17/12/2025

Université Grenoble Alpes

Rapport de Projet – Implémentation et comparaison de stratégies d'intelligence artificielle pour le jeu 2048

UE Introduction à l'Intelligence Artificielle

L3 MIASHS

Rim GUERCHALI 12112185

German PARDO TERAN 12318655

Sommaire

Introduction

Le jeu 2048, créé par Gabriele Cirulli en 2014, est un puzzle numérique qui se joue sur une grille de 4×4 cases. Le joueur combine des tuiles (portant des puissances de 2) en les déplaçant dans quatre directions possibles (haut, bas, gauche, droite) (Fig. 1). Lorsque deux tuiles de même valeur se touchent lors d'un déplacement, elles fusionnent en une seule tuile dont la valeur est la somme des deux. Après chaque mouvement valide, une nouvelle tuile de valeur 2 (avec une probabilité de 90%) ou 4 (avec une probabilité de 10%) apparaît aléatoirement au tableau. Le but du jeu est d'atteindre la tuile 2048, mais on peut continuer à jouer. La partie se termine lorsqu'aucun mouvement n'est plus possible, c'est-à-dire, quand il n'y aura aucune case vide.

Ce projet s'inscrit dans le cadre de notre formation en Licence MIASHS et vise à implémenter le jeu 2048 en Java, puis à développer et comparer plusieurs stratégies algorithmiques capables de jouer automatiquement. L'objectif pédagogique est d'approfondir nos compétences en programmation orientée objet et en conception de systèmes de jeu ; et d'explorer différentes approches d'intelligence artificielle appliquées à la résolution de problèmes combinatoires.

Fig.1 : Un tableau typique du jeu 2048

Pour le choix des heuristiques, on s'est inspiré des articles qu'on a consultés pendant la revue de la littérature. Entre elles nous avons identifié les méthodes gloutonnes (greedy), les algorithmes de recherche adversariale (Minimax, Expectimax), et les méthodes de simulation Monte Carlo.

Pour l'implémentation, nous avons retenu six stratégies. La stratégie aléatoire sert de référence de base, effectuant des mouvements aléatoires à chaque coup, comme l'indique son nom. La stratégie Greedy évalue chaque coup isolément en choisissant celui qui maximise immédiatement le score. Nous avons également implémenté deux variantes heuristiques : une stratégie privilégiant le vidage du plateau (maximisation des cases vides) et une autre favorisant la monotonicité (organisation ordonnée des valeurs). Les stratégies Minimax et Expectimax explorent plusieurs coups à l'avance, et cette dernière est mieux adaptée au 2048 car elle prend en compte l'aspect aléatoire du jeu. Enfin, la méthode Monte Carlo utilise des simulations aléatoires pour estimer la qualité de chaque coup.

Travail réalisé

Gestion de Projet

Nous avons organisé le projet avec deux outils principaux. On a implémenté un échéancier sur Google Spreadsheets pour planifier les grandes étapes : implémentation du jeu, développement des stratégies, tests et analyse statistique. Et une to-do list sur un Document de Google pour suivre les tâches et faciliter la coordination entre membres de l'équipe.

Outils utilisés

Nous avons choisi Java comme langage de programmation et Eclipse comme environnement de développement, car notre formation MIASHS nous a déjà familiarisés avec ces outils. Cela nous a permis de nous concentrer mieux sur l'algorithmique du projet plutôt que sur l'apprentissage d'un nouveau langage.

Pour l'analyse des résultats, nous utilisons RStudio. Il nous a permis de réaliser des tests statistiques et de créer des graphiques comparatifs.

Claude, l'assistant IA, a servi d'outil de support pour le débogage et pour la création du fichier CSV du Benchmark. Il nous a aidés à identifier et corriger des erreurs dans notre code en cas de blocages.

Et enfin la rédaction du rapport se fait avec LaTeX, ce qui nous garantit une mise en page professionnelle et facilite la gestion des références et de la numérotation.

Programmation du jeu

Architecture générale

Le jeu repose sur trois classes principales : Carre, Tableau, et Jeu.

Classe Carre

La classe Carre représente une case de la grille. Chaque case stocke une valeur entière (0 pour vide, sinon une puissance de 2). Un booléen indique si la case a déjà fusionné durant le tour pour empêcher les fusions multiples. La méthode clone() crée des copies indépendantes, nécessaires pour simuler des coups sans modifier le jeu réel.

