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 – L'homme rivalisera-t-il avec l'agent q-learner ?

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 de créer des agents jouant de manière relativement intelligentes pour permettre au Q-learner de s'entrainer avec des bons joueurs. Dans un deuxième temps, j'ai créé un deuxième agent utilisant une stratégie différente du premier pour gagner.

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.

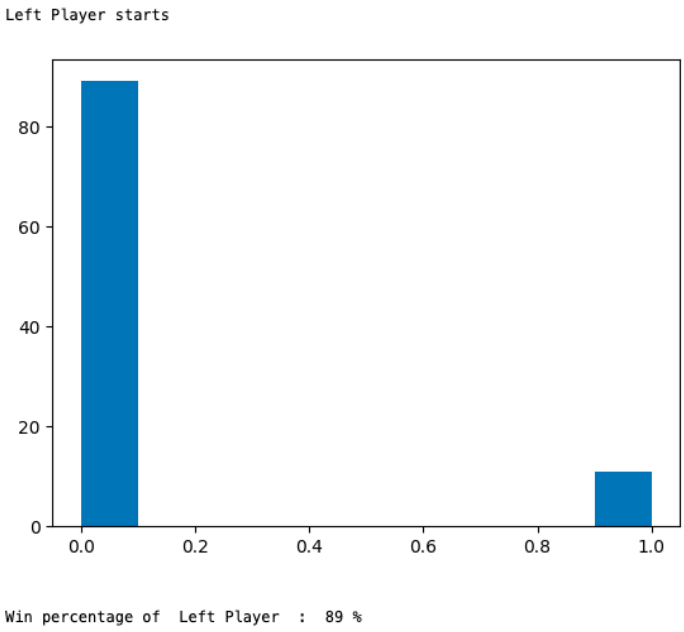

FIGURE 2 – Left Player starts against Random player

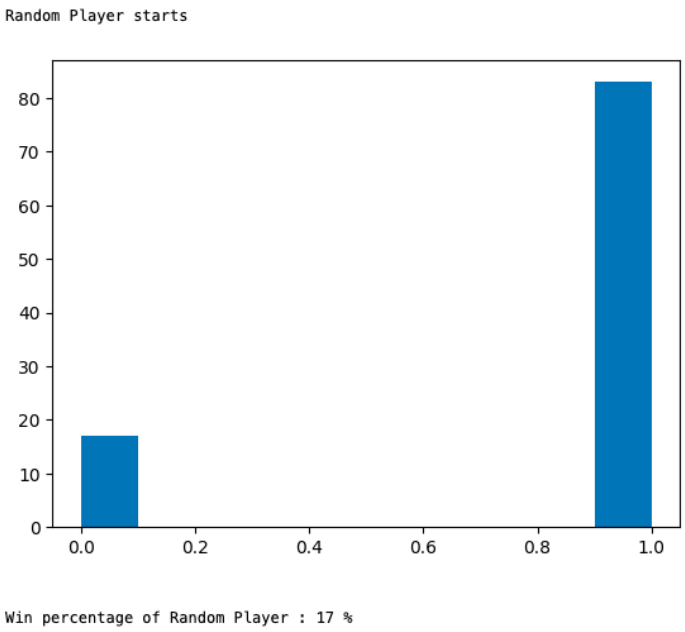

FIGURE 3 – Random Player starts against Left Player

Bien que PlayLeftmostLegal gagne contre RandomPlayer la majorité du temps (Figure 5), 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 Malynx Avoiding Blunder

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).

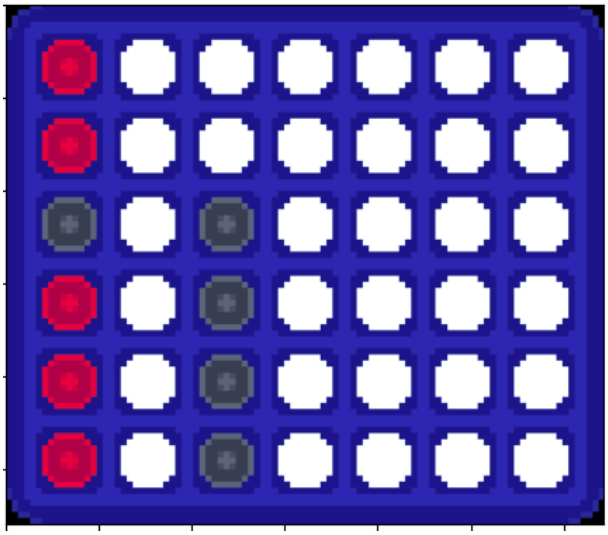

FIGURE 4 – Malynx Avoiging Blunder gagnant contre Left Player

Cet algorithme possède également des performances avantageuses contre Random Player.

