

Projet Intelligence Artificielle

Implémentation d'une intelligence artificielle dans le jeu Awalé

Rapport d'activité

Licence MIASHS
Université Grenoble Alpes
Semestre 5 – 2025

Ducruet Pierre-Léo
Guibert Maëlle
Lama Adodo Joyce
Perrier Albane

Engagement de non-plagiat

Nous soussigné.e.s, Ducruet Pierre-Léo, Guibert Maëlle, Lama Adodo Joyce, Perrier Albane, étudiant.e-s en licence MIASHS, déclarons sur l'honneur que ce document ne comporte aucun plagiat intégral ou partiel de ressources publiées sur toute plateforme de support ;

- Son contenu (textes, images, graphiques ...) cite toutes les sources utilisées pour sa rédaction et uniquement celles consultées ;
- Il reflète le travail personnel de chacun des membres signataires à parts égales.

Nous sommes pleinement conscient.es que le non-respect de cet engagement nous rend passibles de poursuites devant la commission disciplinaire de l'UGA, voire devant les tribunaux de la République française. Nous autorisons l'analyse de tous les documents fournis à l'aide du logiciel compilatio.net avec lequel ils seront archivés.

Fait à Saint Martin d'Hères, le 15/12/2025.

Ducruet Pierre-Léo - Guibert Maëlle - Lama Adodo Joyce - Perrier Albane



Table des matières

Engagement de non-plagiat	2
Table des matières	3
1 - INTRODUCTION	4
2 - RÈGLES DU JEU	5
3 - GESTION DU PROJET	7
4 - LE JEU "AWALE"	8
5- FONCTION HEURISTIQUE	9
6 - GREEDY BEST FIRST	12
7 - Monte Carlo Tree Search (MCTS)	13
8 - L'INTERFACE	16
9 - LES INTERACTIONS ENTRE LES DIFFÉRENTES STRATÉGIES	17
10 - CONCLUSION	18

1 - INTRODUCTION

Dans le cadre de notre licence MIASHS, nous avons eu l'opportunité d'être initiés au monde de l'intelligence artificielle. Aujourd'hui, l'intelligence artificielle (IA) occupe une très grande place dans notre quotidien.

Le cours d'initiation à l'intelligence artificielle a pour objectif de nous présenter les notions fondamentales de l'IA, son fonctionnement général ainsi que ses principaux domaines d'application. Au cours du semestre, nous nous sommes particulièrement focalisés sur l'aspect jeux de l'IA, ce qui nous a permis de nous concentrer sur l'étude des algorithmes de recherche et de prises de décision. Les connaissances acquises au cours du semestre nous ont permis de réaliser ce projet dont l'objectif est de concevoir une interface qui inclut une intelligence artificielle capable de jouer en fonction de stratégies confectionnées par nos soins.

Nous avons choisi le jeu Awalé. C'est un jeu de stratégie traditionnel, qui mélange simplicité des règles et richesse tactique.

Afin de concevoir une IA performante, nous avons implémenté différentes approches de recherche et de prise de décision, notamment la stratégie Greedy Best-First Search ainsi qu'un algorithme Monte Carlo, basé sur une simulation de plusieurs parties afin de trouver le meilleur coup.

Dans la suite de ce rapport, nous présenterons les règles du jeu "**Awalé**" ainsi que les algorithmes que nous avons développés dans le cadre de ce projet.

- **Objectif** : concevoir une IA capable de jouer à l'Awalé en utilisant différentes stratégies.
- **Approches utilisées** :
 - Greedy Best-First Search
 - Monte Carlo Tree Search (MCTS)
 - Fonctions heuristiques développées par le groupe et avec l'aide du professeur.
- **Motivation** : combiner compréhension du jeu et notions d'IA apprises en cours.

2 - RÈGLES DU JEU

1 - But du jeu :

Le but du jeu est de s'emparer d'un maximum de graines. Le joueur qui atteint les 24 graines en premier remporte la partie.

2 - Terrain de jeu :

Le terrain de jeu est divisé en deux territoires de 6 trous chacun.

