

Deep Reinforcement Learning

Université Claude Bernard Lyon 1

Alexandre DEVILLERS
p1608591

Solayman AYOUBI
p1608583

Décembre 2020

1 Introduction

Dans le cadre de l'UE *Intelligence Bio-Inspirée*, dispensée en *Master 2 IA* à l'*Université Claude Bernard Lyon 1*, il est demandé d'implémenter et d'étudier le *DQN* dans le but de mieux le comprendre.

Cela c'est faire via la réalisation d'un agent capable d'évoluer dans des environnements *Gym* notamment *CartPole-v1*, mais aussi dans des environnements issus de *VizDoomGym* comme *VizdoomBasic-v0*, *VizdoomCorridor-v0*, ou *VizdoomTakeCover-v0*.

2 Architecture

Dossiers et fichiers :

- 'best_networks' contient tout les réseaux de neurones pré entraînés.
- 'best_replays' contient tout les enregistrements *démo* des différents réseaux de neurones.
- 'network' est le dossier où le réseau courant est sauvegardé suite à une exécution.
- 'replay' est le dossier où les replays courants sont sauvegardé suite à une exécution.
- 'agent.py' contient la classe avec les algorithmes liés au DQN.
- 'main.py' contient en haut du fichier tous les hyper-paramètres des entraînements et phases de tests (Si vous voulez le détail des valeurs des hyper-paramètres, et pour la reproductibilité), ainsi que les boucles d'itération sur les environnements.

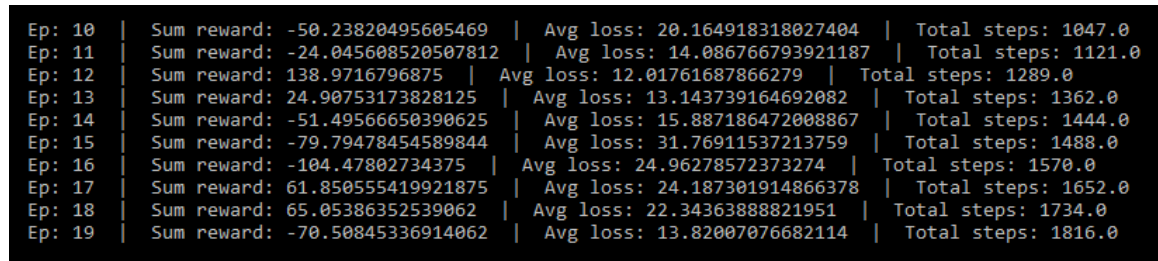
Plus de détail dans le README. Par ailleurs, on vous conseille vivement d'avoir le dossier 'best_replays' ouvert à coté pour lire ce rapport, car les replays s'y trouvant illustrent bien ce qui est dit par la suite.

3 Méthodologie

3.1 Méthode de suivi de l'agent

Afin de suivre l'apprentissage de nos agents, mais aussi de les évaluer, on propose de suivre à chaque épisode la somme des récompenses obtenues durant ledit épisode. On affiche ainsi dans la console à la fin de chaque épisode le numéro de l'épisode, la somme des récompenses obtenues, la valeur moyenne de la fonction de coût, et enfin le nombre total d'étapes effectuées. Un aperçu de la console durant l'exécution est disponible Figure 1. De plus à la fin de tous les épisodes, et donc de l'exécution, on affiche un nuage de points avec comme abscisse le numéro de l'épisode et en ordonnée la somme des récompenses, un aperçu est disponible Figure 2.

Enfin, à l'aide du wrapper *Monitor* nous avons pu enregistrer fréquemment des vidéos de l'environnement et donc de l'évolution de notre agent. Normalement il n'est pas possible de faire ça avec VizDoomGym, cependant nous avons analysé le code source de VizDoomGym et de Gym, puis nous avons fait un fork de VizDoomGym pour ajouter cette fonctionnalité. Par ailleurs, à chaque enregistrement nous sauvegardions les poids de notre modèle afin de pouvoir reproduire ce qui est dans la vidéo (même si la stochasticité empêche une reproductibilité parfaite).



