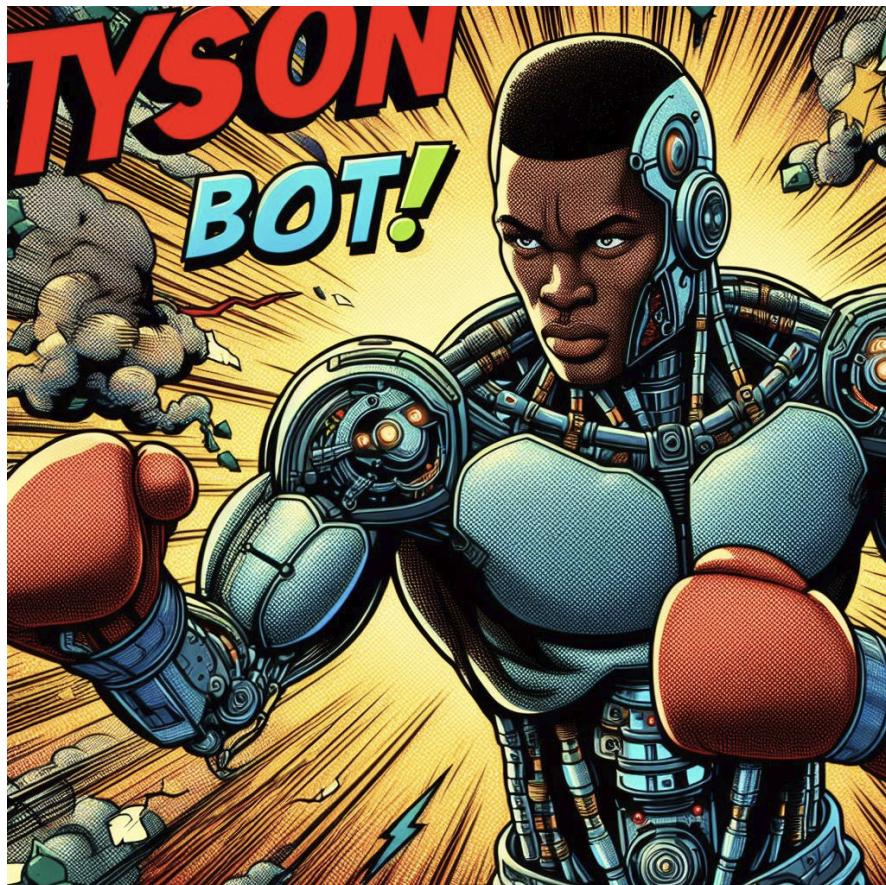


Projet d'Intelligence Artificielle : TYSOMBOT

Alexandre Untereiner

14 mars 2024



Résumé

Dans ce rapport, je présente mon projet consistant à développer un moteur d'intelligence artificielle pour le jeu traditionnel africain Awélé. Le jeu Awélé est un jeu de stratégie combinatoire abstrait qui oppose deux joueurs dans une lutte pour la récolte maximale de graines. Mon approche consiste à construire un modèle de prise de décision efficace basé sur un ensemble d'algorithmes d'intelligence artificielle, en utilisant des techniques d'analyse de données et de recherche opérationnelle. Je détaille ma méthodologie et les défis rencontrés dans ce rapport.

Table des matières

1	Introduction	3
2	Règles du Jeu	3
3	Recherche de Techniques sur le Jeu Awélé	3
3.1	Stratégies Utilisées dans le Jeu Awélé	4
3.2	les réseaux de neurones	4
3.3	Choix de l'algorithme MinMax	5
4	Développement de l'algorithme MinMax	5
4.1	MinMax et heuristique	5
4.2	problème sur la profondeur de recherche	5
4.3	Recherche d'heuristiques performantes	6
4.4	Optimisation de la profondeur de recherche	7
4.4.1	Triage des noeuds pour l'algorithme Alpha-Bêta	8
5	Résultats	9
6	Améliorations Futures de l'Algorithme	10
7	Conclusion	10

1 Introduction

L’Awélé, jeu de stratégie traditionnel africain. Avec ses règles simples mais ses possibilités stratégiques complexes, il offre un terrain intéressant pour l’exploration des algorithmes d’IA. Ce rapport présente mon projet visant à construire un bot capable de jouer à l’Awélé en utilisant des méthodes avancées d’intelligence artificielle.