Au départ dans les douze trous sont réparties équitablement les 48 graines, donc 4 graines par trou.

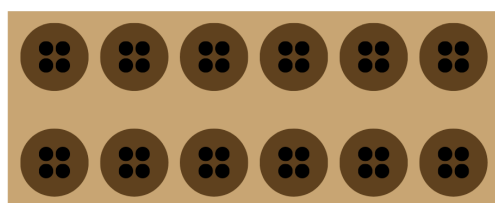


FIGURE 1 : PLATEAU DE JEU

3 - Jouer un tour :

Chaque joueur joue tour à tour.

Le joueur choisit un trou de son territoire, et va distribuer l'ensemble des graines présentes, en les déposant une à une dans chacun des trous suivants, dans le sens inverse des aiguilles d'une montre.

Attention si le joueur possède assez de graines dans ce trou pour faire un tour complet ou plus, il devra sauter le trou duquel proviennent les graines.

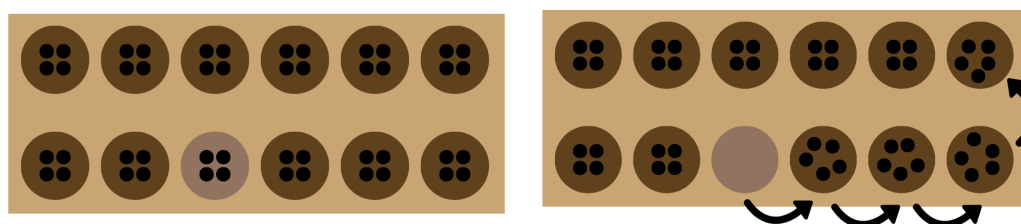


FIGURE 2 : PLATEAU DE JEU PENDANT LA PARTIE

4 - Règle de capture des graines

Si la dernière graine distribuée tombe dans un trou de l'adversaire comportant déjà 1 ou 2 graines, le joueur capture les 2 ou 3 graines résultantes. Si la case précédente contient également deux ou trois graines, elles sont capturées aussi, et ainsi de suite, jusqu'à ce qu'il n'y ait plus de trou avec 2 ou 3 graines ou que le joueur atteigne son territoire. Les graines capturées sont sorties du jeu et additionnées au score du joueur. (Le trou est alors laissé vide)

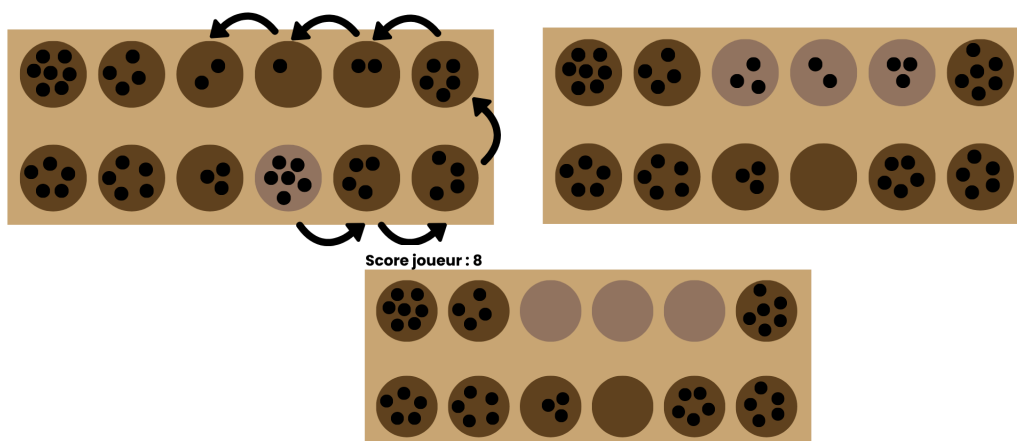


FIGURE 3 : PLATEAU DE JEU PENDANT LA PARTIE

5 - Règle supplémentaire

Un joueur n'a pas le droit de jouer un coup qui prend toutes les graines du camp de l'adversaire.