Classe Tableau

La classe Tableau gère la grille 4×4 et la logique des mouvements. Elle contient une matrice d'objets Carre, un score, et un générateur aléatoire. Les quatre méthodes haut(), bas(), gauche(), droite() déplacent les tuiles selon les règles. Elles fusionnent les tuiles identiques adjacentes. Puis elles ajoutent une nouvelle tuile (2 avec 90% de probabilité, 4 avec 10%) dans une case vide aléatoire. Chaque méthode retourne un booléen qui indique si le mouvement a modifié le plateau.

Un constructeur par copie profonde permet de dupliquer un tableau existant. La méthode hasLost() détecte la fin de partie en vérifiant qu'aucun mouvement n'est possible.

Classe Jeu

La classe Jeu gère une partie. Elle fonctionne en mode manuel ou automatique. En mode automatique, elle utilise une instance de l'interface Strategie. À chaque tour, elle appelle la méthode choisirCoup() de cette stratégie. Le patron Strategie permet de changer facilement d'IA sans modifier le code du jeu.

Programmation des stratégies

Interface Strategie

Toutes les stratégies implémentent l'interface Strategie. Elle définit une seule méthode : choisirCoup(Tableau t), qui prend en paramètre l'état actuel du plateau et retourne une direction (HAUT, BAS, GAUCHE, ou DROITE). Cela nous permet de tester différentes stratégies avec le même code de jeu.

Stratégie Aléatoire

La stratégie aléatoire sert de référence de base. Elle choisit une direction au hasard parmi les quatre possibles sans faire aucune analyse du plateau. Cette stratégie nous permet d'établir un score minimal de comparaison. Toute stratégie intelligente devrait surpasser largement les scores de cette stratégie.

Pour l'implémenter on utilise un générateur de nombres aléatoires. Un entier entre 0 et 3 est tiré, et ce nombre correspond à une des quatre directions. Cette stratégie est très rapide, mais elle effectue souvent des coups inutiles ou contre-productifs.

Stratégie Greedy

La stratégie Greedy évalue chaque coup individuellement. Pour chaque direction possible, elle simule le mouvement sur une copie du plateau, et ensuite elle calcule un score pour cette situation. Le coup qui donne le meilleur score immédiat est choisi.

Le critère principal de notre fonction d'évaluation est le score du jeu. Lorsqu'un mouvement fusionne des tuiles, il augmente le score. Greedy va choisir à chaque fois le mouvement qui va fusionner le plus de tuiles.

Le limite de cette approche est qu'elle ne regarde qu'un seul coup à l'avance.

Stratégie Vider

La stratégie Vider privilégie la maximisation des cases vides. L'heuristique repose sur le fait que, plus il y a de cases vides, plus le jeu peut continuer longtemps (Kohler et al., 2019).

Pour chaque direction possible, la stratégie simule le mouvement. Elle compte ensuite le nombre de cases vides résultantes, et elle choisit le coup qui laisse le plus d'espace libre. Un bonus supplémentaire est ajouté si le coup augmente aussi le score.

Cette approche fonctionne mieux que l'aléatoire car elle évite de remplir inutilement le plateau. Cependant, elle ne considère pas l'organisation des tuiles sur le plateau, et dans ce jeu c'est important que les grandes valeurs ne soient pas trop dispersées pour pouvoir faire des fusions plus efficacement.

Stratégie Monotonicité

La stratégie Monotonicité évalue l'organisation du plateau. Elle favorise les configurations où les valeurs sont ordonnées, par exemple, les grandes valeurs d'un côté et les petites de l'autre (Kohler et al., 2019).

La fonction d'évaluation calcule un *score de monotonicité*. Elle vérifie si les valeurs décroissent (ou croissent) de manière cohérente dans les lignes et colonnes. Un plateau parfaitement monotone a ses tuiles organisées en ordre décroissant depuis un coin. Par exemple, 512, 256, 128 et 64 dans une ligne, respectivement.

Cette fonction combine plusieurs critères. Le score de base du jeu reste important, et la monotonicité ajoute un bonus significatif. De plus, un bonus supplémentaire récompense le placement de la plus grande tuile dans un coin.

Cette organisation facilite les fusions successives, car les tuiles de valeur similaire restent proches.

Stratégie Minimax

Minimax est un algorithme de recherche adversariale qui anticipe plusieurs coups à l'avance en simulant les réponses possibles. Selon cette approche, le jeu 2048 est modélisé comme un affrontement entre deux joueurs. Le joueur MAX (nous) cherche à maximiser le score, et le joueur MIN (le jeu) ajoute des tuiles de manière à minimiser nos chances (Nie et al., n.d.).

