Projet d'intelligence artificielle pour un jeu de type Awalé

GIACCHERO Baptiste Minh Quan HOANG

Janvier 2026

1 Objectif du projet

L'objectif du projet est de concevoir une intelligence artificielle (IA) pour un jeu inspiré de l'Awalé, en Java, capable de jouer de manière compétitive contre d'autres IA.

Les contraintes principales que nous nous sommes imposés étaient :
- utiliser un algorithme de recherche de type minimax (avec élagage alpha–bêta) ;
- définir une fonction d'évaluation numérique des positions, typiquement dans $[0, 100]$;
- respecter les règles spécifiques fournies (captures sur cases à 2/3 graines, famine, condition de fin, etc.) ;
- produire un code propre, structuré, et un rapport expliquant les choix effectués, en particulier la fonction d'évaluation.

2 Architecture générale du code

Le projet est organisé autour de trois grands ensembles de classes :
- **Modèle du jeu** : classe `Board` (plateau), `Hole` (trou), `Player` (joueur), commandes de coups (`RedMoveCommand`, `BlueMoveCommand`, etc.). Ces classes encapsulent l'état du jeu et la logique de mise à jour après un coup.
- **Contrôleurs** : `RuleController` (règles de capture et de fin de partie), `PlayerController` (interface abstraite), `HumanPlayerController` (lecture clavier) et `MinimaxPlayerController` (IA). L'évaluation est centralisée dans `Evaluator` et `PhaseEvaluator`.
- **Programmes principaux** : `Main` pour jouer (humain vs IA, IA vs humain, IA vs IA, humain vs humain) et `AITournament` pour lancer des tournois automatiques IA vs IA et comparer différentes configurations.

Les coups sont représentés par des commandes (patron Command) qui reçoivent un plateau et des joueurs, appliquent les règles, et retournent un booléen indiquant si le coup est légal. Pour la recherche minimax, nous réutilisons exactement ces commandes sur des copies du plateau et des joueurs, mais en mode "silencieux" (sans affichage), afin de ne pas polluer la sortie pendant les simulations.

3 Algorithme de recherche : minimax + alpha–bêta

L'IA principale est implémentée dans `MinimaxPlayerController`. Elle repose sur :
- une recherche minimax à profondeur limitée, avec élagage alpha–bêta ;
- une limite de temps par coup ;
- une *itérative deepening* : on explore successivement les profondeurs $1, 2, \ldots, d_{\max}$ tant que le temps le permet, en conservant le meilleur coup trouvé à la dernière profondeur complète ;

— un *ordre des coups* amélioré : les coups issus des trous les plus chargés sont explorés en premier, puis affinés par des heuristiques de type *killer move* et *history heuristic* (cf. ci-dessous) ;
— un peu d'aléatoire : lorsque plusieurs coups obtiennent le même meilleur score, l'IA en choisit un au hasard parmi eux, ce qui évite de rejouer toujours exactement la même partie.

L'élagage alpha–bêta est classique : chaque appel récursif reçoit α (meilleure valeur déjà trouvée pour le joueur maximisant) et β (meilleure valeur pour le minimisant). Dès que $\beta \leq \alpha$, on sait que la suite de ce sous-arbre ne peut pas améliorer la décision globale, et on coupe.

Pour accélérer cette coupe, nous utilisons deux techniques simples :
— **Killer move** : pour chaque profondeur et pour chaque joueur, on mémorise un coup qui a provoqué une coupe (β-cut). Lorsqu'on explore d'autres nœuds à la même profondeur, on essaie ce coup en priorité.
— **History heuristic** : on maintient un tableau de scores H[joueur][trou][typeDeCoup] qui est incrémenté chaque fois qu'un coup provoque une coupe, avec un poids croissant avec la profondeur. Les coups ayant souvent provoqué des coupes sont donc tentés en premier.

Ces optimisations n'altèrent pas la correction de minimax, mais permettent de visiter moins de nœuds à profondeur fixée, donc d'explorer plus profondément dans le même temps.

4 Fonction d'évaluation de base

La fonction d'évaluation de base est implémentée dans la classe `Evaluator`. Elle prend en entrée un plateau `Board`, un tableau de joueurs et l'indice du joueur à évaluer i (0 ou 1). Elle renvoie un score entier dans $[0, 100]$ où 0 représente une position très mauvaise pour le joueur i et 100 une position très favorable.

