AI Game Programming : Awalé, où comment jouer automatiquement

Arno Lesage (no. 22202985)[a,b] **and Jean-Jacques Viale (no. 22202859)**[a,c]

[a]Université Côte d'Azur EUR - DS4H; [b]Master 1 Informatique, parcours IA; [c]Master 1 Informatique, parcours Informatique

L'Awalé est un jeu de stratégie tour-à-tour à deux joueurs, avec pour composante des graines en quantité limitée et des trous attribués aux joueurs de manière uniforme, dans lesquels sont réparties les graines. Le but de chaque joueur est de maximiser le nombre de graines récoltées au cours de la partie, grâce à un ensemble de règles simples. Bien que simples, ces règles, une fois combinées, peuvent engendrer des comportements complexes, éloignant le jeu de l'aléatoire et rendant l'établissement de stratégies importantes pour accomplir l'objectif des joueurs. Ce rapport cherche à implémenter des agents autonomes remplaçant les joueurs humains à des fins compétitives, et à explorer diverses stratégies afin d'évaluer la position de chacun. Celui-ci met également en évidence un biais important favorisant le premier joueur, et les plus-values apportées par les différences de profondeur d'exploration de chaque agent.

Awalé | MinMax | Coupe Alpha Beta | Fonction d'évaluation

L'Awalé est un jeu de stratégie tour-à-tour à deux joueurs. Ce jeu se base sur un système de règles simples, construit autour de deux composants : des graines (en quantité limitée) et des trous assignés à chacun des joueurs. Le but d'un joueur est de maximiser le nombre de graines qu'il a récolté, le joueur en ayant le plus à la fin de la partie étant le gagnant du jeu.

Dans ce rapport, la partie 1 présente et explicite les règles du jeu. La partie 2 s'intéresse à l'automatisation du jeu de l'Awalé, ou en d'autres termes, comment développer une IA rapide et efficace. Enfin, la partie 3 met en évidence les stratégies d'évaluation des positions, ou plutôt, comment dire si une position est avantageuse ou, au contraire, désavantageuse.

1. Présentation et règles du jeu

L'Awalé est un jeu de plateau et de stratégie tour-à-tour à deux joueurs d'origine Africaine datant du VIIIème siècle Raabe (2006). Le jeu se compose de deux composant :

- Un plateau constitué de trous,
- Des graines.

Au delà des règles traditionnelles variants de régions en régions, nous détaillerons dans la suite des règles adaptées, ajoutées et/ou simplifiées à des fins d'automatisations.

Plateau et graines. Dans cette version modifiée de l'Awalé, nous utiliserons un plateau composé de 16 trous numérotés de 1 à 16. Les trous impairs appartiennent au premier joueur `[J1]` et les trous pairs au deuxième joueur `[J2]`. Les graines, au nombre de 96, sont réparties de manière égale entre trois couleurs : rouge, bleu et transparent.

Début de la partie. Au début de la partie, deux graines de chaque couleur sont attribuées dans chaque trous du plateau. `[J1]` commence.

Déroulé de la partie. Au tour de `[J1]`, `[J1]` choisi un trou lui appartenant et peut exécuté l'une des actions suivantes (sous réserver que les graines concernées soient dans le trou sélectionné) :

- Jouer les graines rouges `[r]`,
- Jouer les graines bleues `[b]`,
- Jouer les graines transparentes en tant que graines rouges `[tr]`,
- Jouer les graines transparentes en tant que graines bleues `[tb]`.

Si `[J1]` ne peut pas jouer à partir de ces trous à son tour, alors nous sommes dans une situation de famine, le tour de `[J1]` est passé et `[J2]` récupère l'ensemble des graines du plateau. Une fois son coup joué, `[J2]` joue de la même manière et rends la main à `[J1]`.

Remarque 1.1. Un coup est noté `Nc` où `N` est le numéro du trou joué et $\texttt{c} \in \{\texttt{r}, \texttt{b}, \texttt{tr}, \texttt{tb}\}$.

Essaimage. L'essaimage correspond au mécanisme à travers lequel les graines sont jouées. Ainsi, une fois jouées, les graines rouges sont supprimés du trou sélectionné et sont réparties une à une dans les trous suivants en ordre croissant. Les graines bleues suivent un mécanisme similaire, mais ne répartissent les graines que dans les trous ne lui appartenant pas. Enfin, les graines transparentes peuvent êtres jouées soit comme les graines rouges soit comme les graines bleues.

