# PUISSANCE 4

Vaut-il mieux choisir un déplacement en toute sécurité, ou prendre des risques ?

OCTOBER 10, 2023

VU HOANG THUY DUONG – HALIMATOU DIALLO
Sorbonne Université

# Table des Matières

# Introduction

Le dilemme dit de *l'exploration* et de *l'exploitation* s'apparait dans une multitude de contextes réels, que ce soit dans la gestion d'entreprises, la prise de décisions financières, la recherche scientifique ou même la navigation dans le labyrinthe complexe de l'information numérique. Ces concepts se posent comme un défi complexe, car ils nous obligent à trouver un équilibre délicat entre la recherche de nouvelles opportunités et l'exploitation de celles que nous connaissons déjà pour maximiser nos gains, nos récompenses ou nos avantages à long terme.

**L'exploration**, première facette de cette dichotomie, consiste à s'aventurer hors des sentiers battus, à prendre des risques calculés pour découvrir de nouvelles informations, options ou pistes. Les choix exploratoires peuvent aboutir à des résultats initialement décevants, car ils se basent sur l'inconnu. D'un autre côté, **l'exploitation** représente la stratégie de tirer le meilleur parti des connaissances, des compétences ou des ressources que nous avons déjà acquises. Elle incarne une approche plus conservatrice, où nous optons pour des actions éprouvées et des décisions basées sur notre expérience passée. Cet équilibre délicat entre exploration et exploitation est résolu grâce à des techniques sophistiquées, notamment des algorithmes d'apprentissage par renforcement, qui guident les décisions et les actions dans des domaines aussi variés que l'automatisation industrielle, les jeux informatiques et même la conduite autonome.

Dans notre projet, nous allons comparer quatre algorithmes fondamentaux pour le choix des manœuvres avec l'exemple du jeu de Puissance 4 afin d'arriver à une réponse pour la question : ***« Est-ce que les choix purement aléatoires renvoient le meilleur taux de victoire ? Vaut-il mieux choisir les leviers ayant le nombre maximal de victoire après $N$ fois joué? Ou faut-il aussi basé sur le taux de victoire calculé ? ».*** Nous étudions leur efficacité en termes d'assurance de victoire, et leur influence sur le nombre de coups a jouer avant de gagner. Il s'agira dans un premier temps d'étudier la combinatoire du jeu, puis de proposer une modélisation du jeu afin d'optimiser les chances d'un joueur de gagner et enfin d'étudier les variantes du jeu plus réalistes.

Dans la dernière partie du projet, la technique d'exploration d'arbres de décision Monte-Carlo Tree Search sera appliqué sur un plateau de taille donné. Là encore, cette approche sera comparée à une autre : une méthode algorithmique visant à calculer une valeur numérique approchée en utilisant des techniques probabilistes contre un algorithme de recherche heuristique utilisé dans le cadre de la prise de décision.

### Pour exécuter le programme :

- Ouvrir un terminal en direction de la répertoire du jet et taper : `jupyter notebook`
- L'analyse et la visualisation du jeu sera mise dans le fichier puissance-4-game.ipynb
- Les autres fichiers contiennent les codes de base et fonctions de manipulation qui assurent le moteur du jeu

# Les règles du jeu

**Grille :** Le jeu se joue sur une grille verticale de 6 lignes et 7 colonnes.

**Deux joueurs** : Deux joueurs s'affrontent, l'un utilisant des jetons rouges et l'autre des jetons jaunes.

**Tour de jeu :** Les joueurs jouent à tour de rôle en plaçant un jeton de leur couleur dans une des colonnes. Le jeton tombe vers le bas de la colonne jusqu'à ce qu'il atteigne la première case vide.

**Objectif :** Le but du jeu est d'aligner quatre de ses jetons consécutifs (horizontalement, verticalement ou en diagonale) avant l'adversaire.

# Grandes étapes

## I.Combinatoire du jeu

### 1. Borne théorique des nombres de parties distinctes possibles à jouer

Supposons que la taille de notre plateau est de 10 colonnes par 10 lignes.

Si le plateau est fraiche, il est possible de choisir n'importe quel parmi les 10 colonnes libres à joueur. Par conséquent, le nombre de parties possibles à jouer est donc 10.

D'ailleurs, si le tableau n'est pas vide et sur lequel ses $n$ colonnes sont occupées (avec $n$ dans l'intervalle $[0, 10]$), le nombre de parties distinctes possibles à jouer sera $10 - n$.

