Université de Caen Normandie

Licence 3 Informatique

Projet 2

Jeu de Hex et bandits manchots

IA

Réalisé par :

Joyce DIAB

Miguel Jordan Kamgang Kenmoe

Ayath ABOGOUNRIN

Lina BOUAMAR

Année Universitaire
2024/2025

Table des matières

1 Introduction

1.1 Description générale du projet

Ce rapport présente le travail qu'on a réalisé dans le cadre de notre Licence 3 en Informatique à l'Université de Caen. Il décrit l'implémentation du Jeu de Hex et bandits manchots, un jeu de stratégie combinatoire où deux joueurs s'affrontent sur un plateau hexagonal pour relier ses bords opposés. L'objectif de ce projet est de permettre à des joueurs humains ou robots de jouer avec une certaine stratégie en s'appuyant sur plusieurs concepts de programmation avancés, tels que les algorithmes de recherche de chemin, ainsi qu'une intelligence artificielle s'appuyant sur l'algorithme Monte Carlo Tree Search (MCTS) et son optimisation (RAVE).

En histoire, le jeu Hex est un jeu abstrait conçu par Piet Hein en 1942 et développé indépendamment par John Nash en 1948. La taille standard du plateau est de 11x11, mais elle peut être modifiée pour ajuster la complexité du jeu.

Chaque joueur est associé à une couleur :

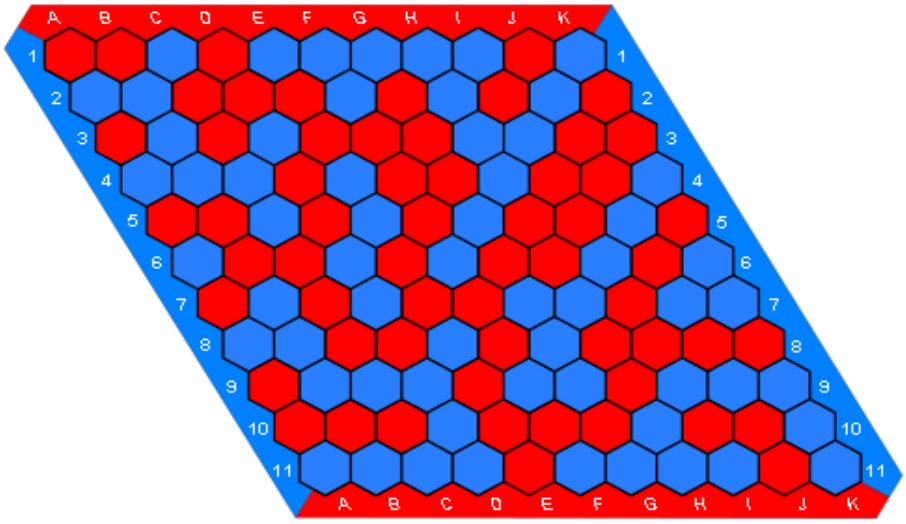
Rouge : son objectif est de relier le bord supérieur au bord inférieur du plateau.

Bleu : son objectif est de relier le bord gauche au bord droit.

Les joueurs jouent chacun leur tour en plaçant un pion de leur couleur sur une case vide du plateau. Une fois qu'un pion est placé, il ne peut plus être déplacé ni retiré.

Le jeu assure une victoire, ce qui signifie qu'il ne peut y avoir de match nul. En effet, une fois que toutes les cases sont occupées, au moins un des joueurs aura réussi à relier ses bords opposés. Le théorème de Nash démontre que le premier joueur a une stratégie gagnante, bien que cette stratégie optimale ne soit pas facile à identifier.

Hex est également reconnu pour son intérêt mathématique, notamment en théorie des jeux et en topologie, car il est directement lié au théorème du cheminement.



Exemple du plateau de jeu. Source : image

1.2 Présentation du plan du rapport

Le plan de ce rapport est comme suit :

— Objectifs du projet : d'abord présenter la problématique, ensuite les objectifs.

— Fonctionnalités implémentées : description détaillée des fonctionnalités réalisés et répartition des taches.

— Éléments techniques : explications détaillées des algorithmes qu'on a utilisés ainsi que les structures de données.

— Architecture du projet : organisation du code et interaction entre classes

— Expérimentations : on analyse l'impact des paramètres principaux (budget, taille, algorithmes utilisés) sur les résultats obtenus.

— Conclusion : bilan final sur ce qu'on a appris et les améliorations possibles.

2 Objectifs du Projet

2.1 Problématique

L'objectif principal de ce projet est d'étudier l'influence du rapport de budget entre les deux joueurs, sur leur probabilité de victoire en fonction de la taille de la grille, plus précisément, nous cherchons a répondre aux questions suivantes :

1. Est ce qu'il existe un seuil de budget ou un joueur a plus d'avantages que l'autre ?
2. Comment la taille du plateau affecte-t-elle cette relation entre budget et résultats ?

Le "budget" se réfère à une mesure de la puissance ou des ressources disponibles pour chaque joueur. Cela peut inclure des choix stratégiques comme la possibilité de jouer plus de coups, ou encore l'intelligence artificielle utilisée pour prendre des décisions.

2.2 Modélisation du jeu

On a représenté le plateau sous forme de graphe ou chaque case hexagonale représente un nœud et chaque connexion entre deux cases adjacentes représente une arête, ce qui nous a permis d'utiliser des algorithmes de recherche comme DFS pour déterminer si un joueur a pu relier ses bords opposés ou non(DFS explore chaque branche du graphe en profondeur, ce qui permet de vérifier si un joueur possède un chemin gagnant en examinant toutes les possibilités de connexion)

Ensuite la création d'une IA : (Monte Carlo Tree Search MCTS) : c'est un algorithme base sur des simulations aléatoires et une exploration stratégique des coups possibles. Il repose sur la construction d'un arbre de recherche, où chaque nœud représente un état du jeu. L'algorithme suit quatre étapes principales :

L'un des avantages de MCTS est qu'il ne nécessite pas de base de connaissances spécifique sur le jeu, ce qui en fait une méthode efficace pour l'optimisation des décisions.

Optimisation avec RAVE :
L'idée principale derrière RAVE est d'utiliser une approche d'estimation rapide pour améliorer l'évaluation des actions dès les premières simulations. Au lieu d'attendre qu'une action spécifique soit suffisamment explorée dans un sous-arbre donné, RAVE applique les informations collectées à travers plusieurs simulations. L'algorithme repose sur la notion de valeurs, d'action globales et combine deux sources d'information :

Q-value standard de MCTS : Basée uniquement sur les parties jouées dans une branche spécifique. Valeur RAVE : Basée sur toutes les simulations où une action donnée a été observée. L'équation d'évaluation des actions devient alors une combinaison de ces deux valeurs :

mise en place d'une interface graphique Une interface graphique a été réalisée pour permettre une visualisation facile du jeu, avec des interactions intuitives pour le joueur (clics, déplacements, etc.).

réalisation des expérimentations (tests) nous avons créé un script shell (script.sh). Ce script permet de lancer différentes configurations de parties, notamment les quatre types d'expérimentations suivants :

— RAVE vs RAVE : Deux intelligences artificielles utilisant RAVE s'affrontent.

— MCTS vs MCTS : Deux intelligences artificielles utilisant MCTS s'affrontent.

— MCTS vs RAVE : Une intelligence artificielle utilisant MCTS affronte une autre utilisant RAVE. Le joueur qui commence utilise MCTS.

— RAVE vs MCTS : Une intelligence artificielle utilisant RAVE affronte une autre utilisant MCTS. Le joueur qui commence utilise RAVE.

Chaque partie est lancée automatiquement avec ces configurations. Le script prend en charge la génération de ces parties et enregistre les résultats dans son dossier correspondant.

Chaque résultat contient les informations suivantes pour chaque expérience : nombre de parties, la taille de la grille, 1er joueur, gagnant, nb de victoires du gagnant , le budget du gagnant, l'algo du gagnant, nb de victoires du perdant, budget du perdant, algo du perdant

Le but de cette approche est de faciliter la collecte et l'analyse des résultats, ce qui nous permet ensuite d'évaluer l'impact des différents algorithmes (MCTS et RAVE) sur la performance des joueurs dans divers scénarios de jeu.