Dans une première, je commencerai par décrire brièvement les règles du jeu Awélé, suivi par une présentation des objectifs et des contraintes du projet.

Le rapport détaillera mon approche pour résoudre ce défi. Je présenterai les algorithmes d’intelligence artificielle que j’ai choisis d’utiliser, ainsi que les méthodes d’analyse de données et d’apprentissage. Je expliquerai également la structure de mon bot et les choix de conception que j’ai faits. Enfin, je évaluerai les performances de mon bot et discuterai des résultats obtenus.

2 Règles du Jeu

Le jeu se joue sur un plateau de jeu comportant douze trous, avec deux rangées de six trous chacune, une pour chaque joueur. Chaque trou contient initialement quatre graines.



À tour de rôle, les joueurs prennent toutes les graines d’un trou de leur côté du plateau et les sèment une par une dans les trous suivants de leur rangée, puis de la rangée de leur adversaire, en suivant le sens de rotation (généralement anti-horaire). Si la dernière graine semée tombe dans un trou du côté de l’adversaire et que ce trou contient maintenant deux ou trois graines, le joueur capture ces graines et les ajoute à son score. Ce processus se poursuit jusqu’à ce qu’il n’y ait plus deux ou trois graines dans les trous.



Si un joueur sème 12 graines ou plus, le trou d’où proviennent ces graines est ”sauté” durant le semis. De plus, un coup qui affamerait l’adversaire en ne lui laissant plus aucune graine sur son côté n’est pas autorisé. La partie prend fin lorsque le total des graines restantes sur le plateau est inférieur à six ou lorsqu’un joueur ne peut plus jouer de coup valide, auquel cas ce joueur récupère toutes les graines restantes de son côté.

Les joueurs doivent donc adopter une stratégie astucieuse pour maximiser leur score tout en empêchant leur adversaire de faire de même.

3 Recherche de Techniques sur le Jeu Awélé

Avant de commencer le développement du bot, j’ai entrepris une phase de recherche pour explorer les différentes stratégies et techniques utilisées dans le jeu Awélé. J’ai étudié différant articles, publi-

cations de chercheurs et même télécharger un jeu sur Android "Awale Online - Oware Awari" pour expérimenter et comprendre les stratégies.

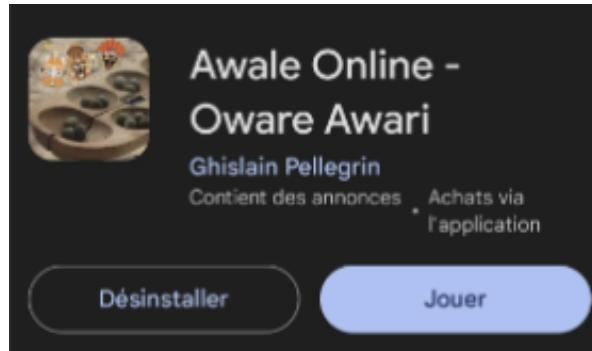


FIGURE 1 – Awale Online - Oware Awari

Cette recherche m'a permis de découvrir plusieurs stratégies populaires, telles que la stratégie de "capture maximale", où le joueur cherche à capturer autant de graines que possible à chaque coup, et la stratégie de "semi-défense", où l'accent est mis sur la défense tout en cherchant à maintenir une pression constante sur l'adversaire.

Ces recherches ont été cruciales pour orienter le développement de mon bot et choisir les algorithmes les plus appropriés pour la prise de décision. J'ai intégré ces connaissances dans la conception de mon IA.

