Rapport Projet Diversité

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Équipe Barracuda des Atlantides

INF8175 - Intelligence Artificielle

Automne 2024
Département de génie informatique
École Polytechnique de Montréal

Dernière mise à jour: 8 décembre 2024

Ines LOPEZ 2404168
Romain DUCHADEAU 2311547

Table des matières

1 Introduction

L'objectif de ce projet est de mettre en pratique les notions vues en classe dans un projet autonome sur la recherche adversarielle. L'objectif est de coder un algorithme capable de jouer au jeu de société divercité (dont nous ne rappellerons pas les règles ici) dans le but d'explorer les notions vues en classe : arbre de recherche, les paramètres à prendre en compte afin d'avoir une bonne heuristique ainsi que d'un bon algorithme, la difficulté à respecter les contraintes de temps etc... Finalement, nous avons réussis à implémenter différents agents et à gagner en performance au cours des essais. Nous vous présentons dans les sections suivantes les résultats de notre projet ainsi que le cheminement qui y a conduit.

2 Méthodologie

2.1 MinMax

Dans un premier temps, nous avons implémentés un algorithme MinMax avec une heuristique qui reposait sur le calcul du score. Ça n'est pas un algorithme optimal mais il constitue une bonne base pour la suite. Nous avons vu deux problèmes principaux à cet algorithme :

- L'heuristique choisie (le score) ne donne pas une bonne idée de si une situation est avantageuse ou non (par exemple une tour avec 3 ressources différentes autour est très avantageuse mais ne vaudra qu'au plus 1 point si on ne considère que le score)
- Le nombre d'état exploré est beaucoup trop grand : pour un algorithme MinMax la complexité temporelle est alors en $O(b^m)$ avec b le facteur de branchement qu'on estime à $\frac{(\sum_{i=0,i \text{ pair}}^{41} i)+(\sum_{i=0,i \text{ impair}}^{41} i)}{2m} = \frac{420+441}{38} \approx 12$ environ et m le nombre moyen de coup qu'il faut pour terminer la partie de 19 soit au total une complexité temporelle en $O(12^{19})$. Afin de respecter la contrainte de temps il est alors primordial de réduire le nombre d'état exploré.

2.2 Algorithme MinMax avec élagage Alpha-Beta et Stratégie B

L'algorithme implémenté dans ce projet repose alors sur l'approche MinMax combinée avec l'élagage Alpha-Beta pour optimiser les performances. Il intègre également une stratégie B pour limiter l'exploration des coups.

MinMax avec élagage Alpha-Beta

L'algorithme suit une structure typique de l'algorithme MinMax, alternant entre les fonctions `max` (pour le joueur Max) et `min` (pour le joueur Min). L'objectif est d'explorer l'arbre de jeu et de choisir le meilleur coup en fonction du score heuristique de chaque état. Le joueur Max cherche à maximiser le score, tandis que le joueur Min cherche à le minimiser.

Nous avons ensuite intégré un élagage Alpha-Beta. L'algorithme coupe l'exploration de certaines branches de l'arbre de décision lorsque l'on sait qu'elles ne peuvent pas influencer la décision finale. Plus précisément, si le score potentiel d'un nœud dépasse un seuil (`alpha`) ou est inférieur à un autre seuil (`beta`), il devient inutile d'explorer davantage cette branche, ce qui améliore l'efficacité de la recherche, et nous a permis d'augmenter la profondeur.

Stratégie B

L'algorithme intègre également une approche spécifique de type stratégie B, qui consiste à limiter l'exploration aux *n meilleurs coups* à chaque niveau de recherche. L'objectif est d'utiliser cette stratégie pour réduire (en largeur) encore davantage l'espace de recherche exploré par l'Alpha-Beta pruning, ce qui permet de gagner davantage en temps de calcul tout en conservant une exploration suffisante pour prendre des décisions informées.

2.3 Fonctionnement de l'heuristique

Dans le cadre de la conception de l'agent, nous avons mis en place plusieurs heuristiques visant à guider les décisions du joueur. Ces heuristiques évaluent les différents aspects du jeu, tels que la gestion des ressources, la protection contre les stratégies adverses, et l'optimisation des placements sur le plateau. Nous avons également appliqué une pondération sur ces heuristiques afin de définir leur influence respective sur la stratégie globale de l'agent.

