



INTELLIGENCE ARTIFICIELLE

PROJET – AWÉLÉ
RAPPORT

OWARENESS

Étudiants :

BOURAOUI Amir
SONG Lewei

Professeur :

BLANSCHÉ Alexandre

Février 2025

Table des matière

S

INTRODUCTION.....	1
RECHERCHES.....	1
LE PROCESSUS DE RÉALISATION.....	2
1.1 MINMAX ET ALPHA-BÊTA ÉLAGAGE.....	2
1.2 NEGAMAX ET ALPHA-BÊTA ÉLAGAGE.....	3
1.2.1 Heuristique de Tri des Coups.....	4
1.2.2 Implémentation de l'arbre récursif.....	4
1.3 ÉVALUATION ET DÉCISION.....	4
1.3.1 Évaluation statique du plateau.....	5
1.3.2 Recherche dynamique et adaptation de la profondeur.....	5
1.4 APPRENTISSAGE.....	6
1.4.1 Les données existantes et lire les mémoires.....	6
1.4.2 La génération des données existantes.....	6
1.4.3 Intégration de l'apprentissage historique.....	8
1.4.4 Mise en place de l'apprentissage historique.....	8
1.4.5 Génétique.....	8
OWARENESS.....	9
1.5 FONCTIONNALITÉS CLÉS DE NOTRE IA.....	9
CONCLUSION.....	10
BIBLIOGRAPHIE.....	11

Introduction

Ce cours nous demande de créer une intelligence artificielle (IA) capable de jouer à l'Awélé en utilisant l'algorithme MinMax. L'objectif est de remporter le plus de victoires possibles lors des affrontements contre les IA créées par d'autres participants.

Dans le jeu d'Awélé, deux joueurs s'affrontent en tentant de récolter le plus grand nombre de graines en respectant les règles du jeu. Le joueur qui en collecte le plus remporte la partie. Il est important de noter qu'en 2002, il a été démontré que le jeu d'Awélé comporte **889 063 398 406** situations possibles. (Romein & Henri, 2003) Si toutes ces configurations étaient stockées en mémoire pour jouer, chaque partie aboutirait systématiquement à une égalité. Nous devons donc, dans le cadre des contraintes imposées, concevoir une IA aussi performante que possible en respectant ces règles.

Ainsi, nous avons cherché à développer une IA performante et adaptative en explorant différentes approches d'optimisation et d'apprentissage. Ce rapport présente notre méthodologie et les résultats obtenus.

Recherches

Avant de concevoir notre propre intelligence artificielle (IA), nous avons d'abord étudié les IA existantes afin de comprendre leurs approches et leurs méthodologies. En nous basant sur le rapport de Mathis Saillot, ainsi que sur les autres articles, nous avons découvert plusieurs méthodes permettant d'implémenter une IA pour l'Awélé.

Parmi ces approches, nous avons identifié différentes stratégies :

- **L'algorithme MinMax** : une méthode de base utilisée pour évaluer les coups en anticipant les réponses de l'adversaire.
- **L'algorithme NegaMax** : une variante du MinMax, simplifiant le calcul des évaluations tout en conservant la même efficacité.
- **L'optimisation de la structure de l'algorithme** : en augmentant la profondeur d'itération dans l'arbre de décision, nous pouvons améliorer la capacité d'anticipation de l'IA.
- **L'utilisation d'algorithmes génétiques** : ajuster dynamiquement certains coefficients pour influencer la prise de décision de l'IA en fonction des résultats de parties précédentes.

- **L'ajout de coefficients pondérés pour guider les choix de l'IA** : en attribuant des valeurs spécifiques à certaines configurations du plateau, l'IA peut apprendre à privilégier des stratégies gagnantes.
- **L'intégration d'un réseau de neurones** : une approche plus avancée qui permettrait à l'IA d'apprendre de manière plus flexible en analysant un grand volume de données.

Après cette analyse, nous avons choisi d'implémenter progressivement ces approches, en commençant par MinMax, avant d'explorer des optimisations plus avancées.