Remarque 1.2. Le trou suivant le trou numéro 16 est le trou numéro 1.

Remarque 1.3. Si un graine transparente est joué, alors les graines de la couleur sélectionnée sont aussi jouées. Ainsi $(\texttt{Play[tr]} \Rightarrow \texttt{Play[r]}) \wedge (\texttt{Play[tb]} \Rightarrow \texttt{Play[b]})$. Si les graines de couleur correspondante ne se trouvent pas dans le trou sélectionné, alors les graines transparentes sont jouées sans impacts sur les autres couleurs.

Remarque 1.4. Si l'essaimage parvient à revenir sur le trou sélectionné, alors celui-ci est sauté et l'essaimage continue sur le trou suivant.

Capture des graines. À la fin de l'essaimage, le trou ayant reçu la dernière graine est vérifié. Si ce trou contient deux ou trois graines, alors la récolte commence et le joueur ayant fait l'essaimage récolte les graines contenu dans le trou et répète l'action pour tous les trous précédents (en ordre décroissant) jusqu'à ce que la condition ne soit plus vérifiée.

Fin de la partie. La fin de la partie peut arriver de plusieurs manières :

- **Limite de coup :** Si 400 coups ont été exécutés (200 coups par joueur), alors la partie s'arrête et le joueur ayant le plus de graine remporte la partie.

- **Limite de graine :** S'il y a moins de dix graines sur le plateau, alors la partie s'arrête et le joueur ayant le plus de graine remporte la partie.
- **Victoire :** Si un joueur a récolté au moins 49 graines, alors le joueur gagne la partie.
- **Égalité :** Dans le cas où la limite de coup ou la limite de graine est dépassée, si aucun joueurs ne domine, alors il y a égalité.

2. Implémentation des MinMax

L'objectif de cette partie est de présenter notre IA pour l'Awalé basée sur l'algorithme MinMax. Cette approche permet à un joueur automatisé d'évaluer les positions possibles jusqu'à une certaine profondeur et de choisir le coup maximisant sa probabilité de victoire.

A. Choix de MinMax et de C++. Nous avons choisi **MinMax** pour sa simplicité conceptuelle et sa capacité à explorer exhaustivement l'espace des coups possibles dans un jeu à information parfaite et à deux joueurs. Le **C++** a été retenu pour sa rapidité et sa gestion fine de la mémoire, essentielle pour les appels récursifs et la manipulation des états de jeu. La compilation avec des flags d'optimisation (`-O2` ou `-O3`) améliore encore les performances, notamment pour les fonctions récursives.

B. MinMax. L'algorithme MinMax explore récursivement toutes les positions possibles jusqu'à une profondeur donnée :

- À chaque niveau, tous les coups possibles du joueur actif sont générés.
- Chaque coup est simulé sur un nouvel état de jeu, et MinMax est appelé sur cet état.
- Lorsque la profondeur maximale est atteinte ou qu'un état terminal est rencontré (victoire, défaite ou égalité), une fonction d'évaluation estime l'avantage pour le joueur maximisant.
- Les valeurs sont propagées vers le haut de l'arbre : le joueur maximisant choisit la valeur la plus élevée, et le joueur minimisant la plus faible.

Bien que MinMax permette de simuler toutes les options, le nombre de positions croît de manière exponentielle avec la profondeur, ce qui peut rendre l'algorithme très lent.

C. Alpha-Beta. MinMax explore toutes les positions jusqu'à une profondeur donnée, mais le nombre de coups croît exponentiellement, ce qui peut être lent. Pour optimiser, nous utilisons **Alpha-Beta pruning**, qui permet d'ignorer les branches de l'arbre ne pouvant pas améliorer le résultat pour le joueur courant.

Deux valeurs, *alpha* et *beta*, représentent les bornes des résultats possibles pour le joueur maximisant et le joueur minimisant. Lorsqu'une branche ne peut plus influencer la décision finale, elle est coupée, réduisant fortement le nombre de simulations nécessaires.

Des fonctions d'évaluation spécifiques estiment la qualité des positions intermédiaires selon la stratégie (attaque, défense, capture potentielle). Cette combinaison d'Alpha-Beta et d'évaluations adaptées permet de prendre des décisions rapides tout en maintenant une qualité stratégique élevée.

