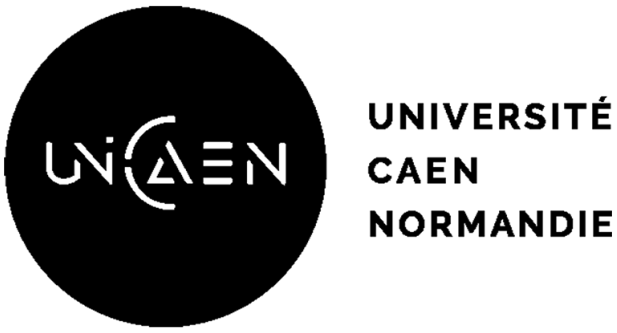


Université de Caen Normandie

Rapport : Jeu de Hex

EL-AASMI Yassine	22011193
QACH Yahya	22109246
MORABET Ahmed	22107739
ELOUARDI Salah eddine	22110344

L3 Informatique

Groupe 2A

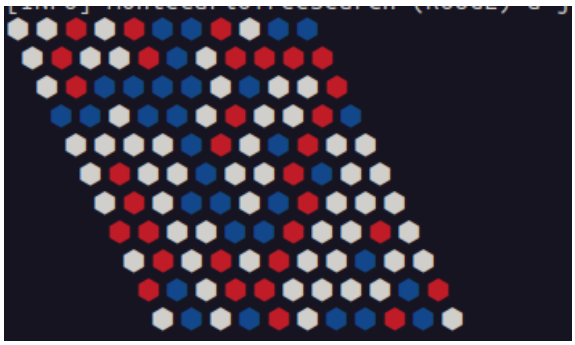
Chargé de TP : ZANUTTINI Bruno

March 28, 2025

Contents

1 Introduction

Le jeu de Hex est un jeu combinatoire abstrait pour deux joueurs, joué sur un plateau composé d'hexagones taille $n \times n$ (souvent, 14×14). Chaque joueur tente de relier deux côtés opposés du plateau avec ses pions en placent alternativement leurs pions sur des cases vides jusqu'a atteindre leurs buts, dans notre Implémentation le Joueur Bleu essaie de relier **le Coté gauche au Coté Droit** tandis que le Joueur Rouge essaie de relier **le Haut au Bas**.



Exemple de hexGame sur une grille 11x11 avec Rouge gagnant

2 Objectif

2.1 Problématique

La combinatoire élevée du jeu rend son analyse complexe (un grand nombre de possibilité et de résultats), ce qui en fait un terrain d'étude idéal pour les algorithmes d'intelligence artificielle, car ces algorithmes sont conçus pour prendre des décisions optimales dans des situations où il existe beaucoup d'options possibles.

L'objectif ici est d'explorer l'influence de deux paramètres sur la performance des joueurs : **le budget d'itérations et la taille de la grille du jeu**.

Le budget de simulation correspond au nombre d'itérations qu'un joueur peut effectuer avant de prendre une décision, l'algorithme utilisé pour cette étude est l'algorithme de recherche arborescente de Monte Carlo (ou MCTS pour Monte Carlo Tree Search), qui permet d'explorer différentes options de jeu et d'évaluer la meilleure action possible à chaque tour en explorant l'arbre de recherche et simulant des parties entières à chaque itération pour converger vers la décision optimale après un nombre d'itérations.

La question principale qu'on essaie de résoudre est : **quelle influence le rapport de budget entre les deux joueurs a-t-il sur leur probabilité de victoire en fonction de la taille de la grille ?**

2.2 Points-clés et Grandes étapes

Pour essayer de répondre à cette question, il faut s'intéresser à deux points :

- **Budget d'iterations**: Comment le rapport de budget entre les deux joueurs influence-t-il leur probabilité de victoire ?

  Exemple: Imaginons que le joueur A ait un budget élevé (il peut faire 1 000 calculs avant chaque coup) tandis que le joueur B a un budget réduit (il peut faire 100 calculs), Plus le budget est élevé, plus un joueur pourra approximer les meilleurs coups à jouer.

  On veut comprendre si cet écart donne un avantage clair à A, et si oui, dans quelle mesure

- **Taille de la grille** : Quelle est la relation entre la taille de la grille et le budget sur l'influence à la probabilité de victoire des joueurs ?

  Par exemple, une grille de petite taille peut limiter les options stratégiques, tandis qu'une grande grille ouvre davantage de possibilités. Cela pourrait influencer la façon dont le budget de calcul affecte la victoire des joueurs

En étudiant ces deux paramètres (budget et taille de la grille), l'objectif est d'observer si un joueur ayant un budget supérieur a toujours un avantage, ou si cet avantage varie en fonction des dimensions du plateau.

3 Fonctionnalitées

Pour Atteindre l'objectif de projet on a été mené à implémenter les fonctionnalitées suivantes :

- Un jeu de Hex avec différents types de joueurs: humain,robot.
- Une visualisation du jeu en ligne de commande et interface graphique
- Un algorithme de recherche arborescente de Monte Carlo (MCTS)
- Une Optimisation RAVE sur MCTS
- Tests unitaires pour la robustesse de code
- Expérimentations configurables et automatisées avec différents paramètres
- Analyse des résultats de l'expérimentation.

4 Répartition des Taches

La répartition des responsabilités est comme suit :

- **ELOUARDI Salah eddine**
  - Grille de jeu, règles de jeu
  - Structure (MVC,Design patterns, etc)
  - Expérimentation
- **QACH Yahya**
  - Algorithme MCTS
  - Optimisation RAVE-MCTS
- **MORABET Ahmed**
  - Tests Unitaires
  - Interfaces Graphiques (Affichage Initial, Configuration des joueurs)
  - scripts d'automatisation
- **EL-AASMI Yassine**
  - Interfaces Graphiques (Affichage de la grille, déroulement des tours)
  - Jeu de Hex (Placement des joueurs, conditions de victoires)
  - Analyse de l'expérimentation

5 Architecture et Design Patterns

5.1 Architecture du projet

L'ensemble des fichiers du projet sont répartis en packages comme suit : **Livraison :** le package principal du rendu contenant à son tour tous les autres sous-packages.

