Sorbonne Université
Faculté des Sciences et Inginierie
Département d'Informatique

Statistiques et informatique

Mini projet N1 - 3I005

Thème

Problématique de l'exploration vs l'exploitation

Réalisé par

M. BENMESSAOUD Hamza M^{lle} BOURAI Assia

Chargé de TME : Mickael Chen

2018-2019

Table des matières

Introduction

Ce mini projet s'intéresse au dilemme dit de l'exploration vs exploitation.Ce dernier est considéré comme étant un problème fondamental que l'on retrouve dans plusieurs domaines de l'intelligence artificielle.

L'exploitation consiste à choisir l'action estimée la plus rentable à partir d'un certain nombre d'information collectée.

l'exploration consiste à choisir d'autres actions afin d'acquérir le plus d'information. Il est parfois préférable, souvent au début d'un processus, de faire des sacrifices et de ne pas choisir l'option a priori la plus rentable an d'améliorer le gain à long terme.

Mais la question qui se pose est de savoir quand estime-t-on avoir recueilli assez d'informations et que l'exploration n'apportera pas de connaissances supplémentaires.

Le compromis entre la phase exploratoire et la phase d'exploitation est donc cruciale pour optimiser le rendement sur le long terme. L'objectif de ce projet est donc d'étudier différents algorithmes de ces deux concepts pour des IAs de jeux. L'étude se fera en trois parties :

La première partie est dédiée à l'expérimentation des algorithmes classiques d'exploration vs exploitation dans un cadre simple. On s'intéressera là à l'exemple des bandits manchots appelé aussi machine à sous dont le principe est de considérer N leviers chacun ayant une probabilité de gain inconnue du joueur, chacun de ses leviers est une action possible parmi lesquelles le joueur doit choisir à chaque pas de temps. Le but sera donc de maximiser le gain et de minimiser le regret.

La deuxième partie est consacrée à l'implémentation de deux algorithmes pour la résolution du jeu du morpion, le premier étant un algorithme aléatoire et le deuxième un algorithme de Monte Carlo. Quand à la troisième partie, elle est dédiée à l'étude d'un algorithme avancé pour les jeux combinatoires à savoir l'algorithme MCTS (Monte Carlo Tree Search) qui est algorithme d'exploration adapté aux arbres de jeux.

Chapitre 1

Bandits-manchots

Bandits Manchots (ou machine à sous), est un jeu de hasard avec N leviers, numérotés par les entiers {1,2,.., N}. Chacun de ses leviers est une action possible a_t parmi lesquelles le joueur doit choisir à chaque pas de temps t. Chaque levier possède un rendement qui est attribué au joueur de façon aléatoire. On suppose que le rendement du levier i suit une loi de Bernoulli de paramètre μ^i constant en temps. Le gain obtenu avec ce levier est noté r_t.Le but est donc de maximiser ce gain. Pour cela, il faut que le joueur identifie le levier au rendement le plus élevé i^*. L'action choisie à l'instant t est notée : $a_t = argmax_{i\in\{1,,N\}}\mu^i$
Si le joueur joue un autre levier que i^* , il aura un gain total inférieur au gain maximal espéré. Soit G_T le gain au bout de T parties $G_T = \sum_{t=1}^{T} r_t$ et G_T^* le gain maximal espéré $G_T^* = \sum_{t=1}^{T} r_t^*$.
D'autre part, l'objectif sera de minimiser le regret. On appelle regret au temps T la différence entre le gain maximal espéré et le gain du joueur, noté $L_T = G^* - \sum_{t=1}^{T} r_t$.

1.1 Stratégies et description des algorithmes

1.1.1 Algorithme aléatoire

Dans cet algorithme l'action à jouer est choisie de manière aléatoire et uniforme parmi toutes les actions possibles. Cette stratégie est basée sur l'exploration des N leviers ne disposant d'aucune information concernant le gain de chaque levier. Cette stratégie aléatoire est donc considérée comme la moins rentable et servira de référence pour l'étude des autres stratégies.

