Projet de Programmation C – Autour du jeu Othello

LC4 – Licence 2 – Université Paris Diderot

10 avril 2012

1 Présentation du jeu Othello

Othello est un jeu de société combinatoire abstrait, qui oppose deux joueurs : Noir et Blanc.

Il se joue sur un tablier unicolore de 64 cases, 8 sur 8, appelé othellier. Les colonnes sont numérotées de gauche à droite par les lettres A à H ; les lignes sont numérotées de haut en bas par les chiffres 1 à 8.

Les joueurs disposent de 64 pions bicolores, noirs d'un côté et blancs de l'autre. En début de partie, quatre pions sont déjà placés au centre de l'othellier : deux noirs, en E4 et D5, et deux blancs, en D4 et E5.

Chaque joueur, noir et blanc, pose l'un après l'autre un pion de sa couleur sur l'othellier selon les règles définies ci-dessous. Le jeu s'arrête quand les deux joueurs ne peuvent plus poser de pion. On compte alors le nombre de pions. Le joueur ayant le plus grand nombre de pions de sa couleur sur l'othellier a gagné.

1.1 Règles du jeu

Noir commence toujours la partie. Puis les joueurs jouent à tour de rôle, chacun étant tenu de capturer des pions adverses lors de son mouvement. Si un joueur ne peut pas capturer de pion(s) adverse(s), il est forcé de passer son tour. Si aucun des deux joueurs ne peut jouer, ou si l'othellier ne comporte plus de case vide, la partie s'arrête. Le gagnant en fin de partie est celui qui possède le plus de pions.

La capture de pions survient lorsqu'un joueur place un de ses pions à l'extrémité d'un alignement de pions adverses contigus et dont l'autre extrémité est déjà occupée par un de ses propres pions. Les alignements considérés peuvent être une colonne, une ligne, ou une diagonale. Si le pion nouvellement placé vient fermer plusieurs alignements, il capture tous les pions adverses des lignes ainsi fermées. La capture se traduit par le retournement des pions capturés. Ces retournements n'entraînent pas d'effet de capture en cascade : seul le pion nouvellement posé est pris en compte.

Par exemple, la figure de gauche ci-dessous montre la position de départ. La figure centrale montre les 4 cases où Noir peut jouer, grâce à la capture d'un pion Blanc. Enfin, la figure de droite montre la position résultante si Noir joue en C4. Le pion Blanc D4 a été capturé (retourné), devenant ainsi un pion Noir.

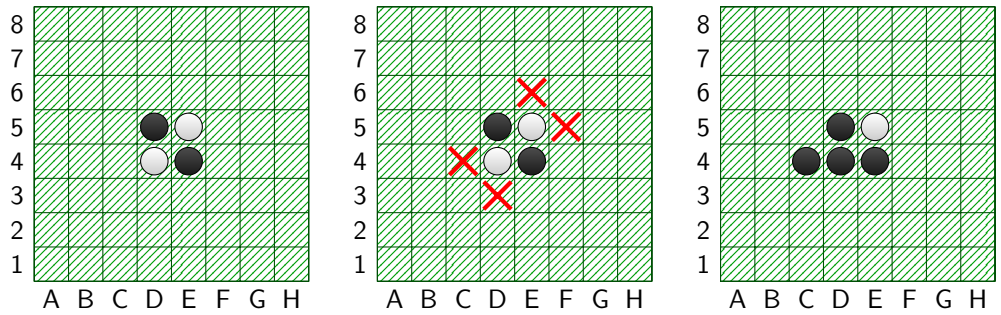

FIGURE 1 – De gauche à droite : la position initiale, les coups que peut jouer noir, et après le coup C4 de noir.

1.2 Représentation et fonctions de base

Plusieurs types sont imposés pour ce projet :

– Un joueur est représenté par le type de données suivant :

```
typedef char joueur;
```

`'B'` représente le joueur blanc et `'N'` représente le joueur noir.