- **src :** le package contenant le code source
  - **model :** le package contenant les classes du modèle (Orchestrateur, Grille, etc ..)
    - **mcts :** le package contenant les classes concernant l'algorithme mcts (MCTS.java, RAVE.java, Noeud.java)
    - **player :** le package contenant toutes les classes concernant les joueurs (PlayerFactory, les differentes strategies, etc..)
  - **vue :** le package contenant toutes les classes concernant la partie graphique
  - **controller :** le package contenant toutes les classes concernant le controlleur entre la vue et le modèle
  - **utils :** le package contenant toutes les classes utilitaires de support (messageHandler, pour gestion des entrées et affichage etc..)
  - **config :** le package contenant toutes les classes de configuration du jeu (Constants.java qui gère les constantes partagées avec toutes les autres classes) pour une configuration centralisée et modulaire
- **experimentation :** le package contenant tous les fichiers concernant l'expérimentation (les logs, les résultats en csv, etc ..)
- **testUnitaire :** le package contenant touts les sous-packages et les classes concernant les tests unitaires du modèle
- **lib :** le package contenant toutes les librairies nécessaires au fonctionnement du projet (JUnit, JSON)
- **javadoc :** le package contenant la documentation du code
- **rapport :** le package contenant le rapport en pdf et code source Latex
- **ressources :** le package contenant d'autres ressources nécessaires (images pour l'interface graphique).

5.2 1. Modèle-Vue-Contrôleur (MVC)

L'architecture MVC (Modèle-Vue-Contrôleur) a été utilisée dans ce projet pour séparer clairement les responsabilités et faciliter la maintenance du code. Voici comment elle est implémentée :

- **Modèle** : Gère la logique métier du jeu, y compris la grille, les joueurs, et les règles du jeu.
- **Vue** : Affiche l'interface utilisateur et capture les interactions de l'utilisateur.
- **Contrôleur** : Fait le lien entre le modèle et la vue, en traitant les événements et en mettant à jour l'état du jeu.

Exemples d'Utilisation de MVC dans le Projet

- **Création de Joueur** :
  - Lorsqu'un utilisateur clique sur un bouton pour créer un joueur, la vue capture cet événement et le transmet au contrôleur.
  - Le contrôleur appelle la méthode `creerPlayer` via la `PlayerFactory` pour instancier un joueur (humain ou IA).
  - Le modèle est mis à jour avec le nouveau joueur, et la vue est rafraîchie pour refléter ce changement.
- **Placement de Pion** :
  - La vue détecte un clic sur le plateau et transmet les coordonnées du clic au contrôleur.
  - Le contrôleur demande au modèle de placer un pion à la position spécifiée.
  - Le modèle vérifie si le coup est valide, met à jour la grille, et notifie le contrôleur en cas de victoire ou de défaite.
  - La vue est ensuite mise à jour pour afficher le nouvel état du plateau.
- **Vérification de Victoire** :
  - Après chaque action (par exemple, un clic sur le bouton "Suivant"), le contrôleur interroge le modèle pour vérifier si une victoire a été atteinte.
  - Si une victoire est détectée, le contrôleur met à jour la vue pour afficher un message de fin de partie.

Cette séparation des responsabilités permet une meilleure modularité et facilite les tests unitaires.

5.3 2. Pattern Factory

Le Pattern Factory est utilisé pour centraliser la création des objets `PlayerStrategy`. Cela permet d'éviter de disperser la logique d'instanciation dans le code et facilite l'ajout de nouvelles stratégies.

Avantages du Pattern Factory

- **Maintenabilité** : Toute la logique de création est centralisée dans une seule classe (`PlayerFactory`), ce qui simplifie les modifications futures.
- **Extensibilité** : Ajouter une nouvelle stratégie (par exemple, une IA basée sur un autre algorithme) ne nécessite que des modifications minimales dans la factory.
- **Séparation des responsabilités** : La logique de création est séparée de la logique métier, ce qui rend le code plus clair.

Exemple de Code

```
public class PlayerFactory {
    public static PlayerStrategy createPlayer(String type) {
        switch (type) {
            case "Humain":
                return new HumanStrategy();
            case "Aléatoire":
                return new RandomStrategy();
            case "MonteCarlo":
                return new MonteCarloStrategy();
            default:
                throw new IllegalArgumentException("Type de joueur inconnu : " + type);
        }
    }
}
```

Diagramme

Le diagramme suivant illustre le fonctionnement du pattern Factory dans le projet :

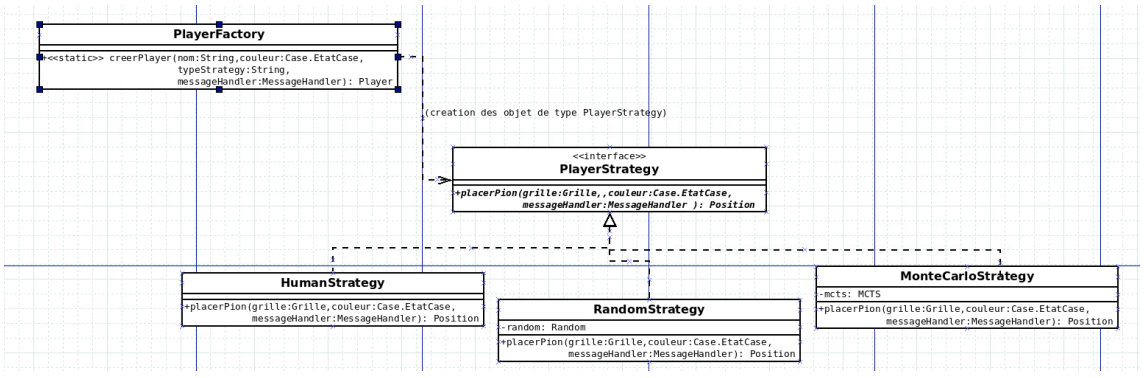


Diagramme du Pattern Factory

5.4 3. Pattern Strategy (Placement de Pion)

Le Pattern Strategy est utilisé pour définir différentes stratégies de placement de pion (humain, aléatoire, Monte Carlo). Cela permet de rendre les stratégies interchangeables sans modifier la classe cliente (`Player`).

