HONORÉ Alexandre

RIGONNET Arthur

BLANCHÉ Thomas



INFO 3 IA pour jeux à 2 joueurs

Table des matières

[Introduction 2](#_Toc134529429)

[I ) Recherches et démarrage du projet 2](#_Toc134529430)

[1. Recherches préliminaires 2](#_Toc134529431)

[2. Création git, cahier des charges, répartition des tâches 3](#_Toc134529432)

[II) Développement du puissance 4 3](#_Toc134529433)

[1. Version textuelle 3](#_Toc134529434)

[2. Version graphique 4](#_Toc134529435)

[III) Les intelligences artificielles 4](#_Toc134529436)

[1) Minimax 5](#_Toc134529437)

[2) Alpha-bêta 6](#_Toc134529438)

[3) MCTS 6](#_Toc134529439)

[IV) Statistiques 7](#_Toc134529440)

[1) Alpha-bêta VS Alpha-bêta 7](#_Toc134529441)

[2) Minimax VS Alpha-bêta 8](#_Toc134529442)

[3) MCTS VS Alpha-bêta 8](#_Toc134529443)

[Conclusion 9](#_Toc134529444)

[Bibliographie : 10](#_Toc134529445)

# Introduction

Dans le cadre des mini-projets de PEIP2, nous avons décidé de nous concentrer sur la mise au point de différentes formes d'intelligence artificielle appliquées au célèbre jeu de société "Puissance 4". Le concept est simple, aligner 4 pions de sa couleur pour gagner. Les pions peuvent être alignés horizontalement, verticalement ou diagonalement. Pour cela, les joueurs laissent tomber chacun à leur tour un jeton de leur couleur dans la colonne de leur choix. Malgré des règles aussi simples, les stratégies pour gagner sont nombreuses. Ainsi nous pouvons trouver un intérêt particulier au développement d’intelligences artificielles. En effet, il existe un nombre colossal de parties possibles. Soit environ 1,6x1013 parties et seulement 69 sont gagnantes. Calculer toutes les parties afin de gagner à coup sûr est donc impossible. C'est pourquoi nous avons recours aux intelligences artificielles pour développer des algorithmes de jeu capables d'apprendre et d'adapter leur stratégie en fonction des mouvements de l'adversaire.

Jeu de puissance 4 classique

L'objectif principal de ce projet était de nous immerger pour la première fois dans l'univers de l'intelligence artificielle afin d'acquérir une compréhension plus approfondie de son fonctionnement, de ses capacités et de ses limites. Nous avons ainsi développé trois programmes distincts : Minimax, Alpha-Bêta et Monte Carlo Tree Search, dans le but de les faire s'affronter les uns contre les autres et de recueillir et d’analyser différentes statistiques comme le temps d’exécution, le pourcentage de victoire etc…

Dans ce rapport final, nous allons exposer en détail le processus de développement de ce projet. Nous commencerons par présenter les recherches préliminaires que nous avons menées, ainsi que l'organisation que nous avons adoptée pour assurer une collaboration efficace. Nous aborderons ensuite les différentes étapes de création du jeu, à la fois sous sa forme textuelle et graphique, et nous expliquerons les méthodes que nous avons employées pour intégrer les trois types d'intelligence artificielle dans le jeu. Enfin, nous procéderons à une analyse détaillée des statistiques recueillies lors des différents matchs qui ont opposé les programmes entre eux.

# I ) Recherches et démarrage du projet

## Recherches préliminaires

Lorsque nous avons commencé à travailler sur ce projet, notre première étape a été de nous informer sur les différentes intelligences artificielles que nous allions développer. Pour ce faire, nous avons exploré diverses ressources et nous allons maintenant détailler comment chacune d'entre elles nous a été utile.

Pour les algorithmes "minimax" et "alpha-bêta", nous avons trouvé des vidéos très instructives d'Hugo Larochelle sur YouTube. Dans ces vidéos, il explique en détail comment fonctionnent ces deux types d'intelligences artificielles, en soulignant leurs avantages et leurs inconvénients. En outre, il nous a fourni des informations sur les propriétés de ces deux algorithmes, tels que leur complexité en temps et en mémoire.