D. Profondeur adaptative. La profondeur de recherche n'est pas fixée de manière statique mais adaptée dynamiquement selon l'état du plateau, notamment le nombre de graines restantes. Cette technique, appelée **profondeur adaptative**, permet de concentrer les ressources de calcul lorsque chaque décision est critique.

Lorsque le plateau est très chargé, la profondeur est réduite pour économiser du temps de calcul. En fin de partie, lorsque chaque coup a un impact majeur, la profondeur est augmentée pour explorer plus finement les options stratégiques. Ainsi, l'algorithme reste rapide tout en garantissant une précision élevée.

E. Structure de données. Le cœur de l'IA repose sur la structure `GameState`, qui représente l'état complet du plateau à un instant donné. Elle contient notamment :

- le tableau des trous avec le nombre de graines pour chaque joueur ;
- les compteurs de graines récoltées ;
- l'indication du joueur dont c'est le tour.

Cette abstraction permet de passer un état du jeu aux fonctions récursives sans effets de bord et de simuler des coups indépendamment de l'état principal.

F. Implémentation pratique dans le projet. Dans notre projet, nous avons implémenté MinMax et Alpha-Beta en manipulant la structure `GameState` de manière centralisée. Chaque appel récursif reçoit une copie de l'état du plateau, ce qui permet de simuler un coup sans modifier l'état réel. Cela évite les effets de bord et rend la fonction récursive plus sûre et modulable.

Pour l'**Alpha-Beta**, nous avons introduit les bornes *alpha* et *beta* dès le début de la fonction, et nous les mettons à jour à chaque évaluation. Si une branche ne peut plus améliorer le résultat pour le joueur courant, elle est immédiatement ignorée, ce qui réduit fortement le nombre de coups explorés.

La fonction d'évaluation a été conçue pour être flexible : selon la stratégie choisie (attaque, défense, capture potentielle), elle calcule une valeur estimée de la position pour guider MinMax. Cela nous a permis de combiner rapidité et qualité stratégique.

Enfin, la **profondeur adaptative** a été intégrée en ajustant dynamiquement la profondeur maximale en fonction du nombre de graines restantes sur le plateau. Au début, lorsque les coups possibles sont nombreux, la profondeur est limitée pour gagner du temps. En fin de partie, elle augmente pour permettre une analyse plus précise des positions critiques. Cette approche a été implémentée dans la fonction principale de décision de l'IA, avant chaque appel à Alpha-Beta.

Des optimisations supplémentaires ont été appliquées :

- éviter de recalculer des coups impossibles ou des états déjà évalués ;
- ordonner les coups les plus prometteurs en début de liste pour maximiser les coupures Alpha-Beta ;
- utiliser des copies locales de `GameState` pour réduire les coûts mémoire et les effets de bord.

3. Implémentation des fonctions d'évaluation

Dans le cadre des fonctions d'évaluation, l'objectif est de déterminer les facteurs déterminants contribuant à la victoire d'un joueur si la profondeur d'un arbre MinMax (ou dérivé) n'est pas suffisante pour connaître l'entièreté du jeu.

À termes, une évaluation entre -1 (avantage `[J2]`) et 1 (avantage `[J1]`) est émise pour chaque coup afin d'estimer le mouvement le plus intéressant pour chaque joueur.

A. Une première tentative orienté attaque *raw*. Dans cette première tentative, nous avons d'abords essayer d'identifier des caractéristiques "évidentes" pouvant déterminer la victoire d'un joueur :

