INF8175 - Intelligence artificielle : méthodes et algorithmes

Automne 2024

Compte rendu projet Divercité

Équipe Poisson-Lanterne des Récifs coralliens

2147174 – Bakashov, Marsel

2117902 – Madzou, Prince

7 décembre 2024

Introduction

Ce compte rendu met en rétrospective l'évolution de notre agent intelligent jouant à Divercité. Ce jeu de société mettant en compétition deux adversaires était idéal pour mettre en pratique l'ensemble des concepts théoriques enseignés, tels que la recherche adversariale, les stratégies heuristiques et l'optimisation locale. Dans ce rapport, nous détaillons les étapes suivies pour le développement de notre agent, en commençant par l'analyse des règles du jeu et des défis associés, jusqu'à l'élaboration de stratégies basées sur des heuristiques et des algorithmes de recherche compétitifs. En complément, nous explorons les décisions conceptuelles et techniques qui ont guidé notre implémentation, ainsi que l'évaluation des performances de notre agent dans différents scénarios.

1. Méthodologie

Le développement de notre agent intelligent a suivi une approche itérative et progressive, intégrant progressivement des techniques d'intelligence artificielle afin d'améliorer ses performances.

1.1 Développement initial avec l'algorithme Minimax

La première étape a consisté à implémenter un agent basé sur l'algorithme Minimax classique. Cet algorithme explore l'arbre de jeu en considérant que chaque joueur agit de manière optimale pour maximiser (ou minimiser) ses gains. À ce stade, l'agent évaluait exhaustivement toutes les configurations possibles jusqu'à une profondeur définie, permettant de choisir les actions les plus favorables. Cependant, cette approche brute s'est rapidement révélée coûteuse en temps de calcul en raison de la taille de l'espace de recherche.

1.2 Optimisation avec l'élagage Alpha-Beta

Pour pallier les limites computationnelles du Minimax, nous avons introduit l'algorithme d'élagage Alpha-Beta. Cette optimisation permet de réduire significativement le nombre de nœuds évalués dans l'arbre de recherche en éliminant les branches inutiles, sans altérer la qualité des décisions prises par l'agent. Cette amélioration a permis une exploration plus efficace de l'arbre, augmentant ainsi la profondeur atteignable par la recherche.

1.3 Ajout d'heuristiques

A. Raison

Afin d'améliorer la prise de décision de l'agent, nous avons intégré des heuristiques personnalisées pour évaluer les états intermédiaires du jeu. Ces heuristiques reposent sur des principes propres au jeu Divercité, touchant notamment les cités, les ressources et les différents mouvements possibles. Nos heuristiques étaient basées sur un principe de bonus et de malus selon la qualité du prochain état. Enfin, pour garantir une évaluation cohérente des états de jeu, nous avons normalisé (tanh, sigmoid, min-max) les valeurs d'évaluation dans la plage $-1 < H(x) < 1$ (min-max scaling inclue 1 et -1). Les états terminaux sont également contraints à cette plage, ou la fonction d'utilité $-1 >= U(x) >= 1$. Cette normalisation permet de réaliser des combinaisons linéaire d'heuristique et d'y ajuster l'importance d'une heuristique grâce aux poids qu'on attribue. On évite aussi la surévaluation de certains coups et en assurant que les évaluations de l'heuristique restent dans les limites de la fonction d'utilité : $U(x\_min) <= H(x) <= U(x\_max)$. Cela permet de privilégier les états gagnants plutôt que ceux simplement bien évalués.

B. **Les heuristiques utilisées**

Divercite Heuristic

Placement des ressources: Un des critères clés entrant dans le calcul des bonus et des malus était le placement stratégique des ressources. Dans le jeu Divercité, les ressources placées sur le plateau profitent simultanément à toutes les cités environnantes, quel que soit leur propriétaire. Ainsi, il est crucial pour l'agent de maximiser les avantages pour ses propres cités tout en minimisant ceux de l'adversaire. Voici les facteurs pris en compte : Les cités alliées bénéficient d'un bonus proportionnel au nombre de cités amies. Les cités adverses subissent des pénalités si elles exploitent la même ressource ou si l'action les rapproche d'une configuration de Divercité. Les

positions stratégiques mal exploitées, comme celles situées dans des coins, pénalisent les ressources qui profitent à l'adversaire. Enfin, des pénalités sont appliquées aux placements sans impact sur les cités.

Placement des cités: Le critère des placements des cités est semblable à celui du placement des ressources, afin d'optimiser leur positionnement autour de ressources. Voici les facteurs pris en compte :Un bonus significatif est accordé aux cités formant une configuration Divercité, et un bonus modéré pour celles entourées de ressources de leur couleur. L'agent pénalise les placements manquant de ressources nécessaires pour une Divercité et applique une pénalité élevée aux cités placées dans des zones sans ressources proches, surtout si d'autres pièces sont déjà présentes.

Potentiel de blocage des cités adverses
Ce calcul a été développé pour évaluer l'efficacité des coups visant à bloquer les cités adverses, notamment celles proches de former une configuration Divercité. Si la couleur bloquante (une couleur autre que celle de la cité) permettait de bloquer une Divercité adverse, un bonus significatif était accordé. Lorsque seulement deux ou trois ressources entourent une cité adverse, l'agent favorise des coups qui perturbent le potentiel de Divercité, même si celle-ci n'est pas imminente.

Potentiel de divercité: Le calcul du score Divercité évalue la capacité de l'agent à maximiser ses points en formant des configurations Divercité ou en optimisant les cités selon les ressources adjacentes. Un bonus élevé est attribué lorsque quatre ressources de couleurs différentes entourent une cité, représentant une configuration optimale. Les cités entourées de ressources de leur propre couleur ou d'un nombre réduit de couleurs uniques reçoivent un score ajusté en fonction de leur potentiel à compléter une Divercité. L'agent anticipe la complétion de la Divercité en identifiant les couleurs manquantes et leur disponibilité parmi ses ressources restantes. Une pénalité est appliquée si une action visant à compléter une Divercité profite également aux cités adverses, réduisant ainsi l'efficacité.