1.1.2 Algorithme greedy (glouton)

Il s'agit d'un algorithme basé sur l'exploitation. Il est décomposé en deux phases : la première phase est consacrée à l'exploration : pendant n itérations chaque levier est joué de manière uniforme puis dans la seconde phase on joue de manière à choisir le levier dont le rendement estimé est maximal.

1.1.3 Algorithme ε-greedy(ε-glouton)

L'algorithme glouton détermine une solution après avoir effectué une série de choix. Pour chaque point de décision, il retient le choix qui semble le meilleur à cet instant. Cette stratégie ne produit pas toujours une solution optimale. Pour y remédier, l'algorithme ε-glouton modifie l'algorithme glouton en introduisant ε, et utilise à chaque itération un tirage semi-uniforme, qui consiste à suivre la meilleure politique à chaque instant t :

- Avec une probabilité de 1 - ε_t,recommander l'action ayant le plus fort taux de récompense estimée (Exploitation) ;
- Avec une probabilité de ε_t , recommander l'action tiré aléatoirement selon une loi uniforme (Exploration).

1.1.4 Algorithme UCB (Upper Confidence Bound)

L'algorithme UCB, tout comme ε-glouton, prend en compte des itérations d'exploration pendant la phase d'exploitation, sauf que contrairement à ce dernier, l'algorithme UCB cherche à faire une exploration d'une façon optimale et choisi l'action tel que : $a_t = argmax_{i \in \{1,,N\}} \left(\hat{\mu}_t^i + \sqrt{\frac{2\log(t)}{N_i(t)}} \right)$ ($\hat{\mu}_t^i$: la récompense moyenne estimée pour l'action i à partir des essais du joueur, $N_i(t)$:le nombre de fois où l'action i a été choisi jusqu'au temps T). Le premier terme est identique aux autres algorithmes, il garantit l'exploitation ; le deuxième terme lui devient important lorsque le ratio entre le nombre de coups total et le nombre de fois où une action donnée a été choisie devient grand, c'est-à-dire qu'un levier a été peu joué : il garantie l'exploration.

1.2 Expériences

Nous avons implémenté les différentes stratégies décrites dans la section 1.1. Afin d'affiner nos résultats nous avons fait varier nos paramètres à savoir μ :la liste des récompenses moyennes estimées pour chaque levier et N_a :nombre de fois où chaque levier a été choisi, avec un $T = 1000$.

La Figure 1.1 montre le regret en fonction de t pour les stratégies présentées précédemment. Pour cette expérience nous avons mis le nombre de leviers à $N = 5$ avec des μ générées aléatoirement, on a choisi une phase d'exploration de 100 coups pour l'algorithme glouton et $\varepsilon = 0.7$ pour l'algorithme ε-glouton. Les courbes représentent les regrets obtenus en fonction du temps.
On constate que l'algorithme aléatoire possède le taux de regret le plus élevé suivi de ε-glouton, glouton et enfin de UCB. On remarque que la courbe de UCB devient plus stable après 800 coups(T).

