Université Paris 7 - DEUG-MIAS Année 2004-2005

Projet de *Logique et Circuits*

Dates

Ce projet doit être résolu par groupes de deux personnes. A l'issue du projet, vous devrez fournir un rapport et les sources du programme en OCaml. Note bien les deux dates importantes :

La semaine du 4 avril, en TP, vous devrez rendre le rapport et les sources pour les parties a) et b) ;

La semaine du 15 mai, vous devrez rendre le rapport final et les sources pour tout le projet. Des soutenances seront organisées cette semaine là. Les chargés de TP communiqueront les horaires de passage en temps voulu.

Modalités

Dans l'évaluation des projets, il sera accordé la plus grande attention aux points suivants :

Spécification : chaque fonction sera accompagnée de son type et d'une phrase en français expliquant sont rôle et la nature des ses arguments.

Réalisation : correction des algorithmes, considération de cas exceptionnels, choix des opérations.

Codage : lisibilité du code et donc, en particulier, choix pertinent des noms, indentation, commentaires.

Jeux de tests : cas exceptionnels, cas quelconques mais non triviaux.

Rapport : bonne structuration et présentation du problème.

Soutenance : chaque étudiant devra être bien familiarisé avec son programme et il devra être capable de le faire tourner sur un jeu de test arbitraire fourni par son enseignant.

Chacune des fonctions demandées dans l'énoncé doit être écrite, mais rien ne vous interdit de définir d'autres fonctions ou d'autres types si celà vous parait pertinent.

I) Présentation du jeu

Othello est un jeu de stratégie à deux joueurs (noir et blanc). Il se joue sur un plateau de 8 cases sur 8. Les joueurs disposent de pions bicolores (en quantité suffisante), noirs sur une face et blancs sur l'autre. La couleur d'un pion est celle de sa face visible. Le but du jeu est d'avoir plus de pions de sa couleur que l'adversaire à la fin de la partie.

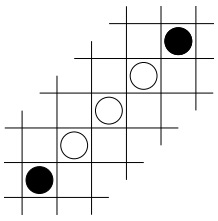


FIG. 1: Un encadrement diagonal par les noirs.

Le principe du jeu est de retourner les pions de l'adversaire en les encadrant. Un encadrement est un alignement de pions d'une couleur avec à chaque bout un pion de l'autre couleur, sans case vide intermédiaire. L'alignement peut être horizontal, vertical ou diagonal (voir figure 1). Lorsque la pose d'un pion du joueur crée un encadrement de pions de l'adversaire, ces pions sont retournés et deviennent donc de la couleur du joueur.

Au début de la partie, les quatre cases centrales sont occupées par deux pions de chaque couleur, comme indiqué figure 2. Les joueurs jouent ensuite à tour de rôle en plaçant un pion de leur couleur sur une case vide. Un coup est légal s'il crée au moins un encadrement, et dans ce cas tous les encadrements créés par la pose du pion sont retournés. Par exemple, dans la figure 3, la pose d'un pion noir retourne deux lignes de pions blancs. Si un joueur ne peut pas jouer de coup légal, il doit passer son tour, autrement il est obligé de jouer. La partie s'achève lorsqu'aucun des deux joueurs ne peut plus jouer de coup légal. En général, cela intervient soit lorsque les 64 cases sont occupées, soit lorsque tous les pions sont de la même couleur.

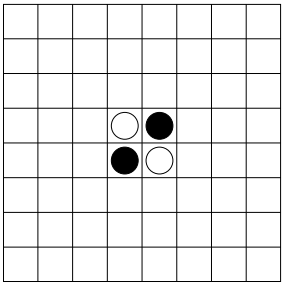
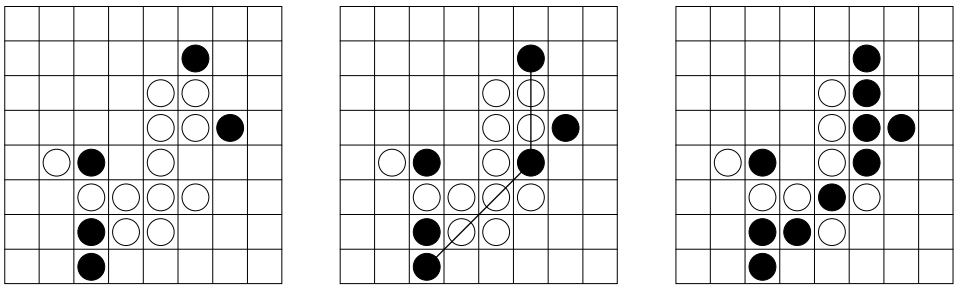


FIG. 2: Position initiale.



(a) Position de départ. (b) Pose d'un pion. (c) Retournement.

FIG. 3: Exemple de pose d'un pion.