6 - Fin de partie

Le jeu se termine lorsque :

- Un joueur obtient 24 graines, il est donc déclaré vainqueur.
- Un joueur n'a plus de graines dans son camp et ne peut donc plus jouer. L'adversaire capture alors les graines restantes de son camp.
- La partie boucle indéfiniment, la limite est fixée à 16 coups rapportant 0 graines. Dans ces cas-là, les joueurs récupèrent les graines dans leur camp.
- Si aucun coup valide n'est possible pour les deux joueurs, les deux joueurs sont donc dans l'incapacité de jouer

3 - GESTION DU PROJET

La gestion du projet a été particulièrement compliquée, puisque chaque membre du groupe avait des contraintes en dehors de la vie universitaire. De plus, nous n'avions pas tous commencé le projet en même temps, ce qui a compliqué le lancement du projet.

Nous avons donc décidé de communiquer majoritairement par messages. Cependant, lorsque nous étions bloqués et que nous avions besoin d'explications plus claires sur le code de chacun, soit nous communiquions le lundi après-midi lors des heures de cours dédiées à ce projet, soit nous nous appelions directement, et c'est grâce à cette disponibilité quasi constante que nous avons pu avancer et fournir ce projet. Nous avons également souhaité utiliser GitHub afin de travailler plus efficacement sur le projet. Cependant, nous avons rencontré plusieurs problèmes lors de son utilisation. Nous avons donc décidé d'utiliser un document Google Docs. Même si ce n'était pas idéal pour un travail collaboratif en programmation, nous avons tout de même choisi cette solution.

Nous avons décidé de nous répartir le travail sans véritable cahier des charges, ce qui n'était pas une bonne initiative quand nous prenons du recul sur le projet. Nous n'avions ni cahier des charges précis ni échéancier nous permettant d'avancer de manière efficace. Cela s'est révélé très problématique, puisque nous avons rencontré à de nombreuses reprises des problèmes de gestion du temps tout au long du projet. En effet, à cause de cet oubli de notre part, l'avancement du projet était peu structuré.

Dans un premier temps, nous avons défini ce que nous voulions avoir à la fin du projet. Nous nous sommes mis d'accord sur le fait qu'il fallait une classe pour le jeu "Awalé" et plusieurs classes avec les différentes stratégies implémentées. Nous avons donc écrit les lignes de code pour le jeu "Awalé" en implémentant l'ensemble des règles. Suite à cela, nous avons décidé d'avancer en fonction de ce qui était faisable avec les notions vues en cours et le fonctionnement du jeu "Awalé". C'est de cette manière que les stratégies « Greedy Best First » et « Random Player » ont vu le jour. Ce n'est qu'ensuite que nous avons implémenté les fonctions heuristiques. La réflexion qui a précédé la conception des différentes stratégies était complexe à cause de la nature du jeu "Awalé", de la complexité des règles et du fonctionnement du jeu. Même les stratégies qui paraissent simples devenaient beaucoup trop compliquées à programmer avec notre niveau actuel.

4 - LE JEU "AWALE"

- Structure des Classes

Pour commencer notre projet, nous avons eu l'idée de prendre exemple sur notre projet du semestre 3 : la création d'un jeu de bataille navale. Nous sommes donc partis sur 3 classes principales :

- **La Classe Awale** : C'est le cœur du programme. Elle gère l'initialisation du plateau, la vérification de la validité des coups et la logique de distribution des graines. Elle contient également la fonction de simulation, permettant aux algorithmes (Greedy, Monte Carlo et Fonction Heuristique) d'évaluer un coup potentiel sans modifier l'état réel de la partie.
- **La Classe Joueur** : Elle définit l'entité qui interagit avec le plateau. Chaque instance stocke le nom, le score actuel et, surtout, la stratégie associée (humain ou type d'IA).
- **La Classe Main** : Elle orchestre le déroulement de la partie. Elle initialise les objets, gère l'alternance des tours et maintient la boucle de jeu jusqu'à ce qu'une condition de fin de partie soit rencontrée.

Ensuite nous avons créé l'interface **Stratégie**, elle permet de créer toutes les stratégies sur la même base.