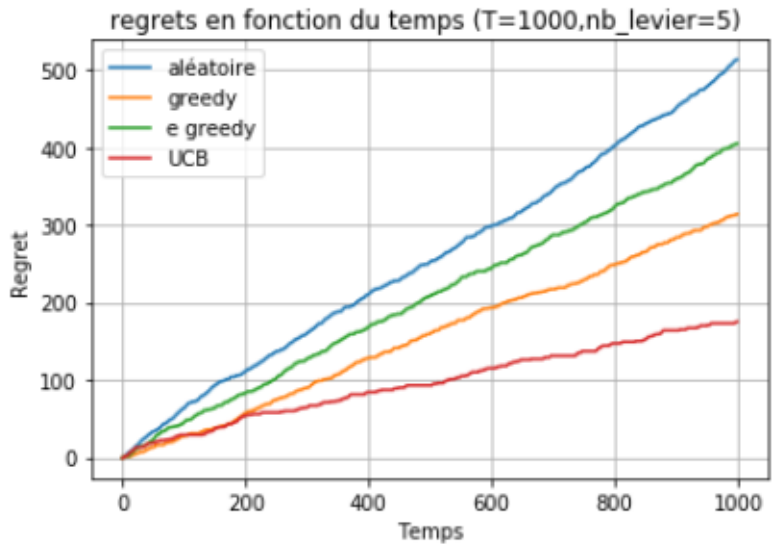

FIGURE 1.1 – Évolution du regret en fonction du temps

La Figure 1.2 montre le regret en fonction de t pour les stratégies présentées précédemment. Pour cette expérience nous avons mis le nombre de leviers a $N = 20$ avec des μ générées aléatoirement, on a choisi une phase d'exploration de 100 coups pour l'algorithme glouton et $\varepsilon = 0.7$ pour l'algorithme ε-glouton. Les courbes sont les regrets obtenus.
On remarque quand on dépasse un certain nombre de leviers, l'algorithme Greedy(glouton) devient inefficace, faute du grand nombre de leviers et de son temps d'exploration *(il n'aura pas le temps d'explorer tout les leviers 100*20 > $T = 1000$)*, tandis que l'algorithme UCB a un regret qui croit façon plus lente que les autres, lui donnant le plus petit regret à la fin.

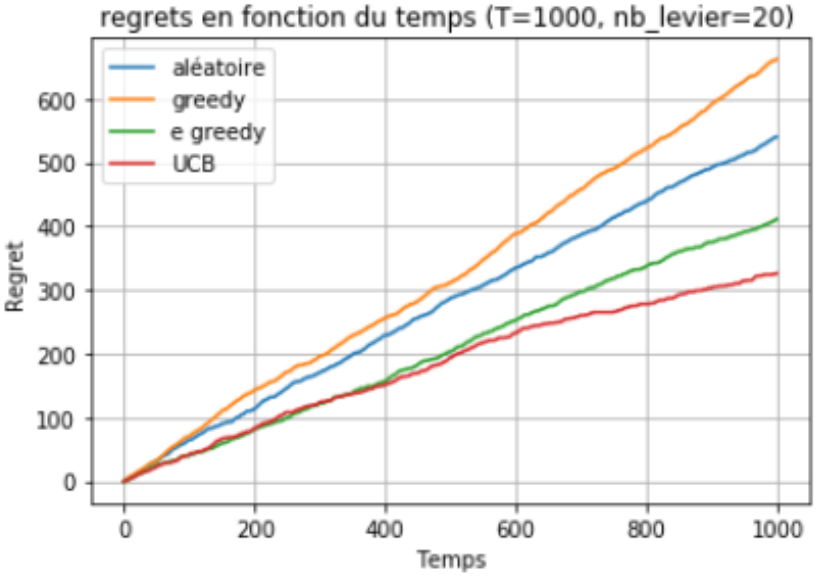

FIGURE 1.2 – Évolution du regret en fonction du temps

Chapitre 2

Morpion et Monte-Carlo

Le jeu du Morpion se joue à 2 joueurs sur un plateau de 3X3 cases. Chaque joueur à son tour marque une case avec son symbole {x||o} parmi celles qui sont libres. Le but du jeu est de parvenir à aligner trois symboles identiques.

2.1 Stratégies et description des algorithmes

2.1.1 Joueur aléatoire

La stratégie du joueur aléatoire est similaire à celle décrite dans Bandits-manchots, c'est à dire que le joueur ne possède aucune information sur les coups qu'il doit choisir pour gagner, il choisi une case à jouer de manière totalement aléatoire parmi les coups possibles disponibles sur la grille.

2.1.2 Joueur Monte-Carlo

L'algorithme de Monte-Carlo est un algorithme probabiliste qui vise à donner une approximation d'un résultat trop complexe à calculer : il s'agit d'échantillonner aléatoirement et de manière uniforme l'espace des possibilités et de rendre comme résultat la moyenne des expériences. Dans notre cas (jeu du morpion), à un état donné, pour chaque action possible un certain nombre de parties est joué au hasard c'est à dire en adoptant la stratégie aléatoire décrite ci-dessus, cette étape représente la phase d'exploration, à l'issue des N parties simulées le nombre de victoire avec chaque action est calculé. L'action avec la moyenne de victoire la plus élevé est ensuite choisie, cette étape est quant à elle celle de l'exploitation.