Le processus de réalisation

Après avoir examiné ces différentes possibilités, nous avons décidé de commencer par l'implémentation de l'algorithme MinMax, puis d'améliorer progressivement notre IA en appliquant des optimisations et des stratégies avancées.

1.1 MinMax et Alpha-Bêta Élagage

L'algorithme MinMax est une méthode de prise de décision utilisée dans les jeux au tour par tour. Son principe fondamental consiste à maximiser le score du joueur actif tout en minimisant celui de l'adversaire, en explorant récursivement un arbre de jeu pour identifier la meilleure stratégie. L'IA anticipe plusieurs tours à l'avance et suppose que l'adversaire joue toujours de manière optimale, ce qui lui permet de déduire le meilleur coup à jouer. Étant donné que l'espace de recherche peut être très vaste, l'algorithme est souvent optimisé à l'aide de l'élagage Alpha-Bêta, ce qui réduit le nombre de calculs nécessaires et améliore son efficacité.

En nous basant sur notre première compréhension du MinMax grâce au jeu du morpion ainsi que sur l'IA MinMax déjà présente dans le demo, nous avons créé notre première version en suivant presque exactement l'approche du bot existant. L'algorithme est le suivant :

MinMaxNode(Plateau, Profondeur, α , β)

- Si le plateau est un état terminal ou si la profondeur atteint la limite maximale, alors retourner l'évaluation du plateau.
- Si c'est un nœud Max (tour du joueur qui maximise) :
 1. Initialiser valeur = $-\infty$.
 2. Pour chaque coup possible dans le plateau :
 - Si le coup est jouable :
 1. Copier l'état du plateau et appliquer le coup.
 2. Calculer score = MinMaxNode(Nouveau Plateau, Profondeur + 1, α , β).
 3. Si score > valeur, alors mettre à jour valeur = score.
 4. Si valeur $\geq \beta$, alors retourner valeur (coupure Alpha-Bêta).
 5. Si valeur > α , alors mettre à jour α = valeur.
- Sinon (c'est un nœud Min, tour du joueur qui minimise) :

1. Initialiser valeur = $+\infty$.
2. Pour chaque coup possible dans le plateau :
 - Si le coup est jouable :
 1. Copier l'état du plateau et appliquer le coup.
 2. Calculer score = MinMaxNode(Nouveau Plateau, Profondeur + 1, α , β).
 3. Si score < valeur, alors mettre à jour valeur = score.
 4. Si valeur $\leq \alpha$, alors retourner valeur (coupure Alpha-Bêta).
 5. Si valeur < β , alors mettre à jour β = valeur.
- Retourner valeur (meilleur score trouvé).

1.2 NegaMax et Alpha-Bêta Élagage

Après avoir consulté les documents pertinents (ÉPREUVE COMMUNE DE TIPE 2012), (Sala, s.d.), nous avons décidé de mettre à jour notre algorithme en implémentant un algorithme NegaMax.

Comparé à MinMax, NegaMax applique une logique de maximisation à tous les nœuds, ce qui permet d'unifier les étapes Max et Min et de simplifier le code.

Dans un jeu à somme nulle :

Si le score du joueur actuel est +S, alors le score de l'adversaire est -S.

Autrement dit, maximiser son propre score revient à minimiser celui de l'adversaire avec un signe opposé.

La formule de NegaMax est la suivante :

$$v = \max(-\text{NegaMax}(\text{nœud fils}))$$

Nous avons mis à jour notre algorithme et l'avons testé contre le MinMax Bot présent dans la démo. Nous avons constaté qu'à profondeur égale, l'algorithme NegaMax est légèrement supérieur à MinMax. Nous supposons que cela est dû au fait que NegaMax simplifie le code, ce qui lui confère un avantage particulièrement notable lorsqu'il est combiné avec l'élagage Alpha-Bêta.

L'algorithme mis à jour de NegaMax est le suivant :

NegaMaxNode(Plateau, Profondeur, α , β)

- Si le plateau est un état terminal ou si la profondeur atteint la limite maximale, alors retourner l'évaluation du plateau.
- Initialiser valeur = $-\infty$.
- Pour chaque coup possible dans le plateau :
 - Si le coup est jouable :
 1. Copier l'état du plateau et appliquer le coup.
 2. Si le nouveau plateau a ≤ 6 graines ou si Profondeur \geq la profondeur maximale :
 - Calculer score = l'évaluation du plateau.