Avantages du Pattern Strategy

- **Flexibilité** : Les stratégies peuvent être changées dynamiquement pendant l'exécution.
- **Évolutivité** : Ajouter une nouvelle stratégie ne nécessite pas de modifier le code existant.
- **Réutilisabilité** : Les stratégies peuvent être réutilisées dans d'autres parties du projet ou dans d'autres projets.

Exemple de Code

```
public interface PlayerStrategy {
    Position placerPion(Grille grille);
}

public class HumanStrategy implements PlayerStrategy {
    @Override
    public Position placerPion(Grille grille) {
        // Logique pour un joueur humain
    }
}

public class RandomStrategy implements PlayerStrategy {
    @Override
    public Position placerPion(Grille grille) {
        // Logique pour un joueur aléatoire
    }
}

public class MonteCarloStrategy implements PlayerStrategy {
    @Override
    public Position placerPion(Grille grille) {
        // Logique pour un joueur utilisant MCTS
    }
}
```

Diagramme

Le diagramme suivant illustre le fonctionnement du pattern Strategy pour les joueurs :

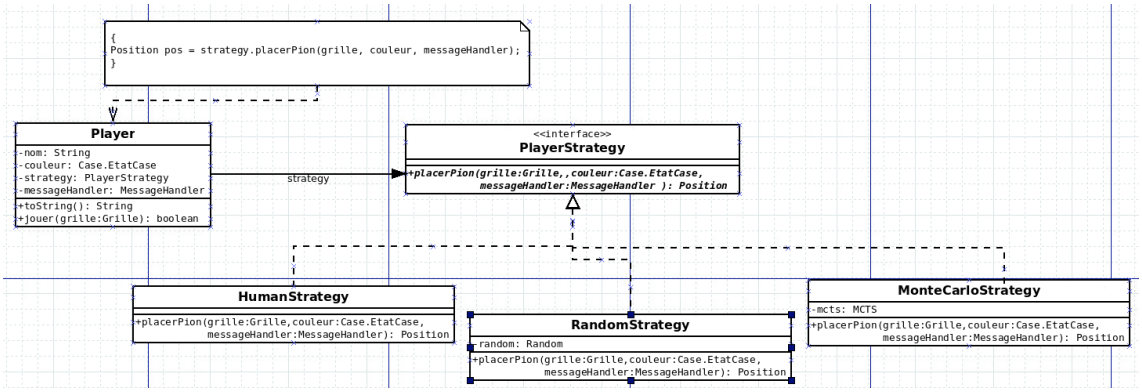


Diagramme du Pattern Strategy (Player)

5.5 4. Pattern Strategy (Gestion des Messages)

Le Pattern Strategy est également utilisé pour gérer l'affichage des messages (console ou interface graphique). Cela permet de séparer la logique d'affichage du reste de l'application.

Avantages

- **Modularité** : Changer de mode d'affichage (console ou graphique) ne nécessite que de changer la stratégie utilisée.
- **Testabilité** : Les stratégies peuvent être testées indépendamment du reste de l'application.
- **Maintenabilité** : La logique d'affichage est centralisée dans des classes dédiées.

Exemple de Code

```
public interface MessageHandler {
    void afficherMessage(String message);
}

public class ConsoleMessageHandler implements MessageHandler {
    @Override
    public void afficherMessage(String message) {
        System.out.println(message);
    }
}
```

```
public class GraphicMessageHandler implements MessageHandler {
    @Override
    public void afficherMessage(String message) {
        // Afficher le message dans l'interface graphique
    }
}
```

Diagramme

Le diagramme suivant illustre le fonctionnement du pattern Strategy pour la gestion des messages :

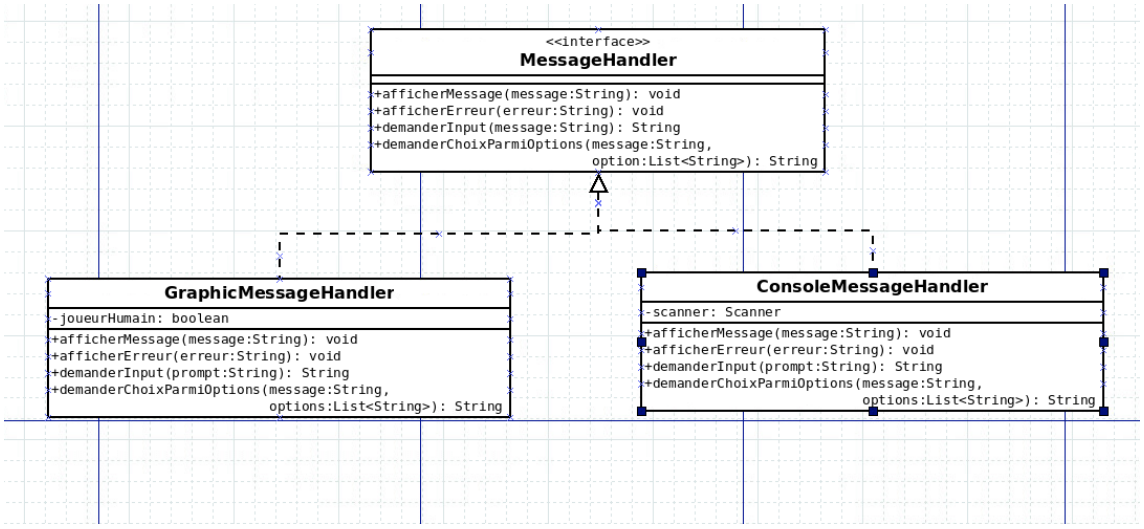


Diagramme du Pattern Strategy (Messages)

6 Monte Carlo Tree Search

6.1 Explication

MCTS : Monte Carlo Tree Search est un algorithme de prise de décision utilisé pour explorer les choix/coups possibles dans un jeu ou un problème complexe. Il est particulièrement utile pour les jeux comme Hex, où il y a un grand nombre de coups possibles et où il est difficile de prédire quel coup est le meilleur.