3.1 Stratégies Utilisées dans le Jeu Awélé

De nombreuses stratégies sont utilisées par les joueurs d'Awélé. En voici quelques-unes :

- **Les stratégies de début de partie :** Ces stratégies visent à éviter les séquences de coups qui exposent le joueur à des captures multiples et à privilégier des séquences de coups plus sûres.
- **Les stratégies de milieu de partie :** Ces stratégies impliquent la préparation d'attaques en augmentant le nombre de graines dans ses trous pour réaliser des captures multiples. Elles comprennent également des tactiques de défense pour contrer les attaques de l'adversaire.
- **Les stratégies de fin de partie :** Ces stratégies visent à récolter des graines lorsque le nombre de graines restantes sur le plateau est faible en utilisant des pièges pour forcer l'adversaire à jouer dans des cases défavorables.

3.2 les réseaux de neurones

Lors du processus de décision pour concevoir le bot pour le jeu Awélé, j'ai envisagé diverses approches, y compris l'utilisation de réseaux de neurones. Cependant, après une analyse approfondie, j'ai décidé de ne pas opter pour cette technologie pour plusieurs raisons clés :

1. **Complexité et temps de développement :** Le développement et la formation d'un réseau de neurones pour jouer à Awélé auraient nécessité une exploration approfondie des architectures de réseaux adaptées, ainsi que des efforts considérables pour collecter et tester des données d'entraînement, avec au maximum une heure d'entraînement qui me semblait trop peu pour réaliser un réseau efficace qui dépend trop de la durée d'un match ou du jeu donné. Sans compter que cette approche aurait demandé un investissement en temps considérable, ce qui aurait pu retarder le développement de l'IA dans les délais impartis pour ce projet.
2. **Interprétabilité et transparence :** Les réseaux de neurones sont souvent perçus comme des boîtes noires en raison de leur complexité. Dans le contexte du jeu, il aurait été difficile de comprendre comment et pourquoi l'IA prend des décisions spécifiques, ce qui aurait pu rendre le processus de débogage et d'optimisation plus compliqué.

En fin de compte, bien que les réseaux de neurones offrent des capacités d'apprentissage avancées, j'ai décidé de me concentrer principalement sur la fonction de décision plutôt que sur l'apprentissage, même si j'avais initialement envie de tester cette approche.

3.3 Choix de l'algorithme MinMax

J'ai pris alors la décision d'utiliser l'algorithme MinMax pour concevoir l'intelligence artificielle de l'Awalé. Ce choix s'est basé sur plusieurs considérations pratiques et théoriques.

Tout d'abord, l'Awalé est un jeu à somme nulle, ce qui signifie que les gains d'un joueur correspondent aux pertes de l'autre joueur. L'algorithme MinMax est particulièrement adapté à ce type de jeu, car il permet de trouver la meilleure stratégie pour maximiser les gains tout en minimisant les pertes.

De plus, l'optimisation Alpha-Bêta, qui peut être appliquée à l'algorithme MinMax, permet de réduire efficacement l'espace de recherche en éliminant les branches qui ne contribuent pas à la décision finale. Cela rend l'algorithme MinMax plus efficace pour des jeux complexes comme l'Awalé, où le nombre de coups possibles peut être très élevé.

Un autre facteur important dans le choix de l'algorithme MinMax est le temps limité dont je disposais pour développer l'IA de l'Awalé suite au choix de réalisé seul le projet. En optant pour l'algorithme MinMax, j'ai pu me concentrer sur la conception et l'implémentation d'une solution fonctionnelle dans les délais impartis.

Bien que l'algorithme MinMax puisse ne pas être aussi sophistiqué que certains modèles d'apprentissage automatique, il offre une approche solide et bien établie pour la résolution de jeux comme l'Awalé. De plus, sa compréhension et sa mise en œuvre sont relativement accessibles, ce qui en fait un choix approprié pour ce projet individuel avec des contraintes de temps.

4 Développement de l'algorithme MinMax

4.1 MinMax et heuristique

Dans cette section, je vais décrire le processus de développement de l'algorithme MinMax pour l'intelligence artificielle de l'Awalé. Initialement, je me suis appuyé sur l'algorithme MinMax fourni dans le sujet du projet, en y ajoutant une nouvelle heuristique à celle existante qui prenait uniquement la différence de score, cette nouvelle heuristique prend en compte les mauvais trous, et vise à évaluer les positions où le bot est facilement attaqué que j'ai appellé **BetaOne**. cela ma donné de très bon résultat, soit 9 victoire sur 10, comparer à l'heuristique initial qui calculer uniquement la différence de score. Cependant ce n'est pas assez pour dire que mon IA est la meilleur.