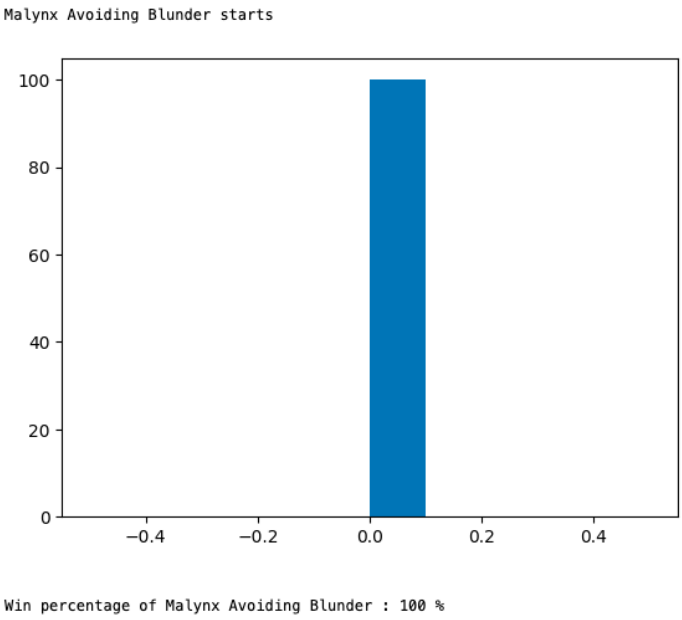

FIGURE 5 – MAB starts 100 games against Random player

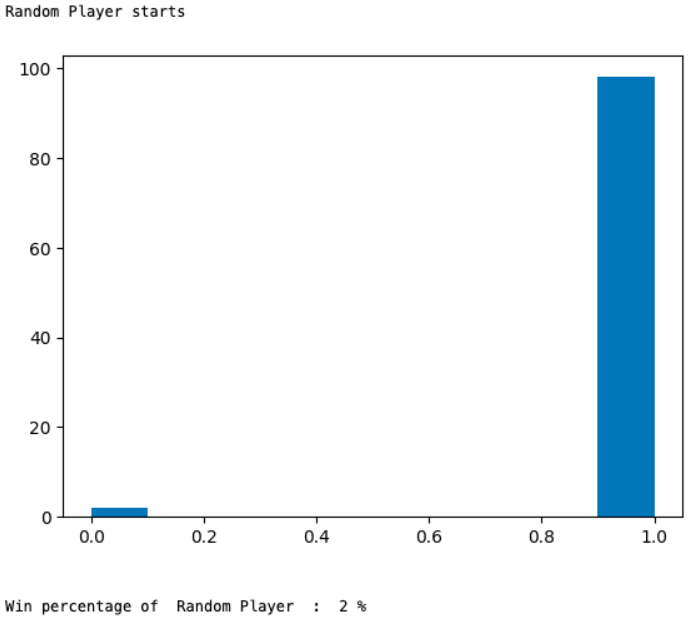

FIGURE 6 – Random Player starts 100 games against MAB

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.

3.2.2 Malynx Deep

Le fonctionnement de Malynx Deep est similaire à celui de Malynx Avoiding Blunder. La seule différence étant qu'avant de jouer un coup, Malynx Deep ne regarde pas seulement la capacité de l'adversaire à termnier la partie, mais le score de chacun de ses coups. L'agent choisi ensuite un coup qui maximise :

```
(score après coup i de l'agent) - (score de l'adversaire après la meilleur réponse au coup i)
```

Cet algorithme a des performances comparables à Malynx Avoiding Blunder concernant PlayLeftmostLegal et RandomPlayer.

