Othello

8, 11, 18 mars

Page web : `http://ljk.imag.fr/membres/Clement.Pernet/L3MI_POO.html`

Le TP peut être préparé seul ou en binôme. Le rendu doit consister en un fichier archive `<Nom1_Nom2>.tar.gz` contenant

- votre code C++, commenté, facilement compilable : si la commande `make` ne suffit pas à compiler votre code, ajoutez un fichier README expliquant précisément comment compiler votre code.
- (optionel) un compte-rendu de TP répondant aux questions qui ne demandent pas un programme.

Le TP doit être rendu par couriel à l'adresse de l'encadrant de votre groupe :

Groupe 1 : `clement.pernet@univ-grenoble-alpes.fr`

Groupe 2 : `hippolyte.signargout@univ-grenoble-alpes.fr`

avec en préfixe du subject `[L3MI-TP4]` suivi de votre (vos) nom(s). La date limite de rendu est jeudi 24 mars 2022.

Un rapport décrivant ce que vous avez implémenté ainsi que le code correspondant est demandé pour le vendredi suivant le dernier TP. Dans celui-ci, vous décrirez (rapidement) les différentes stratégies implémentées. Une évaluation des performances de chaque algorithme sera appréciée.

L'objectif de ce dernier TP est de réaliser un jeu d'othello. On s'intéressera dans un premier temps à la programmation du jeu, puis à réaliser une interface graphique sur laquelle on puisse jouer à deux joueurs et on réalisera enfin un ou des programmes d'intelligence artificielle contre lequel on puisse jouer.

Exercice 1. Le Jeu

Othello est est un jeu à deux joueurs qui se joue sur une grille 8×8 avec des pions noirs et blancs. Voir `https://fr.wikipedia.org/wiki/Othello_(jeu)` pour les règles du jeu.

a. Programmer une classe `Grille` qui stocke l'état du plateau de jeu : un tableau 8×8 décrivant la nature de chaque case : vide, noire ou blanche, le score de cet état défini par la différence entre le nombre de pions noirs et de pions blancs. Cette classe permettra en outre de lire et modifier l'état d'une case. Elle disposera d'une méthode `affiche` qui affichera l'état de la grille sur la sortie standard.

b. Programmer une classe `Jeu` qui sera en charge du déroulement du jeu. Elle contiendra en particulier :

- un attribut grille pour représenter l'état courant
- une ou des méthodes qui permmette aux joueurs blancs et noirs de placer un pion. Par exemple, deux méthodes `jouerBlanc` et `jouerNoir` prenant en argument une position. Cette ou ces méthodes renverront un booléen valant faux si et seulement si le coup n'est est valide.
- une méthode `fini` qui détermine si la partie est finie est retourne quelle couleur à gagné.
- une méthode `executer` qui se charge de faire jouer alternativement les joueurs blancs et noirs dans une boucle jusqu'à la fin du jeu. Il s'agit ici de permettre un jeu à deux joueurs humains via l'interface des entrées/sorties standard d'un terminal.

c. Écrire une classe de test qui vous permette de jouer à 2 joueurs en mode texte.

Exercice 2. Construction d'une interface graphique

On se propose maintenant d'ajouter une interface graphique `wxWidget` à notre jeu en utilisant la classe `Jeu` ci-dessus.

a. Implémenter cette interface qui devra :

— dessiner le fond du plateau vert, une grille des cases, ainsi que les pions (blancs ou noir) ;
— récupérer des événements de clic souris sur cette grille pour faire jouer alternativement les 2 joueurs ;
— afficher dans une zone de texte la couleur qui doit jouer le prochain coup ;
— gérer la fin de la partie avec l'affichage du vainqueur ;
— proposer des boutons "Nouvelle partie" et "Quitter"

b. (Bonus) faire une animation lorsqu'un jeton est retourné (on pourra utiliser `wxTimer` et tracer des ellipses.

c. (Bonus) Améliorer l'interface graphique : remplacer les ronds par des images, ajouter des menus...

d. (Bonus) Implémenter le choix d'une grille de dimensions supérieures.

Exercice 3. Implémentation la stratégie de l'ordinateur

Dans la suite, on se propose d'implémenter diverses heuristiques afin de pouvoir jouer contre l'ordinateur.

a. (Pour se chauffer) : programmer un ordinateur qui joue au hasard.

b. Implémenter un algorithme glouton, c'est à dire qui va choisir le coup qui maximise le score du plateau suivant (à horizon de temps 1).

c. Implémenter un algorithme min-max. Quelle profondeur atteignez vous en un temps raisonnable (1 à 2s) ?

d. On se donne une limite de temps de 2 secondes par coup. Afin de respecter cette limite, on propose de chercher le coup optimal pour une profondeur 1 puis 2 puis 3 etc, jusqu'à ce que les 2 secondes soient écoulées. Fait-on beaucoup de calculs pour rien en calculant les première profondeurs ?

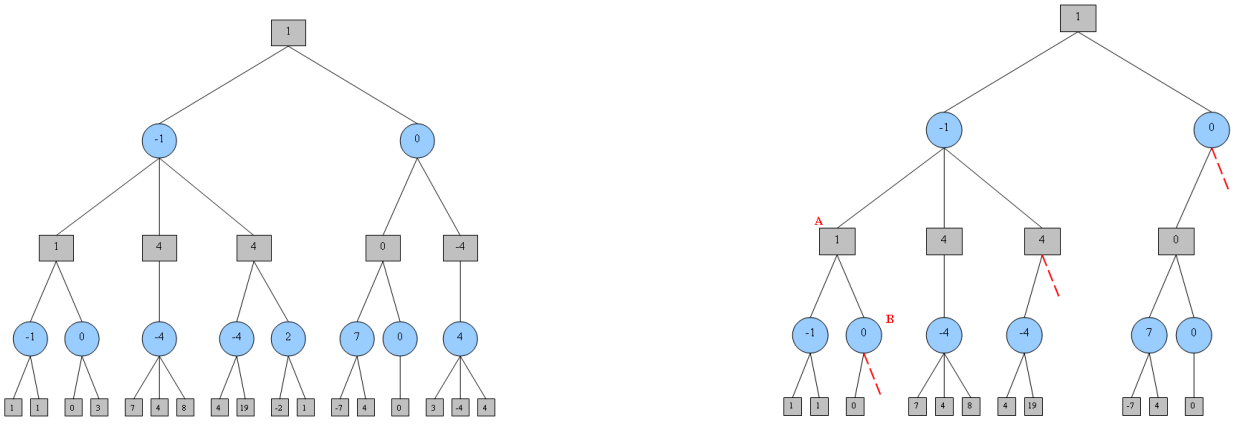

FIGURE 1 – Minmax vs alpha-beta

e. L'élagage alpha-beta permet d'accélérer grandement les calculs en n'évitant de parcourir inutilement certains noeuds tout en donnant le même résultat. On peut donc atteindre une meilleure profondeur. Implémenter l'alpha-beta.

f. Choisir une profondeur pour laquelle votre min-max calcule en une durée raisonnable. Combien de noeuds l'alpha-beta permet il d'élaguer ? Quelle profondeur pouvez vous atteindre maintenant ?