Projet d'Intelligence Artificielle

Implémentation d'une IA pour le jeu de Dames

Licence MIASHS – Option IA Université Grenoble Alpes

Noé Vander Schueren, Ulysse Ansoux et Léo Maquin-Testud

Semestre 5 – 2025

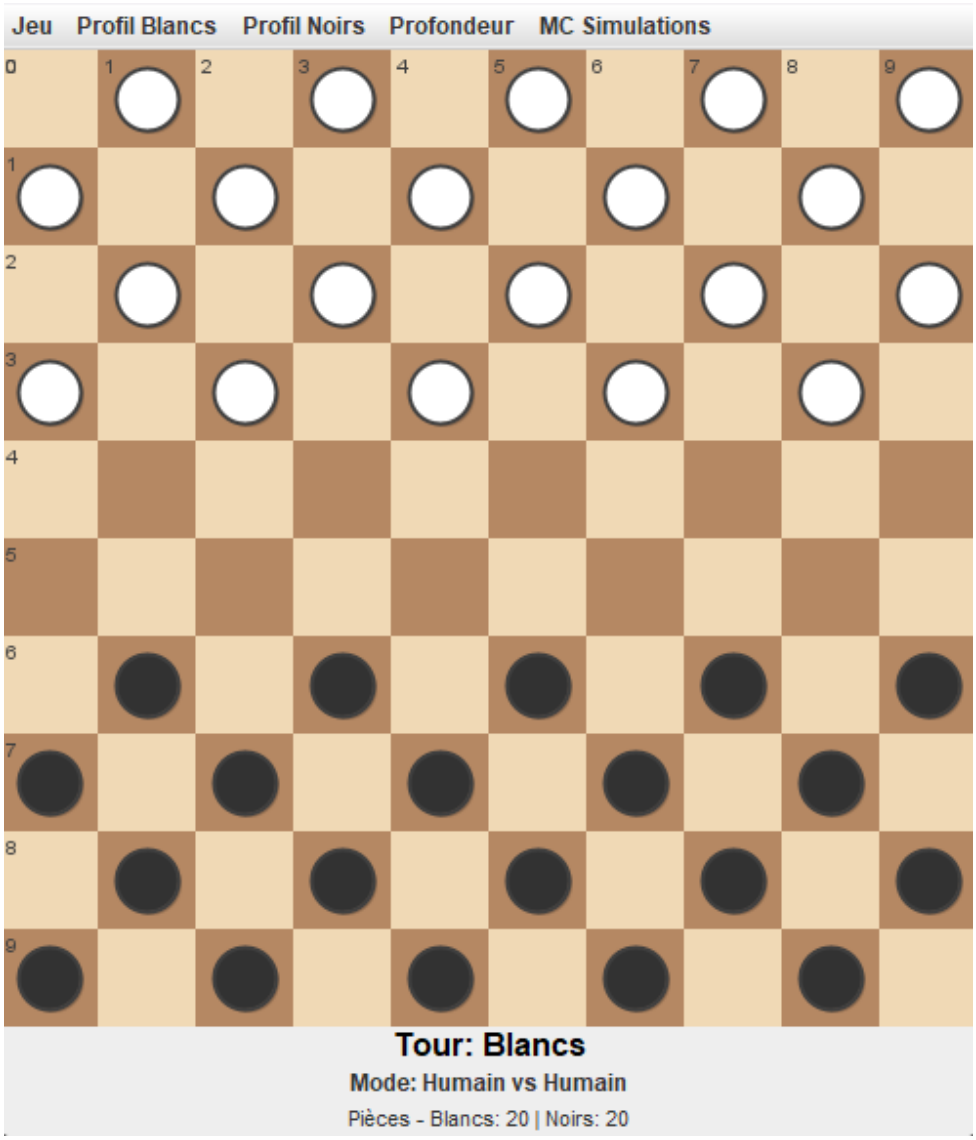

Table des matières

1 Introduction

Ce projet s'inscrit dans le cadre du cours d'Intelligence Artificielle du semestre 5 de la Licence MIASHS. L'objectif principal est de concevoir et d'implémenter une Intelligence Artificielle capable de jouer de manière autonome et compétitive au jeu de dames internationales (plateau 10x10).

Le jeu de dames constitue un problème classique en intelligence artificielle. Il s'agit d'un jeu à information complète, déterministe, avec une forte combinatoire et des règles complexes telles que la prise obligatoire, les prises multiples et les règles d'égalité. Le cœur du projet repose sur l'exploration d'arbres de jeu à l'aide de l'algorithme Minimax avec élagage Alpha-Bêta, enrichi par une fonction d'évaluation heuristique avancée. Une approche alternative basée sur le *Monte Carlo* est également proposée.