Pour comprendre le fonctionnement des Monte Carlo Tree Search, nous avons exploré différentes ressources, et en particulier, nous avons trouvé une vidéo très instructive de Random Coder qui explique en détail le fonctionnement de cet algorithme. Dans cette vidéo, il fournit une description complète de l'algorithme, ainsi qu'une implémentation en python de l'intelligence artificielle pour le jeu du puissance 4. Nous avons trouvé cette vidéo très utile pour comprendre les concepts clés des Monte Carlo Tree Search et leur application pratique. En outre, Random Coder a également réalisé un tutoriel écrit qui nous a aidé à mieux comprendre les concepts expliqués dans la vidéo. Ce tutoriel écrit nous a permis de revenir sur les points clés et de mieux comprendre certains aspects plus complexes de l'algorithme. Nous avons également trouvé que la combinaison de ces deux ressources nous a permis de bien saisir les différents concepts liés aux Monte Carlo Tree Search et leur application au jeu du puissance 4.

## Création git, cahier des charges, répartition des tâches

Lors de la réalisation de ce projet, nous étions trois et nous avons dû nous organiser efficacement pour travailler ensemble. L'un des premiers défis auxquels nous avons été confrontés a été de trouver un moyen de travailler simultanément sur le projet. Heureusement, grâce à nos cours d'informatique du semestre quatre, nous avons appris à utiliser la plateforme GitHub, qui s'est révélée être une solution idéale. GitHub est une plateforme de développement collaborative en ligne qui permet à plusieurs développeurs de travailler simultanément sur le même projet. Elle fournit également des outils de gestion de versions et de suivi de bugs pour faciliter la collaboration et la coordination. En outre, GitHub est une solution totalement gratuite, ce qui était un avantage considérable pour notre projet. En utilisant GitHub, nous avons ainsi pu créer un dépôt de code commun pour le projet, où nous avons pu partager et collaborer sur les différents éléments du code. Grâce à cette solution, nous avons pu travailler efficacement en équipe et gérer le développement du projet de manière collaborative.

Une fois ce problème résolu, nous nous sommes concentrés sur la réalisation d'un cahier des charges pour nous donner des objectifs clairs sur ce que nous allions réaliser. Le cahier des charges a été une étape essentielle pour la planification et l'organisation du projet, car il a défini les objectifs, les moyens, les délais, les fonctionnalités, les exigences et les contraintes de l'application que nous devions développer. Le cahier des charges a également été utile pour nous permettre de suivre notre avancement tout au long du projet. Nous avons pu nous référer à ce document à tout moment pour nous assurer que nous étions sur la bonne voie et que nous respectons les délais et les spécifications du projet. Le document est par ailleurs disponible en annexe de ce rapport.

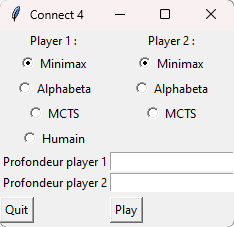
# II) Développement du puissance 4

## Version textuelle

Pour commencer la réalisation de ce projet, nous avons décidé d'utiliser le langage de programmation Python, qui est largement utilisé dans le domaine de l'intelligence artificielle et que nous voulions approfondir. Avant de pouvoir développer des intelligences artificielles pour le jeu de puissance 4, nous avons dû créer une version textuelle du jeu. Cette version fonctionne uniquement dans la console et dispose d'un affichage rudimentaire mais suffisant pour jouer. Afin de créer ce jeu, nous avons dû mettre en place plusieurs éléments clés. Tout d'abord, il nous a fallu un plateau de jeu qui est représenté par une liste contenant six autres listes de sept éléments chacune. Chaque liste représente une colonne du plateau. Nous avons également créé une fonction qui permet à un joueur de laisser tomber un jeton de sa couleur dans une colonne donnée, mais qui vérifie auparavant si la colonne n'est pas déjà remplie. Cette fonction est utilisable aussi bien par le joueur humain que par les intelligences artificielles. Afin d'assurer un affichage du plateau dans la console, nous avons créé une autre fonction qui parcourt le plateau et affiche, pour chaque case, un caractère différent en fonction du type de jeton (par exemple, '1' ou '2'). Enfin, la dernière fonction que nous avons créée est celle qui teste si un des joueurs a gagné. Cependant, cette fonction n'était pas si simple à implémenter car il fallait tester toutes les combinaisons possibles pour déterminer si quatre jetons étaient alignés. Ainsi, il fallait examiner chacune des colonnes, chacune des lignes et chacune des diagonales possibles. Nous avons aussi dû apprendre à utiliser des arguments lors de l'exécution de notre programme afin de choisir les deux joueurs qui s’affronteront, ainsi que la profondeur des algorithmes. Nous nous sommes servis du module python argparse.

