Grégorie Méné – 2SN Logiciels

Rapport personne d'apprentissage par renforcement

Problématique : Comment faire converger la fonction Q de manière efficace et précise ?

Sommaire

Introduction

Tout au long de cette dernière partie, mon but a été de faire converger la fonction Q le plus vite possible et le plus précisément possible, afin d'obtenir de bons résultats à l'entrainement.

J'ai donc tout d'abord réfléchi à différents réseaux de neurones possibles, puis je me suis attardé sur l'importance des différentes récompenses possibles en fonction du « mode » de jeu (classique, survivaliste, agressive ou autre), puis j'ai finalement opté pour l'implémentation d'une mémoire et l'introduction de mini-batch sur lesquels la fonction s'appuierait pour converger plus efficacement.

L'étude des différentes stratégies m'a amené à choisir la classique. En effet une stratégie défensive n'est pas un choix optimal. Le nombre de coups augmentera mais sans pour autant donner l'avantage à notre IA. Cette dernière cherchera simplement à ne pas mourir (elle ne cherchera donc pas forcément à gagner mais simplement à faire des mouvements loin de la deuxième IA, en évitant le conflit). La méthode agressive n'est pas non plus un choix pertinent : avec -1 pour la survie, l'IA va finir par chercher à mourir et donc limite se suicider. On pourrait mettre une récompense très haute mais cela fausse significativement les entraînements.

I- Performances du code donné et idées de protocole

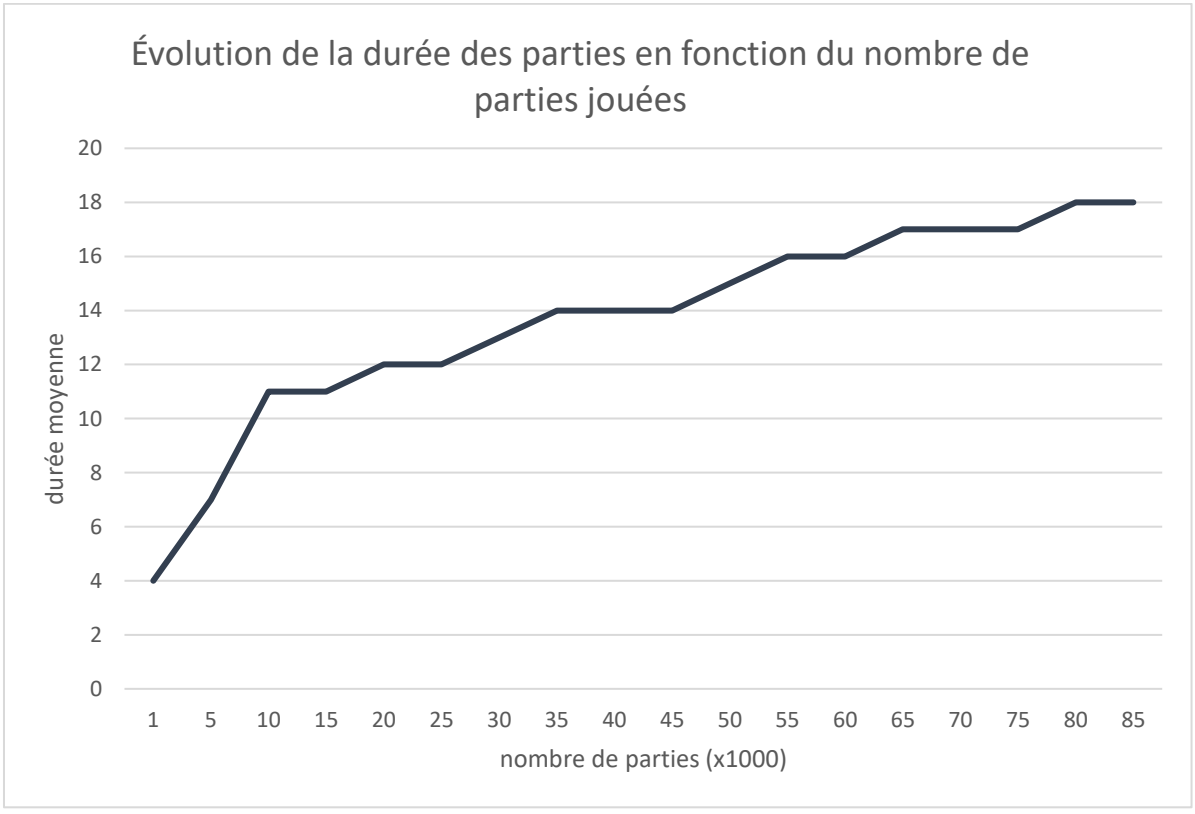
a. Explications et mesures.