- **Nombre de graine récoltées par joueur :** Plus un joueur a récolté de graine, mieux il est positionné pour la victoire. Ainsi, l'évaluation doit favoriser les positions maximisant le nombre de graine récolté par un joueur. Plus précisément, nous nous intéressons aux positions maximisant la différence de graine récoltées par les joueurs. [$\mathbb{R}_1$]
- **Nombre de graine par joueur :** Plus un joueur a de graines dans ces trous, plus il a de marge de manœuvre et donc plus élevée est la probabilité d'avoir des coups intéressants plus tard. Ainsi, nous voulons maximiser le nombre de graine dans les trous d'un joueur. Par soucis d'uniformisation, nous raisonnons comme précédemment sur la différence graine. [$\mathbb{R}_2$]
- **Nombre de graine transparentes par joueur :** Comme pour le nombre de graine dans les trous d'un joueur, plus un joueur a de graines transparentes, plus élevée est la marge de manœuvre en raison de la nature même de ces graines. Nous voulons donc maximiser cette valeur et raisonnerons sur la différence comme précédemment. [$\mathbb{R}_3$]
- **Potentielles captures :** Sûrement la variable la plus importante, il s'agit ici de savoir combien de graines peuvent êtres capturés pour chaque position évalué. Bien sûr, nous cherchons à maximiser ce nombre de potentielles captures. [$\mathbb{R}_4$]

Il ne nous reste maintenant qu'à gérer l'agrégation des valeurs ci-dessus mentionnées. Pour cela, nous appliquons déjà une étape de normalisation où chaque valeur est transposée entre -1 et 1. Ensuite, nous appliquons une simple somme pondéré afin d'obtenir un résultat entre -1 et 1.

Les poids de la sommes ont été ajustés à la main et n'ont pas été optimisés parfaitement en raison de contraintes de temps. Voici la somme utilisé dans le code :

$$raw(\text{pos}) = 0.4\,[\mathbb{R}_1] + 0.2\,[\mathbb{R}_2] + 0.1\,[\mathbb{R}_3] + 0.3\,[\mathbb{R}_4] \qquad [1]$$

En raison d'une mauvaise implémentation de la logique de différence, cette fonction d'évaluation a tendance à faire des erreurs en présentant à `[J1]` des positions favorables pour `[J2]` comme étant favorable pour `[J1]`. Grâce aux poids, cet impact néfaste s'est avéré mitigé, mais à aussi rendu la détection du problème beaucoup plus chronophage lors des tests, ce qui a rendu çà correction tardive. À date de remise du rapport, une version corrigée existe et est sobrement nommé ***corrected***.

B. Deuxième tentative : ALL-IN en défense *defence*. Dans une deuxième fonction d'évaluation, nous cherchons à résoudre un point faible de la fonction d'évaluation précédente : l'attaque est trop privilégié par rapport à la défense, ce qui rend certaines positions plus vulnérables à une attaque ennemi.

Pour résoudre ce problème, ou plutôt l'atténuer, nous faisons l'observation simple que moins un joueur possède de trous avec deux ou trois graines, moins il a de chance de se faire capturer ces graines ou de voir naître une chaîne de récolte trop longue profitant à l'adversaire.

Ainsi, nous cherchons à maximiser le nombre de trous ayant une graine ou plus de trois graines. [$\mathbb{R}_5$]

Une fois normalisé entre -1 et 1, nous avons l'agrégation suivante :

$$defence(\text{pos}) = 0.25\,[\mathbb{R}_1] + 0.06\,[\mathbb{R}_2] + 0.04\,[\mathbb{R}_3] + 0.3\,[\mathbb{R}_4] + 0.25\,[\mathbb{R}_5] \qquad [2]$$

Maintenant que nous avons nos trois fonctions d'évaluation, il nous faut les comparer pour extraire la fonction la plus adaptée dans le cadre d'une compétition.

C. Méthodologie et Résultats. Pour évaluer nos fonctions d'évaluation, nous nous reposerons sur le fait que notre système est complètement déterministe, c'est à dire qu'il n'existe pas dans notre système une séquence d'actions aléatoire rendant notre évaluation stochastique. Cela veut dire que l'évaluation des modèles peut se faire en un unique match singulier pour chaque pairs de fonction d'évaluation. Pour déterminer la meilleur fonction d'évaluation, il nous suffire in-fine de compter le nombre de victoire et de prendre celle maximisant le nombre de victoire.

Concernant les arbres que nous allons utiliser pour ces tests, nous testerons d'abord avec un Alpha-Beta de profondeur fixé à 6, car cette profondeur est déjà bonne et qu'elle dispose d'une stabilité dans les temps de calculs, ensuite nous utiliserons un Alpha-Beta à profondeur variable, cela nous permettant également de vérifier la plus-value d'un tel système.