## Version graphique

Le cahier des charges établi pour le projet inclut la réalisation d'une version graphique du jeu de Puissance 4. Pour cela,nous avons choisi d'utiliser la librairie graphique Tkinter, car certains d'entre nous l'avaient déjà utilisée lors de projets de spécialité NSI au lycée et en avaient donc une certaine familiarité. Nous avons ensuite développé une fonction spécifique chargée de l'affichage du plateau de jeu. Cette fonction commence par dessiner les cases vides lors de l'initialisation du programme. Ensuite, après chaque action d'un joueur ou d'une intelligence artificielle, la fonction dessine le jeton qui vient d'être déposé. L'implémentation de cette fonction a été essentielle pour la réalisation de la version graphique du jeu de Puissance 4, car elle permet aux joueurs de visualiser le plateau de jeu en temps réel et de suivre l'évolution de la partie.

De plus, comme stipulé dans le cahier des charges, nous avons également travaillé sur le développement d'un menu pour permettre aux utilisateurs de choisir les deux joueurs pour la prochaine partie, ainsi que la profondeur des intelligences artificielles si celles-ci sont choisies. Dans ce but, nous nous sommes appuyés sur la documentation française de Tkinter pour concevoir un menu fonctionnel et ergonomique pour les utilisateurs. La mise en place de ce menu a été une étape importante pour permettre une utilisation simplifiée de l'application, car elle permet aux utilisateurs de personnaliser leurs parties selon leurs préférences.

Menu graphique

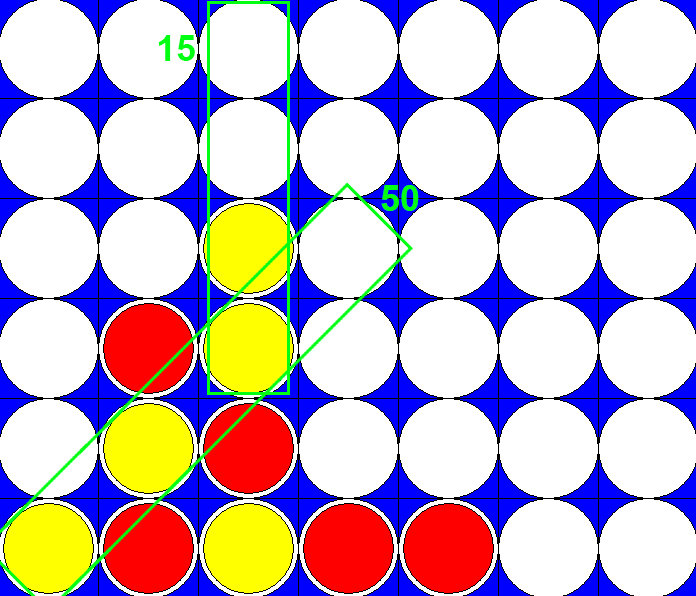
# III) Les intelligences artificielles

Affichage avec Tkinter

## Minimax

Une image contenant diagramme

Description générée automatiquementL'algorithme Minimax est un algorithme de décision qui est souvent utilisé dans les jeux à deux joueurs à tour de rôle. Il fonctionne en construisant un arbre de tous les coups possibles, sur plusieurs tours de jeu. Une fois que l'algorithme a simulé les parties jusqu'à un certain nombre de tours de jeu, il évalue l'état actuel de la partie pour voir s'il est favorable ou non. Ensuite, les valeurs de toutes les parties possibles sont remontées dans l'arbre afin de jouer le coup le plus optimal.

Cette image représente un jeu dans lequel seuls trois coups différents sont possibles. Les rectangles gris représentent les coups joués par l'adversaire, tandis que les ronds bleus représentent les coups joués par l'intelligence artificielle. Les valeurs marquées dans les rectangles gris du bas sont les évaluations du plateau de jeu. Lorsque les valeurs des nœuds de l'adversaire sont remontées, on garde uniquement le moins bon. Puis, lorsque les valeurs des nœuds de l'intelligence artificielle sont remontées, on ne garde que la plus grande valeur, d'où le nom Minimax. On suppose donc lors de la remontée de ces valeurs que l'adversaire va jouer le coup optimal.

Évaluation de Minimax (en jaune)

Schéma de Minimax

La principale difficulté de cet algorithme est la fonction d'évaluation du jeu. Pour cela, les alignements de quatre pions possibles sont récupérés et un score est attribué en fonction du nombre de places libres, des pions de l'IA et des pions de l'adversaire. Cette fonction d'évaluation est cruciale pour l'efficacité de l'algorithme, car elle détermine la façon dont l'algorithme Minimax évalue les différentes parties possibles et prend des décisions en conséquence.