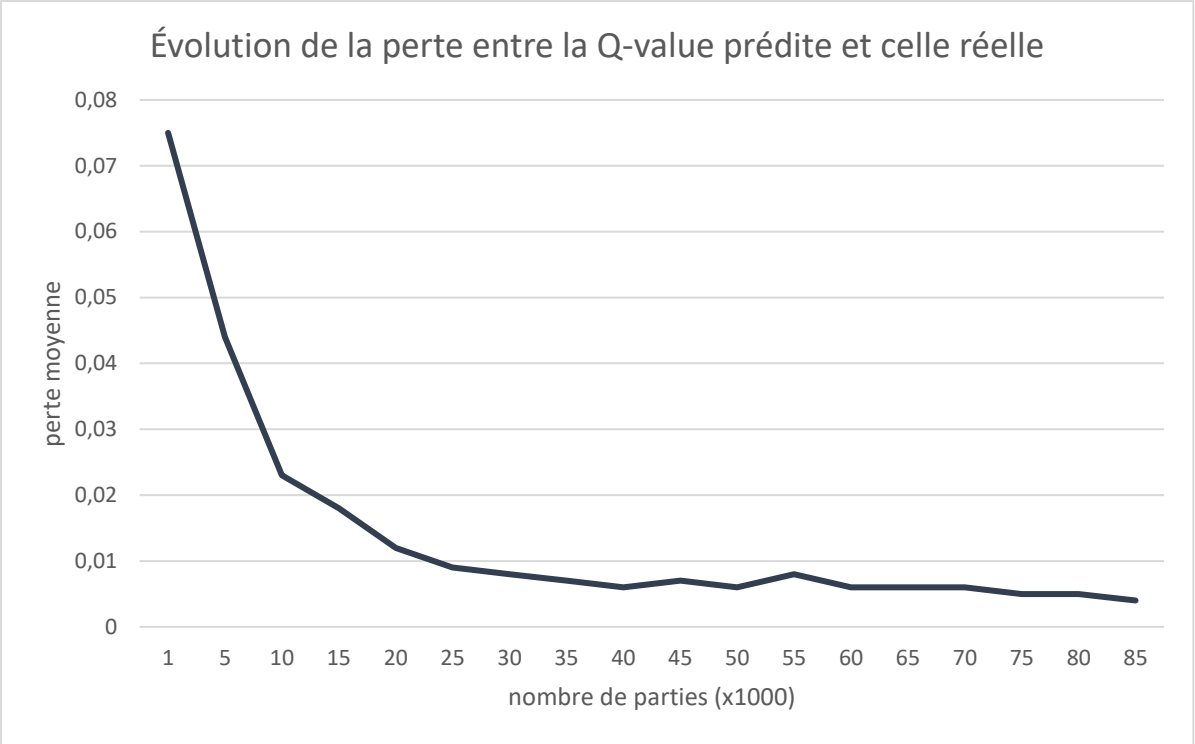
A l'aide du squelette, j'ai pu rapidement mettre en place une idée de protocole se basant sur une mémoire des actions, états et récompenses.

J'ai tout d'abord implémenté le réseau de neurones. J'ai simplement mis 2 couches de convolution et 3 couches fully-connected. Cela donne des résultats assez intéressant (voir partie b.).

Pour ce qui est de la partie entrainement, j'ai simplement calculé une moyenne de ***game_duration*** et de ***loss*** afin d'afficher à l'écran des statistiques toutes les 5000 parties. Les 2 IAs jouent donc l'une contre l'autre et Q converge plus ou moins rapidement.

b. Résultats





On voit clairement que le nombre de coups augmente assez régulièrement au fil des parties jouées et que Q converge vers sa valeur prédite de plus en plus (surtout au début où *loss* décroit très rapidement).

