

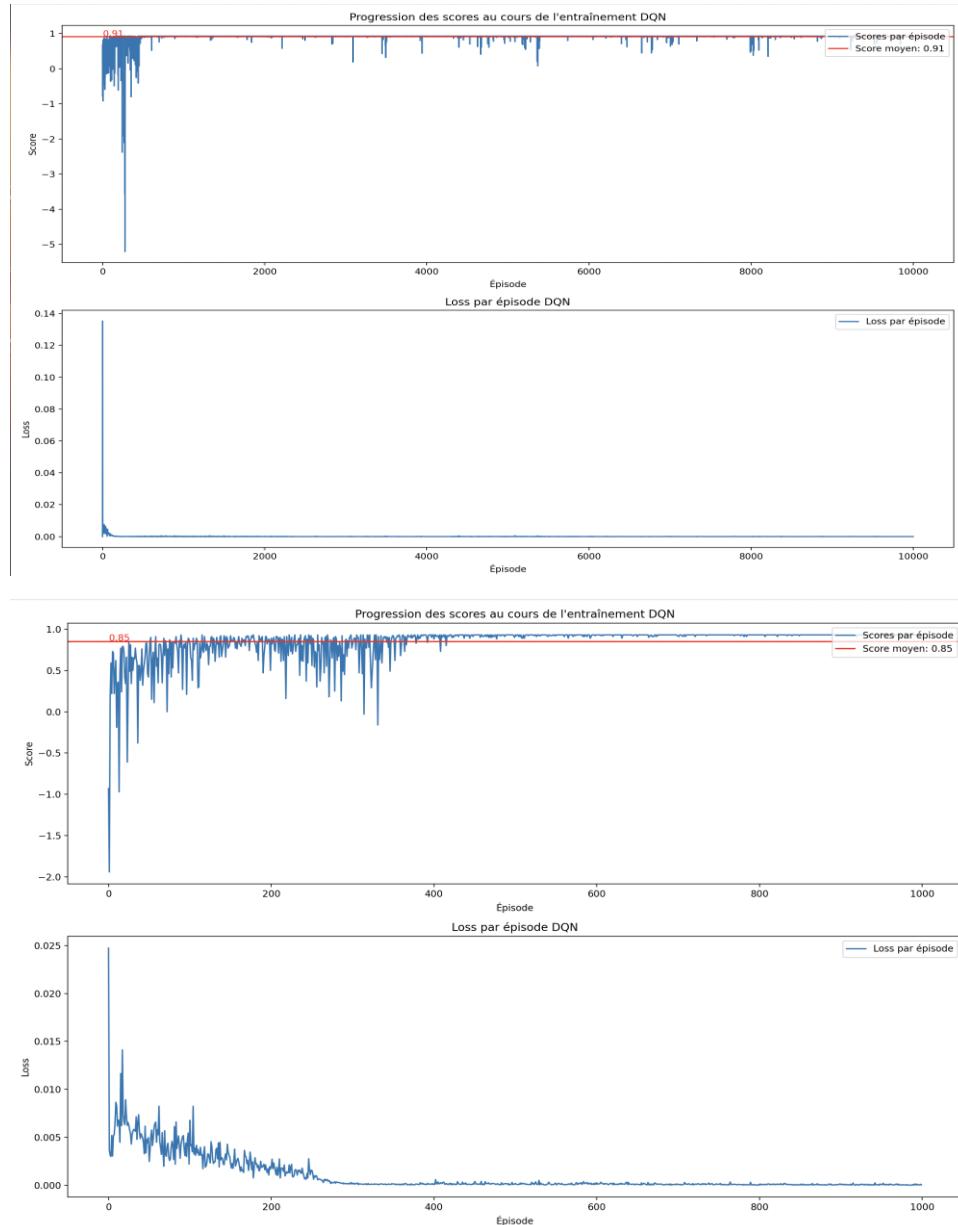
Rapport

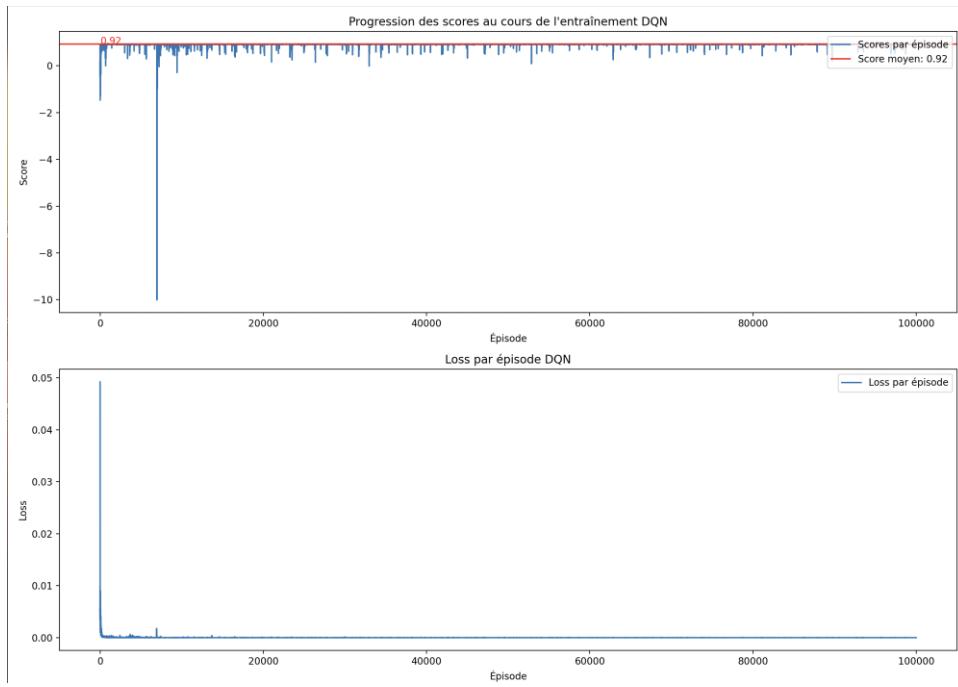
Pour ce projet nous avons choisis les jeux BalloonPop et Can't stop. Nous avons commencé par développer les jeux puis des environnements adaptés à nos models. Nous pouvons lancer les modèles ainsi que le jeu via notre main.

Nous avons d'abord testé nos models sur Tic-Tac-Toe, Gridword et LineWord. Puis nous les avons lancés sur nos environnements.

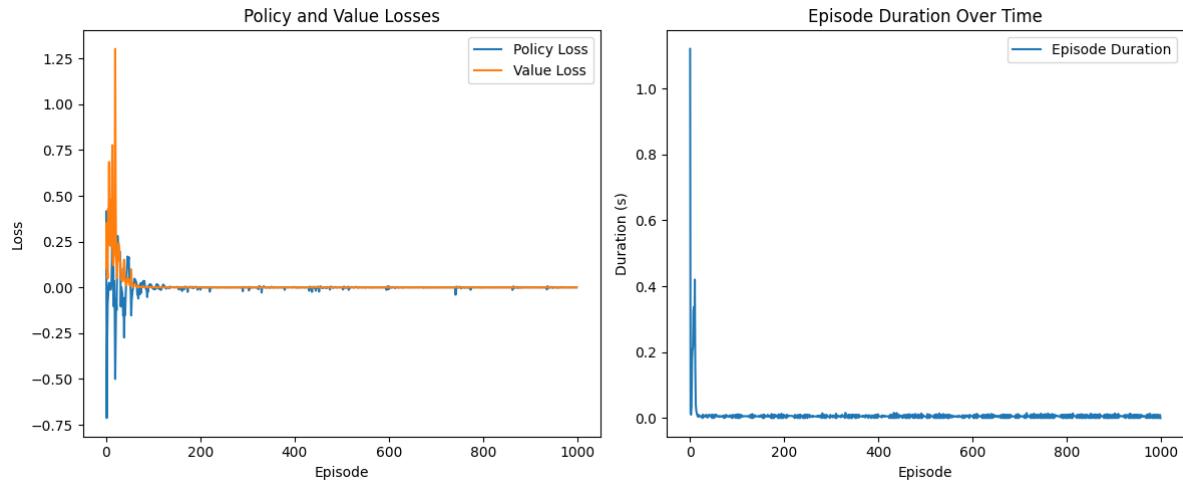
Résultat Gridword

DQN





PPO

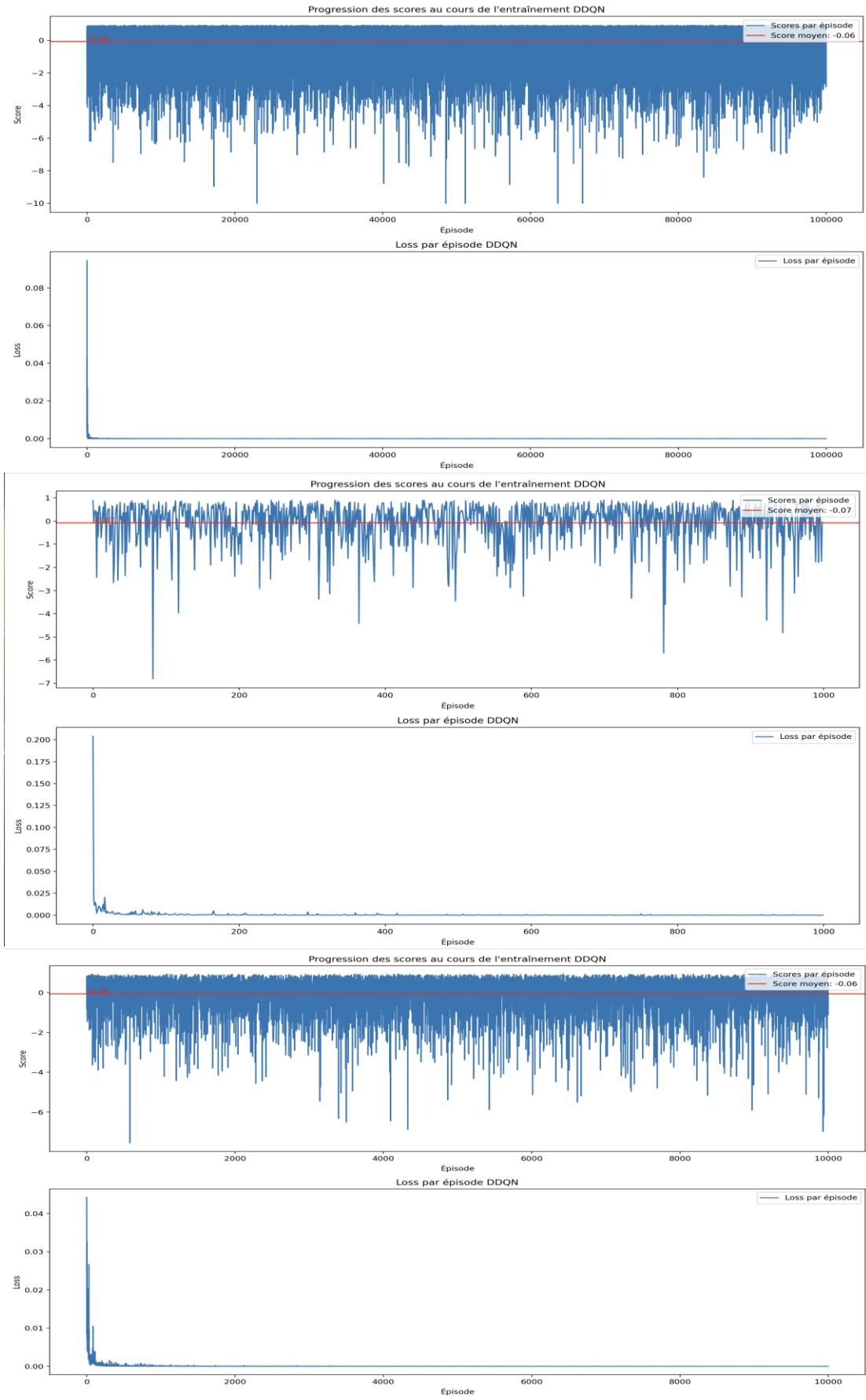


Ce graphique montre deux métriques de "perte" (loss).

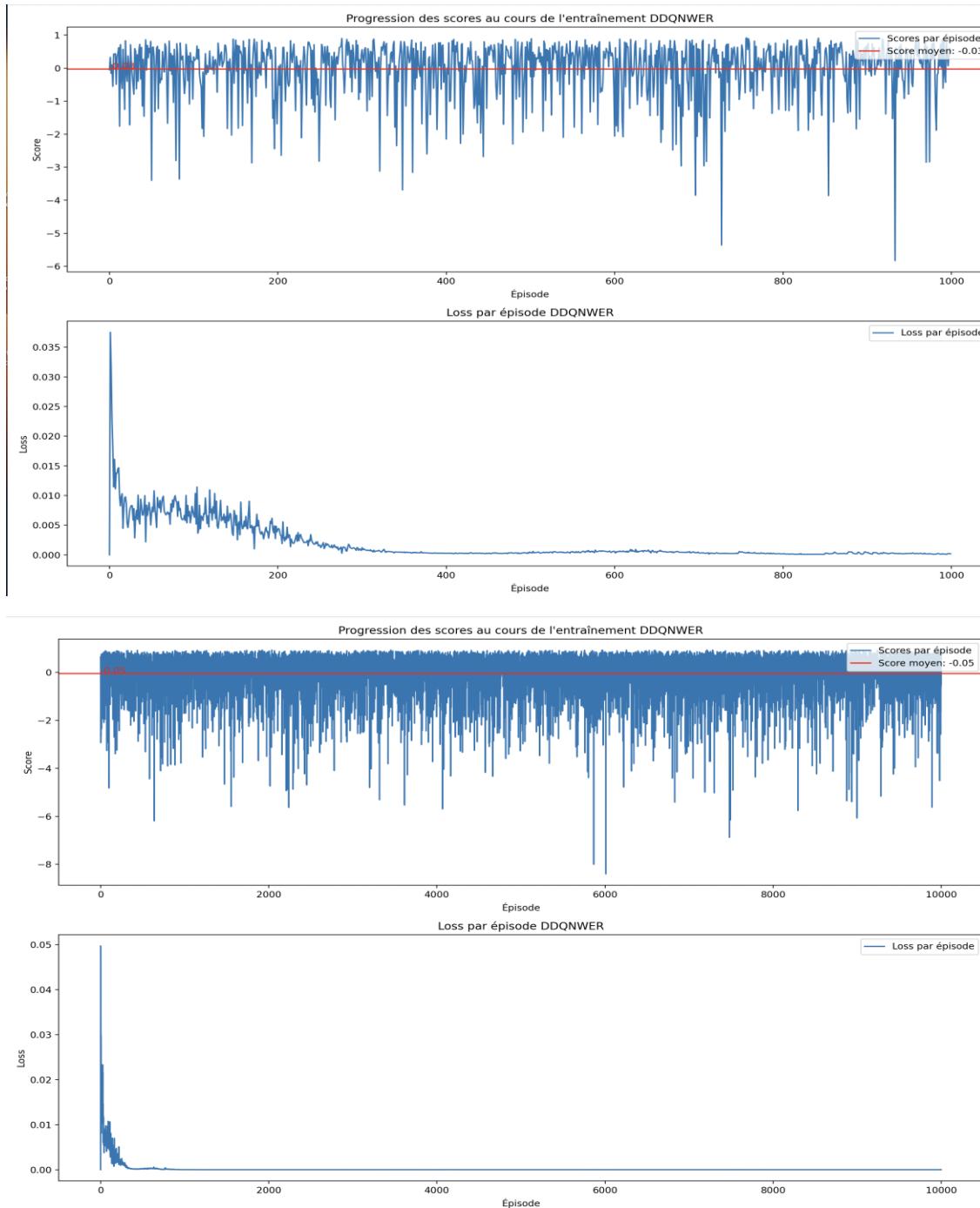
Dans ce graphique, on observe que les deux pertes diminuent rapidement au début, ce qui indique que l'algorithme apprend rapidement. Les pertes semblent se stabiliser, indiquant que l'algorithme a atteint une sorte de plateau dans son apprentissage.