Score final: L'évaluation heuristique finale combine plusieurs scores partiels afin de guider l'agent vers des décisions stratégiquement optimales à chaque étape du jeu. Ces scores partiels reflètent différents aspects importants du jeu Divercité, et leur pondération évolue dynamiquement en fonction de la progression de la partie. En plus des critères mentionnés auparavant, on y ajouté la rareté des ressources et le score du joueur. De plus, une pondération dynamique était appliqué à certains critères afin de varier le score selon l'avancement de la partie. Par exemple, en début de partie, l'accent est mis sur l'accumulation de Divercité et une gestion prudente des ressources.

$$Score\ final = w_{divercité} \cdot Score\ Divercité + w_{blocage} \cdot Score\ Blocage + Score\ Ressource + Score\ cité + w_{divercité} \cdot Score\ Divercité + w_{raret.=.} \cdot Score\ Rareté$$

$$avec\ w : poids\ ajustés\ dynamiquement\ selon\ l'avancement\ de\ la\ partie$$

Score Heuristic:

L'heuristique du score vise à évaluer l'état du jeu en se basant sur la comparaison entre le score du joueur et celui de son adversaire.

Piece Variance Heuristic:

L'heuristique de la variance des pièces d'un joueur permet de mesurer la flexibilité stratégique offerte par la diversité des couleurs et types de pièces à disposition. L'objectif est de minimiser cette variance, ce qui encourage un comportement optimal consistant à potentiellement utiliser une couleur différente à chaque tour. Par défaut, un joueur dispose généralement de plus de ressources que de cités. Pour équilibrer cette asymétrie, une "pièce" est virtuellement ajoutée à chaque cité de couleur distincte. Une variance nulle est pénalisée, car cela entraînerait l'épuisement successif d'un type de pièce. Par ailleurs, cette méthode permet également de

pénaliser les configurations avec un nombre insuffisant de cités. L'algorithme calcule la variance des cités et des ressources, puis applique une fonction sigmoïde pour chaque variance pour en faire une combinaison linéaire équilibrée.

Control Index Heuristic:

L'heuristique de l'indice de contrôle du plateau permet d'évaluer l'influence qu'un joueur exerce en plaçant ses cités sur le plateau. En d'autres termes, des tests manuels ont révélé que certaines positions permettent davantage de créer, bloquer ou voler des points, ou encore de diversifier les opportunités. Pour quantifier cette influence, les positions centrales du plateau, entourées de 8 cases potentielles, sont considérées comme stratégiquement avantageuses. À l'inverse, les positions situées dans les coins n'offrent que 3 cases adjacentes, tandis que les positions intermédiaires disposent de 5 cases potentielles. L'algorithme évalue également les cités réellement présentes autour d'une position donnée, en tenant compte des diagonales et des alignements horizontaux et verticaux. Les configurations sont hiérarchisées en fonction de deux critères principaux : la couleur et la propriété des cités environnantes. Les cas prioritaires en ordre sont :Différente couleur et différent propriétaire,Même couleur et différent propriétaire,Différente couleur et même propriétaire,Même couleur et même propriétaire. Enfin, une pondération accrue est attribuée aux cités situées horizontalement que celle diagonale par rapport à la position analysée, reflétant leur importance stratégique.

1.4 Table de transposition et symétrie.

Pour optimiser nos recherches de coups et éviter les calculs redondants, nous avons implémenté une table de transposition, ou cache, dans chaque recherche. Cela permet de réaliser un élagage d'arbre complet, réduisant ainsi l'espace de recherche. De plus, certains états du jeu sont similaires à d'autres lorsqu'on fait une rotation du plateau, ce qui contribue également à l'élagage des arbres de recherche.

1.5 Encore plus complexe ?

Outre le Minimax classique (Type A), des stratégies avancées ont aussi été implémentées afin de tenter d'optimiser l'exploration de l'espace de recherche. Le Minimax Type B ordonne les voisins d'un état à l'aide d'une heuristique, maximisant ainsi l'efficacité de l'élagage alpha-bêta et sélectionne ensuite les *n* meilleurs voisin, pour une exploration plus profonde, La recherche peut également être interrompue si l'heuristique retourne une valeur inférieure à un seuil défini, limitant les calculs inutiles. Enfin, pour éviter de surestimer une action dite favorable, une profondeur supplémentaire est explorée au tour de l'adversaire, ajustant les scores heuristiques de manière proactive. Ces approches améliorent la précision et l'efficacité des décisions. Ensuite, nous avons adopté la stratégie de recherche MCTS (Monte Carlo Tree Search),qui sélectionne des actions lors des simulations soit aléatoirement, soit en utilisant une distribution normale, où la variance est calculée à partir de la moyenne des évaluations du voisinage de l'état. Cette approche évalue l'importance d'un coup en se basant sur la métrique UCB (Upper Confidence Bound. Par la suite, pour améliorer le MCTS classique et éviter les mauvaises évaluations d'états, telles que les pièges ou les coups apparemment favorables, nous avons intégré, lors de l'évaluation des nœuds dans la phase d'expansion, une recherche Minimax Type A ou préférablement Type B, qui explore plus profondément. Cette valeur est incorporée dans le calcul de l'UCB, ajustant la formule comme suit :

$$UCB = [(1 - \alpha) * (n\_valeur / n\_visit) + \alpha * h(x)] + C * \sqrt{(ln(n\_parent\_visit) / n\_visit)}$$ [2]