Ep: 10		Sum reward: -50.23820495605469		Avg loss: 20.164918318027404		Total steps: 1047.0
Ep: 11		Sum reward: -24.045608520507812		Avg loss: 14.086766793921187		Total steps: 1121.0
Ep: 12		Sum reward: 138.9716796875		Avg loss: 12.01761687866279		Total steps: 1289.0
Ep: 13		Sum reward: 24.90753173828125		Avg loss: 13.143739164692082		Total steps: 1362.0
Ep: 14		Sum reward: -51.49566650390625		Avg loss: 15.887186472008867		Total steps: 1444.0
Ep: 15		Sum reward: -79.79478454589844		Avg loss: 31.76911537213759		Total steps: 1488.0
Ep: 16		Sum reward: -104.47802734375		Avg loss: 24.96278572373274		Total steps: 1570.0
Ep: 17		Sum reward: 61.850555419921875		Avg loss: 24.187301914866378		Total steps: 1652.0
Ep: 18		Sum reward: 65.05386352539062		Avg loss: 22.34363888821951		Total steps: 1734.0
Ep: 19		Sum reward: -70.50845336914062		Avg loss: 13.82007076682114		Total steps: 1816.0

Figure 1: Aperçu de la console durant l'exécution

Cet affichage dans la console nous permet ainsi de traquer à la volée pendant l'entraînement si l'apprentissage se passe correctement. On notera que comme nous utilisons un γ élève, les valeurs *target* augmentent au début de l'entraînement, or cette augmentation est généralement plus rapide que la minimisation de la fonction de coût, il est donc normal que la valeur de la fonction de coût augmente au lieu de diminuer en début d'entraînement.