2.3 Travaux Similaires

Le jeu de Hex, bien que moins populaire que d'autres jeux comme les échecs ou le Go, partage avec eux des défis similaires en termes de complexité et de stratégies gagnantes. Mais les gens le préfèrent parfois car ses règles sont plus simples et conduisent à des résultats élégants. Il faut savoir que plusieurs thèses lui ont été consacrés, et donc utilisé même auprès des étudiants et élève comme un exercice de raisonnement.

MCTS a permis aux IA de mieux explorer les décisions en effectuant des simulations aléatoires. Cette approche a été améliorée, comme l'a démontré AlphaGo (pour le jeu de Go), développé par DeepMind en 2016. AlphaGo a utilisé une combinaison de MCTS, réseaux de neurones convolutifs et apprentissage supervisé pour surpasser les meilleurs joueurs humains.

En utilisant ces techniques, AlphaGo a pu explorer efficacement l'espace de recherche en se concentrant sur les positions les plus prometteuses, ce qui a révolutionné la manière dont les IA abordent les jeux de stratégie. Donc le jeu de Hex et AlphaGo ont des principes en communs : l'utilisation de MCTS pour explorer l'espace des coups possibles, dans Hex, il y a l'amélioration de la sélection des coups avec RAVE, qui joue un rôle similaire à celui des réseaux de politique dans AlphaGo.

Un autre exemple aussi, est le jeu Havannah, un jeu de société similaire au jeu de Hex, qui se joue entre 2 joueurs, mais au lieu de relier des bords, plutôt compléter une des trois structures ; un anneau, un pont ou une fourche. Par contre, le MCTS a Havannah est plus dur à optimiser en raison du nombre de chemins varié gagnants, donc plus complexe.

3 Fonctionnalités implémentées

3.1 Description des fonctionnalités

1. Mode player vs player :deux joueurs humains jouent contre chacun
2. Mode player vs robot : un joueur humain s'affronte contre l'IA développée avec MCTS et RAVE.
3. Mode robot vs robot : Deux IA s'affrontent entre elles. Cela permet de voir comment l'algorithme fonctionne sans intervention humaine.
4. Visualisation graphique : un plateau interactif permet de visualiser le jeu, avec des cases hexagonales et des pions affichés. Cela aide le joueur à suivre l'évolution de la partie en temps réel.

3.2 Répartition des Taches

Afin d'assurer une répartition efficace du travail, nous avons défini les responsabilités de chaque membre de l'équipe selon leurs compétences et préférences. Voici un tableau récapitulatif des tâches attribuées :

Tâche	**Responsable(s)**
Modélisation du jeu et implémentation du plateau (graphe, règles, conditions de victoire)	Miguel, Joyce
Développement de l'IA (MCTS, RAVE)	Miguel, Joyce, Ayath
Implémentation de l'interface utilisateur (CLI)	Ayath
Développement de l'interface graphique (GUI)	Miguel
Expérimentations et collecte des données	Joyce
Analyse des résultats et visualisation (Python, Jupyter Notebook)	Tous
Rédaction du rapport et documentation technique	Lina

TABLE 1 – Répartition des tâches entre les membres du groupe

4 Éléments techniques

4.1 Algorithmes

Nous avons utilisé plusieurs techniques pour gérer le déroulement des parties, détecter les victoires, et implémenter une intelligence artificielle.

1. Algorithme de recherche de chemin DFS :il permet de vérifier s'il existe un chemin continu de pions reliant deux bords opposés, il commence l'exploration à partir des cases du joueur situées sur son bord initial, puis visite récursivement toutes les cases adjacentes de la même couleur qui n'ont pas encore été explorées. Si on atteint le bord opposé en suivant un chemin ininterrompu, alors le joueur a gagné.

Algorithm 1 Depth First Search (DFS)

```
1: procedure DFS(node, visited)
2:     if node est nul ou node est dans visited then
3:         return
4:     end if
5:     Marquer node comme visité
6:     Traiter node
7:     for all neighbor dans Adj(node) do
8:         if neighbor n'est pas dans visited then
9:             DFS(neighbor, visited)
10:        end if
11:    end for
12: end procedure
```

2. L'algorithme Monte Carlo Tree Search - MCTS : permet de prendre des décisions intelligentes en effectuant de nombreuses simulations aléatoires des parties pour estimer les meilleurs coups possibles.

   (a) Sélection : L'algorithme navigue dans l'arbre en choisissant les coups les plus prometteurs selon une stratégie d'exploration.
   (b) Expansion : Un nouvel état de jeu est ajouté à l'arbre si l'algorithme rencontre une configuration inexplorée.
   (c) Simulation : Une partie est jouée de manière aléatoire à partir de cet état jusqu'à atteindre une victoire ou une défaite.
   (d) Rétropropagation : Le résultat de la simulation est enregistré dans l'arbre et influence les décisions futures.

Algorithm 2 Algorithme MCTS

```
1:  procedure MCTS(root)
2:      while temps restant do
3:          node ← SELECTION(root)
4:          result ← SIMULATION(node)
5:          BACKPROPAGATION(node, result)
6:      end while
7:      return MEILLEURENFANT(root)
8:  end procedure
9:  procedure SELECTION(node)
10:     while node est entièrement développé et non terminal do
11:         node ← MEILLEURENFANT(node)
12:     end while
13:     return node
14: end procedure
15: procedure SIMULATION(node)
16:     state ← état correspondant à node
17:     while state n'est pas terminal do
18:         state ← SELECTIONALEATOIRE(state)
19:     end while
20:     return VALEURFIN(state)
21: end procedure
22: procedure BACKPROPAGATION(node, result)
23:     while node ≠ NULL do
24:         node.visits ← node.visits + 1
25:         node.reward ← node.reward + result
26:         node ← node.parent
27:     end while
28: end procedure
```

3. L'algorithme Rapid Action Value Estimation - RAVE : permet d'utiliser les informations des simulations plus efficacement en partageant les données entre plusieurs branches de l'arbre.

Algorithm 3 Algorithme MCTS-RAVE

```
1: procedure MCTS-RAVE(root)
2:     while temps restant do
3:         node ← SELECTION(root)
4:         (result, actionsSimules) ← SIMULATION(node)
5:         BACKPROPAGATION(node, result, actionsSimulées)
6:     end while
7:     return MEILLEURENFANT(root)
8: end procedure
9: procedure SELECTION(node)
10:    while node est entièrement développé et non terminal do
11:        node ← MEILLEURENFANTRAVE(node)
12:    end while
13:    return node
14: end procedure
15: procedure SIMULATION(node)
16:    state ← état correspondant à node
17:    actionsSimules ← liste vide
18:    while state n'est pas terminal do
19:        action ← SELECTIONALEATOIRE(state)
20:        Ajouter action à actionsSimules
21:        state ← APPLIQUERACTION(state, action)
22:    end while
23:    return (VALEURFIN(state), actionsSimules)
24: end procedure
25: procedure BACKPROPAGATION(node, result, actionsSimulées)
26:    while node ≠ NULL do
27:        node.visits ← node.visits + 1
28:        node.reward ← node.reward + result
29:        for all action ∈ actionsSimules do
30:            node.AMAF[action].visits ← node.AMAF[action].visits + 1
31:            node.AMAF[action].reward ← node.AMAF[action].reward + result
32:        end for
33:        node ← node.parent
34:    end while
35: end procedure
```

RAVE est une amélioration de MCTS visant à accélérer la convergence vers les meilleures décisions en exploitant des informations globales sur les actions jouées. Contrairement à MCTS classique, qui met à jour les valeurs des actions uniquement dans les branches spécifiques explorées, RAVE généralise ces informations pour influencer plus largement la prise de décision.

