CentraleSupélec

Projet Reinforcement Learningr « Connect 4 »

Création d'agents apprenant le puissance 4

Auteurs :
Mathian Akanati
Martin Bridoux
Faustine Malatray
Côme Stephant

Encadrant :
Hédi Hadiji
Céline Hudelot

Avril 2023

CentraleSupélec

Table des matières

1 Introduction et contexte

Le Reinforcement Learning est une technique qui permet à un agent de prendre des décisions dans un environnement complexe pour maximiser une récompense donnée. Lors de ce projet, nous nous sommes intéressés au Puissance 4 qui est un terrain de jeu idéal pour tester les performances d'un tel agent. Le jeu est simple en apparence, mais les possibilités sont nombreuses. Cela offre un environnement propice à la mise en place et à l'évaluation d'un agent d'apprentissage par renforcement.

Nous nous sommes basés sur la méthode du Q-learning afin d'entraîner notre agent. Nous avons obtenu des performances intéressantes. Dans ce rapport, je me propose de détailler ma contribution au projet.

FIGURE 1 – Photo d'un humain s'entraînant contre lui-même au puissance 4

2 Choix d'implémentation

Le Q-Learning est une méthode d'apprentissage par renforcement qui s'adapte bien aux problèmes de décision séquentielle tels que le jeu de Puissance 4. En effet, le Q-Learning permet à un agent de prendre des décisions en fonction des états de l'environnement et des récompenses potentielles associées à ces décisions, ce qui est exactement le type de situation rencontrée dans le jeu de Puissance 4.

Tous nos programmes ont été conçus en python. Nous utilisons des Jupyter Notebook, des fichiers python qui contiennent nos différents agents et un fichier utils.py qui répertorie l'ensemble des fonctions d'intérêt.

L'entraînement d'un agent utilisant le Q-Learning est primordial pour obtenir de bonnes performances. Nous avons fait le choix de l'entraîner avec plusieurs agents différents. Mon rôle principal a été tout d'abord de réfléchir avec Mathian à de réfléchir avec des règles basiques de stratégies qui peuvent s'implémenter simplement sur un agent pour qu'il joue de manière intelligence sans necéssairement avoir besoin d'apprentissage. Ensuite, après la création de ces types d'agents et de la création d'agents de Q-learning par mes autres camarades, j'ai entraîné ces agents entre eux selon différentes approches pour établir la meilleure façon d'entraîner l'agent. Enfin, j'ai participé à la mise en place d'une interface permettant à un utilisateur de jouer directement contre un agent, via l'interface PettingZoo.

Agents ayant contribué à l'entraînement :

— RandomPlayer

— PlayLeftmostLegal

— MalynxDeep : Calcul avec une profondeur de 1

— MalynxAvoidingBlunder (MAB) : Détecte les coup où l'agent se ferait battre au prochain coup et évite de les jouer

3 Description des agents

3.1 RandomPlayer PlayLeftmostLegal

Ces deux algorithmes sont proposés dans le squelette du projet qui nous a été proposé.