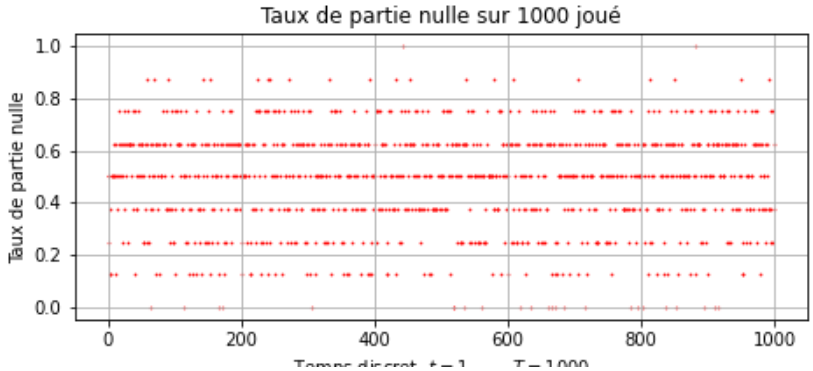
Donc, en terme théorique, pour un plateau de dimension $M \times N$, le borne supérieur des nombres de parties distinctes possibles à jouer est marginalisé à $N$.

### 2. Taux de parties nullles sur $N$ fois joué

Afin d'étudier la probabilité d'une partie nulle lorsque les deux joueurs jouent aléatoirement, on itère 1000 sessions sur le même plateau de jeu pour les mêmes joueurs. À chaque tour de boucle, à la fin d'une session, on calcule la probabilité $p_i$ du nombre de colonne vide $ColVide$ du plateau par rapport à sa largueur $L$ et les stocke dans une liste nommée $P$ en utilisant la formule suivant :

$$p_i = \frac{\sum ColVide_{j_{j \in [1 \dots L]}}}{L}$$

En observant le graph ci-dessous, on trouve que le taux de colonnes vides du plateau se trouve fréquemment dans l'intervalle de $[0.4, 0.65]$.




### 3. Étude de la distribution du nombre de coups avant une victoire

#### Processus

Afin d'étudier la distribution du nombre de coups avant une victoire lorsque les deux joueurs jouent aléatoirement en différenciant selon que ce soit le premier ou le deuxième joueur qui gagne, nous regénérons 1000 fois le moteur automatique du jeu.

Après avoir obtenu les résultats et passé par des traitements afin de raccourcir les opérations et de créer des bases de données nécessaires, nous trouvons bien que le nombre de coups avant une victoire suit la Loi Binomiale.

#### Loi binomiale

En théorie des probabilités et en statistique, la loi binomiale modélise la fréquence du nombre de succès obtenus lors de la répétition de plusieurs expériences aléatoires identiques et indépendantes.

Plus mathématiquement, la loi binomiale est une loi de probabilité discrète décrite par deux paramètres : $n$ le nombre d'expériences réalisées, et $p$ la probabilité de succès. Pour chaque expérience appelée épreuve de Bernoulli, on utilise une variable aléatoire qui prend la valeur $1$ lors d'un succès et la valeur $0$ sinon. La variable aléatoire, somme de toutes ces variables aléatoires, compte le nombre de succès et suit une loi binomiale. Il est alors possible d'obtenir la probabilité de $k$ succès dans une répétition de $n$ expériences :

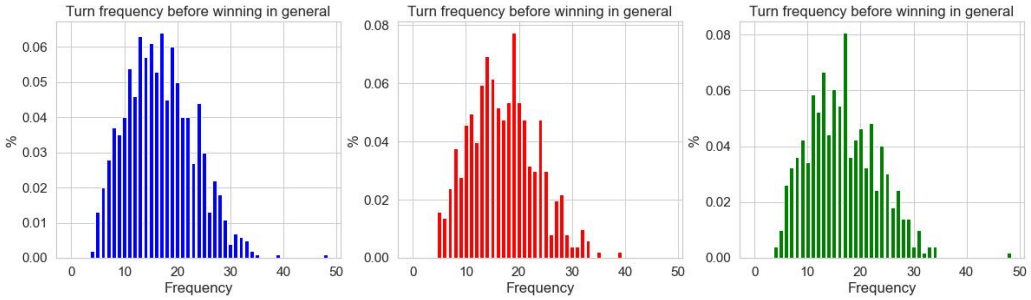
$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Cette formule fait intervenir le coefficient binomial $\binom{n}{k}$ duquel provient le nom de la loi.

