Semestre Septembre 2021-2022

SORBONNE
UNIVERSITÉ

RAPPORT DE PROJET--LU3IN005

Exploration/Exploitation et Puissance 4

Groupe 1 Binôme 6

GUO Youheng 28711440
YANG Kai 21107392

Présentation du projet

Dans ce projet, notre objectif principal est d'implémenter un jeu (puissance 4) pour qu'on puisse le jouer entre deux joueurs et d'optimiser les algorithmes pour des joueurs (IAs) de jeu.

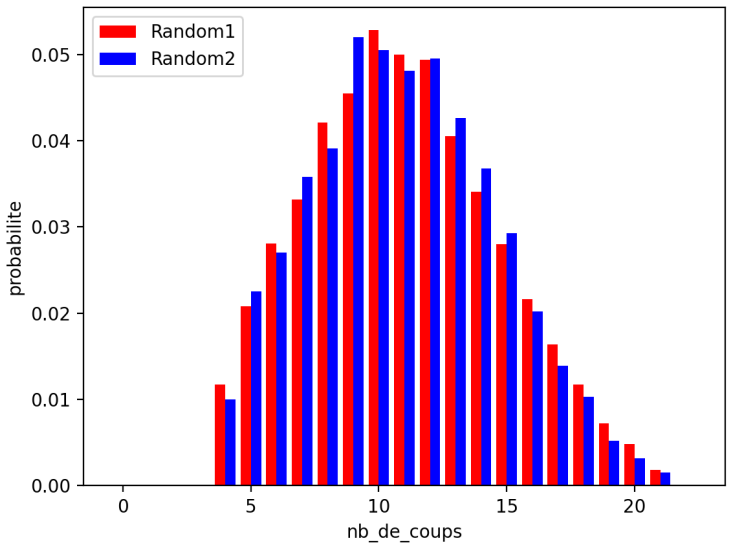
Ce projet consiste à faire réaliser des stratégies de joueur en utilisant des différents algorithmes. Des algorithmes principaux étaient à implémenter : algorithme de Monte-Carlo, l'algorithme aléatoire, l'algorithme greedy, l'algorithme e-greedy, l'algorithme UCB et l'algorithme UCB adapté aux arbres de jeu.

Explication des statégies et analyse des résultat

Partie 1 : Combinatoire du puissance 4

Tout d'abord, on a obtenu une liste de 69 quadruplets de cases (pour le plateau 6 * 7) après avoir implémenté la fonction qui calcule des cas gagnants.

Ensuite, nous avons implémenté le moteur de jeu dans la classe « Game » et un joueur dans la classe « Player » pour qu'un joueur puisse jouer une partie classique avec un autre joueur (les joueurs jouent aléatoirement sans stratégie). On peut bien obtenir un résultat de la partie, ça nous permettra d'analyser le nombre de coups avant une victoire lorsque les deux joueurs jouent aléatoirement. Après 10000 parties, on a un graphique qui décrit la distribution de nombre de coups. Ça suit la loi de poisson.

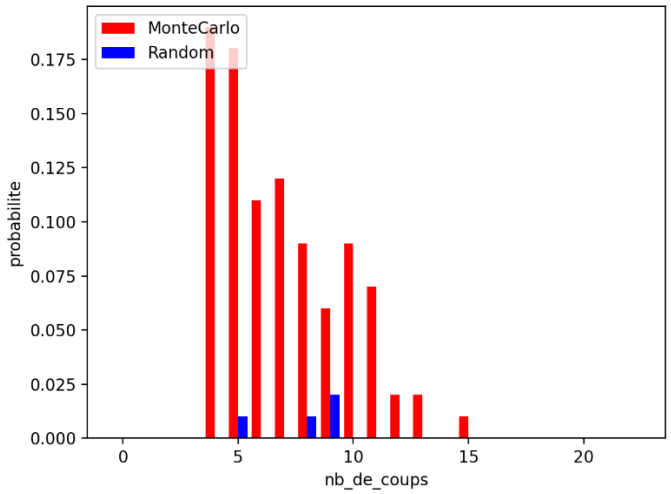


La distribution du nombre de coups

```
Random1 won [0, 0, 0, 0, 117, 208, 281, 332, 421, 455, 528, 500, 494, 405, 341, 280, 216, 164, 117, 72, 48, 18, 0] in total 4997 times
Random2 won [0, 0, 0, 0, 100, 225, 270, 358, 391, 520, 505, 481, 495, 426, 368, 293, 202, 139, 103, 52, 32, 15, 0] in total 4975 times
there are  28 ties during the games, and the probability is 0.0028
```