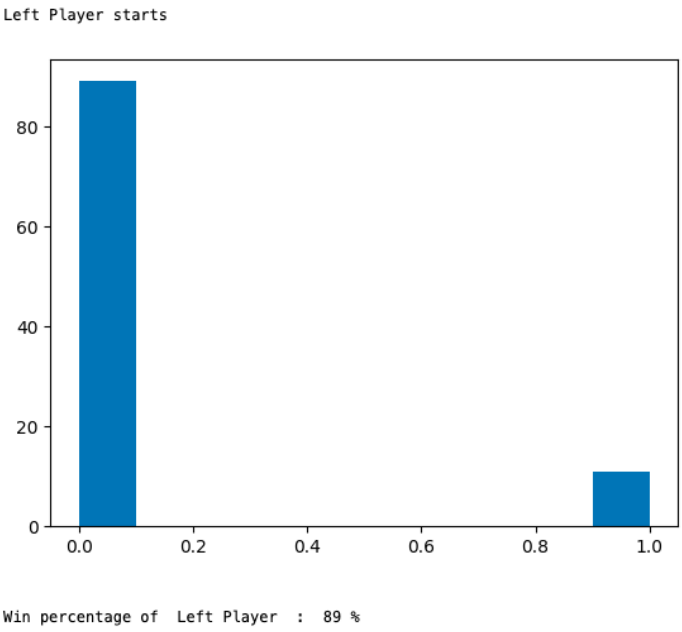

FIGURE 2 – Left Player starts against Random player

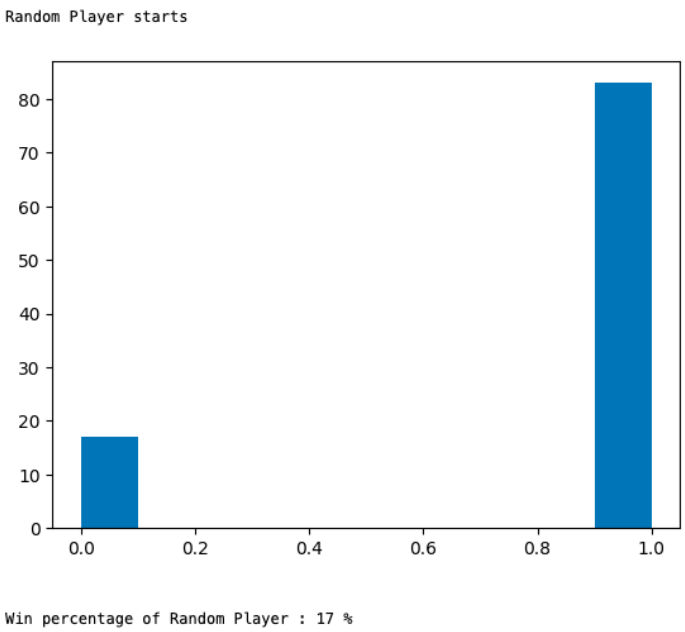

FIGURE 3 – Random Player starts against Left Player

Bien que PlayLeftmostLegal gagne contre RandomPlayer la majorité du temps (Figure 2), nous allons très peu nous baser dessus compte tenu de son déterminisme et du faible nombre de situation proposé par cet agent.

3.2 Malynx

Ces agents sont plus complexes que les deux précédents. Leurs implémentations est située dans les fichiers :

`malynx_avoiding_blunder.py`
`malynx_deep.py`

Le principe de ces algorithmes est d'attribuer un score à l'agent pour une position donnée. Voici comment est calculé ce score :

— Pour chaque alignement de pion, l'agent se voit attribuer 1 point par pion aligné si il y a une case vide dans le prolongement de l'alignement

— Pour chaque alignement de pion, l'agent se voit attribuer 2 points par pion alignés s'il y a 2 cases vides de part et d'autre de l'alignement

— Le premier pion est toujours joué au centre (il a été établi que c'était le meilleur coup pour commencer une partie de puissance 4, voir Solution exacte)

— Le score final de la position est la différence entre le nombre de point de l'agent et celle de son opposant

— Si 4 pions sont alignés, le score de la position est 1000, afin que les agents ne regardent pas d'autres coup s'il voient une possibilité directe de gagner

L'agent choisit ensuite un coup au hasard parmi les coups lui offrant le meilleur score. Il y a donc toujours une part d'aléa dans ces algorithmes.

3.2.1 Implémentation

L'idée avec ce agents était de créer cette série de règles précédemment citées de sorte à ce que l'agent les suivent à chaque coup, sans exception. Pour cela, nous avons créé une série de fonctions qui représentent les règles citées plus haut. Ensuite nous avons créé une fonction `evaluate_position` qui utilise toutes ces fonctions pour, à partir d'un board de puissance 4 fourni en entrée, dans un certain état, évaluer le score du board pour le joueur 1. Plus le score est élevé, plus le joueur 1 est en bonne position. Ensuite, nous avons créé une fonction `evaluate_best_choice` qui prend en entrée un jeu dans un état donné, et va lister les scores de tous les coups suivants possibles. Elle va alors renvoyer la liste des coups dont les scores suivant sont les plus élevés. Ainsi, l'agent choisit aléatoirement parmi cette liste de coups. En plus des fonctionnalité précédemment citées, l'agent Malynx Avoiding Blunder peut détecter si après son coup, l'adversaire est en position de gagner la partie en 1 coup. Il évite dès lors de jouer les coups pour lesquels c'est le cas. Ce qui augmente drastiquement ces performances. Il gagne invariablement contre Left Player, même lorsque Left Player joue en 1er (voir Figure 4). Enfin, il a été ajouté à ces agents une probabilité de supprimer les meilleurs coups que l'agent allait jouer, ce qui le ferait choisir son 2e "meilleur choix", et ajouterait donc une part d'aléatoire à ces agents et empêcherait des agents de learning d'apprendre trop facilement à jouer contre eux, car leur comportement serait trop déterministe autrement.