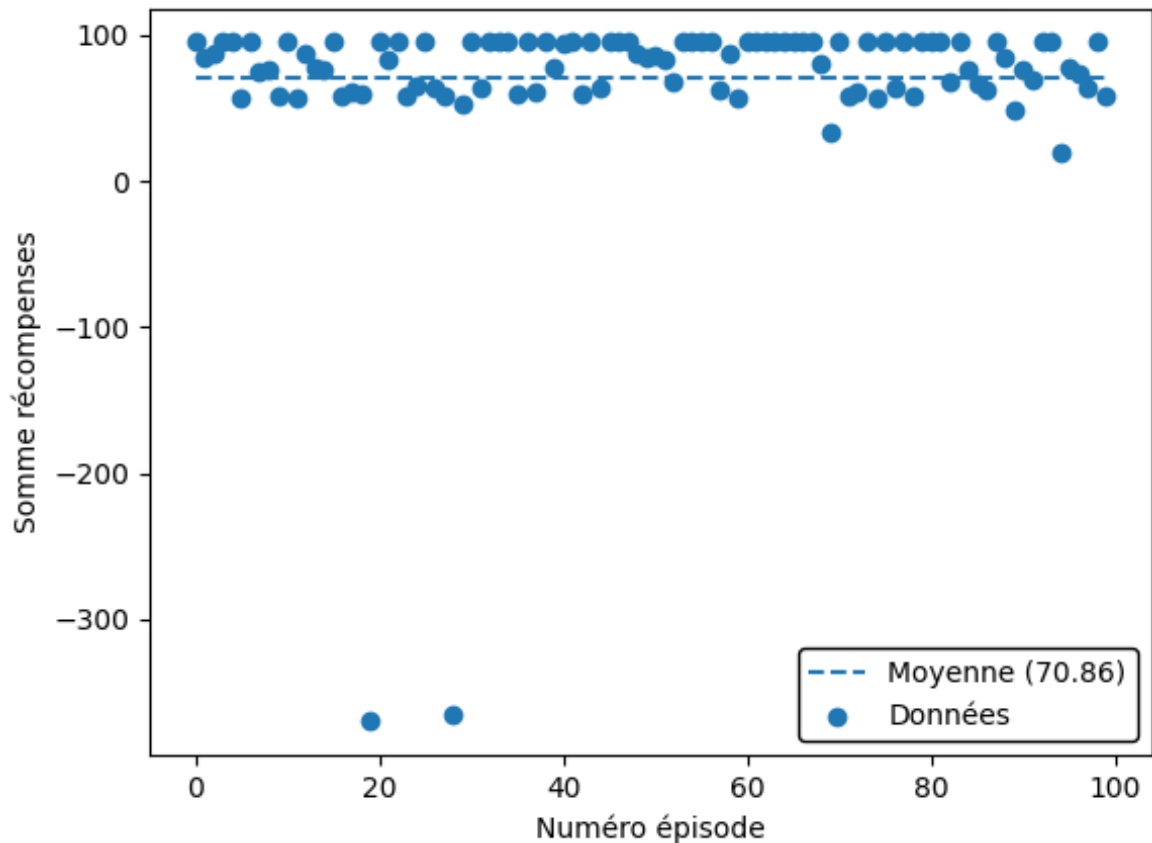


Figure 2: Nuage de point à la fin de l'exécution

Le nuage de points, quant à lui, permet de résumer ce qui a été affiché dans la console tout au long de l'exécution. On peut ainsi rapidement voir les performances de notre modèle et de nos hyper paramètres, mais aussi observer s'il y a eu une progression. On notera que nous avons préféré utiliser un nuage de points plutôt qu'une courbe, au vu des violentes fluctuations qui ont lieu en début d'entraînement.

3.2 Stratégie d'exploration

Afin d'accélérer l'apprentissage, mais aussi de permettre de meilleures performances, nous avons rajouté une stratégie d'exploration. Nous avons essayé les deux méthodes présentées dans le sujet, et c'est avec une version légèrement modifiée de ϵ -greedy que nous avons eu les meilleures performances.

Pour l'exploration de Boltzmann, bien que plus intuitive, nous avons été confrontés à un problème. Comme dit précédemment, c'est avec une valeur élevée de γ (0.999), que nous avons eu les meilleurs résultats. Cela a pour effet de faire converger les Q-Valeurs vers des valeurs relativement élevées, notamment pour CartPole-v0 qui offre une récompense à 1 pour chaque étape non finale. Ainsi avec des Q-Valeurs élevées, il arrive que l'exponentielle présente dans la formule de l'exploration de Boltzmann retourne une valeur *infini*, ce qui fait crash le programme.

Pour la méthode ϵ -greedy, nous étions confrontés à un autre problème, l'apprentissage était meilleur avec un ϵ élevé, mais ce dernier était tellement élevé qu'il faisait des fois échouées notre agent dans des situations qu'il maîtrisait. Nous avons donc modifié cette méthode en fixant une valeur de diminution de ϵ , on notera $\epsilon_{decay} \in]0, 1]$ cette valeur. Ensuite à chaque itération on note $\epsilon = \epsilon \times \epsilon_{decay}$, ainsi epsilon peut être grand au début, il tendra vers 0 au cours des itérations. De plus nous avons défini une valeur minimum pour ϵ , qui permet si on le souhaite, de conserver une légère stochasticité et de ne jamais avoir $\epsilon = 0$.

Pour finir, on propose de désactiver la stochasticité, ce qui peut être utile lors d'une phase d'évaluation ou de démonstration. Cela permet à notre agent de ne suivre que les actions qu'il estime comme meilleures, et de ne pas échouer à cause du suite d'aléa *malchanceux*.

3.3 Mise à jour de target

Afin de stabiliser l'apprentissage, on utilise un réseau *target*, censé être un duplicata de notre réseau. Pour mettre à jour notre duplicata, nous avons essayé les deux méthodes présentées dans le sujet, on nommera 'hard' la méthode recopiant les poids toutes les N itérations, et 'soft' celle utilisant une valeur alpha.

Ces deux méthodes ont donné des résultats relativement similaires, et à cause de la stochasticité nous n'avons pas réussi à exhiber une méthode à privilégier.

3.4 Réseau de neurones

Les architectures des réseaux de neurones utilisés sont disponibles Figure 3, pour celui de CartPole-v1, notre inspiration est venue du TP de IBI 1. Pour celui utilisé pour VizDoom-Gym, l'inspiration vient du tutoriel PyTorch DQN. On note que pour VizDoomGym les '_' sont là car ces dimensions dépendent de la taille de l'entrée, or on utilise différentes taille selon les environnement (plus de détail par la suite) :

- VizdoomBasic-v0 : [1, 120, 160]
- VizdoomCorridor-v0 / VizdoomTakeCover-v0 : [4, 60, 80]