– Un coup est représenté par le type de donnée suivant :

```
typedef char coup[3];
```

Le coup D3 de noir sera représenté par `{'N', 'D', '3'}`, le coup E3 de blanc sera représenté par `{'B', 'E', '3'}`, si blanc passe (blanc ne peut pas jouer de coup valide), ce coup est représenté par `{'B', 'P', 'A'}`.

– L'othellier est représenté par le type suivant [1] :

```
typedef char othellier[8][8];
```

Si la case A7 d'un othellier `t` contient un pion blanc, alors `t[0][6] == 'B'`, si la case B1 contient un pion noir, alors `t[1][0] == 'N'`, si la case C3 est vide, alors `t[2][2] == '␣'` (le caractère espace).

– Une suite de coups est représenté par le type suivant :

```
typedef struct {
  int nbcoups;
  coup coups[];
} partie;
```

Avec nbcoups représentant le nombre de coups dans la suite de coups.

Question Écrire une fonction `othellier* create_othellier()` qui crée un othellier en position initiale (Figure 1.1) et renvoie un pointeur sur l'othellier créé.

Question Écrire une fonction `void affiche_othellier(othellier p)` qui affiche sur la sortie standard d'erreur [2] l'othellier `p`.

Question Écrire une fonction `int est_possible(othellier p, coup c)` qui teste si le coup `c` est possible sur l'othellier `p`.

Question Écrire une fonction `partie coups_possible(othellier p, joueur j)` qui renvoie la liste des coups possibles du joueur `j` sur l'othellier `p`.

Question Écrire une fonction `void joue_coup(othellier p, coup c)` qui joue le coup `c` sur l'othellier `p`. On suppose que `c` est un coup légal par rapport à la position dans `p`.

Question Écrire une fonction `joueur continue_partie(othellier t, partie p)` qui joue la partie p sur l'othellier t. La fonction renverra le joueur qui a le prochaîn coup si p est une partie légale rélative à la configuration dans t, 0 si la partie n'est pas légale.

1. si `p` est un pointeur sur `othellier`, on accède à la case B3 de cet othellier avec `(*p)[1][2]`
2. Pour afficher `Hello World!` sur la sortie standard d'erreur on peut utiliser `fprintf(stderr, "Hello␣World%c\n", '!');` Attention vous devrez inclure `#include<stdlib.h>`

2 Implémentation du moteur du jeu

On s'intéresse maintenant à implémenter un moteur de jeu. Votre programme devra avoir ce comportement lorsqu'il sera appelé avec l'option `"--play"`.

Votre programme lira sur l'entrée standard des instructions, qui seront chacune exactement sur une ligne. Il devra pouvoir traiter 2 types de commandes différentes :

- des coups, de la forme `BA2`, `NE6`, `BPA`, etc. (qui indiquent respectivement un coup de blanc en `A2`, un coup de noir en `E6` ou blanc qui passe).
  Si le coup est valide votre programme devra afficher sur sa sortie standard une ligne commençant par `OK`,
  Si le coup est invalide, une ligne commençant par `ERREUR`, par exemple :
  - `ERREUR il y a deja un pion en A2`
  - `ERREUR Noir n a pas le droit de passer (il peut jouer en A3)`
- une des commandes suivantes :
  - `score`, il devra alors afficher une ligne commençant par `SCORE XXX` avec XXX la valeur du score [3]
  - `coups`, il devra alors afficher une ligne commençant par `COUPS` et contenant la liste des coups possibles
  - `affiche`, il devra alors afficher sur la sortie standard d'erreur la configuration du jeu
  - `newgame`, qui recommence la partie.

Vous êtes libres d'ajouter d'autres commandes qui vous sembleront utiles (par exemple `aide`, qui afficherait la liste des commandes disponibles).

Si la partie est terminée, il devra afficher une ligne commençant par `BLANC gagne` ou `NOIR gagne` selon lequel des deux joueurs a gagné et terminer.

Toutes les commandes peuvent en plus afficher ce qu'elles veulent sur la sortie standard d'erreur, mais rien d'autre que spécifié sur la sortie standard.