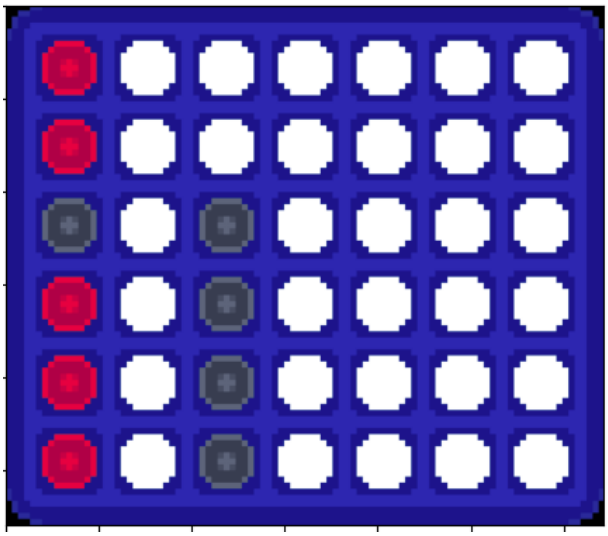

FIGURE 4 – Malynx Avoiging Blunder gagnant contre Left Player

Ces figures illustrent qu'en moyenne, l'agent aléatoire ne remporte que 1% des parties conrtre l'agent Malynx Avoiding Blunder. Ces performances sont déjà remarquables.

3.3 Q-learning agent

L'agent de Q-learning, implémenté par Martin est un agent qui implémente l'algorithme Q-learning pour jouer au jeu. L'agent maintient une table Q qui contient les valeurs Q pour chaque état-action possible dans le jeu. Lorsqu'il doit sélectionner une action, il utilise un epsilon-greedy policy, qui choisit soit la meilleure action selon les valeurs Q, soit une action aléatoire avec une probabilité epsilon. J'ai participé à étudier son comportement contre les autres agents cités précédemment.

4 Résultats et performances - Q-learner vs le reste du monde

L'intégralité du code permettant d'aboutir à ces résultats se trouve sur ce dépôt Github.

4.1 Against Left et RandomPlayer

Observons les résultats de ces différents agents.
On remarque que l'agent de Q-learning n'a aucun mal à battre le LeftPlayer.
Contre le RandomPlayer, ses statistiques sont très bonnes, avec en moyenne plus de 90% de victoires. On remarque cependant qu'il ne s'améliore pas vraiment plus que ça avec le temps, et qu'il stagne autour de ce pourcentage. On a aussi étudié l'évolution de ses parties si l'on se met à l'entraîner contre les agents Malynx entre deux entraînements contre RandomPlayer. Mais ce ne sont pas ces entraînements intermédiaires qui font que l'agent va s'améliorer contre RandomPlayer.