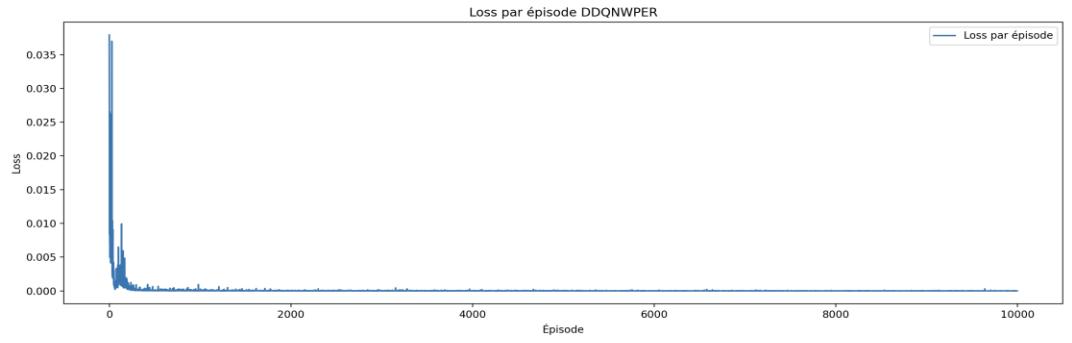
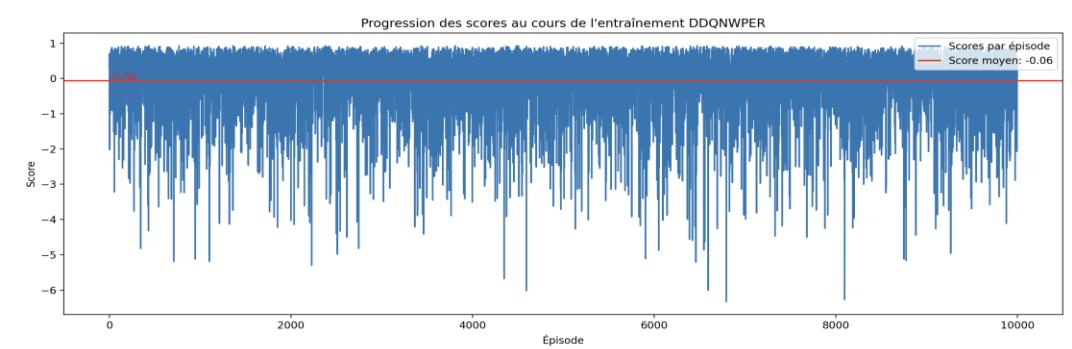
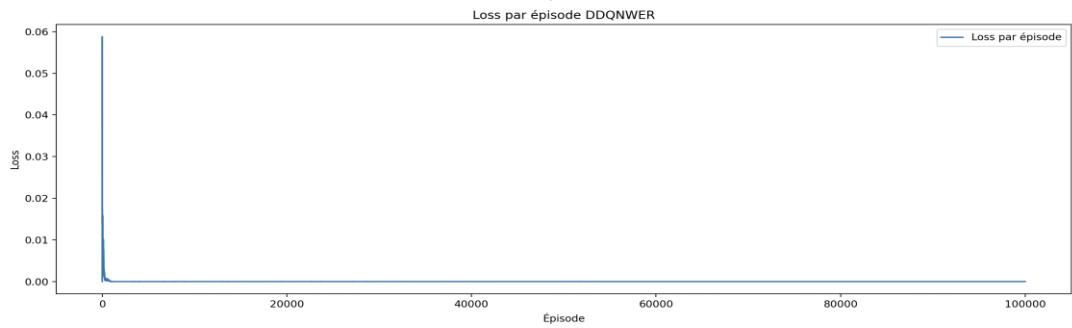
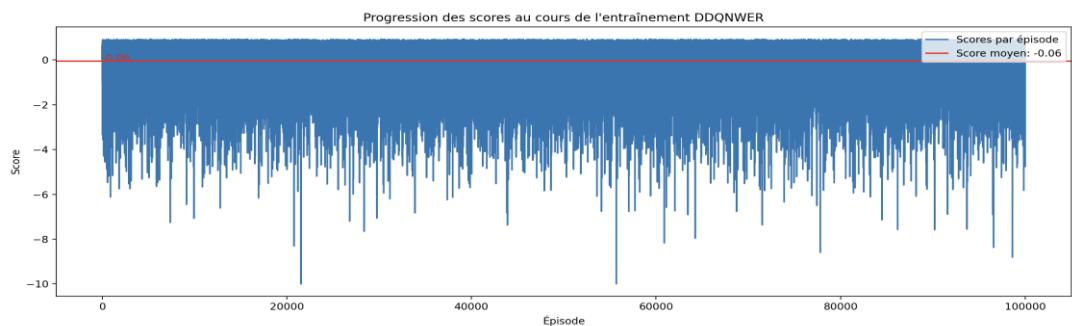
Qui n'ont pas très bien marcher mais voici les résultats

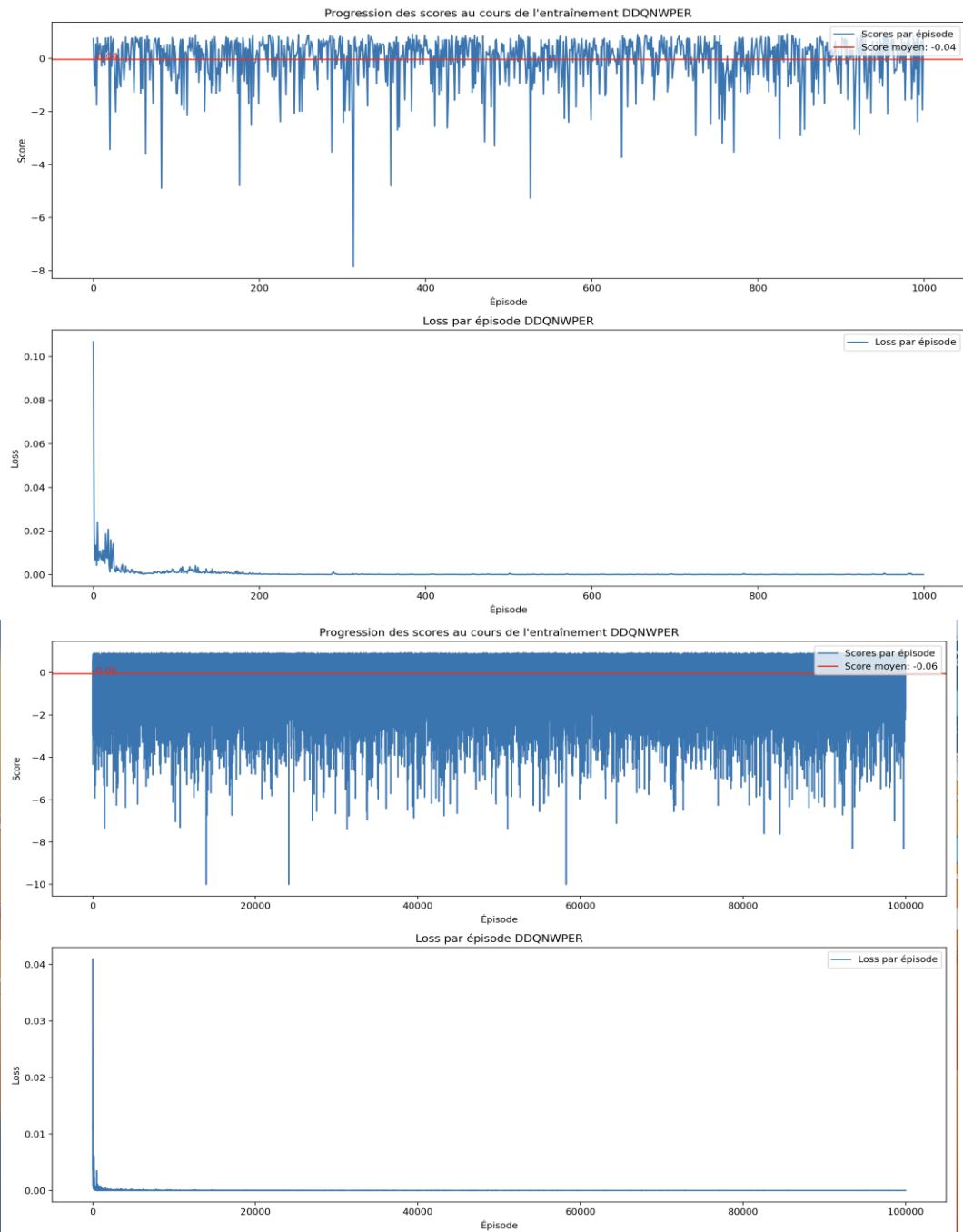
DDQN



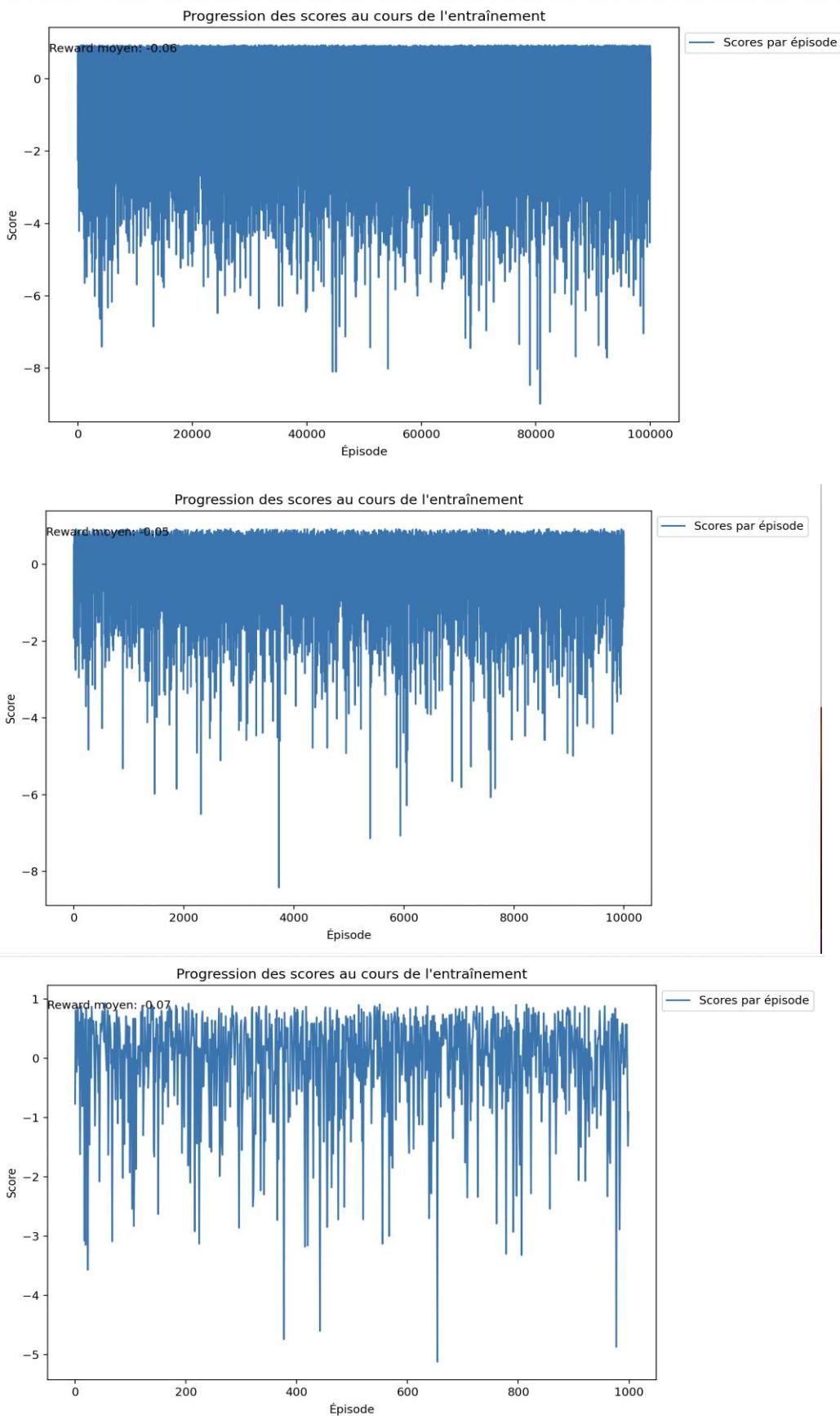
DDQNWER





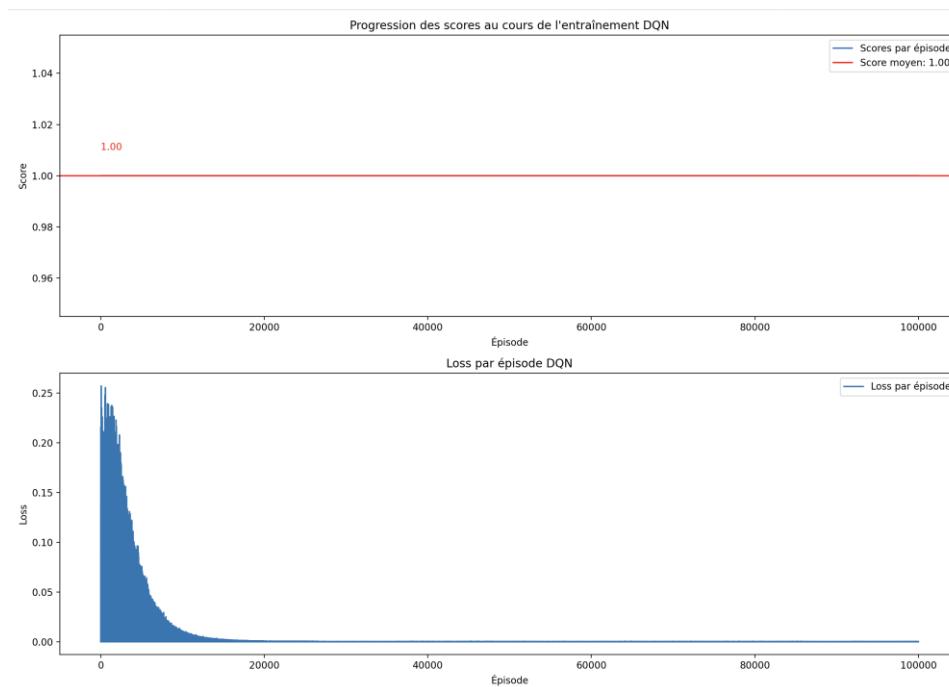
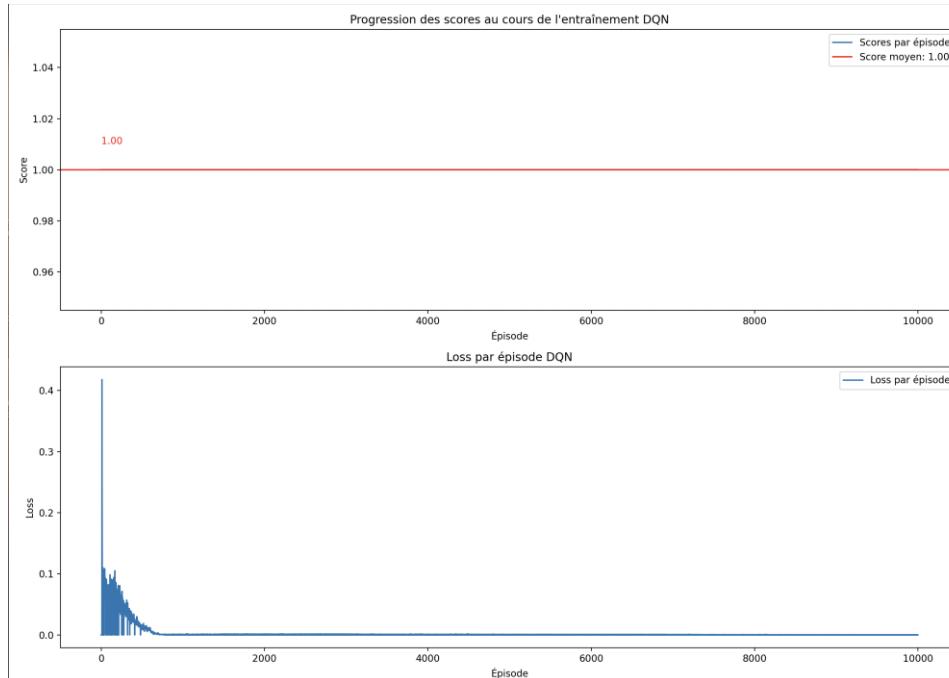