2.2 Expériences

2.2.1 Joueur aléatoire VS joueur aléatoire

La Figure 2.1 montre l'avancement des parties gagnées de chaque joueurs et les parties nulles en fonction du nombre de parties. On alternant le joueur qui commence le début de la partie, on

se retrouve avec un nombres de parties gagnées presque la même pour les deux joueurs.

```
le joueur1 a une moyenne de parties gagnées de:0.43812709030100333
le joueur2 a une moyenne de parties gagnées de:0.4414715719063545
la moyenne des parties nulles :0.12040133779264214
```

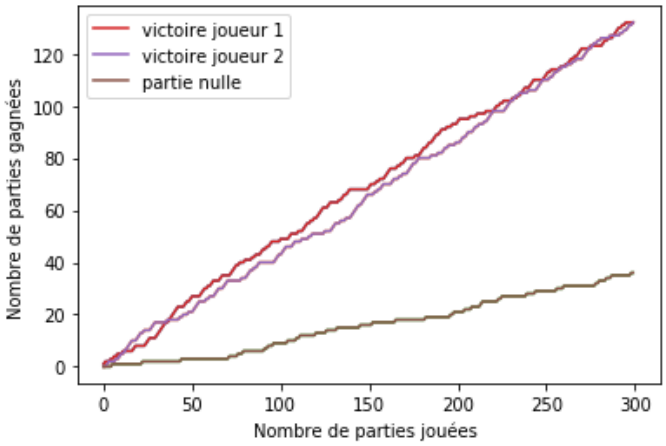

FIGURE 2.1 – Nombre de parties gagnées de chaque joueur par rapport au nombre de parties jouées (deux joueurs aléatoires).

La Figure 2.2 représente l'évolution de la moyenne des partie gagnées du premier et du second joueur, quand le joueur1 commence toutes les parties. La variable aléatoire qui dénote la victoire du premier joueur obéit a la loi de Bernoulli. Dans notre exemple, à l'issue des 1000 parties l'espérance de gain tend vers (0.6). $Lavariance : V(X) = p(1-p), ducoupV(X) = 0.24$.

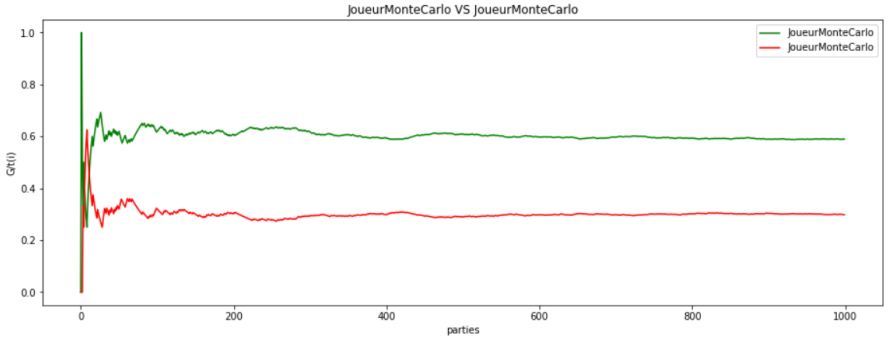

FIGURE 2.2 – Espérance de Gain pour Joueurs aléatoire

2.2.2 Joueur Monte Carlo VS joueur Monte Carlo

La figure 2.3 représente le nombre de parties gagnées de chaque joueurs Monte Carlo, chaque joueur a une simulation $N = 100$, on arrive a constater que le joueur qui commence la partie a plus de chance de la gagner, sans que la différence de taux de parties gagnées des joueurs soient immense.

```
le joueur1 a une moyenne de parties gagnées de:0.5353535353535354
le joueur2 a une moyenne de parties gagnées de:0.46464646464646464
la moyenne des parties nulles :0.0
```