- Défis Techniques et Résolution de Problèmes

Au cours du développement, nous avons été confrontés à plusieurs problématiques majeures qui ont nécessité une refonte de notre approche :

Tout d'abord la gestion du plateau lors de la simulation. L'un des obstacles principaux a été la fonction de simulation des coups. Initialement, nos tests de simulation modifiaient directement l'objet plateau original. Cela corrompait l'état réel du jeu, rendant les décisions de l'IA incohérentes. Nous avons dû corriger cela en nous assurant que chaque simulation travaille sur une copie indépendante du plateau.

Ensuite l'implémentation des règles complexes. La gestion de la fin de partie a révélé des cas particuliers que nous n'avions pas anticipés, notamment la situation de "famine" (lorsqu'un joueur ne peut plus jouer aucun coup légal pour nourrir son adversaire). L'intégration de ces conditions d'arrêt a été essentielle pour garantir la conformité du jeu aux règles officielles.

Et pour finir, l'optimisation. La première version du code reposait sur des structures de contrôle rudimentaires (while True, break). Bien que fonctionnelle, cette approche rendait la maintenance et l'intégration des algorithmes d'IA complexe. Nous avons entrepris un travail de réorganisation pour rendre le code plus fonctionnel, améliorant ainsi la lisibilité globale du projet malgré les contraintes de temps.

5- FONCTION HEURISTIQUE

Les fonctions heuristiques ont pour but de guider l'IA vers les coups les plus prometteurs afin d'augmenter ses chances de gagner. Dans le cas du jeu Awalé, leur conception s'est révélée compliquée, notamment parce que tous les membres du groupe n'étaient pas familiers avec ce jeu au départ. Même si ses règles sont simples en apparence, elles rendent l'évaluation des meilleurs coups assez complexe. Malgré ces difficultés, nous avons réussi à concevoir deux fonctions heuristiques.

La première heuristique repose principalement sur les gains possibles et les pertes potentielles associées à un coup. L'un des principaux problèmes a été le choix des poids à attribuer à ces paramètres. Il est difficile de savoir s'il faut privilégier les gains immédiats ou limiter les pertes, car certaines situations nécessitent de prendre des risques pour obtenir de meilleurs résultats plus tard. En effet, même si les deux facteurs sont importants pour le choix du coup, nous ne savons pas si les deux se valent ou si l'un est plus important que l'autre et si c'est le cas, lequel. Idéalement, on souhaiterait gagner plus que l'on ne perd, mais il arrive que pour obtenir de gros gains, il faille accepter de prendre des risques importants. Pour simplifier, nous avons testé plusieurs pondérations différentes et retenu la meilleure valeur obtenue afin de choisir le coup le plus intéressant.

La mise en place de cette heuristique nous a demandé de simuler chaque coup valide sans modifier le plateau initial, ce qui a nécessité la création d'une méthode spécifique pour récupérer les informations nécessaires. Cette partie du projet s'est révélée assez complexe, car il s'agissait d'un type de raisonnement algorithmique que nous n'avions encore jamais abordé en licence. Nous avons dû également créer une nouvelle méthode appelée « InfosCoups », permettant de récupérer les informations nécessaires au calcul des fonctions heuristiques.

La seconde fonction heuristique, développée avec l'aide de M. Pellier, prend en compte le nombre de coups nécessaires pour obtenir des gains. Nous avons choisi de privilégier les gains rapides, car cette heuristique ne tient pas compte des coups de l'adversaire. Chercher des gains trop lointains rendrait la stratégie trop fragile face aux interventions adverses. Si nous nous focalisons sur de gros gains nécessitant de nombreux coups, l'intervention de l'adversaire pourrait facilement perturber notre stratégie et fausser les résultats. Elle doit donc être vue davantage comme une suggestion car le fait de ne pas prendre en compte les actions de l'adversaire constitue une limite importante.

