Développement d'une IA pour Awale
Approche Incrémentale et Optimisations

Romain STEFANI - Guillaume FAURE

1 Introduction

L'objectif du projet était de développer une IA compétitive pour une variante complexe de l'Awale (16 trous, 3 couleurs de graines, captures en cascade) dans le cadre d'un tournoi étudiant. La contrainte principale était de jouer chaque coup en moins de 3 secondes.

L'approche suivie a été incrémentale : partir d'une base simple et fonctionnelle, puis améliorer progressivement en testant chaque modification. À chaque étape, les différentes versions ont été mises en compétition pour valider que les changements apportaient une amélioration réelle.

2 Première Étape : Les Bases

2.1 Moteur de Jeu

Avant de réfléchir à l'IA, il fallait un moteur de jeu solide. Les règles principales sont : 16 trous avec 3 couleurs de graines (rouge, bleu, transparent) ayant des comportements de distribution différents (rouge se distribue dans tous les trous, bleu seulement dans les trous adverses, transparent comme les rouges ou comme les bleus au choix du joueur), capture des trous contenant exactement 2 ou 3 graines avec cascade, et victoire à 49 graines capturées ou moins de 10 graines restantes sur le plateau.

J'ai implémenté en C toutes ces règles : génération des coups légaux, application du sowing avec les 3 couleurs, détection des captures en cascade, et conditions de fin de partie. Le moteur a été testé manuellement avec des parties et les exemples des règles pour s'assurer que tout fonctionnait correctement.

2.2 IA Random

La première IA développée était volontairement basique : elle choisit un coup au hasard parmi tous les coups légaux. Cette approche permet d'avoir de quoi tester que les parties se déroulent bien de bout en bout, et surtout d'établir une baseline pour comparer les futures IAs. Une IA qui réfléchit devrait battre le random très largement.

J'en ai profité pour créer un système de simulation automatique qui lance des centaines de parties entre deux IAs et calcule les statistiques. Ce système allait être indispensable pour tester les améliorations futures.

2.3 Minimax

Première IA intelligente : un Minimax classique. L'algorithme explore l'arbre de jeu à une profondeur fixe (2-3 coups), en alternant entre maximiser mon score et minimiser celui de l'adversaire. La fonction d'évaluation était ultra basique : juste la différence de captures.

Cette version bat largement le random, ce qui valide l'approche. Cependant, l'exploration complète de l'arbre est lente et la profondeur atteinte reste faible.

3 Amélioration : Alpha-Beta

Le problème du Minimax est qu'il explore tout l'arbre sans discrimination. Alpha-Beta permet de couper des branches entières qui ne peuvent pas influencer la décision finale. L'algorithme maintient deux bornes [alpha, beta] : si à un moment une branche donne un score qui sort de ces bornes, l'exploration de cette branche peut être arrêtée car elle ne changera pas la décision.

Les résultats sont immédiats : on explore 50-70% de nœuds en moins que Minimax, ce qui permet d'aller plus profond (4-5 coups) dans le même temps.

3.1 Iterative Deepening

Avec la contrainte de temps de 3 secondes, fixer une profondeur à l'avance n'est pas viable. L'Iterative Deepening résout ce problème en effectuant des recherches successives à profondeur croissante (1, puis 2, puis 3...) jusqu'à épuisement du temps. À chaque itération, le meilleur coup trouvé est sauvegardé. Cette approche garantit qu'on a toujours un coup légal à jouer même si la recherche est interrompue. De plus, les résultats des profondeurs précédentes aident à mieux ordonner les coups pour les profondeurs suivantes.

3.2 Fonction d'Évaluation Améliorée

La simple différence de captures était trop basique. J'ai développé une évaluation plus complète qui prend en compte plusieurs critères. La différence de captures reste le critère principal ($\times 100$), complétée par la différence de graines restantes sur le plateau. Des pénalités sont appliquées pour nos trous à 1 graine car ils sont vulnérables : l'adversaire peut facilement les amener à 2-3 graines et les capturer. À l'inverse, des bonus sont attribués pour les trous adverses à 1 graine (qu'on peut capturer en les amenant à 2-3) et ceux déjà à 2-3 graines (capturables si on termine notre sowing dessus). Enfin, un bonus supplémentaire s'applique en fin de partie quand on a l'avantage en captures.

Cette version bat la version avec évaluation basique.

4 Optimisations Avancées

4.1 Table de Transposition

Dans l'Awale, différents ordres de coups peuvent mener à la même position. Recalculer le score de ces positions à chaque fois représente un gaspillage de ressources. La table de transposition agit comme un cache : elle stocke les positions déjà évaluées via leur hash (64 bits), avec leur score, leur profondeur de recherche, et le meilleur coup trouvé.

Avant d'explorer une position, on vérifie si elle est déjà dans le cache. Si c'est le cas et que la profondeur stockée est suffisante, on réutilise directement le résultat. Cette optimisation est particulièrement utile pour l'Iterative Deepening : les résultats des profondeurs précédentes sont réutilisés pour les profondeurs suivantes, évitant de recalculer les mêmes positions à chaque itération.