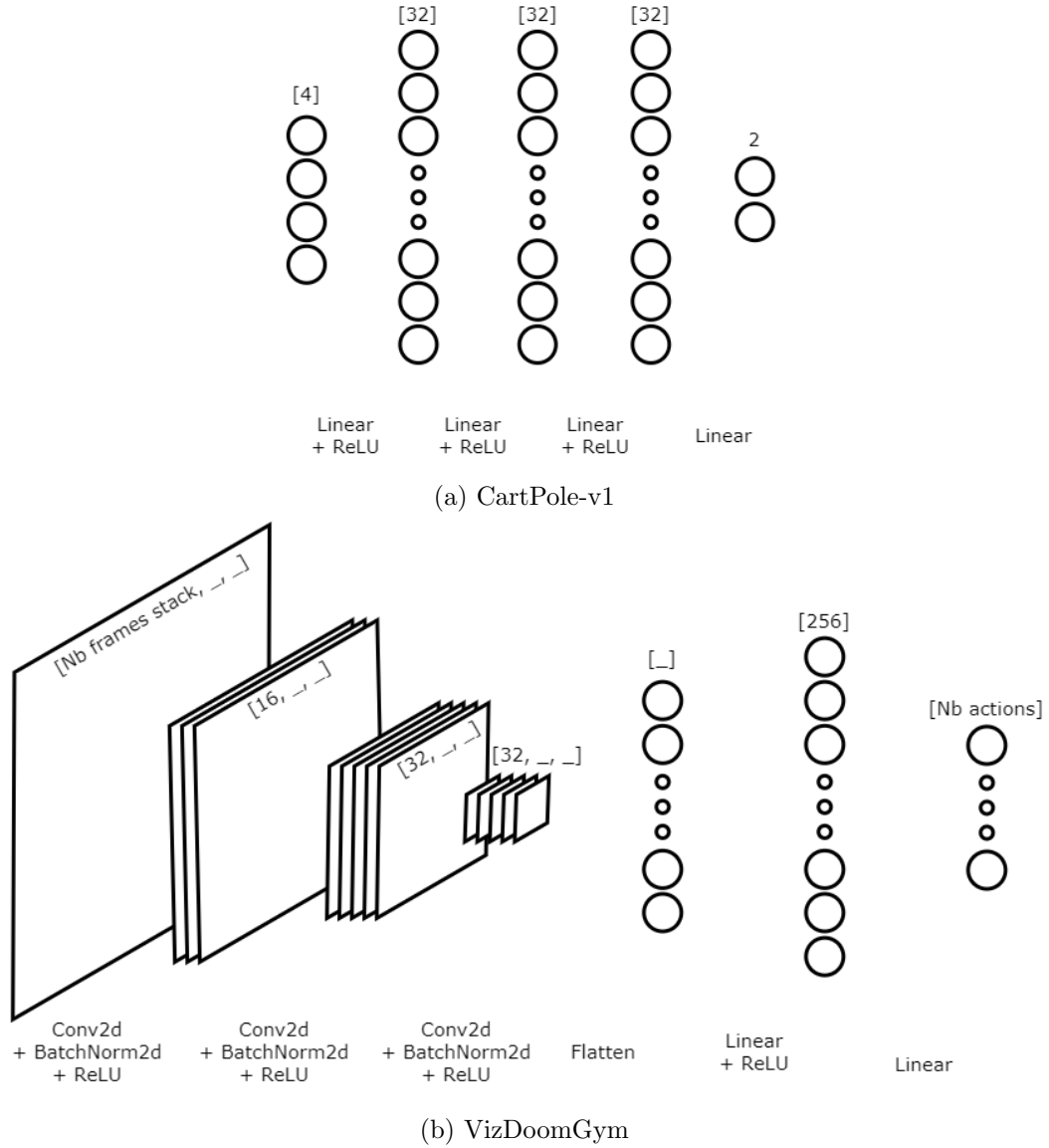


Figure 3: Nos différents réseaux de neurones

3.5 Frame stacking

Visiblement VizDoomGym est capricieux pour être l'utiliser avec les wrappers, nous n'avons donc pas pu utiliser le wrapper FrameStack pour faire la *frame stacking*, cependant cela ne nous a pas arrêtés, car nous avons ré implémenté l'équivalent du *frame stacking* à la main.

Dans un premier temps, on notera que avec du *frame stacking* les calculs sont plus long, ce qui nous a forcés à réduire taille de batch et taille d'image. Nous avons utilisé le *frame stacking*, avec 4 frames, pour les environnements VizdoomCorridor-v0 et VizdoomTakeCover-v0.