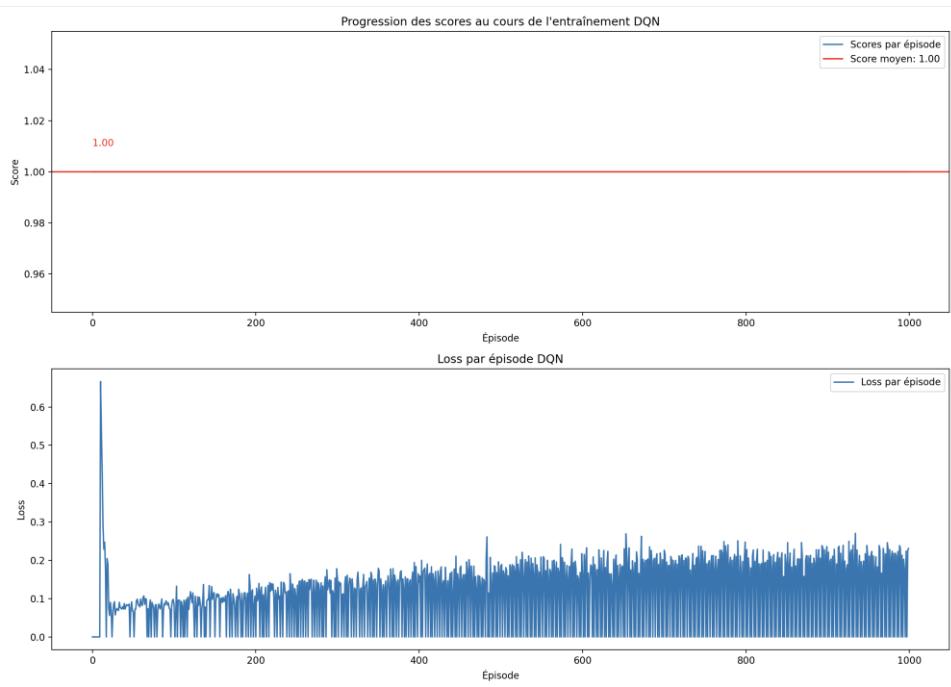
Random Rollout



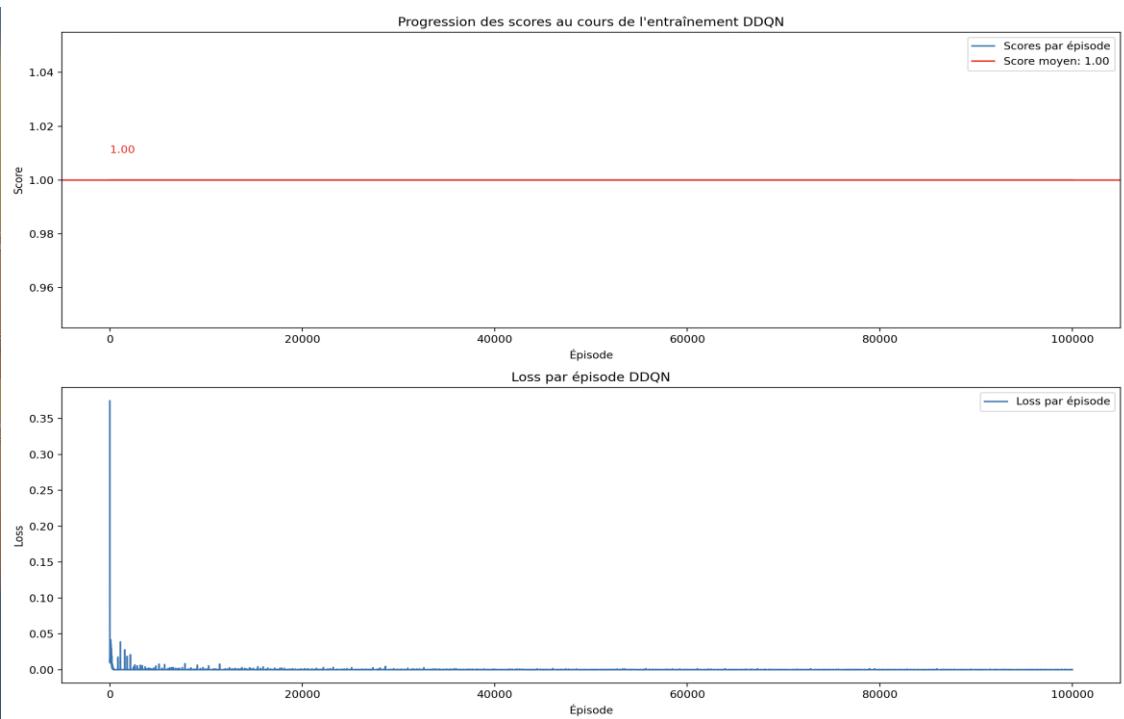
Résultat LineWord

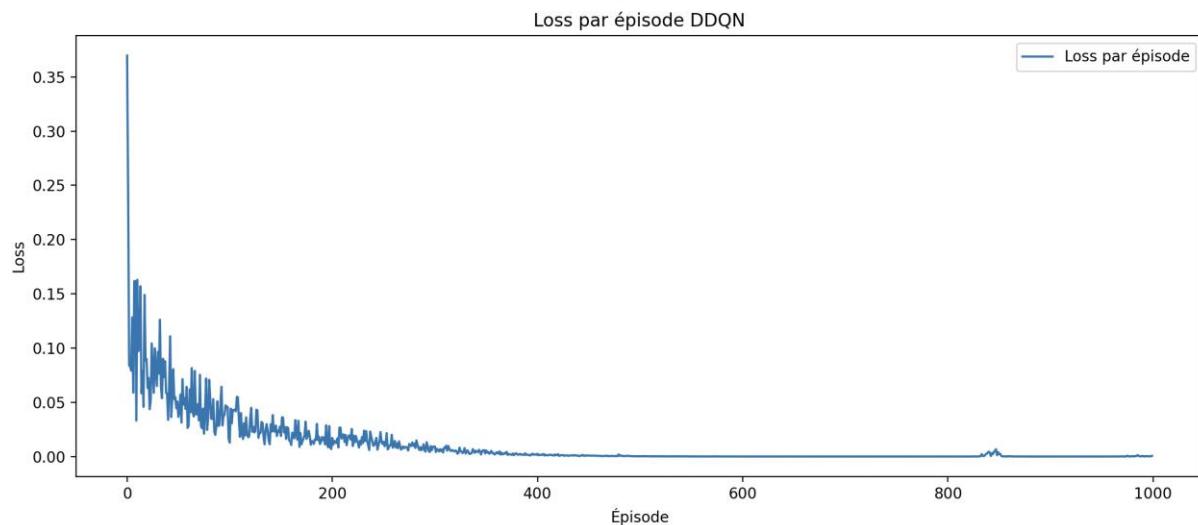
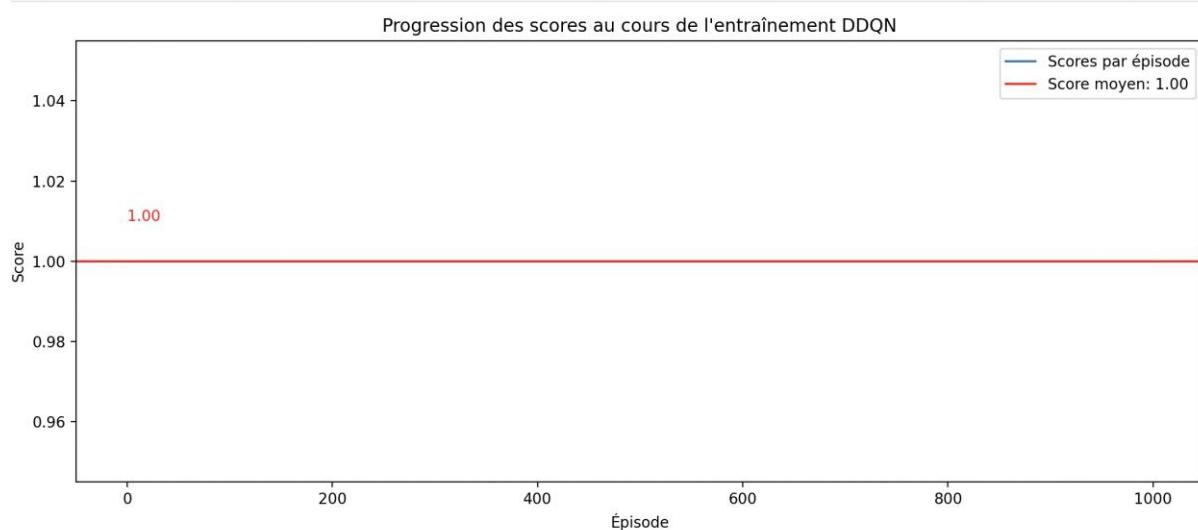
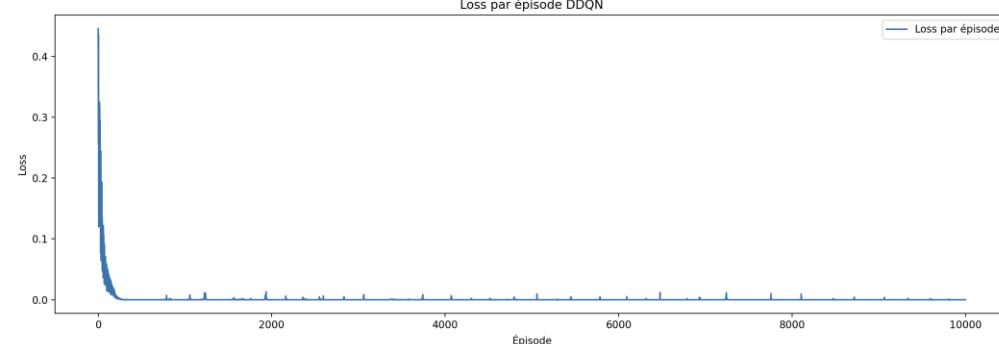
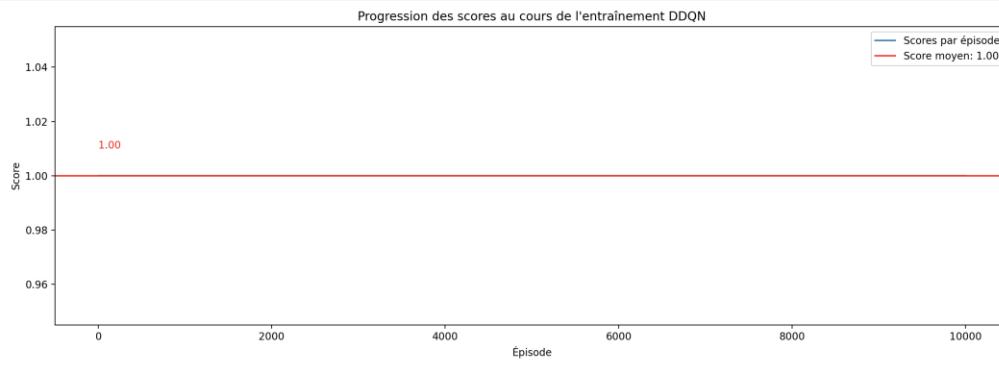
DQL