L'idée principale de **MCTS** est de simuler de nombreuses parties aléatoires à partir de l'état actuel du jeu, puis d'utiliser les résultats de ces simulations pour décider quel coup est le plus prometteur.

6.2 Fonctionnement

L'arbre de Recherche de l'algorithme est représentée par des Noeuds, où chaque noeud contient un ensemble d'informations (Le coup concérné, la grille representant l'état du jeu après cette action, le noeud parent, le nombre de victoires, le nombre de pertes, et le nombre de visites).

On désigne par un **noeud terminal/état final**, un noeud où il n'y a plus de coups possibles depuis son état (toutes les cases sont remplies) ou il existe un vainqueur (jeu terminé).

On désigne par un **noeud complet**, un noeud où toute action possible depuis son état existe dans l'un de ses enfants, autrement dit; un noeud qui a le meme nombre d'enfants que de coups possibles depuis son état.

l'algorithme MCTS fonctionne en répétant 4 étapes Majeures de manière itérative et renvoie à la fin des itérations le coup qui le plus grand score Victoires/Défaites.

6.2.1 Étape 1 : Sélection

Objectif :
Parcourir l'arbre de recherche pour trouver un nœud "intéressant" à explorer où chaque noeud represente un coup et la grille resultante de ce coup. **Comment** :

- l'algorithme parcours l'arbre en commencant à la racine de l'arbre (l'état actuel du jeu) et choisit l'enfant (coup possible) qui maximise la valeur obtenue en utilisant une formule appelée UCT (Upper Confidence Bound for Trees) à chaque fois jusqu'à atteindre **un noeud qui n'est pas été totalement éxploré et qui n'est pas un état final** cette epproche garantit que les nœuds partiellement développés ont une chance de produire de nouveaux enfants.
- **Formule UCT :**
$$\frac{w}{n} + C \cdot \sqrt{\frac{\ln(N)}{n}}$$
.
- w : est le nombre de parties gagnées simulées depuis ce noeud
- n : est le nombre de fois où le noeud a été visité
- N : est le nombre de fois où le nœud père a été visité
- C : est le paramètre d'exploration — en théorie égal à $\sqrt{2}$, en pratique choisi expérimentalement.

Cette formule équilibre entre **l'exploitaion** des noeuds et **l'exploration** d'autres neouds

- **Exploitation** : Choisir des coups qui ont déjà donné de bons résultats.
- **Exploration** : Essayer des coups qui n'ont pas encore été beaucoup explorés.
- $\frac{w}{n}$: C'est le terme d'exploitation, qui est la moyenne des gains pour cette action .
  - Quand w (le nombre de victoires) augmente, la valeur de la fraction augmente.
  - Quand n (le nombre de fois que cette action a été choisie) augmente, la valeur de la fraction diminue.
- $C \cdot \sqrt{\frac{\ln(N)}{n}}$: C'est le terme d'exploration.
  - N est le nombre total de simulations.
  - Quand n (le nombre de fois que cette action a été choisie) augmente, ce terme diminue, encourageant ainsi à explorer d'autres actions moins visitées.
  - C est une constante qui ajuste l'importance de l'exploration.
- Si une action a eu beaucoup de succès (grand w), la première partie de la formule ($\frac{w}{n}$) sera grande, ce qui favorise cette action et vice versa.

- Si une action n'a pas été beaucoup explorée (petit n), la deuxième partie de la formule ($C \cdot \sqrt{\frac{\ln(N)}{n}}$) sera grande, ce qui incite à explorer cette action et vice versa.

6.2.2 Étape 2 : Expansion

Objectif :
Ajouter un nouveau nœud à l'arbre pour explorer un coup non encore essayé.

Comment :

- Si **le noeud séléctionné dans la phase de selection a déja été visité et qu'il n'est pas un état terminal et qu'il n'est pas un noeud complet**, on le développe en lui ajoutant une action possible choisie au hasard depuis l'ensemble des coups possibles à partir de son état.
  cet ajout est représenté par un ajout d'un nouveau nœud contenant le coup joué et l'état résultant à l'arbre.
  Sinon si le nœud sélectionné n'a jamais été visité on passe directement à la phase de simulation.

6.2.3 Étape 3 : Simulation

Objectif : Simuler une partie aléatoire à partir du meilleur enfant inclut le nouveau noeud ajouté pour estimer sa valeur.

Comment :

- On joue une partie aléatoire à partir de l'état du jeu représenté par le nouveau nœud en choisissant un coup aléatoire pour chacun des joueurs à chaque tour jusqu'a atteindre un état final.
- La simulation se termine lorsque la partie est finie (victoire, défaite ou match nul) où la grille est entièrement remplie.
- On enregistre le résultat de la simulation (1 pour une victoire, -1 pour une défaite).

6.2.4 Étape 4 : Rétropropagation

Objectif : Mettre à jour les statistiques des nœuds visités pendant la sélection avec le résultat de la simulation.