3.6 Dernière étape

Dans CartPole-v1 les récompenses sont toujours de 1 chaque tick, mais au bout de 500 ticks l'environnement ferme. On ajoute donc une expérience qui sera considérée comme mauvaise comparée au reste (car reward de 1, au lieu de plus grâce à γ) alors qu'on vient de faire le meilleur score. Pour éviter ça on ajoute pas l'interaction au buffer dans ce cas très spécifique.

3.7 Cuda

Comme dit précédemment, certaines fois les calculs étaient longs, nous avons donc décidé d'installer cuda, et de faire un script modulable, capable de s'exécuter sur CPU ou sur GPU. Nous n'avons pas quantifié formellement l'accélération procurée par l'utilisation de cuda, mais on passe avec VizDoomGym d'un affichage image par image à un affichage fluide.

3.8 Set d'hyper paramètres

Au début, nous n'étions pas très organisés pour la recherche de bonnes combinaisons d'hyper paramètres. On avançait à tâtons, et on ne sauvegardait pas toujours les résultats obtenus pour un certain jeu d'hyper paramètres. Sans surprise les résultats n'étaient pas présents.

On s'est donc rapidement tourné vers une méthode plus *logique*, on a cherché sur internet des exemples d'implémentations, des articles, et des tutoriels... Ce qui nous a permis d'avoir une idée des bornes pour la plupart des hyper paramètres, suite à ça on a rapidement pu avoir des débuts de résultats.

Comme nous avons eu à manipuler plusieurs environnements, nous devons sauvegarder les jeux d'hyper paramètres associés à chaque environnement ou expérience. Pour ce faire on a utilisé une méthode simple, en haut de notre fichier principal, se trouve un ensemble de jeu d'hyper paramètres (PARAM_SET), et un autre paramètre (USED_SET) qui permet de sélectionner le jeu d'hyper paramètres à utiliser. Ainsi en définissant USED_SET à 'VizdoomBasic-v0_Test' on lance un test sur VizdoomBasic-v0, avec un réseau pré entraîné sauvegardé au préalable.

Cette dernière méthodologie nous a permis d'alterner rapidement entre nos différentes expérimentations, de passer rapidement d'un entraînement à une phase de test, ou bien de changer d'environnement.

4 Résultats

4.1 CartPole-v1

Premièrement, CartPole-v1 est un environnement sur lequel nous avons passé beaucoup de temps à chercher des bons hyper-paramètres, comme dit précédemment nous n'avions pas une méthode efficace pour chercher les bons hyper-paramètres au début. Mais ce n'est pas la seule raison pour laquelle nous avons passé beaucoup de temps à affiner nos paramètres sur cet environnement, en effet lors de nos recherches on a pu apprendre que pour considérer CartPole-v1 comme résolu il fallait une moyenne supérieure à 475, et à partir de ce moment-là c'est devenu notre objectif principal. Objectif que nous sommes fiers d'avoir atteint, car

après un grand nombre d'étapes d'entraînement, comme on peut le voir Figure 4 notre agent ne fait que des scores maximums, c'est-à-dire 500.

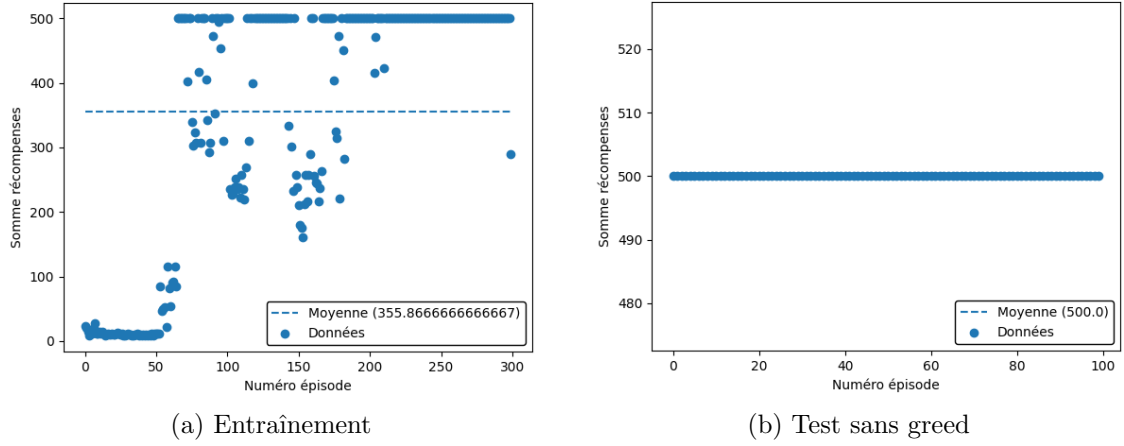


Figure 4: Nuage de point issus d'entraînements et de phases de tests

4.2 VizdoomBasic-v0

Premièrement, on tient à dire que pour le réseau à convolution, on s'est fortement inspiré de celui exposé dans le tuto PyTorch sur le DQN.