Cependant, lorsqu'on fait joué les deux algorithmes l'un contre l'autre, le résultat est souvent décevant, l'agent qui commence gagne souvent en très peu de coup.

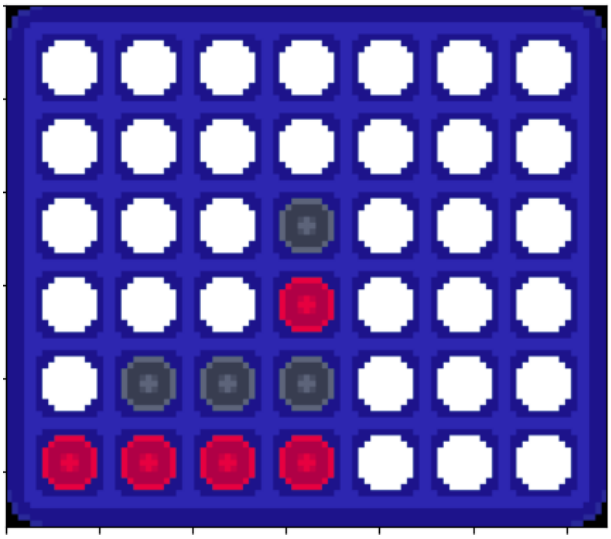

FIGURE 7 – Malynx Deep losing against Malynx Avoiding Blunder

L'origine de ces parties courtes est simple : jouer au dessus du pion de son adversaire enlève beaucoup de possibilité (et donc de points) à ce dernier, c'est l'option qui est souvent privilégiée par ces agents. Par ailleurs, c'est aussi un élément qui permet souvent au 2e joueur de faire partie nulle s'il s'y prend plus intelligemment (voir Éléments stratégiques). Cependant, dans notre cas, cela mène à une situation ou l'agent qui commence à jouer à 3 pions alignés en bas et a donc 2 possibilités de gagner, la victoire est imparable.

Pour palier à ce problème, j'ai encouragé mes algorithmes à briser cet aspect rectiligne en les incitant à jouer non pas au dessus mais à côté des pions de l'adversaire en début de partie. Cela mène à des parties plus intéressantes (voir figure 8).

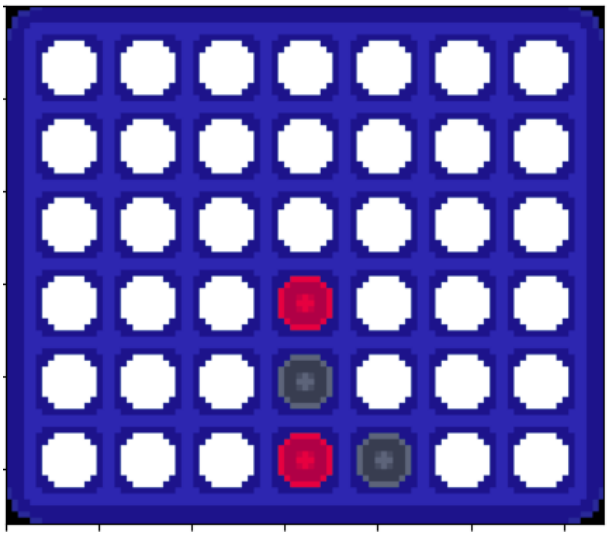

FIGURE 8 – MAB avoids opening trick against MalynxDeep

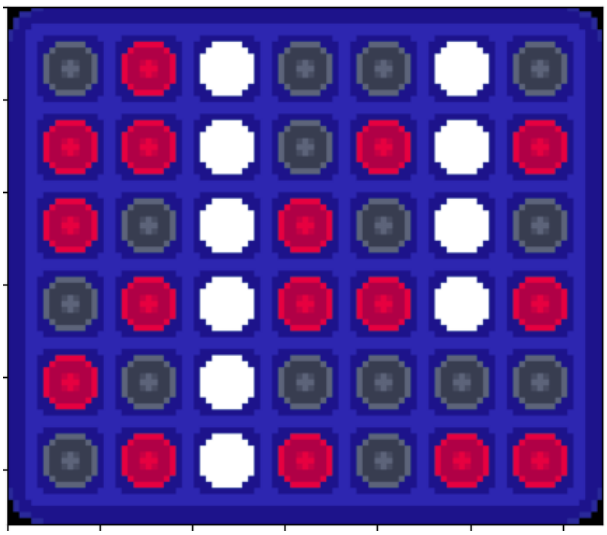

FIGURE 9 – MAB finally wins against Malynx Deep

4 Résultats et performances

L'intégralité du code permettant d'aboutir à ces résultats se trouve sur ce dépôt Github. Le Jupyter Notebook « project_starter »fournit tous les éléments pour retrouver ces résultats