4.2 Structures de Données

Pour organiser et manipuler les informations du jeu, nous avons utilisé plusieurs structures de données :

1. Un tableau 2D – le plateau du jeu est un graphe représenté sous forme de tableau 2D (Grid[][]) où chaque case contient soit un pion du joueur 1, un pion du joueur 2, soit une case vide, ce qui permet un accès rapide aux cases adjacentes(crucial pour DFS).
2. Les dictionnaires(MAP)
3. Les listes : utilisées pour stocker les coups possibles à chaque état du jeu, et également pour stocker les voisins accessibles lorsqu'on parcours le graphe, ainsi pour garder les traces des coups joués dans une partie.

5 Architecture du projet

5.1 les packages utilisés

Nous avons organisé le projet en plusieurs packages pour séparer les différentes fonctionnalités :

- hex.analysis : (Game)
- hex.model : Gestion du plateau, jeu et players.
- hex.mcts : implémentation de l'algo MCTS.
- hex.rave : implémentation de l'algo RAVE.
- hex.view.cli : Interface console.
- hex.view.gui : Interface graphique(Swing).
- hex.util : Contient les classes utilitaires pour la gestion des événements et des interactions entre les différentes composantes du jeu, notamment via un système d'écouteurs les listeners

5.2 Diagramme des Classes

1. Le package model est responsable de la gestion de la logique et des règles du jeu.

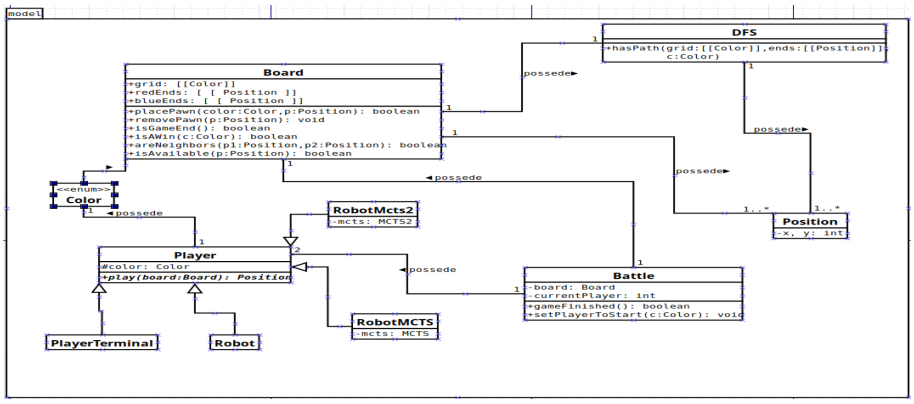


FIGURE 1 – Diagramme de classes du package model

2. Le package mcts est dédié à l'implémentation de l'algorithme Monte Carlo Tree Search(MCTS).Ce package regroupe plusieurs classes essentielles qui permettent de réaliser des simulations et de gérer l'exploration de l'arbre de recherche.

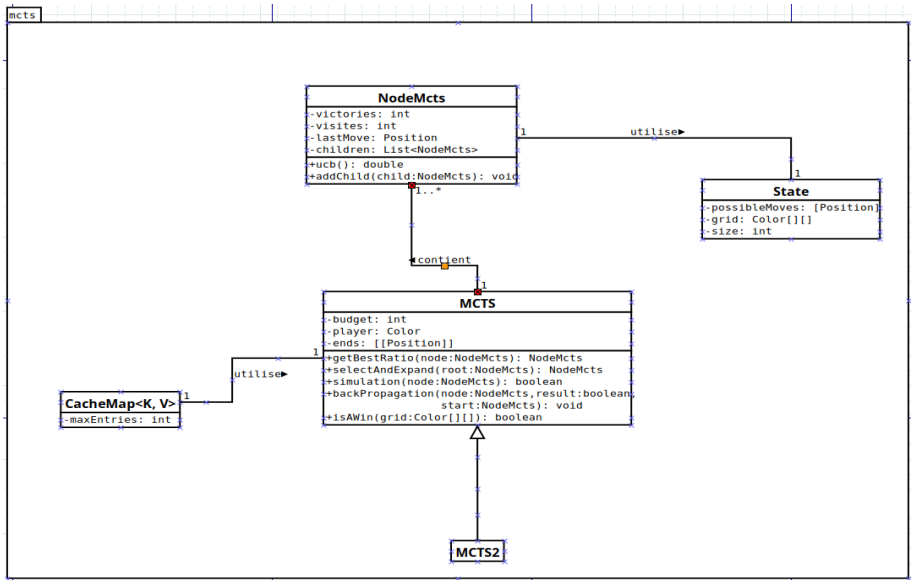


FIGURE 2 – Diagramme de classes du package mcts.

3. Le package rave est dédié à l'implémentation de l'approche Rapid Action Value Estimation(RAVE).

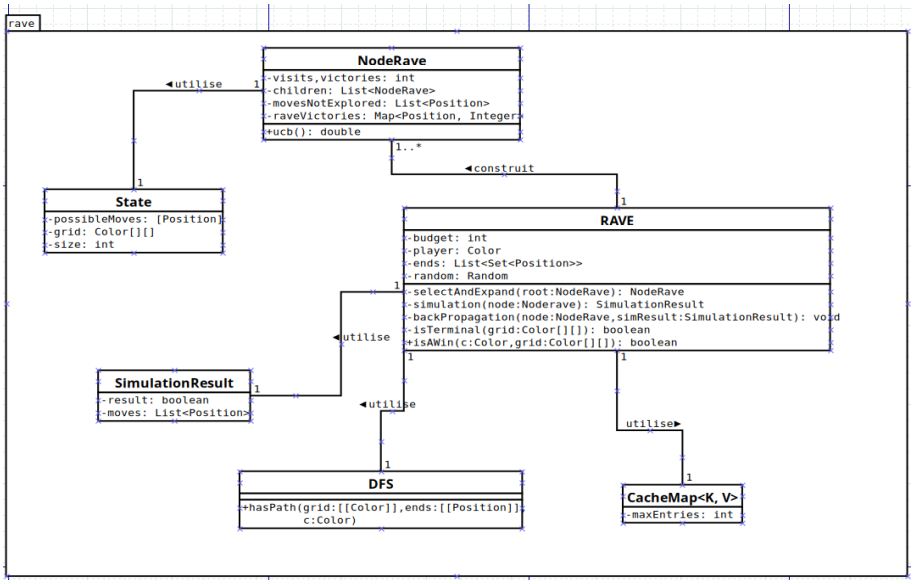


FIGURE 3 – Diagramme de classes du package rave

4. Le package view représente l'interface graphique et console.

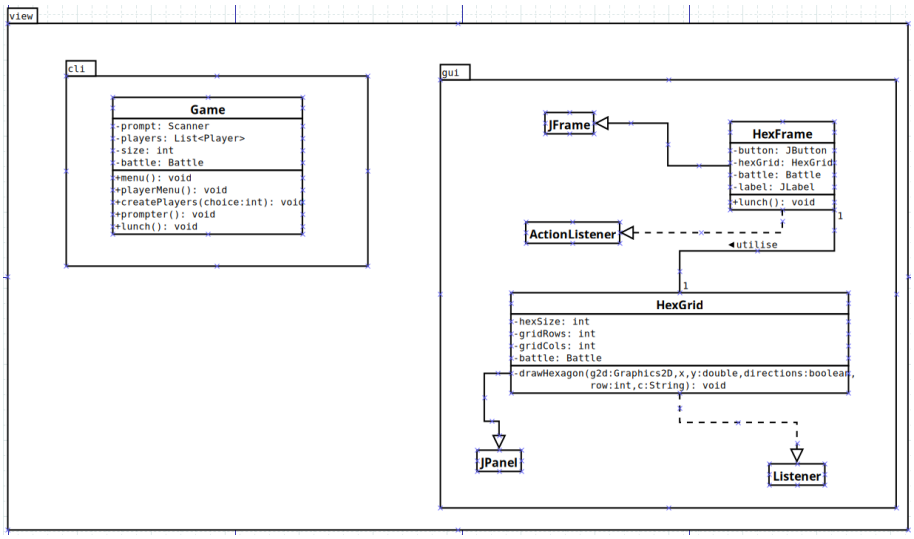


FIGURE 4 – Diagramme de classes du package view

5.3 Chaines de Traitement

1. On a d'abord fait la représentation du plateau sous forme de graphe ou chaque case hexagonale devient un nœud, et chaque connexion entre cases adjacentes est une arête. Cette représentation permet d'appliquer des algorithmes de recherche tels que DFS pour vérifier si un joueur a réussi à relier ses bords opposés. Ensuite, la modélisation des joueurs, les deux joueurs (rouge et bleu) ont des objectifs spécifiques (relier des bords opposés), et chaque joueur choisit une case vide sur le plateau pour y placer un pion à chaque tour.