## Alpha-bêta

L’algorithme alpha-bêta est considéré comme une amélioration de l’algorithme Minimax. Tout comme Minimax, Alpha-bêta utilise un arbre pour représenter les coups possibles sur plusieurs manches. Cet arbre est constitué de nœuds représentant les différents états possibles du jeu et de branches représentant les coups possibles à partir de chaque nœud.

L'objectif principal de l'algorithme Alpha-bêta est de réduire le nombre de branches à explorer afin de pouvoir explorer plus de manches possibles. Pour cela, l'algorithme va comparer la valeur de la branche à venir avec celles qu'il a déjà explorées pour savoir s'il doit continuer de l'explorer ou non. Il va utiliser une fonction d'évaluation pour déterminer si le plateau de jeu lui est favorable. Si la valeur de la branche n'est pas assez bonne, Alpha-bêta ne va pas continuer à explorer cette branche, car cela ne lui permettrait pas d'optimiser sa recherche.

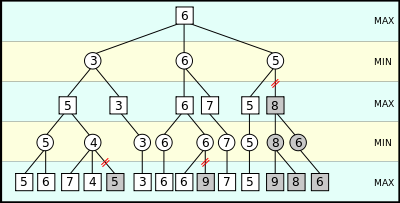
Cette méthode lui permet de prévoir plus de coups et d'explorer plus de manches avec autant de calculs que Minimax. Ainsi, Alpha-bêta est une méthode plus rapide et plus efficace pour trouver le coup optimal.

Schéma d'Alpha-bêta

# MCTS

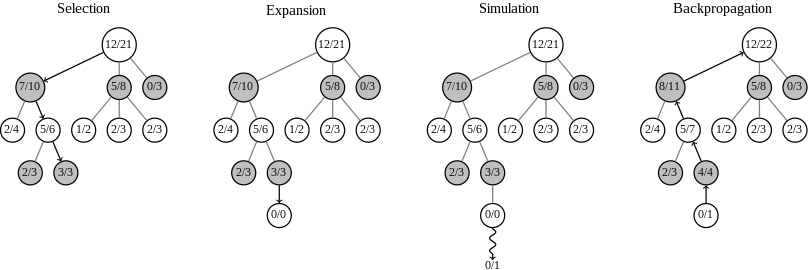
L'algorithme Monte Carlo Tree Search est un algorithme sophistiqué qui fonctionne en explorant un arbre de possibilités. L'arbre est constitué de nœuds qui représentent les différentes configurations du jeu, chaque nœud étant associé à un nombre de simulation gagnantes et un nombre total de simulations. Les nœuds fils sont les configurations obtenues après l'ajout d'un jeton à la configuration de leur nœud père, tandis que les feuilles sont des nœuds sans fils qui peuvent représenter soit une configuration dont on n'a pas encore exploré les prochains coups possibles, soit une configuration finale de jeu.

L'algorithme se décompose en quatre étapes : la sélection, l'expansion, la simulation et la backpropagation. Pendant la sélection, on part de la racine et on parcourt successivement les enfants jusqu'à atteindre une feuille. Pour décider quel enfant sélectionner, on doit faire un compromis entre l'exploitation des choix prometteurs et l'exploration des nœuds à partir desquels peu de simulations ont été faites. Pour ce faire, on cherche à maximiser la valeur de la formule de l'UCT (Upper Confidence Bound 1 applied to Trees).

Cette formule s’écrit de cette manière :

* *w* est le nombre de parties gagnées depuis le nœud
* *n* est le nombre de fois où le nœud a été visité
* N est le nombre de fois où le nœud père a été visité
* *c* est le paramètre d’exploration

Une fois que la phase de sélection est terminée, nous passons à la phase d'expansion. Cette phase consiste à créer les nœuds enfants de la feuille choisie, si celle-ci n'est pas une configuration finale du jeu. En d'autres termes, nous ajoutons de nouvelles configurations au sein de l'arbre de possibilités, qui seront explorées lors des prochaines itérations de l'algorithme. Après avoir étendu notre arbre, nous passons à la phase de simulation. À ce stade, l'algorithme joue une partie aléatoire à partir du nœud choisi jusqu'à ce qu'il atteigne une victoire, une égalité ou une défaite. Enfin, l'algorithme va faire remonter le résultat de la partie qu'il vient de simuler dans les nœuds père, en incrémentant le nombre de parties simulées et le nombre de parties gagnées. Cette phase de backpropagation permet de mettre à jour les statistiques de tous les nœuds visités pendant la sélection. Les résultats de cette phase seront ensuite utilisés lors des prochaines itérations de l'algorithme pour guider la sélection des nœuds les plus prometteurs.