## Explication

On cherche maintenant à modélise la fréquence du nombre de coups avant une victoire lors de la répétition de 1000 expériences aléatoires identiques et indépendantes sur un plateau de jeu. Dans le cadre du jeu, cette fréquence suit donc une distribution binomiale avec $n = 15 \times 15$ et $k = 4$.

L'étude de la distribution du nombre de coups avant une victoire renvoie une graphe prenant la forme ci-dessous :

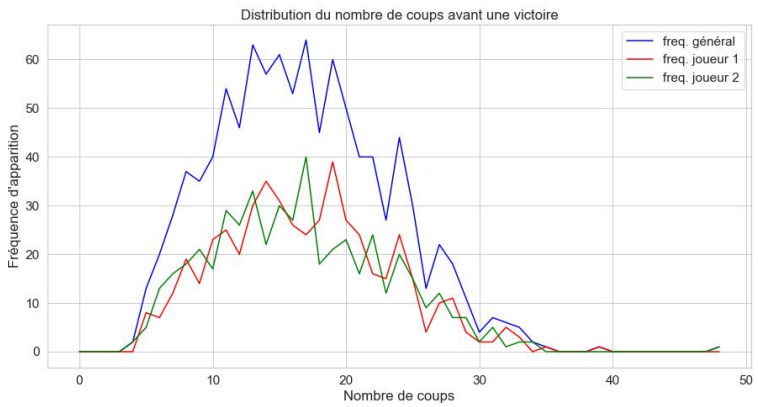



En observant le graphe, on trouve que le nombre de coups qui possède la fréquence d'apparition la plus grande est 16 pour le premier joueur, et 13 pour l'autre.

Dans ce cas-là, le nombre de coups qui possède la fréquence d'apparition la plus grande est 13.

En classifiant les fréquences selon la victoire de chaque joueur, on obtient le graphe ci-dessous, dont :

- la ligne *bleue* représente *la fréquence des nombres de coups en général*
- celle en *rouge* pour la *fréquence des nombres de coups par le premier joueur*
- et celle en *verte* pour *l'autre*



En observant le graphe, on trouve que le nombre de coups qui possède la fréquence d'apparition la plus grande est 16 pour le premier joueur, et 13 pour l'autre.

## II.Bras stochastiques, de Bernoulli

Fonction de simulation : `binary_gain` (classe `BDMC`)

Les récompenses de tels bras, notées $r_k(t)$ pour le bras $k$ à l'instant $t$, sont tirées de facons identiquement distribuées et indépendants, selon une loi de Bernoulli :

$$\forall t \in \mathbb{N}, \forall k \in \{1, \dots K\}, r_k(t) \in \{0,1\}, et\ r_k(t) \sim B(\mu_k)$$

## III.Technique statistique : Stratégie de 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 le cadre du jeu Puissance 4, on a implémenté cet algorithme de sorte qu'il retourne la moyenne des parties jouées au hasard pour chaque action possible, à l'aide des probabilités calculées via la fonction `MovesCounter`.

## IV.Phase E-T-C

La phase E-T-C, abréviation d'Exploration – Tirage – Confidence Bound *(d'où Exploration – Exploitation – Tirage)*, est une étape clé de l'algorithme UCB1 *(Upper Confidence Bound 1)*, qui est utilisé dans le contexte de l'apprentissage par renforcement, de l'optimisation sous incertitude, et particulièrement dans le problème du bandit multi-bras. L'objectif de l'algorithme UCB1 est de prendre des décisions séquentielles pour maximiser la récompense cumulative tout en explorant de manière efficace les différentes options disponibles.

Afin de faciliter cette phase du projet, on va d'abord générer les bases de données nécessaires pour l'analyse de graph qui suit. On divise la simulation en 6 bases de données, respectivement pour les algorithmes de : Baseline, Greedy, ε-Greedy, UCB ($\alpha = 0.5$), UCB ($\alpha = 1$), UCB ($\alpha = 2$), avec une horizon de 1000 essaies pour l'exploitation.