Où, **C** représente le poids donné à l'exploration par rapport à l'exploitation, **h(x)** est la valeur issue de la recherche Minimax, et **α** détermine l'importance relative de cette recherche dans l'évaluation. Cette approche combine les forces du MCTS et du Minimax pour une prise de décision plus robuste et précise. Finalement, nous avons aussi exploré une hybridation entre le Minimax et les playouts (simulations jusqu'à la fin de la partie) du MCTS [1]. À chaque état terminal atteint par la recherche, n playouts sont donc

effectués. Lors de ces simulations, les actions sont choisies de manière stochastique,comme pour notre MCTS, selon une distribution (normale, uniforme ou aléatoire simple), ce qui permet d'explorer une plus grande diversité d'états. L'évaluation du nœud feuille dans la recherche principale Minimax est alors calculée comme suit : $1/n \ * \ \Sigma \ if \ win: \ +1 \ else \ -1$. Cette méthode fournit une évaluation probabiliste des chances de gagner la partie associées à une action depuis un état terminal donné, augmentant ainsi la précision et la robustesse de l'évaluation des décisions prises lors de la recherche.

2. Résultats et évolution de l'agent

2.1 Évolution de l'agent

Comme mentionné dans la section de Méthodologie, on a suivi une méthode itérative et progressive en se basant sur les aspects vus en cours.

L'implémentation initiale de l'agent reposait sur l'algorithme Minimax, permettant d'évaluer les actions possibles jusqu'à une profondeur fixe. Bien que fonctionnel, cet algorithme s'est avéré inefficace face à la complexité croissante de l'arbre de jeu, limitant sa capacité à explorer les scénarios en profondeur.

Pour optimiser les performances de recherche, l'élagage Alpha-Beta a été intégré. Cette technique a réduit drastiquement le nombre de nœuds à explorer, permettant à l'agent d'approfondir son analyse tout en respectant les contraintes de temps. Nous avons également intégré une table de transposition ainsi qu'une détection des états symétriques pour optimiser le processus de recherche.

Étant donné que le facteur de branchement est très élevé en début de partie, nous avons ajouté deux coups d'ouverture pour économiser du temps de calcul. Ces coups tentent contrôler les positions clés (celle du milieu) du plateau et à se rapprocher des positions importantes de l'adversaire, si elles existent. De plus, si une ressource est jouée au centre, l'algorithme tente de jouer la même couleur pour maximiser l'influence sur cette zone stratégique.

En raison de la complexité de nos stratégies de recherche, nous avons implémenté un *fail-safe move* qui sélectionne le coup offrant le plus grand nombre de points en cas de détection d'une situation incertaine.

Pour pallier l'impossibilité d'une recherche exhaustive, nous avons mis en place plusieurs heuristiques permettant d'estimer la valeur d'un état. Initialement, une heuristique a été implémentée pour identifier l'action présentant la plus grande différence de points, avec des recherches effectuées à profondeur constante. Cependant, nous avons constaté que le facteur de branchement diminuait considérablement à mesure que le jeu avançait. En réponse, nous avons instauré une recherche à profondeur variable, où la profondeur de recherche est augmentée lorsque ce facteur devient plus faible.La fonction d'utilité de base évaluait les états entre -1 et 1, où -1 correspondait à une défaite, 1 à une victoire, et 0 à un match nul. Toutefois, nous avons rapidement réalisé que cette approche était insuffisante, car elle permettait à la recherche Minimax de favoriser des actions qui menaient à une victoire, sans prendre en compte l'écart de score. Pour y remédier, la fonction d'utilité a été modifiée pour refléter la différence de score, ce qui empêche la surestimation tout en maximisant l'écart, augmentant ainsi les chances de victoire.Nous avons également introduit deux heuristiques supplémentaires : l'indice de contrôle et la variance des pièces. L'indice de contrôle mesure l'avantage stratégique lié à la prise de contrôle du plateau, en plaçant les pièces de manière à optimiser les possibilités d'attaque ou de défense. La variance des pièces évalue la diversité des coups possibles, en évitant de vider un type ou une couleur de pièces, ce qui enrichit les options stratégiques.

En parallèle, pour améliorer la qualité de l'estimation des états avec la recherche Minimax Alpha/Beta, nous avons constaté que la simple différence de points n'était pas suffisamment discriminante. En effet, plusieurs états pouvaient avoir la même différence de points, sous-estimant ainsi un état plus favorable. Nous avons donc développé plusieurs fonctions de perte pour caractériser quatre types d'estimations : état avant la recherche et à un nœud feuille de la recherche (la nôtre et celle de l'adversaire). Les types de fonction de perte sont: la différence d'état final, la *cross-différence*, le delta (différence entre l'état final et initial), etc. Ces fonctions visent à mieux évaluer un état en augmentant la variance des actions à explorer et en permettant l'élagage alpha-bêta pour réduire l'espace de recherche lorsque la meilleure estimation est trouvée en premier.

Ensuite, la prise en compte des spécificités de la **diversité** a nécessité l'intégration d'heuristiques précises, basées sur des critères de bonus et de malus lors de la création ou du blocage de diversité. Il était essentiel de trouver un équilibre entre les aspects défensifs, offensifs, l'agressivité et la réflexion sur les coups futurs pour assurer un comportement optimal. Pour ce faire, nous avons implémenté des **poids dynamiques** pour certains facteurs pris en compte dans le calcul du score final, notamment la rareté des ressources lors de la création ou du blocage de diversité. Il était crucial de préserver ces ressources en début de partie afin de ne pas limiter les possibilités de diversité futures.Cependant, un point faible majeur est apparu lors des confrontations contre des adversaires non-optimaux, c'est-à-dire ceux qui adoptent des stratégies non conventionnelles. Par exemple, notre agent rencontrait des difficultés face à des adversaires qui ignoraient complètement les diversités et se concentrait uniquement sur l'accumulation de points via des couleurs adjacentes aux cités homologues. Ainsi, une **combinaison linéaire** de toutes ces heuristiques nous a permis d'améliorer l'évaluation d'un état, en optimisant la prise de décision dans des situations variées. Finalement, nous avons opté uniquement pour des recherches **Minimax Type A**,portant le meilleur équilibre entre la profondeur et la qualité d'estimation, car les autres stratégies de recherche ne nous permettaient pas de gagner les parties.

