Rapport AI Game Programming

Jefferson Storck

2025-2026

Introduction

Le but de ce project est de programmer un jeu à deux joueurs en tour par tour, ainsi qu'une AI capable de jouer. Dans ce rapport nous allons détailler le fonctionnement de l'application, l'implementation de l'AI, notamment la fonction d'evaluation, et la raisonnement derrière certaines decisions,comme profondeur de recherche utilisée.

Fonctionnement

L'application possède deux versions: une version basique, qui permet de jouer entre deux joueurs humains ou un joueur contre l'AI, et une version specialement conçue pour une compétition entre deux AI. Les deux versions sont écrites en **C++** et son à disposition sur le **GitHub** suivant:`https://github.com/JS-291/AIGamesJeffersonStorck`. Pour chaque version on fournit un code source ainsi qu'un **makefile** pour compiler l'application.

2.1 Version basique

La version basique proposent deux modes de jeu: un mode joueur contre joueur et un mode joueur contre AI. Le joueur **P1** joue toujours en premier. Lors de l'éxecution le programme demande quel mode de jeu le joueur souhaite utiliser:

Single pour deux joueurs humains

AI pour un joueur contre l'AI.

Si le mode **AI** est choisi, le programme demande ensuite quel joueur l'utilisateur souhaite incarner: **P1** ou **P2**.
L'application s'utilise entièrement dans la console. A chaque tour le plateau est affiché et le programme demande au joueur courant de rentrer son coup. Si le coup est invalide le programme redemande un coup jusqu'à ce qu'un coup valide soit entré.

2.2 Version compétition

La version compétition s'utilise avec un abitre programmer en **Java**. Pour choisir l'AI à utiliser, il suffit de changer les lignes suivantes dans le fichier **Arbitre.java**:

```
1 Process A = Runtime.getRuntime().exec("./AI");
2 Process B = Runtime.getRuntime().exec("./AI");
```

2.3 Règles du jeu

Le plateau de jeu est une grille de 16 trous. Le joueur **P1** contrôle les trous impairs. Le joueur **P2** contrôle les trous pairs. les trous peuvent contenir des graines, il en existe trois types:

- Graines rouges, semées par les coups nR
- Graines bleues, semées par les coups nB
- Graines transparentes, semées par les coups nTR ou nTB

Ici **n** est le numéro du trou joué.

Au début de la partie chaque trou contient 2 graines de chaque type. Le but du jeu est de capturer le plus de graines possible.

2.3.1 Semage

Le semage consiste à choisir un trou, prendre toutes les graines d'un certain type, puis à les répartir dans les trous suivants dans le sens horaire

les graines sont semées une par une dans les trous suivants.

les graines bleues sont semées uniquement dans les trous de l'adversaire.

les graines transparentes peuvent être semées comme des graines rouges ou bleues. Si le trou possède aussi des graines bleues ou rouges elles sont semées après les graines transparentes en fonction du choix. Les graines transparentes restent transparentes après avoir été semées.

2.3.2 Capture

La capture se fait après le semage. On vérifie si le dernier trou semé contient trois ou deux graines. Si c'est le cas, le joueur capture ces graines. On vérifie ensuite le trou précédent, et ainsi de suite jusqu'à ce qu'on trouve un trou qui ne contient pas deux ou trois graines. La partie se termine s'il reste moins de dix graines sur le plateau, ou si un joueur score est superieur à 49. Si un joueur ne peut plus jouer, l'autre joueur capture toutes les graines restantes.

Implémentation

Le programme est implémenté en **C++**, il utilise les classes suivantes:

- **Game**: La main classe qui gère le choix du mode de jeu, parse les coups, et assure le déroulement de la partie.
- **Utils**: Une classe qui contient les fonctions partagées entre les differentes classes, ainsi que la définition des structures de données utilisées.
- **Engine**: La classe qui gère les mécanismes du jeu comme le semage, la capture, et la vérification des coups.
- **AI**: La classe qui implémente la logique de l'AI, comme l'algorithme Minimax ou la fonction d'évaluation.

3.1 Implémentation de l'AI

3.1.1 Algorithme Minimax

L'AI utilise un algorithme Minimax avec coupes alpha-béta pour choisir le meilleur coup à jouer. Cet algorithme simule les coups possibles à l'aide de fonctions similaires à celles de de la classe **Engine**. Les états de jeu sont conservés à l'aide de la structure **Node** définie dans la classe **Utils**, leur valeur est ensuite determinée par la fonction d'évaluation. La profondeur de recherche est fixée à 4, une profondeur supérieure augmente significativement le temps de calcul.

3.1.2 Fonction d'évaluation

La fonction d'évaluation calcule une valeur pour un état de jeu donné, elle est définie symétriquement pour les deux joueurs. Si plusieurs états ont la même valeur, l'AI choisit aléatoirement entre eux. Elle évalue plusieurs critéres à qui on attribue des poids différent:

- Le score actuel du joueur moins le score de l'adversaire (poids 20).
- Si le joueur courrent est affamé (poids -50).
- Si l'adversaire est affamé (poids 50).
- Si le joueur courrent à perdue (poids -500).
- Si l'adversaire à perdue (poids 500).
- Si la partie termine sur un match nul (poids 10).
- Le nombre de trous vides possédés par l'adversaire (poids 2 pour chaque). L'idée ici est de limiter les possibilités de semage de l'adversaire.

- Nombre de graines transparentes possédées par l'adversaire (poids -2 pour chaque). Les graines transparentes sont les plus flexibles que les graines rouges ou bleues, on cherche donc à en limiter le nombre chez l'adversaire.
- Nombre de graines rouges possédées par l'adversaire (poids -1 pour chaque). Même raisonnement que pour les graines transparentes, les graines rouges sont plus flexibles que les graines bleues.
- Si l'adversaire posséde des trous avec 1 ou 2 graines (poids 3 pour chaque). Augmente les chances de capturer les graines de l'adversaire.
- Si le joueur courrent posséde des trous avec 1 ou 2 graines (poids -3 pour chaque). Logique inverse du critére précédent.

Chaine de captures

La fonction d'évaluation effectue aussi une recherche des trous qui peuvent être capturés par l'adversaire au prochain tour. L'idée ici est de prendre chaque trou que l'adversaire peut atteindre en semant ses graines, puis de simuler une capture et determiner si une chaine de captures est possible. En fonction de la longueur de la chaine de capture trouvée, on subtrait 2*n à l'évaluation. La logique inverse est appliquée pour le joueur courant, on ajoute 2*n à l'évaluation pour chaque trou capturable trouvé.

Conclusion

A travers ce projet nous avous pu étudier comment implémenter une AI pour un jeu de stratégie en tour par tour. Nous avons vu comment choisir des coups optimals à l'aide d'un algorithme Minimax, appris l'importance des coupes alpha-béta pour réduire le temps de calcul et avons réalisé l'importance d'une fonction d'évaluation performante.