4.2 problème sur la profondeur de recherche

Après avoir mis en place cette première version de l'algorithme **BetaOne**, j'ai tenté d'augmenter la profondeur de recherche pour améliorer les performances de l'IA. Cependant, j'ai rapidement constaté que l'algorithme fourni n'était pas assez rapide pour respecter la contrainte de temps de réponse de 100 ms imposée par le projet.

Face à cette difficulté, j'ai décidé d'analysé en profondeur le MinMax fourni, et j'ai pu remarquer que ça conception n'était pas optimal pour réaliser des tâches limités dans le temps.

Raisons pour lesquelles MinMax peu être améliore :

1. **Complexité du code** : Le code actuelle est complexe avec nombreuses boucles, et de manipulations de tableau, opération mathématique coûteuse. Plus de complexité implique généralement plus de temps d'exécution.
2. **Répétition d'opérations** : de plus certaines valeur sont recréer à chaque itération de la boucle récursive, comme la création d'un tableau de décisions et la copie de la grille de jeu. Ces opérations répétées peuvent ajouter un surcoût en temps.
3. **Structure du code** : bien que le code actuelle est récursive et généralement efficaces pour résoudre certains types de problèmes, ça structure implique des construction de nouvelle classe à chaque noeud, par exemple, pour un algorithme qui appliquerai juste une fonction récursif 1 million de fois, le code actuelle serait obliger de crée 1 million de fois une class qui hérite MinMaxNode, cette nuance ci ralenti fortement la rapidité de l'algorithme.

C'est avec la ferme attention de corriger ces défauts que j'ai décidé de entièrement retravailliez l'algorithme MinMax moi-même. Cette tâche s'est avérée ardue. Mais après plusieurs heures de travail et de nombreuses tentatives, j'ai finalement réussi à mettre en œuvre une version optimisée de l'algorithme MinMax deux fois plus rapide que l'ancienne version que j'ai nommée **Alpha2**.

Points améliorés de l'algorithme Alpha2 :

1. **Suppression de toutes les classes** : élimination de toutes l'utilisation de classes et de méthodes supplémentaires, simplifiant ainsi la structure du code et réduisant potentiellement les frais généraux associés à la création d'instances de classe.
2. **Création de variables statiques pour éviter de recréer des variables et des tableaux inutilement** : L'utilisation de variables statiques permet d'économiser des ressources en évitant la création répétée de nouvelles instances de variables à chaque appel de fonction ou d'itération de boucle.
3. **Simplification de l'algorithme** : simplification de l'algorithme en réduisant le nombre d'opérations répétitives ou coûteuses.

Ces améliorations contribuent à rendre le code plus efficace et plus rapide en réduisant la complexité et en optimisant l'utilisation des ressources disponibles. grâce à cette optimisation j'ai pu passer d'une profondeur de recherche de 6 à 8, ce qui veut dire que mon algorithme **Alpha2** est capable de prévoir deux coups de plus que mon premier algorithme **BetaOne** avec un taux de victoire de 100%. Bien évidemment, avec une telle méthode d'optimisation sur le code actuelle, certain morceau du code peut sembler redondant à d'autres endroits, et peut potentiellement être plus dur à maintenir si il y a une modification. mais j'ai accepté ce risque pour les quelques millisecondes essentielles que je peut gagner sur le temps d'exécution de la décision.

4.3 Recherche d'heuristiques performantes

Dans ma quête pour développer une intelligence artificielle compétitive, j'ai consacré du temps à la recherche d'heuristiques efficaces. J'ai exploré diverses stratégies, telles que le calcul des coups rentables comme les Krou, ou l'évitement des Krou adverses, ainsi que la prise en compte des trous qui peuvent mener à des gains importants en graines. J'ai également testé différentes combinaisons d'heuristiques dans le but de trouver la configuration optimale.