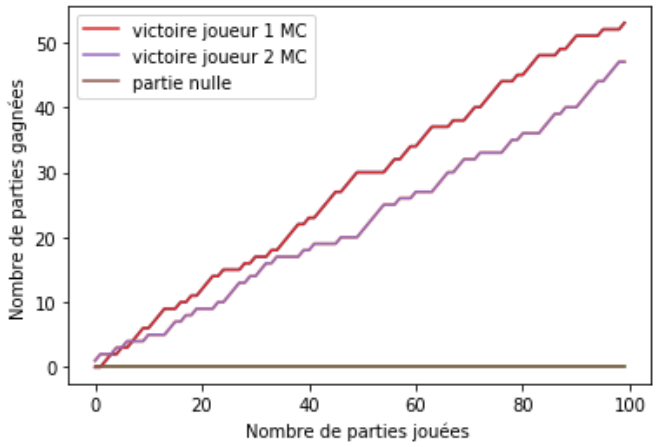

FIGURE 2.3 – Nombre de parties gagnées de chaque joueur par rapport au nombre de parties jouées (deux joueurs MC-Nombre de simulations=100).

2.2.3 Joueur Monte Carlo VS joueur aléatoire

Dans cette partie nous avons fait varier le nombre de parties de simulations du joueurs Monte Carlo. La figure 2.4 qui représente l'avancement des parties gagnées par les deux joueurs, Aleatoire et MonteCarlo, avec un nombre de simulation pour le joueur Monte Carlo de $N = 100$.

```
le joueur1 a une moyenne de parties gagnées de:0.5858585858585859
le joueur2 a une moyenne de parties gagnées de:0.3838383838383838
la moyenne des parties nulles :0.030303030303030304
```

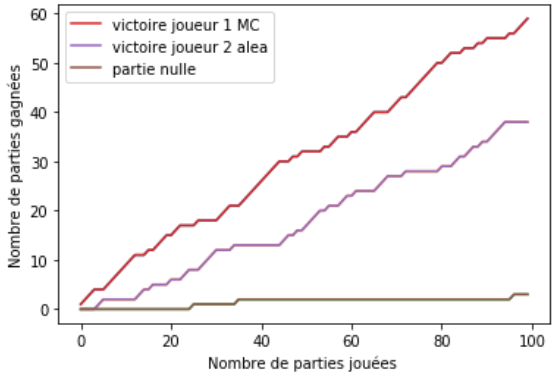

FIGURE 2.4 – Nombre de parties gagnées de chaque joueur par rapport au nombre de parties jouées (Joueur MC VS joueur aléatoire-Nombre de simulations=100).

La figure 2.4 qui représente l'avancement des parties gagnées par les deux joueurs, Aléatoire et Monte Carlo, avec un nombre de simulation pour le joueur Monte Carlo de $N = 10$.On remarque que le résultat est très proche, a cause du nombre de simulation du joueur Monte Carlo qu'on a diminué.

```
le joueur1 a une moyenne de parties gagnées de:0.5151515151515151
le joueur2 a une moyenne de parties gagnées de:0.43434343434343436
la moyenne des parties nulles :0.050505050505050504
```

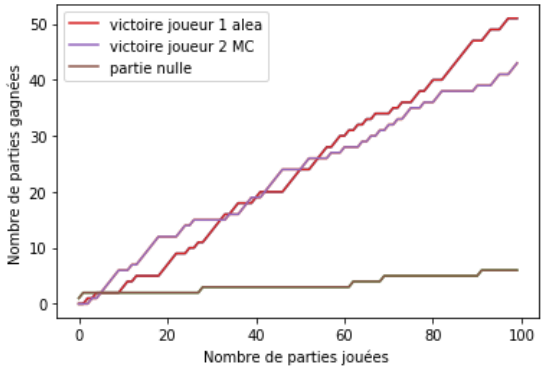

FIGURE 2.5 – Nombre de parties gagnées de chaque joueur par rapport au nombre de parties jouées (Joueur aléatoire VS MC-Nombre de simulations=100).

Remarque :

L'algorithme Monte Carlo est plus performant a chaque fois qu'on augmente le nombre de simulations.

