POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

INF8175

Intelligence artificielle : méthodes et algorithmes

Rapport

Projet - Agent intelligent pour le jeu Abalone

Élèves :
Marc Zhang 2312403
Guilhem Garnier 2230323
Équipe garzha

Enseignant :
Quentin Cappart

3 avril 2024

Table des matières

1 Introduction

Dans le cadre de notre cours INF8175 - Intelligence artificielle : méthodes et algorithmes, notre projet consistait à mettre en œuvre un agent capable de jouer à un jeu de stratégie. Pour cette édition, le jeu choisi est Abalone.

2 Implémentation

2.1 Structure du code

L'ensemble de l'implémentation de notre agent est contenu dans le fichier `my_player.py`, dans la classe `MyPlayer` qui hérite de la classe `PlayerAbalone` fournie. Voici les méthodes de la classe que nous avons implémentées :

- `player.manhattanDist(A, B)` retourne la distance de Manhattan entre une position A et B.
- `player.compute_score(state, player_id, enemy_id)` retourne notre score pour le joueur en fonction de l'état de la partie.
- `player.compute_action(state, player_id, enemy_id)` retourne l'action à effectuer pour notre agent en fonction de l'état actuel de la partie. Cette méthode définit deux fonctions `max_value` et `min_value` qui aident au calcul.

2.2 Choix de conceptions faits

L'implémentation de notre agent est basée sur l'algorithme minimax, auquel on a ajouté l'élagage alpha-beta afin de réduire les répétitions de calcul. L'heuristique utilisé pour évaluer la valeur de chaque feuille de l'arbre est déterminée par l'équation suivante :

$$Score_{\text{player}} = Pieces_{\text{player}} - Pieces_{\text{opponent}} - \frac{DistanceCentre_{\text{opponent}} - DistanceCentre_{\text{player}}}{1000} \tag{1}$$

Où $Pieces_{player}$ correspond au nombre de pièces restantes du joueur, $Pieces_{opponent}$ représente le nombre de pièces restantes du joueur adverse, et $DistanceCentre_{player}$ est la somme des distances des pièces du joueur par rapport au centre du terrain.

L'idée derrière cette heuristique est simplement de détecter qui serait désigné vainqueur à l'instant choisi. En effet, en divisant par 1000 la valeur des distances au centre, cette valeur n'intervient que lorsque $Pieces_{player} = Pieces_{opponent}$. Le principal désavantage est le coût de calcul de distances si l'implémentation n'est pas incrémentale.

Pour la gestion du temps qui nous est alloué, nous avons varié la profondeur maximale explorée par l'algorithme au cours de la partie. Elle est fixée à 1 lorsque le jeu est entre le tour 1 et 9, à 2 lorsque le jeu est entre le tour 10 et 41, et à 3 lorsque le jeu est entre le tour 42 et 50. En effet, les premiers coups sont moins déterminants pour le résultat final que les derniers.

3 Résultats

3.1 Face à l'agent aléatoire

Dans les 30 simulations de match effectuées, notre agent n'a jamais perdu un seul match contre l'agent aléatoire. De plus, lors de chaque match, il n'a jamais perdu une seule pièce face à son adversaire.

3.2 Face à l'agent d'autres groupes

Pour tester l'efficacité de notre agent, nous l'avons testé contre celui du groupe algebros sur 10 parties, voici les résultats :

	Victoire	Défaite	Pièces gagnés	Pièces perdues	Manches
Partie 1	X		6	0	50
Partie 2	X		4	0	50
Partie 3	X		4	0	50
Partie 4	X		4	0	50
Partie 5	X		6	0	48
Partie 6	X		4	0	50
Partie 7	X		4	0	50
Partie 8	X		4	0	50
Partie 9	X		3	0	50
Partie 10	X		2	0	50

TABLE 1 – Résultats des match contre le groupe algebros

Nous avons obtenu de très bons résultats lors des simulations, avec un taux de victoire de 100 %. Cependant, beaucoup de ces victoires sont dues à la limite de tours plutôt qu'à une élimination complète des six pièces adverses.