Sans plus tarder, voici les résultats obtenus avec un Alpha-Beta de profondeur fixé à 6 :

`[J1]` **VS** `[J2]`	***raw***	***defence***	***corrected***	**Victoire** `[J1]`/`[J2]`
raw	**51**/14	**50**/7	**42**/32	**3**/0
defence	**50**/8	31/**50**	18/**50**	1/**2**
corrected	**51**/14	**50**/7	**42**/32	**3**/0
Victoire `[J1]`/`[J2]`	**3**/0	**2**/1	**2**/1	**7**/2

Table 1. Score des matchs entre `[J1]` et `[J2]` sous Alpha-Beta fixe et par fonction d'évaluation.

Sur la table 1, nous observons plusieurs choses intéressantes. Premièrement, au delà des résultats qui nous intéressait initialement, nous observons un potentiel biais en faveur de `[J1]` dans cette version du jeu de l'Awalé. Nous emmétrons donc l'hypothèse suivante, à confirmer dans la suite :

Hypothèse (Biais favorable à `[J1]`)**.** Il existe un biais dans la version modifiée de l'Awalé rendant la victoire de `[J1]` plus simple comparé à celle de `[J2]`.

En faisant abstraction de ce biais, nous observons tout de même deux phénomènes importants :

- La fonction d'évaluation ***raw*** et ***corrected*** sont équivalentes du point de vue de `[J1]` en termes de résultat, mais cette dernière se révèle bien plus efficace au mains de `[J2]` en réduisant le nombre de capture de `[J1]` de 33.1% en moyenne et en augmentant les captures de `[J2]` de 260% en moyenne par rapport à ***raw***.
- De même, la fonction d'évaluation ***defence*** ne semble pas convenir à `[J1]` étant donné que celle-ci se retrouve affaiblis face à une stratégie défensive ou très offensive de l'adversaire. Néanmoins, celle-ci semble bien fonctionner dans les mains de `[J2]` à condition que `[J1]` suive aussi une stratégie défensive.

Au final, en cas d'Alpha-Beta à profondeur fixe, il semble préférable pour les deux joueurs d'exprimer une stratégie

purement offensive, car celle-ci se retrouve être la plus versatile face à des stratégies inconnues.

Maintenant, voyons ce que cela donne pour un Alpha-Beta à profondeur évolutive :

[J1] VS [J2]	*raw*	*defence*	*corrected*	Victoire [J1]/[J2]
raw	**54**/26	**30**/27	**50**/32	**3**/0
defence	**51**/6	**27**/22	**48**/38	**3**/0
corrected	**54**/26	**30**/27	**50**/32	**3**/0
Victoire [J1]/[J2]	**3**/0	**3**/0	**3**/0	**9**/0

Table 2. Score des matchs entre [J1] et [J2] sous Alpha-Beta adaptatif et par fonction d'évaluation.

Ici, nous observons bien le biais mentionnée précédemment d'une manière d'autant plus amplifié par la profondeur. Après en avoir parlé avec d'autres groupes, il s'avère que cette observation est étendue, ce qui nous pousse à confirmer cette hypothèse.

Concernant les résultats en eux même la fonction d'évaluation ***corrected*** reste équivalente à ***raw*** du point de vue de [J1], mais se détache d'autant plus du point de vue de [J2] en maximisant ces captures par rapport à toutes les autres fonctions d'évaluations. Cela confirme son rôle de fonction d'évaluation maîtresse pour maximiser les résultats offensifs.

D'un autre côté, la fonction d'évaluation défensive voit ses capacités augmenter en assurant au global une meilleur minimisation du score adverse contre des stratégies offensives légère ou défensive, mais reste vulnérable contre des fonctions très offensive comme ***corrected*** d'un point de vue de [J1]. Néanmoins, du point de vue de [J2], tout change dans la mesure où l'ensemble des matchs où la stratégie défensive a été utilisé, le score de [J1] a été grandement minimisé par rapport aux autres fonctions d'évaluation, cela avançant ces capacités défensives dans le cas de [J2] uniquement.