Chapitre 3

Arbre d'exploitation et UCT

3.1 Description de l'algorithme

Désigne l'algorithme UCB adapté aux arbres de jeu, la racine correspond à l'état courant du jeu. Chaque nœud est une configuration et ses enfants sont les configurations suivantes. Le MCTS(Monte Carlo Tree Search)construit un ensemble de feuilles. Une feuille de cet arbre est soit une configuration finale (on sait si un des joueurs a gagné, ou s'il y a match nul), soit une configuration à partir de laquelle aucune simulation n'a encore été lancée. Dans chaque nœud, on stocke deux informations : le nombre de simulations gagnantes dans une partie qui contient n, et le nombre de simulations totale. Comme dans le cas de l'algorithme de Monte Carlo, on a besoin d'une stratégie par défaut pour explorer rapidement la qualité d'un état en d'autres termes la probabilité de gagner à partir de cet état. Cette stratégie par défaut est la stratégie aléatoire. Au tout début, aucune information n'est disponible, on va donc simuler pour chaque action une partie avec le joueur aléatoire pour initialiser les nœuds enfants de la racine. A partir d'un arbre déjà en partie exploré, les différentes étapes sont les suivantes : un nœud à explorer est sélectionné dans l'arbre. Pour cela, en partant de la racine, on choisit le nœud suivant en fonction de l'algorithme UCB parmi les enfants du nœud courant,à partir d'un nœud à simuler, un jeu entre deux joueurs aléatoires est déroulé jusqu'à atteindre un état final (victoire, défaite ou match nul). Et a la fin de chaque partie simulée on met à jour en fonction du résultat le nombre de victoires et le nombre de visites.

3.2 Expériences

3.2.1 Joueur MCTS VS autres joueurs

On remarque d'après les figures 3.1 3.2 3.3 que le joueur UCT est meilleur que le joueur Monte Carlo et Aléatoire. La cause de cette différence est due à la diversité stratégique des algorithmes. Par exemple, Monte Carlo s'appuie sur des probabilités pour choisir une action à jouer, alors que le UCT fait des simulations de jeux afin de choisir la meilleure action a jouer.

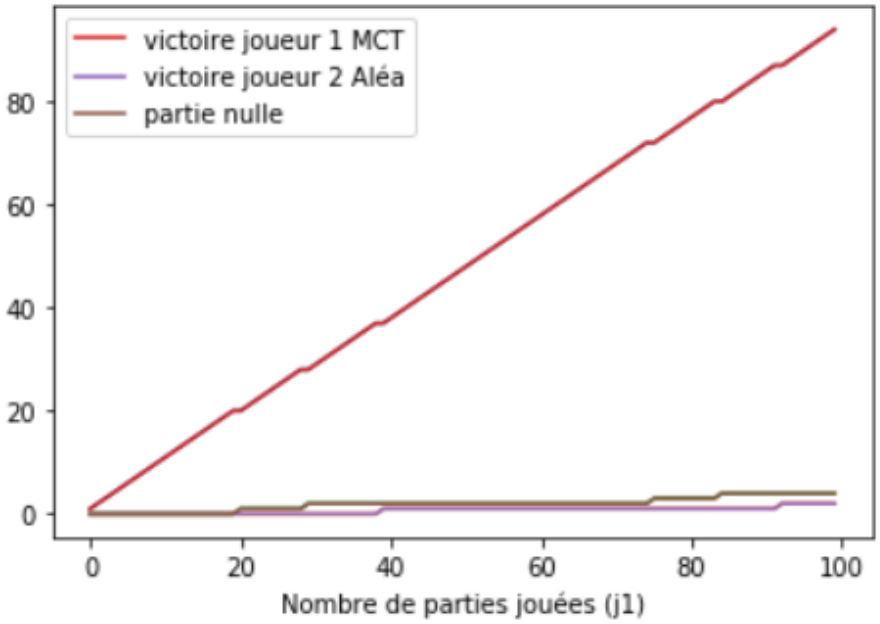

FIGURE 3.1 – Nombre de parties gagnées de chaque joueur par rapport au nombre de parties jouées