Une image contenant regarder

Description générée automatiquement

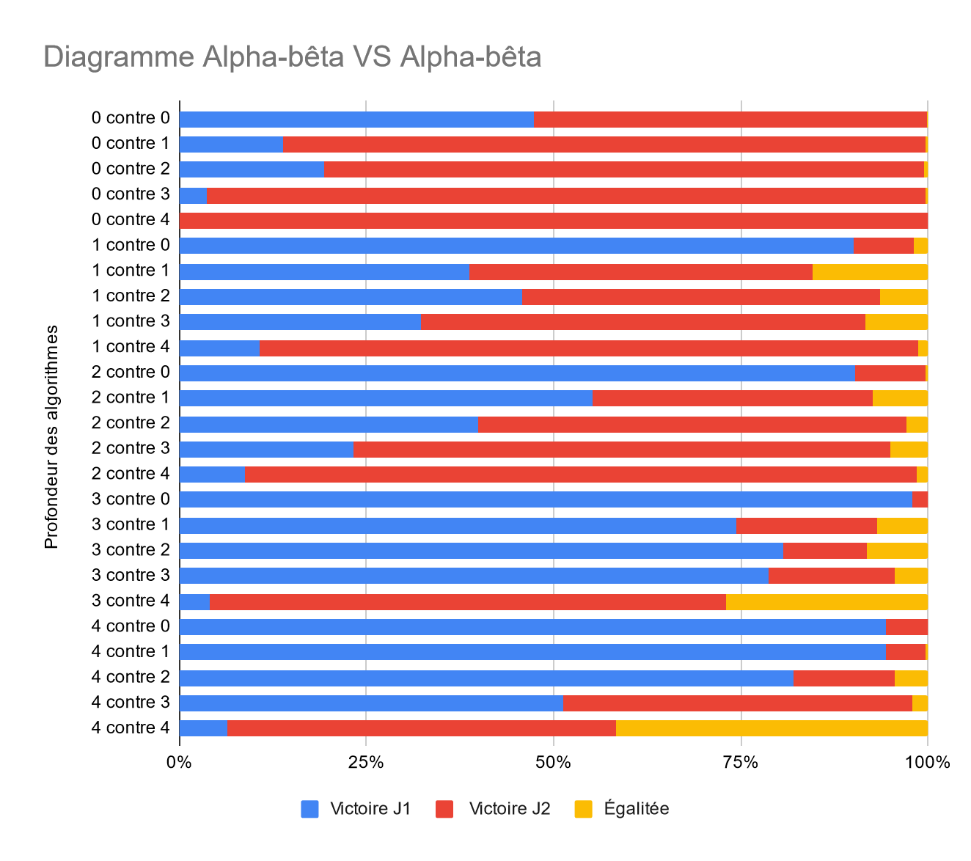
Schéma de Monte Carlo Tree Search

# IV) Statistiques

L'un des éléments cruciaux de notre projet était de collecter des statistiques pour mesurer les performances de différentes intelligences artificielles. Pour atteindre cet objectif, nous avons opté pour la méthode de faire s'affronter les différentes intelligences artificielles entre elles. Cependant, nous avons rencontré un obstacle majeur: comment effectuer ces affrontements de manière rapide et efficace ? Pour résoudre ce problème, nous avons cherché une solution et avons finalement décidé d'utiliser le module Python "concurrent.futures". Ce module offre une fonctionnalité permettant de simuler plusieurs matchs en même temps en utilisant plusieurs cœurs du processeur de l'ordinateur. En conséquence, nous avons été en mesure de collecter un grand nombre de données sur les performances de chaque IA dans une période relativement courte. Grâce à ces données collectées, nous allons maintenant être en mesure d'analyser les résultats et d'en tirer des conclusions significatives sur les performances des différentes intelligences artificielles impliquées dans notre projet. Nous avons choisi de présenter les affrontements Alpha-bêta contre Alpha-bêta, Minimax contre Alpha-bêta et MCTS contre Alpha-bêta. Minimax étant seulement une version moins optimisée de l’algorithme Alpha-bêta, nous avons décidé de ne pas le faire jouer contre MCTS ni contre lui-même.