2. Algorithmes de Recherche de Chemin DFS

3. Implémentation de l'intelligence artificielle : d'abord MCTS puis son optimisation RAVE.

4. L'interface en ligne de commande qui permet d'interagir avec le jeu (simple et textuelle).

5. L'interface graphique avec des cases hexagonales pour visualiser le plateau du jeu.

6. Expérimentation et collecte des données (en lançant plusieurs parties avec différentes configurations).

7. Analyse des résultats qui seront visualisés et interprétés à l'aide de Python et des scripts.

6 Expérimentations

Dans le cadre de nos tests, nous avons dû distribuer la charge de travail sur plusieurs machines afin de pouvoir exécuter toutes les expérimentations dans des délais raisonnables. Chaque membre de notre groupe a pris en charge des tailles de grille allant de 1 à 10, et les tests ont été répartis de manière à optimiser l'utilisation des ressources disponibles.

Cela a permis de répartir la charge de manière équitable et d'éviter une surcharge sur une seule machine. Toutefois, ces tests ont pris un temps considérable et ont exigé une grande quantité d'énergie, ce qui a entraîné des difficultés logistiques. En effet, malgré la distribution des tests sur plusieurs machines, nous

avons rencontré des problèmes techniques avec nos ordinateurs personnels, qui ont commencé à se bloquer après quelques exécutions de tests. En conséquence, nous n'avons pas pu réaliser tous les tests prévus, et nous avons dû recourir à d'autres machines pour compléter le processus. Cette gestion de la charge de travail a ainsi été un défi important et a conduit à des délais plus longs que prévus pour l'exécution complète des tests.

6.1 Analyse de l'impact du critère de sélection dans MCTS-UCB

Dans le cadre de notre étude sur l'algorithme MCTS, nous nous sommes interrogés sur l'influence du critère de sélection final dans la phase de décision. En effet, après l'exploration de l'arbre de jeu, UCB doit choisir le meilleur coup à jouer. On avait 2 idées :

— Privilégier le nœud avec le taux de victoire le plus élevé, ce qui était le plus logique, car il maximise directement la probabilité de gain, indépendamment du nombre de simulations.

— Sélectionner le nœud le plus visité, ce qui semblait plus pratique, car ubc tend à équilibrer l'exploration et l'exploitation, et donc un grand nombre de visites suggère une exploration approfondie et une fiabilité statistique.

Pour tester cette hypothèse, nous avons implémenté deux variantes de MCTS :

— MCTS : Retient le nœud le plus visité après simulation.

— MCTS2 : Sélectionne le nœud avec le meilleur taux de victoire (moyenne des résultats des simulations).

Nous avons ensuite confronté ces algorithmes dans des matchs croisés dans des simulations de grille de taille 4 : MCTS (Rouge) vs MCTS2 (Bleu) et MCTS2 (Rouge) vs MCTS (Bleu), tout en conservant l'avantage du premier coup pour Rouge, afin d'isoler l'effet du critère de sélection.

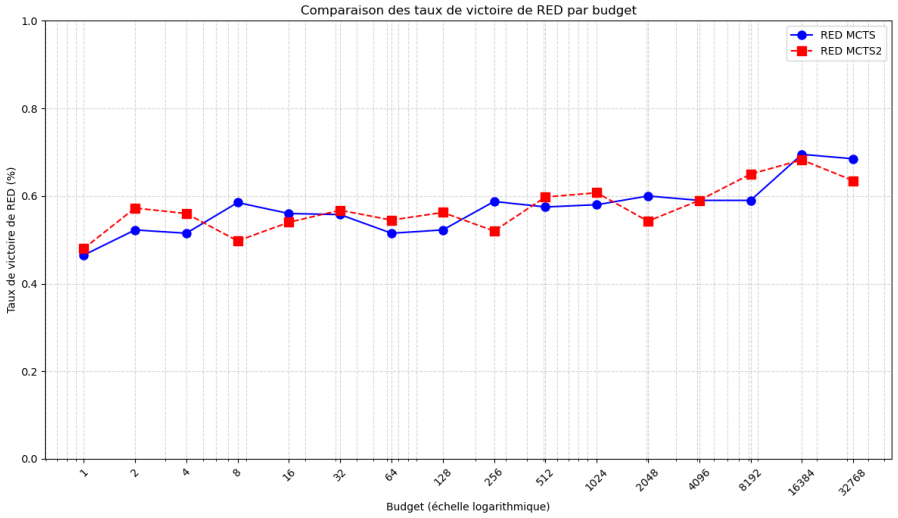


FIGURE 5 – pourcentage de victoire du premier joueur(RED) dans les 2 combinaisons

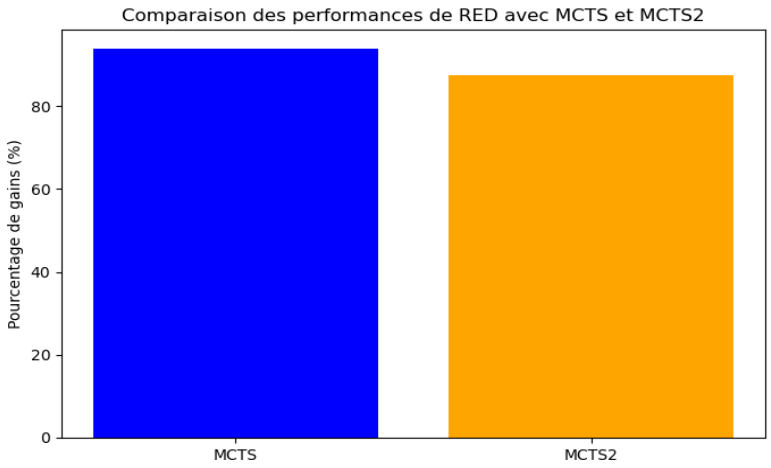


FIGURE 6 – pourcentage total de gain de RED avec MCTS et de MCTS2

Analyse et synthèse :

Quelque soit l'algorithme utilisé (MCTS ou MCTS2), c'est systématiquement le joueur qui commence (Rouge) qui gagne, avec des taux de victoire très proches entre les deux approches.

Aussi, il y a peu de différence entre MCTS et MCTS2 (causé par le random). Les courbes de performance sont quasi superposées, indiquant que le choix du nœud "le plus visité" ou "au meilleur taux" à un impact marginal.

Alors on peut vraiment conclure que UTC équilibre quasi parfaitement l'exploration et l'exploitation et le nœud le plus visité, c'est le nœud ayant le plus haut pourcentage de gain.

6.2 Conditions de tests

Nous avons lancé plusieurs expérimentations allant d'un budget de 1 à 32768 (2^0 à 2^{15}) et pour chaque budget, un ratio de budget Rouge :Bleu tel que :

- 1 : 1 => Les deux joueurs ont le même budget
- 1 : 2 => Le deuxième joueur a 2 fois plus de budget
- 2 : 1 => Le premier joueur a 2 fois plus de budget
- 1 : 5 => Le deuxième joueur a 5 fois plus de budget
- 5 : 1 => Le premier joueur a 2 fois plus de budget

Nous avons fixé RED comme starting Player, et on a mené des expériences pour les 4 combinaisons possibles : MCTS vs MCTS, RAVE vs RAVE, MCTS vs RAVE et RAVE vs MCTS. Pour chacune d'elle, nous avons pu atteindre les tailles de la grille allants de 1 à 10. Par manque de temps, nous n'avons pas pu aller plus loin.