3 Implémentation d'une intelligence artificielle (IA)

Un peu de théorie. Dans ce jeu, les chances de victoire ne dépendent que de l'état de l'othellier et du joueur dont c'est le tour (nous appelerons l'ensemble de ces deux données une *position*). Ainsi, il n'est utile de considérer que les stratégies dites *positionnelles* ou *sans mémoire*, c'est-à-dire où le prochain coup à jouer ne dépend que de la position courante.

La théorie dit aussi qu'un tel jeu peut être totalement déterminé, c'est-à-dire que depuis chaque position, il est possible de calculer (mais ce calcul peut être affreusement long !) lequel des deux joueurs va gagner à supposer que les deux joueurs jouent de façon optimale (le calcul d'une stratégie optimale étant tout aussi long). Une manière de calculer la valeur des positions (ainsi que les stratégies optimales) est de donner la valeur `'B'` aux positions finales correspondant à des parties remportées par Blanc et `'N'` aux positions finales correspondant à des partries remportées par Noir, puis de propager aux autres positions P de la façon suivante :

- si en P, c'est à `'B'` de jouer et qu'un coup permet d'arriver dans une position de valeur `'B'`, alors P a la valeur `'B'` ;
- si en P, c'est à `'N'` de jouer et qu'un coup permet d'arriver dans une position de valeur `'N'`, alors P a la valeur `'N'` ;
- si en P, c'est à `'B'` de jouer et que tous les coups possibles mènent dans une position de valeur `'N'`, alors P a la valeur `'N'` ;
- si en P, c'est à `'N'` de jouer et que tous les coups possibles mènent dans une position de valeur `'B'`, alors P a la valeur `'B'`.

De proche en proche, toutes les positions obtiennent une valeur et on peut en déduire les stratégies optimales.

3. C'est le nombre de pions blancs moins le nombre de pions noirs sur l'othellier

En pratique. Comme le calcul d'une stratégie optimale peut être très long et coûteux en mémoire (et en outre il n'est pas intéressant de jouer contre une IA qui gagne tout le temps !), l'IA tentera de calculer la meilleure stratégie possible avec un horizon d'exploration borné par une constante N fixée. Ainsi, au lieu de propager les valeurs à partir des positions finales, l'IA propagera les valeurs des positions que l'on peut atteindre en, au plus, N coups.

Évidemment, si on n'explore pas jusqu'aux positions finales, la valeur exacte des positions explorées n'est pas connue. C'est pour cela qu'au lieu de calculer une valeur dont la signification est la certitude de victoire ou de défaite, l'IA calculera une estimation. Nous proposons la méthode d'estimation suivante :

- pour une position finale, l'estimation vaut 127 si c'est une victoire de Blanc et -128 si c'est une victoire de Noir ;
- pour les positions non-finales obtenues après N coups, l'estimation est le nombre de pions blancs moins le nombre de pions noirs (ce qu'on appelle aussi le *score*). Notez que si c'était une position finale, le signe de cette différence indiquerait le vainqueur. Comme la position n'est pas finale, ce n'est qu'une indication que la victoire semble plus ou moins plausible.
- pour les autres positions explorées, il faudra *propager* l'estimation par un algorithme dit de *min-max* :
  - s'il s'agit d'une position de profondeur $k-1$ pour laquelle c'est à Blanc de jouer, alors l'estimation de cette position est le **maximum** des estimations des positions que Blanc peut atteindre au coup suivant (donc à la profondeur k) ;
  - s'il s'agit d'une position de profondeur $k-1$ pour laquelle c'est à Noir de jouer, alors l'estimation de cette position est le **minimum** des estimations des positions que Noir peut atteindre au coup suivant (donc à la profondeur k).

Programmation de l'IA en C. L'IA devra stocker en mémoire toutes les positions explorées ainsi que les estimations calculées. Il est clair que depuis une position donnée, les futurs sont arborescents, les "fils" d'une position dans l'arbre des futurs étant les positions que l'on peut atteindre en un coup depuis cette position. La racine est la position courante dans la partie.