Dès le début nous avons eu des résultats corrects, mais ce n'était pas assez à nos yeux. Faire varier les hyper paramètres ne donnait pas d'améliorations, on s'est donc penché sur le *pre-processing* et sur le réseau de neurones. Pour la partie *pre-processing* on s'est rendu compte que les images étaient *déformées*, en faite les images sont au format $[h, w, c]$, et le `resize` attend une résolution au format $[h, w]$, ce qui fait que la résolution donnée dans le sujet produisait une image difforme. On a donc changé la résolution pour conserver le ratio et au passage augmenté légèrement la taille des images, ce qui nous donne $[120, 160]$. On observe ainsi Figure 5 et Figure 7 les différences entre avant et après ces améliorations. La différence est flagrante, le nombre de cas où le monstre n'est pas tué (score dans les -300) est grandement réduit grâce aux améliorations.

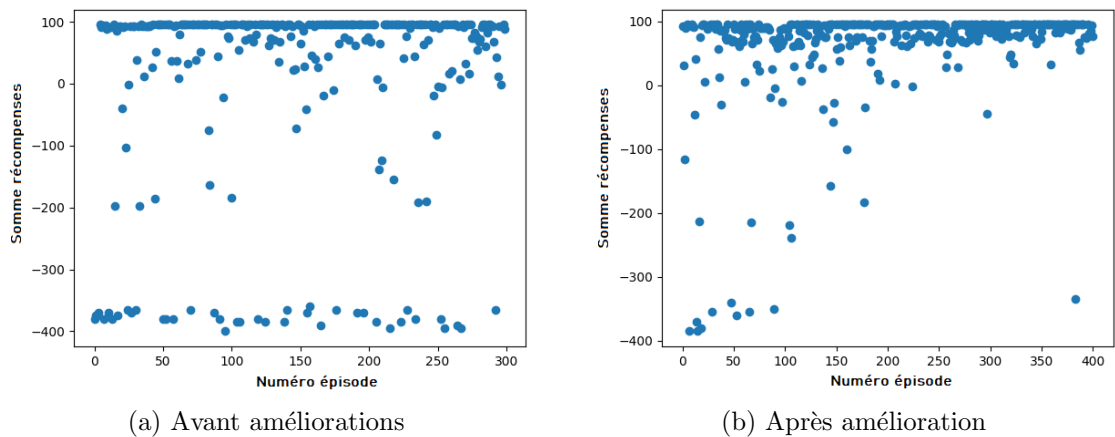
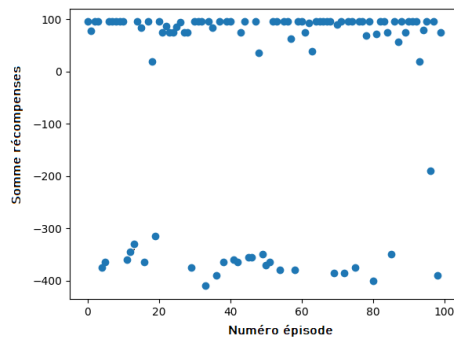


Figure 5: Nuage de point issu d'entraînements

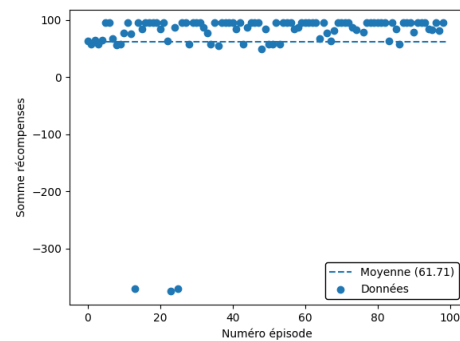
Durant les phases de tests, dont les résultats sont Figure 7, on remarque que notre agent performe très bien. En fait il n'y a qu'un seul cas dans lequel notre agent échoue, c'est quand le monstre est tout à gauche et qu'il est tourné, ce qui arrive très rarement. Une image du monstre tourné est disponible Figure 6, les états dans lesquels le monstre est tourné étant rares cela explique la faiblesse de notre agent face à cette situation.