4.1 Quantités de base

On note :
— C_i : nombre de graines déjà capturées par le joueur i ;
— C_j : nombre de graines capturées par l'adversaire $j = 1 - i$;
— B_i : nombre total de graines présentes sur le côté du plateau appartenant au joueur i (somme sur ses trous) ;
— B_j : nombre de graines présentes sur le côté de l'adversaire.

Les captures sont plus importantes que les graines encore sur le plateau, mais ces dernières restent pertinentes (famine potentielle, possibilités de jeu). Nous définissons donc une valeur matérielle pondérée pour chacun :

$$M_i = w_c \cdot C_i + w_b \cdot B_i, \qquad M_j = w_c \cdot C_j + w_b \cdot B_j,$$

où w_c et w_b sont des poids configurables (par exemple $w_c = 1{,}0$ et $w_b = 0{,}7$ pour le profil « Balanced »).

La différence matérielle principale est alors

$$\Delta M = M_i - M_j.$$

4.2 Terme de famine

Le jeu permet la « famine » : si un joueur concentre l'essentiel des graines de son côté, il peut empêcher l'autre de jouer et le forcer dans des captures défavorables. Pour refléter cet aspect, on ajoute un terme de *famine* basé directement sur le déséquilibre des graines sur les côtés :

$$\Delta F = B_i - B_j.$$

Ce terme est pondéré par un coefficient w_f (par exemple $w_f = 0{,}3$). L'idée est que l'on préfère légèrement les positions où l'on a plus de graines de notre côté, même à matériel total égal.

4.3 Terme de mobilité

Afin d'éviter les positions où l'adversaire a beaucoup plus d'options que nous (forte activité potentielle) nous introduisons un terme de *mobilité*. Plutôt que de recalculer exactement tous les coups possibles, nous utilisons une approximation simple : pour chaque trou appartenant au joueur, on compte combien de types de coups sont possibles (R, B, TR, TB) en fonction des graines présentes (rouges, bleues, transparentes).

On obtient ainsi une estimation Mob_i du nombre de coups possibles pour le joueur i et Mob_j pour l'adversaire, puis

$$\Delta\mathrm{Mob} = \mathrm{Mob}_i - \mathrm{Mob}_j.$$

Ce terme est pondéré par un coefficient w_m (dans notre code une constante modérée, par exemple $w_m = 0{,}2$) afin de ne pas dominer les aspects matériels mais de départager des positions proches.

4.4 Combinaison et normalisation

La combinaison finale avant normalisation est :

$$\Delta = \Delta M + w_f \cdot \Delta F + w_m \cdot \Delta\mathrm{Mob}.$$

On centre ensuite la valeur autour de 50 et on applique un facteur d'échelle s pour que les écarts raisonnables de Δ se traduisent par des valeurs dans $[0, 100]$:

$$\mathrm{scoreBrut} = 50 + s \cdot \Delta.$$

Enfin, on *clipse* le résultat dans $[0, 100]$ et on arrondit à l'entier le plus proche :

$$\mathrm{score} = \min\big(100, \max(0, \mathrm{round}(\mathrm{scoreBrut}))\big).$$

Les paramètres typiques pour le profil équilibré (Balanced) sont :

$$w_c = 1{,}0, \quad w_b = 0{,}7, \quad w_f = 0{,}3, \quad s = 0{,}5.$$

Nous avons expérimenté d'autres combinaisons, par exemple « BalancedCapture » (captures un peu plus valorisées) ou « BalancedFamine » (famine plus lourdement pondérée), puis comparé leurs performances via le programme `AITournament`.

4.5 Tentatives abandonnées

Une idée initiale consistait à compter les trous contenant exactement 2 ou 3 graines sur le côté de chaque joueur, en pensant que cela refléterait la vulnérabilité aux captures. Après analyse détaillée des règles et plusieurs tests, nous avons constaté que cette heuristique n'était pas fiable :

- la vulnérabilité d'un trou à 2/3 graines dépend surtout des trajectoires de semis (coups précédents) et pas simplement de « à qui appartient le trou » ;
- le même schéma de capture peut survenir sur les deux côtés ;
- ce compteur ajoutait du bruit sans apporter de gain mesurable dans les tournois.

Nous avons donc retiré ce terme pour simplifier la fonction d'évaluation et éviter les sur-ajustements.

5 Évaluateurs par phases

La fonction `Evaluator` précédente est « statique » : les mêmes poids s'appliquent pendant toute la partie. Intuitivement, cependant, le début, le milieu et la fin de partie ne demandent pas exactement la même stratégie. Nous avons donc introduit `PhaseEvaluator`, qui encapsule plusieurs évaluateurs internes et choisit l'un d'eux en fonction du nombre total de graines restantes sur le plateau.