Comment :

- l'aglorithme remonte l'arbre depuis le nœud simulé jusqu'à la racine.
- Pour chaque nœud visité, met à jour :

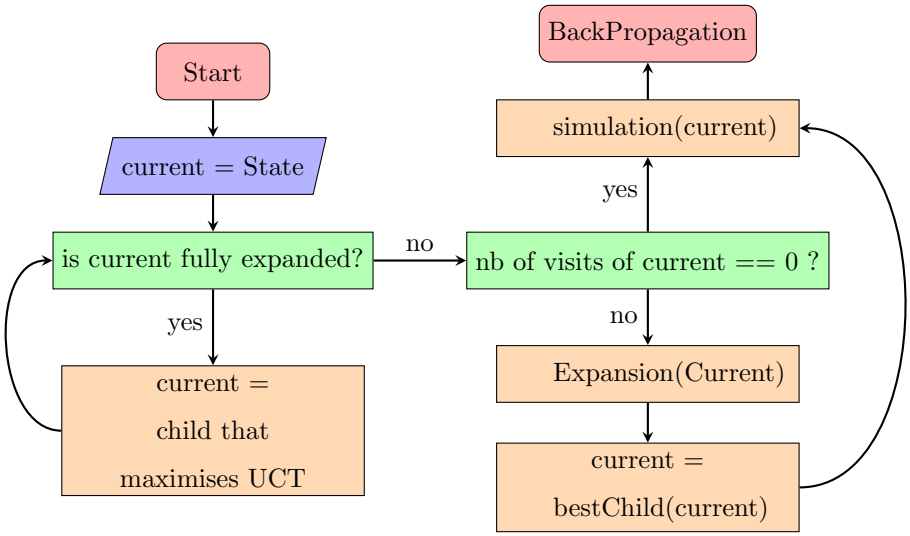
– Le nombre de visites.

– Le nombre de victoires.

– Le nombre de défaites.

6.2.5 Algorithme

La figure ci-dessous Illustre le fonctionnement des 4 étapes de l'algorithme MCTS pour un état donné.



6.3 MCTS dans le jeu de Hex et Complexité

MCTS est particulièrement efficace pour le jeu de Hex car pour chaque état on a un très grand nombre de coups possibles, et cet algorithme explore de manière intelligente en se concentrant sur les coups prometteurs. Ainsi qu'on a pas besoin d'évaluation heuristique pour estimer la valeur d'un coup contrairement à d'autres algorithmes, car MCTS repose sur des simulations aléatoires pour estimer cette valeur.

Dans notre implémentation du jeu avec l'algorithme, chaque joueur a un arbre de recherche propre à lui pour séparer les résponsabilités et avoir des estimations plus précises selon le joueur, avantages d'utiliser deux arbres de recherche pour chaque joueur :

- Indépendance : Chaque joueur a son propre arbre, ce qui permet une exploration indépendante.
- Flexibilité : facilité d'attribuer des budgets de calcul différents à chaque joueur.
- Simplicité : Chaque instance de MCTS est responsable d'un seul joueur, ce qui simplifie la logique.

Ainsi que pour question d'éfficaité, seulement le sous arbre enraciné au noeud selectionné à la fin est consérvé à la fin de l'execution de l'algorithme, au lieu de reconstruire un arbre à chaque fois et perdre le resultat des Iterations précédantes dans les tours précédant réduisant ansi la charge sur la mémoire à chaque itération et assurant la persistance de l'arbre de recherche à chaque tour pour chacun des joueurs.

6.4 RAVE optimisation

6.4.1 Explication

RAVE (Rapid Action Value Estimation) est une extension de l'algorithme MCTS (Monte Carlo Tree Search) qui vise à accélérer la convergence des estimations de valeur des actions en exploitant des informations supplémentaires provenant des simulations.
RAVE estime que :

- la valeur d'une action est similaire dans tous les sous-états du sous-arbre du noeud selectionné.
- Chaque coup joué dans une simulation est traité comme si c'est la première fois où ce coup a été joué, et sa contribution au résultat de la simulation est enregistrée, et le résultat est utilisé comme généralisation de la valeur de ce coup dans l'ensemble du sous arbre commençant au parent du noeud actuel.

Contrairement à MCTS classique, qui utilise le résultat de la simulation pour mettre à jour les valeurs seulement des noeuds séléctionnés lors de la phase de selection (l'ensemble des noeuds constituant le chemin du noeud séléctionné remontant jusqu'a la racine),

RAVE enregistre toutes les actions jouées pendant la phase de simulation et À la fin de la simulation, le résultat (victoire/défaite) est attribué à tous les coups qui ont été joués pendant cette simulation, pas seulement au premier coup.

Ce principe permet à l'algorithme de recueillir des statistiques sur les coups, même s'ils ont été joués dans des contextes différents ou à des profondeurs différentes, ainsi qu'une valeur approximative de chaque coup dans l'arbre de recherche après seulement quelques Itérations.

Pour effectuer cela, chaque noeud stocke deux sets de statistiques :

- **Statistiques classiques** : Nombre de visites et nombre de victoires, comme dans MCTS standard.
- **Statistiques RAVE** : Nombre de fois où l'action a été jouée dans une simulation (*RAVE visits*) et nombre de fois où elle a conduit à une victoire (*RAVE wins*).

La valeur RAVE d'une action est calculée comme suit :

$$\text{RAVE Value} = \frac{\text{RAVE Wins}}{\text{RAVE Visits}}$$

La valeur MCTS d'une action est calculée comme suit :

$$\text{MCTS Value} = \frac{\text{victoires}}{\text{défaites}}$$

6.4.2 Implémentation

Dans notre projet, nous avons implémenté RAVE en étendant la structure de nœuds de l'arbre MCTS pour inclure les statistiques RAVE.

La différence entre le MCTS et le RAVE repose sur trois points :

- **Séléction :** au lieu de choisir l'enfant qui maximise **UCT**, on choisi l'enfant qui maximise la valeur combinée Obtenue par la formule :

  $$\text{Combined Value} = (1 - \beta) \cdot \text{MCTS Value} + \beta \cdot \text{RAVE Value}$$

  où β est un paramètre qui contrôle l'importance relative de RAVE par rapport à MCTS. Pour donner plus de poids à MCTS à mesure que l'arbre se développe, nous faisons décroître β en fonction du nombre de visites RAVE du nœud :

  $$\beta = \frac{k}{k + n}$$

  où k est une constante et n est le nombre de visites RAVE du nœud.