L'algorithme fonctionne par récursion alternée. La phase de maximisation teste tous les coups possibles (HAUT, BAS, GAUCHE, DROITE), et pour chaque coup, elle appelle la phase de minimisation. Celle-ci simule l'ajout de tuiles dans toutes les cases vides, elle teste les valeurs 2 et 4 pour chaque position, et elle retourne le pire scénario pour nous. Enfin, la maximisation choisit alors le coup qui donne le meilleur résultat même dans ce pire cas.

Cette approche suppose que le jeu place toujours la tuile la plus gênante (alors qu'en réalité, l'apparition des tuiles est aléatoire). C'est une stratégie "pessimiste", elle joue le jeu de façon trop prudente. C'est donc dommage car elle évite des risques qui pourraient être bénéfiques.

La profondeur de recherche limite le calcul. Nous utilisons une profondeur de 3. Symbolisé, cela signifie :

1. notre coup,
2. ajout de tuile,
3. notre coup,
4. ajout de tuile,
5. notre coup.

Au-delà de cette profondeur, une fonction d'évaluation estime la qualité du plateau. Cette fonction combine le score, le nombre de cases vides, et la position des grandes valeurs.

Stratégie Expectimax

Expectimax améliore Minimax en prenant bien en compte l'aspect aléatoire du jeu. Au lieu d'une phase de minimisation, il utilise des nœuds de chance. Ces nœuds calculent une valeur espérée basée sur les probabilités réelles.

L'algorithme alterne entre deux types de nœuds. Les nœuds de maximisation testent tous les coups possibles et ils choisissent le meilleur. Les nœuds de chance simulent l'ajout de tuiles. Pour chaque case vide, ils calculent : probabilité 90% d'un 2 × valeur si 2 + probabilité 10% d'un 4 × valeur si 4. La moyenne de toutes ces possibilités donne la valeur espérée.

Cette modélisation est plus réaliste. Le jeu n'est pas un “adversaire intelligent”. Il place des tuiles aléatoirement selon des probabilités fixes, du coup Expectimax exploite cette connaissance. Il prend des risques calculés que Minimax éviterait, par exemple.

La fonction d'évaluation d'Expectimax est plus complexe. Elle considère le score de base, elle récompense fortement les cases vides, elle calcule la monotonicité du plateau, elle vérifie si la plus grande tuile est dans un coin, et elle pénalise la rugosité (grandes différences de score entre cases adjacentes).

Nous utilisons une profondeur de 4 pour cette stratégie. On peut se permettre une profondeur plus élevée car l'algorithme est légèrement plus rapide.

Stratégie Monte Carlo

Monte Carlo n’a rien à voir avec les stratégies précédentes. Au lieu de calculer tous les coups possibles, elle utilise des simulations aléatoires. Pour chaque direction, elle joue de nombreuses parties simulées et elle calcule le score moyen obtenu. Le coup qui donne le meilleur score moyen est choisi.

Voici comment ça marche. Pour un coup candidat (par exemple GAUCHE), le plateau est copié. Le coup est joué sur la copie. À partir de cette position, une partie complète est simulée en jouant aléatoirement, et le score final de cette simulation est enregistré. Dans les paramètres on a établit que ce processus se répète 100 fois. Ensuite la moyenne des ces 100 scores donne une estimation de la qualité du coup GAUCHE. Les trois autres directions sont évaluées de la même manière. Le coup avec la meilleure moyenne est retenu.

Cette méthode repose sur la loi des grands nombres. Avec suffisamment de simulations, la moyenne converge vers la vraie valeur espérée. Monte Carlo ne nécessite pas de fonction d'évaluation complexe car les simulations capturent naturellement les bons et mauvais aspects d'un coup.

On a donc établi 100 simulations par coup. Chaque simulation joue jusqu'à 20 mouvements (ou jusqu'à la défaite si elle arrive avant). Par contre, pour le Benchmark, .

L'inconvénient de cette stratégie est le temps de calcul. 100 simulations pour 4 directions représentent 400 parties simulées par coup réel. C'est pour cette raison qu'on a établi 5 mouvements par simulation au Benchmark car sinon son exécution prenait trop longtemps (plus d'une heure pour 100 parties du 2048 simulées).

Benchmark

Pour comparer objectivement les stratégies, nous avons développé, avec l'aide de l'IA de Claude, une classe Benchmark. Cette classe lance automatiquement 200 parties pour chaque stratégie (2 × 100) et enregistre les résultats dans un fichier CSV. Pour chaque partie, elle mesure le score final, le nombre de coups joués, la tuile maximale atteinte, et le temps d'exécution. La classe calcule ensuite des statistiques descriptives (moyenne, médiane, écart-type, quartiles) et affiche la répartition des tuiles maximales atteintes. Le fichier CSV généré (resultats_2048.csv) contient toutes les données brutes nécessaires pour l'analyse statistique sur RStudio. Cette approche automatisée garantit la reproductibilité des tests et facilite la comparaison rigoureuse des performances.