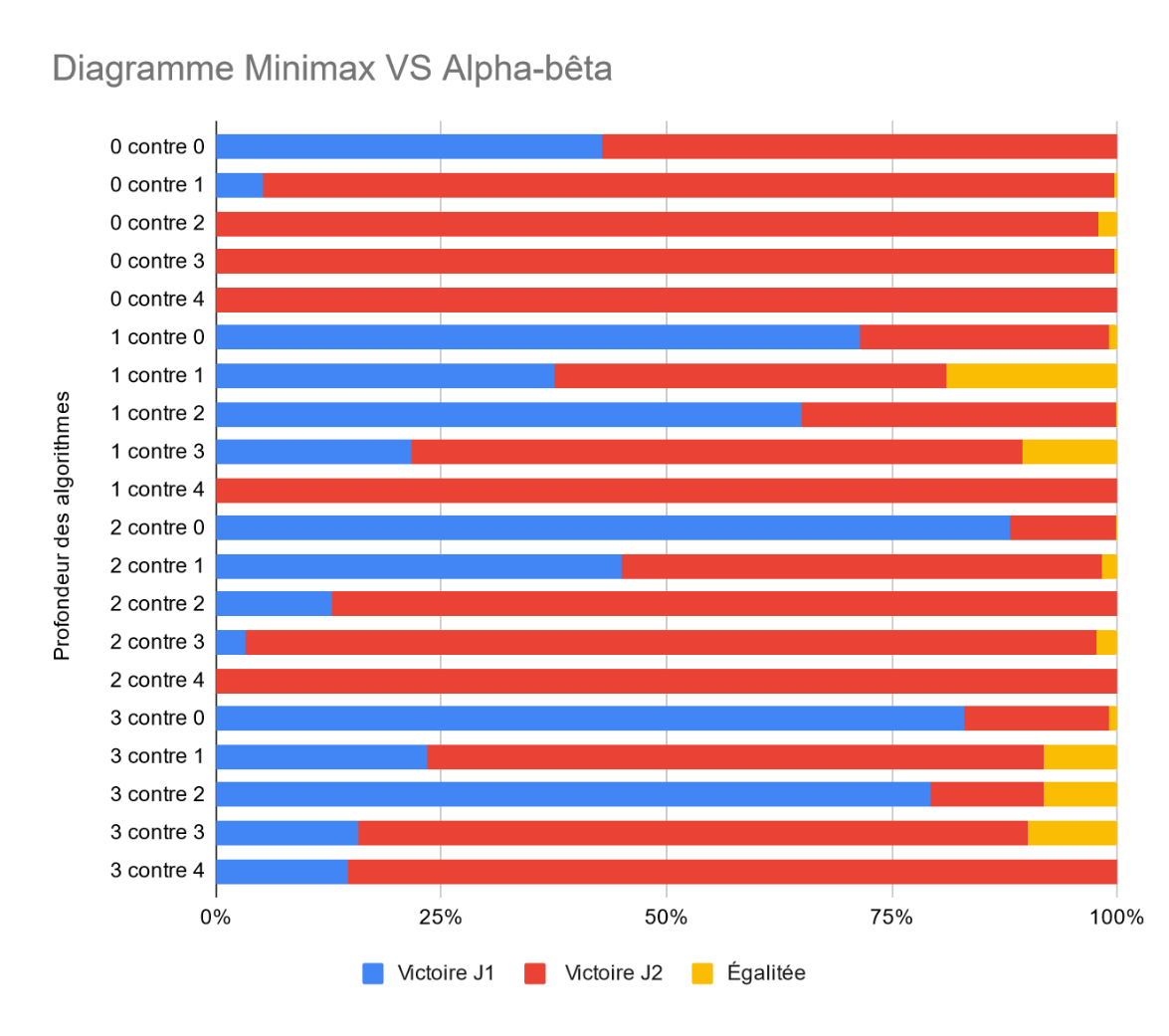
## Alpha-bêta VS Alpha-bêta

Nous avons fait s'affronter l'algorithme alpha-bêta contre lui-même à des profondeurs différentes. Dans le cadre de ces tests, nous avons simulé un échantillon de parties dont la taille variait en fonction de la profondeur de l’algorithme. Pour les profondeurs les plus importantes, nous avons simulé 250 parties, tandis que pour les profondeurs les plus faibles, nous avons augmenté l'échantillon à 1000 parties. Les résultats de ces simulations étaient plutôt prévisibles. En effet, nous avons observé que, pour une profondeur plus importante, l'algorithme gagnait plus souvent. De plus, à profondeur équivalente, le taux de victoire était similaire. Cependant, nous avons également noté une tendance à l'égalité de plus en plus importante lorsque la profondeur augmentait. Cette tendance s'est manifestée de manière significative lors de simulations à la profondeur 4, où le taux d'égalité atteignait 41 %.