- **Simulation :** pendant la phase de simulation toutes les actions jouées par le joueurOriginal sont enregistrées pour mise à jour avec le résultat de la simulation dans la phase suivante.

- **BackPropagation :** pendant la phase de backpropagation, l'algorithme parcours tout le sous-arbre en recherche de noeuds contenant les actions enregistrées, et mets à jour la valeur RAVE wins et RAVE visits de ces noeuds concérnés.

6.4.3 Avantages et Inconvénients

RAVE

- **Compléxité** : RAVE permet de trouver un meilleur coup en moins d'Itérations par rapport à MCTS mais il est plus lent par Itération.

- **Meilleure exploration** : En utilisant les statistiques RAVE, l'algorithme peut explorer des actions prometteuses plus tôt.

- **Efficacité dans les jeux à grande branchement** : RAVE est particulièrement utile dans les jeux où le nombre de coups possibles est élevé, comme le jeu de Hex.
- **Plus Lent en temps par Itération** : puisque RAVE doit maintenir deux statistiques différentes, et la mise à jour des statiqtiques RAVE nécessite un parcours de l'arbre à vhaque fin de simulation cela ajoute une compléxité dans le temps d'execution d'une Itération
- **Manque de précision** : puisque RAVE repose sur l'estimation et la généralisation de valeurs des coups n'importe le temps où le contexte cela peut conduire à des éstimations fausses car un coup effectif dans un contexte n'est pas forcément aussi effectif dans un different contexte.

MCTS

- **Précision** : MCTS repose sur la simulation directe d'un coup pour estimer sa valeur au lieu de généraliser sur un ensemble, ce qui mene à des valeurs beacoup plus précises.
- **Compléxite** : puisque MCTS repose sur la simulation directe, il a besoin d'un nombre elevé de simulations pour simuler plusieurs coups et atteindre un bon résultat, Mais il est plus rapide en temps par Itération par Rapport à RAVE.
- **Influence du budget d'Itérations** : le budget d'Itérations (nb d'itérations effectuées par tour) a le plus grand influence sur l'efficacité et la capabilité de MCTS à trouver de bons coups

6.4.4 Cas d'utilisation de RAVE

RAVE est particulièrement efficace dans les situations suivantes :

- **Jeux à grande branchement** : Lorsque le nombre de coups possibles est élevé, RAVE permet d'explorer plus rapidement les actions prometteuses.
- **Simulations courtes** : RAVE est utile lorsque les simulations sont courtes et que les résultats partiels peuvent fournir des informations précieuses.
- **Convergence rapide nécessaire** : lorsque il faut trouver un résultat dans un nombre limité d'itérations .

RAVE est particulièrement efficace dans les premiers tours du jeu où l'ensemble de coups possibles est très large, ce qui permet de trouver de bons coups rapidement, mais il devient graduellement moins éfficace au fil du jeu lorsque des coups plus forts et précis sont requis, et pour contourner cela on a utilisé la selection basée sur la valeur combinée de RAVE et MCTS.

6.4.5 Cas d'utilisation de MCTS

MCTS est particulièrement efficace Lorsque le nombre de coups possibles est bas, MCTS permet d'effectuer plus de simulations sur les noeuds et converger vers une valeur plus précise et un résultat plus précis, comme les derniers tours du jeu HEX. :

7 Expérimentations

Dans le cadre de notre projet, plusieurs expérimentations ont été menées pour explorer l'impact de deux facteurs clés sur la probabilité de victoire des joueurs dans le jeu de Hex :

- Le **budget d'itérations**, c'est-à-dire le nombre de calculs que chaque joueur peut effectuer avant de jouer son coup ;
- La **taille de la grille**, qui peut varier en fonction de la configuration du jeu.

Ces expérimentations ont opposé deux joueurs utilisant les stratégies **MCTS** et **RAVE**. Les configurations des parties changeaient à chaque itération.

Les paramètres suivants ont été utilisés et variant durant les expérimentations :

- **Taille de grille:** 5, 7, 8, 9, 11, 14 ;
- **Budget d'itérations des deux joueurs:** 100, 250, 500, 1000, 2000, 3000, 5000 ;
- **Stratégies des joueurs:** MCTS, RAVE ;
- **Nombre de parties par expérimentation:** 30 ;
- **Joueur qui commence:** Chaque configuration est expérimentée deux fois, une fois avec le joueur Bleu qui commence et une autre avec le joueur Rouge qui commence.

Une configuration d'expérimentation se définit par:

- La taille de la grille ;
- Le budget d'itérations du premier joueur ;
- Le budget d'itérations du deuxième joueur ;
- La stratégie du premier joueur ;
- La stratégie du deuxième joueur ;
- Le nombre de parties jouées.

7.1 Plan factoriel

Le **plan factoriel** utilisé pour cette expérimentation a permis de tester toutes les combinaisons des paramètres suivants :

- **Taille de la grille** ;
- **Budget d'itérations** des joueurs ;
- **Stratégie des joueurs**.

Ce plan nous a permis de réaliser une série d'expérimentations couvrant l'ensemble des configurations possibles. Nous avons également veillé à réaliser suffisamment de répétitions pour garantir la robustesse et la représentativité des résultats.

7.2 Logs des expérimentations

Les **résultats d'une expérimentation** incluent les informations suivantes :

- Les détails de la configuration ;
- Le nombre de parties gagnées par le premier joueur ;
- Le nombre de parties gagnées par le deuxième joueur ;
- Le pourcentage de victoire du premier joueur ;
- Le pourcentage de victoire du deuxième joueur.