Observons les résultats de ces différents agents. Le pourcentage est calculé suivant ces règles :

— 1 point pour une victoire

— 0 point pour une défaite

— 0.5 point pour une partie nulle

Ainsi, si un agent réalise 2 victoires, 2 défaites et 3 parties nulles contre un autre agent, on considère que son taux de victoire est de 50%

Taux de victoires des 2 agents implémentés sur 100 parties :

Opponent	Malynx Avoiding Blunder	Malynx Deep
RandomPlayer starts	98%	95%
RandomPlayer responds	100%	100%
LeftPlayer starts	100%	100%
Left Player responds	100‰	100%
MAB responds	74%	58%
MalynxDeep responds	72%	56%

Comme vu précédemment, les deux agents sont largement supérieurs aux agents PlayLeftmostLegal et RandomPlayer. De plus, les deux agents ont un niveau semblables et sont avantagés lorsqu'ils commencent s'ils jouent l'un contre l'autre.
Un autre constat que l'on peut faire est que MalynxAvoidingBlunder à des pourcentages de victoire plus élevé contre les agents Malynx.

5 Style de jeu

Empiriquement, on observe que l'agent MAB est plus offensif que l'agent MalynxDeep. En effet, lorsque MAB joue contre lui-même, la probabilité d'obtenir une partie nulle n'est que de 1.5% (voir Figure 12). Alors que pour MalynxDeep, on obtient 18% de parties nulles. Malgré cela, comme on l'a vu dans le tableau précédent (voir 4), les 2 agents ont des niveaux comparables.