Nous avons remarqué que lors des parties simulées, notre agent a tendance à adopter une stratégie défensive en restant au centre plutôt que de chercher à éliminer les pièces ennemies. Cela est dû à une profondeur qui n'est pas suffisamment grande, rendant plus difficile l'élimination des pièces adverses en plusieurs coups.

3.3 Concours challonge

Dans le concours Challonge, notre agent a présenté des résultats honorables en passant les phases de poule à la deuxième position avec 4 victoires sur 5, puis en remportant son premier match de la phase finale. Voici le récapitulatif de ses matchs :

Adversaire	Victoire	Défaite	Pièces gagnés	Pièces perdues
RandomPlayer	X		5	0
AbaloneGo	X		3	1
EAGLE	X		5	0
2021242	X		3	2
Thomaxel		X	2	3

TABLE 2 – Résultats des match en phase de poule

	Victoire	Défaite	Pièces gagnés	Pièces perdues
Partie 1		X	5	6
Partie 2	X		6	5
Partie 3	X		1	0
Partie 4		X	3	5
Partie 5	X		6	5

TABLE 3 – Résultats du match contre TeamAlpha

	Victoire	Défaite	Pièces gagnés	Pièces perdues
Partie 1		X	4	6
Partie 2		X	3	5
Partie 3		X	0	1
Partie 4		X	2	3
Partie 5	X		4	2

TABLE 4 – Résultats du match contre BriqusBaltrus

On remarquera que sur les 8 parties remportées par notre agent, seules deux d'entre elles n'ont pas été remportées par l'élimination des 6 pièces adverses. Cela résulte encore une fois d'un manque d'agressivité dans la stratégie de notre agent.

4 Avantages et limites de l'agent

Les résultats du concours challonge ont montré que, si notre agent se défendait correctement contre les autres agents, il n'était pas assez optimisé pour figurer parmis les meilleurs agents du concours. Nous avons pu identifier quelques pistes pour améliorer notre agent :

— Une meilleure gestion du temps, en prenant en compte le temps restant pour gérer la profondeur de la recherche.
— Un calcul incrémental de l'heuristique (quand une personne joue, seule une ou deux pièces sont mises à jour).
— Un meilleur filtrage de l'arbre de recherche en retirant haut dans l'arbre les actions absurdes.

— Une meilleure gestion de la profondeur de recherche lors de la partie pour permettre l'apparition de stratégies plus offensives.

5 Tentatives d'améliorations et conclusion

En suivant les résultats du concours et les idées évoquées au point précédent, nous avons essayé d'améliorer notre agent. Notamment, nous avons filtré les actions absurdes (par exemple des actions "suicides" où le joueur pousse sa propre boule hors du terrain) lors de l'exécution de l'algorithme min-max (en ajout de l'alpha/beta pruning). Cette semble avoir légèrement améliorer les performances de l'agent (de l'ordre de 10 pourcents plus rapide) mais nous n'avons pas pu lancer suffisamment de tests pour avoir une confiance suffisante.

Il nous a été assez difficile de mesurer l'impact des modifications apportées à l'algorithme étant donnés le temps par partie (30 minutes) et les écarts faibles entre agent (un agent peut gagner puis perdre contre un même opposant, ce qui nécessite beaucoup de parties pour conclure).

En conclusion, nous estimons qu'obtenir un meilleur agent basé sur la méthode min-max revient à des questions d'optimisation de l'algorithme. Principalement, le calcul de l'heuristique pourrait être largement accéléré en calculant incrémentalement le coût d'un état par rapport à l'état précédent. Il serait aussi intéressant de comparer la méthode min-max avec d'autres méthodes, comme des algorithmes de Monte Carlo ou des méthodes d'apprentissage par renforcement.