3. Sinon :
 - Calculer score = -NegaMaxNode(Nouveau Plateau, Profondeur + 1, $-\beta$, $-\alpha$).
 - 4. Si score > valeur, alors mettre à jour valeur = score.
 - 5. Si Profondeur > 0 :
 - Mettre à jour α = max(α , valeur).
 - Si $\alpha \geq \beta$, alors arrêter la boucle (coupure Alpha-Bêta).
- Retourner valeur (meilleur score trouvé).

1.2.1 Heuristique de Tri des Coups

Afin d'optimiser l'élagage Alpha-Bêta et d'accélérer le temps de calcul par coup, nous avons intégré une heuristique de tri des coups (getMoveOrder()). Cette méthode permet d'analyser en priorité les coups les plus prometteurs, améliorant ainsi l'efficacité de l'élagage et réduisant le nombre de nœuds évalués.

L'algorithme classe les coups selon plusieurs critères : capturer des graines adverses, contrôle du centre, mobilité et heuristique historique.

Cependant, après plusieurs tests, nous avons constaté que ce tri ralentissait paradoxalement l'algorithme. Nous avons donc décidé de ne pas l'utiliser.

1.2.2 Implémentation de l'arbre récursif

Après avoir implémenté NegaMax avec l'élagage Alpha-Bêta, nous avons voulu optimiser davantage la structure de recherche afin de rendre le code plus clair et le processus de calcul plus intuitif. Dans la version traditionnelle de NegaMax, l'exploration de l'arbre repose entièrement sur des appels récursifs de fonctions, et chaque calcul est transmis par des valeurs de retour. Bien que cette méthode permette de parcourir efficacement l'arbre du jeu, elle devient difficile à gérer lorsque la profondeur de recherche augmente. C'est pourquoi nous avons choisi d'adopter une structure d'arbre récursif sous forme d'objets, où chaque nœud est représenté par un objet NegaMaxNode, qui effectue son calcul dès sa création.

L'objectif principal de cette amélioration n'est pas d'accélérer les calculs, mais plutôt de rendre la récursion plus compréhensible et d'optimiser le stockage et la gestion des informations de recherche. Comparée à l'approche récursive classique, cette structure offre plusieurs avantages :

- Une meilleure organisation du code : Chaque nœud de recherche est un objet, ce qui rend la structure de l'arbre plus claire et évite de dépendre de la pile d'appels pour stocker des variables temporaires.
- Une gestion plus flexible des états : Les objets peuvent conserver les informations de l'état actuel du jeu, ce qui permet d'y accéder plus facilement, sans risque de perte après le retour de la récursion.
- Une exécution optimisée de l'élagage Alpha-Bêta : Chaque nœud stocke directement les décisions de recherche, ce qui permet d'appliquer l'élagage de manière plus intuitive et d'éviter les recalculs inutiles.

1.3 Évaluation et décision

Après avoir mis en place l'algorithme NegaMax et amélioré l'élagage Alpha-Bêta, nous avons ajouté une fonction d'évaluation pour aider l'IA à mieux choisir ses coups. (Ayenon, 2023) Cette fonction donne un score à chaque position sur le plateau, ce qui permet à l'IA de comparer les différentes options et de choisir le meilleur coup possible.

Notre méthode repose sur trois idées principales : une évaluation statique de l'état du jeu, une adaptation basée sur l'apprentissage historique, et une modulation dynamique de la profondeur de recherche.