- **Diffscore** : Cette heuristique évalue la différence entre le nombre de graines dans les trous du joueur et celles de l'adversaire. Une différence positive indique une position favorable.
- **Coups rentables (Krou)** : Cette heuristique recherche les coups qui offrent des opportunités de captures multiples, comme les coups permettant de former des configurations Krou.
- **Évitement des Krou adverses** : Cette heuristique vise à éviter les coups qui permettent à l'adversaire de réaliser des captures multiples, en limitant les configurations qui pourraient conduire à des Krou adverses.
- **Trous favorables/défavorable** : Cette heuristique prend en compte les trous qui pourraient potentiellement mener à des gains importants en graines à court ou à long terme ou inversement en fonction du joueur, en privilégiant les coups qui exploitent ces opportunités.

Cependant, malgré mes efforts et mes expérimentations, je n'ai pas réussi à déterminer une heuristique qui se démarque comme étant nettement supérieure aux autres. Après des centaines de combats entre différentes versions de l'IA utilisant des heuristiques variées, je me suis retrouvé confronté à une conclusion déconcertante : chaque lancement de la compétition de 10 combats produisait un classement différent.

Ces résultats indiquaient qu'aucune stratégie n'était significativement plus forte que les autres. Chaque fois, les performances de l'IA variaient, soulignant ainsi la complexité du jeu et la difficulté de trouver une heuristique universellement performante. Cependant une approche possible pour trouver une heuristique s'en approchant aurait été de développé un algorithme génétique durant le temps d'apprentissage.

Exploration d'une approche avec un algorithme génétique

En envisageant les défis rencontrés dans la détermination d'une heuristique optimale pour le jeu Awélé, l'idée d'utiliser un algorithme génétique est apparue comme une piste prometteuse. L'objectif

était d'exploiter les principes de l'évolution pour trouver automatiquement une combinaison d'heuristiques performantes.

Bien que l'idée d'un algorithme génétique m'a été une idée prometteuse, sa mise en œuvre complète n'a malheureusement pas été réalisée dans le cadre de ce projet, du à un manque de temps pour sa mise en œuvre.

Suite à cette constatation, j'ai opté pour une approche simple, en combinant deux heuristiques dans l'objectif de créer une IA efficace, avec l'heuristique suivante.

$$\text{heuristic} = \text{score} - \text{badHole} - \text{goodHole} \quad (1)$$

Dans cette équation :

`score(board)` représente le score actuel du joueur.

`badHole(board)` évalue la présence de trous "mauvais" dans le plateau de jeu.

`goodHole(board)` évalue la présence de trous "bon" dans le plateau de jeu.

4.4 Optimisation de la profondeur de recherche

Face à la complexité de trouver une heuristique optimale pour le jeu Awélé, j'ai choisi de me concentrer sur l'optimisation de la profondeur de recherche de l'algorithme de recherche en profondeur utilisé par mon bot. Cette approche visait à compenser le manque de précision des heuristiques en explorant plus en profondeur les arbres de jeu.

Pour ce faire, j'ai recueilli des données sur le nombre de nœuds explorés et les performances de l'élagage alpha-bêta en fonction de la profondeur de recherche. Ces données ont été collectées à partir de centaines de parties simulées, ce qui m'a permis d'analyser les tendances et de déterminer la profondeur optimale pour mon algorithme.