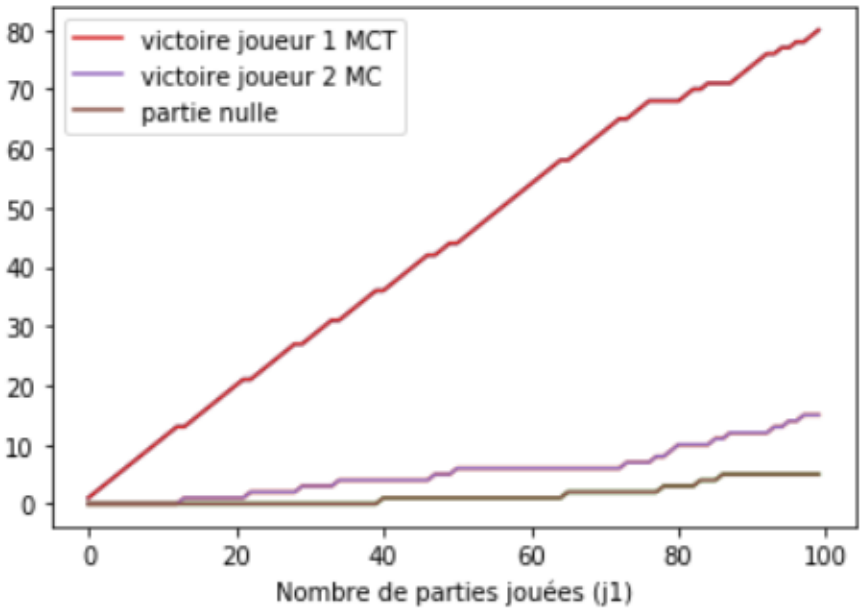

FIGURE 3.2 – Nombre de parties gagnées de chaque joueur par rapport au nombre de parties jouées

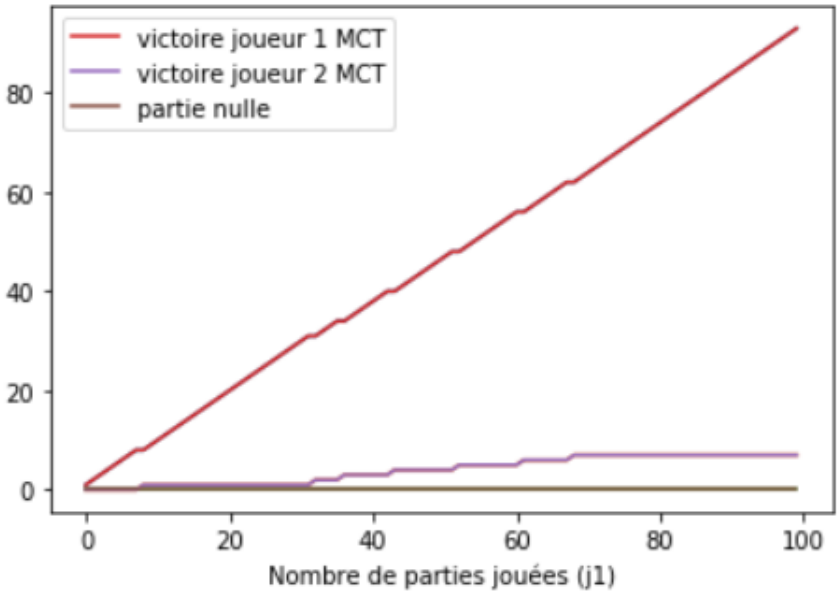

FIGURE 3.3 – Nombre de parties gagnées de chaque joueur par rapport au nombre de parties jouées

Conclusion

Ce rapport a permis de poser plusieurs questions concernant le dilemme de l'exploitation *vs* exploration et d'y répondre. Nous avons tout d'abord étudié l'exemple des bandits-manchots en se basant sur des algorithmes classiques d'exploration et d'exploitation où nous avons constaté que UCB réalise le parfait équilibre entre ces deux stratégies vu son résultat plutôt satisfaisant. Nous nous sommes ensuite concentré sur le jeu du Morpion, tout en utilisant un algorithme de décision aléatoire, un algorithmes Monte Carlo et une stratégie Monte Carlo Tree Search, cette dernière étant basé sur un algorithme UCB a permis d'avoir des joueurs très performants ayant de très bons résultats