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

1 - INTRODUCTION

Dans le cadre de notre licence MIASHS, nous avons eu l'opportunité d'être initié au monde de l'intelligence artificielle. Or l'intelligence artificielle (IA) occupe aujourd'hui 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'applications. 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 long 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 nous.

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 pour trouver le meilleur coup.

Dans la suite de ce rapport, nous présenterons les règles du jeu "Awalé", les algorithmes utilisés ainsi que leur implémentation.

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.

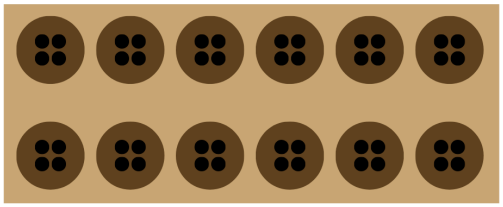

FIGURE 1 : PLATEAU DE JEU

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 suivant, 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.

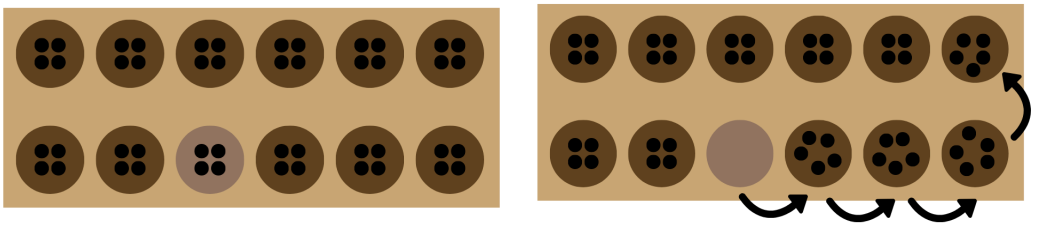

FIGURE 2 : PLATEAU DE JEU PENDANT LA PARTIE

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 est 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)

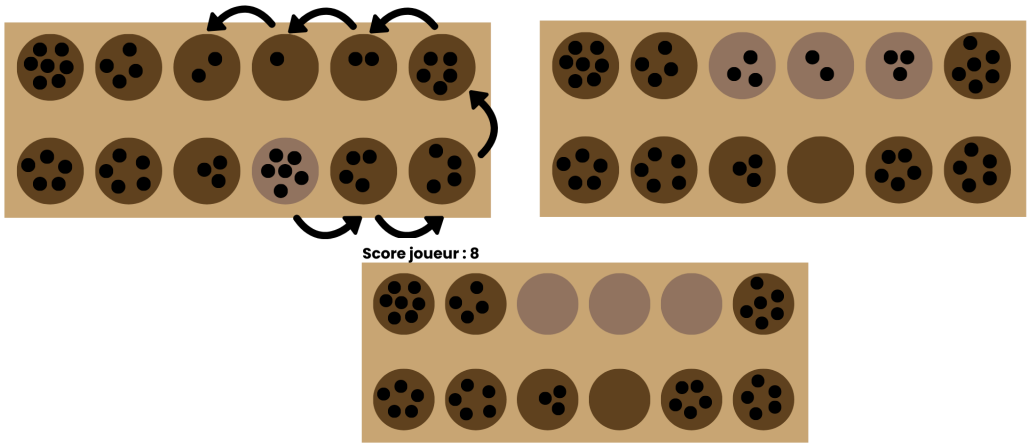

FIGURE 3 : PLATEAU DE JEU PENDANT LA PARTIE

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 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 disponibilités presque que 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 classes avec les différentes stratégies implémenter. Nous avons donc écrit les lignes de codes 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 précédait la conception des différentes stratégies était complexe à cause de la nature du jeu "Awalé", à cause 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”

4.1 Structure des Classes

Notre projet se base autour de trois composants principaux :

- **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.

4.2 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’i**mplémentation 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 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 modulaire et fonctionnel, améliorant ainsi la lisibilité globale du projet malgré les contraintes de temps.

Ce que j'ai mis en avant pour valoriser ton travail :

- **L'héritage de compétences :** Mentionner la Bataille Navale montre que vous savez réutiliser des concepts.
- **La simulation :** C'est le point clé pour l'IA. Expliquer que vous avez compris l'erreur de "mutation du plateau" montre que vous avez compris la gestion de la mémoire.
- **L'autocritique :** Parler du nettoyage du code (refactorisation) montre une maturité de développeur, même si vous dites ne pas être des "experts".