Nombre de Noeud/Elagage par nombre de graines

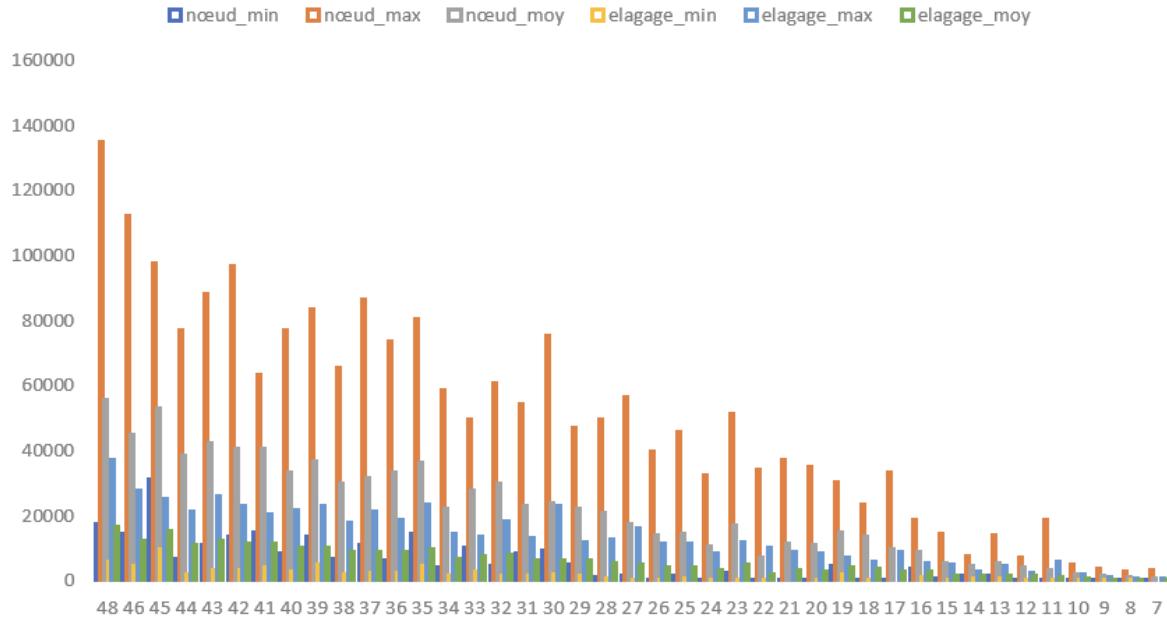


FIGURE 2 – Graphique des analyse des données

Au fur et à mesure de cette analyse, j'ai constaté que le nombre de noeud et d'élagage rester environ proportionnelle au nombre de graine sur le terrain. Ce qui signifie que moins on a de graine sur le terrains, moins on parcours de noeud pour une même profondeur. Donc un solution envisageable pour évité de perdre l'opportunité d'explorer plus de noeud, est d'adapter la profondeur au nombre de graine sur le terrains, grâce à cette formule pour la profondeur :

$$\text{depth} = \text{DEPTH} + \frac{\text{MAX_SEED}}{\max(\text{MIN_SEED}, \text{board.getNbSeeds}()))}$$

Dans cette formule :

- DEPTH représente la profondeur de recherche initiale.
- MAX_SEED et MIN_SEED sont des constantes définissant respectivement le nombre maximal et minimal de graines sur le plateau.
- board.getNbSeeds() donne le nombre actuel de graines sur le plateau.
- La division ajuste la profondeur en fonction du nombre de graines présentes sur le plateau. Plus il y a de graines, moins la profondeur de recherche sera augmentée.

Cette formule permet d'ajuster dynamiquement la profondeur de recherche en fonction de la situation actuelle du plateau, en augmentant la profondeur lorsque le nombre de graines est faible pour explorer plus en profondeur les noeuds de jeu et en réduisant la profondeur lorsque le nombre de graines est élevé pour économiser des ressources.

4.4.1 Triage des nœuds pour l'algorithme Alpha-Bêta

Arrivé à ce point là, il me restait plus qu'une idée pour optimiser la vitesse de recherche de mon algorithme Alpha-Bêta, introduit une technique de tri des nœuds. Cette méthode consiste à évaluer les nœuds enfants générés par l'algorithme en fonction de leur valeur heuristique, puis à les trier dans un ordre spécifique avant de les évaluer avec l'algorithme Alpha-Bêta.

Le tri des nœuds permet d'explorer en priorité les nœuds les plus prometteurs, c'est-à-dire ceux qui sont susceptibles de conduire à une meilleure valeur pour le joueur en cours, tout en minimisant le nombre de nœuds à évaluer. Bien que ce tri entraîne une petite perte d'efficacité sur les solution exploré, cette perte est largement compensée par la capacité à parcourir plus en profondeur l'arbre de jeu.