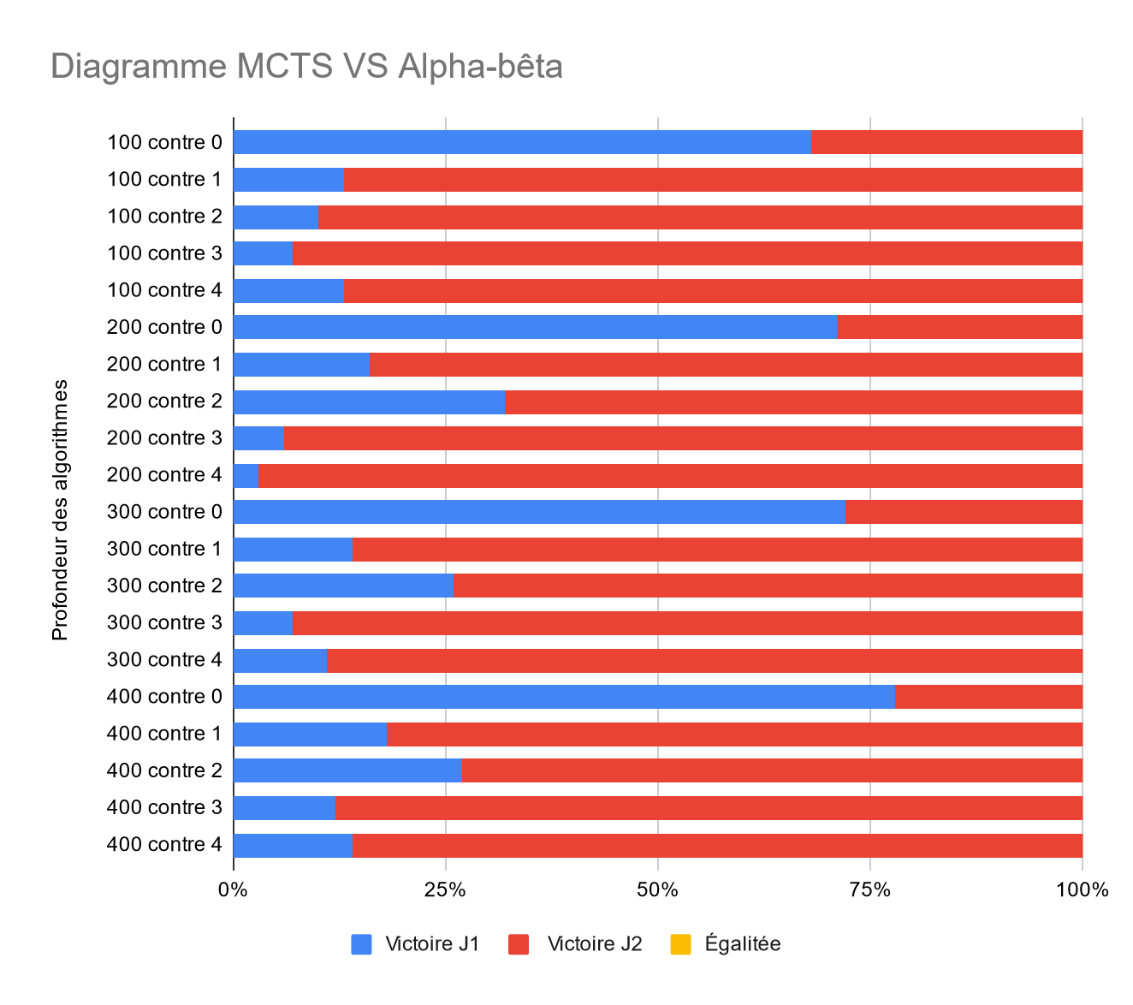
## Minimax VS Alpha-bêta

Nous avons ensuite décidé de faire s’affronter Minimax et Alpha-bêta, à des profondeurs différentes. Nous avons généré un échantillon de parties assez large, allant de 300 pour les profondeurs les plus importantes à 1000 pour les plus faibles. Nous avons pu constater que, globalement, les résultats étaient prévisibles : lorsque la profondeur de l'un des deux algorithmes augmentait, il devenait plus efficace que l'autre. Cependant, certains affrontements nous ont posé des difficultés de compréhension. Par exemple, Minimax, avec une profondeur de 3, a perdu contre Alpha-bêta avec une profondeur de 1, ce qui peut sembler contre-intuitif. Cela est peut-être dû au fait que ces deux algorithmes jouent en considérant que leur adversaire joue de façon optimale. Lorsque la profondeur est égale, les pourcentages de victoires sont équivalents, ce qui est tout à fait normal car les programmes sont identiques. Cependant, nous avons observé que Alpha-bêta s'exécute plus rapidement que Minimax, ce qui nous a permis de tester ce dernier en profondeur 4 alors que nous nous sommes arrêtés en profondeur 3 pour Minimax.



## MCTS VS Alpha-bêta

Nous avons finalement décidé de faire s'affronter l'algorithme MCTS et Alpha-bêta. Cependant, contrairement à nos précédents tests, nous avons réduit considérablement la taille de notre échantillon de parties pour cette nouvelle expérience, avec seulement 100 parties jouées à chaque fois. Cette décision a été prise car MCTS est un algorithme relativement lent, et le paramètre que nous pouvons lui donner est le temps de calcul qu'il utilise pour prendre chaque décision. Pour tester l'efficacité de MCTS par rapport à Alpha-bêta, nous avons effectué plusieurs tests en variant le temps de calcul accordé à MCTS, en allant de 1 seconde à 4 secondes, et en testant Alpha-bêta à des profondeurs allant de 0 à 4. Cependant, nous avons été surpris par les résultats de ces tests. En effet, nous nous attendions à ce que MCTS soit plus performant que Alpha-bêta. Cependant, dès lors qu'Alpha-bêta avait une profondeur supérieure à 0, il gagnait largement. Néanmoins, nous avons noté que l'augmentation du temps de calcul accordé à MCTS avait légèrement augmenté son taux de victoire. Nous supposons donc qu'avec un temps de calcul plus long, MCTS pourrait finir par devenir plus efficace qu'Alpha-bêta.