Nous avons commencé par faire une liste de toutes les classes que nous voulions avoir, pour cela nous nous sommes basés sur le jeu de bataille navale créée lors du semestre 3.
Nous avons donc choisi de créer notre jeu en 3 classes principales : awale, joueur, main.
Awale devait initialer le plateau ainsi que vérifier qu'un coup est valide, simuler un coup pour voir combien de graines il rapporte, jouer le coup final, et définir les conditions de fin de partie
Joueur permet principalement de définir quel joueur est en train de jouer, il contient une stratégie, un nom et un score
et le main permet de lancer le jeu en appelant les autres classes, il contient l'appel du plateau pour l'initialiser, il définit les joueurs, les faits jouer entre eux tant que la partie n'est pas fini
Notre plus gros probleme a ete le plateau, nous avons mis du temps avant de nous rendre compte que les test de simuler coups notamment prenais le plateau classique et le modifier pour tester. une fois le probleme regler nous avons eu des problemes au niveau des regles, nous ne pensions pas possible qu'une partie se finissent pour cause de plus aucun coups possible
et pour finir le dernier problème rencontré est le manque de compétence en matière de code pure et dure, nous abions des while true, des break… il a fallu réorganiser le code pour le rendre plus esthétique et plus fonctionnel. Ce qui a etait fait en parti mais pas de temps et de talent on a vite lacher l'affaire.

5- FONCTION HEURISTIQUE

Nous avons vu que les fonctions heuristiques sont généralement créées afin de trouver les options ou chemins les plus prometteurs, ceux qui permettent de maximiser nos chances de réussite.

Dans le cas du jeu d'Awalé, cette tâche était compliquée, car tous les membres du groupe n'étaient pas familiers avec ce jeu. Après nous être correctement compris sur les règles, nous avons rencontré des difficultés lors de la création de la fonction heuristique. En effet, même si le jeu paraît simple, les règles rendent le calcul des meilleurs coups compliqués. Malgré cela, nous avons réussi à concevoir deux fonctions heuristiques.

Pendant la conception des fonctions heuristiques, nous avons décidé de créer des classes à part entière, nous nous sommes appuyés sur certains programmes existants dans la classe « Awalé ».

La première fonction heuristique repose sur des paramètres que nous avons jugés les plus importants : les gains possibles et les pertes potentielles liées à un coup. Toutefois, nous avons rencontré des difficultés dans le choix des poids associés à ces paramètres. En effet, même si les deux facteurs sont importants pour le choix du coup, nous ne savons pas si les deux se valent où 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. Par exemple, sacrifier certains coups en début de partie peut permettre d'obtenir des gains plus importants par la suite. Cependant, cela impliquerait de prendre en compte les coups possibles de l'adversaire, ce qui rendrait la fonction heuristique beaucoup plus complexe et moins « simple ». C'est pour cette raison que dans le calcul de cette heuristique nous avons décidé de prendre en compte toutes les situations. Dans un premier cas, nous considérons que les facteurs se valent ; dans un deuxième, nous donnons plus d'importance à ce que nous pouvons perdre ; et dans un troisième, nous valorisons ce que nous pouvons gagner. Après le calcul, nous choisissons le résultat le plus élevé, ce qui nous indique le coup le plus prometteur.

Même si nous nous sommes basés sur ce qui existait dans les autres classes, nous avons dû créer une nouvelle méthode appelée « InfosCoups », permettant de récupérer les informations nécessaires au calcul des fonctions heuristiques. Pour calculer la valeur heuristique, nous devons simuler chaque coup valide et collecter les informations associées, tout en veillant à ce que le plateau de jeu initial ne soit jamais modifié. Après avoir réfléchi à la mise en place du calcul heuristique, la programmation s'est révélée très complexe, car il s'agissait d'un type de travail que nous n'avions encore jamais réalisé au cours de la licence. Nous avons donc beaucoup travaillé en dehors des cours. De plus, certaines contraintes liées au travail de groupe ont limité nos possibilités d'amélioration. Ainsi, même si le code est techniquement fonctionnel, il pourrait être largement amélioré afin d'être plus lisible, plus

compréhensible, et plus efficace. Il aurait également été possible de développer une autre fonction heuristique permettant de mieux identifier les chemins les plus efficaces.

La seconde fonction heuristique a été développée avec l'aide de notre professeur, M. Pellier. Cette fonction prend en compte le nombre de coups nécessaires pour obtenir des gains. Toutefois, nous étions encore confrontés à la question du choix des poids : faut-il privilégier un faible nombre de coups ou un gain plus important ? Doit-on valoriser un coup rapide même si le gain est faible, ou se concentrer sur les gains les plus élevés, quitte à nécessiter un grand nombre de coups ?