2 Règles du jeu de dames internationales

Le jeu de dames internationales se joue sur un plateau de 10,×,10 cases, dont seules les cases sombres sont utilisées. Chaque joueur dispose de 20 pions placés sur les quatre premières rangées de son côté du plateau.

Les règles principales sont les suivantes :

— Les pions se déplacent en diagonale vers l'avant.

— La prise est obligatoire et la prise maximale doit être jouée lorsqu'elle existe.

— Les prises peuvent être multiples au cours d'un même coup.

— Un pion est promu en dame lorsqu'il atteint la dernière rangée adverse.

— Les dames peuvent se déplacer et capturer sur plusieurs cases en diagonale.

Des règles d'égalité sont également implémentées, notamment la répétition de positions, la règle des 25 coups sans prise ni déplacement de pion, ainsi qu'une limite maximale de coups dans les tournois automatisés.

Des règles d'égalité sont également implémentées, notamment la répétition de positions, la règle des 25 coups sans prise ni déplacement de pion, ainsi qu'une limite maximale de coups dans les tournois automatisés.

3 Architecture générale du projet

Le projet est développé en Java et repose sur une architecture modulaire séparant clairement la logique du jeu, l'intelligence artificielle et l'interface graphique. Cette organisation facilite la maintenance, l'extension du projet et la comparaison de différentes stratégies d'IA.

Les principaux modules sont les suivants :

— **Board** : gestion du plateau, des règles et des conditions de fin de partie.

— **Piece** et **Move** : représentation des pièces et des mouvements.

— **IA** : implémentation de Minimax avec élagage Alpha-Bêta et heuristiques.

— **IA MC** : implémentation de l'algorithme Monte Carlo .

— **GameUI** : interface graphique développée avec java Swing.

— **TournoiUI** : gestion des tournois IA vs IA et collecte de statistiques.

4 Représentation du plateau et des pièces

Le plateau de jeu est représenté par une grille 10,×,10. Chaque case peut contenir une pièce blanche ou noire, distinguée selon qu'il s'agit d'un pion ou d'une dame. Cette représentation permet une manipulation efficace des états du jeu lors de l'exploration de l'arbre de recherche.

La classe `Board` assure la génération des coups légaux, la gestion des prises multiples, l'application et l'annulation des coups (fonctionnalité essentielle pour Minimax), ainsi que la détection des états terminaux.

5 Algorithmes d'intelligence artificielle

5.1 Algorithme Minimax

Le jeu de dames internationales peut être formalisé comme un problème de recherche adversariale à deux joueurs, à somme nulle, à information complète et déterministe. Dans ce cadre, chaque état du jeu correspond à une configuration du plateau, et chaque action possible correspond à un coup légal selon les règles du jeu.

L'algorithme Minimax constitue une approche classique pour résoudre ce type de problèmes. Il repose sur l'hypothèse que chaque joueur adopte une stratégie optimale : le joueur courant cherche à maximiser sa récompense, tandis que l'adversaire cherche à la minimiser. Cette modélisation est conforme au cadre général présenté dans *Intelligence Artificielle : A Modern Approach* (Russell & Norvig).

Formellement, un problème de jeu peut être défini par les éléments suivants :

— un ensemble d'états S, incluant un état initial s_0,
— une fonction *Actions*(s) retournant l'ensemble des coups légaux dans l'état s,
— une fonction *Result*(s,a) donnant l'état successeur après l'action a,
— un test terminal *Terminal-Test*(s) indiquant si l'état est final,
— une fonction d'utilité *Utility*(s) attribuant une valeur numérique aux états terminaux.

L'algorithme Minimax explore récursivement l'arbre de jeu engendré par ces transitions. Les nœuds correspondant au joueur maximisant sont appelés *MAX*, tandis que ceux de l'adversaire sont appelés *MIN*. La valeur minimax d'un état s est définie récursivement comme suit :

function MINIMAX-DECISION(state)
  return argmax$_{a \in Actions(state)}$ MIN-VALUE(Result(state, a))

function MAX-VALUE(state)
  if Terminal-Test(state) **then return** Utility(state)
  $v \leftarrow -\infty$
  for each $a \in Actions(state)$ **do**
    $v \leftarrow \max(v,$ MIN-VALUE(Result(state, a)))
  return v

function MIN-VALUE(state)
  if Terminal-Test(state) **then return** Utility(state)
  $v \leftarrow +\infty$
  for each $a \in Actions(state)$ **do**
    $v \leftarrow \min(v,$ MAX-VALUE(Result(state, a)))
  return v

5.2 Élagage Alpha-Bêta

L'élagage Alpha-Bêta permet de réduire considérablement le nombre de nœuds explorés dans l'arbre de jeu. Lorsqu'il est démontré qu'une branche ne peut pas influencer le choix final, celle-ci est ignorée, ce qui permet d'atteindre des profondeurs de recherche plus importantes dans un temps raisonnable.

