TP15 – Compétition de Reversi

Règles du jeu

On joue sur une grille, une couleur est attribuée à chaque joueur. À chaque tour, un joueur pose un pion sur une case de la grille telle qu'il existe un chemin en ligne droite entre ce pion et un pion de la même couleur, avec au moins un pion d'une autre couleur entre les deux. Tous les pions qui se trouvent entre le pion qui vient d'être posé et un premier pion de la même couleur sont retournés pour prendre la couleur du joueur. À chaque coup, le joueur qui joue doit nécessairement placer son pion à un endroit qui lui permet de retourner des pions adverses.

Le jeu s'arrête dès qu'il ne peut plus jouer. Le gagnant est alors le joueur disposant du plus grand nombre de pions de sa couleur.

Liberté du langage

Vous programmez en OCaml ou en C, à votre choix.

Programmer le jeu de Reversi

Il peut être utile de programmer les règles du jeu de Reversi :

- choisir une représentation pour le plateau de jeu (la taille n'est pas fixée même si la version officielle est un carré de 8 cases de côté) ;
- déterminer, étant donné une couleur de pion et un plateau, dans quelles cases il est autorisé de poser de pion ;
- modifier le plateau en jouant un coup donné.

Élagage $\alpha\beta$

On suppose que l'on dispose déjà d'une heuristique donnée, qui évalue le score attribué à un plateau (sur un ensemble de valeurs de votre choix). Programmer l'algorithme d'élagage $\alpha\beta$ vu en cours, en limitant la recherche avec une profondeur donnée en paramètre.

(Indication : si besoin, commencez par programmer minmax, puis ajoutez les paramètres nécessaires pour l'élagage.)

Discussions avec le serveur

Vous devez écrire un programme qui se connecte au serveur arbitre, dont le rôle est d'organiser des parties entre vos programmes. La communication avec le serveur via le réseau s'effectue au travers d'une *socket*, qui se manipule quasiment comme un fichier.

Le protocole entre un client et le serveur est le suivant. Dans les exemples, on préfixe par `<--` les messages envoyés par le client et par `-->` ceux envoyés en réponse par le serveur. Chaque message est écrit sur une ligne qui se termine par le caractire `\n`. Chaque ligne contient des données au format JSON, sans aucun retour à la ligne.

Lorsque votre programme se connecte au serveur, il se présente en indiquant son nom et en indiquant le nom d'une partie à laquelle il veut jouer. C'est le nom de la partie qui permet de faire jouer l'un contre l'autre deux joueurs :

```
<-- {"name":"Eve","game":"partie_contre_adam_42"}
--> {"status":"joined","your_color":"#"}
```

Ensuite, le serveur envoie l'état du plateau, en indiquant à chaque fois si c'est au joueur de jouer ou bien si c'est à l'adversaire de jouer :

```
--> {"board":"...O#.
(etc.)","width":8,"height":8,"status":"your_turn","your_color":"#","your_turn":true}
```

Les couleurs des pions sont soit `O` soit `#`. Le champ `your_color` permet de savoir quelle couleur est jouée. Le jeu commence toujours par le joueur ayant la couleur `O`. Le plateau est une grande chaine de caractères où `.` représente une case encore vide. Le plateau est donné ligne par ligne. Le champ `width` donne le nombre de colonnes, le champ `height` donne le nombre de lignes.

Quand c'est à son tour de jouer, le joueur envoie son coup en donnant les coordonnées. L'origine est en haut à gauche du plateau, les coordonnées commencent à 0. Le serveur répond alors avec le nouvel état du plateau :

```
<-- {"do_move":[3,6]}
-->
{"board":"...","width":8,"height":8,"your_turn":false,"your_color":"#","status":"move_d
one"}
```

Le serveur peut également répondre l'une des lignes suivantes en cas d'erreur ou en cas de fin du jeu :

```
--> {"status":"wrong_move","board":"...","width":w,"height":h}
-->
{"status":"game_over","board":"...","width":w,"height":h,"winner":"#","your_color":"#"}
```

J'ai commencé à rédiger des fichiers `joueur.c` et `joueur.ml` pour vous aider à discuter avec le serveur.

Last updated 2023-03-23 09:49:37 +0100