1.3.1 Évaluation statique du plateau

L'évaluation du plateau se base sur plusieurs critères qui aident l'IA à prendre ses décisions :

- Le contrôle du plateau : Plus un trou contient de graines, plus il est utile, car il permet de semer plus facilement. À l'inverse, un trou vide est un problème, car il limite les options du joueur.
- Les opportunités de capture : Si un coup permet de prendre des graines à l'adversaire, il est intéressant. L'IA va donc privilégier ces coups pour attaquer.
- Les risques et la vulnérabilité : Si un trou contient peu de graines, il risque d'être capturé facilement par l'adversaire. L'IA doit donc éviter de se retrouver avec trop de trous vulnérables.
- L'état du jeu de l'adversaire : Si l'adversaire a beaucoup de trous vides, c'est une bonne nouvelle, car il a moins d'options. En revanche, s'il a plusieurs trous bien remplis, il peut préparer des coups dangereux.
- Les trous "Krou" (≥ 12 graines) : Owareness donne plus d'importance aux trous contenant 12 graines en début de partie. Leur impact diminue progressivement pour éviter que l'IA ne se repose trop sur eux.

L'influence des krous est calculée avec la formule suivante :

$$krouEvaluation = (playerKrous \times 3.5) - (opponentKrous \times 3.5) \times \left(1.0 - \frac{board.getNbSeeds()}{48.0}\right)$$

L'IA adapte aussi son évaluation en fonction du moment du jeu :

- Au début, l'important est de contrôler le plateau, c'est-à-dire garder un maximum d'options pour jouer et empêcher l'adversaire d'en avoir trop.
- À la fin, ce qui compte surtout, c'est de marquer des points. L'IA va donc privilégier les coups qui lui permettent de creuser l'écart au score et de s'assurer la victoire..

1.3.2 Recherche dynamique et adaptation de la profondeur

L'IA ajuste dynamiquement sa profondeur de recherche pour améliorer son efficacité et optimiser ses décisions. Contrairement à une profondeur fixe qui peut être inefficace (trop faible, elle limite la vision ; trop grande, elle ralentit la prise de décision), nous avons introduit une adaptation dynamique basée sur la progression du jeu.

Ajustement en fonction du jeu

- Début de partie (> 40 graines restantes) → Recherche plus profonde pour anticiper les stratégies à long terme.
- Milieu de partie → Ajustement progressif de la profondeur grâce à une fonction de décroissance exponentielle, assurant une transition fluide entre les phases.
- Fin de partie (< 12 graines restantes) → Réduction de la profondeur pour accélérer la prise de décision, en priorisant l'évaluation immédiate du plateau.

Optimisation par l'apprentissage historique L'IA ajuste également sa profondeur selon l'historique des parties :

- Si un coup a un taux de victoire > 60%, l'IA réduit sa recherche et privilégie une exécution rapide.
- Si un coup a un taux de victoire < 40%, elle augmente la profondeur de recherche pour identifier une alternative plus optimale.
- Le facteur de confiance (playCountWeight) évite que l'IA ne se base trop tôt sur des données insuffisantes.
- L'impact des données historiques augmente au fil du jeu : en début de partie, l'IA se repose davantage sur l'exploration, tandis qu'en fin de partie, elle intègre davantage l'apprentissage.

Ce système hybride permet à l'IA de trouver un équilibre entre performance et vitesse, en adaptant intelligemment son niveau d'analyse aux besoins réels de la partie.

1.4 Apprentissage

Dans une partie d'Awélé, se baser uniquement sur une fonction d'évaluation statique ne suffit pas toujours pour prendre les meilleures décisions, surtout dans des situations complexes. C'est pourquoi nous avons ajouté un mécanisme d'apprentissage (Apprentissage) à notre IA, lui permettant d'améliorer sa stratégie en accumulant des données issues des parties précédentes.

Le principe clé de cet apprentissage est d'enregistrer les configurations du plateau, les coups joués et les résultats des parties, puis d'utiliser ces informations lors des prochaines décisions. Ainsi, l'IA peut évoluer progressivement au fil des parties et devenir plus performante.

1.4.1 Les données existantes et lire les mémoires

L'apprentissage doit commencer par les bases les plus simples. Les données fournies sont décrites ainsi :

"On dispose d'un ensemble de données correspondant à 303 situations de jeu observées au cours de plusieurs parties entre un joueur expérimenté et un joueur novice. Le joueur expérimenté a remporté toutes les parties."