2.2 Résultats

A. Objectifs d'évaluation
   En ce qui concerne la phase d'évaluation des résultats, nous nous sommes basés sur les performances de l'agent contre les agents de base. Parmi les agents de base, nous avions l'agent glouton, l'agent aléatoire et un agent dit "déviant", qui avait pour objectif de simuler un comportement non optimal, soit de focuser le blocage de divercité adverse. De plus, nous avons également pris en compte l'impact des heuristiques sur la qualité des décisions prises et nous avons testé la robustesse de l'agent face à différentes stratégies adverses et dans divers scénarios via la plateforme Abyss.
B. Méthodologie d'évaluation
   Nous avons donc testé l'agent contre les agents de base (agent aléatoire, agent glouton, agent déviant et les agents humains sur Abyss). Par ce fait, nous avons pu observer le comportement de notre agent face à diverses stratégies différentes et nous avons pu observer comment il agissait en présence de divers scénarios qui nous ont échappés. C'est notamment en testant contre les agents sur Abyss que nous avons pu voir que notre agent gaspillait rapidement ses ressources. Nous avons donc pu réagir en conséquence en ajoutant un facteur de rareté qui s'ajuste dynamiquement selon la progression de la partie. Les métriques d'évaluation incluent :

   - **Taux de victoire** contre différents adversaires.
   - **Différence de Score moyen** par partie.
   - **Temps moyen de calcul** par coup.
   - **Nombre de noeuds étendues**

C. Résultats observés
   Les résultats présentés caractérisent l'évolution de notre agent notamment pour deux versions majeurs :
   1. **Version prototype** (remise pour le concours).
   2. **Version finale** (intégrant toutes les améliorations).

   La version prototype ne battait pas systématiquement les agents de base. Elle n'avait pas été testée pour les deux configurations de départ (lorsque l'agent commence ou ne commence pas), ce qui a potentiellement biaisé certaines performances. Lors du concours, cette version a tout de même terminé 2e de son groupe, avec un score de 3 victoires - 1 défaite, démontrant une certaine efficacité malgré ses limites. Toutefois, ce n'était pas assez pour passer au tour suivant. De plus, cet version avait un taux de victoire positif avec 100 victoires et 68 défaites sur Abyss.

   La version finale a intégré les pondérations dynamiques et les améliorations des heuristiques, notamment pour mieux gérer la rareté des ressources et d'autres facteurs importants. Cette version a montré une meilleure capacité à s'adapter, notamment contre des agents humains, en optimisant ses Divercités tout en minimisant les opportunités adverses. Toutefois, cette version avait de la difficulté contre certains types d'agent déviant, surtout les agents ignorant complètement les divercités.

D. Performance

L'algorithme Minimax Type A a été privilégié pour son équilibre entre exploration et qualité des choix. En comparaison, le Minimax hybride avec MCTS étend davantage de nœuds au cours de la partie, tandis que les Minimax Type A et B étendent les nœuds de manière plus constante. Le Minimax Type A étend le moins de nœuds, mais donne les meilleurs scores. Comme le démontre la Figure 2. Pourquoi pas MCT?. MCTS nous a déçus, avec une limite de 45 secondes par coup. Le nombre de simulations reste relativement constant pour ces types de recherche (fluctue réellement beaucoup en début de partie pour chaque type), bien que le graphique montre une croissance exponentielle en fin de partie. De plus, un taux d'erreur élevé a été observé, comme le montre le temps de recherche dans la Figure 3.
Le choix d'une heuristique bien quantifiée est crucial : une heuristique permettant plusieurs possibilités donne des recherches plus approfondies, ce qui permet de mieux évaluer un coup. En revanche, une heuristique basée sur la différence de score produit des états similaires, ce qui rend la recherche moins exhaustive et affecte le temps de calcul dans la majorité des cas prouver à la figure 4 L'importance de la cache est évidente : comme le montre le graphique à la Figure 5, sans cache, nous étendons systématiquement plus de nœuds lors des recherches. En ajoutant la gestion des tables de transposition, le nombre de nœuds étendus diminue, suivant la tendance du facteur de branchement. Cependant, lorsque la symétrie n'est pas utilisée, le nombre de nœuds étendus est encore plus faible, ce qui peut s'expliquer par le fait que sans symétrie, le rejet des nœuds est déjà géré efficacement par l'algorithme alpha-bêta.

Nous avons testé différentes stratégies face à un agent greedy. La première stratégie, avec une profondeur constante, maximise la différence de points, mais l'écart reste proche de zéro, se terminant par une différence de 2 points. En ajustant la profondeur pendant le match, nous obtenons une amélioration modeste, atteignant une différence de 3 points et une meilleure évolution. La première version sur Abyss donne un ratio de victoires proche de 1. Modifier la fonction d’utilité pour se concentrer sur la différence de points a produit des résultats similaires à cette amélioration. Ajouter une combinaison linéaire avec les heuristiques de contrôle et de variance des pièces améliore encore les performances, réduisant le temps passé dans les scores négatifs. (Prototype Challonge) Ce prototype a permis de battre la majorité des agents sur Abyss, en évitant les vidanges rapides de certaines pièces. Grâce aux données collectées sur Abyss et Challonge, nous avons intégré des fonctions de perte efficaces tel que évolutif (maximisation de l’estimation et minimisation de celle de l’adversaire) ainsi qu’une fonction de potentiel (maximisation de l'écart final entre les joueurs et de la notre avant et après la recherche), ce qui a significativement amélioré nos résultats contre l'agent greedy. Enfin, la gestion de la diversité a optimisé la tendance de notre agent, réduisant les fluctuations négatives tout en accélérant son évolution vers une différence de plus de 15 points et qui nous donne le meilleurs ratio et elo (1336) qu’on a pu atteindre. (Figure 1)