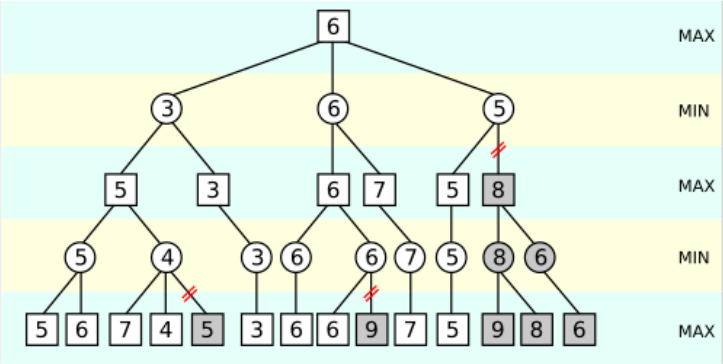

FIGURE 1 – Illustration de l'élagage Alpha-Bêta

5.3 Ordonnancement des coups

Avant l'exploration de l'arbre de recherche, les coups légaux sont triés selon des critères heuristiques afin d'améliorer l'efficacité de l'algorithme Minimax avec élagage Alpha-Bêta. Cet ordonnancement vise à examiner en priorité les coups les plus prometteurs, ce qui augmente la probabilité de provoquer des coupures précoces dans l'arbre de jeu.

Dans un premier temps, les coups correspondant à des captures sont privilégiés, conformément aux règles du jeu de dames et à leur importance stratégique. Ces coups sont évalués en priorité, car ils ont généralement un impact direct et significatif sur la position.

Dans un second temps, lorsque cela est possible, le score Minimax associé à l'état du plateau résultant du coup est récupéré depuis la table de transposition. Si la position enfant a déjà été évaluée lors d'une exploration précédente, son score est utilisé pour affiner le tri des coups. Cette information permet de classer plus précisément les mouvements selon leur potentiel, en s'appuyant sur des évaluations déjà calculées.

Cet ordonnancement des coups améliore sensiblement l'efficacité de l'élagage Alpha-Bêta en favorisant l'exploration rapide des branches les plus favorables, ce qui réduit le nombre total de nœuds visités et permet d'atteindre des profondeurs de recherche plus importantes dans un temps de calcul limité.

5.4 Recherche à approfondissement itératif (Iterative Deepening)

Dans le cadre des jeux adversariaux à forte combinatoire, tels que le jeu de dames internationales, le choix d'une profondeur de recherche fixe peut poser plusieurs problèmes pratiques. Une profondeur trop faible limite la qualité des décisions, tandis qu'une profondeur trop élevée peut entraîner des temps de calcul incompatibles avec une utilisation interactive.

La recherche à approfondissement itératif (*Iterative Deepening*) constitue une solution efficace à ce compromis. Cette technique consiste à exécuter plusieurs recherches successives avec Minimax, en augmentant progressivement la profondeur maximale : $1, 2, 3, \ldots, d$. À chaque itération, le meilleur coup trouvé est conservé et potentiellement réutilisé.

Bien que cette approche semble redondante à première vue, elle présente plusieurs avantages majeurs. Tout d'abord, elle permet d'obtenir à tout moment une solution valide, même si la recherche est interrompue prématurément (par exemple en raison d'une contrainte de temps). De plus, les résultats des recherches précédentes fournissent un excellent ordonnancement des coups pour les recherches plus profondes, ce qui améliore considérablement l'efficacité de l'élagage Alpha-Bêta.

D'un point de vue théorique, Russell et Norvig montrent que, lorsqu'elle est combinée à un ordonnancement efficace des coups, la recherche à approfondissement itératif n'augmente que marginalement le coût total de calcul, tout en maximisant la profondeur atteignable dans un temps donné.

Dans le contexte du jeu de dames, cette méthode est particulièrement pertinente en raison des prises obligatoires et des séquences de captures multiples, qui génèrent des variations tactiques profondes et imprévisibles. L'approfondissement itératif permet ainsi à l'IA d'explorer progressivement ces lignes critiques tout en garantissant une réactivité suffisante.