Ainsi, nous avons d'abord utilisé ces données comme mémoire initiale. Nous avons stocké ces informations dans une table de hachage, ce qui nous permet de retrouver rapidement les situations déjà rencontrées. Lors d'une vraie partie, si l'IA rencontre un plateau identique à l'un de ceux enregistrés, elle peut directement jouer le coup mémorisé, car il a déjà été prouvé comme étant optimal.

Après avoir intégré cette mémoire, l'IA avec mémoire peut facilement battre une IA sans mémoire.

1.4.2 La génération des données existantes

Dans la section précédente, nous avons commencé par utiliser les données des parties existantes. Lors d'un match, si l'IA rencontrait une position identique, elle appliquait directement l'action enregistrée dans la base de données. Cependant, nous avons rapidement constaté que cette approche posait deux problèmes majeurs :

- Correspondance imparfaite des positions : Même si une position semble très similaire, une légère différence dans la répartition des graines (ou un coup différent de l'adversaire) peut rendre l'application directe des données inefficace, voire trompeuse pour la prise de décision.
- Adaptation aux nouvelles positions : Le jeu d'Awélé comporte un nombre immense de configurations possibles. Il est donc impossible d'enregistrer toutes les situations. Lorsqu'un état inconnu apparaît, l'IA doit toujours être capable de rechercher une solution et, idéalement, s'appuyer sur l'expérience accumulée pour améliorer ses décisions. Ainsi, nous avons trouvé un compromis entre l'utilisation pure de la mémoire et la recherche dynamique : nous avons transformé les données historiques en un système de pondération dynamique.

L'IA analyse chaque observation enregistrée (position du plateau, coup joué et résultat de la partie) pour calculer la fréquence de victoire associée à chaque trou. Si un trou a souvent conduit à une victoire, il se voit attribuer un coefficient de pondération plus élevé.

Par exemple, si le trou 2 a un taux de victoire élevé, nous augmentons légèrement son score dans l'évaluation interne, ce qui incite l'algorithme de recherche à considérer davantage cette option lors de la prise de décision.

Lorsqu'on applique l'algorithme NegaMax / Alpha-Bêta, l'IA ne suit pas aveuglément les décisions de la mémoire. Au lieu de cela, elle intègre une pondération historique dans l'évaluation des coups.

Si une action a un taux de victoire élevé dans les données passées, elle reçoit un bonus de score. À l'inverse, si elle a souvent conduit à une défaite, elle est légèrement pénalisée. La décision finale est prise en combinant l'évaluation statique, la recherche en profondeur, et l'influence des données historiques.

Même si certains trous sont statistiquement plus favorables, l'IA conserve une certaine profondeur de recherche et une part de variabilité. Cela permet d'éviter le sur-apprentissage des anciennes données, qui pourrait rigidifier la stratégie, et de maintenir la flexibilité face aux nouvelles stratégies adverses.

Grâce à cette approche hybride, nous exploitons les connaissances issues des parties précédentes, tout en permettant à l'IA de continuer à explorer et s'adapter.

L'apprentissage historique ne sert plus de modèle rigide où chaque position a une réponse fixe, mais devient une aide dynamique à la décision. Cette méthode a prouvé son efficacité, améliorant significativement les performances de l'IA tout en évitant une dépendance excessive aux anciennes données.

1.4.3 Intégration de l'apprentissage historique

L'évaluation statique du jeu ne suffit pas toujours pour prendre la meilleure décision. C'est pour cela que nous avons ajouté un système d'apprentissage qui permet à l'IA d'adapter sa stratégie en fonction des parties précédentes.

1.4.4 Mise en place de l'apprentissage historique

L'IA mémorise les coups joués et leur résultat (victoire ou défaite). Plus un coup a été efficace dans le passé, plus il sera favorisé. Cependant, Owareness ajuste cette influence selon le moment de la partie pour éviter un apprentissage trop rigide.

$$\text{gamePhaseMultiplier} = \frac{48 - \text{board.getNbSeeds}()}{48.0}$$
$$\text{historicalInfluence} = 5.0 \times \text{gamePhaseMultiplier}$$

Cela signifie que :

- En début de partie, l'IA explore plus de possibilités et se base principalement sur l'évaluation statique.
- En fin de partie, elle se fie davantage aux décisions ayant déjà prouvé leur efficacité.