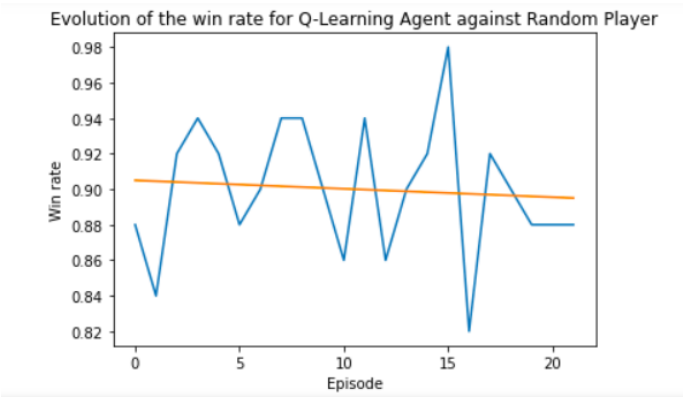

FIGURE 5 – Q-learner against Random Player - 1st round of training

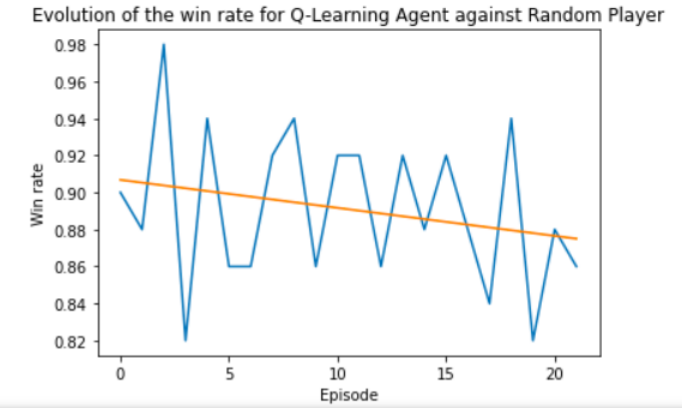

FIGURE 6 – Q-learner against Random Player - after training against Malynx

4.2 Against the Malynx twins

4.2.1 Sans aléa

Contre les agents Malynx, son apprentissage est intéressant : en commençant à l'entraîner, il ne fait quasiment que perdre contre ces agents, tant que ces agents sont avec le pourcentage d'aléatoire fixé à 0.

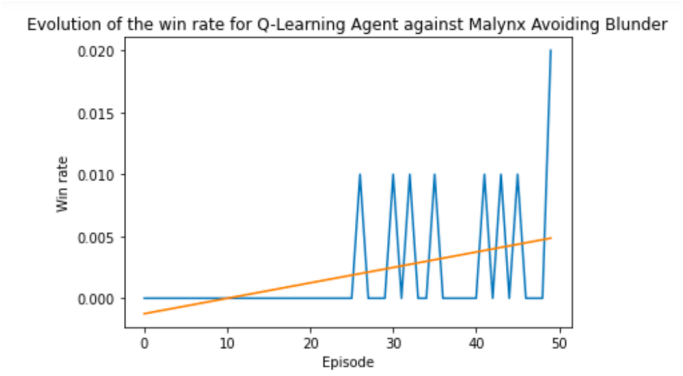

FIGURE 7 – Q-learner against Blunder - 50 ep. of 100 games

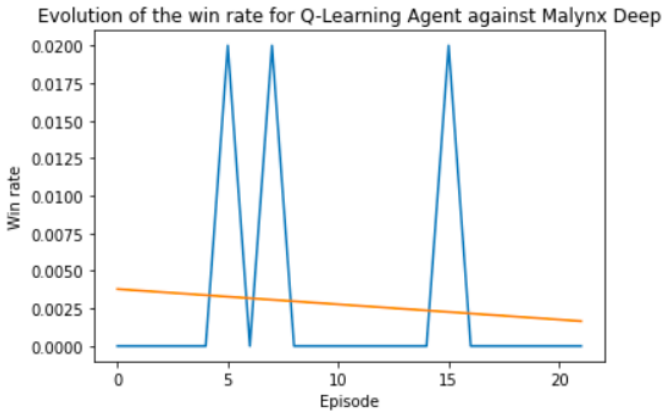

FIGURE 8 – Q-learner against MalynxDeep - 22ep. of 50 games

En revanche, il a fallu de quelques entraînements de plus pour qu'en quelques victoires, le Q-learner comprenne complètement la logique des agents Malynx et se mette à gagner très rapidement, jusqu'à atteindre 100% de victoire.