4.2 Killer Moves

L'ordre dans lequel on explore les coups est crucial pour Alpha-Beta : explorer le meilleur coup en premier maximise les coupures. Les Killer Moves exploitent cette observation en mémorisant, à chaque profondeur de l'arbre, les 2 derniers coups qui ont causé une coupure. Ces coups sont ensuite explorés en priorité dans les positions suivantes car un coup qui était excellent quelque part a de bonnes chances de l'être ailleurs à la même profondeur.

Cette technique est simple à implémenter et améliore significativement les performances sans coût de calcul supplémentaire.

4.3 Move Ordering

Pour maximiser l'efficacité, les coups sont triés avant exploration selon une hiérarchie claire : d'abord le coup de la table de transposition (priorité maximale), puis les Killer Moves, puis les coups qui capturent des graines, et enfin les autres coups.

Un bon ordering peut multiplier par 2-3 le nombre de coupures obtenues.

5 Ce Qui N'a Pas Marché

5.1 History Heuristic

J'ai testé l'History Heuristic, une technique qui maintient un compteur global pour chaque coup (trou + couleur). Ce compteur s'incrémente quand le coup cause une coupure, et ces scores sont utilisés pour l'ordering.

Les résultats ont été décevants : la performance s'est **dégradée**. La raison est claire : dans ce jeu, un même coup (par exemple "3R") peut être excellent ou catastrophique selon le contexte du plateau. L'historique global crée plus de bruit qu'il n'apporte d'information utile, ce qui dégrade l'ordering au lieu de l'améliorer.

5.2 Autres Tentatives

D'autres optimisations classiques ont été testées, notamment le Null Move Pruning et le Late Move Reduction. Les résultats ont été mitigés : des gains très marginaux voire contre-productifs. Le jeu est trop tactique avec ses captures en cascade pour que ces techniques fonctionnent efficacement.

5.3 Leçon Importante : La Profondeur n'est Pas Tout

Au début du développement, je pensais que l'objectif principal était d'atteindre la profondeur maximale dans l'arbre de recherche. Cette hypothèse s'est révélée fausse. J'ai

observé qu'une profondeur de 8 avec un mauvais move ordering performe moins bien qu'une profondeur de 6 avec un excellent ordering.

L'objectif réel n'est pas d'explorer le plus profond possible, mais d'explorer intelligemment en coupant rapidement les branches inutiles. Le ratio entre les coupures et les nœuds explorés est aussi important que la profondeur brute atteinte.

6 Solution Finale : Principal Variation Search (PVS)

Après avoir optimisé Alpha-Beta au maximum, j'ai testé Principal Variation Search (PVS), une variante qui exploite une observation clé : après un bon move ordering, le premier coup exploré est très souvent le meilleur.

6.1 Principe

PVS explore le premier coup normalement avec une fenêtre complète [alpha, beta]. Pour tous les coups suivants, l'algorithme effectue d'abord une recherche rapide avec une fenêtre quasi-nulle [alpha, alpha+1] pour vérifier rapidement s'ils sont meilleurs. Si un coup dépasse cette fenêtre étroite, il est re-recherché avec la fenêtre complète pour obtenir son score exact.

Cette approche repose sur l'intuition que la fenêtre nulle permet de répondre rapidement à la question "ce coup est-il meilleur que le premier ?". Dans 90% des cas, la réponse est négative et on économise énormément de temps de calcul.

6.2 Résultats

PVS explore environ 40% de nœuds en moins qu'Alpha-Beta optimisé, ce qui permet d'atteindre des profondeurs de 7-8 coups contre 6-7 pour Alpha-Beta. Dans les tournois internes, PVS bat Alpha-Beta de manière convaincante.

6.3 Architecture Finale

L'IA finale combine PVS comme algorithme de base avec l'ensemble des optimisations développées : Table de Transposition de 1M d'entrées, Killer Moves (2 par profondeur), Move Ordering hiérarchisé (TT move >Killers >Captures), Iterative Deepening avec gestion stricte du temps, et la fonction d'évaluation multi-critères.

Les performances typiques sont de 7-8 coups de profondeur, environ 100k nœuds explorés, en 1-2 secondes par coup.

7 Conclusion

L'approche incrémentale suivie tout au long du projet s'est révélée efficace : partir d'un joueur Random, passer par Minimax, puis Alpha-Beta, optimiser progressivement, et enfin tester PVS. À chaque étape, des tournois automatisés ont permis de valider ou invalider les modifications apportées.

Les principales leçons tirées de ce développement sont multiples. D'abord, la mesure systématique est essentielle : chaque optimisation doit être validée par des tests concrets plutôt que par des suppositions théoriques. Ensuite, la profondeur brute d'exploration

n'est pas le seul critère de qualité : un bon move ordering qui maximise les coupures est tout aussi crucial. De plus, les optimisations classiques ne fonctionnent pas universellement : il faut adapter les techniques au jeu spécifique. Enfin, maintenir un code propre et modulaire facilite grandement l'expérimentation et les itérations.

L'IA finale basée sur PVS avec l'ensemble des optimisations s'est montrée compétitive lors du tournoi.