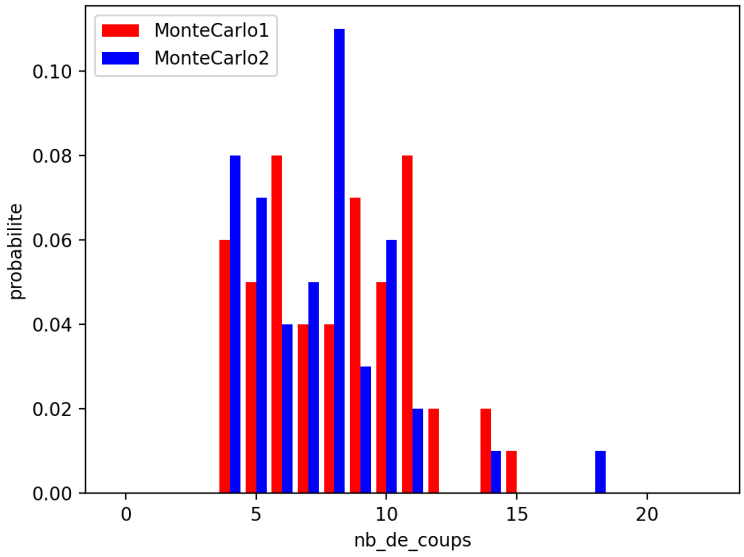
Ici on peut bien calculer la probabilité de la partie nulle (nombre de parties null/nombre de parties totales), environ 0.28%.

Partie 2 : Algorithme de Monte-Carlo



MonteCarloPlayer VS RandomPlayer

```
MonteCarlo won [0, 0, 0, 0, 19, 18, 11, 12, 9, 6, 9, 7, 2, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0] in total 96 times
Random won [0, 0, 0, 0, 0, 1, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] in total 4 times
there are  0 ties during the games, and the probability is 0.0
```

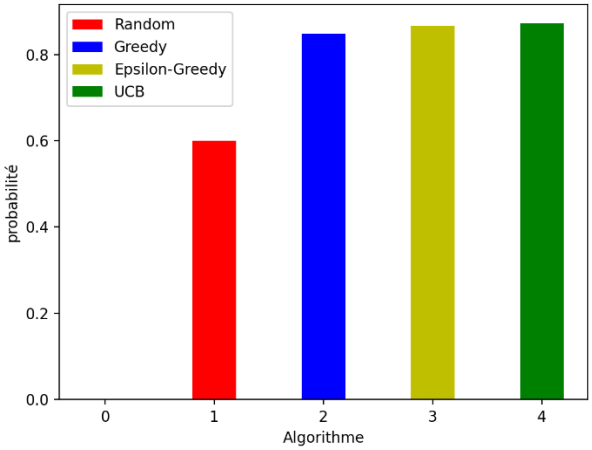


MonteCarloPlayer VS MonteCarloPlayer

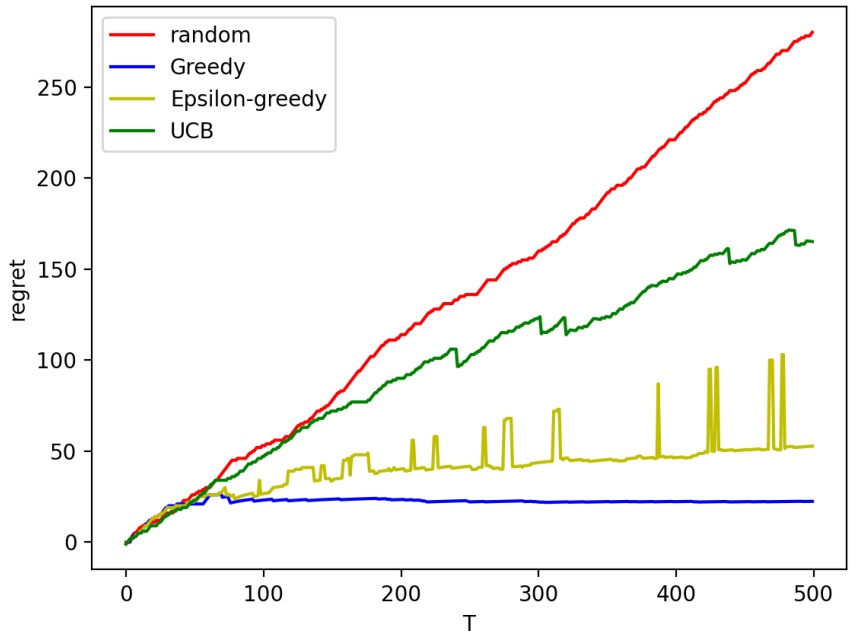
```
MonteCarlo1 won [0, 0, 0, 0, 6, 5, 8, 4, 4, 7, 5, 8, 2, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0] in total 52 times
MonteCarlo2 won [0, 0, 0, 0, 8, 7, 4, 5, 11, 3, 6, 2, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0] in total 48 times
there are  0 ties during the games, and the probability is 0.0
```

C'est normal que les deux joueurs ont la probabilité très proche.