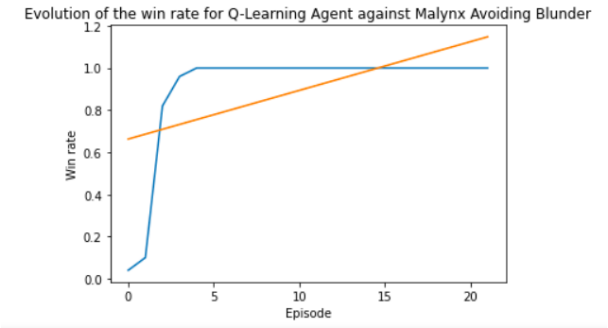

FIGURE 9 – Q-learner against Blunder - after a few training rounds

4.2.2 Avec aléa

Cependant, on peut facilement vérifier l'importance de la composante aléatoire dans les agents Malynx. En augmentant progressivement le pourcentage d'aléatoire, on se rend compte que le Q-learner a beaucoup plus de mal à comprendre le fonctionnement des agents Malynx et gagne très très peu. Voici un exemple de training de 100 épisodes de 100 parties contre le Blunder avec un aléa à 0.3 :

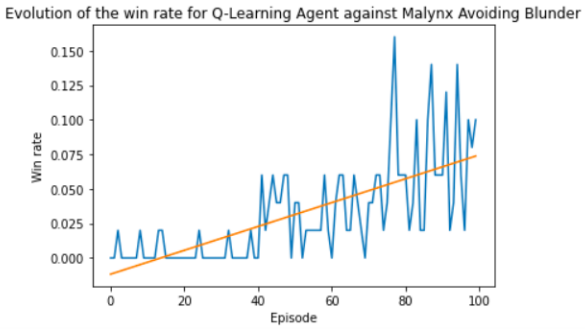

FIGURE 10 – Q-learner against Blunder($\text{random}_p ercentage = 0.3$)

Finalement, en conservant un seuil d'aléa inférieur à 0.5, le pourcentage de victoire de notre Q-learning agent ne dépassera pas 1%. On peut donc en conclure que de combiner un agent muni de règles précises le rendant intelligent, avec de l'aléa qui le rendent plus imprévisible est très puissant. Battre un tel agent demanderait d'implémenter des Q-learner bien plus élaborés.

5 Interface

Enfin, pour pouvoir jouer contre les agents, j'ai participé à l'implémentation d'une interface utilisant PettingZoo qui permet à un utilisateur de jouer contre un agent de son choix. L'interface fonctionne parfaitement lorsque l'on fixe l'agent comme joueur qui commence. Cependant, si l'on fixe l'utilisateur comme joueur qui commence, l'interface laisse le joueur jouer deux fois d'affilée au début, puis ne parvient pas à faire une alternance parfaite entre les deux joueurs.

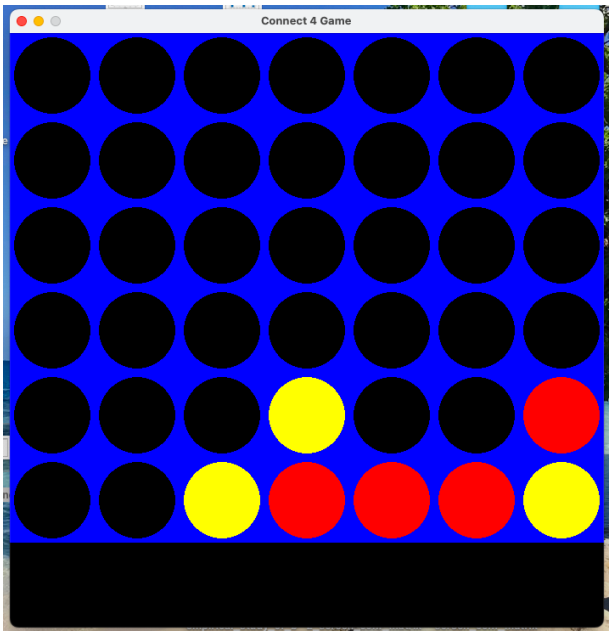

FIGURE 11 – Partie de puissance 4 contre un agent

6 Conclusion

Il est intéressant d'implémenter à la fois des agents de Q-learning et des agents intelligents déterministes pour entraîner notre Q-learner. De plus, l'implémentation de 2 stratégies différentes ainsi que l'autorisation de se fixer un seuil d'aléa permet à l'agent de Q-learning d'explorer plus de situations différentes. L'agent est mis en difficulté avec de l'aléa et son pourcentage de victoire chute drastiquement.