6.3 Résultat MCTS vs MCTS

1. Taille de grille : 4

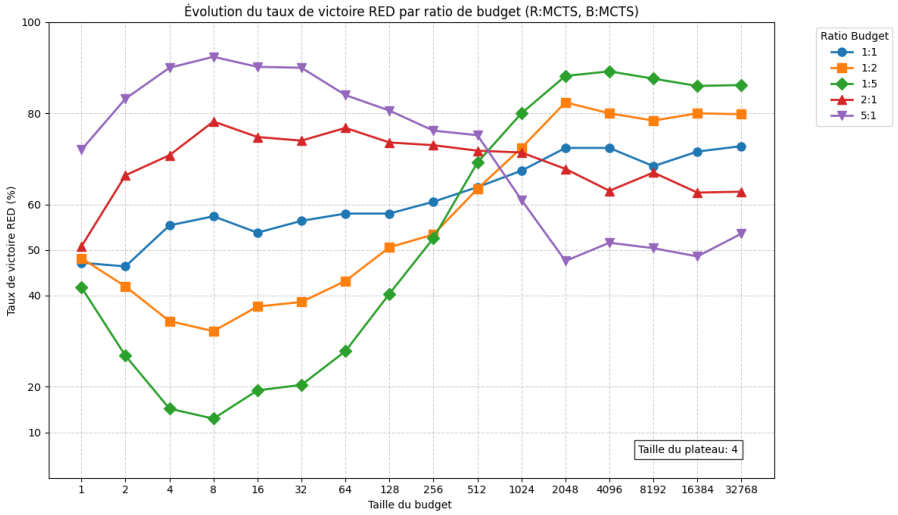


FIGURE 7 – Courbes pour la taille 4.

2. Taille de grille : 8

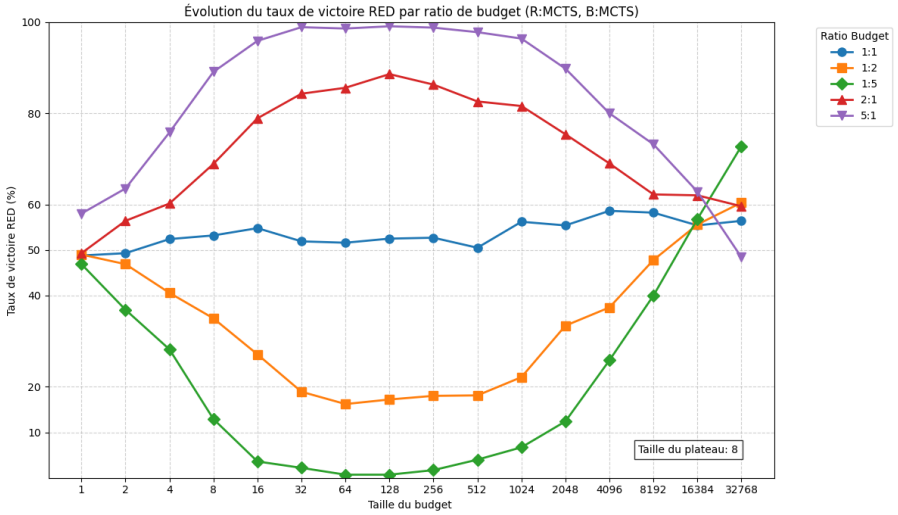


FIGURE 8 – Courbes pour la taille 8.

3. Taille de grille : 10

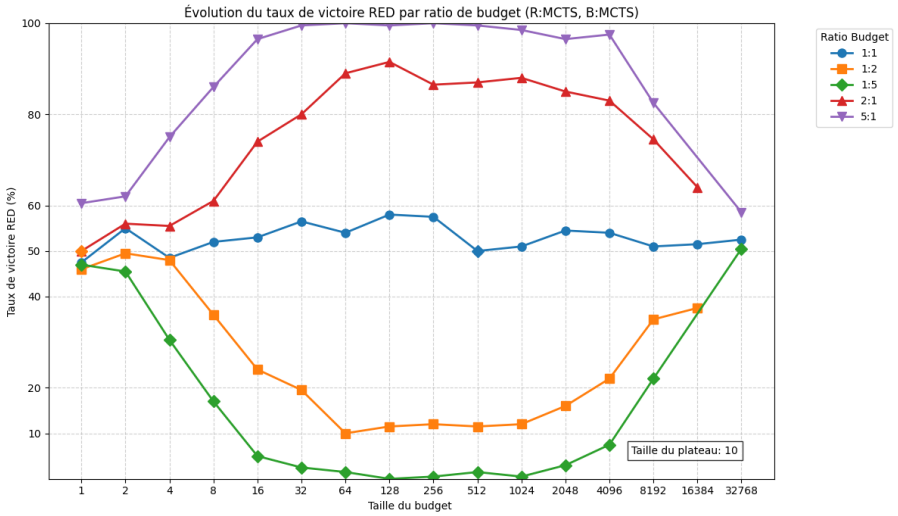


FIGURE 9 – Courbes pour la taille 10.

Les graphes fournis montrent le taux de victoire de RED (premier joueur) en fonction de son budget de simulations pour différents plateaux de Hex. Chaque courbe représente un ratio de budget Rouge :Bleu. De façon générale, on observe que :

1. L'avantage du premier coup existe : avec des budgets égaux (1 :1), RED garde généralement un taux de victoire légèrement supérieur à 50%. Cependant, cet avantage n'est pas écrasant sans aide supplémentaire, surtout sur les grandes grilles.
2. Un déséquilibre de budget influence fortement le résultat : si BLUE a plus de budget (ratios 1 :2 ou 1 :5 défavorables à RED), il parvient à annuler voire inverser l'avantage du premier joueur pour des budgets totaux modestes à intermédiaires. Inversement, si RED a un budget supérieur (2 :1 ou 5 :1), son avantage est grandement amplifié et il gagne la plupart des parties tant que les budgets ne sont pas trop élevés.
3. Effets de seuil (budget total) : A très faible budget, RED gagne autour de 50% (les deux joueurs jouent quasi au hasard) ; à budget intermédiaire, le joueur avec un budget supérieur domine (avantage de BLUE si RED est handicapé, ou presque 100% de victoires de RED s'il a beaucoup plus de budget) ; enfin, au-delà d'un certain seuil de budget absolu pour Rouge, la tendance s'inverse : sur 4×4 par exemple, il suffit d'environ 256–512 simulations pour que RED commence à remonter, et l'écart se resserre à nouveau. Lorsque les deux joueurs disposent tous deux de nombreuses simulations, le jeu approche d'un niveau quasi-optimal, ce qui redonne du poids à l'avantage structurel du premier joueur (RED) même s'il avait moins de budget au départ.

Conclusion :
Un déséquilibre de budget en faveur de Bleu peut complètement neutraliser l'avantage du premier joueur aux budgets moyens, mais cet avantage refait surface dès que Rouge peut explorer suffisamment de coups.

6.4 Résultat MCTS vs RAVE

1. Taille de grille : 4

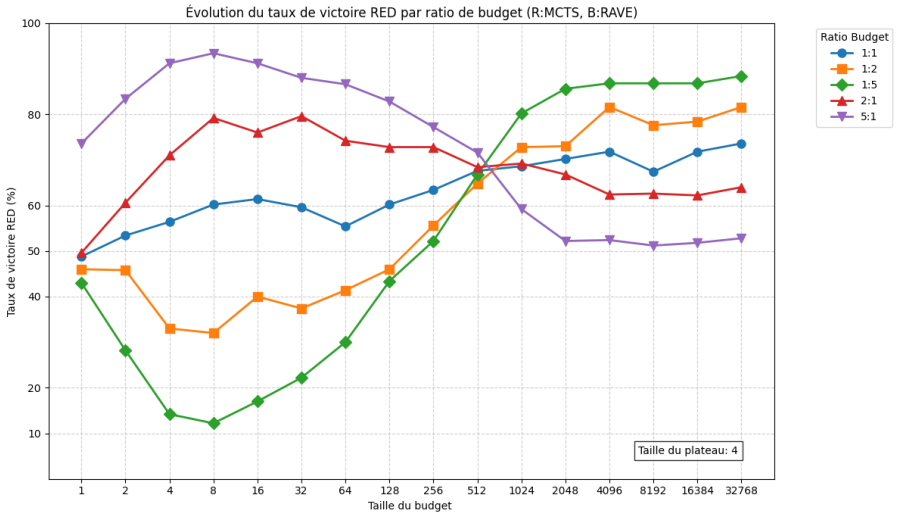


FIGURE 10 – Courbes pour la taille 4.