Cependant, la mise en œuvre du tri des nœuds a nécessité une refonte complète de mon code, car il a fallu adapter l'algorithme de génération des nœuds enfants pour qu'il génère des nœuds triés selon la heuristique choisie.

En fin de compte, cette optimisation a permis d'augmenter de manière significative la profondeur avec une profondeur de base de 10 soit 10 coup en avance et par conséquent la force de l'IA donnant naissance à **TYSONBOT**.

5 Résultats

Les résultats de la compétition sont présentés dans le tableau ci-dessous :

Rangs	Nom du bot	Score	Détails
1.	TYSONBOT	14.7	Auteur(s) : Alexandre Untereiner Durée d'une prise de décision : 00 :00 :00.069
2.	OmegaHeuristique	10.8	Auteur(s) : Alexandre Untereiner Durée d'une prise de décision : 00 :00 :00.058
3.	ALLHeuristique	10.2	Auteur(s) : Alexandre Untereiner Durée d'une prise de décision : 00 :00 :00.060
4.	alpha2	6.3	Auteur(s) : Alexandre Untereiner Durée d'une prise de décision : 00 :00 :00.022
5.	MinMax	3.0	Auteur(s) : Alexandre Blansché Durée d'une prise de décision : 00 :00 :00.049
6.	Random	0.0	Auteur(s) : Alexandre Blansché Durée d'une prise de décision : 00 :00 :00.000

TABLE 1 – Classement des bots selon leur performance

L'analyse des résultats de la compétition permet de tirer plusieurs conclusions intéressantes :

1. **TYSONBOT** a obtenu le meilleur score avec 14.7, ce qui indique une performance significativement supérieure aux autres bots. Cela peut s'expliquer par une stratégie efficace et une prise de décision bien optimisée, comme en témoigne sa durée de prise de décision relativement courte (0.069 secondes).
2. **OmegaHeuristique** et **ALLHeuristique** se classent respectivement deuxième et troisième avec des scores de 10.8 et 10.2. Bien qu'ils aient obtenu des scores plus bas que **TYSONBOT**, leur performance reste notable, surtout compte tenu de leur durée de prise de décision relativement courte (0.058 et 0.060 secondes respectivement).
3. **alpha2** se classe quatrième avec un score de 6.3. Bien que son score soit inférieur à celui des trois premiers, il reste dans le haut du classement. Sa durée de prise de décision (0.022 secondes) est également compétitive, ce qui indique une efficacité dans ses calculs.
4. **MinMax** arrive en cinquième position avec un score de 3.0. Bien que son score soit plus bas que les autres bots, il reste significatif.
5. **Random** se classe dernier avec un score de 0.0. due à une stratégie aléatoire.

En résumé, **TYSONBOT** a clairement dominé la compétition avec le meilleur score et une durée de prise de décision compétitive. Les autres bots ont également obtenu des performances respectables, chacun ayant ses propres forces et faiblesses en termes de stratégie et de temps de calcul.

6 Améliorations Futures de l'Algorithmme

Malgré les progrès réalisés dans l'optimisation de **TYSONBOT**, l'algorithme possède encore de nombreuses points à améliorer, plusieurs pistes d'amélioration restent à explorer pour augmenter davantage sa performance et son efficacité. Voici quelques améliorations futures possibles :

1. **Optimisation de l'algorithme de recherche :** Continuer à explorer des moyens d'optimiser l'algorithme de recherche en profondeur, en cherchant des techniques plus avancées d'élagage ou d'optimisation pour réduire davantage le nombre de noeuds explorés tout en préservant la qualité de la recherche.
2. **Intégration d'une approche par apprentissage automatique :** Malgré la décision de ne pas utiliser de réseaux de neurones dans ce projet en raison des contraintes de temps et de complexité, explorer cette voie à l'avenir pourrait être prometteur. L'intégration d'un système d'apprentissage automatique pourrait permettre à l'IA de s'améliorer progressivement au fil du temps en apprenant des stratégies et des schémas de jeu à partir de données d'entraînement.
3. **Exploration d'algorithmes avancés :** Investiguer d'autres algorithmes d'intelligence artificielle, tels que les méthodes de Monte-Carlo Tree Search (MCTS), qui peuvent être particulièrement efficaces dans les jeux de stratégie. Ces approches pourraient offrir de nouvelles perspectives pour améliorer les performances de l'IA.
4. **Développement d'heuristiques plus sophistiquées :** Continuer à rechercher et à développer des heuristiques plus avancées et sophistiquées pour évaluer les positions de jeu, en tenant compte de facteurs tels que la structure du plateau, les séquences de coups et les schémas de jeu spécifiques.