Les logs suivants montrent les résultats des différentes expérimentations. Chaque entrée contient la configuration de l'expérience et les résultats détaillés des victoires de chaque joueur :

```
[CONFIG] GridSize=5 | BudgetJ1=250 | BudgetJ2=500 | StrategieJ1=MCTS | StrategieJ2=RAVE | Starting=ROUGE | nbGames=2
Bilan => J1=1 WIN | J2=1 WIN [Grid=5, b1=250, b2=500, StrategieJ1=MCTS, StrategieJ2=RAVE, start=ROUGE]

[CONFIG] GridSize=5 | BudgetJ1=250 | BudgetJ2=1000 | StrategieJ1=MCTS | StrategieJ2=MCTS | Starting=ROUGE | nbGames=2
Bilan => J1=1 WIN | J2=1 WIN [Grid=5, b1=250, b2=1000, StrategieJ1=MCTS, StrategieJ2=MCTS, start=ROUGE]

[CONFIG] GridSize=5 | BudgetJ1=500 | BudgetJ2=500 | StrategieJ1=RAVE | StrategieJ2=RAVE | Starting=BLEU | nbGames=2
Bilan => J1=2 WIN | J2=0 WIN [Grid=5, b1=500, b2=500, StrategieJ1=RAVE, StrategieJ2=RAVE, start=BLEU]
```

Les logs ci-dessus montrent les configurations expérimentées et les résultats des victoires des joueurs. Notez que dans certaines expériences, les joueurs ont utilisé la même stratégie, telles que MCTS contre MCTS ou RAVE contre RAVE. Cependant, selon le plan factoriel, nous avons également testé différentes combinaisons des stratégies des joueurs et des budgets, afin de couvrir toutes les configurations possibles.

7.3 Analyse des résultats

Voici une analyse des résultats expérimentaux montrant l'influence du rapport de budget entre les deux joueurs sur leur probabilité de victoire en fonction de la taille de la grille.

Pour mieux comprendre les données, nous avons réalisé des **heatmaps** qui illustrent l'impact du budget d'itérations des joueurs en fonction de la taille de la grille sur leurs probabilités respectives de victoire. Chaque graphique représente la probabilité de victoire du joueur 1 (J1) en fonction des différents budgets des deux joueurs, tout en considérant que les deux joueurs utilisent la même stratégie (soit **MCTS**, soit **RAVE**). L'axe des ordonnées (Y) correspond au **budget du joueur 1 (J1)** et l'axe des abscisses (X) au **budget du joueur 2 (J2)**. La couleur dans chaque case de la heatmap représente le pourcentage de victoire du joueur 1 (J1), allant de bleu (faible victoire) à rouge (forte victoire).

7.4 Analyse des résultats

Voici une analyse des résultats expérimentaux montrant l'influence du rapport de budget entre les deux joueurs sur leur probabilité de victoire en fonction de la taille de la grille.

Pour mieux comprendre les données, nous avons réalisé des **heatmaps** qui illustrent l'impact du budget d'itérations des joueurs en fonction de la taille de la grille sur leurs probabilités respectives de victoire. Chaque graphique représente la probabilité de victoire du joueur 1 (J1) en fonction des différents budgets des deux joueurs, tout en considérant que les deux joueurs utilisent la même stratégie (soit **MCTS**, soit **RAVE**). L'axe des ordonnées (Y) correspond au **budget du joueur 1 (J1)** et l'axe des abscisses (X) au **budget du joueur 2 (J2)**. La couleur dans chaque case de la heatmap représente le pourcentage de victoire du joueur 1 (J1), allant de bleu (faible victoire) à rouge (forte victoire).

Analyse générale des résultats :

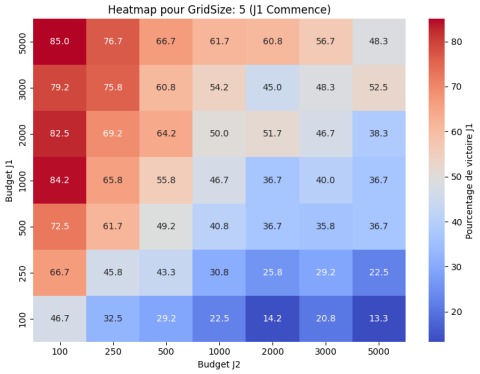
Les résultats montrent que l'augmentation du budget d'itérations a un impact significatif sur les probabilités de victoire des joueurs, en particulier pour les grilles de plus grande taille.

- Grille de taille 5 : Lorsque le joueur 1 commence, il bénéficie d'un avantage stratégique plus marqué avec un budget d'itérations supérieur à celui du joueur 2. À mesure que le budget de J1 augmente, ses chances de victoire augmentent aussi, notamment lorsque son budget dépasse celui de J2. Toutefois, lorsque le joueur 2 commence, l'avantage est inversé. J2, avec un budget plus élevé, a un pourcentage de victoire beaucoup plus élevé.

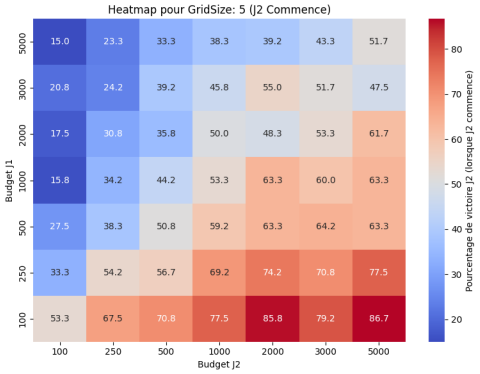
- Grille de taille 9 : Les résultats sont similaires, bien que l'impact du budget soit moins prononcé qu'avec la grille de taille 5. Le joueur 1 continue à bénéficier de l'avantage d'un budget supérieur, mais l'écart entre les pourcentages de victoire est moins marqué. Lorsqu'il commence, J1 a des chances de victoire élevées lorsque son budget dépasse celui de J2.

- Grille de taille 14 : Sur la grille de taille 14, l'écart de victoire entre les joueurs est beaucoup plus marqué, avec des victoires décisives lorsque le joueur avec un budget plus élevé commence. En revanche, si le joueur 2 commence, il obtient presque systématiquement des victoires, surtout lorsque son budget est élevé.