Analyse des Résultats

Le graphique suivant présente la performance de chaque stratégie en termes de tuile maximale atteinte au cours de 200 simulations :

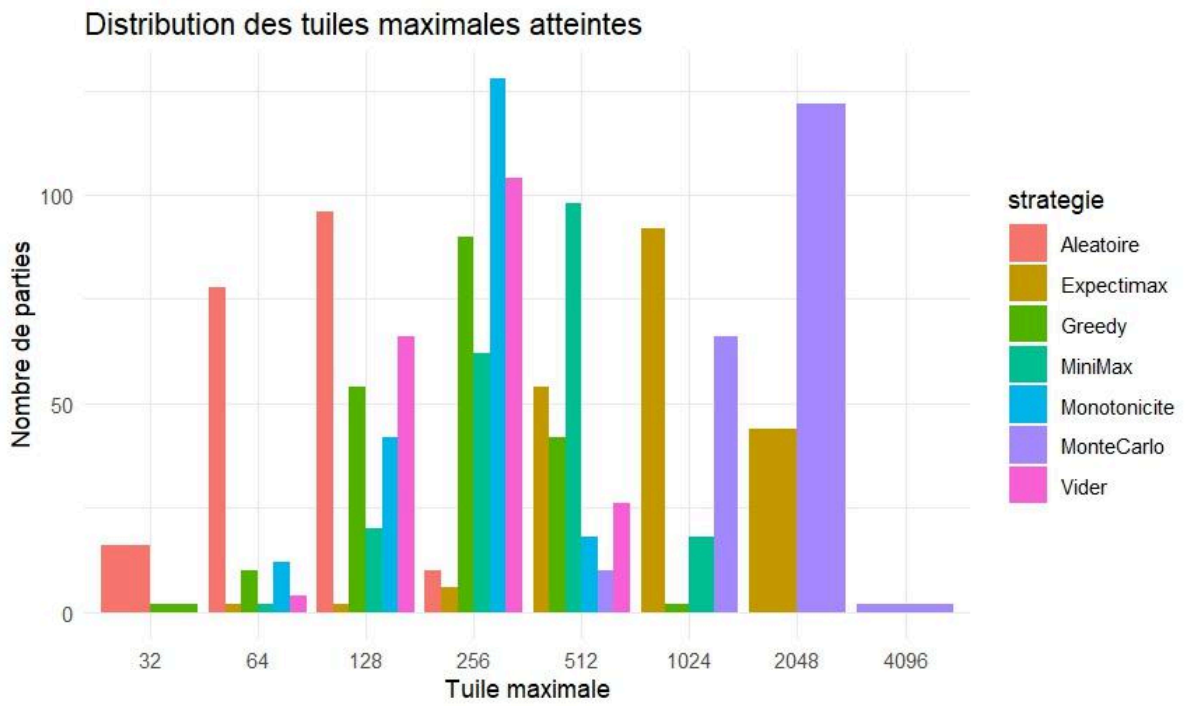

Fig. 2 : Performance des stratégies sur 200 parties

Voici un graphique qui compare le pourcentage de réussite des deux seules stratégies qui ont réussi à gagner le jeu (donc d'arriver à au moins la tuile maximale de 2048) :

