Rapport de projet

Agent intelligent pour le jeu Abalone

Équipe *Frabalon*

François MARIE
2309736

François GOYBET
2307249

POLYTECHNIQUE MONTRÉAL

INF8175 - Intelligence Artificielle : Méthodes et Algorithmes

15 April 2024

Sommaire

1. Introduction

Le but de ce projet est de développer un agent intelligent pour le jeu Abalone. Le jeu consiste en un plateau hexagonal où deux joueurs s'affrontent avec chacun 14 billes. Le but est de pousser les billes adverses hors du plateau dans un maximum de 50 tours. Certaines règles de déplacements ont été modifiées dans le sujet pour simplifier le jeu et réduire la compléxité.

2. Analyse du jeu

Afin de développer un agent intelligent, il est d'abord nécessaire de comprendre les règles et stratégies du jeu. Nous connaissions déjà Abalone avant ce projet ce qui nous a permis de rapidement faire émerger des idées lors de nos parties :

- Plus une bille se trouve proche d'un bord, plus elle est en danger. Les cases sur les "sommets" de l'hexagone sont d'autant plus dangereuses car une bille peut se faire éjecter depuis trois directions différentes.
- Le centre du plateau, en particulier la case centrale, est un endroit stratégique qui permet de dominer l'adversaire en le forçant à disposer ses billes en croissant autour de la case centrale et donc de le pousser plus facilement.
- Les formations en "blob" sont plus faciles à protéger que les billes isolées ou les formation en ligne. En effet, cette formation offire le moins de surface d'attaque et permet de riposter dans plusieurs directions.
- Il est souvent contreproductif de chercher à ramener une bille isolées dans un groupe si cela prend plus de 1 ou 2 tours.
- Les priorités évoluent avec le temps. En début de partie, il est souvent préférable de chercher à contrôler le centre et construire une formation avec ses billes. Inversement, lors des derniers tours, il est préférable d'attaquer l'adversaire, quitte à perdre le centre ou déconstruire sa formation.

3. Agents développés

3.1. Alphabeta

3.1.1. Description

La version principale de l'agent de notre projet utilise l'algorithme alphabeta. C'est la version que nous avons utilisée pour le tournoi et la remise.
La première version de l'agent développée fut un agent minimax simple afin de tester les mécanismes du projet fourni. Nous avons ensuite rapidement implémenter l'élagage alphabeta pour améliorer les performances de l'agent et obtenir un meilleur niveau de profondeur.
Par la suite nous avons ajouté une heuristique basée sur les stratégies évoquées dans **Analyse du jeu**. L'heuristique a été modifiées et améliorée au fur et à mesure de l'avancement du projet. La version finale est décrite dans la section **Heuristique**.

3.1.2. Évolution

3.2. Monte Carlo Tree Search

4. Conception des agents

4.1. Heuristique

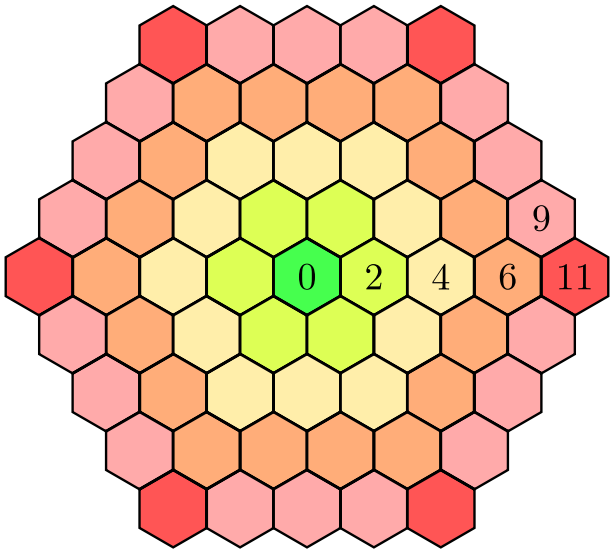

Figure 1: Représentation de la dangerosité des cases du plateau

4.2. Optimisations

4.2.1. Précalculs

Le fichier `utils.py` contient des données précalculées qui permettent de gagner du temps d'exécution.

Distance au centre
Le système de coordonnées particulier du jeu rend difficile d'évaluer la distance au centre à partir d'une pièce. De plus nous souhaitons différencier les sommets du plateau des autres cases du bord, comme en Figure 1. Nous avons donc décidé de créeer manuellement une hashmap constante `DANGER` contenant l'ensemble des positions possibles avec un niveau de danger personnalisé. Cet approche permet une grande flexibilité sur l'ajustement des niveaux de danger et évite le calcul de la distance au centre à chaque tour.

Distribution normale du facteur de clustering
L'importance du score de clustering évolue selon une loi normale (centrée en 20 avec un écart-type de 5). Dans un premier temps, nous calculions la valeur avec la librairie *scipy* à chaque tour. Cependant, cela a gravement nuit aux performances de l'agent et a quasiment doublé le temps d'exécution : sur Alphabeta avec une profondeur de 3, le temps est passé de 1min30 à presque 3min. Pour pallier à ce problème, les valeurs possibles ont été précalculées et stockées dans une liste constante.

4.2.2. Gestion du temps

Pour éviter de dépasser le temps maximal de 15min avec Alphabeta, le niveau de profondeur est ajusté dynamiquement lors de l'appel à la fonction `compute_action()` :

- Dans la quasi-première moitié de la partie (jusqu'au tour 19), la profondeur est fixée à 3. En effet l'agent doit seulement se concentrer sur le contrôle du centre et la formation de ses billes.
- Après cette phase terminée, la profondeur est augmentée à 4 pour attaquer l'adversaire et éjecter ses billes.
- Si le temps restant est inférieur à 2min30, la profondeur passe à nouveau à 3 pour éviter de dépasser le temps imparti et risquer de perdre la partie.
- Lors des 3 derniers tours, la profondeur est fixée au nombre de tours restants afin d'éviter de calculer des coups inutiles.

5. Résultats

5.1. Performances

5.2. Tournoi

Lors du tournoi, notre agent a réussi à finir 2ème de sa poule avec 2 victoires pour 1 défaite, avec un total de 37 points. Cependant, le premier round a été perdu 4-1 contre un autre agent. Bien que cette défaite soit un peu décevante, nous sommes satisfaits de la performance globale de notre agent. Le système de gestion de temps semble avoir fonctionner correctement sur le serveur de tournoi.

6. Conclusion