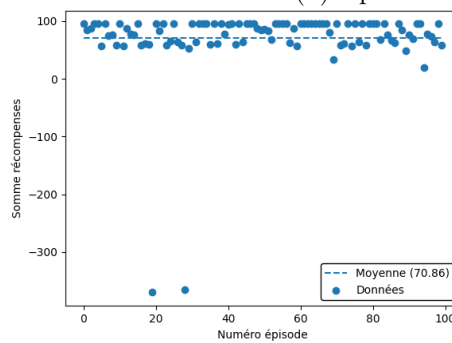
Figure 6: Monstre normal, et tourné



(a) Avant améliorations



(b) Après amélioration sans greed



(c) Après amélioration avec greed

Figure 7: Nuage de point issus de phases de tests

4.3 VizdoomCorridor-v0

Cet environnement est très difficile, les seules récompenses qu’il y a sont positives quand on se rapproche du bout du couloir, ou négative si on s’en éloigne, et il n’y a aucune récompense quand on tue un monstre. Or il est impossible d’avancer sans tuer les monstres, qui en plus nous tirent dessus dès le début.

Dans un premier temps, sans *frame stacking*, nous avons réussi à avoir un agent qui courrait vaguement tout droit, mais qui mourrait dès le début. On est directement passé au *frame stacking*, ce qui a grandement amélioré les choses, cependant nous ne sommes pas du tout satisfaits des résultats. Ce qu’il y a de frustrant c’est qu’on observe différents comportements selon les entraînements (et ce pour les mêmes hyper-paramètres), des fois l’agent court tout droit, des fois il tue le premier monstre de gauche, et des fois celui de droite. Mais quand il commence à converger vers un de ces comportements, il ne changera plus, nous n’avons eu qu’une seule fois le cas où l’agent tue les deux monstres mais il n’avançait jamais après. Alors que nous avons essayé un grand nombre de combinaison d’hyper-paramètres, et passé l’équivalent de plus d’une journée à faire chauffer la carte graphique. Outre les tentatives de changement d’hyper-paramètres, nous avons aussi tenté de relancer des entraînements avec des réseaux pré entraînés, en espérant qu’il y aurait conservation du comportement pré appris, et grâce au greed, découverte d’un deuxième.

On observe donc Figure 8, les résultats obtenus, avec comme meilleur agent celui qui court tout droit combiné à de l’aléatoire, car cela lui permet d’éviter des balles avec un peu de chance.