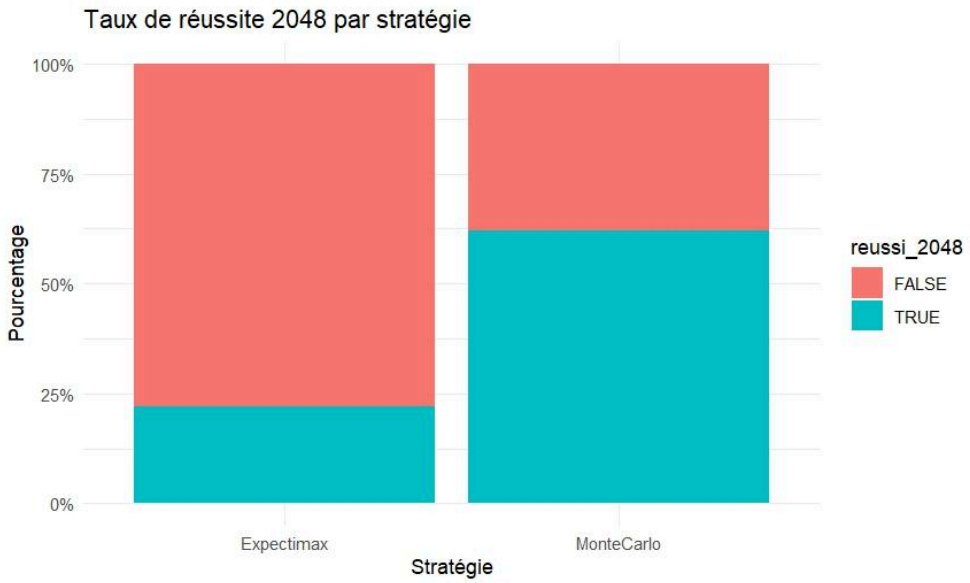

Fig. 3 : Graphique de comparaison de pourcentage de réussite

Les stratégies Expectimax et Monte Carlo démontrent une performance clairement supérieure, atteignant régulièrement la tuile 2048, et même 4096. Les autres stratégies montrent des performances plus limitées, avec des tuiles maximales strictement inférieures à 2048, c'est-à-dire, elles n'ont jamais atteint le but du jeu. Afin de pouvoir analyser plus en détail les performances et caractéristiques des différentes approches et faciliter l'analyse graphique, nous avons divisé les stratégies en deux groupes distincts. Groupe 1 constitué

par les stratégies Aléatoire, Greedy, Vider, Monotonicité et MiniMax, et Groupe 2 pour Expectimax et Monte Carlo.

Analyse du Groupe 1

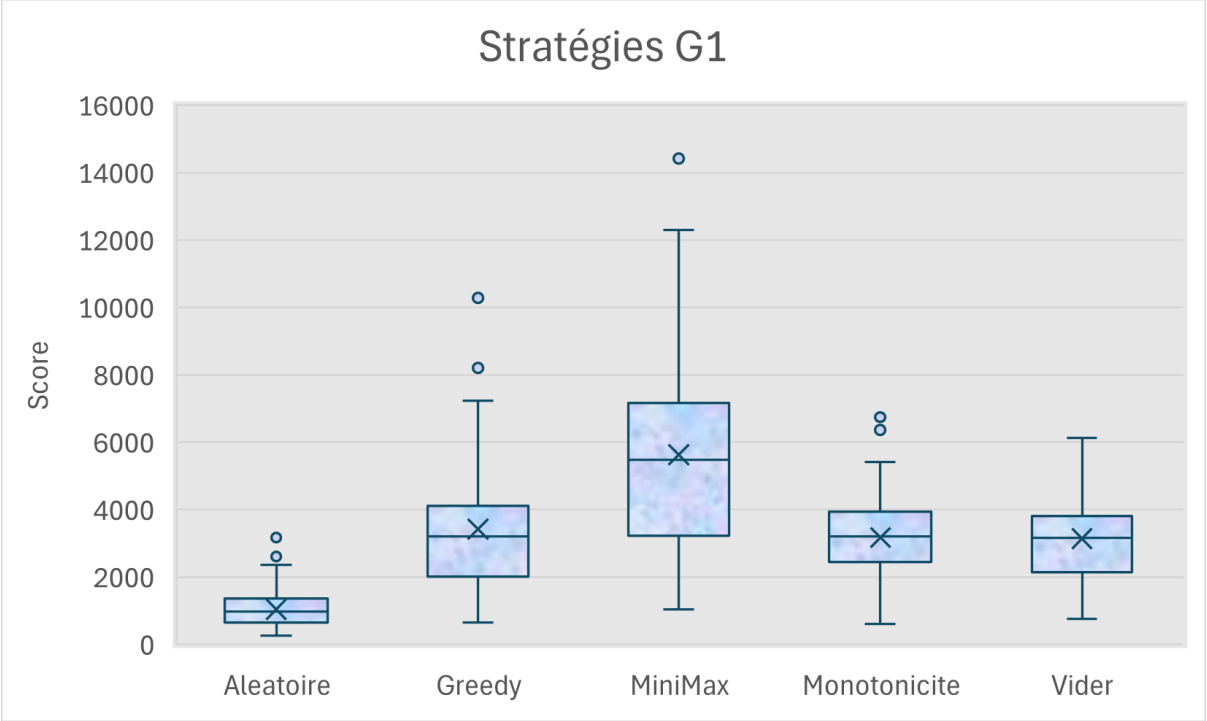

Fig. 4 : Distribution des scores par stratégie (Groupe 1)

Fig. 5 : Score en fonction du temps d'exécution (Groupe 1)

Les résultats du Groupe 1 montrent une hiérarchie claire. MiniMax obtient le meilleur score médian (5 484 points), suivi de Greedy (3 210 points), Monotonicité (3 210 points) et Vider (3 156 points). Et comme on se l'attendait, la stratégie Aléatoire obtient les plus faibles résultats (986 points).