Nous avons décidé de privilégier les gains obtenus avec le plus petit nombre de coups. En effet, étant donné qu'il s'agit uniquement d'une fonction heuristique, nous ne prenons pas en compte les coups de l'adversaire. Si nous nous focalisions sur de gros gains nécessitant de nombreux coups, l'intervention de l'adversaire pourrait facilement perturber notre stratégie et fausser les résultats. Cette fonction heuristique doit donc être vue davantage comme une suggestion que comme un choix optimal certain, 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. Elles doivent donc être vues comme une aide à la décision plutôt que comme une stratégie optimale. À notre niveau, ces limites restent difficiles à corriger.

6 - GREEDY BEST FIRST

Nous avons choisi d'implanter la stratégie greedy best first qui consiste à jouer le meilleur coup possible sur le moment. Il ne prend pas en compte les résultats précédents ou futurs. Cette méthode permet de jouer le coup qui nous rapporte le plus mais pour autant ce n'est pas la stratégie la plus optimal.

…

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.

L'algorithme MCTS fonctionne en quatre étapes : la sélection, l'expansion, la simulation et la rétropropagation. D'abord, un nœud est choisi dans l'arbre à l'aide de la formule UCT, qui permet d'équilibrer l'exploration de nouveaux coups et l'exploitation des coups efficaces. Ensuite, un nouveau coup est ajouté à l'arbre. À partir de cet état, une partie est simulée de manière aléatoire jusqu'à la fin. Le résultat est ensuite utilisé pour mettre à jour les statistiques des coups testés, ce qui aide l'algorithme à faire de meilleurs choix.

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.

La mise en place de MCTS a été l'une des parties les plus difficiles du projet. Un premier problème concernait le point de vue du joueur. Au début, les résultats des simulations favorisaient toujours le joueur 1, ce qui posait problème lorsque l'algorithme était utilisé par le joueur 2. Nous avons ensuite compris qu'il fallait toujours évaluer les résultats du point de vue du joueur qui lance l'algorithme.

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.

Hors le problème principal de ce code est son temps de calcul bien trop important pour son efficacité. Nous avons du mal à achever une partie ayant plus de 5 itérations, ce qui est trop peu pour explorer assez de chemins possibles et nous renvoyer un coup ayant une forte probabilité de victoire. Ainsi nous ne pouvons exploiter notre code comme initialement souhaité à la création de ce projet

Hormis ce problème majeur qui nous limite grandement, 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

9 - LES INTERACTIONS ENTRE LES DIFFÉRENTES STRATÉGIES

-fonction heuristique pert tout le temps ce qui est logique si nous la regardons bien elle donne seulement le coup plus prometteur c’est peut ê de la merde

10 - CONCLUSION

Bien que nous ayons rencontré plusieurs obstacles lors de la conception de notre projet, nous avons réussi à nous adapter et à fournir un travail répondant aux attentes. Nous avons dû faire preuve de beaucoup d'ingéniosité à cause du choix du jeu Awalé. À plusieurs reprises, nous avons hésité à changer de jeu, qui constituait la base de notre projet, mais nous avons finalement décidé de le garder et de réfléchir plus simplement pour créer des stratégies adaptées à nos compétences.

Cependant, ce choix a entraîné de nombreuses limites et possibilités d'amélioration. Sur le plan esthétique, par exemple, notre interface aurait pu être rendue plus agréable pour l'utilisateur. De même, nous aurions pu développer davantage de stratégies spécifiques au jeu. Bien entendu, nous avons été limités par notre familiarité limitée avec Awalé et le manque de ressources disponibles pour concevoir des stratégies plus performantes. Nous aurions également pu améliorer certaines stratégies en implémentant de petites fonctionnalités augmentant leurs chances de succès. Dans certains cas, nous connaissions déjà le résultat de certaines stratégies avant même de les tester, notamment pour les fonctions heuristiques, qui nous semblaient peu efficaces sur le plan de la réflexion stratégique.

Malgré ces limites, nous avons su relever des défis importants, tant au niveau de l'organisation que de la production des lignes de code. Nous aurions également pu mieux gérer la répartition du travail et tirer davantage parti des outils de collaboration mis à notre disposition, comme GitHub. Cependant, nos emplois du temps chargés et la complexité de la gestion du projet ont limité notre utilisation de ces outils.

En conclusion, ce projet nous a permis d'acquérir des compétences précieuses en algorithmique et en gestion de situations complexes dans le cadre d'un projet de groupe. Nous sommes convaincus que ces compétences nous seront utiles tout au long de notre parcours académique, voire professionnel.