Ainsi, l'IA trouve un équilibre entre tester de nouvelles stratégies et utiliser ce qu'elle a appris.

1.4.5 Génétique

Lors de la conception de l'IA pour ce jeu, nous avons envisagé d'utiliser un algorithme génétique. Ce type d'algorithme est souvent employé pour résoudre des problèmes complexes d'optimisation et de recherche. Il s'inspire du processus d'évolution biologique et fonctionne en sélectionnant, combinant et modifiant des solutions pour trouver la meilleure. (Koza, 1992)

Dans notre IA, nous avons tenté d'ajuster certaines décisions en utilisant des poids pour influencer chaque coup. Par exemple, nous avons cherché à maintenir un certain nombre de graines dans nos trous, à vider autant que possible les trous de l'adversaire, et à éviter que l'adversaire puisse capturer un grand nombre de graines au tour suivant. Nous avons voulu entraîner ces paramètres à l'aide d'un algorithme génétique afin d'optimiser les choix de l'IA.

Nous avons défini six coefficients représentant des éléments stratégiques :

1. Le poids du score du joueur actuel.

2. Le poids du score de l'adversaire.
3. Un bonus si un trou du joueur contient plus de 12 graines.
4. Une pénalité si un trou de l'adversaire contient plus de 12 graines.
5. Une pénalité si un trou du joueur contient moins de 3 graines.
6. Un bonus si un trou de l'adversaire contient moins de 3 graines.

Pendant l'apprentissage (learn), nous avons intégré une mutation simple, en modifiant chaque coefficient de manière aléatoire de ± 0.05 . Chaque exécution ne comportait qu'une seule génération d'entraînement, sans évaluation de fitness ni conservation du meilleur individu. En conséquence, l'IA pouvait être légèrement plus forte ou plus faible après chaque session d'apprentissage.

Finalement, nous avons constaté que l'entraînement de ces paramètres nécessiterait bien plus d'une heure, ce qui le rendait inefficace en termes de temps et de ressources. C'est pourquoi nous avons décidé d'abandonner l'approche par algorithme génétique.

OWARENESS

OWARENESS est notre joueur artificiel conçue dans le but d'être imbattable au jeu de l'Awélé, connu en anglais sous le nom "Oware". Son nom, Owareness, est fusion astucieuse entre "Oware", qui désigne le jeu et "ness". Ainsi "Owareness" fait un clin d'œil au mot anglais "awareness" qui signifie conscience. Ce nom reflète parfaitement l'essence de notre bot qui est une IA dotée d'une perception fine de son environnement et capable de s'adapter avec intelligence à chaque situation de jeu. Grâce à des algorithmes avancés, une analyse stratégique et un apprentissage basé sur l'expérience.

1.5 Fonctionnalités Clés de Notre IA

1. Recherche Optimisée avec NegaMax et Alpha-Bêta
 - Notre IA repose sur l'algorithme NegaMax, une version améliorée de Minimax, couplée à l'élagage Alpha-Bêta.
 - Elle explore les coups possibles jusqu'à une profondeur maximale par défaut de 11, tout en réduisant le temps de calcul en éliminant les options inutiles grâce à l'élagage.
2. Adaptation Dynamique de la Profondeur
 - OWARENESS ajuste sa profondeur de recherche en fonction du nombre de graines sur le plateau :

- o En début de partie (40 graines ou plus), elle privilégie une analyse approfondie et valorise les trous centraux (indices 2 et 3) pour dominer le jeu.
 - o En fin de partie (12 graines ou moins), elle réduit la profondeur pour des décisions rapides et précises.
3. Apprentissage Intelligent via l'Histoire
- Notre bot exploite un fichier de données (owareness.data) pour analyser les parties passées, associant chaque position et coup à un taux de victoire.
 - Ces données influencent ses choix, avec une pondération plus forte en début de partie et une confiance ajustée selon la fréquence des situations rencontrées.
4. Gestion Fiable des Données
- OWARENESS initialise ses données à partir de awele.data, crée un fichier si nécessaire et gère les erreurs sans s'arrêter.
 - Une classe dédiée, OwarenessTrainer, simule 55 minutes de parties entre deux bots (profondeurs 8 et 7) pour enrichir sa base de connaissances.
5. Évaluation Stratégique des Positions
- L'IA évalue les positions en tenant compte :
 - o Du nombre de graines par trou, avec des bonus pour les captures et des malus pour les faiblesses.
 - o De la différence de score, amplifiée en fin de partie (x3 si 12 graines ou moins).
 - o De poids ajustés par trou, basés sur les succès historiques de +10 % si taux de victoire > 60 %
6. Entraînement Autonome
- Des simulations opposent deux versions de notre bot, enregistrant chaque état de jeu et marquant les coups gagnants ou perdants.
 - Les parties s'arrêtent à 25 points, 6 graines ou après 1000 coups sans capture pour éviter les boucles.