Monotonicité présente la meilleure consistance avec un écart-type de 1 291 points. À l'inverse, Aléatoire montre une forte dispersion (écart-type de 509 points), ce qui reflète son imprévisibilité.

En termes de temps d'exécution, Aléatoire (1,67 ms) est la plus rapide, suivie de Vider (3,86 ms par partie), Greedy (7,71 ms) et Monotonicité (7,66 ms). MiniMax, malgré ses performances supérieures, nécessite 167,74 ms par partie, soit 43 fois plus que Vider.

Ces stratégies basiques restent limitées. Le score maximal observé pour MiniMax (troisième quartile) n'excède pas 7 159 points. Ces performances contrastent fortement avec celles du Groupe 2.

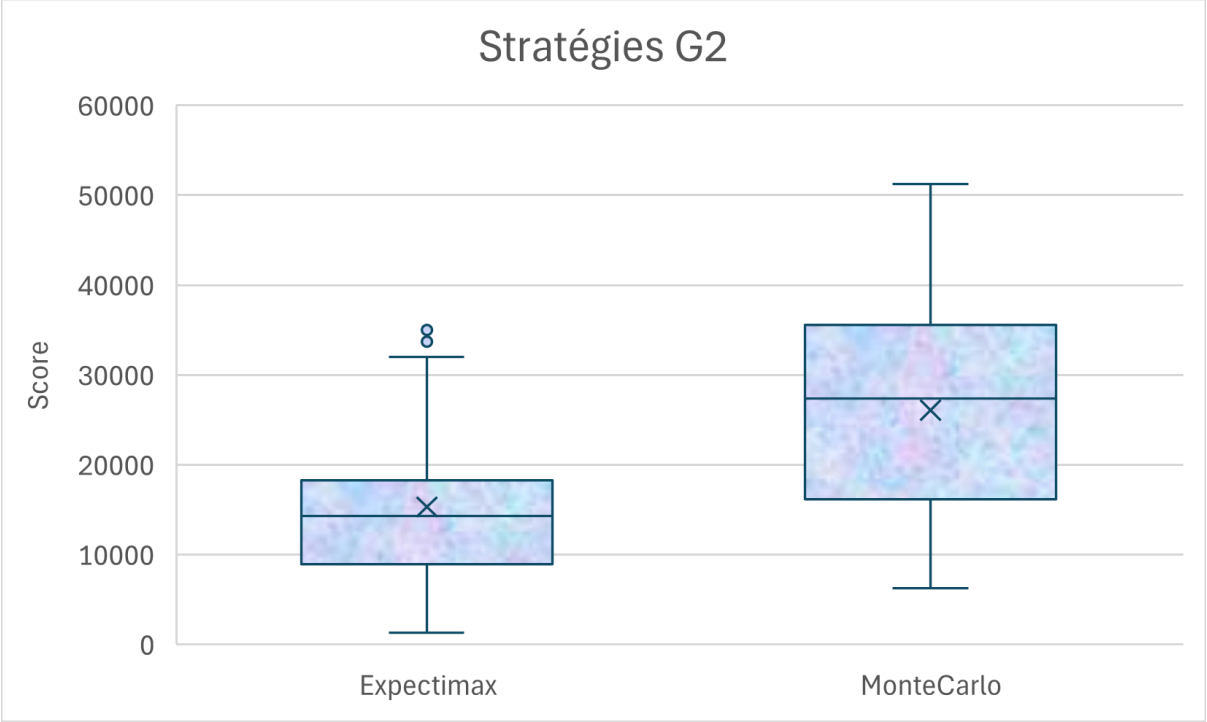

Fig. 6 : Distribution des scores par stratégie (Groupe 2)