5.5 Table de transposition

5.5.1 Présentation Globale

La **table de transposition** (aussi appelée *transposition table* ou *cache de positions*) est une structure de données cruciale dans les algorithmes de recherche d'arbres de jeux comme le Minimax. Il s'agit essentiellement d'un **cache intelligent** qui mémorise les positions de jeu déjà évaluées pour éviter de les recalculer.

Pourquoi est-elle nécessaire ? Dans le jeu de dames, il est fréquent qu'une même position soit atteinte par différentes séquences de coups (on parle de **transposition**). Par exemple :

— Coup 1 : A→B puis C→D

— Coup 2 : C→D puis A→B

Ces deux séquences mènent à la même position finale. Sans table de transposition, l'algorithme Minimax évaluerait cette position deux fois, ce qui représente un gaspillage de temps de calcul considérable.

Structure de données utilisée Dans notre projet, la table de transposition est implémentée comme une **HashMap Java** :

private Map<String, CacheEntry> transpositionTable ;

Chaque entrée (`CacheEntry`) contient :

— **score** : l'évaluation minimax de la position

— **move** : le meilleur coup trouvé pour cette position

Clé unique d'identification Pour identifier de manière unique une position, nous utilisons une clé composite :

`clé = hashCode_profondeur_maximizing_couleur`

Où :

— **hashCode** : identifiant unique du plateau (position des pièces) (redéfini dans la classe Board)

— **profondeur** : profondeur de recherche restante

— **maximizing** : true si l'IA cherche à maximiser, false si elle minimise

— **couleur** : couleur de l'IA ('w' ou 'b')

Cette clé garantit qu'une position est bien unique dans son contexte d'évaluation.

5.5.2 Utilisation dans Notre Projet

A. Initialisation La table de transposition est créée au démarrage de l'IA et **vidée au début de chaque nouveau coup.**

Important : Le cache est conservé entre les itérations de l'*iterative deepening* (profondeur 1, 2, 3...), mais **effacé entre deux coups consécutifs**. Cela permet de bénéficier de l'accélération lors de l'approfondissement progressif, tout en évitant d'utiliser des données obsolètes d'un coup précédent.

B. Consultation du cache (lecture) Au début de chaque appel à `minimax()`, l'algorithme vérifie si la position a déjà été évaluée :

```
String key = getCacheKey(board, depth, maximizing);
if (transpositionTable.containsKey(key)) {
    cacheHits++;  // Compteur de performances
    CacheEntry entry = transpositionTable.get(key);
    return new MinimaxResult(entry.score, entry.move);
}
```

(on compare des String donc pas besoin de redéfinir `equals()`)

Si la position est trouvée en cache, **on évite toute la sous-arborescence de recherche** et on retourne directement le résultat mémorisé. C'est un gain de temps considérable.

C. Stockage dans le cache (écriture) Après avoir évalué une position complètement, l'algorithme enregistre le résultat :

Ainsi, si cette position est rencontrée à nouveau (dans une autre branche ou une autre itération), le résultat sera immédiatement disponible.

D. Optimisation du tri des coups (*Move Ordering*) La table de transposition joue aussi un rôle dans l'**ordonnancement des coups**. Pour maximiser l'élagage Alpha-Beta, nous testons en priorité :

1. **Les captures** (poids × 50)
2. **Les coups dont la position enfant est en cache** (on utilise leur score)

En testant d'abord les coups les plus prometteurs, on provoque davantage de coupures Alpha-Beta, réduisant ainsi le nombre de nœuds explorés.

5.5.3 Gains de Performance

Lors de nos tests de performance (détaillés dans `PERFORMANCE_ANALYSIS. md`), nous avons mesuré l'impact de la table de transposition :

Profondeur	Nœuds visités	Cache hits	Gain
4	486	47	∼10%
6	3 311	113	∼3%
8	15 000+	800+	∼5%

TABLE 1 – Impact de la table de transposition sur les performances

Observations :
— Le nombre de *cache hits* augmente avec la profondeur
— L'*iterative deepening* remplit progressivement le cache (profondeur 1→2→3...)
— Les coupures Alpha-Beta sont plus nombreuses grâce au *move ordering* basé sur le cache

5.5.4 Limitations et Choix de Conception

Pourquoi vider le cache entre les coups ? Nous avons choisi de **ne pas conserver** la table entre deux coups successifs pour plusieurs raisons :

1. **Simplicité** : évite de gérer la validité des entrées après un changement de plateau
2. **Pertinence** : une position évaluée à profondeur 6 il y a 3 coups n'est plus forcément pertinente
3. **Mémoire** : évite l'accumulation d'entrées obsolètes

Gestion de la mémoire Avec une `HashMap` Java, la gestion de la mémoire est automatique. En pratique :
— Profondeur 6 : ∼3 000 entrées
— Profondeur 8 : ∼15 000 entrées
— Mémoire utilisée : négligeable (< 10 Mo)

Collisions de hash Les collisions (deux positions différentes avec le même `hashCode`) sont théoriquement possibles mais **extrêmement rares** en pratique grâce à la clé composite (hash + profondeur + maximizing + couleur).