2. Taille de grille : 8

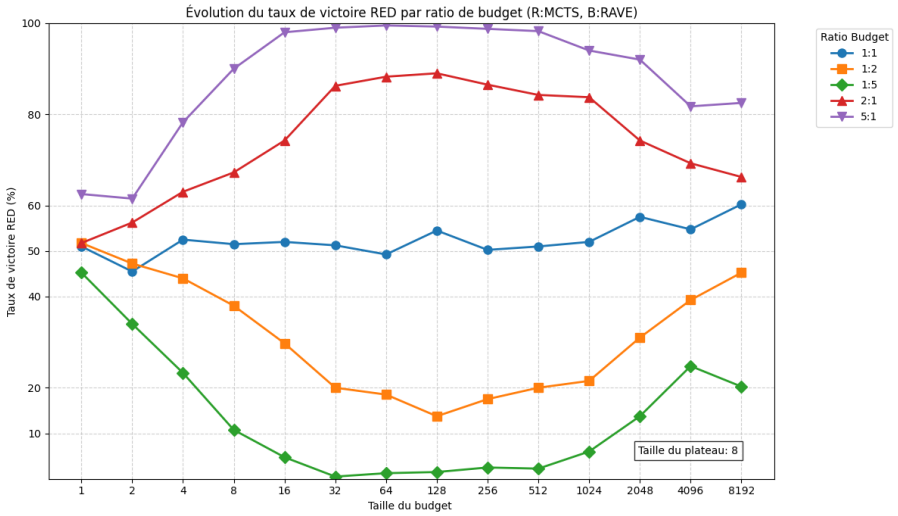


FIGURE 11 – Courbes pour la taille 8.

3. Taille de grille : 10

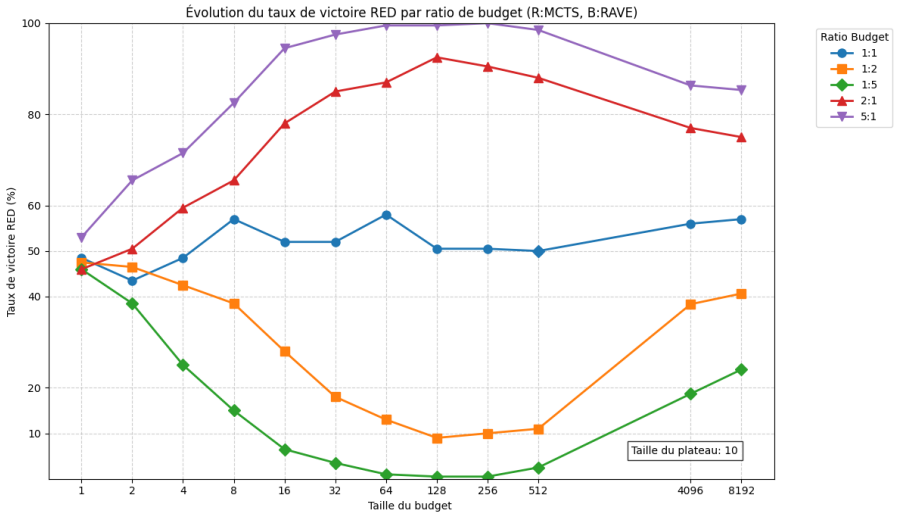


FIGURE 12 – Courbes pour la taille 10.

Les graphes fournis montrent le taux de victoire de RED (premier joueur) utilisant MCTS en fonction de son budget de simulations pour différents plateaux de Hex, contre BLUE qui utilise RAVE. Chaque courbe représente un ratio de budget Rouge :Bleu. De façon générale, on observe que :

1. L'avantage du premier joueur est visible que pour les petites tailles comme 4x4, or pour les grandes tailles, un taux de victoires de 50-60% est remarqué pour le ratio 1 :1.
2. On remarque aussi le point de convergence pour les tailles 4 et 6, qui a un peu le même effet que dans MCTS vs MCTS, en donnant plus d'avantage pour le premier joueur peu importe le budget.
3. Un déséquilibre de budget influence quasi totalement le résultat. Par exemple, pour la taille 8x8 et le ratio 5 :1, MCTS arrive à gagner presque à 100% pour un budget de x5 contre Rave avec un budget de x1 et vice versa, où RAVE arrive à dévaster MCTS pour un ratio plus grand, même pour les petits budgets.

Or, regardons RAVE vs MCTS avec RAVE comme premier joueur.

6.5 Résultat RAVE vs MCTS

1. Taille de grille : 4

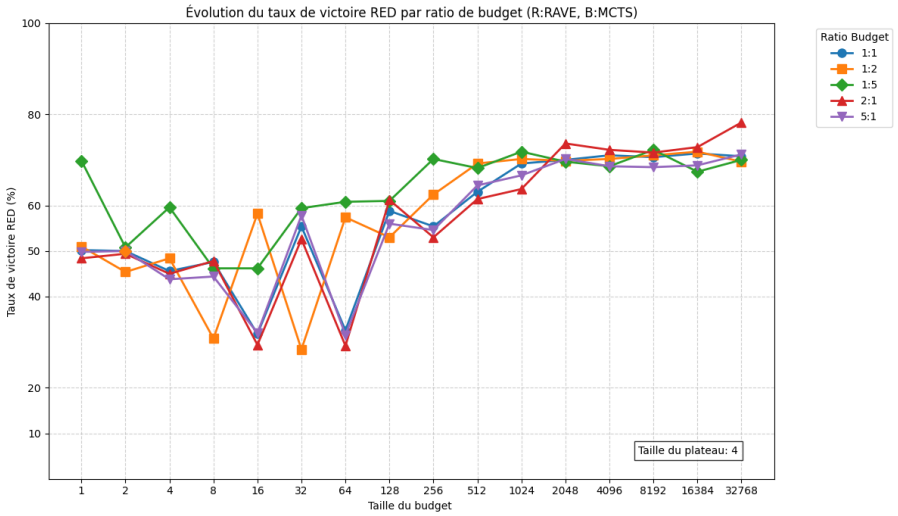


FIGURE 13 – Courbes pour la taille 4.

2. Taille de grille : 8

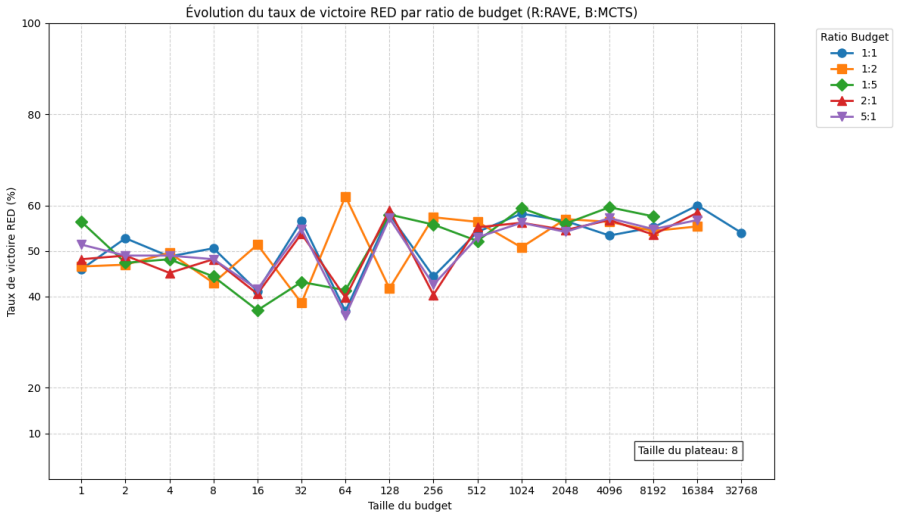


FIGURE 14 – Courbes pour la taille 8.