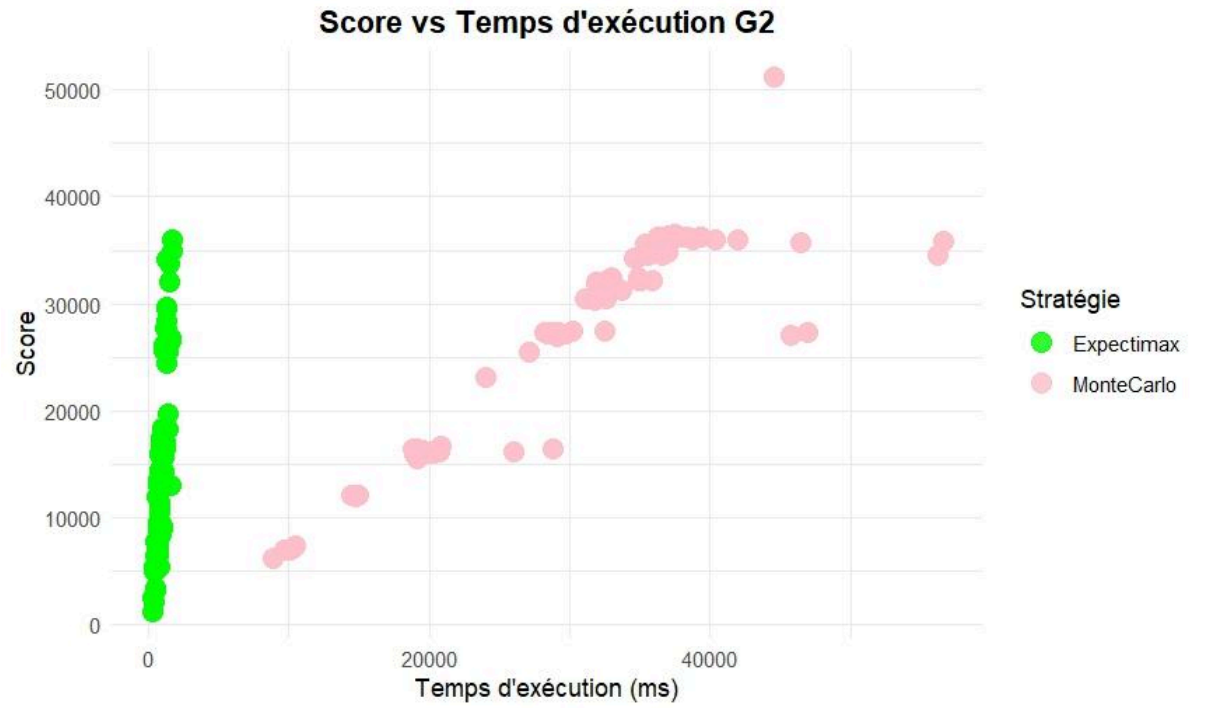

Fig. 7 : Score en fonction du temps d'exécution (Groupe 2)

Les résultats du Groupe 2 montrent une nette supériorité de Monte Carlo. Son score médian atteint 27 412 points, soit 91% de plus qu'Expectimax (14 326 points). Le troisième quartile de Monte Carlo

s'élève à 35 563 points. Cela signifie que 25% des parties dépassent ce seuil. Expectimax n'atteint pratiquement jamais ce niveau (Q3 à 18 252 points).

Cette belle performance a un coût très important. Monte Carlo nécessite en moyenne 29 231 ms par partie (environ 30 secondes), alors qu'Expectimax ne demande que 946 ms (environ 1 seconde). Expectimax génère 16,2 points/ms contre seulement 0,89 point/ms pour MonteCarlo. Expectimax est donc 18 fois plus efficace en termes de temps.

Le graphique score versus temps (Fig. 7) montre ce compromis. Chaque point supplémentaire obtenu par Monte Carlo demande beaucoup plus de temps. Monte Carlo privilégie la performance absolue, alors qu'Expectimax optimise le rapport performance-temps.

La supériorité de Monte Carlo s'explique par son approche stochastique. Ses simulations multiples explorent un espace de possibilités plus vaste. Cette méthode anticipe mieux les coups probabilistes du jeu. Expectimax reste limité par sa profondeur de recherche fixe et gère moins bien l'aléa du jeu.

Le choix entre ces deux stratégies dépend des contraintes. Pour des applications en temps réel, Expectimax offre un bon compromis. Pour maximiser la performance sans contrainte de temps, Monte Carlo reste la référence.

Limites et perspectives

La stratégie Monte Carlo aurait pu être optimisée par parallélisation des simulations. Cette technique aurait réduit le temps de calcul ou augmenté le nombre de simulations. Cependant, on n'a pas eu assez de temps pour explorer cette possibilité. Nous avons donc privilégié la bonne implémentation des stratégies.