3. Discussion

La version finale de notre agent intègre une combinaison linéaire d'heuristiques, mettant l'accent sur l'évolution du score et la divercité, avec des profondeurs variables, et utilise uniquement la recherche Minimax Type A, qui offre le meilleur compromis temps/qualité. Bien que nous n'ayons pas utilisé MCTS dans l'agent final, l'hybridation Minimax avec des playouts MCTS pourrait améliorer nos chances de victoire. Cette approche nous permettrait de choisir le coup qui maximise nos chances de gagner, malgré une évaluation minimale par Minimax. Cependant, cette méthode a été écartée, car elle pourrait entraîner des délais supplémentaires, notamment en lançant une seconde recherche pour un même coup, ce qui compromettrait le temps de calcul malgré une meilleure évaluation de l'espace de recherche.

Difficulté?

La plus grande difficulté a été de normaliser les heuristiques en choisissant la meilleure fonction de normalisation. Le min-max scaling est proportionnel, tandis que le *tanh* et le *sigmoid scaled* dans un domaine de -1 à 1. Une petite différence dans l'espace original de l'heuristique peut avoir un impact majeur ou minime dans le cas du tanh ou du sigmoid. Nous avons dû également testé toutes nos stratégies pour évaluer leur efficacité.

.Piste d'amelioration?

L'architecture proposée est très flexible, offrant la possibilité de combiner différentes stratégies de recherche et de créer des combinaisons linéaires d'heuristiques. Cela ouvre la voie à l'utilisation du machine learning ou du reinforcement learning, permettant d'ajuster dynamiquement les poids des heuristiques ou les types de recherche à certains moments clefs du jeu.

Références

[1] Vieira, M. A. A. D., Tavares, A. R., & Ribas, R. P. (2024). Hybrid Minimax-MCTS and Difficulty Adjustment for General Game Playing. *SBGames '23: Proceedings of the 22nd Brazilian Symposium on Games and Digital Entertainment*, 20-27. https://doi.org/10.1145/3631085.3631331.

[2] Lanctot, M., Winands, M. H. M., Pepels, T., & Sturtevant, N. R. (2014). Monte Carlo Tree Search with heuristic evaluations using implicit minimax backups. In *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games* (pp. 1-8). IEEE. https://doi.org/10.1109/CIG.2014.6932903.

Annexes

Figure 1. La difference score au cours du match

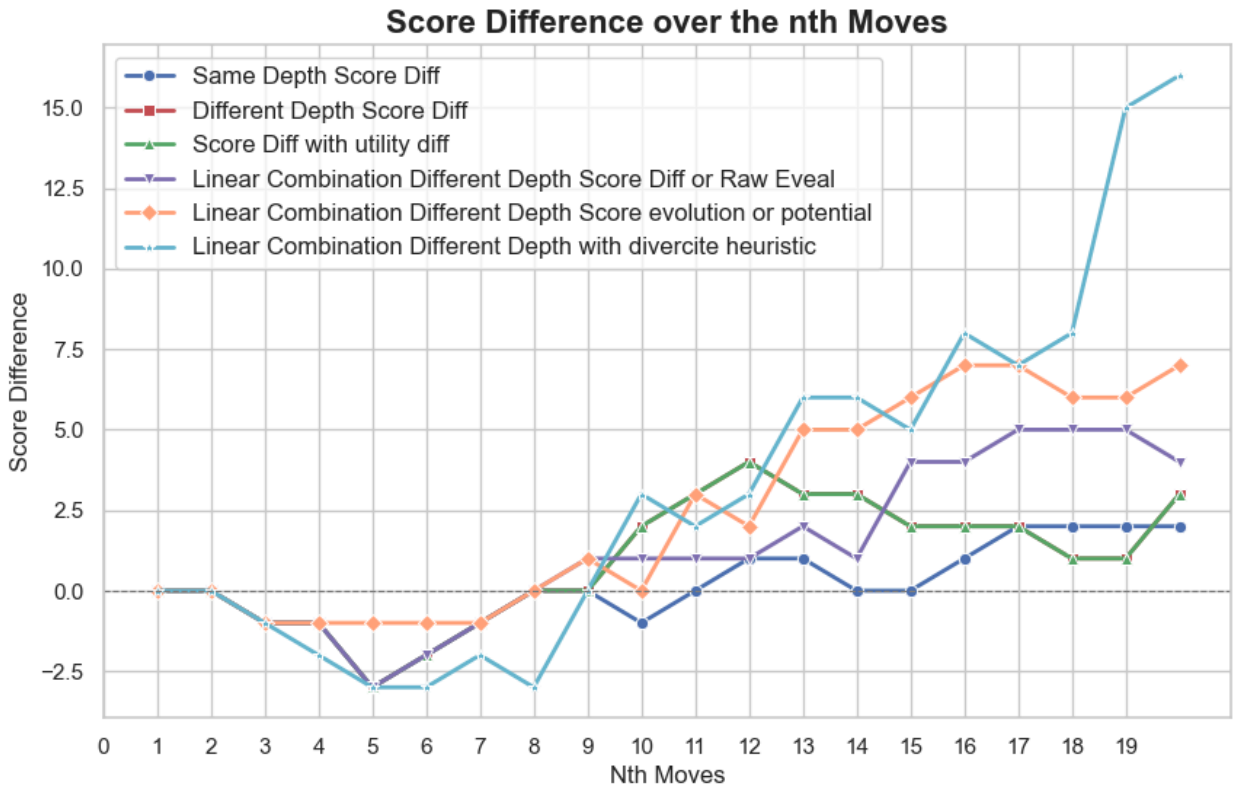


Figure 2. Le nombre de noeuds étendues et le temps de calcul au cours du match par algorithme Minimax