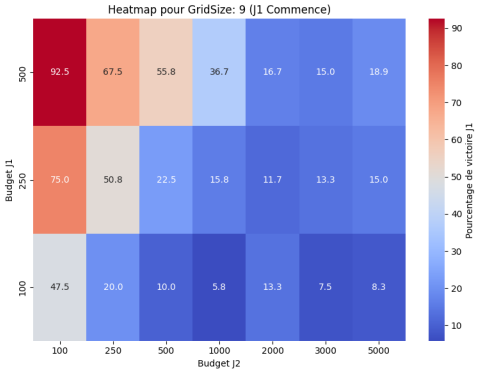
Remarque importante : Les graphes ci-dessous montrent les résultats pour des joueurs utilisant la même stratégie (soit **MCTS**, soit **RAVE**). Pour des résultats plus complets et diversifiés, y compris des configurations avec des stratégies différentes (RAVE contre MCTS), nous vous invitons à consulter le dossier *experimentation/graphs* pour d'autres expérimentations et graphiques.



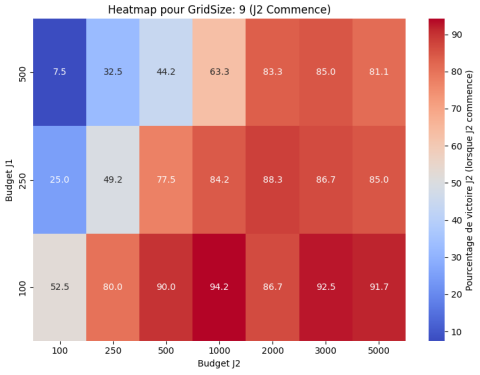
Heatmap pour GridSize = 5 (J1 commence). L'axe Y représente le budget de J1, l'axe X représente le budget de J2 et les couleurs montrent la probabilité de victoire de J1.



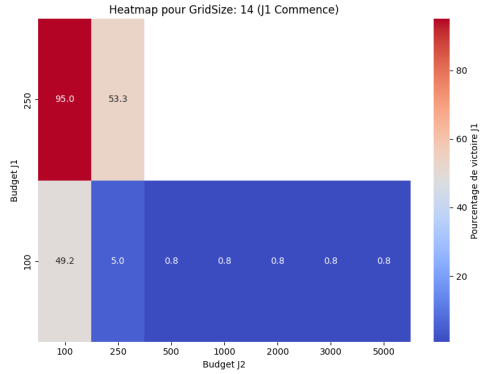
Heatmap pour GridSize = 5 (J2 commence). L'axe Y représente le budget de J1, l'axe X représente le budget de J2 et les couleurs montrent la probabilité de victoire de J2.



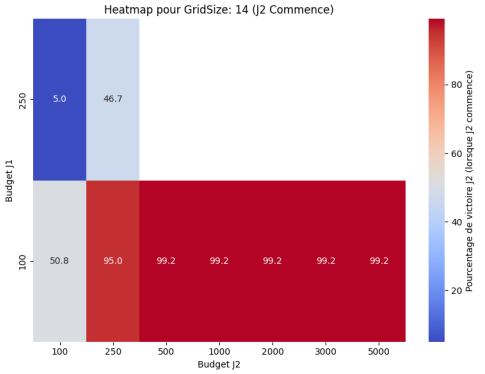
Heatmap pour GridSize = 9 (J1 commence). L'axe Y représente le budget de J1, l'axe X représente le budget de J2 et les couleurs montrent la probabilité de victoire de J1.



Heatmap pour GridSize = 9 (J2 commence). L'axe Y représente le budget de J1, l'axe X représente le budget de J2 et les couleurs montrent la probabilité de victoire de J2.



Heatmap pour GridSize = 14 (J1 commence). L'axe Y représente le budget de J1, l'axe X représente le budget de J2 et les couleurs montrent la probabilité de victoire de J1.



Heatmap pour GridSize = 14 (J2 commence). L'axe Y représente le budget de J1, l'axe X représente le budget de J2 et les couleurs montrent la probabilité de victoire de J2.

7.5 Conclusion préliminaire

L'analyse des heatmaps montre que l'influence du budget d'itérations est fortement liée à la taille de la grille. Sur les petites grilles, un budget plus élevé pour un joueur peut lui accorder un avantage, mais cet avantage devient beaucoup plus marqué sur les grandes grilles. En effet, sur la grille de taille 5, le joueur avec un budget plus élevé obtient un pourcentage de victoire plus élevé, surtout lorsqu'il commence. À partir de la grille de taille 9, cet avantage est encore plus perceptible, et sur la grille de taille 14, il devient décisif pour celui qui commence avec un budget plus élevé.

Les résultats suggèrent qu'un budget plus élevé donne un avantage stratégique important, en particulier pour les jeux avec une plus grande taille de grille. Cependant, des expérimentations supplémentaires avec des configurations plus variées, y compris des joueurs avec des stratégies différentes, peuvent fournir des informations encore plus détaillées.

8 Conclusion

Ce projet a permis d'étudier l'impact du budget d'itérations et de la taille de la grille sur les probabilités de victoire dans le jeu de Hex, en utilisant les algorithmes MCTS et RAVE. Nos expérimentations ont révélé que :

- Un budget d'itérations plus élevé donne un avantage stratégique, surtout sur les grandes grilles.
- La taille de la grille amplifie l'impact du budget d'itérations, particulièrement pour les grilles de grande taille.

- RAVE permet des résultats plus rapides, mais MCTS reste plus précis, surtout avec un budget élevé.

En conclusion, ce projet a montré que la gestion du budget et la taille de la grille sont cruciales pour l'efficacité des algorithmes dans un environnement complexe comme Hex. Des expérimentations futures avec des stratégies variées pourraient offrir des insights supplémentaires.