Heuristiques définies

L'agent utilise plusieurs heuristiques, chacune visant à maximiser un aspect spécifique de la stratégie du jeu :

- **Empêchement de divercité** : Cette heuristique cherche à bloquer les tentatives de l'adversaire de compléter une divercité.
- **Répartition équilibrée des pièces** : L'heuristique favorise une répartition équitable entre les ressources et les tours en fonction des couleurs disponibles. Cela permet de maximiser les points à long terme en garantissant une flexibilité dans les placements.
- **Éviter la proximité des ressources de l'adversaire** : Cette heuristique pénalise les placements de ressources trop proches des tours adverses, réduisant ainsi les opportunités de l'adversaire de compléter une diversité autour de ses pièces.
- **Full divercité** : Cette heuristique permet de favoriser un état qui est plus susceptible de mener à une diversité

Choix des poids des heuristiques

Le choix des poids pour ces heuristiques repose sur une analyse du jeu et une expérimentation pour évaluer leur impact sur les performances de l'agent. Les étapes suivantes ont été suivies pour ajuster ces poids :

1. **Identification de l'importance relative de chaque critère :** Chaque heuristique représente un aspect différent du jeu. Par exemple, l'empêchement de la divercité adverse est crucial pour éviter des gains importants de l'adversaire, tandis que la répartition des pièces joue un rôle plus stratégique à long terme. Certaines heuristiques sont jugées plus cruciales que d'autres.
2. **Tests et ajustements empiriques :** Des tests ont été réalisés en simulant des parties avec des poids différents attribués à chaque heuristique. Par exemple, en attribuant un poids plus élevé à l'heuristique *empêchement de diversité*, l'agent devient plus agressif dans ses tentatives

de bloquer l'adversaire. En revanche, un poids plus élevé pour *répartition équilibrée des pièces* permet d'assurer une meilleure gestion à long terme.

3. **Équilibre entre les critères :** Un objectif important a été de garantir un équilibre entre la défense et l'optimisation des ressources. Par conséquent, les poids des heuristiques ont été ajustés pour éviter que l'agent ne se concentre trop sur l'une de ces stratégies au détriment des autres.

Combinaison des heuristiques

L'heuristique finale qui guide l'agent combine les valeurs calculées par chaque heuristique. L'agent évalue chaque action possible en fonction de l'heuristique globale qui intègre l'impact combiné de la gestion des ressources, des placements stratégiques et de la défense contre les diversités adverses.

Profondeur

Nous avons pour notre agent final décidés de mettre une profondeur de recherche de 5. Cette profondeur est basée sur les retours de jeux et est principalement limitée par le temps alloué à notre joueur pour jouer.

3 Résultats

Nous avons fait jouer notre agents contre plusieurs autres afin d'évaluer les performances de celui-ci. Nous n'avons pas ici reportés les résultats des combats contre nos précédentes versions car ils n'offrent pas un bon point de comparaison. Voici un tableau récapitulatif des performances de notre agent :

Temps total	**Adversaire**	**Profondeur**	**Résultat (Victoire)**
8 min 20 s	Greedy	5	27 - 19
10 min 40 s	Greedy	6	24 - 19
8 min	MinMax de base	5	29 - 15
15 min 10 s	MinMax de base	6	14 - 9
6 min	Random	5	28 - 19
11 min	Random	5	20 - 7

TABLE 1 – Résultats de l'algorithme contre différents adversaires.

Nous pouvons voir plusieurs choses dans ces résultats :

— Bien que dans tous nos essais nous gagnons contre l'agent random le résultat est parfois plus serré que d'autres. Cela est du au caractère imprévisible de l'agent random qui peut venir malgré lui contrecarrer nos heuristique.

— Avoir une profondeur plus élevé n'est pas synonyme de performance (comme on peut le voir contre le greedy) : cela peut s'expliquer par le fait que nos heuristiques n'ont pas les mêmes

impacts selon la profondeur et nos poids étant adaptés à une profondeur de 5 il faudrait réévaluer les pour cette nouvelle profondeur de 6.

Nous avons par ailleurs uploadé nos différents agents sur abyss pour les évaluer :

Nom de l'agent	**Performance**	**ELO**
Karma	6V-10D	889
Gagne_stp	7V-18D	733
BIG_DRAGON	7V-13D	684
Pink_unicorn (MinMax de base)	180V-832D	208

TABLE 2 – Résultats sur Abyss

Nous pouvons voir qu'au fur et a mesure des versions notre élo à augmenté mais il aurait fallu faire tourner nos modèles plus longtemps pour avoir des résultats statistiquement exploitable mais nous pouvons noter que plus le temps avance plus les derniers agents uploadés se battent contre des modèles performants (et battent la précédente version).

Pourquoi alors ne pas avoir augmenté la profondeur ? Pour des raisons de temps d'exécution de l'algorithme. En effet avec une profondeur de 6 nous dépassions toujours le temps impartis et réduire d'une dimension réduit suffisamment le temps de calcul pour être sur de prendre moins de 15min. Voici des graphiques de temps de calcul par tour contre l'algorithme random :

