Rapport de Projet

AI Game Programming

CLAVEILLE Hugo

CUVILLIEZ Fiona

Master 1 Informatique

13 janvier 2026

Table des matières

1. Introduction

Dans le cadre de l'Unité d'Enseignement *AI Game Programming* du premier semestre de l'année universitaire 2025–2026, nous avons réalisé un projet consistant à implémenter une intelligence artificielle capable de jouer et de gagner à une variante du jeu de de l'Oware.

1.1 Règles du jeu

Le jeu se déroule sur un plateau composé de 16 trous disposés en cercle et numérotés de 1 à 16 dans le sens horaire. Le joueur 1 contrôle les trous impairs, tandis que le joueur 2 contrôle les trous pairs.

Au début de la partie, chaque trou contient exactement 6 graines :
— 2 graines rouges,
— 2 graines bleues,
— 2 graines transparentes.
Il y a donc un total de 96 graines sur le plateau.

L'objectif est de capturer plus de graines que son adversaire. La partie peut se terminer par une égalité si chaque joueur capture 48 graines.

Lors de son tour, un joueur choisit un trou qu'il contrôle et sélectionne une couleur de graines à jouer parmi celles disponibles dans ce trou.

Les graines sélectionnées sont retirées du trou et semées une par une dans les trous suivants dans le sens horaire, selon les règles de distribution associées à la couleur.

— **Graines rouges** : elles sont distribuées dans tous les trous (alliés et adverses).
— **Graines bleues** : elles sont distribuées uniquement dans les trous de l'adversaire.
— **Graines transparentes** : le joueur choisit si elles doivent être jouées comme des graines rouges ou bleues. Elles restent cependant transparentes après la distribution.

Lorsqu'une couleur est jouée, les graines transparentes correspondantes sont toujours distribuées en premier, suivies des graines de la couleur choisie.

Le trou de départ est toujours ignoré lors de la distribution, même si un tour complet du plateau est effectué.

Un coup est noté sous la forme NC, où N est le numéro du trou et C la couleur jouée. Par exemple :

— `3R` : jouer les graines rouges du trou 3,

— `4TR` : jouer les graines transparentes du trou 4 en les considérant comme rouges.

La capture se déroule dans le sens antihoraire tant que les trous précédents contiennent exactement 2 ou 3 graines, que ce soit les trous adverses ou ses propres trous. Les couleurs des graines n'ont aucune influence sur la capture.

L'affamement de l'adversaire est autorisé. Si un joueur capture toutes les graines restantes de son adversaire, il récupère l'intégralité des graines encore présentes sur le plateau.

La partie se termine lorsque :

— un joueur atteint 49 graines capturées ou plus,

— ou qu'il reste strictement moins de 10 graines sur le plateau.

Dans ce dernier cas, les graines restantes ne sont pas comptabilisées.

Le vainqueur est le joueur ayant capturé le plus de graines.

1.2 Stratégies gagnantes

L'analyse du jeu et des différentes parties jouées nous a permis d'identifier plusieurs stratégies efficaces permettant d'augmenter significativement les chances de victoire :

— **Conservation des graines transparentes** : les graines transparentes offrent une grande flexibilité puisqu'elles peuvent être jouées comme des graines rouges ou bleues. Il est donc avantageux de les conserver le plus longtemps possible, suivies des graines rouges, tandis que les graines bleues sont généralement jouées en priorité car elles profitent principalement à l'adversaire.

— **Affamement de l'adversaire** : tenter de réduire au maximum le nombre de coups disponibles pour l'adversaire permet de limiter ses options stratégiques et d'augmenter les opportunités de capture.

— **Maximisation des coups disponibles** : conserver un maximum de trous jouables de son côté du plateau permet de maintenir une plus grande liberté de décision et de mieux contrôler le rythme de la partie.

— **Optimisation des captures** : l'objectif principal restant la capture de graines, les stratégies favorisant des captures multiples ou en chaîne sont systématiquement privilégiées.

1.3 Déroulement de la compétition

Les matchs opposant les différentes intelligences artificielles se déroulent selon les contraintes suivantes :

— un maximum de 400 coups par partie ;

— une limite de temps de 2 secondes par coup, imposée pour la prise de décision.

Ces contraintes nécessitent une optimisation rigoureuse des algorithmes de recherche et d'évaluation.

2. Intelligence artificielle

2.1 Choix du langage

Le langage `C++` a été choisi pour l'implémentation de notre intelligence artificielle en raison de ses performances élevées et de son contrôle sur la gestion de la mémoire. Ces caractéristiques sont particulièrement importantes dans le cadre d'algorithmes de recherche récursive, tels que Minimax et Alpha-Bêta, où un très grand nombre d'états doit être exploré en un temps limité.