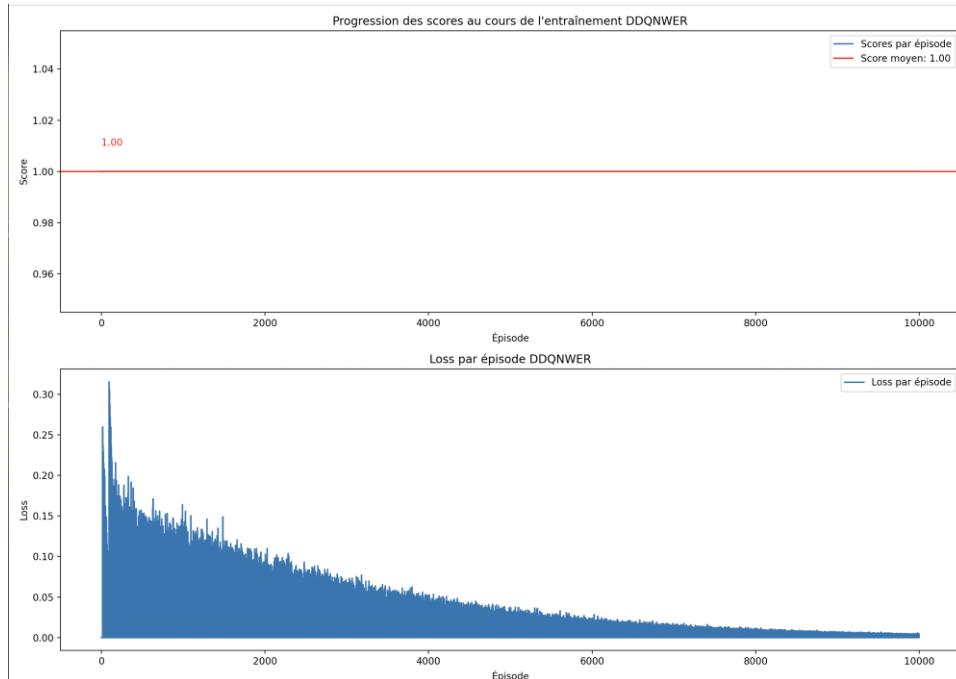


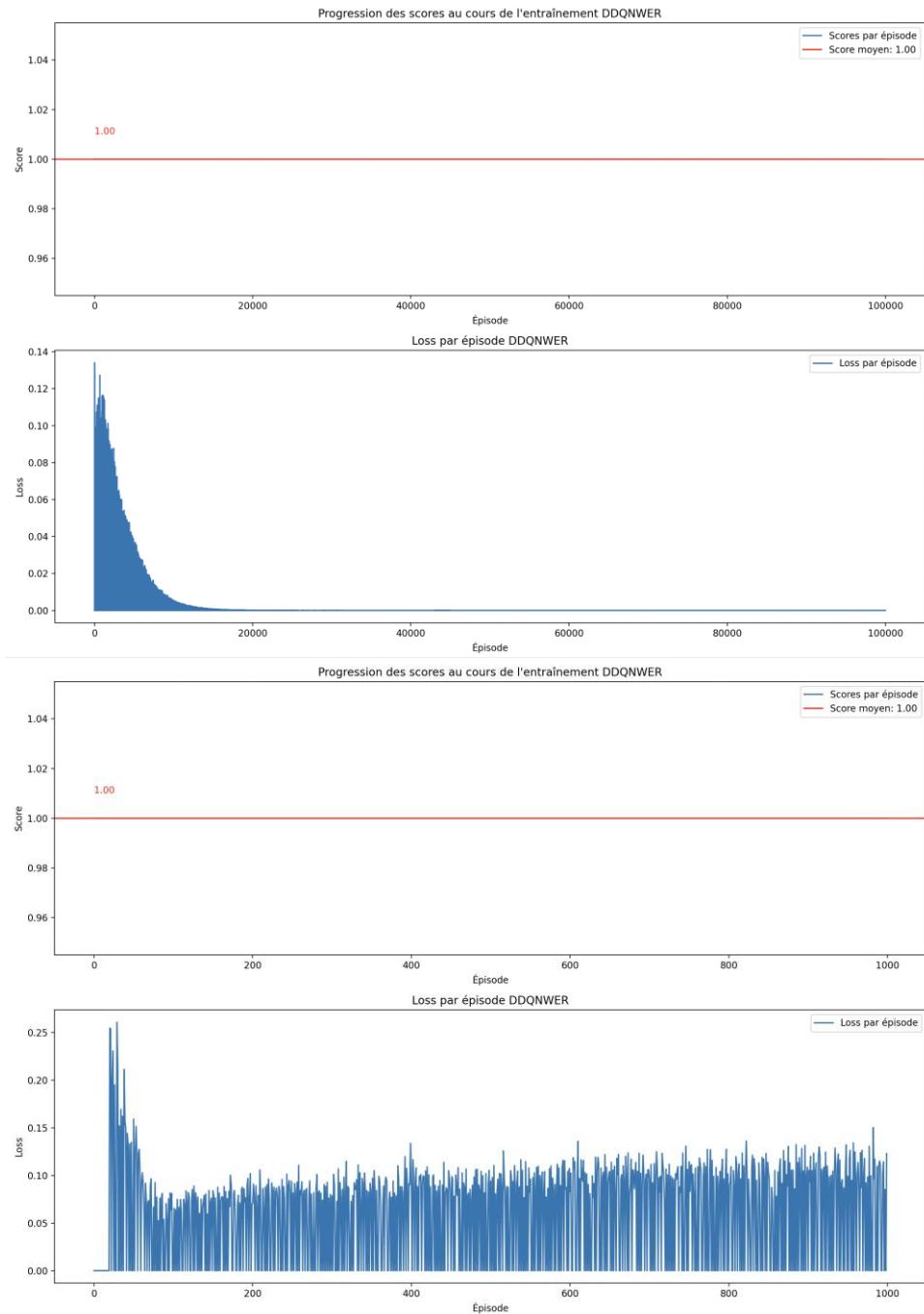
DDQN



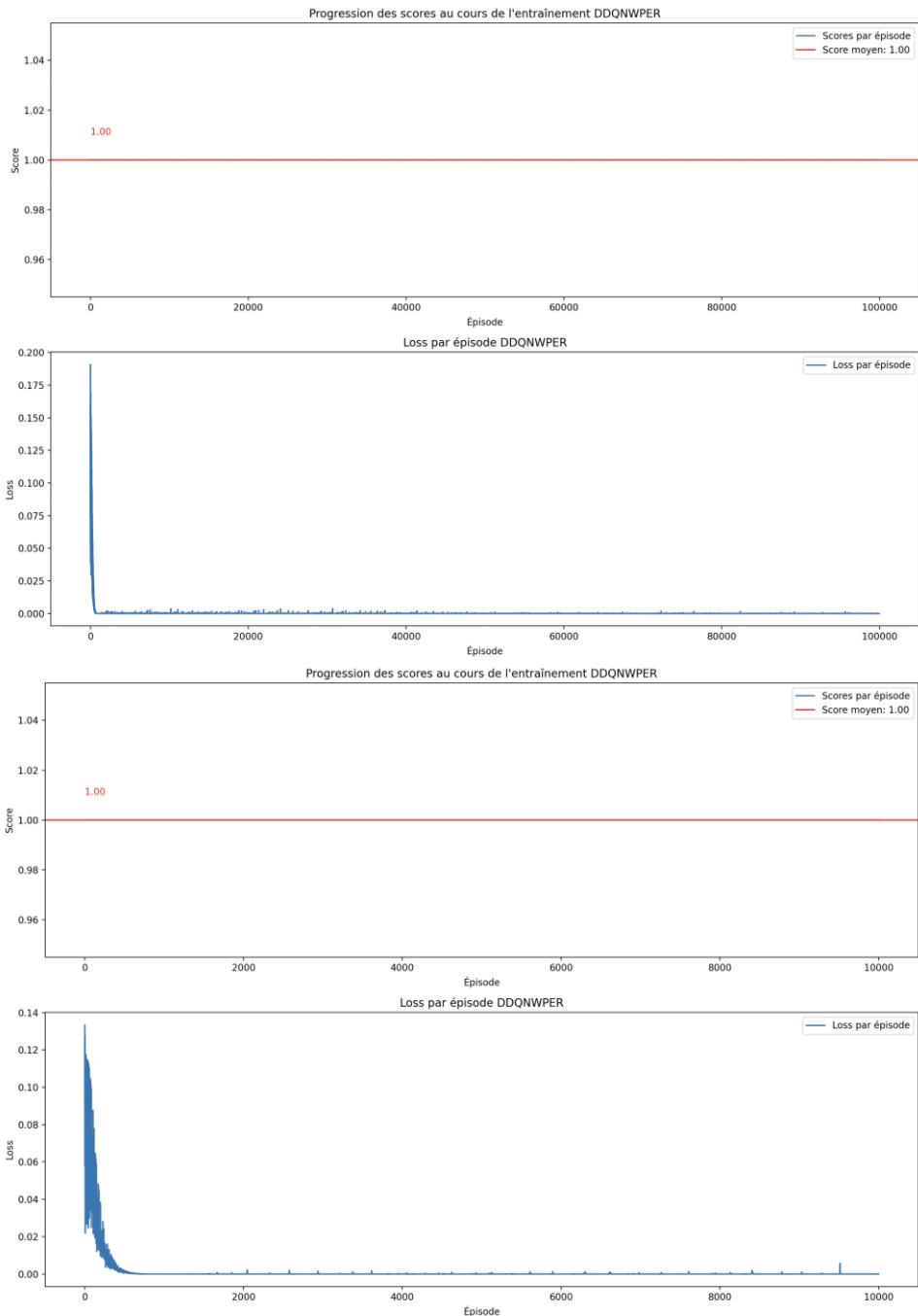


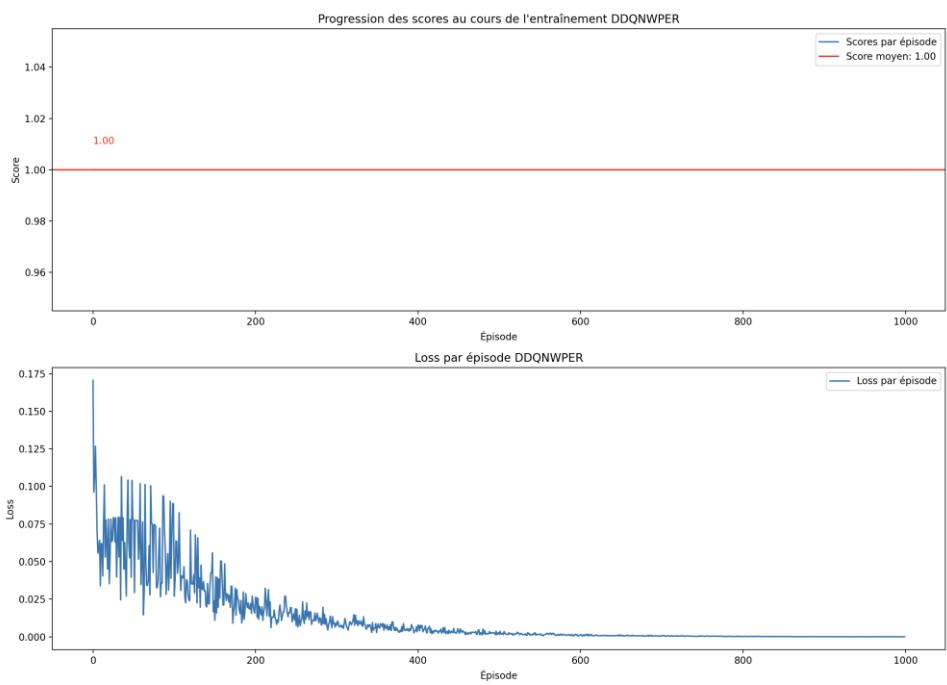
DDQLWER



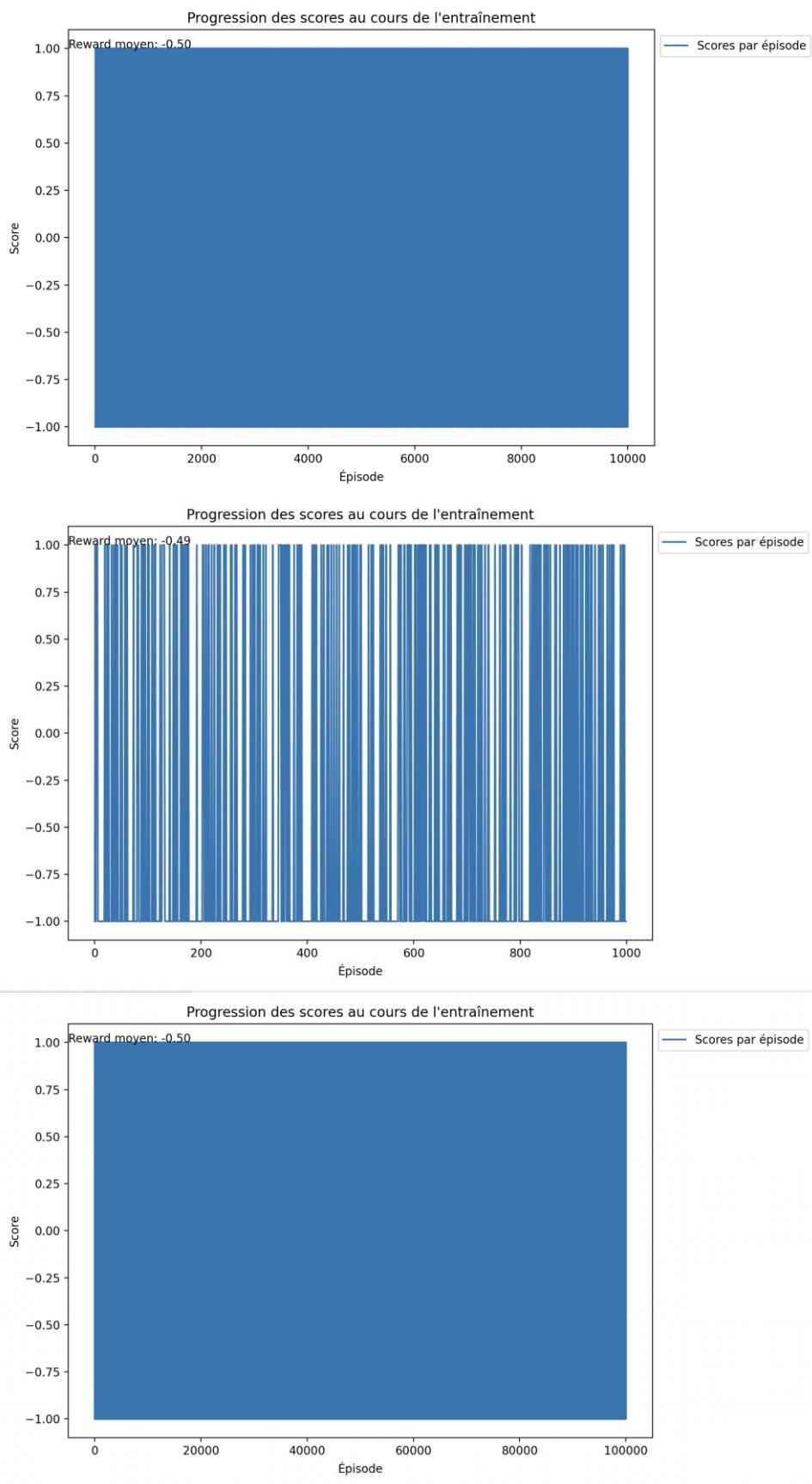


DDQNWPER



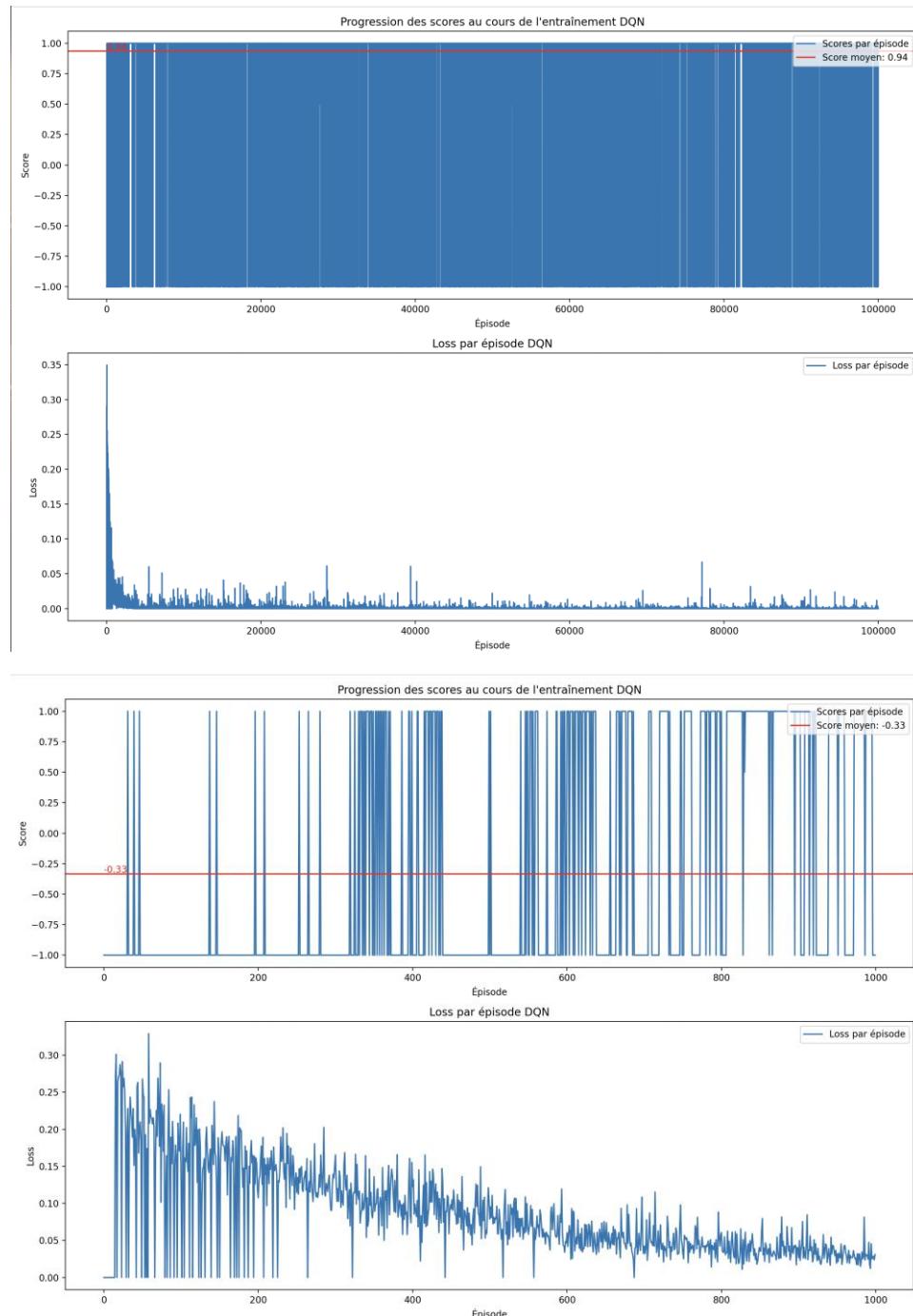


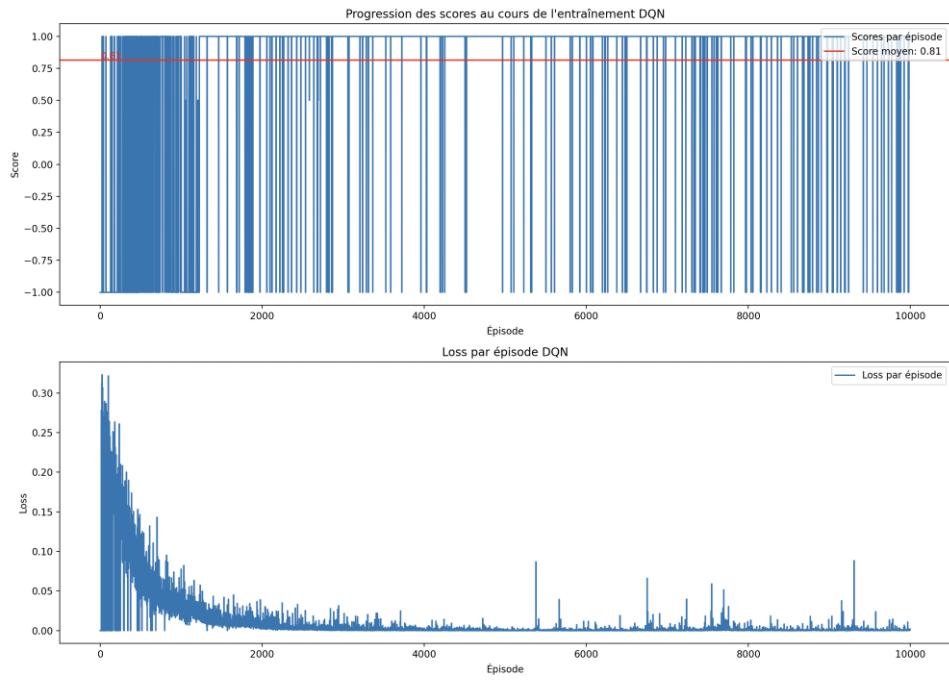
Qui n'ont pas très bien marcher mais voici les résultats



Résultat Tic-Tac-Toe

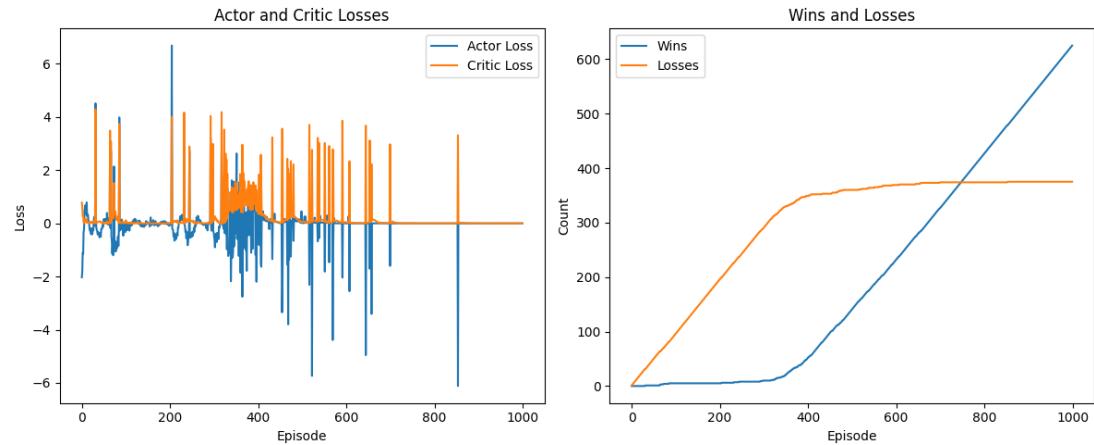
DQN





PPO:

Résultat sur 1000 épisodes:



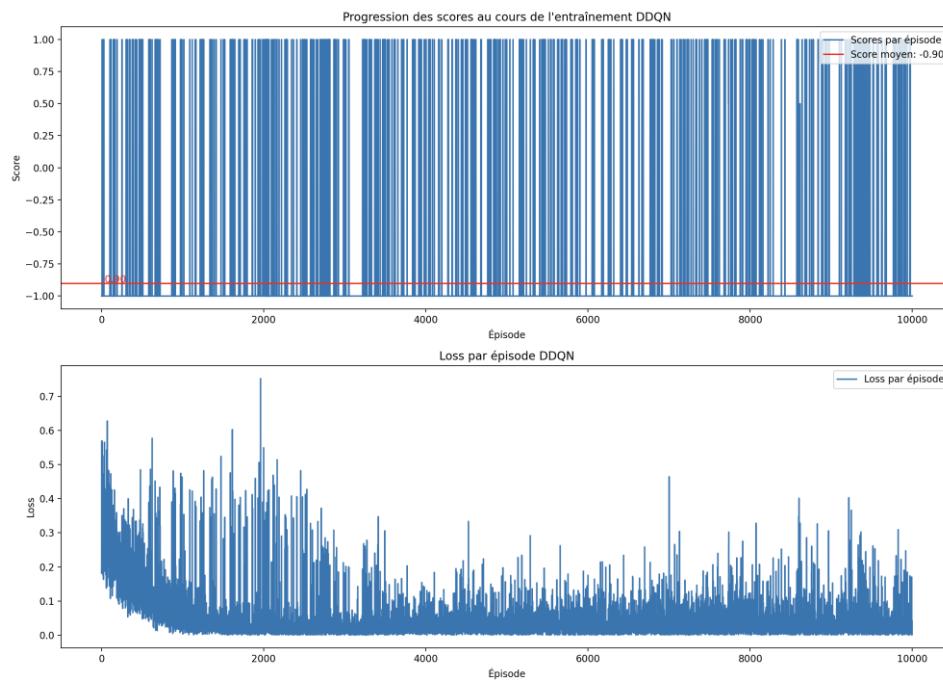
Score : Total Wins: 781, Total Losses: 219