Au-delà des résultats présentés, d'autres observations lors des tests restent néanmoins importante à mentionner :

- Le temps de calcul pour atteindre une profondeur élevé est en moyenne stable en dessous de deux secondes, mais dans certaines situations, non-élucidés à date d'écriture, nous observons des pics en temps de calculs pouvant atteindre 5 secondes voir 15 secondes dans le pire des cas observé et ce sans raisons apparentes. Plus généralement, nous observons que plus la profondeur augmente, plus des instabilités dans les temps de calculs apparaissent malgré une moyenne stable. Cela est particulièrement vrai dès que la fonction d'évaluation ***raw*** est impliqué. Cela laisse penser que les erreurs d'implémentations cités dans la sous-section A interagissent mal avec l'Alpha-Beta rendant les coupes moins probables.
- Nous observons également l'apparition de boucles dès que la victoire est détecté par une fonction d'évaluation. Cela est dû au fait que l'on cherche à trouver la victoire sans se soucier du chemin le plus cours pour l'atteindre. En d'autres mots, nous détectons la victoire, mais n'y allons pas. Afin d'y remédier, nous avons essayer plusieurs approches, notamment un compteur vérifiant que la longueur du chemin vers la victoire doit diminuer à chaque itération ou en ajoutant un bonus au chemin les plus cours menant à la victoire dans la fonction d'évaluation ; malheureusement, ces méthodes se sont révélés inefficaces. Néanmoins, cela ne se révèle pas être un gros problème en considérant la limite de coup qui avantage celui qui détecte la victoire le plus tôt, même sans l'atteindre à la fin en cas de boucle.

Remarque 3.1. L'évolution de la profondeur utilisé par les tests est la suivante : Moins de 15% de graines récupéré au total implique une profondeur de 6, moins de 30% une profondeur de 7, moins de 70% une profondeur de 8, moins de 80% une profondeur de 9, au-delà une profondeur de 10. Ces valeurs pourront évoluer lors de la compétition afin de limiter les instabilités dans le temps de calcul.

Maintenant que ces comparaisons ont été faites, il ne nous reste qu'une seule question relative à la valeur ajouté de la profondeur dynamique. Pour savoir si celle-ci est efficace, nous allons profiter du biais envers [J1] et émettre la réflexion suivante : si [J2] en utilisant une profondeur dynamique fait mieux qu'en utilisant une profondeur fixe face à [J1] utilisant une profondeur fixe, alors c'est que le biais a été mitigé et que la profondeur dynamique a une réelle plus-value.

Voici donc les résultats que nous avons eu avec [J1] en Alpha-Beta de profondeur 6 et [J2] en Alpha-Beta à profondeur adaptative :

[J1] VS [J2]	*raw*	*defence*	*corrected*	Victoire [J1]/[J2]
raw	**54**/17	42/**44**	**48**/15	**2**/1
defence	**51**/12	25/**42**	30/**49**	1/**2**
corrected	**54**/17	42/**44**	**52**/15	**2**/1
Victoire [J1]/[J2]	**3**/0	0/**3**	**2**/1	**5**/4

Table 3. Score des matchs entre [J1] sous Alpha-Beta fixe et [J2] sous Alpha-Beta adaptatif et par fonction d'évaluation.

Ici, nous observons bien que le biais est mitigé et même presque renversé en faveur de [J2], la profondeur adaptative a donc réellement une plus-value. Cela se confirme également en observant que l'ensemble des victoires de [J2] est un sur-ensemble des victoires obtenues dans la table 1.

Aussi, nous confirmons notre observations précédente que la fonction d'évaluation défensive du point de vue de [J2] se révèle être la plus efficace et celle qui profite le plus lorsqu'une différence de profondeur est observé entre les deux agents. Cela revient à dire pour caricaturer qu'une bonne défense doit être préparé, tandis qu'une bonne attaque joue sur la vitesse en début de partie, qui semble au fur et à mesure des tests être le moment le plus important pour une stratégie offensive (la stratégie défensive a l'air moins dépendante de l'avancement de la partie).

Dans cette section, nous avons finalement démontré expérimentalement qu'une approche se basant sur une profondeur adaptative possède une réelle valeur ajoutée. De plus, une stratégie très offensive comme ***corrected*** se révèle être le plus versatile et offre de meilleurs résultats dans un environment où l'adversaire possède une profondeur et une stratégie inconnue ce qui justifiera son utilisation pour [J1] et [J2], même si la meilleur fonction d'évaluation pour [J2] semble être la stratégie défensive à grande différence de profondeur.

Conclusion

REMERCIMENTS. Pas de remerciments particulliers

References

Raabe, J. (2006). *Le Jeu de l'Awalé*. Editions L'Harmattan.