5.5.5 Conclusion

La table de transposition est une optimisation **essentielle** pour rendre l'IA performante. Dans notre projet, elle permet :

- ✓ D'éviter les recalculs de positions identiques (gain de temps majeur)
- ✓ D'améliorer le *move ordering* pour maximiser l'élagage Alpha-Beta
- ✓ De supporter l'*iterative deepening* efficacement

5.6 Monte Carlo Tree Search (MCTS)

5.6.1 Principe Général

En complément de l'approche Minimax, une intelligence artificielle basée sur la méthode **Monte Carlo Tree Search (MCTS)** est intégrée dans notre projet. Cette méthode repose sur un paradigme radicalement différent : plutôt que d'évaluer les positions avec des heuristiques, elle estime la qualité des coups par des **simulations aléatoires** de parties complètes.

Le principe est simple mais puissant : pour chaque coup candidat, l'algorithme simule des centaines de parties jusqu'à leur conclusion (victoire, défaite ou match nul), puis sélectionne le coup ayant obtenu le meilleur taux de victoire statistique.

5.6.2 Avantages par Rapport à Minimax

Cette approche présente plusieurs atouts complémentaires à Minimax :

— **Indépendance heuristique** : Aucune fonction d'évaluation n'est nécessaire. L'IA apprend directement des résultats concrets des parties simulées.

— **Efficacité tactique** : Particulièrement performante dans les positions complexes où les heuristiques de Minimax peuvent être trompeuses (sacrifices, pièges tactiques).

— **Évaluation objective** : Les résultats sont basés sur des fins de partie réelles, pas sur une estimation intermédiaire.

— **Simplicité conceptuelle** : Pas besoin de définir des poids pour chaque aspect du jeu (matériel, position, mobilité, etc.).

5.6.3 Implémentation dans Notre Projet

Notre implémentation de MCTS suit un cycle en trois phases pour chaque coup à jouer :

1. Sélection des coups candidats L'algorithme récupère tous les coups légaux depuis la position actuelle. Contrairement à Minimax qui explore un arbre en profondeur, MCTS évalue chaque coup de manière indépendante.

2. Simulations aléatoires (*rollouts*) Pour chaque coup candidat :

1. Appliquer le coup sur une copie du plateau
2. Simuler une partie complète en jouant aléatoirement pour les deux camps
3. Enregistrer le résultat final
4. Répéter ce processus N fois (par défaut $N = 300$ simulations)

3. Choix du meilleur coup Le coup ayant obtenu le **taux de victoire le plus élevé** est sélectionné. En cas d'égalité, un choix aléatoire est effectué pour éviter la prévisibilité.

5.6.4 Configuration et Performance

Le nombre de simulations est un paramètre crucial qui influence directement la qualité de jeu et le temps de calcul :

Simulations	**Temps (ms)**	**Qualité de jeu**
100	∼190	Faible (aléatoire)
300	∼300	Moyenne (défaut)
500	∼470	Bonne
1000	∼930	Très bonne
2000	∼1850	Excellence

TABLE 2 – Impact du nombre de simulations sur les performances MCTS

Le temps de calcul est **linéaire** par rapport au nombre de simulations, ce qui rend l'algorithme très prévisible et configurable selon les contraintes temporelles.

5.6.5 Comparaison avec Minimax

Dans notre projet, les deux approches coexistent et offrent des profils complémentaires :

Critère	**Minimax**	**Monte Carlo**
Base d'évaluation	Heuristiques	Simulations
Profondeur typique	6-8 coups	Jusqu'à la fin
Temps (défaut)	20-200ms	300ms
Points forts	Tactique précise	Stratégie globale
Points faibles	Horizon limité	Variance statistique
Configuration	Poids heuristiques	Nb. simulations

TABLE 3 – Comparaison Minimax vs Monte Carlo

5.6.6 Cas d'Usage Recommandés

Monte Carlo excelle dans :