### Deux algorithmes d'optimisation

#### Baseline

Un algorithme qui choisit un bras de facon complètement uniforme, $A^1(t) \sim U(1, \dots K), \forall t$, à chaque instant $t \in \mathbb{N}$, via la fonction `baseline` appartenant à la classe `BDMC` *(l'abréviation de `Bandits-Manchots`).*

#### Gloutons

- Greedy : Un algorithme assez naif, qui utilise un estimateur empirique $\widehat{\mu_k}(t) = \frac{X_k(t)}{N_k(t)}$ de la moyenne de chaque bras, et tire $A^2(t) \in argmax_k \widehat{\mu_k}(t)$ à chaque instant $t \in \mathbb{N}$. Ici, $X_k(t) = \sum_{\tau=0}^{t} \mathbb{I}\,(A(\tau) = k) r_k(\tau)$ compte les récompenses accumulées en tirant les bras $k$, sur les instants $t = 0, \dots \tau$. Et $N_k(t) = \sum_{\tau=0}^{t} \mathbb{I}\,(A(\tau) = k)$ compte le nombre de sélection de ce bras $k$. Via la fonction `greedy` de la classe `BDMC`.
- $\varepsilon$-Greedy : après une première phase d'exploration optionelle, à chaque itération : avec une probabilité $\varepsilon$ on choisit au hasard uniformément parmi les actions possibles, avec une probabilité $1 - \varepsilon$ on applique l'algorithme greedy : $a_t = argmax_{i \in \{1 \dots N\}} \widehat{\mu_t^i}$. Cet algorithme explore continuellement.

## V.Approche fréquentiste, UCB1, « Upper Confidence Bound »

Il s'agit d'une améloration de l'algorithme précédent, où on utilise un autre indice.

Au lieu d'utiliser la moyenne empirique $g_k(t) = \hat{\mu}_k(t) = \frac{X_k(t)}{N_k(t)}$ et $A(t) = argmax_k g_k(t)$, on utilise une borne supérieure d'un intervalle de confiance autour de cette moyenne :

$$g'_k(t) = \hat{\mu}_k(t) + \sqrt{\alpha \frac{\log(t)}{N_k(t)}}$$

Et cet indice est toujours utilisé pour décider le bras à essayer à chaque instant :

$$A^{UCB1}(t) = \underbrace{argmax_k}_{k} g'_k(t)$$

Il faut une constante $\alpha \geq 0$, qu'on choisi $\alpha \geq \frac{1}{2}$ pour avoir des performances raisonnables. $\alpha$ contrôle le compromis entre exploitation et exploration, et ne doit pas être trop grand. $\alpha = 1$ est un bon choix par défaut.

Dans le cadre de notre projet, $\alpha = 2$.

### 1. Simulations

Dans cette section, on va aborder la comparaison des 4 algorithmes définies ci-dessus.

On va considérer qu'ils prennent tous deux arguments, le premier une liste des récompenses moyennes estimées pour chaque levier *(la liste des $\hat{\mu}^i$)* et le deuxième le nombre de fois où chaque levier a été joué *(la liste des $N(i)$)*.

On affichera danss dess graphiques au cours du temps $t = 0, \dots T$ pour un horizon $T = 1000$ étapes :

- Leurs taux de sélection du meilleur bras $k^*$
- Leurs récompenses accumulées $R(t) = \sum_{\tau=0}^{t} \sum_{k=1}^{K} X_k(\tau) \mathbb{I}(A(t) = k)$, pour chaque algorithme
- Les récompenses moyennes
- Leurs regret $\blacksquare(t) = \mu^* t = R(t)$. On souhaite maximiser $R(t)$, donc minimiser $\blacksquare(t)$. Les algos ont typiquement un regret logarithmiques, ie $\blacksquare(T) = \blacksquare(log(T))$.

En court terme, le but de cette partie est de comparer ces différents algorithmes afin d'en déduire celui qui est le plus efficace, c'est-à-dire, celui dont le regret *(différence entre le gain maximal espéré et le gain du joueur)* est minimal.

### 2. Un problème à 10 bras

Dans notre projet, on considère progressivement un plateau de jeu de dimension 10 lignes et 10 colonnes. Le nombre de leviers possible sera donc 10, et on souhaite déterminer le meilleur bras à choisir à chaque déplacement.