5.1 Principe

On note T le nombre total de graines encore présentes sur le plateau. Au départ $T \approx 96$, et la partie se termine quand il reste moins de 10 graines (règle imposée). Nous distinguons trois zones :

— **Ouverture** : $T > T_{\text{open}}$;
— **Milieu de partie** : $T_{\text{mid}} < T \leq T_{\text{open}}$;
— **Fin de partie** : $T \leq T_{\text{mid}}$.

Selon le profil choisi, les seuils $(T_{\text{open}}, T_{\text{mid}})$ et les poids internes changent. Le cas « PhaseBalanced » (utilisé comme IA principale dans `Main`) est, par exemple :

— **Ouverture** ($T > 70$) : on valorise plus fortement les graines sur notre côté que les captures ($w_c = 0{,}8$, $w_b = 0{,}9$) afin d'éviter les gros cadeaux précoces.
— **Milieu** ($40 < T \leq 70$) : on revient au profil équilibré « Balanced » ($w_c = 1{,}0$, $w_b = 0{,}7$, $w_f = 0{,}3$).
— **Fin** ($T \leq 40$) : on augmente le poids des captures ($w_c \approx 1{,}3$) et on diminue celui des graines sur le plateau ($w_b \approx 0{,}5$), car à ce stade chaque capture peut décider de la partie.

Formellement, `PhaseEvaluator` contient trois instances de `Evaluator` (ouverture, milieu, fin) et délègue simplement l'appel à l'une d'elles selon T.

5.2 Profils testés

Nous avons décliné plusieurs profils de phases :

— **BALANCED** : le profil décrit ci-dessus, utilisé comme IA forte par défaut (« PhaseBalanced »).
— **AGGRESSIVE_ENDGAME** : ouverture et milieu proches de Balanced, mais fin de partie très agressive avec w_c fortement augmenté (« PhaseAggressive »).
— **SAFE_OPENING** : ouverture très prudente (plus de poids encore sur les graines de notre côté, famine légèrement réduite) avant de revenir à un milieu équilibré.
— **ULTIMATE** : combinaison de SAFE_OPENING en début de partie, d'un milieu inspiré d'AdvancedHeuristic (captures un peu plus valorisées), et d'une fin de partie très agressive.

Le programme `AITournament` (présent dans le dossier code) permet de confronter ces profils entre eux (et contre « AdvancedHeuristic », basé uniquement sur `Evaluator` statique) avec les mêmes paramètres de profondeur et de temps.

Les résultats ont montré un comportement intéressant et non transitif (de type pierre–feuille–ciseaux) : par exemple, PhaseBalanced domine souvent AdvancedHeuristic, PhaseUltimate peut battre Advanced dans certains scénarios, mais PhaseBalanced reste globalement le profil le plus stable. Nous avons donc choisi PhaseBalanced comme configuration finale pour la compétition.

6 Outils d'évaluation de l'IA

Pour comparer systématiquement nos choix d'heuristique, nous avons développé un petit outil de tournoi automatique (classe `AITournament`) qui :

— définit plusieurs configurations d'IA (nom, profondeur de recherche, temps maximal par coup, poids de l'évaluation, profil de phases) ;
— joue un nombre paramétrable de parties IA vs IA entre deux configurations, en alternant qui commence ;
— compte les victoires, défaites et matchs nuls.

Cet outil nous a permis de valider empiriquement les réglages de poids (« Balanced », « BalancedCapture », « BalancedFamine », « AdvancedHeuristic », « PhaseBalanced », etc.) et d'itérer rapidement sur la fonction d'évaluation.

7 Conclusion

Nous avons développé une IA basée sur minimax avec alpha–bêta, enrichie par des techniques classiques d'optimisation (itérative deepening, ordering, killer move, history heuristic) et surtout par une fonction d'évaluation construite par étapes :

— d'abord un simple score matériel (captures + graines sur son côté) ;
— puis l'ajout d'un terme de famine, puis d'un terme de mobilité ;
— enfin, la déclinaison en profils par phases (ouverture, milieu, fin) afin d'adapter les priorités de l'IA au stade de la partie.

Les expérimentations par tournoi IA vs IA ont permis d'identifier une configuration Phase-Balanced comme bon compromis entre agressivité et robustesse, et nous l'avons retenue comme IA « forte » pour les parties contre d'autres groupes.