II- Adoption et mise en place d'une méthodologie

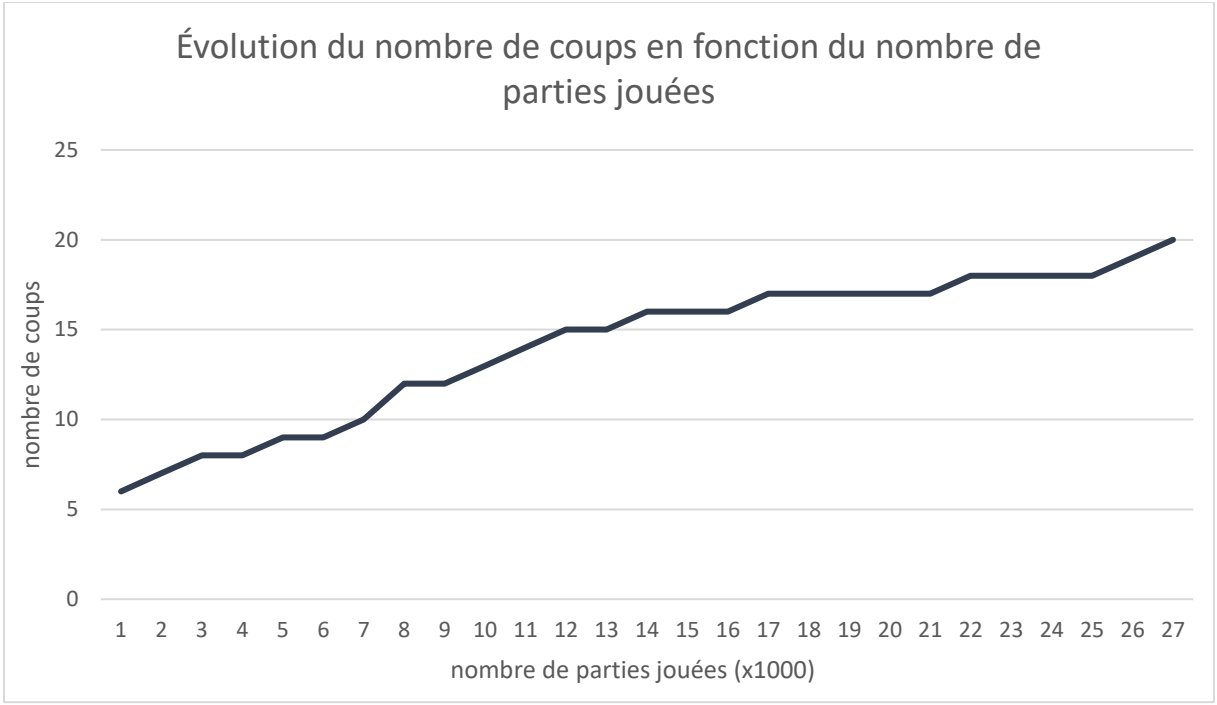
a. Explications et mesures

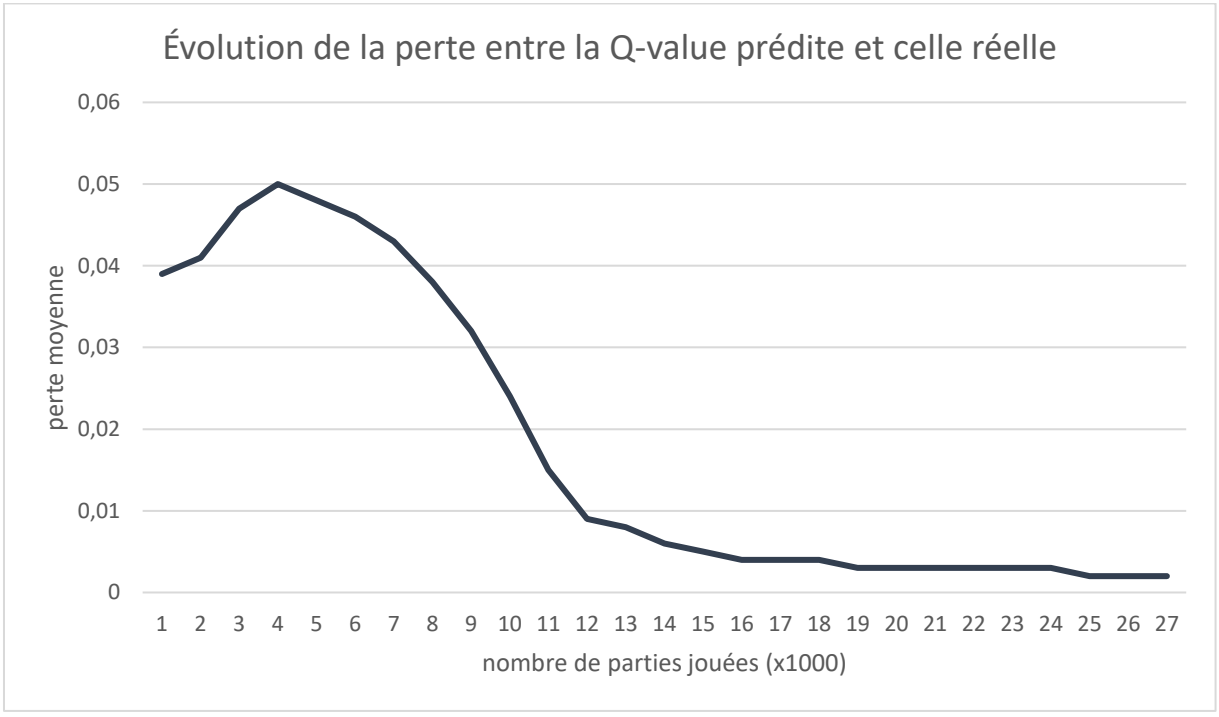
Mon idée a donc été de faire comme pour la base de donnée d'images et de travailler sur des mini-batch. J'ai donc sorti de la boucle *while* les trois tableaux *states*, *actions* et *rewards* afin d'y avoir accès même après une partie jouée.

J'ai donné une taille maximum à ma mémoire : 10 000, et j'ai considéré des mini-batch de taille 100. Je stocke donc dans cette mémoire la même chose que dans le squelette sauf que quand j'atteins la taille maximum de la mémoire, je remplace les premiers éléments des tableaux à l'aide d'un modulo (voir code).

Je tire au hasard 100 indices dans tous les indices possibles de la mémoire et j'applique la formule avec le **discount factor** et la **predicted_q_value** à tout mon mini batch. Ainsi, au lieu de travailler uniquement sur une partie, notre IA s'entraîne sur 100 mouvements de la mémoire. Q converge ainsi plus rapidement (puisque s'appuie sur beaucoup plus de données) malgré un temps d'exécution légèrement plus lent (boucle for plus importante).

b. Résultats





On remarque directement qu'en à peine 25 000 parties jouées on obtient le même nombre de coups que pour 85 000 parties dans l'autre entrainement. La convergence est bien plus rapide.

III- Discussions