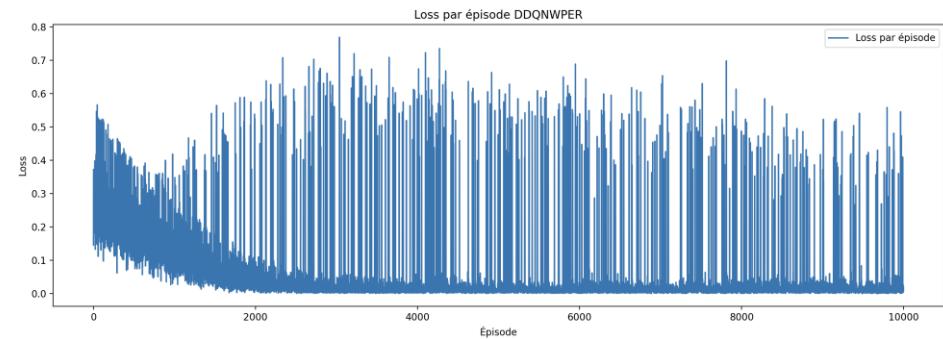
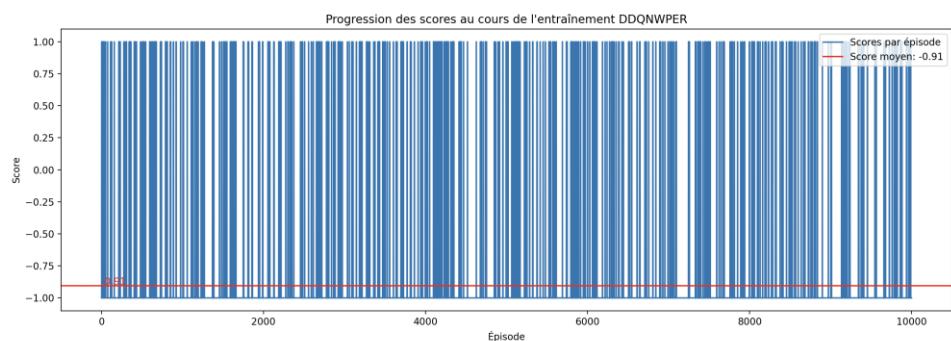
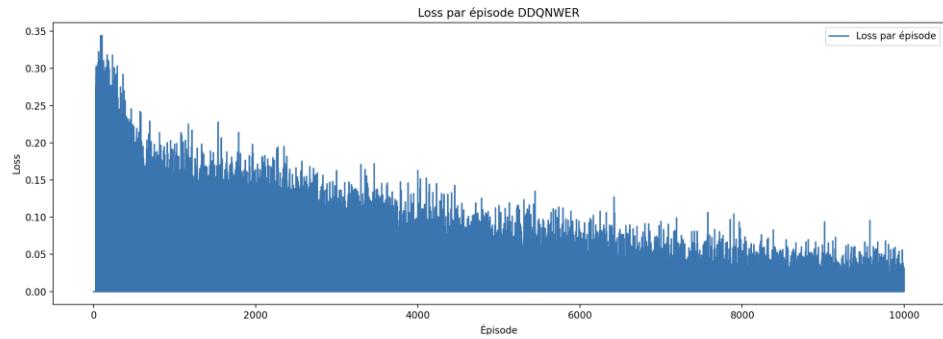
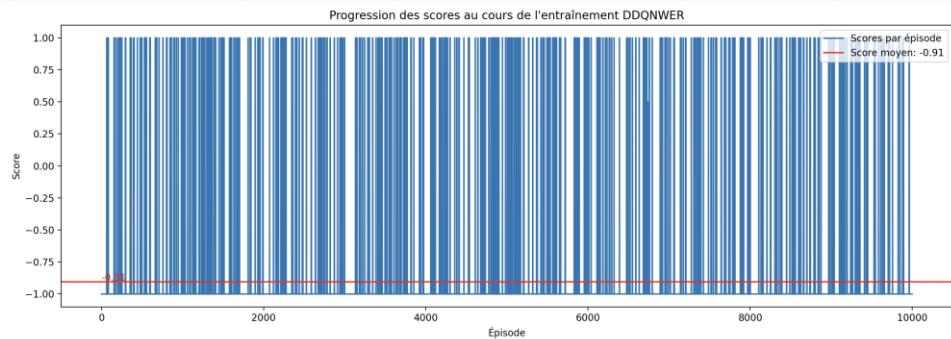
Temps: 0.01 pour 100 épisodes

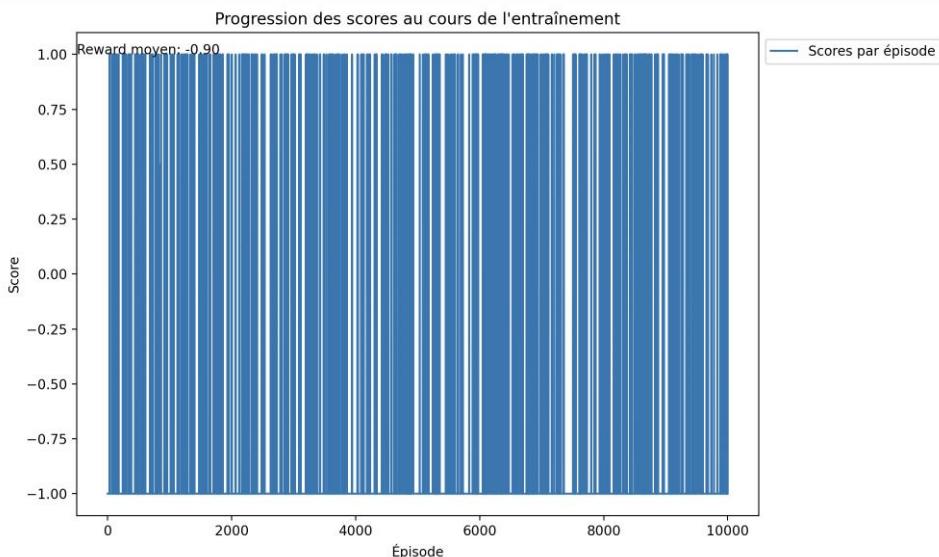
Le premier graphique, montre deux métriques de perte qui varient au fil des épisodes. Le "Actor Loss" (en bleu) et le "Critic Loss" (en orange) semblent avoir une tendance générale à la baisse, ce qui suggère que l'algorithme apprend et s'améliore au fil du temps. Les pertes négatives indiquent que l'algorithme fait mieux que les prédictions aléatoires. Cependant, il y a une certaine irrégularité, surtout avec la perte du critique, ce qui peut indiquer que l'apprentissage n'est pas totalement stable ou que l'algorithme explore encore de nouvelles stratégies.

Le second graphique, montre le cumul des victoires (en bleu) et des défaites (en orange). On constate que le nombre de victoires augmente de manière plus significative que le nombre de défaites, ce qui suggère que l'algorithme devient meilleur pour gagner le jeu au fil du temps.

Qui n'ont pas très bien marcher mais voici les résultats







Construction de Balloon Pop !!!

Le jeu de "Balloon Pop" est un jeu de dés avec un plateau dans lequel les joueurs doivent atteindre le score le plus élevé avant d'éclater 3 ballons. Chaque colonne du jeu représente un type de ballon ou une forme, et les joueurs gagnent des points en faisant progresser leurs pions jusqu'à faire atteindre 3 sommets.

Dans ce projet, nous avons modélisé le jeu de "Balloon Pop" en utilisant des concepts de programmation orientée objet (POO) et avons implémenté une simulation pour jouer au jeu.

Nous avons 4 fichiers principaux pour le jeu Balloon Pop !!!

- Main
- Model
- View
- Controller

Pour modéliser le jeu de "Balloon Pop", nous avons créé une classe Python appelée `BalloonGameSolo` dans `Model`. Cette classe comprend nos propriétés et méthodes pour représenter et simuler le jeu.

Nous avons défini principalement des définitions pour :

- Que Chaque colonne représente un type de ballon et soit définie avec une liste de scores possibles correspondants à la progression du ballon dans cette colonne.
- Nous avons un ensemble initial de dés, chaque dé étant capable de générer une combinaison aléatoire de couleur et de forme de ballon.
- Nous suivons la progression dans chaque colonne du jeu et enregistrons les scores obtenus à chaque étape.
- Nous gardons une trace du nombre de "breaks" (ruptures de ballons) et des scores obtenus à chaque "break".
- Nous avons une méthode pour lancer les dés et enregistrer les résultats, en tenant compte de la possibilité de relancer certains dés.
- Une méthode pour enregistrer les scores obtenus après chaque lancer de dés, mettant à jour la progression dans chaque colonne en fonction des résultats.
- Nous avons une méthode pour simuler le jeu complet jusqu'à ce que trois "breaks" se produisent, en enregistrant les scores à chaque étape.

Pour la mise en forme de notre jeu nous utilisons les fichiers view.py et controller.py. Cela permet de gérer toute l'interface graphique du jeu. Elle est chargée de l'initialisation de la fenêtre Pygame, du chargement des ressources visuelles, de l'affichage des éléments du jeu et de la gestion des événements utilisateur.

Nous avons défini principalement des définitions pour :

- Initialiser la fenêtre Pygame, charger les ressources visuelles et configurer les positions des éléments du jeu.
- Afficher tous les éléments du jeu, y compris le plateau, les dés, les scores, les boutons et les messages contextuels.
- Gérer les événements utilisateur tels que les clics de souris et les redimensionnements de fenêtre.
- Dessiner les scores des colonnes sur le plateau.
- Dessiner les scores réels des colonnes, prenant en compte les progrès actualisés par le joueur.
- Afficher le score final obtenu après chaque "break".
- Afficher les scores des "breaks" sur l'interface.

Nous avons fait un fichier main pour pouvoir appeler le jeu en mode solo, en mode random et lancer les modèles depuis le main.

Résultat Balloon Pop!!!

Nous avons d'abord lancé en random pour voir les résultats et pouvoir les comparer voici nos résultats :

Après 1000 simulations (Point d'évaluation):

- Temps écoulé: 0.34 secondes (2956.03 parties/seconde)
- Score moyen: 122.06
- Tours moyens par partie: 7.64

Après 10000 simulations (Point d'évaluation):

- Temps écoulé: 3.12 secondes (3205.38 parties/seconde)
- Score moyen: 120.98
- Tours moyens par partie: 7.63

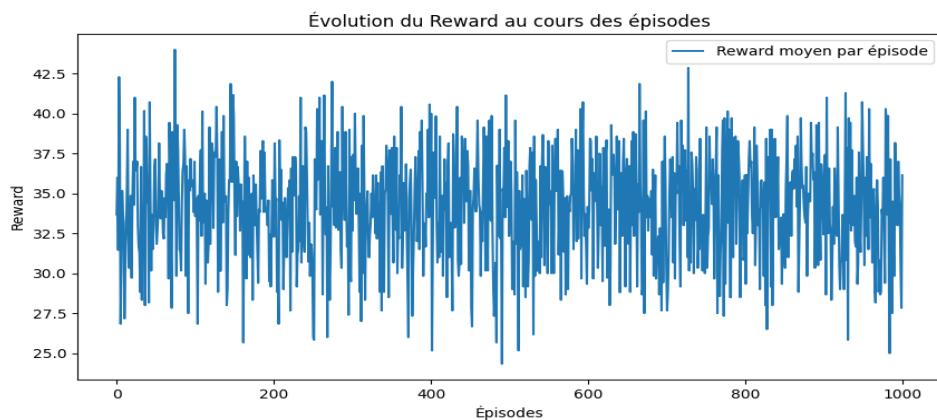
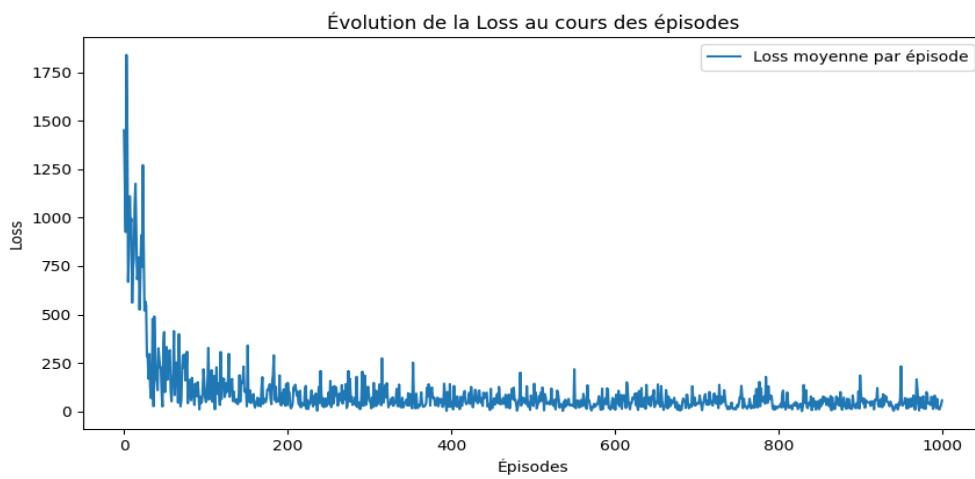
Après 100000 simulations (Point d'évaluation):

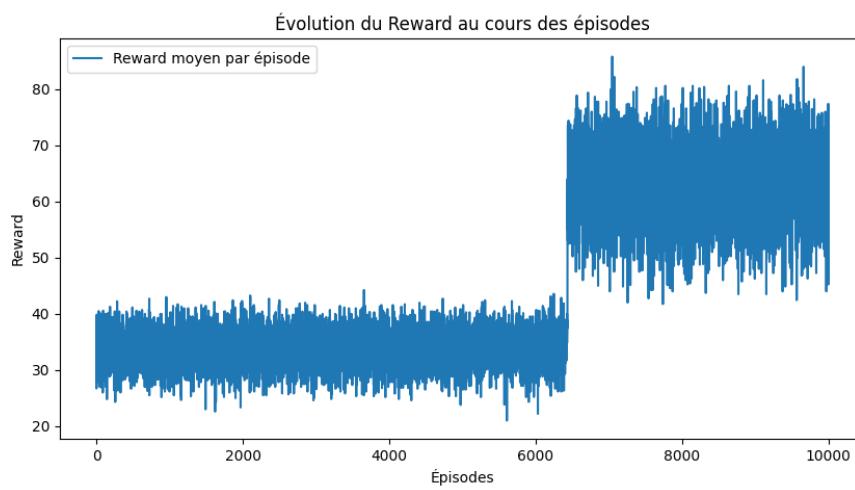
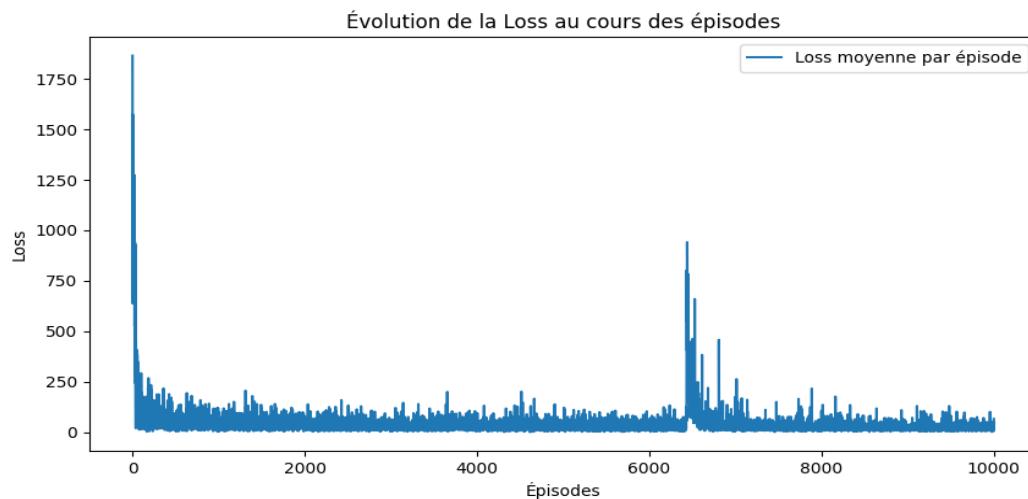
- Temps écoulé: 35.30 secondes (2832.77 parties/seconde)
- Score moyen: 121.02
- Tours moyens par partie: 7.63

On peut voir qu'on tourne autour d'une moyenne de 122 donc nous avons pour objectif de dépasser ce score.

DQN

Premier essai :





10 000 parties

Temps d'exécution total : 4.0 minutes et 40.77377891540527 secondes

Score moyen sur 10000 jeux: 155.4158

Loss moyen sur 10000 jeux: 40.84244918823242

Steps moyen sur 10000 jeux: 11.4171

Reward moyen sur 10000 jeux: 44.49218071428574

Les deux premiers graphiques de notre exploration montrent l'évolution de la perte moyenne par épisode sur 10 000 épisodes et une vue plus détaillée sur les 1 000 premiers épisodes.

Sur 10 000 épisodes La perte commence très haute, ce qui est courant au début de l'apprentissage lorsque l'agent fait beaucoup d'erreurs car il est en pleine exploration. Elle baisse rapidement, ce qui indique un apprentissage rapide. Par la suite, la perte se stabilise avec des pics. Ces pics indiquent que l'agent explore de nouvelles stratégies.

La récompense moyenne commence à un niveau plutôt bas et augmente petit à petit, ce qui indique que l'agent apprend et s'améliore avec le temps. Contrairement à la perte, la

récompense semble avoir plus de variabilité vers la fin, on peut dire qu'il y a toujours des variations significatives dans ses performances.

Sur 1 000 épisodes on observe la même tendance avec une chute rapide de la perte. La perte se stabilise à un niveau bas avec quelques variations. Ces variations peuvent être dues à l'exploration ou à des ajustements de la politique de l'agent.