Une approche hybride combinant Expectimax et Monte Carlo représente une solution prometteuse. Cette stratégie adaptative fonctionnerait en deux phases. En début de partie, Expectimax assure une progression rapide grâce à son efficacité (16,2 points/ms). Une fois un seuil atteint (tuile 512 ou 1024), le système bascule sur Monte Carlo. Cette transition permet d'optimiser les coups décisifs en fin de partie. L'approche hybride réduit le temps moyen tout en préservant les hautes performances. Elle devient ainsi réaliste pour des applications en temps quasi-réel.

Ces résultats illustrent un phénomène fondamental : la loi des rendements décroissants computationnels. Au-delà d'un certain seuil, l'effort supplémentaire devient disproportionné par rapport au gain. Monte Carlo nécessite 29 231 ms pour 26 079 points. Chaque point additionnel exige un investissement exponentiel en temps.

Cette observation reflète une vérité plus large en optimisation algorithmique : la recherche de la perfection absolue est souvent économiquement et computationnellement irrationnelle. Dans de nombreux systèmes complexes, y compris les jeux stratégiques, il existe un point optimal où le ratio performance-coût atteint son maximum avant de décliner. Cette frontière, visible dans le graphique score-temps, représente le seuil au-delà duquel la quête de performance marginale cesse d'être justifiée par les ressources engagées.

Bibliographie

1. Kohler, I., Migler, T., & Khosmood, F. (2019). Composition of basic heuristics for the game 2048. *Proceedings of Foundations of Digital Games (FDG 2019)*. ACM. https://doi.org/10.1145/3337722.3341838

2. Nie, Y., Hou, W., & An, Y. (n.d.). AI plays 2048. [Rapport de projet non publié].

3. Rodgers, P., & Levine, J. (2014). An investigation into 2048 AI strategies. *2014 IEEE Conference on Computational Intelligence and Games*, 1-2. https://doi.org/10.1109/CIG.2014.6932920

Outils logiciels

1. Anthropic. (2025). *Claude (Version Sonnet 4)* [Assistant IA]. Utilisé pour le débogage, la vérification du code Java et les spécificités de la classe Benchmark. https://claude.ai

Annexes

1. Graphes R (Instructions pour exécuter ce code sur RStudio : Importer le fichier AnalyseResultats depuis R avec le bouton ImportDatabase)

```
library(ggplot2)
library(dplyr)

#Distribution des tuiles maximales atteintes
ggplot(resultats, aes(x = factor(tuile_max), fill = strategie)) +
  geom_bar(position = "dodge") +
  labs(
    title = "Distribution des tuiles maximales atteintes",
    x = "Tuile maximale",
    y = "Nombre de parties"
  ) +
```

```r
  theme_minimal()

# Pourcentage de parties  G2 atteignant 2048

reussite <- resultats_seulement_em %>%
  mutate(reussi_2048 = tuile_max >= 2048) %>%
  group_by(strategie, reussi_2048) %>%
  summarise(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)
reussite <- resultats_seulement_em %>%
  mutate(reussi_2048 = tuile_max >= 2048) %>%
  group_by(strategie, reussi_2048) %>%
  summarise(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)

ggplot(reussite, aes(x = strategie, y = percentage, fill = reussi_2048)) +
  geom_bar(stat = "identity", position = "fill") +
  labs(title = "Taux de réussite 2048 par stratégie",
       x = "Stratégie", y = "Pourcentage") +
  scale_y_continuous(labels = scales::percent) +
  theme_minimal()



# Filtrer juste par G1
resultats_sans_em <- resultats %>%
  filter(!strategie %in% c("Expectimax", "MonteCarlo"))

# Graphique G1
ggplot(resultats_sans_em, aes(x = temps_ms, y = score, color = strategie)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(
    title = "Score vs Temps d'exécution (Groupe 1)",
    x = "Temps d'exécution (ms)",
    y = "Score obtenu",
    color = "Stratégie"
  ) +
  scale_color_brewer(palette = "Set2") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold"),
    legend.position = "right"
  )

# Filtrer juste G2
resultats_seulement_em <- resultats %>%
  filter(strategie %in% c("Expectimax", "MonteCarlo"))

# Graphique G2
ggplot(resultats_seulement_em, aes(x = temps_ms, y = score, color = strategie)) +
  geom_point(size = 4, alpha = 0.8) +
  labs(
    title = "Score vs Temps d'exécution G2",
    x = "Temps d'exécution (ms)",
    y = "Score",
    color = "Stratégie"
```

```
) +
scale_color_manual(values = c("Expectimax" = "green",
                              "MonteCarlo" = "pink")) +
theme_minimal() +
theme(
  plot.title = element_text(hjust = 0.5, face = "bold"),
  legend.position = "right"
)
```