Notre approche possède plusieurs limites. Tout d'abord, nos fonctions heuristiques ne prennent pas en compte les coups futurs de l'adversaire. Elles se basent uniquement sur l'état actuel du plateau, ce qui peut mener à des choix qui semblent bons sur le moment

mais qui deviennent mauvais après l'intervention de l'adversaire. De plus, nos heuristiques ont surtout une vision à court terme. Même si elles permettent de calculer les meilleurs coups immédiatement, il est très difficile d'anticiper les conséquences sur le long terme, car cela nécessiterait de simuler plusieurs coups à l'avance. Un autre problème concerne le choix des poids des différents paramètres. Ces poids ont été choisis de manière assez arbitraire, car nous n'avions pas de méthode précise pour savoir lequel des critères était le plus important. Selon les situations, un mauvais choix de poids peut donc influencer négativement le coup sélectionné. Aussi, les deux fonctions heuristiques ont été utilisées séparément. Les combiner aurait probablement permis d'obtenir de meilleurs résultats, mais nous avons pensé à cette amélioration trop tard dans le projet. Enfin, nos fonctions heuristiques sont fixes et ne s'adaptent pas au déroulement de la partie ni au style de jeu de l'adversaire.

6 - GREEDY BEST FIRST

Pour ce projet, nous avons mis en place la stratégie Greedy Best-First pour notre IA. L'idée est simple : l'IA choisit le coup qui permet de capturer le plus de graines immédiatement, sans se soucier des coups précédents ni des conséquences futures.

Dans notre programme, la méthode "choisirCoup" parcourt tous les trous du joueur et simule chaque coup possible grâce à la fonction "simulerCoup". La simulation reproduit le déplacement des graines sur le plateau et calcule combien de graines seraient capturées. L'IA sélectionne ensuite le coup le plus avantageux. Si aucun coup ne rapporte de graines, elle prend simplement le premier coup valide, ce qui garantit qu'elle joue toujours légalement.

Le code gère également la capture des graines de l'adversaire : après la simulation du déplacement, si le dernier trou atteint contient 2 ou 3 graines, ces graines sont capturées et retirées du plateau. Cette étape est répétée en remontant le plateau tant que les conditions de capture sont remplies, ce qui reproduit fidèlement les règles d'Awalé.

Cette stratégie est simple à mettre en œuvre et efficace sur le court terme, car elle maximise instantanément le gain de chaque coup. Cependant, elle ne prend pas en compte les coups futurs, ce qui limite sa performance sur des parties plus longues ou contre des IA capables de planifier plusieurs tours à l'avance.

Malgré ces limites, nos tests montrent que Greedy Best-First est plutôt robuste. Elle peut même battre certaines IA basées sur des heuristiques, ce qui prouve qu'une approche simple peut être très efficace si elle est bien adaptée au jeu. C'est donc une base solide pour tester et comparer des algorithmes plus complexes.

7 - MONTE CARLO TREE SEARCH (MCTS)

Après avoir essayé plusieurs fonctions heuristiques, nous avons voulu utiliser une méthode plus générale pour mieux anticiper les conséquences des coups sur le long terme. Nous avons donc choisi d'implémenter une intelligence artificielle basée sur l'algorithme Monte Carlo Tree Search (MCTS). Contrairement aux heuristiques, qui se basent uniquement sur l'état actuel du plateau, MCTS simule un grand nombre de parties complètes afin d'évaluer la qualité d'un coup. Cette approche est particulièrement intéressante pour l'Awalé, car elle permet de dépasser une analyse trop limitée au court terme.

Pour chaque coup valide possible, l'algorithme réalise un nombre fixe de simulations (1000 par défaut). Lors de chaque simulation, le coup testé est joué, puis la partie continue avec des coups choisis au hasard pour les deux joueurs. La simulation s'arrête lorsqu'il n'y a plus de coups possibles ou après un nombre limité de tours. Les graines capturées sont comptabilisées afin de déterminer le vainqueur de la simulation.

Pour utiliser MCTS avec notre jeu, nous avons créé une classe appelée *AwaleState*, qui représente un état indépendant du plateau. Cette classe gère les règles principales de l'Awalé, comme les coups valides, le semis, les captures et la fin de partie. Il était très important que les simulations ne modifient pas la partie réelle. Pour cela, chaque coup simulé crée un nouvel état, ce qui rend l'implémentation plus sûre, mais aussi plus complexe.