Le graphique de la récompense (Reward) varie beaucoup au début, mais la tendance générale est à la hausse. Vers la fin de ces 1 000 épisodes, la récompense semble se stabiliser, mais avec une variabilité toujours présente.

Conclusion

En comparant les deux métriques sur deux échelles de temps différentes, on observe un apprentissage initial rapide car il y a la baisse de la perte et l'augmentation de la récompense. Sur le long terme, l'agent semble atteindre un plateau avec moins d'amélioration significative de la performance, comme on peut le voir avec la variabilité persistante dans les deux mesures.

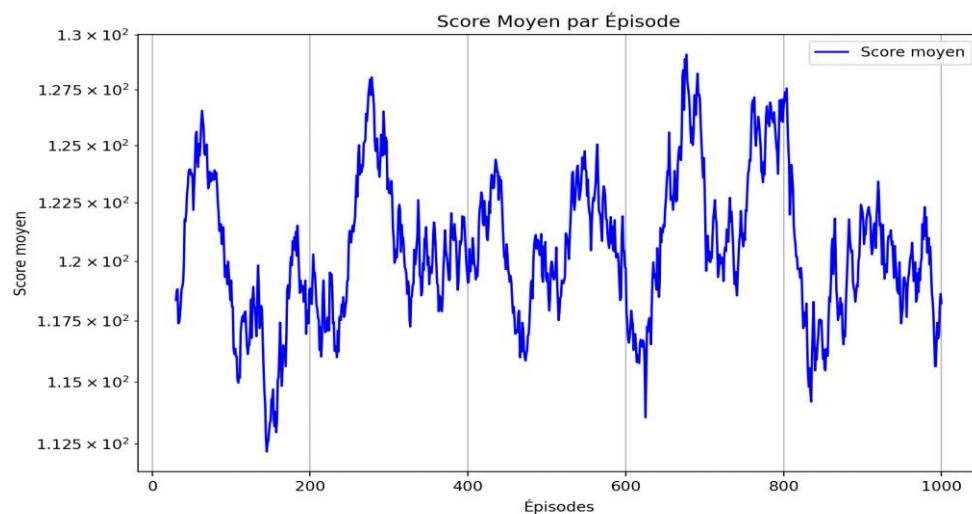
Ce qui n'est pas étonnant car après une période d'apprentissage et une amélioration rapide, l'algorithme entre dans une phase où les gains deviennent plus marginaux et l'exploration peut entraîner une certaine instabilité des performances.

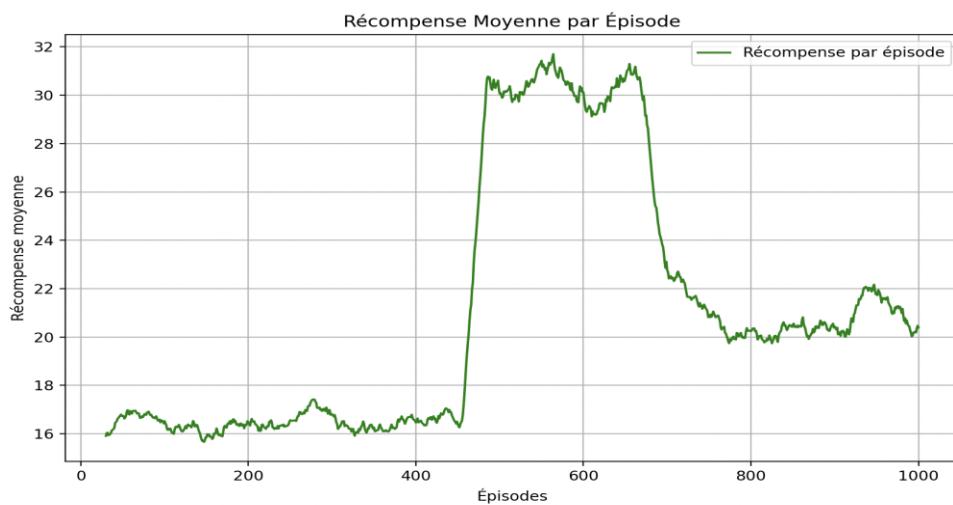
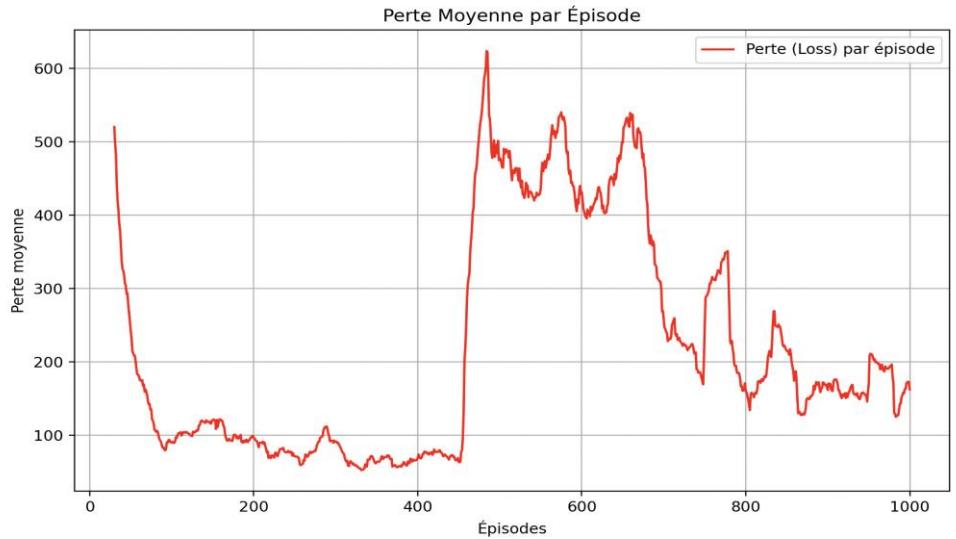
Pour DQN nous avons réussi à dépasser les scores du Random car on tourne autour d'une moyenne de 150

Deuxième essai (version rendu dans le .zip) :

Résultat sur 1000 parties :

Score moyen total par épisodes :





Temps d'exécution total : 37.35105633735657 secondes

Score moyen sur 1000 jeux: 151.819

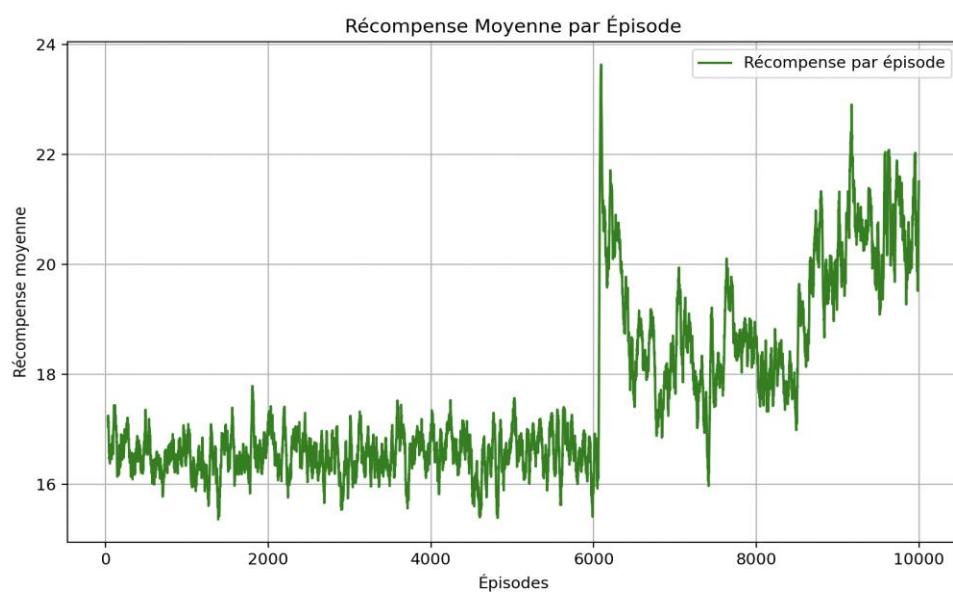
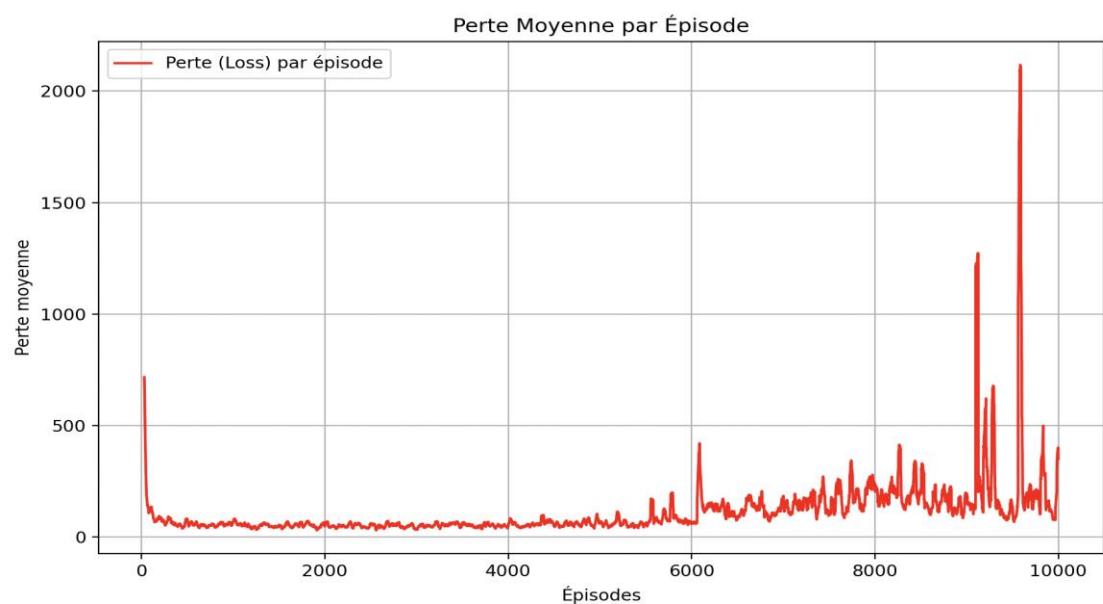
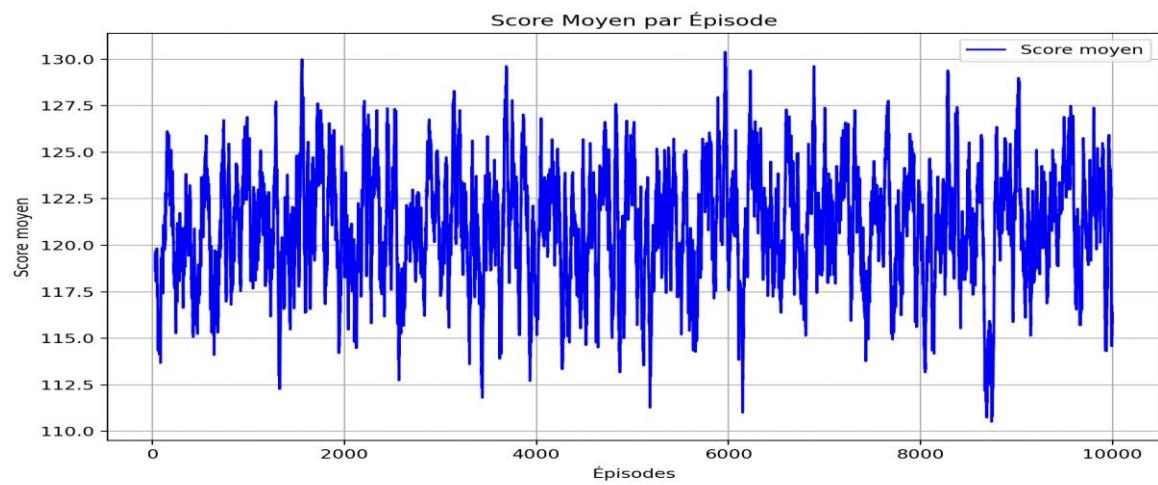
Loss moyen sur 1000 jeux: 130.52627563476562

Steps moyen sur 1000 jeux: 8.874

Reward moyen sur 1000 jeux: 63.20208643578649

On peut voir l'agent explore et ne se stabilise pas totalement encore vers les 1000 premiers épisodes. Le reward donné par épisode augmente, mais cela n'affecte pas le total reward ce qui laisse penser que nous avons mal choisi notre fonction de reward. Nous aurions du donner que le reward de fin en reward positif et donner 0 autrement.

Résultat sur 10000 parties :



Amélioration avec ajustement des hyper paramètres :

DQN Temps d'exécution total : 10.0 minutes et 9.189742803573608 secondes

Score moyen sur 10000 jeux: 120.95

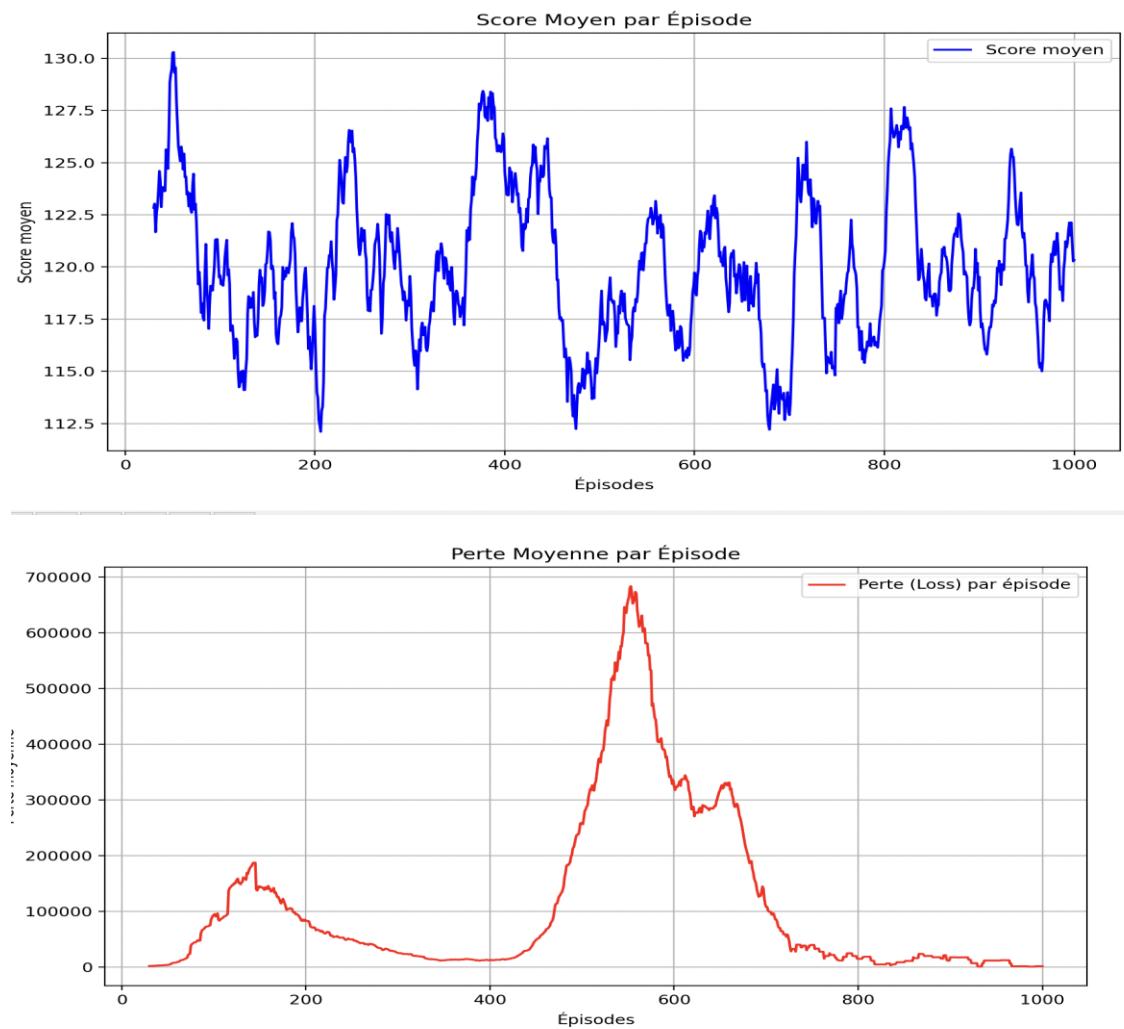
Loss moyenne sur 10000 jeux: 113.88607788085938

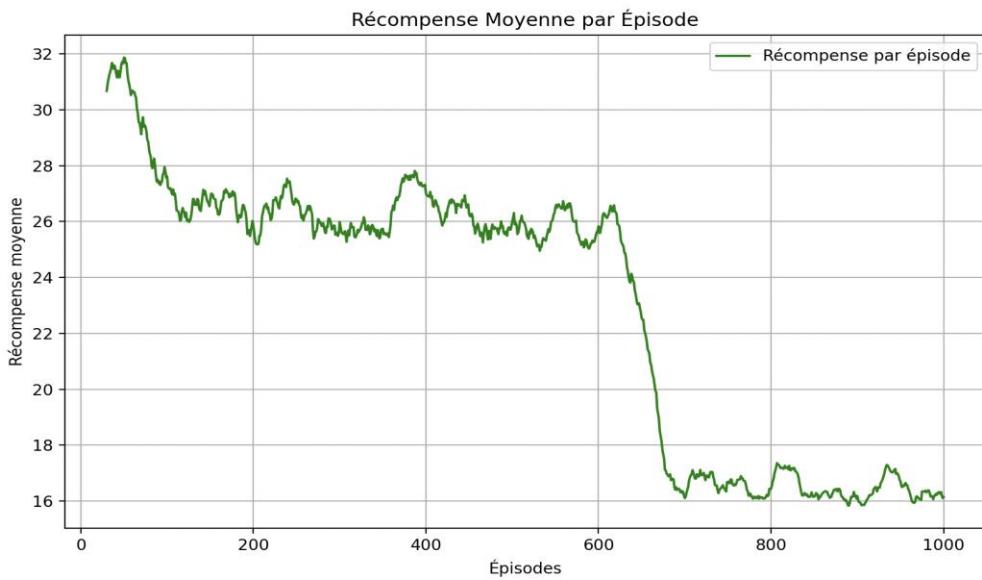
Reward moyen sur 10000 jeux: 17.60402270701519

Sur les 10000 épisodes on peut voir le même problème : le reward par épisode augmente mais cela n'affecte pas le score total (sa moyenne n'augmente pas au fur et à mesure des épisodes).

