Rapport de Projet d'IA : Awalé

LOUNIS Elies OUAHRANI KHALDI Sofiane

13 janvier 2026

Table des matières

1 Introduction

Ce rapport présente l'architecture et les choix algorithmiques de notre IA conçu pour jouer à une variante de l'Awalé (16 trous, graines colorées). L'objectif était de développer une IA performante capable de battre les bots des autres groupes dans un temps de réflexion limité (2 secondes par coup).

2 Fonction d'Évaluation

La fonction d'évaluation est le cœur de notre IA. Elle combine 8 heuristiques distinctes pour estimer la qualité d'une position.

2.1 Heuristiques Principales

2.1.1 1. Différence de Score

Critère principal. Nous calculons la différence de score pondérée selon le stade de la partie (poids de 100 à 200). En fin de partie, ce poids augmente pour forcer la victoire.

2.1.2 2. Contrôle du Plateau (Défense)

Nous valorisons le nombre de graines dans notre camp (+10 par graine de différence). Cela permet de retarder la famine et de conserver du matériel ("munitions") pour de futures attaques.

2.1.3 3. Mobilité

Nous calculons la différence entre notre nombre de coups légaux et celui de l'adversaire. Avoir plus d'options force souvent l'adversaire à réagir plutôt qu'à agir, et évite les situations de blocage (*Zugzwang*).

2.1.4 4. Sécurité (Vulnérabilité)

Un trou est "vulnérable" s'il contient 1 ou 2 graines (prenable au prochain coup). Nous appliquons une pénalité pour chaque trou vulnérable dans notre camp et un bonus pour ceux chez l'adversaire.

2.1.5 5. Potentiel de Capture (Horizon 1)

L'IA regarde "un coup plus loin" pour voir si une capture immédiate est possible. C'est une heuristique gloutonne qui guide la recherche tactique.

2.1.6 6. Trous "Chargés" (Stratégie Long Terme)

Nous comptons le nombre de trous contenant beaucoup de graines (≥ 6). Ces trous représentent un potentiel de gain futur important (Kro ou Grenier) et sont valorisés.

2.1.7 7. Trous Actifs (Début de Partie)

En début de jeu, nous favorisons les trous ayant entre 2 et 10 graines. Cela encourage l'IA à développer son jeu et à ne pas laisser de trous morts (0 ou 1 graine) ou trop chargés trop tôt.

2.1.8 8. Gestion de la Famine et Seuils

— **Anti-Famine :** Pénalité massive (-400) si nous avons trop peu de graines (< 5) alors que l'adversaire en a beaucoup.
— **Seuils de Victoire :** Bonus progressif si nous approchons des 49 points (seuil de victoire absolue).

3 Architecture de l'IA

Notre approche combine des concepts fondamentaux vus en cours avec des techniques avancées issues de la littérature sur la programmation des jeux (Game Programming Patterns).

3.1 Algorithme de Recherche : Negamax Alpha-Beta *(Vu en Cours)*

Nous utilisons l'algorithme **Negamax**, une variante simplifiée du Minimax, couplé à l'élagage **Alpha-Beta**. Ces deux concepts, étudiés en cours, permettent de parcourir l'arbre de jeu efficacement. :

— **Negamax** simplifie l'implémentation en traitant les deux joueurs de manière symétrique ($max(a, b) = -min(-a, -b)$).
— **Alpha-Beta** coupe les branches inutiles (celles que l'adversaire rationnel ne nous laissera jamais jouer), permettant d'explorer plus profondément.

3.2 Optimisations Standards *(Vues en Cours)*

3.2.1 1. Iterative Deepening (Approfondissement Itératif)

Cette technique, également abordée en cours (IDDFS), est cruciale pour la gestion du temps (Time Management). Au lieu de lancer une recherche à une profondeur fixe (ex : 10) au risque de dépasser le temps limite, nous lançons des recherches successives à profondeur progressive (1, 2, 3...). Si le temps est écoulé (timeout), nous renvoyons immédiatement le meilleur coup trouvé à l'itération précédente complète.

3.2.2 2. Table de Transposition

Pour éviter de recalculer des positions identiques atteintes par des ordres de coups différents (transpositions), nous stockons les résultats dans une Table de Transposition (concept vu en cours). Cela nous permet de :

1. Récupérer instantanément l'évaluation d'une position déjà vue.
2. Améliorer l'ordre des coups (Move Ordering) en essayant le meilleur coup mémorisé en premier.

3.3 Optimisations Avancées *(Enrichissements Personnels)*

Pour atteindre un niveau compétitif, nous avons implémenté des structures de données et des algorithmes issus de la littérature spécialisée.

3.3.1 1. Zobrist Hashing & Générateur Xorshift

La gestion efficace de la Table de Transposition repose sur deux composants :

— **Zobrist Hashing** [1] : Permet de calculer une signature unique (Hash 64 bits) du plateau en temps constant $O(1)$ grâce à des opérations XOR incrémentales.

— **Xorshift64** [2] : Pour initialiser les clés de Zobrist, nous utilisons un générateur de nombres pseudo-aléatoires (PRNG) de type Xorshift. Il est extrêmement rapide et suffisant pour éviter les collisions de hachage dans notre contexte.

3.3.2 2. Null Move Pruning (NMP)

Cette heuristique consiste à laisser l'adversaire jouer deux fois. Si notre position reste "trop bonne" (beta-cutoff) même sans jouer, on considère que notre position est gagnante et on coupe la recherche. Cela permet un gain de profondeur significatif.

3.3.3 3. Late Move Reduction (LMR)

Technique moderne consistant à réduire la profondeur de recherche pour les coups jugés moins prometteurs (ceux triés en fin de liste). Cela permet de concentrer les ressources de calcul sur les variantes principales.

3.4 Tri des Coups (Move Ordering)

L'efficacité de l'Alpha-Beta dépend de l'ordre dans lequel on explore les coups. Nous trions les coups dans cet ordre :

1. **Coup de la Table de Transposition :** Le meilleur coup trouvé lors d'une recherche précédente.
2. **Captures :** HEURISTIQUE GLOUTONNE.
3. **Killer Moves :** Coups ayant provoqué une coupure ailleurs à la même profondeur.
4. **Historique :** Coups statistiquement bons.

4 Conclusion

En combinant les acquis du cours (Negamax, Alpha-Beta, Table de Transposition) avec des techniques avancées (Zobrist, NMP, LMR) et une fonction d'évaluation riche (8 paramètres), nous avons construit une IA capable de calculer à grande profondeur (12-15 coups) en moins de 2 secondes.

Références

[1] Wikipedia, *Fonction de hachage de Zobrist.*
`https://fr.wikipedia.org/wiki/Fonction_de_hachage_de_Zobrist`

[2] Wikipedia, *Xorshift RNGs*.
`https://en.wikipedia.org/wiki/Xorshift`

[3] Jean-Charles Régin, *Cours AI Game Programming*, Master 1 Informatique, Université Côte d'Azur.