Conclusion

Ce projet a constitué une aventure passionnante et formatrice dans notre première expérience de conception d'une IA pour l'Awélé. Dès le départ, nous avons rapidement constaté l'immensité de l'espace de recherche du jeu, ce qui nous a poussés à adopter une approche itérative. Nous sommes partis d'un algorithme MinMax classique, pour évoluer vers une implémentation en

NegaMax optimisée avec l'élagage alpha-beta. Cette démarche nous a permis de clarifier le code et de poser des bases solides pour les améliorations ultérieures.

Un aspect central de notre travail a été l'affinement constant des points de récompense et l'ajustement des paramètres, notamment en intégrant et en exploitant les données d'entraînement issues des matchs entre deux versions de notre bot. Nous avons consacré une grande partie de nos efforts à ajuster ces coefficients, car ils conditionnent la pondération des coups et influencent directement la prise de décision de l'IA. L'utilisation judicieuse de l'historique des parties a permis à notre bot de s'adapter et d'apprendre, transformant chaque partie en une opportunité d'optimisation.

Par ailleurs, une préoccupation majeure a été l'optimisation du code pour garantir un temps de décision inférieur à 100 ms. Pour ce faire, nous avons retravaillé la structure récursive en adoptant une approche orientée objet, optimisé l'élagage alpha-beta et mis en place une gestion dynamique de la profondeur de recherche en fonction du stade de la partie. Ces efforts ont permis de concilier rapidité d'exécution et de prise de décision.

En somme, ce projet nous a démontré l'importance d'un équilibre constant entre théorie et pratique, ainsi que la nécessité d'une attention minutieuse aux détails pour développer une IA performante.

Bibliographie

Aualé: Unleash your inner Oware Master

<https://www.oware.co.uk/post/ai-engine-aalina-on-its-way-to-playing-perfectly-against-the-48stones-database>

Design of Artificial Intelligence for Mancala Games

<https://www.politesi.polimi.it/retrieve/a81cb05c-47b8-616b-e053-1605fe0a889a/Thesis.pdf>

Building the Mancala Game Application

<https://www.linkedin.com/pulse/building-mancala-game-application-deep-dive-riaz-farhanian-uh3e/>

The Ancient Game and the AI

<https://medium.com/towards-data-science/the-ancient-game-and-the-ai-d7704bea280d>

An Artificial Intelligence Approach to Mancala

https://fiasco.ittc.ku.edu/publications/documents/Gifford_ITTC-FY2009-TR-03050-03.pdf

Mancala games - Topics in Mathematics and Artificial Intelligence

https://www.researchgate.net/publication/239523032_Mancala_games_Topics_in_Mathematics_and_Artificial_Intelligence

Joys of minimax and negamax

<https://medium.com/@indykidd/joys-of-minimax-and-negamax-ee5e456977e2>

Minimax Algorithm

<https://rediet-abere.medium.com/minimax-algorithm-aa1457521fc7>

The Exploration and Analysis of Mancala

<https://digitalcommons.andrews.edu/cgi/viewcontent.cgi?article=1259&context=honors>

A MiniMax Agent variant of Mancala

<https://ieeexplore.ieee.org/document/9183848>

Mancala game using Greedy, Minimax

<https://www.ankitcodinghub.com/product/solvedmancala-game-using-greedy-minimax-and-alpha-beta-pruning-algorithm-solution/>