— Les positions ouvertes avec de nombreuses possibilités
— Les fins de partie complexes (plusieurs dames en mouvement)
— Les situations où l'évaluation heuristique est incertaine
— L'apprentissage et les tests de robustesse

Minimax est préférable pour :

— Les calculs tactiques précis (séquences de captures)
— Les parties à temps limité (plus rapide à profondeur fixe)
— Les positions fermées où les simulations aléatoires sont biaisées
— La compétition contre des joueurs expérimentés

5.6.7 Intégration dans l'Interface

Dans notre système de tournoi (`TournoiUI. java`), Monte Carlo est proposé comme profil d'IA à part entière (**IA _MC**), avec un réglage configurable du nombre de simulations (50 à 2000). Cela permet de comparer directement ses performances contre les 8 profils Minimax dans des conditions de tournoi standardisées.

6 Passage de Python à Java : motivations et impact sur les performances

Au cours du développement du projet, un choix technique structurant a été opéré concernant le langage de programmation utilisé pour l'implémentation de l'intelligence artificielle. Dans une première phase, le projet a été développé en Python. Ce choix initial s'expliquait par plusieurs facteurs. Tout d'abord, le projet constituait une opportunité de découvrir et de pratiquer le langage Python dans un cadre applicatif concret. De plus, l'écosystème Python propose de nombreuses bibliothèques dédiées aux interfaces graphiques, ce qui facilitait la réalisation de l'interface utilisateur, un critère explicitement évalué dans le cadre du projet encadré par Madame Dima. Enfin, plusieurs algorithmes de recherche nous avaient été fournis sous forme d'implémentations Python, ce qui a naturellement orienté nos premiers développements dans ce langage.

Cependant, au fil de l'avancement du projet, nous avons progressivement pris conscience des limites de Python pour ce type d'application. Le jeu de dames internationales présente une forte combinatoire, notamment en raison des prises obligatoires et des séquences de captures multiples, ce qui entraîne une explosion du nombre de nœuds explorés lors de l'utilisation d'algorithmes de recherche tels que Minimax avec élagage Alpha-Bêta. Dans ce contexte, les performances du langage et la gestion des ressources deviennent des éléments déterminants.

Le déclic pour l'abandon de Python est survenu à la suite d'une discussion avec Monsieur Pellier, au cours de laquelle il est apparu que l'utilisation de Python présentait un intérêt limité pour un projet reposant principalement sur du calcul intensif et de l'exploration d'arbres de jeu. À l'inverse, le langage Java offrait un meilleur compromis entre performances, robustesse et maîtrise technique de l'équipe. Nous avons donc décidé de migrer l'ensemble du projet vers Java. Cette transition a été facilitée par l'utilisation d'une intelligence artificielle générative afin de convertir les classes Python existantes en classes Java, ce qui nous a permis de conserver la structure générale du projet tout en bénéficiant des avantages du nouveau langage.

Ce changement de langage s'est avéré particulièrement judicieux du point de vue des performances. Les analyses réalisées dans le dossier *performance analysis* mettent en évidence des gains significatifs. À titre d'exemple, une recherche Minimax avec élagage Alpha-Bêta et approfondissement itératif jusqu'à une profondeur 4 s'exécute en Java en environ 22 ms, pour 677 nœuds explorés. À profondeur 6, le temps de calcul reste inférieur à 100 ms (environ 84 ms) pour 3311 nœuds explorés, tout en bénéficiant de l'efficacité de la table de transposition, avec plus d'une centaine de réutilisations de positions déjà évaluées. À titre de comparaison, une implémentation équivalente en Python nécessiterait typiquement plusieurs centaines de millisecondes, voire plusieurs secondes, pour des profondeurs bien plus faibles.

Les estimations montrent ainsi que Java est de l'ordre de 50 à 100 fois plus rapide que

Python pour ce type d'algorithmes de recherche. Cette différence s'explique notamment par la compilation Just-In-Time (JIT) de la machine virtuelle Java, une gestion mémoire plus efficace, l'utilisation de types primitifs, ainsi que l'optimisation des boucles et des appels de fonctions. Grâce à ces performances, des profondeurs de recherche de l'ordre de 8 deviennent atteignables en Java dans des temps compatibles avec une utilisation interactive, ce qui serait difficilement envisageable en Python dans le même cadre.

De manière similaire, l'algorithme Monte Carlo bénéficie pleinement du passage à Java. Par exemple, 300 simulations complètes sont exécutées en environ 300 à 350 ms, avec un comportement stable et une diversité satisfaisante des coups choisis. Cette efficacité permet d'envisager l'utilisation de cette approche alternative sans dégrader la fluidité de l'interface ou la réactivité globale du programme.