De plus, le `C++` est un langage compilé, ce qui lui confère un avantage significatif en termes de vitesse d'exécution par rapport à des langages interprétés tels que `Python`. Cette rapidité est essentielle pour respecter la contrainte de temps imposée de deux secondes par coup, tout en permettant une exploration plus profonde et plus efficace de l'arbre de recherche.

2.2 Algorithmes utilisés

Nous avons dans un premier temps implémenté l'algorithme *Minimax*, permettant d'explorer l'arbre des coups possibles en supposant un adversaire optimal. Une fonction d'évaluation basée sur des heuristiques simples, telles que la différence de score ou le nombre de graines contrôlées, a été utilisée pour estimer la qualité des états non terminaux.

Afin d'améliorer les performances et de respecter la contrainte temporelle imposée, nous avons ensuite intégré l'algorithme *Alpha-Bêta*, qui constitue une optimisation du Minimax. Cette variante permet d'élaguer une partie significative de l'arbre de recherche en supprimant les branches qui ne peuvent pas influencer le résultat final.

2.3 Heuristiques

L'évaluation d'un état du jeu repose sur une combinaison d'heuristiques pondérées, parmi lesquelles :

— le nombre maximal de graines présentes dans un trou du joueur ;
— le nombre total de graines du côté du joueur ;
— le nombre de coups légaux disponibles pour le joueur ;
— la différence de score entre le joueur et son adversaire ;
— la minimisation du score potentiel de l'adversaire ;
— la pénalisation des trous dits *vulnérables*, contenant une ou deux graines ;
— la possibilité d'affamer l'adversaire ;
— le nombre de graines capturables à court terme ;
— la conservation des graines transparentes et rouges, jugées plus avantageuses ;
— la réduction du nombre de coups disponibles pour l'adversaire.

Ces heuristiques sont combinées à l'aide de coefficients pondérateurs ajustables. Afin de déterminer les pondérations les plus efficaces, plusieurs tournois entre différentes versions du bot ont été organisés, permettant d'analyser l'impact relatif de chaque heuristique sur les performances globales.

2.4 Éviter les pièges

Afin d'éviter que l'intelligence artificielle ne reproduise systématiquement les mêmes parties, une part d'aléatoire est introduite lors du choix des premiers coups. Plus précisément, le coup joué est sélectionné aléatoirement parmi les meilleurs coups évalués.

Par ailleurs, pour limiter l'effet d'horizon inhérent aux algorithmes de recherche à profondeur limitée, un facteur de profondeur est ajouté à la fonction d'évaluation, favorisant les gains rapides et pénalisant les pertes différées.

3. Gagner du temps

3.1 Profondeur maximale dynamique

Le nombre de coups possibles dans le jeu de l'Owalé est particulièrement élevé en début de partie, en raison du grand nombre de graines présentes et des différentes couleurs jouables. De plus casser des symétries semble compliqué et vite inéfficace. Cette explosion combinatoire rend l'utilisation d'une profondeur de recherche fixe rapidement incompatible avec la contrainte de temps imposée de deux secondes par coup.

Une profondeur maximale trop faible (par exemple $p_{max} = 3$) permet de garantir le respect de la contrainte temporelle, mais conduit à des décisions sous-optimales en milieu et en fin de partie, lorsque l'arbre de recherche devient plus restreint. À l'inverse, une profondeur trop élevée en début de partie entraîne un dépassement du temps alloué.

Afin de résoudre ce compromis, nous avons mis en place une stratégie de profondeur maximale dynamique, adaptée à l'état courant du plateau.

Avant chaque prise de décision, le bot estime le nombre moyen de coups possibles pour les deux joueurs à partir de la position courante. Cette estimation fournit une approximation de la taille de l'arbre de recherche, permettant d'anticiper la complexité de l'exploration.

À partir du facteur de branchement estimé et d'un temps moyen mesuré pour l'évaluation d'un état, le nombre minimal d'évaluations réalisables dans le temps imparti est calculé.

Ensuite nous avons implémentés un `itérative deepening` qui d'utiliser le maximum de temps disponible.

3.1.1 Itérative Deepening

Afin d'exploiter au mieux le temps de calcul disponible tout en garantissant une réponse dans la contrainte imposée de deux secondes par coup, nous avons recours à la technique de *l'itérative deepening* (approfondissement itératif).