DDQN

Résultat sur 1000 parties :





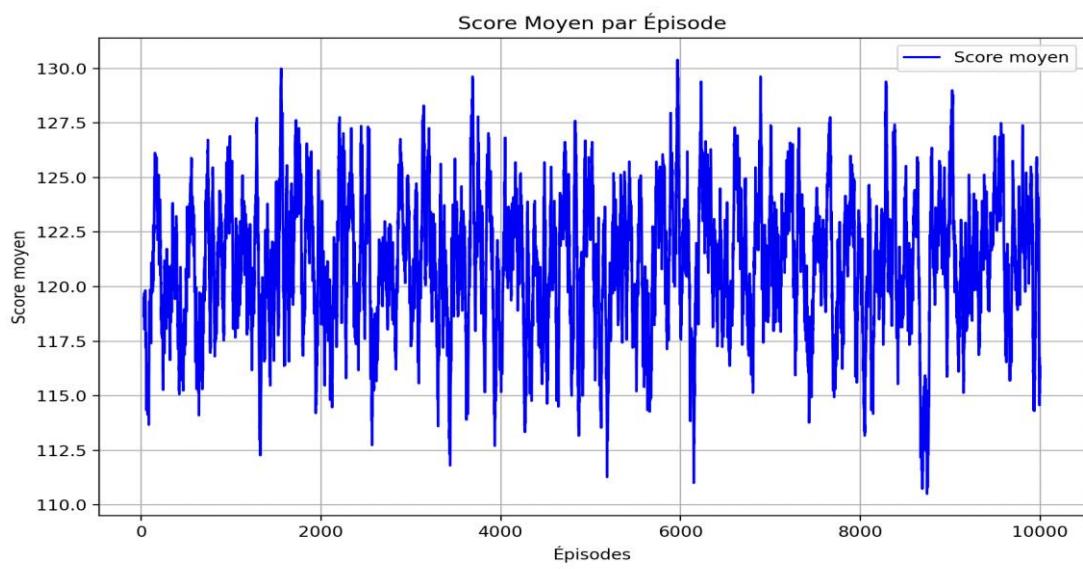
Temps d'exécution total : 1.6851951003074646 minutes

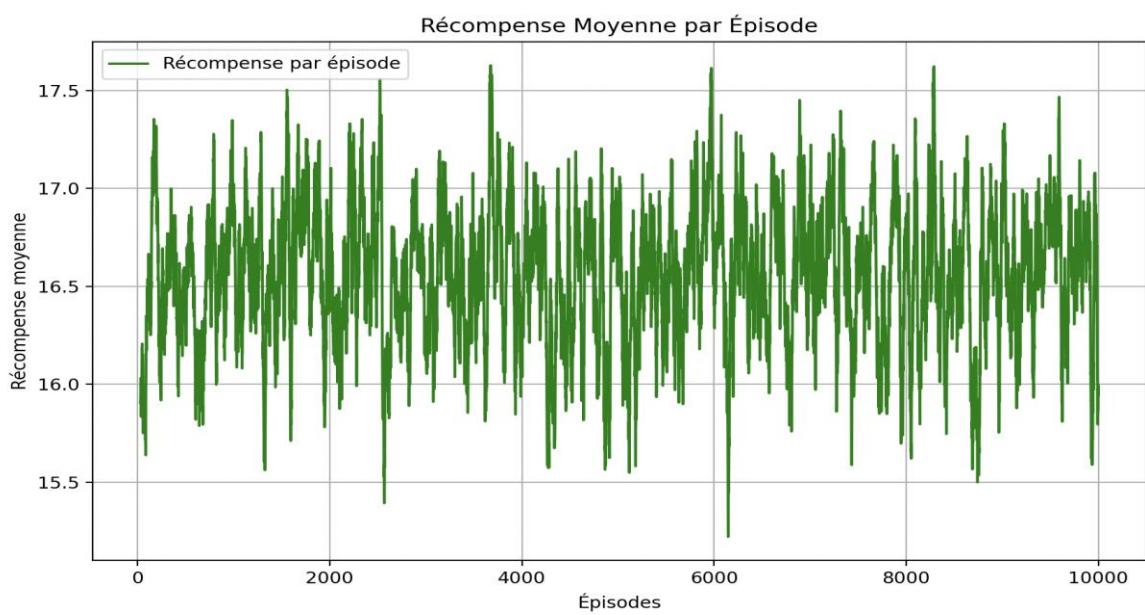
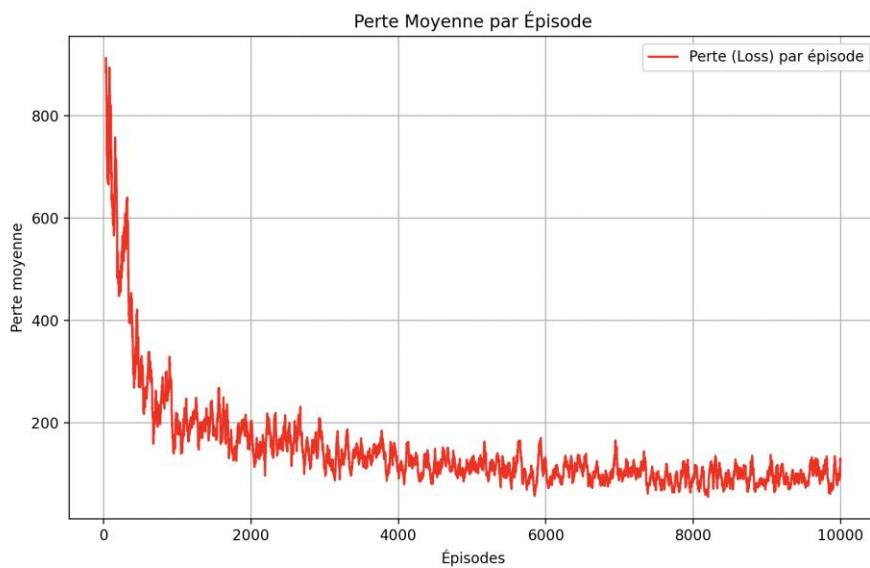
Score moyen sur 1000 jeux: 119.92

Loss moyenne sur 1000 jeux: 108945.078125

Reward moyen sur 1000 jeux: 23.006860841935833

Sur 1000 épisodes, on peut voir que l'agent explore encore et les rewards donnés par épisodes semblent diminuer.





Ici on peut voir que la loss se stabilise sur les 10000 épisodes, mais le reward donné par épisode et le reward total n'augmentent pas.

DDQN

Temps d'exécution total : 19.582812070846558 secondes

Score moyen sur 100 jeux: 158.76

Loss moyen sur 100 jeux: 2148.287353515625

Steps moyen sur 100 jeux: 12.82

Reward moyen sur 100 jeux: 34.144175824175825

Vector d'état final : [1 1 1 2 4 5 6 9 8 7 4 4 31 46 43 36]

Construction Can't Stop

Le jeu "Can't Stop" simule une course de dés entre plusieurs joueurs, où chacun essaie d'atteindre le sommet de trois pistes numérotées en lançant des combinaisons de dés. Chaque piste représente une somme possible de deux dés (de 2 à 12), et les joueurs tentent d'avancer sur ces pistes en fonction des résultats de leurs lancers de dés.

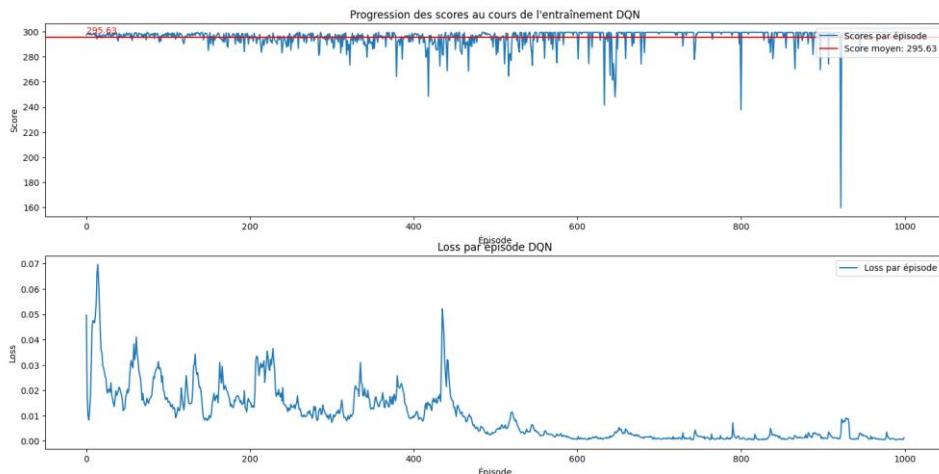
Pour ce jeu nous avons créé un fichier Game dedans il y a la classe Game qui est responsable de la logique principale du jeu. Elle initialise les joueurs, les pistes et gère le déroulement du jeu.

Nous avons défini principalement des définitions pour :

- Initialise le jeu avec un nombre spécifié de joueurs, de pistes et de pistes gagnantes.
- Démarrer le jeu en sélectionnant un joueur aléatoire pour commencer, puis permet à chaque joueur de prendre son tour successivement jusqu'à ce qu'un joueur gagne en atteignant le sommet de trois pistes.
- Gère le tour d'un joueur en lui permettant de lancer des dés, de choisir une combinaison de dés et d'avancer sur les pistes correspondantes.
- Effectue les mouvements pour un joueur automatique (non interactif), choisis aléatoirement parmi les combinaisons de dés disponibles.
- Gère les mouvements pour un joueur humain, en affichant les options de mouvement disponibles et en permettant au joueur de choisir une option.
- Vérifie si le jeu est terminé en vérifiant si un joueur a atteint le sommet de trois pistes.

Résultat Can't Stop

DQN

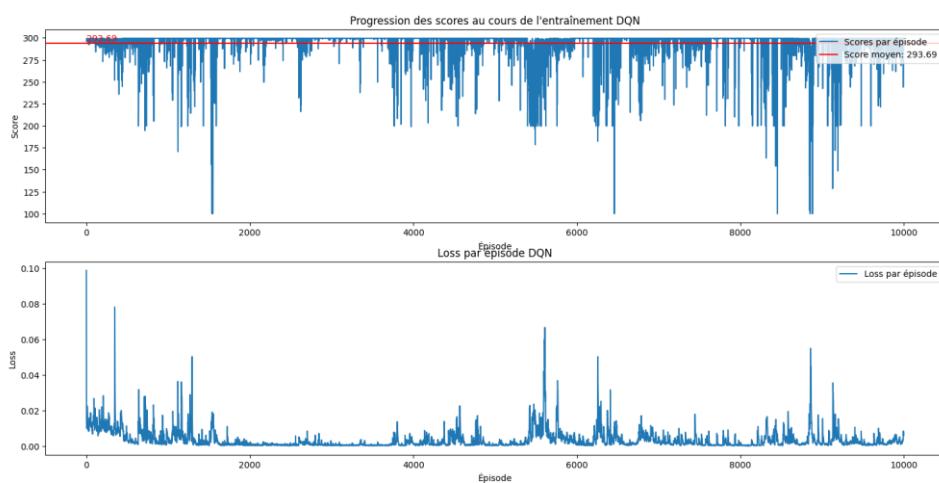


```
ep learning/tictactoe/deep-reinforcement-learning-cantstop-c
Algo Double DQN
Score moyen sur 1000 épisodes: 199.22
Temps d'exécution total pour 1000 épisodes: 628.88 secondes
```

L'algorithme DQN a obtenu un score moyen de 199.22.

Le score moyen est représenté par une ligne rouge horizontale sur le graphique, qui montre une performance constante avec quelques fluctuations tout au long de l'entraînement. La perte par épisode montre une tendance à la baisse rapide au début, qui se stabilise vers la fin.

Le temps total d'exécution pour cette session d'entraînements était de 628.88 secondes.



```
PS C:\Users\louis\OneDrive\Bureau\Master IABD S2\Deep learning\tictactoe & C:/Users/louis/miniconda3/py
ep learning/tictactoe/deep-reinforcement-learning-cantstop-cant_stop/cant_stop/src/DQL.py"
Algo Double DQN
Score moyen sur 10000 épisodes: 293.69
Temps d'exécution total pour 10000 épisodes: 10768.95 secondes
```

Le DQN a atteint un score moyen de 293.69.

Le graphique montre une légère augmentation du score moyen par rapport à l'entraînement de 1 000 épisodes. On peut dire que l'algorithme a continué à apprendre et à s'améliorer sur une période d'entraînement plus longue.

La perte par épisode montre des fluctuations plus importantes au fil du temps, ce qui peut indiquer que l'algorithme explore encore

Le temps d'exécution total était de 10768.95 secondes

Comparaison directe des performances DQN :

L'augmentation du score moyen de 199.22 sur 1 000 épisodes à 293.69 sur 10 000 épisodes montre que le DQN continue d'apprendre et de s'améliorer.

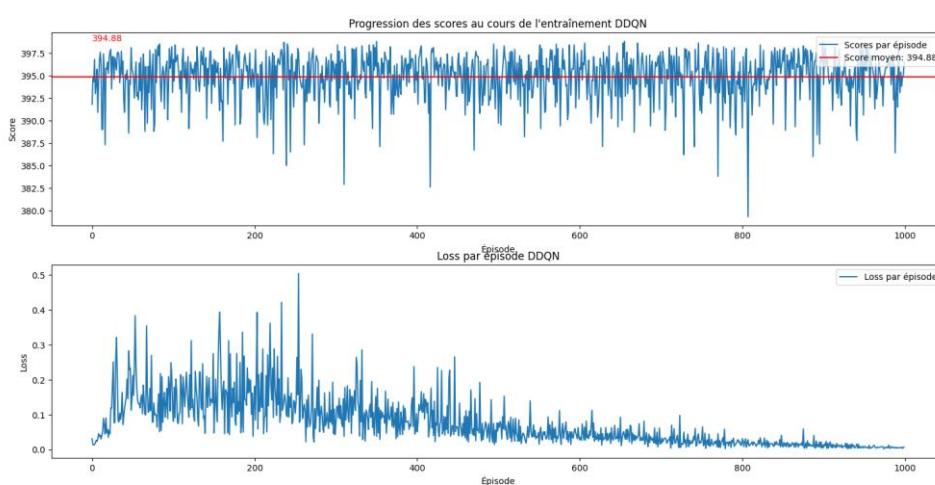
La stabilité de la perte à un faible niveau dans les deux entraînements cela indique que l'algorithme ne s'est pas contenté de mémoriser des stratégies pour un nombre limité de situations mais a développé une politique de jeu générale assez robuste.

La variabilité plus importante de la perte dans la session de 10 000 épisodes pourrait être due à l'exploration.

Conclusion :

L'algorithme DQN a montré une amélioration de performance avec un entraînement plus élevé, ce qui indique une bonne capacité d'apprentissage. L'augmentation progressive du score moyen suggère que l'algorithme continue de s'ajuster et de s'affiner même après un grand nombre d'épisodes.