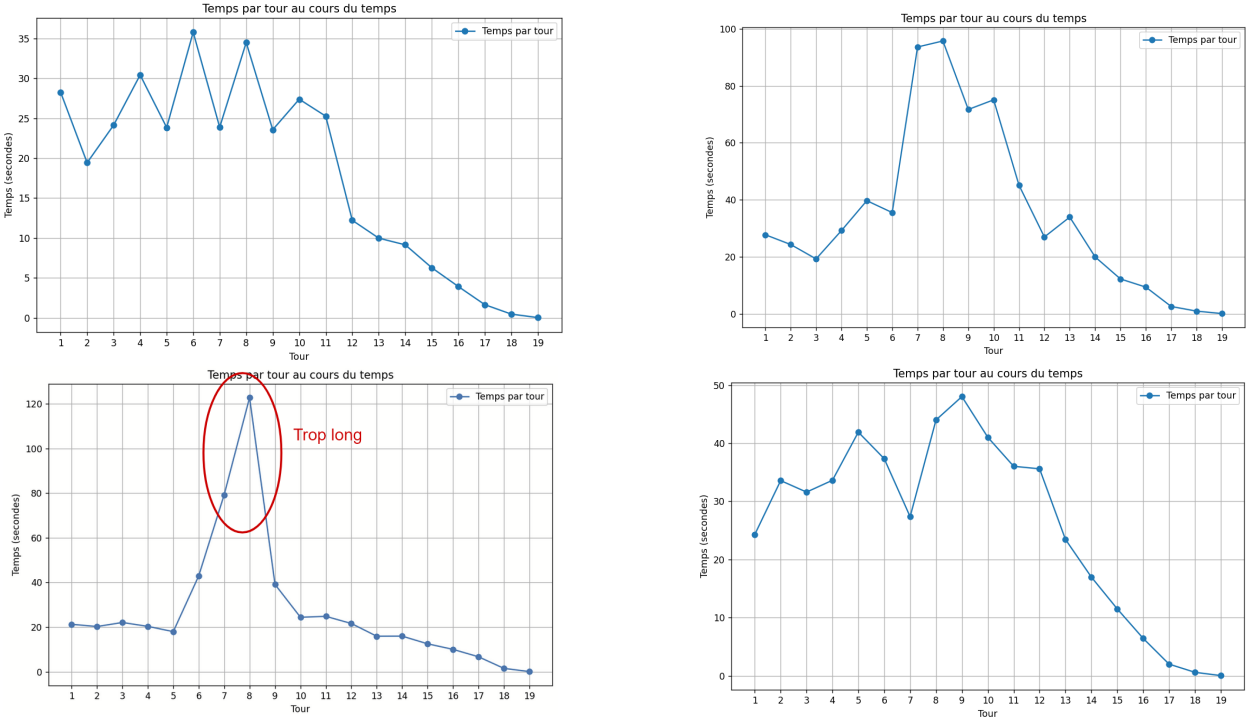

FIGURE 1 – Temps de calculs par tour pour notre heuristique finale contre random player

Notre choix pour cette analyse s'est porté sur random car il à l'avantage d'être suffisamment imprévisible pour confronter notre agents à toutes les configurations. Nous nous attendions à ce que le temps d'exécution (calcul des heuristique) décroisse au fur et à mesure des tours car il y a moins d'états à évaluer. C'est le cas mais nous observons une deuxième choses : certains tours prennent énormément plus de temps que d'autres à s'exécuter si bien qu'on perd trop de temps sur un seul tour. Les tours les plus longs à calculer sont bien souvent en milieu de partie.

4 Discussion

Notre agent, bien que fonctionnel et relativement performant présente encore certaines limites :

Des heuristiques dynamiques
Ses heuristiques sont statiques, ce qui peut entraîner des décisions sous-optimales. Une première amélioration consisterait à moduler dynamiquement les poids des heuristiques en fonction de l'avancement de la partie. Par exemple, au début du jeu, l'agent pourrait privilégier l'expansion rapide et la diversité des ressources, tandis que dans les phases finales, l'accent pourrait être mis sur la maximisation des points de Divercité. Une telle adaptation pourrait permettre à l'agent d'optimiser ses choix à chaque étape du jeu. Par ailleurs certaines pistes n'ont pas pus être explorée ici par manque de temps comme la répartition des tours au centre VS sur les bords etc... d'autres heuristiques pourraient donc être ajouter pour affiner le modèle

Gestion du temps
Nous avons vu dans la partie précédente que le temps de calcul de certaines heuristique dans certaines configuration était particulièrement long. Nous pouvons pousser la recherche pour comprendre pourquoi ainsi que définir un temps maximal par tour (afin d'éviter de tout gaspiller sur deux tours).

Prédiction de la stratégie adverse
Une deuxième piste serait d'introduire un système de prédiction des actions adverses basé sur l'historique des mouvements passés. En analysant les stratégies des joueurs adverses, l'agent pourrait ajuster sa propre stratégie en conséquence. L'apprentissage par renforcement, pourrait permettre à l'agent de mieux anticiper les tendances des adversaires et de s'adapter plus efficacement.

Monte Carlo pour améliorer le temps de calcul
Enfin, pour augmenter l'efficacité du calcul, une amélioration de l'algorithme de recherche pourrait être envisagée en combinant l'élagage Alpha-Beta avec des techniques de Monte Carlo Tree Search (MCTS). Cela permettrait à l'agent d'explorer plus profondément les choix possibles tout en conservant une prise de décision rapide.

En résumé, ces pistes d'amélioration permettraient non seulement de rendre l'agent plus performant, mais aussi d'adapter ses stratégies de manière plus flexible face à des situations changeantes.