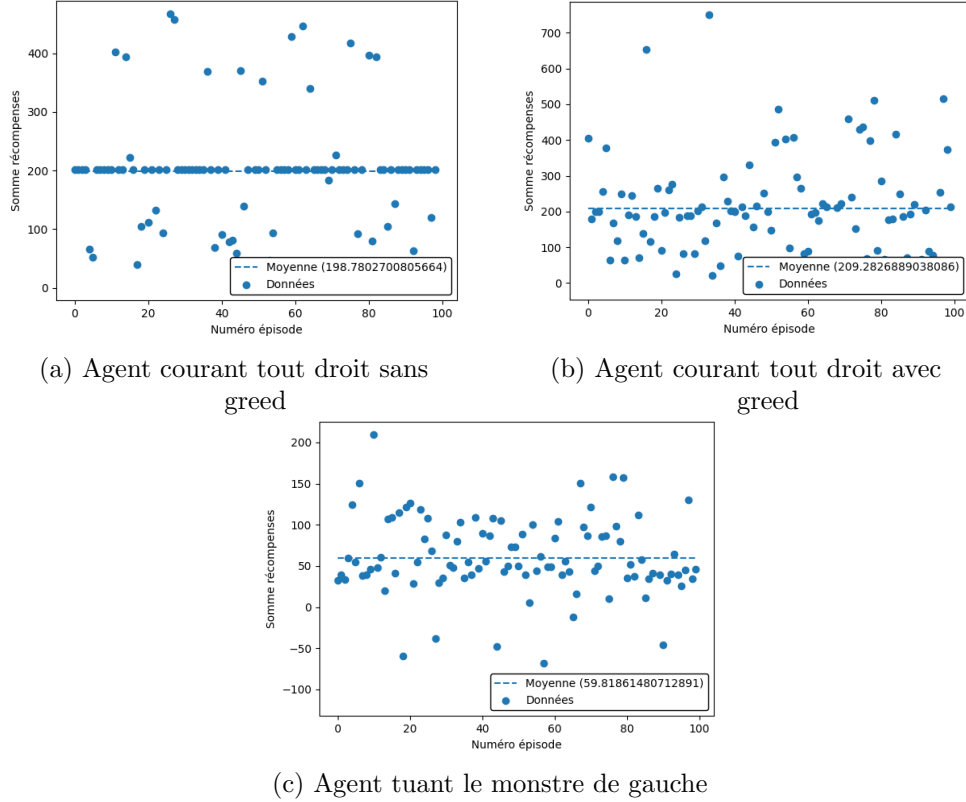


Figure 8: Nuage de point issus de phases de tests

4.4 VizdoomTakeCover-v0

Cet environnement est intéressant, la difficulté est incrémentale vu que le nombre de monstres augmente avec le temps.

Sans *frame stacking* les agents qu'on entraînait se contentaient d'aller soit tout à droite, soit tout à gauche. Cependant nous nous attendions à ce résultat, il semble compliqué de prédire la trajectoire d'un objet avec juste une image, car cela nous donne uniquement une position et donc il nous manque la vitesse. C'est pourquoi une fois de plus, nous avons utilisé du *frame stacking*, dans l'espoir que le réseau de neurones compare sur plusieurs frames consécutifs la position des boules de feu et ainsi connaître leurs vitesses, et donc leurs trajectoires.

Cela s'est avéré relativement efficace comme le montre la Figure 9. Cependant quand le nombre de monstres augmente, et qu'il y a plusieurs boules de feu à l'écran, ça devient plus compliqué pour l'agent. De plus il y a certains cas dont il semble impossible de s'en sortir, notamment quand des projectiles arrivent en même temps de chaque côté, ou que l'agent est collé au mur.

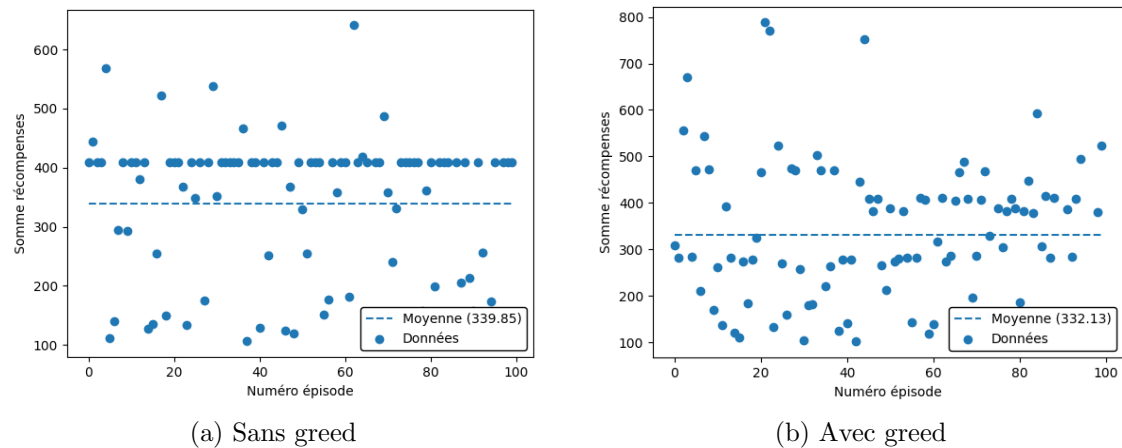


Figure 9: Nuage de point issu de phases de tests

5 Conclusion

En conclusion nous n'avions pas de connaissances sur le DQN avant ce cours, et nous avons donc appris énormément de choses. De plus ça nous a ouvert à l'utilisation de Gym, et il n'est pas impossible qu'on continue même après ce TP à expérimenter sur d'autres environnements. Le temps d'exécution était un peu long, ce qui nous a empêché de faire toutes les expérimentations qu'on aurait voulues, cependant ça nous a poussé à chercher les hyper-paramètres de façon plus intelligente. Nous sommes fiers des résultats obtenus, et nous avons trouvé une satisfaction immense à voir notre agent apprendre en direct que ça soit dans CartPole ou dans VizDoom.