# Conclusion

Si nous devions décrire ce projet en quelques mots, nous dirions qu'il a été extrêmement intéressant et très enrichissant. Nous avons eu l'opportunité de nous avons eu l'opportunité d'explorer les capacités des intelligences artificielles lorsqu'elles sont appliquées à la résolution de jeux à deux joueurs, ce qui nous a permis de mieux comprendre comment ces dernières fonctionnent. De plus, pour certains d'entre nous, ce projet a été l'occasion de consolider et d'approfondir nos compétences en programmation python, ce qui sera très bénéfique pour notre parcours scolaire et professionnel. Cependant, bien que nous aurions aimé pouvoir consacrer plus de temps à ce projet, la réalité est que nous avons été confrontés à une charge de travail importante dans les autres matières, ce qui nous a obligés à nous impliquer moins que ce que nous aurions souhaité. Malgré cela, nous sommes très satisfaits du travail que nous avons accompli et de ce que nous avons appris au cours de ce projet.

# Bibliographie :

* Kei Galli. (2017, Décembre). Connect 4 game Python <https://youtube.com/playlist?list=PLFCB5Dp81iNV_inzM-R9AKkZZlePCZdtV>
* Hugo Larochelle. (2013, Avril). Intelligence Artificielle [4.2] : Jeux à deux adversaires - algorithme minimax <https://youtu.be/QTw8VJzRW6g>
* Hugo Larochelle.(2013, Avril). Intelligence Artificielle [4.3] : Jeux à deux adversaires - élagage alpha-bêta <https://youtu.be/KS2QkHe-hpE>
* Random Coder.(2022, Décembre). Beating Connect 4 with Monte Carlo Tree Search! | Explanation + Code <https://youtu.be/EB-NJtNERBQ>
* Qi Wang.(2022, Décembre) Connect 4 with Monte Carlo Tree Search <https://www.harrycodes.com/blog/monte-carlo-tree-search>
* Module Argparse <https://docs.python.org/3/library/argparse.html>
* Documentation Tkinter <http://pascal.ortiz.free.fr/contents/tkinter/tkinter/>
* Module concurrent.futures <https://docs.python.org/3/library/concurrent.futures.html>