Pour ce faire, nous appelons la méthode `simulerPartieAleatoire` qui simule une partie d'Awalé à partir d'un coup donné. Le plateau est d'abord copié, puis le coup initial est joué. La partie continue ensuite avec des coups choisis aléatoirement pour les deux joueurs, jusqu'à ce qu'il n'y ait plus de coups possibles ou qu'un nombre maximal de tours soit atteint. Les graines capturées sont comptées, et la simulation est considérée comme gagnée si le score du joueur est supérieur à celui de l'adversaire.

Une autre difficulté était de respecter exactement les règles du jeu pendant les simulations. Même si l'Awalé paraît simple, certaines règles sont assez subtiles, notamment celles liées à la famine. La moindre différence entre les règles du jeu réel et celles utilisées dans MCTS pouvait provoquer des incohérences.

Le caractère aléatoire de MCTS rend aussi le débogage plus compliqué. Deux exécutions peuvent donner des résultats différents à partir du même état, ce qui peut sembler étrange au début, mais correspond au fonctionnement normal de l'algorithme.

Même si MCTS est théoriquement plus efficace que les heuristiques, il présente certaines limites. Les simulations étant aléatoires, des coups peu intéressants sont testés, surtout en début de partie. Pour améliorer les résultats, il faut souvent augmenter le nombre

d'itérations, ce qui augmente le temps de calcul. De plus, ce nombre est fixé à l'avance et ne s'adapte pas à la situation de jeu. Enfin, l'adversaire étant simulé de manière aléatoire, l'IA peut être moins performante face à un joueur humain.

Hormis le problème du nombre d'itérations qui nous limite, plusieurs améliorations sont possibles, comme guider les simulations avec des heuristiques simples, adapter le nombre d'itérations selon la situation, ou mieux combiner MCTS avec les heuristiques déjà développées.

8 - L'INTERFACE

Pour ce projet, nous avons choisi de développer une interface classique, afin de pouvoir nous concentrer sur la conception et l'implémentation des algorithmes plutôt que sur l'aspect graphique. L'interface permet de tester facilement les stratégies développées et de visualiser les résultats.

Elle offre deux modes d'utilisation :

- Simulation automatique de 50 parties

Dans ce mode, l'utilisateur peut sélectionner les stratégies des deux joueurs, puis lancer une série de 50 parties. L'interface affiche ensuite le nombre de victoires de chaque stratégie, ce qui nous a permis de mettre nos algorithmes en situation réelle et de comparer leurs performances sur un échantillon prédéfini.

- Partie unique

Ce mode permet de jouer une seule partie, soit entre deux IA, soit entre un humain et une IA.

Lorsque c'est un humain qui joue, l'interface permet de choisir les coups et suivre l'évolution du plateau. Si la partie se joue uniquement entre IA, elle affiche directement le plateau final à l'issue de la partie. Ce mode est utile pour observer le déroulement concret d'une partie et mieux comprendre les choix stratégiques de l'IA.

Cette interface, bien que simple, a été suffisante pour tester nos stratégies, analyser leurs résultats et observer leur comportement en situation réelle, sans alourdir le projet avec des éléments graphiques complexes.

9 - LES INTERACTIONS ENTRE LES DIFFÉRENTES STRATÉGIES

Pour tester l'utilité et la pertinence de nos algorithmes, nous avons organisé des tournois de 50 parties entre différentes IA. L'objectif était de comparer les performances de nos stratégies et de vérifier leur cohérence dans différents scénarios.

Dans un premier temps, nous avons fait s'affronter nos stratégies entre elles et voici les résultats avec quelques explications :

- "Greedy Best First" vs "Greedy Best First"

Le premier joueur gagne systématiquement. Ce qui peut être expliqué par le fait que, pour cet algorithme, l'état actuel du plateau a une conséquence directe sur le coup choisi. Le premier joueur possède donc un léger avantage dès le début, ce qui explique ce résultat.

- "Random" vs "Random"

Même si dans les résultats obtenus, nous n'avons pas d'égalité parfaite, ils se rapprochent énormément de 25/25. Ce qui est logique car les coups sont choisis aléatoirement. La partie est donc quasi équitable sur le long terme, mais il y a des petites variances qui peuvent créer des différences, ce qui est logique.