Partie 3 : Bandits-manchots

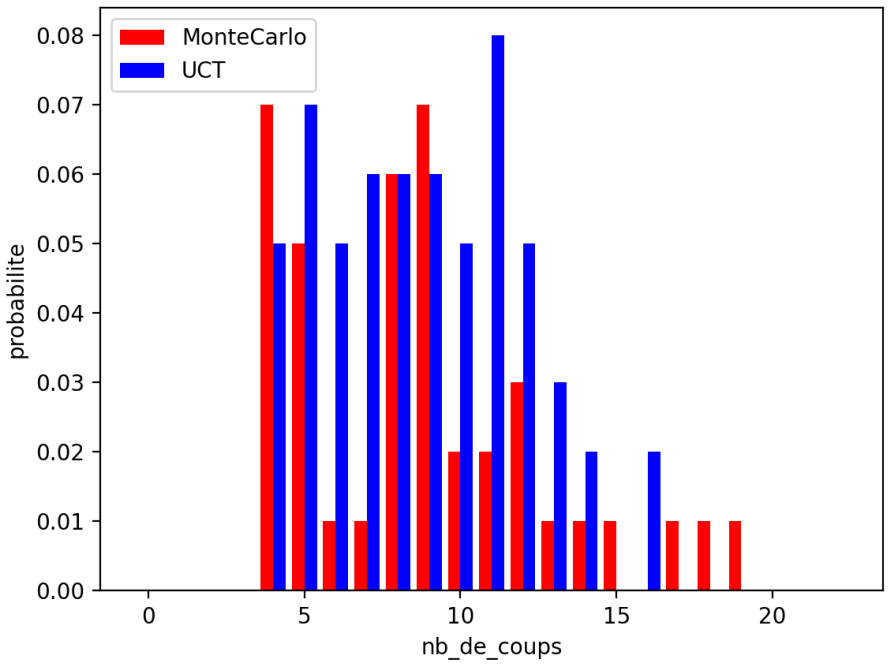


Parmi les 4 algorithmes, l'algorithme aléatoire est le pire cas majoritairement. Pour l'algorithme greedy, si il a chosi le levier avec le rendement le plus haut dans les premiers choix, il est donc le meilleur cas, sinon il est le pire cas, c'est un algoritheme instable. L'algorithme epsilon-greedy évite l'instabilité de l'algorithme greedy, il peut trouver le levier avec le rendement le plus haut après centrain fois d'apprentissage. Et après avoir augmenté le nombre de levier, l'algorithme UCB devient plus en plus utile.



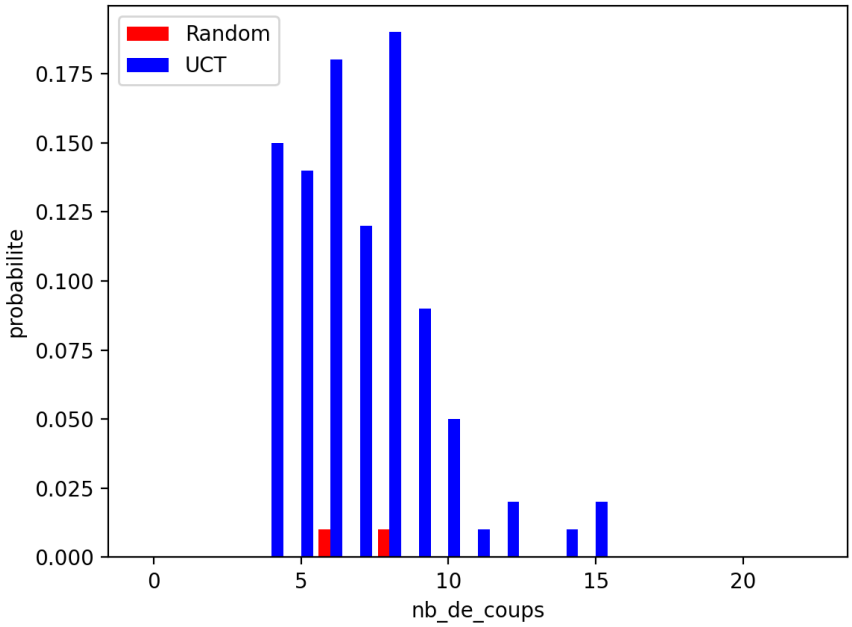
Regrets

Partie 4 : Arbre d'exploration et UCT




```
MonteCarlo won [0, 0, 0, 0, 7, 5, 1, 1, 6, 7, 2, 2, 3, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0] in total 40 times
UCT won [0, 0, 0, 0, 5, 7, 5, 6, 6, 6, 5, 8, 5, 3, 2, 0, 2, 0, 0, 0, 0, 0, 0] in total 60 times
there are  0 ties during the games, and the probability is 0.0
```

On peut observer que l'algorithme UCT est meilleur que l'algorithme Monte-Carlo.




```
Random won [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] in total 2 times
UCT won [0, 0, 0, 0, 15, 14, 18, 12, 19, 9, 5, 1, 2, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0] in total 98 times
there are  0 ties during the games, and the probability is 0.0
```