7 Fonction d'évaluation heuristique

Dans le cadre de l'algorithme Minimax, une fonction d'évaluation heuristique est utilisée afin d'estimer la qualité des positions non terminales du jeu. Cette fonction attribue un score numérique à un état du plateau, permettant de guider la recherche lorsque la profondeur maximale est atteinte.

La conception de cette fonction s'appuie à la fois sur des approches classiques utilisées dans les programmes de jeu de dames de haut niveau, ainsi que sur des principes stratégiques communément admis par les joueurs expérimentés. En particulier, certaines idées sont inspirées de travaux historiques dans le domaine, tels que le programme Chinook, ainsi que de stratégies usuelles décrites dans des ressources pédagogiques accessibles en ligne.

L'évaluation repose sur une combinaison pondérée de plusieurs critères simples mais complémentaires, parmi lesquels :

— le matériel (différenciation entre pions et dames),
— le contrôle du centre du plateau,
— la mobilité des pièces,
— la structure et la sécurité des pions,
— l'activité des dames,
— la progression des pions vers la promotion,
— les situations de blocage des dames.

Chaque critère contribue au score global selon un poids fixé empiriquement. Ces poids peuvent varier selon le profil d'IA sélectionné, ce qui permet de produire des styles de jeu différents (plus agressifs, plus défensifs ou plus positionnels). Cette approche offre un bon compromis entre simplicité de mise en œuvre, interprétabilité des décisions et efficacité pratique.

Bien que cette fonction d'évaluation fournisse une approximation pertinente de la qualité des positions, elle reste intrinsèquement limitée par son caractère heuristique. Elle constitue néanmoins un élément central du comportement de l'IA et conditionne fortement les choix effectués par l'algorithme Minimax.

Document précis de présentation des heuristiques :
HEURISTIQUES.md

8 Choix des poids

Afin de valoriser efficacement les différentes heuristiques de la fonction d'évaluation, nous avons défini un ensemble de **huit profils d'IA** correspondant à des styles et des niveaux de jeu variés. L'objectif était d'observer l'impact du réglage des poids heuristiques sur le comportement et les performances de l'intelligence artificielle.

Dans un premier temps, nous avons envisagé une approche simple consistant à définir des profils correspondant à des niveaux de jeu classiques : *débutant*, *intermédiaire* et *expert*. Cette approche s'est toutefois révélée insuffisante, car les performances ne reflétaient pas toujours la hiérarchie attendue entre les niveaux. En particulier, le profil débutant obtenait des résultats anormalement élevés, en raison d'un poids important accordé à une heuristique simple mais extrêmement efficace : le matériel. Cette observation a mis en évidence le fait qu'un réglage naïf des poids ne permet pas de caractériser correctement un niveau de jeu.

Dans un second temps, nous avons introduit des profils correspondant à des **styles de jeu** distincts, tels que les profils *agressif* et *défensif*. Pour ces profils, les poids ont été attribués en privilégiant les heuristiques naturellement associées au style visé (par exemple les captures et l'activité pour un jeu agressif, ou la sécurité et la structure pour un jeu défensif).

Nous avons également implémenté trois profils particuliers afin d'analyser plus finement l'influence des poids :

- un profil à **poids aléatoires**, destiné à observer l'impact global du réglage des heuristiques sur les résultats ;
- un profil à **poids nuls**, dont les décisions reposent uniquement sur le hasard, servant de référence minimale.
- un profil à **poids tous égaux**, dont les décisions reposent uniquement sur le score attribué à nos heuristiques.

Face aux difficultés rencontrées avec les profils de niveaux, nous avons adopté une approche expérimentale pour classer les heuristiques selon leur efficacité relative. Pour cela, nous avons organisé une série de tournois dans lesquels chaque profil ne valorisait qu'une seule heuristique, avec un poids élevé, différente pour chaque profil. À l'issue de chaque tournoi, le profil vainqueur était retiré de la compétition et le poids associé à son heuristique était redistribué aux autres profils. Cette procédure a été répétée jusqu'à obtenir un classement des heuristiques en fonction de leurs performances observées en tournoi.

Sur la base de ce classement, nous avons ajusté les profils de niveaux. Le profil débutant a été redéfini comme un profil mettant fortement en valeur les heuristiques les moins performantes, tandis que le profil expert privilégie les heuristiques les plus efficaces. Cette méthodologie nous a permis d'obtenir une hiérarchie de profils plus cohérente, tant en termes de résultats que de styles de jeu observés.