Je n'ai pas eu le temps de vraiment considérer toutes les options et méthodes possibles pour améliorer l'apprentissage de mon IA. J'ai eu beaucoup de problèmes avec x2go et j'ai perdu pas mal de mesures effectuées, ce qui m'a amené à être un peu limite au niveau de la deadline.

J'ai par exemple effectué des mesures en modifiant mon **learning rate** de 0.001 à 0.004 : on obtient de meilleurs résultats avec le deuxième. C'est pour ça que dans mon train_perso.py, le learning rate est à 0.004 et que dans train_example.py il est à 0.001 (valeur de départ).

D'autres pistes auraient été de changer le réseau de neurones afin de tester d'autres configurations peut-être meilleures. Mais aussi de s'attarder intelligemment sur le système de récompenses adopté (j'ai pour ma part choisi le classique pour m'intéresser plus en détail aux mini-batch).

Prendre en compte le nombre d'IA s'affrontant aurait aussi pu être une piste de recherche, en créant par exemple plus d'IAs qui s'affrontent les unes contre les autres.

Conclusion

Je pense avoir beaucoup appris à l'aide de ce projet. J'ai vraiment eu du mal à comprendre et à digérer ce système de récompense après qu'une partie ait été jouée : cette logique me paraissait compliquée. Puis petit à petit, je me suis habitué au mode de fonctionnement et j'ai été surpris à quel point une machine peut véritablement apprendre par elle-même.