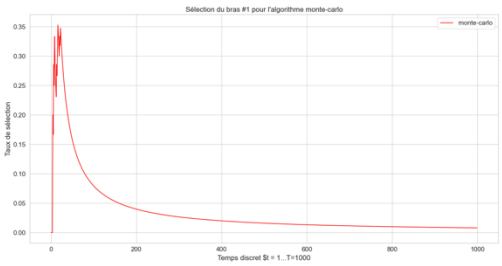
#### Taux de selection

À partir des graphes produits ci-dessous, on constate qu'à travers de 1000 étapes, les taux de sélection de Monte-Carlo et de Baseline se trouvent dans l'intervalle respectivement de $[0,0.35]$ et $[0,0.25]$ et ont tendance de diminuer drastiquement après quelques tour de boucle en comparant avec son départ.

Plus concrètement, le taux de sélection de Monte-Carlo a tombé de son plus haut point à 0.35%, à seulement près de 0.05% au 150è itérations. Depuis, il continue à baissé légèrement jusqu'à 0.01% à la fin de 1000 sessions.

De même, commencé au meilleur taux de sélection à 0.25%, après environ 50-70 étapes, celui de Baseline a atteint le point le plus bas d'au dessous de 0.05%. Pourtant, il a récupéré au fur à mesure jusqu'à arriver à plus de 0.1% à la fin du programme.

*Spécification concernant les graphes : Monte-Carlo (gauche) et Baseline (droite).*

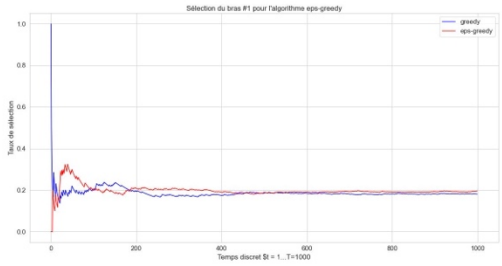


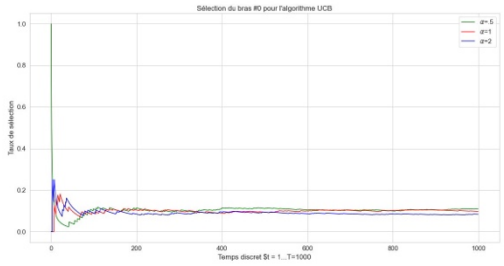


D'autre côté, une convergence est considérée dans les graphes des algprithmes de gloutons et d'UCB. Ces deux taux de sélection varie progressivement entre $[0,1]$ et se converge rapidement respectivement proche de 0.2% et 0.1% après seulement à priori 50 étapes.

À noter que l'algorithme UCB est divisé en 3 sous-problèmes basant sur la valeur d'$\alpha \in \{0.5, 1, 2\}$.

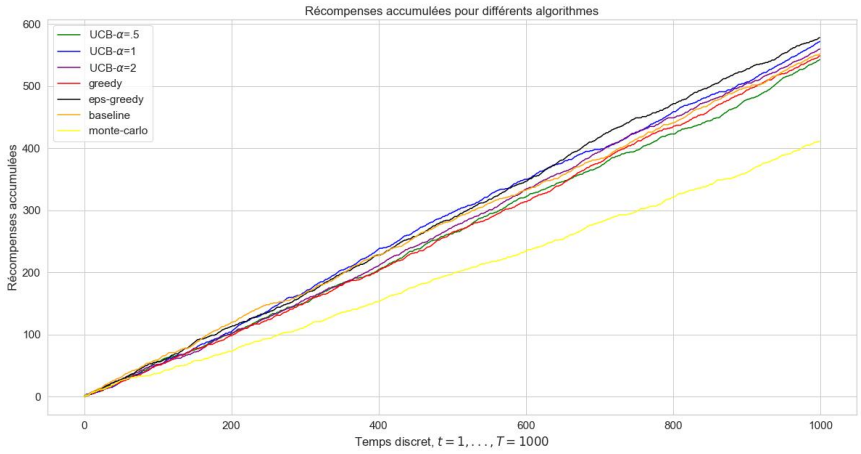
*Spécification concernant les graphes : Gloutons (gauche) et UCB1 (droite).*