Au final, cette démarche a conduit à la définition de huit profils d'IA distincts, offrant une diversité de comportements et permettant une analyse fine de l'impact des heuristiques et de leurs poids sur les performances de l'intelligence artificielle.

9 Interface graphique et interaction utilisateur

Il est important de préciser que l'interface graphique ainsi que l'implémentation complète du jeu de dames n'ont pas constitué le cœur de ce projet. L'objectif principal était avant tout l'étude, la conception et l'implémentation d'algorithmes d'intelligence artificielle appliqués à un jeu adversarial complexe, en particulier les algorithmes Minimax avec élagage Alpha-Bêta et Monte Carlo.

Dans cette optique, le jeu de dames et son interface graphique ont été considérés comme des supports permettant de mettre en œuvre et d'évaluer ces algorithmes, plutôt que comme des fins en soi. La priorité a donc été donnée à la qualité de l'architecture logicielle, à l'efficacité des algorithmes de recherche, à la conception de la fonction d'évaluation heuristique, ainsi qu'à l'analyse des performances et des comportements des différentes intelligences artificielles.

Afin d'optimiser le temps de développement et de concentrer les efforts sur les aspects centraux du projet, nous avons fait le choix d'utiliser une intelligence artificielle générative (Copilot) pour nous assister dans l'implémentation du moteur du jeu, de l'interface graphique ainsi que du système de tournois automatisés. Cette assistance a permis de générer une base fonctionnelle et cohérente, que nous avons ensuite adaptée, intégrée et corrigée afin qu'elle réponde précisément aux besoins du projet et aux contraintes des algorithmes d'intelligence artificielle développés.

Ce choix méthodologique nous a permis de nous concentrer pleinement sur les problématiques fondamentales de l'intelligence artificielle abordées dans le cadre du cours, tout en disposant d'un environnement de test complet et interactif. L'interface graphique, développée en Java avec la bibliothèque Swing, offre ainsi une visualisation claire du plateau, des coups possibles et du déroulement des parties, sans chercher à atteindre un niveau de sophistication graphique élevé. De même, le système de tournois IA vs IA a été conçu comme un outil d'analyse expérimentale permettant de comparer objectivement les performances et les styles de jeu des différentes intelligences artificielles.

10 Analyse des performances et tournois IA vs IA

Un module de tournoi automatisé permet de confronter les différents profils d'IA dans un format *round-robin*. Les statistiques collectées incluent les taux de victoire, les matchs nuls, le nombre de nœuds explorés et les temps de calcul.

Les résultats montrent que l'augmentation de la profondeur de recherche améliore significativement la qualité du jeu, au prix d'un temps de calcul croissant.
Différents résultats de tournoi se trouve sur la page Github. Nos profiles ont bien les résultats attendus, cependant il semblerait que si on augmente en profondeur, certains profiles changent de position (à la profondeur 6, intermédiaire dépasse expert) .
On voit aussi que les stratégies agressives sont plus efficace que les défensives. Avec le profile agréssif en haut de classement et le profile défensif en bas de classement.

11 Conclusion et perspectives

Ce projet a permis de développer une implémentation complète du jeu de dames internationales intégrant des techniques avancées d'intelligence artificielle. L'utilisation de Minimax avec élagage Alpha-Bêta, combinée à une fonction d'évaluation riche, aboutit à une IA performante et configurable.

Parmi les perspectives d'amélioration figurent l'intégration d'un hashage de type Zobrist pour la table de transposition, l'ajout de bibliothèques d'ouvertures et de bases de fins de partie.
On peut aussi penser à ajouter du machine learning et du renforcement, en faisant jouer des IA avec des poids aléatoires et en modifiant les poids entre chaque partie en fonction du résultat.

12 Sources

Intelligence Artificielle : A Modern Approach (Russell & Norvig).

12.1 Règles du jeu :

— `http://www.ffjd.fr/Web/index.php?page=reglesdujeu`

12.2 Sources d'inspiration pour les heuristiques :

— `https://fr.wikipedia.org/wiki/Chinook_(programme_de_dames)`
— `https://www.coolmathgames.com/fr/blog/construire-votre-strat%C3%A9gie-de-dames`

12.3 Documents md :

(Aide de copilot pour la production de ces documents)
— HEURISTIQUES.md
— TOURNOI.md
— PERFORMANCE-ANALYSIS.md
— DOCUMENTATION-COMPLETE.md
— README.md

12.4 Utilisation de LLM :

— Coplilot
— ChatGPT

12.5 Vidéo pour code du jeu :

— `https://www.youtube.com/watch?v=cpN1DErdfZs`