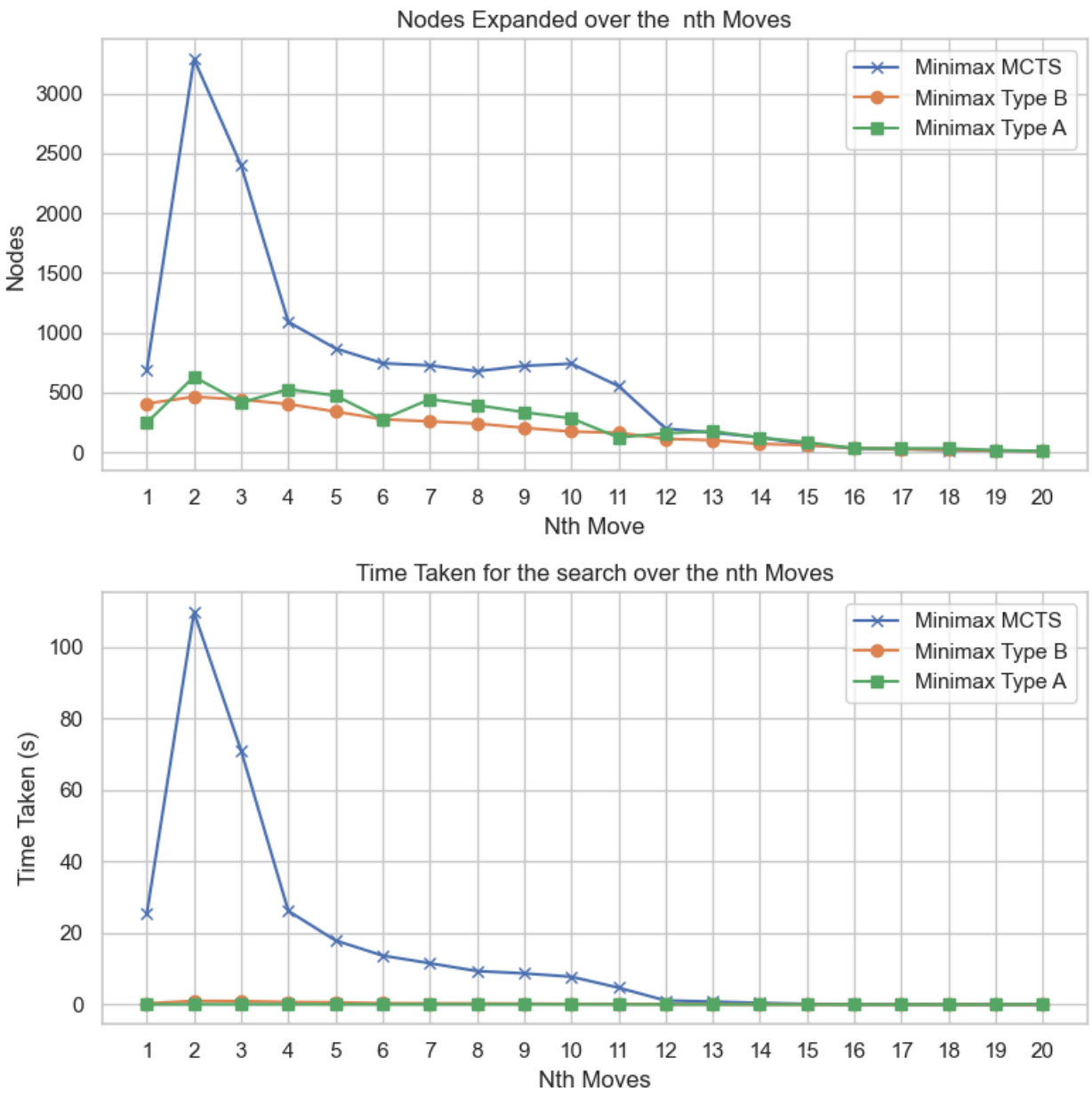


Figure 3. Les noeuds étendues et le temps de recherche aux cours du match pour MCTS