- "Heuristique" vs "Heuristique"

Le deuxième joueur gagne sur toutes les parties. Cela peut indiquer que les paramètres que nous avons choisis, favorise certaines positions ou configurations spécifiques et qu'être le deuxième joueur permet d'obtenir le chemin le plus prometteur fiable.

- "HeuristiqueProf" vs "HeuristiqueProf"

Comme pour "Greedy Best First", c'est le premier joueur qui gagne systématiquement. Cette IA applique une logique similaire, basée sur l'évaluation de la "distance restante" à l'objectif, ce qui renforce l'avantage du joueur initial.

- "MCTS" vs "MCTS"

Les résultats sont équilibrés. Contrairement aux autres stratégies, la position du joueur n'a pas d'influence sur le résultat du tournoi.

D'après les résultats obtenus dans la première partie, nous pouvons dire que la meilleure stratégie est "MCTS", puisqu'elle n'est pas influencée par l'ordre de jeu et semble capable d'obtenir les meilleurs résultats dans toutes les situations.

Maintenant, regardons les résultats lorsque les différentes stratégies s'affrontent les unes contre les autres.

- "HeuristiqueProf" vs "Heuristique"

La stratégie fournie par M. Pellier remporte la majorité des parties. Cela peut s'expliquer par le fait que cette fonction heuristique prend en compte non seulement le coup immédiat le plus prometteur, mais aussi les futurs coups, alors que notre fonction heuristique se concentre uniquement sur le

meilleur coup du moment.

- “MCTS”

Cette IA gagne contre toutes les autres IA. Ce qui est logique car la stratégie “MCTS” utilise une approche plus avancée et intelligente que toutes les autres stratégies

- “Greedy Best First”

Cette IA gagne contre presque toutes les autres stratégies, ce qui montre que sa logique de choix basée sur la “distance restante” à l’objectif est efficace sauf face à “MCTS”

- “Random”

Comme attendu, elle perd contre toutes les autres IA. Sans stratégie, les coups sont purement aléatoires et ne maximisent pas les chances de gagner.

En analysant nos résultats, nous constatons que les stratégies “Greedy Best First” et “HeuristiqueProf” obtiennent des scores similaires lors du tournoi. Cela s’explique par leur logique de conception similaire : les deux se concentrent sur l’évaluation de la “distance restante” à l’objectif depuis le nœud actuel, ce qui guide leurs choix vers les coups les plus prometteurs. Cependant, lorsque nous incluons la stratégie MCTS (Monte Carlo Tree Search) dans le tournoi, nous observons qu’elle surpasse systématiquement toutes les autres IA. Contrairement aux autres stratégies, MCTS anticipe plusieurs coups à l’avance et évalue leurs conséquences grâce à des simulations, ce qui lui permet de prendre des décisions globalement optimales sur l’ensemble de la partie.

Nous pouvons donc conclure que, dans le cas du jeu Awalé, les stratégies qui prennent en compte non seulement le meilleur coup du moment mais aussi les coups futurs sont les plus performantes. Cela confirme qu’une IA capable de planifier sur plusieurs tours, comme MCTS, est la plus adaptée pour gagner une partie entière et non seulement un coup ponctuel.

10 - CONCLUSION

Malgré les obstacles rencontrés, nous avons su nous adapter et mener à bien notre projet autour du jeu Awalé, en développant des stratégies adaptées à nos compétences. Ce choix a présenté des limites, notamment une interface simple et des fonctions heuristiques perfectibles, mais il nous a aussi permis de relever des défis techniques et organisationnels.

Ce projet nous a permis de renforcer nos compétences en algorithmique et programmation (simulations indépendantes, gestion d'objets complexes, implémentation de stratégies variées), en analyse stratégique (tournois et évaluation des IA), et en travail en équipe et gestion de projet. Ces acquis seront précieux pour notre parcours académique et professionnel, et ouvrent la voie à de futures améliorations, tant sur le plan technique que stratégique.