3. Taille de grille : 10

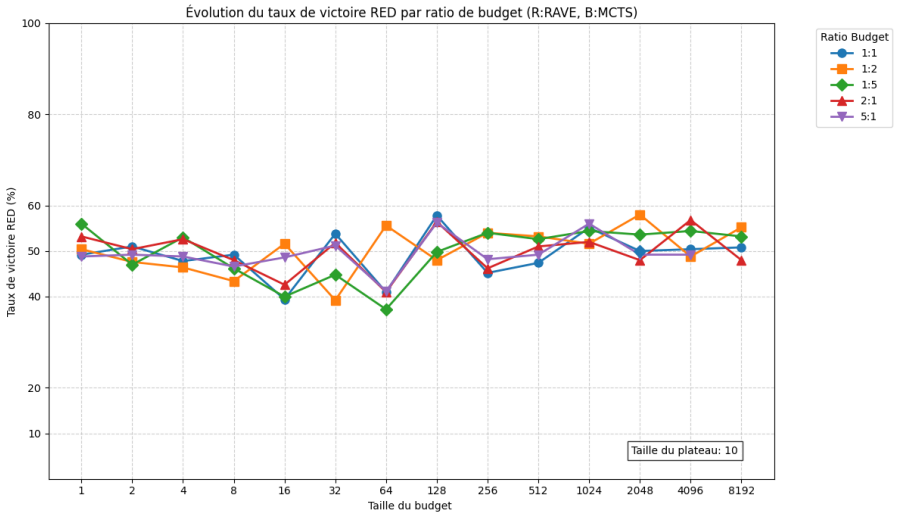


FIGURE 15 – Courbes pour la taille 10.

Les graphes fournis montrent le taux de victoire de RED (premier joueur) utilisant cette fois ci RAVE, contre BLUE qui utilise MCTS. Chaque courbe représente un ratio de budget Rouge :Bleu. De façon générale, on observe que :

1. Même avec un budget multiplié par 5 (ratio 1 :5), BLUE (MCTS) ne parvient pas à inverser la tendance. Sur une grille 6×6, BLUE (MCTS) avec 5× plus de simulations que Rouge (RAVE) n'atteint qu'un taux de victoire de 30–40%, alors que RED conserve une avance significative.
2. RAVE résiste mieux au désavantage de budget : Les courbes montrent que RED maintient un taux de victoire presque >50% même avec un ratio 1 :2 (moitié moins de simulations que Bleu), illustrant la robustesse de RAVE face aux contraintes de ressources.
3. Un déséquilibre de budget n'influence pas trop les résultats. RAVE arrive quand même à limiter le gain de MCTS, même avec un budget de 5x moins.

Conclusion :
Bien que le premier joueur (Rouge) ait un avantage structurel dans Hex, RAVE atténue ou renforce cet avantage selon qui l'utilise. Quand RAVE est le premier joueur, sa supériorité algorithmique le rend quasi imbattable, même face à des budgets MCTS démesurés.

6.6 Résultat RAVE vs RAVE

1. Taille de grille : 4

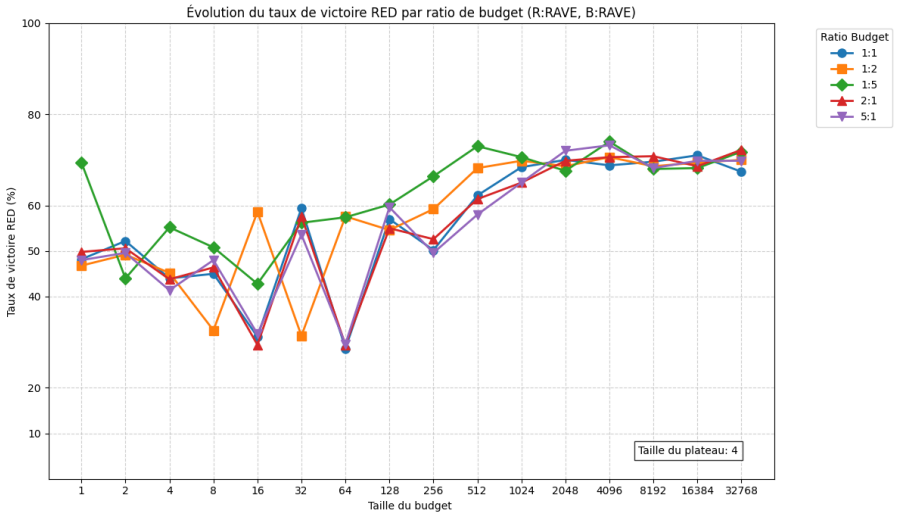


FIGURE 16 – Courbes pour la taille 4.

2. Taille de grille : 8

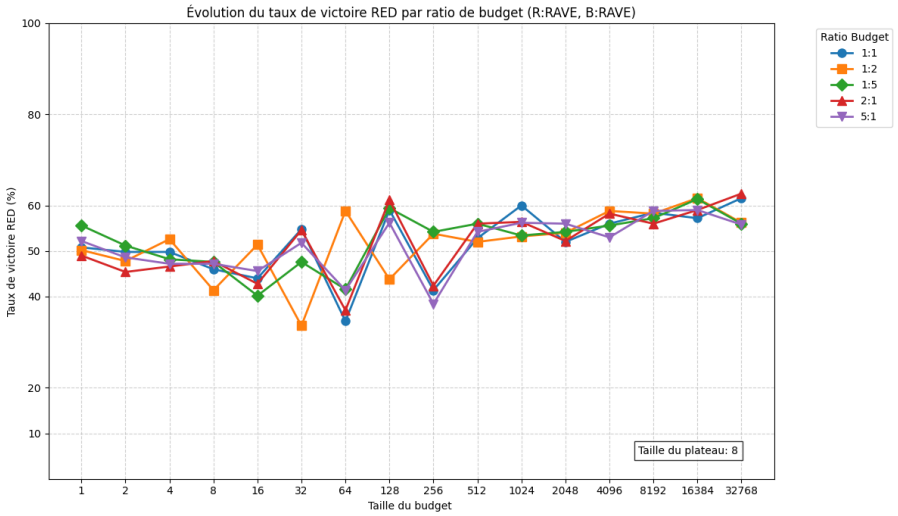


FIGURE 17 – Courbes pour la taille 8.

3. Taille de grille : 10

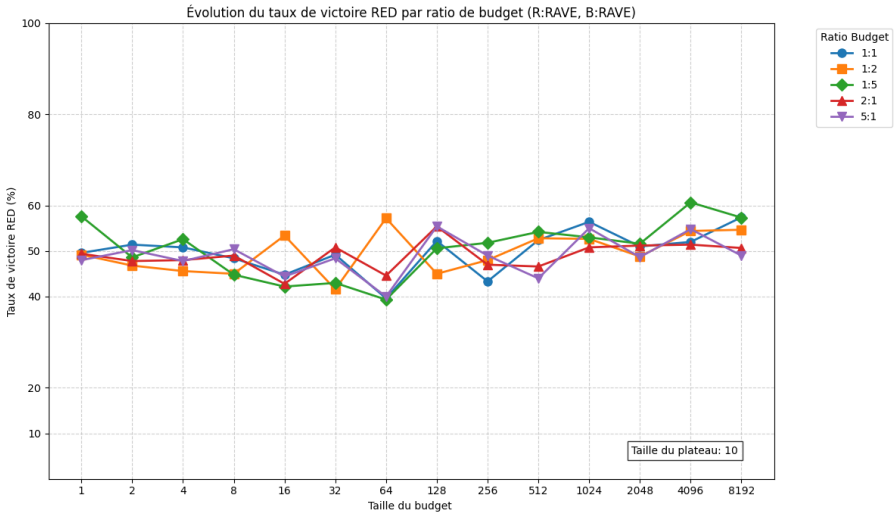


FIGURE 18 – Courbes pour la taille 10.