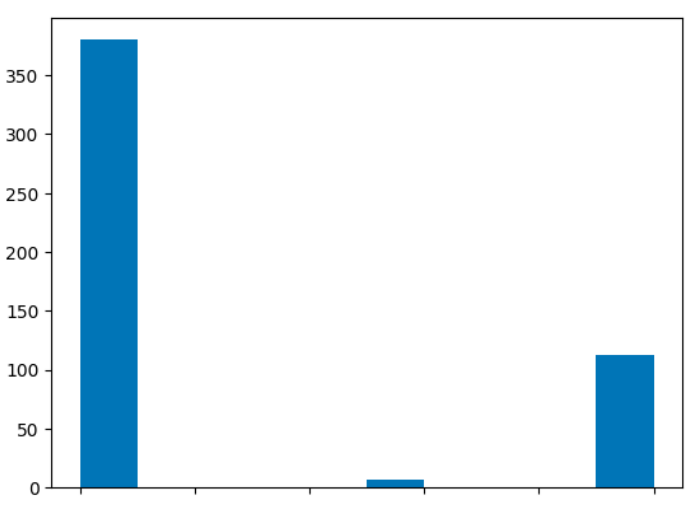

FIGURE 10 – MAB plays 500 games against itself

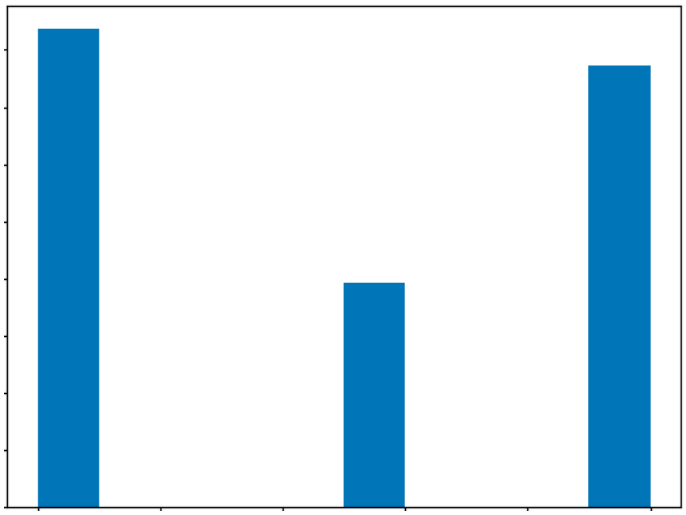

FIGURE 11 – MalynxDeep plays 500 games against itself

Ce phénomène est explicable par le fait que MalynxDeep retranche au score d'une postion les points que pourraientt inscrire son adversaire. Ainsi, il lui ôte plus de possibilité d'attaque, il joue plus défensivement et obtient plus de parties nulles.
Cette différence de comportement est idéale pour entraîner notre agent de Q-learning sur des agents adoptant différentes stratégie.

6 Aléa

Une option aléatoire est disponible. Les algorithmes perdent en performance lorsqu'on leur ajoute de l'aléa, cependant, cela permet encore une fois à l'agent Q-learner de s'entraîner sur des positions variées.

Perte de performance avec l'aléa :

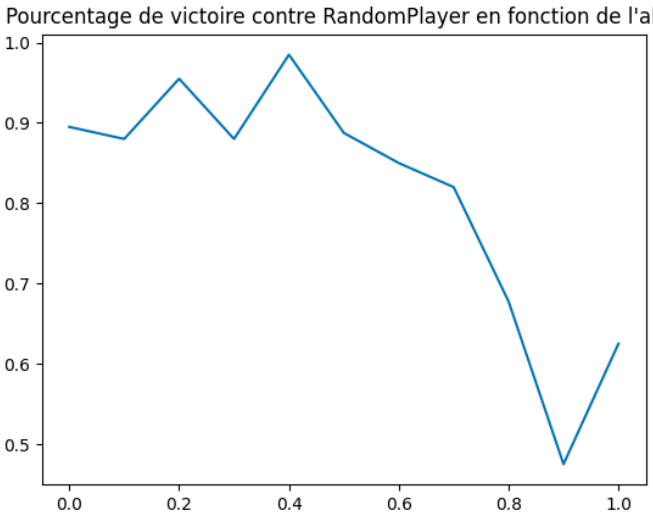

FIGURE 12 – Baisse de performance de MAB contre RandomPlayer

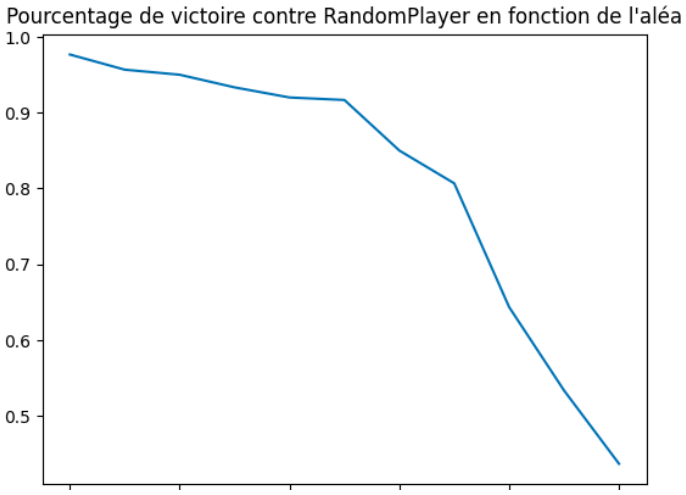

FIGURE 13 – Baisse de performance de Malynx Deep contre RandomPlayer

On remarque que les agents Malynx restent significativement plus forts que l'agent aléatoire (+90% de victoire) tant que le seuil d'aléa ne dépasse pas les 0.5. C'est le seuil que l'on ne dépassera pas pour entraîner notre agent Q-learner.

7 Conclusion

Cette construction d'agents permet d'entraîner le modèle de reinforcement learning sur des agents déjà intelligents et donc d'augmenter significativement ses performances. 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 de voir une myriade de situations différentes.

Par ailleurs, on observe que nos agents 'Malynx' sont déjà très performants et peuvent rivaliser avec des joueurs occasionnels de Puissance 4.