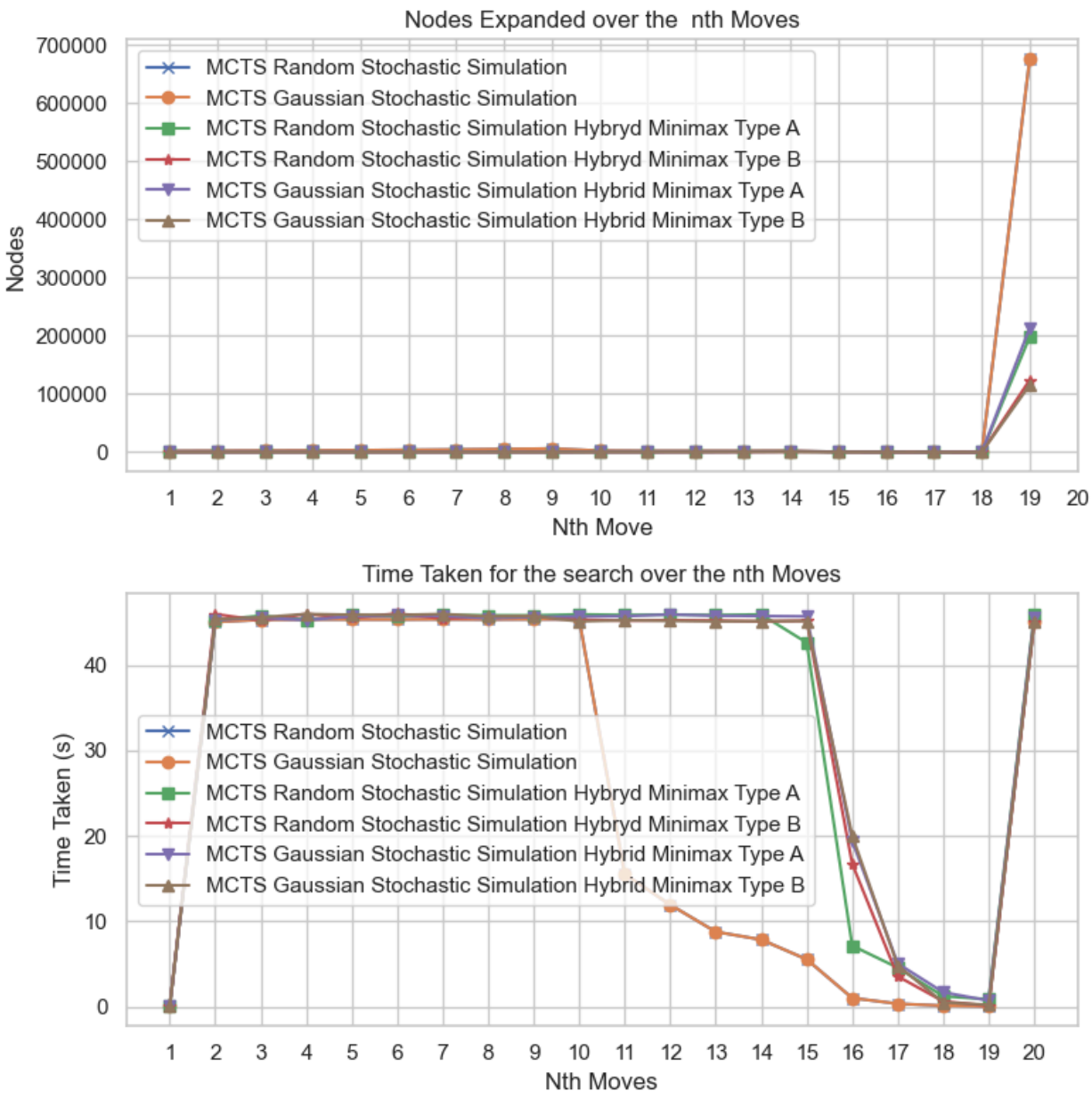


Figure 4. Le nombre de noeud étendues et le temps de recherche par différent type de fonction de perte

