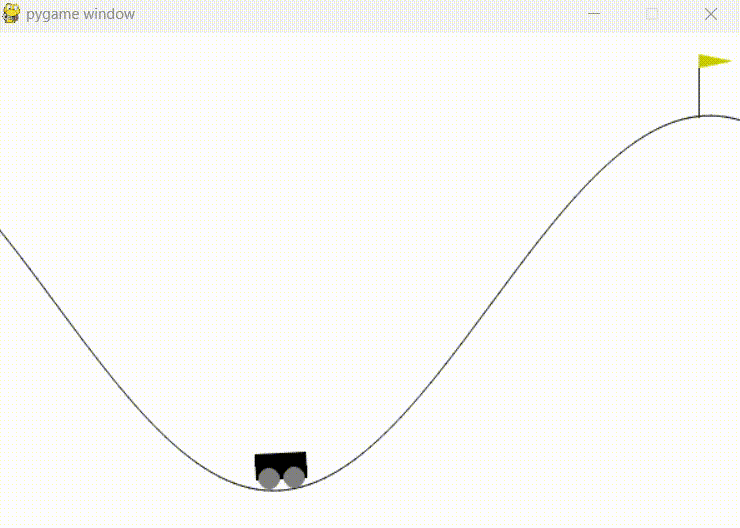
**Figure 3 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour Double DQN.**



**Figure 4 : Courbes d'évolution et de variance des récompenses en fonction des épisodes, pour Double DQN avec Prioritized Experience Replay.**



**GIF 1 : Exemple d'un épisode dans MountainCar-v0 généré par le code.**



**GIF 2 : Exemple d'un épisode dans LunarLander-v3 généré par le code.**