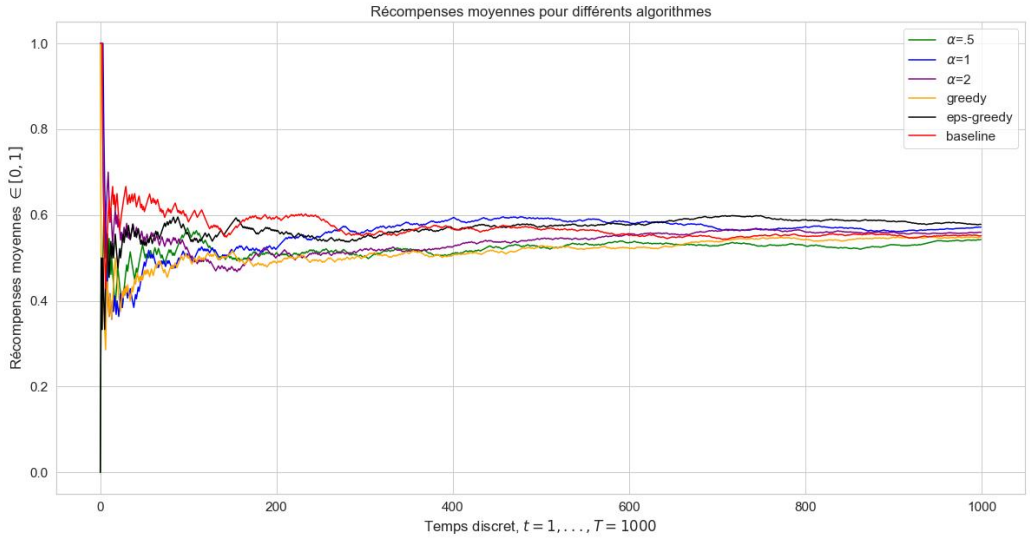

## 3. Taux de récompenses accumulées

En observant le graphe ci-dessous, on trouve que les récompenses accumulées de tous les algorithmes ont tendance d'augmenter à travers de 1000 itérations. Tandis que la valeur maximale de la plupart des techniques appliquées peut arriver jusqu'à proche de 600 point, celle de Monte-Carlo reste les plus faibles pendant cette période, qui m'atteint que 400 à la fin du programme.



## 4. Taux de récompenses moyennes

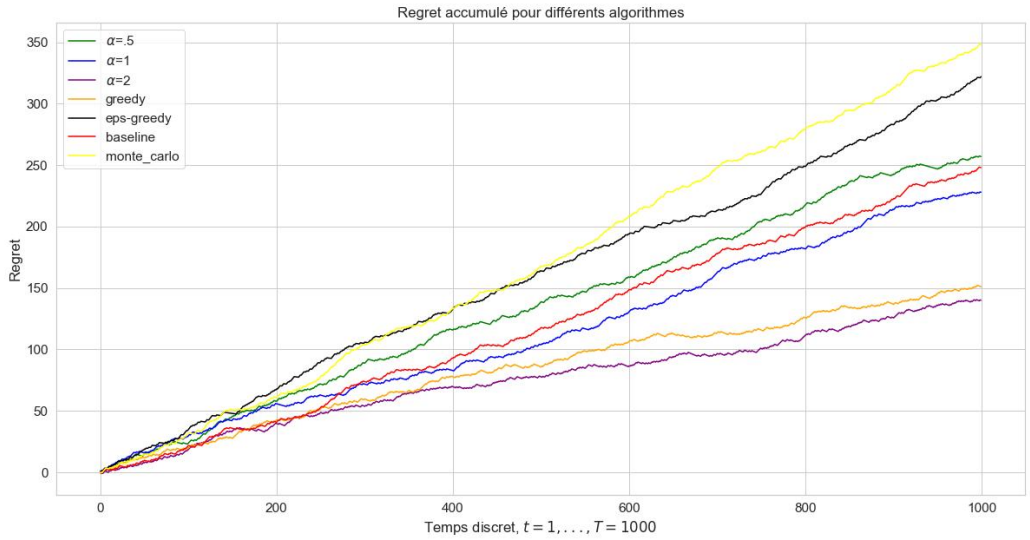
Pour les récompenses moyennes de chaque techniques utilisées, tous les courbes suivent la même tendance que pour le taux de sélection des gloutons et d'UCB, qui se converge après juste 50 itérations, autour de 0.575% à 0.6% à l'arrêt de boucle.




## 5. Taux de regret

Après avoir étudié le taux de sélection et de récompenses de tous les algorithmes, c'est le temps de jeter un coup d'oeil sur leur taux de regret, qui semble comme un élément important dans l'étude du meilleur technique. En observant le graphe généré à partie de la fonction `affiche_regret` de la classe `Graphic`, on constate que cette propriété augmente au fur et à mesure du temps.

En court terme, le taux de regret de Monte-Carlo atteint un sommet de 360 points, à l'opposé de celui d'UCB de valeur d'$\alpha = 1$.