Structure de données proposée pour stocker l'arbre des coups (`mmtree`, pour *min-max-tree*) :

```
typedef struct mmtree {
  othellier *o; /* La position courante */
  joueur j;     /* Le prochain joueur   */
  int nb;       /* Le nombre de coups possible */
  coup *coups;  /* Le tableau des coups */
  struct mmtree *fils; /* Le tableau des fils */
  int valeur;   /* La valeur attribuee a cette position */
  int meilleur; /* L'indice du meilleur coup jouable a cette position */
} mmtree;
```

Ici, `o` et `j` représentent la position courante, `coups` donne la liste des coups jouables ici et `fils` la liste des positions suivantes obtenues par ces coups (une position par coup, en respectant l'ordre du tableau `coups`), `nb` est la longueur de ces deux tableaux, enfin `meilleur` et `valeur` sont des champs qui seront initialisés (ou mis à jour) après exécution de l'algorithme min-max, `valeur` étant la valeur de l'estimation expliquée plus haut et `meilleur` l'indice du fils ayant la meilleure valeur (et donc aussi l'indice du coup permettant d'atteindre cette position).

Question Écrire la fonction

```
mmtree* compute_tree(othellier o, joueur j, int depth);
```

qui développe l'arbre d'exploration à la position actuelle `(o, j)` jusqu'à la profondeur `depth`. Remarquez que si `depth` vaut 0, cela revient à créer une feuille. Son estimation sera la valeur du score, et par convention le nombre de fils est 0.

Remarquez que l'arbre est créé depuis la racine vers les feuilles, alors que la propagation de l'estimation et la détermination du meilleur coup par l'algorithme min-max suit le chemin inverse (des feuilles vers la racine).

Il sera possible de réaliser cette seconde opération soit dans la même fonction, soit dans une autre fonction récursive.

Question Écrire la fonction

```
void free_tree(mmtree *t);
```

qui libère récursivement la mémoire utilisée par l'arbre `*t`.

Question Enfin, écrire une fonction `main` qui permette à un joueur humain de se mesurer à l'IA définie plus haut. Le programme reconnaîtra le paramètre de la ligne de commande `--blanc` (resp. `--noir`) qui lui indique s'il doit se comporter en tant que joueur Blanc (resp. Noir), auquel cas il lira les coups de Noir (resp. Blanc) sur l'entrée standard et imprimera les coups de Blanc (resp. Noir) sur la sortie standard en suivant le format décrit dans la Section 2 :

- si l'IA est Noir, il commencera la partie par `OK`, suivi d'un coup ;
- il répondra au coup de l'adversaire soit par `OK` suivi de son coup suivant, soit par `ERREUR` si le coup lu n'était pas valide.

Par ailleurs, le programme devra pouvoir répondre aux commandes `score, coups, affiche,` et `newgame`.

Indication : on vous conseille de demander à l'IA de calculer sa stratégie en explorant à une profondeur de 8 coups, ce qui est un bon compromis (mieux que ce qu'un humain moyen peut faire, tout en étant raisonnable en termes de coût de calcul).

Pour aller plus loin : quelques optimisations. Améliorer les fonctions de cette partie afin d'éviter de recalculer tout l'arbre des N coups suivants à chaque tour. Pour cela, il s'agira, à chaque tour, en un premier temps, de récupérer la branche correspondant au coup effectivement joué (et de libérer les parties de l'arbre qui ne sont plus utiles), puis de compléter l'arbre en ajoutant des fils aux feuilles restantes (qui sont maintenant de profondeur $N - 1$).

Pour aller encore plus loin : autres heuristiques d'estimation. Votre programme estime actuellement le score des feuilles en faisant la différence du nombre de pions de chaque couleur. Or il est aisé de se rendre compte que la position des pions est souvent plus importante que leur nombre. Par exemple, les pions périphériques ont une plus grande influence.

Essayez d'autres façons d'évaluer une position, et faites jouer votre nouveau programme contre votre première version. Qui gagne ?