Le principe consiste à enchaîner plusieurs recherches Minimax (avec élagage alpha-bêta), en augmentant progressivement la profondeur maximale autorisée. La recherche débute avec une profondeur faible (par exemple $p = 1$), puis est relancée avec des profondeurs croissantes tant que le temps restant le permet. À chaque itération, le meilleur coup trouvé est mémorisé, ce qui garantit qu'un coup valide est toujours disponible, même si la recherche doit être interrompue prématurément.

Cette approche présente plusieurs avantages. Tout d'abord, elle permet d'adapter automatiquement la profondeur effective de recherche à la complexité réelle de la position courante : en début de partie, lorsque le facteur de branchement est élevé, seules les premières itérations peuvent être complétées, tandis qu'en milieu et fin de partie, la réduction du nombre de coups légaux permet d'atteindre des profondeurs plus importantes.

Ensuite, l'itérative deepening utilise l'efficacité de l'élagage alpha-bêta. En effet, les résultats des recherches peu profondes fournissent un ordre de coups pertinent pour les recherches plus profondes, ce qui augmente significativement le nombre de coupures et réduit le nombre de nœuds effectivement explorés.

Dans notre implémentation, la recherche est interrompue dès que le temps restant devient insuffisant pour garantir la complétion d'une itération supplémentaire. Le dernier coup entièrement évalué est alors retourné comme décision finale. Cette stratégie permet d'utiliser quasiment l'intégralité du temps disponible, tout en assurant un compromis robuste entre profondeur de recherche et respect des contraintes de temps.

À première vue, cette méthode peut sembler moins efficace qu'une estimation directe de la profondeur optimale à laquelle effectuer la recherche. Toutefois, grâce à la réutilisation systématique du meilleur résultat obtenu lors de l'itération précédente, la perte de performance reste minime. Aprés quelques tests empirique nous n'avons pas trouvé de différence significative, car le plus souvent la profondeur idéale consomme moins que les deux secondes allouées et tandis que la suivante éxplose le plafond.

4. Exécution

4.1 Exécution du projet en condition de compétition

Afin de participer à la compétition, notre intelligence artificielle est fournie sous la forme d'un exécutable conforme au protocole d'entrée/sortie imposé par l'arbitre.

Le code est compilé à l'aide du compilateur `gcc` sous Linux, ce qui génère un exécutable nommé `aigame`. Un exécutable équivalent `aigame.exe` est également fourni pour les environnements Windows.

La compilation peut être effectuée à l'aide de la commande suivante :

```
gcc main.c data.c game.c bot.c logger.c evaluate.c -o aigame
```

Lors de la compétition, l'exécutable est lancé et communique exclusivement via l'entrée standard et la sortie standard, conformément aux règles imposées. L'intelligence artificielle attend un message `START` lorsqu'elle joue en tant que premier joueur, ou bien le premier coup de l'adversaire lorsqu'elle joue en second.

La gestion des parties et l'arbitrage sont assurés par un programme Java fourni, nommé `Arbitre`. L'arbitre peut être compilé et exécuté à l'aide des commandes suivantes :

```
javac Arbitre.java
java Arbitre
```

L'arbitre se charge de lancer les exécutables des deux joueurs, de transmettre les coups, de faire respecter les contraintes de temps (2 secondes par coup) ainsi que la limite maximale de 400 coups par partie, et d'afficher le résultat final de chaque match.

5. Conclusion

Ce projet nous a permis d'explorer concrètement les problématiques liées à la programmation d'intelligences artificielles pour les jeux. La variante du jeu de l'Owalé étudiée, avec l'ajout de plusieurs couleurs de graines et des règles de capture spécifiques, présente une richesse stratégique qui en fait un excellent support pour l'application d'algorithmes de recherche adversariale.

L'implémentation progressive d'un algorithme Minimax, puis de sa version optimisée Alpha-Bêta, nous a permis de constater l'importance des techniques d'élagage pour respecter des contraintes temporelles strictes. Le recours à une profondeur maximale dynamique ainsi qu'à la mémorisation des états explorés s'est avéré essentiel pour améliorer l'efficacité et la stabilité des performances de l'intelligence artificielle.

Par ailleurs, le choix et la pondération des heuristiques ont joué un rôle central dans la qualité du jeu produit. Les tournois internes entre différentes versions du bot ont permis d'identifier les critères les plus déterminants, tels que la gestion des graines transparentes, l'affamement de l'adversaire et la maximisation des opportunités de capture. L'introduction d'une part d'aléatoire dans le choix des coups a également contribué à rendre le comportement du bot moins prévisible en début de partie.

En conclusion, ce projet a constitué une expérience formatrice, tant sur le plan théorique que pratique, en mettant en évidence les enjeux liés à la conception, l'optimisation et l'évaluation d'intelligences artificielles pour les jeux compétitifs.