### 6. Moyenne et Écart-type

*Table 1 - Taux de récompenses accumulées*

| Algorithme | Moyenne | Écart-type |
|---|---|---|
| **Baseline** | 0.552 | 0.4972886485734417 |
| **Greedy** | 0.549 | 0.49759320734913576 |
| **$\varepsilon$-Greedy** | 0.578 | 0.4938785275753543 |
| **UCB - $\alpha = 0.5$** | 0.543 | 0.4981475684975286 |
| **UCB - $\alpha = 1$** | 0.572 | 0.494788843851597 |
| **UCB - $\alpha = 2$** | 0.56 | 0.4963869458396343 |
| **Monte-Carlo** | 0.411 | 0.49201524366629135 |

À partir du tableau ci-dessus, on trouve que le taux de récompenses de tous les algorithmes sont relativement concentrées autour de leur moyenne. En d'autres termes, lorsque l'écart type est petit par rapport à la moyenne, cela signifie que les données sont relativement homogènes et peu dispersées par rapport à la moyenne.

### 7. Conclusion

Après l'étude de 4 propriétés des algorithmes en étude, on peut dire que UCB prend une place du meilleur algorithmes à choisir afin de détecter le levier qui nous aide à rapprocher les victoires; et que la stratégie de Monte-Carlo reste moins efficace que tous les autres techniques.

## VI.Arbre d'exploration et UTC

Dans cette partie, on implémentera progresivement une méthode utilisée dans l'apprentissage par renforcement et la recherche de l'arbre de décision afin de répondre au dilemme d'exploration/exploitation, pour savoir s'il vaut mieux continuer à lancer des simulations aléatoires sur une action plus performante sur un temps $t$ afin de s'assurer de sa qualité ou vaut-il mieux explorer d'autres actions possibles.

Dans l'implémentation de la méthode d'UCT (une variante de UCB adaptée aux arbres de jeu), une simulatione qui suit fera jouer les joueurs aléatoires, soit nous vs. une machine virtuelle qui calcule les leviers à choisir basant sur les algorithmes simulées, soit 2 joueurs virtuels, l'un utilise la stratégie de Monte-Carlo, l'autre avec la technique pure d'UCT.

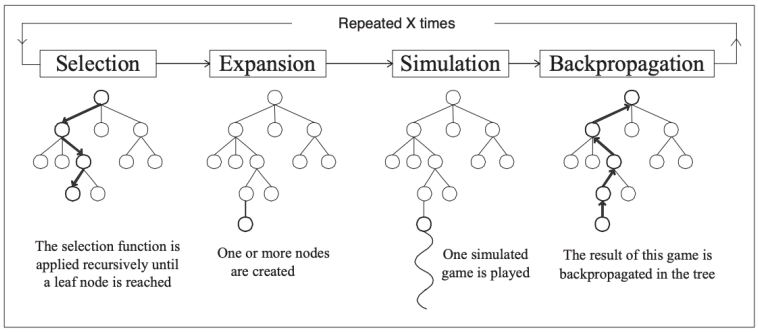



Figure 1: Outline of a Monte-Carlo Tree Search.

*Lien Youtube utile pour mieux comprendre les étapes de MCTS :*
*https://www.youtube.com/watch?v=UXW2yZndl7U*

## VII.Conclusion

Dans un premier temps, on a étudié la combinatoire du jeu, pour laquelle on a estimé les emplacements possibles qui amène à une victoire pour un joueur spécifé, et calculé le taux de parties nulles à la fin de chaque session.

Afin de pouvoir répondre à la question principale menitonnée au début du projet, on a implémenté d'abord la stratégie de Monte-Carlo, qui nous permet de déterminer le bras qui peut nous aider à optimiser le nombre d'essai avant une victoire. Cependant, souhaitant d'approfondir le moteur de jeu et de comprendre ses principes, 4 algorithmes d'optimisation ont été implémentés, parmi lesquels le fameux UCB1 garde pour toujours la meilleure techniques à choisir parmi tous les autres.

Après 3 étapes, dont la phase E-T-C, l'analyse de graphe et l'apprentissage par renforcement en utilisant l'arbre d'exploitation UCT, on est arrivée à une réponse pour notre dilemme de départ, dont l'application d'UCB persiste à peut battre la plupart des sessions dont l'autre joueur suit un technique autre que lui.