Les graphes fournis montrent le taux de victoire de RED (premier joueur) utilisant RAVE, contre BLUE qui utilise aussi RAVE. Chaque courbe représente un ratio de budget Rouge :Bleu. De façon générale, on observe que :

1. L'avantage du premier joueur RED reste dominant à budget élevé, quelle que soit la taille de la grille. Sur toutes les grilles (4x4 à 10x10), dès que Rouge a suffisamment de simulations, il reprend le dessus grâce à son premier coup. Plus la grille est grande, plus il faut de budget pour que cet avantage s'exprime pleinement.
2. Un Bleu mieux doté en budget peut temporairement annuler l'avantage de Rouge, surtout sur les grandes grilles.
3. Contrairement au cas MCTS vs MCTS, où un écart de budget pouvait créer des différences de performance très marquées, le duel RAVE vs RAVE montre des écarts plus modérés. L'utilisation conjointe de RAVE chez les deux joueurs tend à équilibrer les chances, même lorsque le budget est légèrement déséquilibré.

Conclusion :
RAVE permet aux deux joueurs d'atteindre un bon niveau de jeu plus rapidement, ce qui réduit l'écart dû aux ressources seules. L'équilibre stratégique dépend donc plus de la taille de la grille et du budget global que de l'algorithme, dès lors que RAVE est utilisé des deux côtés.

6.7 Test supplémentaire sur le paramètre d'exploitation de MCTS

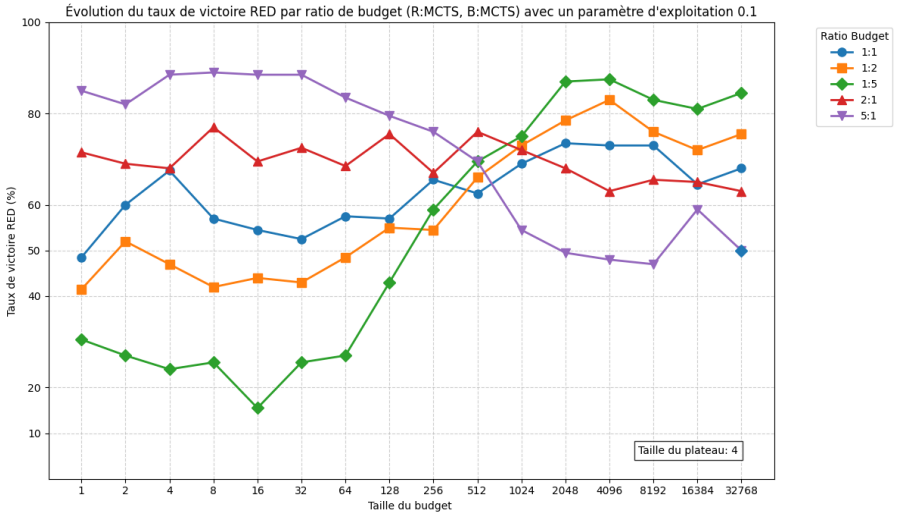


FIGURE 19 – Courbes pour la taille 4 avec un paramètre d'exploitation de 0.1.

Par défaut, nous avons fixé le paramètre d'exploitation De MCTS a $\sqrt{2}$. Nous avons voulu voir si ce paramètre peut affecter la performance de sélection du meilleur enfant dans la formule UCB de MCTS. Donc, nous avons modifié le paramètre d'exploitation a 0.1. Ce graphe étant presque pareil au graphe obtenu dans les résultats de MCTS vs MCTS pour une taille de 4, nous pouvons conclure que le paramètre d'exploitation de UCB n'a pas une grande influence sur la formule pourvu qu'il soit entre 0.1 et $\sqrt{2}$.

6.8 Synthèse de l'impact du budget sur les performances

L'ensemble des expériences menées (MCTS vs MCTS, RAVE vs MCTS, RAVE vs RAVE) montre que le rapport de budget entre les deux joueurs influence fortement le taux de victoire lorsque le budget total de simulations est faible ou modéré, un effet d'autant plus prononcé sur les grandes grilles. En revanche, lorsque les deux joueurs disposent d'un budget de simulations très élevé (de l'ordre de plusieurs milliers), l'avantage conféré au premier joueur redevient prépondérant, même si ce dernier dispose d'un budget moindre que son adversaire. Enfin, l'utilisation de l'approche RAVE permet d'atténuer ces écarts de performance liés au budget, sans pour autant les éliminer totalement.

7 Conclusion

Ce projet a permis d'explorer en profondeur les algorithmes de Monte Carlo Tree Search (MCTS) et l'optimisation RAVE dans le cadre du jeu de Hex. En nous concentrant sur l'optimisation de ces techniques, nous avons pu analyser et comparer leur efficacité pour prendre des décisions stratégiques dans un environnement de jeu dynamique et compétitif. L'application de l'optimisation RAVE a montré des résultats prometteurs en termes d'amélioration des performances de l'algorithme MCTS, rendant le processus de recherche plus rapide et plus pertinent.
Par ailleurs, nous avons mis en évidence un résultat théorique fondamental ; que si la stratégie gagnante appartenait au second joueur, alors le premier pourrait jouer simplement un coup pour rien, puis adopter la stratégie du second joueur. Son premier coup ne pouvant être un handicap, le premier joueur gagne. Par conséquent, le second joueur ne peut avoir de stratégie gagnante et c'est donc le premier qui en possède une, qu'on ne peut pas voir sur les grands plateaux.

Les tests expérimentaux ont été un aspect central du projet. Pour chaque taille de grille, les tests ont été soigneusement répartis entre plusieurs machines afin d'éviter la surcharge d'une seule machine, ce qui a permis de mieux répartir la charge de travail et de garantir l'exécution efficace des simulations.

Au-delà des résultats techniques, ce projet a renforcé notre compréhension des algorithmes de recherche arborescente, tout en nous offrant l'opportunité d'améliorer nos compétences en gestion de projets expérimentaux et en optimisation algorithmique. L'expérience a également souligné l'importance d'une gestion efficace des ressources, tant humaines que matérielles, dans un projet complexe, ce qui sera utile pour de futures recherches ou projets en intelligence artificielle.

En conclusion, ce projet a permis d'approfondir notre compréhension du jeu de Hex et des algorithmes d'intelligence artificielle appliqués à ce type de jeu. Les résultats obtenus ouvrent la voie à de futures améliorations et explorations, notamment en ce qui concerne l'optimisation des stratégies de jeu pour des grilles de taille variable. Ce travail représente une base solide pour des recherches futures sur les algorithmes de recherche et d'optimisation dans des jeux combinatoires complexes.

On remercie M. Bruno ZANUTTINI pour son suivi tout au long de ce projet.

8 Références

Références

[1] Cameron Browne, *Hex Strategy : Making the Right Connections*, A K Peters/CRC Press, 2000.

[2] Wikipedia Contributors, "Hex (jeu de plateau)", 2023. Disponible sur : `https://fr.wikipedia.org/wiki/Hex`

[3] Wikipedia Contributors, "Havannah", 2023. Disponible sur : `https://fr.wikipedia.org/wiki/Havannah`

[4] Ryan Hayward, *A Thesis on Hex*, Université de l'Alberta. Disponible sur : `https://webdocs.cs.ualberta.ca/~hayward/theses/ph.pdf`

[5] "Stratégies pour le jeu de Hex infini", *Pour la Science*, 2023. Disponible sur : `https://www.pourlascience.fr/sr/logique-calcul/strategies-pour-le-jeu-de-hex-infini-26436.php`

[6] "Le jeu de Hex et les mathématiques", *Palais de la Découverte*. Disponible sur : `https://www.palais-decouverte.fr/fileadmin/fileadmin_Palais/fichiersContribs/au-programme/expos-permanentes/mathematiques/Bloc_Formes/pdf_revue/decouverte_math_383.pdf`