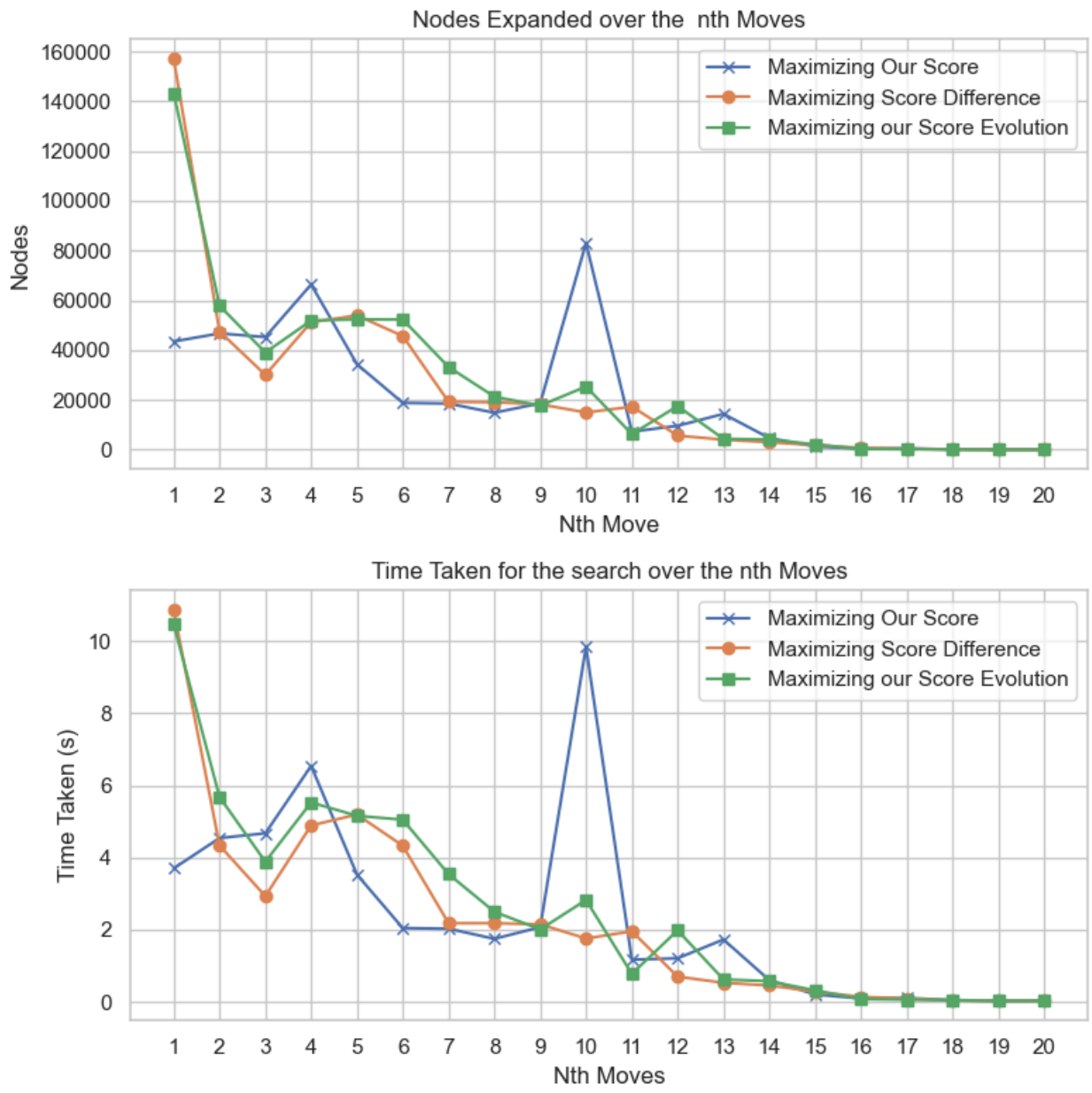
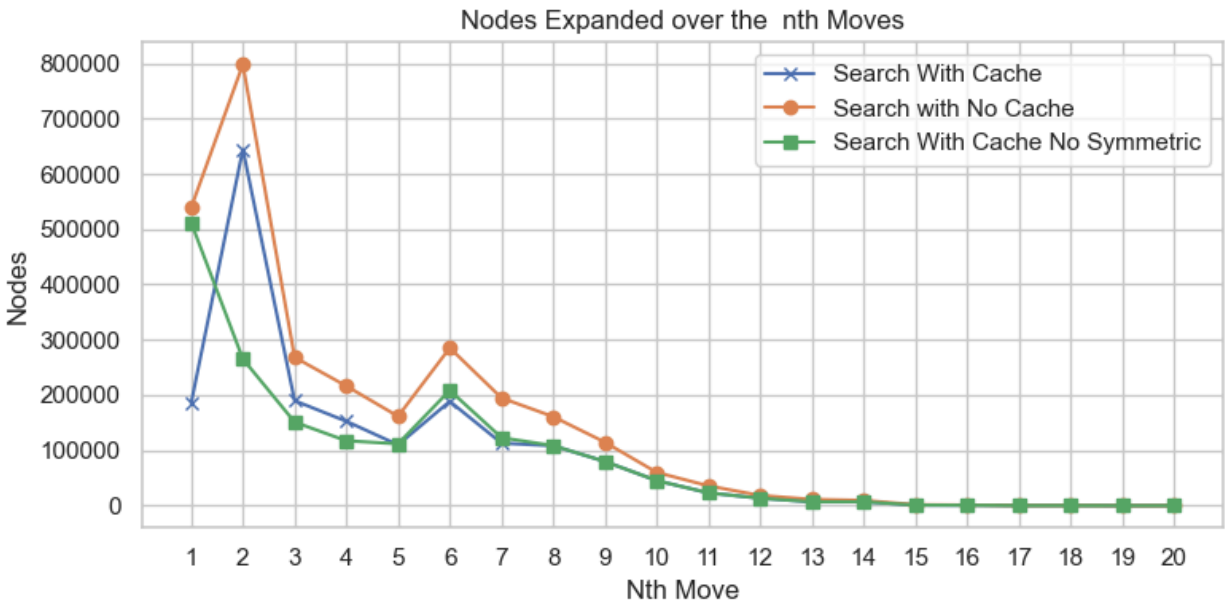
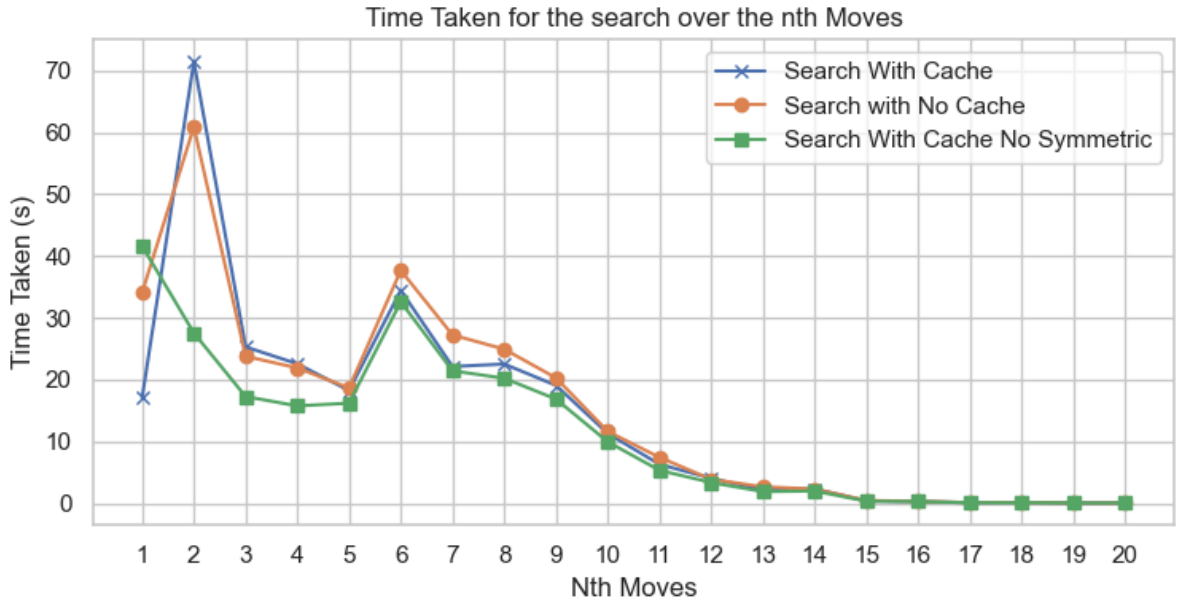


Figure 5. Le nombre de noeuds étendues et le temps de recherche pour différente situation concernant la cache


Nodes Expanded over the nth Moves
Search With Cache
Search with No Cache
Search With Cache No Symmetric
Nodes
800000
700000
600000
500000
400000
300000
200000
100000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Nth Move


Time Taken for the search over the nth Moves
Search With Cache
Search with No Cache
Search With Cache No Symmetric
Time Taken (s)
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Nth Moves