En explorant ces pistes d'amélioration, je suis convaincu que mon **TYSONBOT** pourrait devenir encore plus fort et mettra tout le monde au tapis !

7 Conclusion

Dans le cadre de ce projet d'intelligence artificielle pour le jeu Awélé, j'ai exploré diverses approches et techniques pour développer un bot capable de jouer à un niveau compétitif. J'ai commencé par étudier les règles du jeu et les stratégies traditionnelles utilisées par les joueurs, ce qui m'a permis de mieux comprendre les défis posés par ce jeu de stratégie. Bien que j'ai dû écarté énormément de solution potentiel comme les réseau de neurone ,l'apprentissage etc. par manque de temps suite à la non contribution de mon ancien binôme au projet, d'où mon choix de faire le projet seul.

J'ai donc concentré en particulier sur un seul algorithme d'intelligence artificielle, l'algorithme MinMax avec l'optimisation Alpha-Bêta, que j'ai choisi d'implémenter pour mon **TYSONBOT**. J'ai également exploré l'utilisation d'heuristiques pour évaluer les positions de jeu et guider la recherche.

Malgré mes efforts pour trouver une heuristique performante, je n'ai pas réussi à déterminer une stratégie clairement supérieure aux autres. Cependant, j'ai pu développer un algorithme de recherche en profondeur optimisé, qui ajuste dynamiquement sa profondeur en fonction de la situation du jeu, ce qui a permis d'explorer plus en profondeur l'arbre de jeu.

Enfin, j'ai introduit une technique de tri des noeuds pour optimiser la vitesse de recherche de l'algorithme Alpha-Bêta, ce qui a contribué à améliorer la force de jeu de mon **TYSONBOT**.

En conclusion, bien que je n'aie pas pu trouver une solution parfaite pour résoudre le jeu Awélé, ce projet m'a permis d'explorer différentes techniques d'intelligence artificielle et d'acquérir une expérience précieuse dans le domaine. Malgré les difficultés rencontrées, ce projet m'a profondément inspiré, me faisant réaliser qu'il existe une multitude de possibilités et de combinaisons différentes qui n'attendent qu'à être explorées. Et j'aurais aimé explorer chaque facette de cet horizon avec enthousiasme.

Références

- [1] Molinier, M. (2020). Implémentation d'une IA pour l'Awélé - Partie 1 [Medium article]. Disponible à l'adresse : <https://medium.com/@mol02office/implementation-dune-ia-pour-l-awal-partie-1-b298328d5e14>
- [2] Bagarre, T. (2020-2021). Teluob_Bagarre_-2020-2021.pdf. [Document PDF]

- [3] Inconnu. (2019-2020). Jai Awale de Travers - 2019-2020.pdf. [Document PDF]
- [4] Carré Tropical. (s.d.). Awalé : Le jeu de stratégie africain. Disponible à l'adresse : <https://carre-tropical.fr/products/awale-le-jeu-de-strategie-africain>
- [5] Myriad. (s.d.). Stratégie du jeu Awalé. Disponible à l'adresse : <https://www.myriad-online.com/resources/docs/awale/francais/strategy.htm>
- [6] Africouleur. (2016). AWÉLÉ : Règle du Jeu [Document PDF]. Disponible à l'adresse : <https://www.africouleur.com/wp-content/uploads/2016/10/AW%C3%89L%C3%89-R%C3%A8gle-du-Jeu.pdf>