DDQN.



```

PS C:\Users\louis\OneDrive\Bureau\Master IABD S2\Deep learning\tictactoe & C:/Users/louis/mil
ep learning/tictactoe/deep-reinforcement-learning-cantstop-cant_stop/cant_stop/src/DDQL.py"
Algo Double DQN
Score moyen sur 1000 épisodes: 394.88
Temps d'exécution total pour 1000 épisodes: 2621.87 secondes
PS C:\Users\louis\OneDrive\Bureau\Master IABD S2\Deep learning\tictactoe>

```

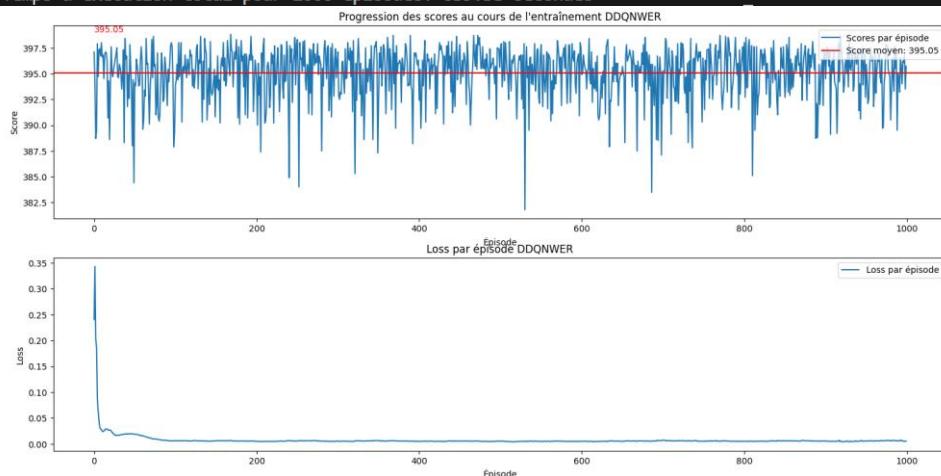
DDQNWER

Sur 1000 épisodes :

```

ep learning/tictactoe/deep-reinforcement-learning-cantstop-cant_stop/cant_stop/src/DDQLWER.py"
Algo Double DQN With Experienced Replay
Score moyen sur 1000 épisodes: 395.05
Temps d'exécution total pour 1000 épisodes: 639.38 secondes

```



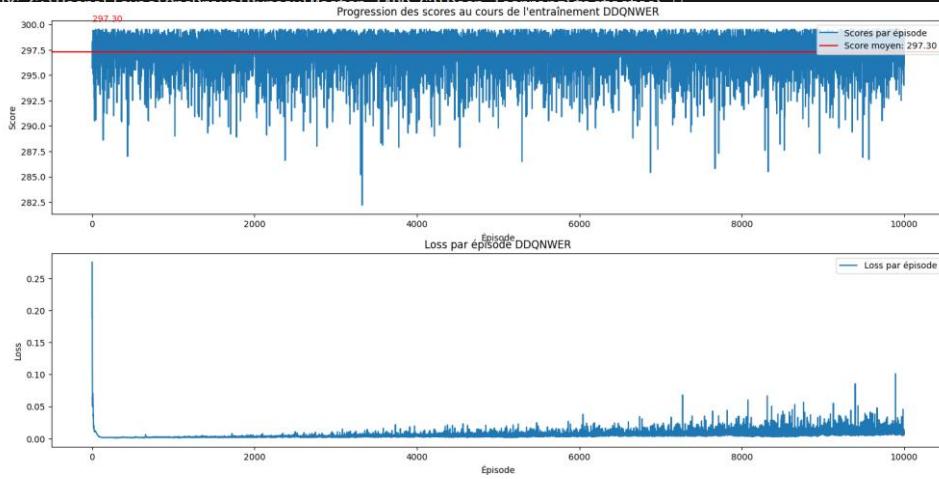
Le score moyen obtenu était de 395.05, comme indiqué par la ligne rouge horizontale sur le graphique des scores par épisode. Ce score indique une performance stable.

La perte (loss) par épisode a rapidement diminué au début de l'entraînement, suggérant que l'algorithme apprenait efficacement les premières stratégies. Au fil des épisodes, la perte s'est stabilisée vers une valeur faible, ce qui implique une convergence de l'algorithme.

Le temps total d'exécution pour cette d'entraînements était de 639.38 secondes.

Sur 10 000

```
PS C:\Users\touss\OneDrive\Bureau\master_IABD_S2\Deep Learning\tictactoe & C:/users/touss/miniconda3/python learning/tictactoe/deep-reinforcement-learning-cantstop-cant_stop/cant_stop/src/DDQNWER.py"
Algo Double DQN With Experienced Replay
Score moyen sur 10000 épisodes: 297.30
Temps d'exécution total pour 10000 épisodes: 8311.99 secondes
```

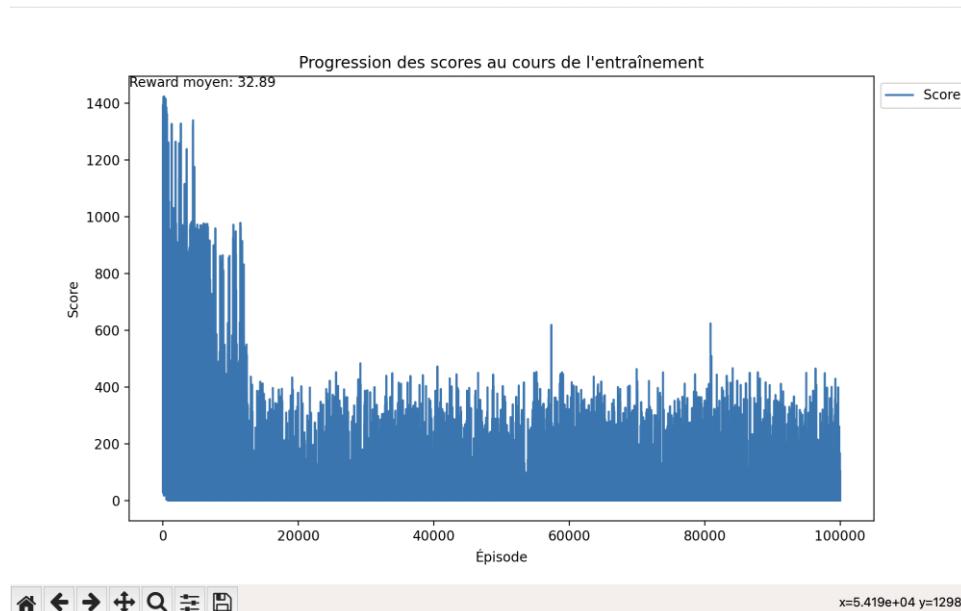


L'algorithme a atteint un score moyen de 297.30. Ce score est inférieur à celui obtenu sur 1 000 épisodes, cela peut être dû à la rencontre de situations de jeu plus complexes au fur et à mesure que l'entraînement progresse.

Comme pour le premier entraînement, la perte par épisode a diminué rapidement, indiquant une phase d'apprentissage efficace. Cependant, la perte a montré des fluctuations plus importantes vers la fin, ce qui pourrait signifier que l'algorithme continuait à apprendre et à ajuster ses stratégies même après un grand nombre d'épisodes.

Le temps d'exécution total est de 8311.99 secondes

Sur 100 000



Comparaison et analyse :

Nous remarquons que La diminution de la perte initiale est presque identique aux deux entraînements, indiquant que l'algorithme s'adapte rapidement au début.

Le score moyen plus élevé sur 1 000 épisodes pourrait être interprété comme une optimisation pour une performance à court terme. Il est possible aussi qu'il y est un coup de chance

Alors que le score moyen plus bas sur 10 000 épisodes pourrait refléter une exploration plus approfondie de l'espace des stratégies.

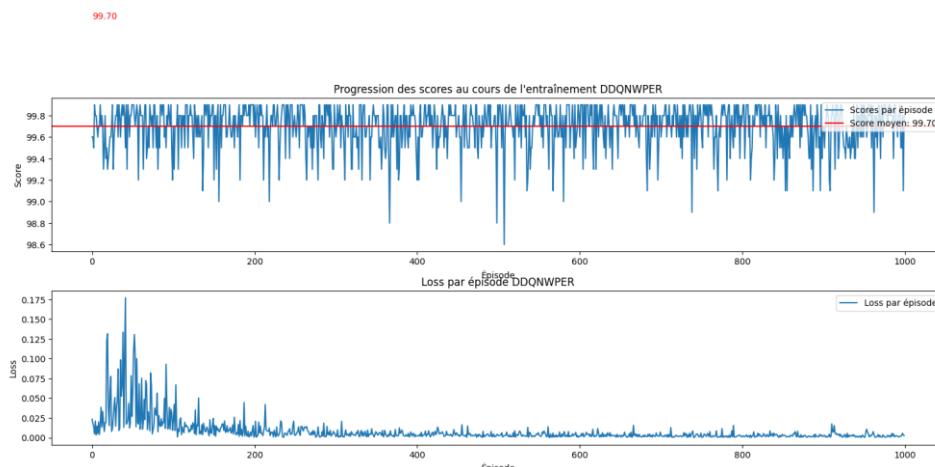
La variance dans les scores par épisode sur 10 000 épisodes est due à une exploration et exploitation.

Conclusion :

L'algorithme Double DQN with Experience Replay a montré qu'il peut apprendre et à s'adapter au jeu "Can't Stop". La performance à court terme semble être meilleure, mais il se peut que nous devions faire une analyse plus approfondie pour déterminer si la baisse du score moyen sur une plus longue période est bénéfique pour l'apprentissage stratégique à long terme ou d'une exploration excessive. Des analyses supplémentaires, comme l'examen des politiques apprises et l'évaluation contre des benchmarks ou des adversaires humains, pourraient fournir un aperçu plus détaillé de la performance de l'algorithme.

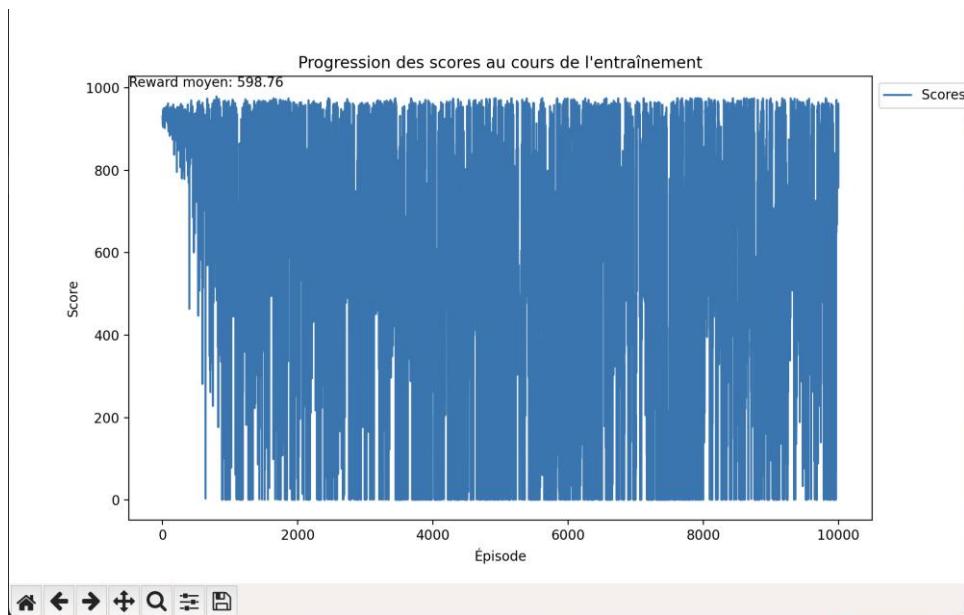
DDQLWPER

Sur 1000

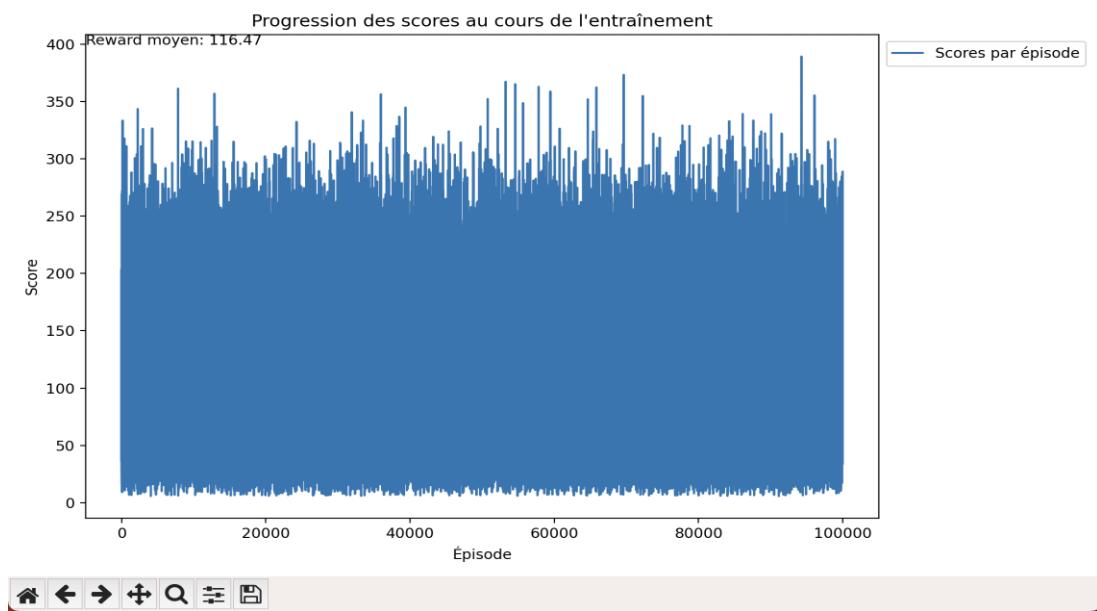


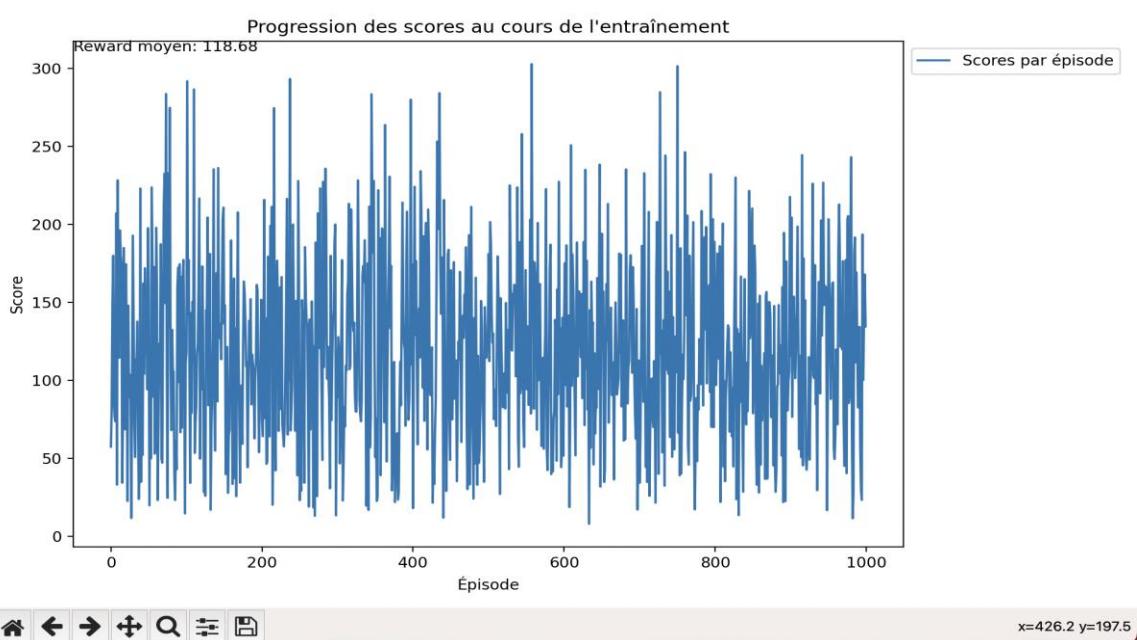
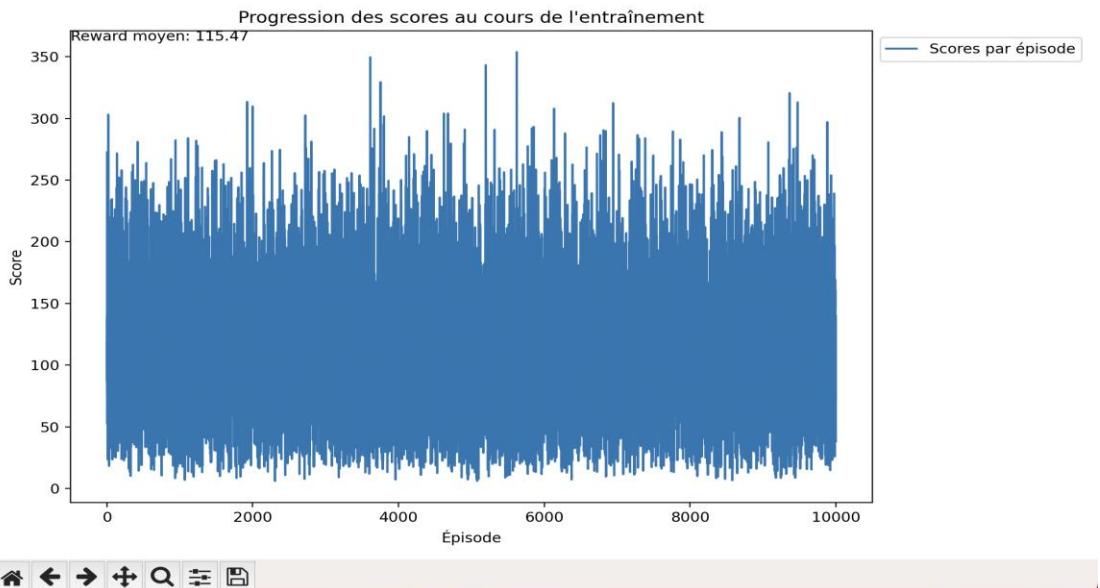
```
deeplearning/tictactoe/deep_reinforcement_learning/cantstop/cant_stop.py
Algo Double DQN With Prioritized Experienced Replay
Score moyen sur 1000 épisodes: 99.70
Temps d'exécution total pour 1000 épisodes: 3021.28 secondes
PS C:\Users\louis\OneDrive\Bureau\Master IABD S2\Deep learning\tictactoe> & C:/Us
deeplearning/tictactoe/deep_reinforcement_learning/cantstop/cant_stop.py
```

10 000



RandomRollout





Annexe

Voir tous les autres résultats dans Images expérimentation dans notre fichier zip