II) Plan du projet

Le but du projet est d'écrire un programme pour jouer à Othello. La première partie consiste à définir les structures de données appropriées pour représenter le jeu. Dans la deuxième partie, on écrit un programme qui permet de jouer à deux, et le programme vérifie que les coups sont valides et retourne les pions. Dans la troisième partie, on étend le programme pour pouvoir jouer contre l'ordinateur, en définissant un algorithme pour calculer des coups intéressants.

a) Structures de données

Le but de cette partie est de modéliser les éléments nécessaires au jeu :
- les 2 joueurs, blanc et noir ;
- le plateau de jeu, de 64 cases qui sont soit vides, soit occupées par un pion d'un des joueurs.

1. Définir les types « `joueur` », « `case` », « `plateau` ». On ajoutera également le type « `position = int * int` » afin de représenter une position du plateau.
2. Définir une fonction

   `valeur_case : plateau → position → case`

   qui, pour un plateau et une postion donnés, renvoie la case du plateau à cette position.
3. Définir une fonction

   `modifie_case : plateau → position → case → plateau`

   tel que « `modifie_case pl pos c` » renvoie le plateau correspondant a « `pl` » dans lequel on a remplacé la case à la position « `pos` » par « `c` ».
4. Définir le plateau initial, c'est à dire un plateau représentant la configuration de la figure 2.
5. Définir une fonction

   `liste_pions : plateau → joueur → position list`

   qui renvoie la liste des positions des pions d'un joueur donné.
6. À l'aide des fonctions graphiques fournies et décrites dans l'annexe, écrire un programme qui affiche le plateau initial et attend que l'utilisateur clique.

b) Jeu à deux joueurs

On désire maintenant permettre à un joueur humain de jouer un coup. Pour cela, on doit être capable de vérifier si un coup est légal, c'est-à-dire s'il retourne au moins un pion adverse.

Lorsque l'on place un pion, on peut retourner une ligne de pions adverse pour chacune des huit directions. On va donc abstraire la direction dans les fonctions de vérification d'un coup valide et de retournement d'une ligne de pions.

7. Définir un type « `direction` » pour représenter les différentes directions, puis

   `liste_directions : direction list`

   la liste de toutes les directions.

8. Écrire une fonction

   `avance : position → direction → position`

   qui donne la position voisine d'une position donnée selon une direction.

9. Écrire une fonction

   `teste_ligne : plateau → joueur → position → direction → int`

   qui compte le nombre de pions que le coup d'un joueur à une position donnée retournerait dans la direction donnée.

10. Écrire une fonction

    `coup_légal : plateau → position → joueur → bool`

    tel que « `coup_légal pl pos j` » soit vrai si « `pos` » représente un coup légal pour le joueur « `j` » sur le plateau « `pl` », et faux sinon.

11. Écrire une fonction

    `retourne_direction : plateau → position → joueur → direction → plateau`

    qui retourne les pions qui doivent être retournés dans la direction donnée pour le coup d'un joueur donné, et renvoie le plateu obtenu. On fera attention au fait qu'il n'y a pas nécessairement des pions à retourner dans toutes les directions.

12. Écrire une fonction

    `jouer_coup : plateau → position → joueur → plateau`

    qui renvoie le plateau résultant d'un coup supposé légal d'un joueur donné.

13. Avec ces fonctions, écrire un programme pour jouer à 2 joueurs à l'Othello.

c) Jeu contre la machine

On désire maintenant faire jouer l'ordinateur. Pour cela, on va développer un algorithme qui choisit un coup pour une configuration de jeu donnée. On veut trouver un coup qui soit bien sûr valide, mais aussi le "meilleur possible".

Pour cela, mettons-nous à la place de l'ordinateur. La première chose à faire est de savoir quels sont les coups possibles (afin d'en choisir le meilleur).

14. Écrire une fonction

    `coups_possibles : plateau → joueur → (position * plateau) list`

    telle que « `coups_possibles pl j` » renvoie la liste des coups que le joueur « `j` » peut jouer sur le plateau « `pl` ». Les éléments de la liste sont des couples « `(pos, pl')` » où « `pos` » est la position où une pièce peut être posée et « `pl'` » est le plateau obtenu.

On veut maintenant comparer ces coups. On va donc évaluer les différentes configurations obtenues en jouant ces différents coups. On veut leur attribuer une sorte de "note" qui, plus elle est élévée, plus elle correspond a une configuration qui nous est favorable. Et puisque le but de l'Othello est d'avoir le maximum de pions de sa couleur sur le plateau, une évaluation naïve consiste à compter notre nombre de pions.

15. Écrire une fonction

    `évalue : plateau → joueur → int`

    telle que « `évalue pl j` » renvoie le nombre de pions de « `j` » dans « `pl` ».

On peut maintenant évaluer toutes les configurations où nous envoie les différents coups possibles et choisir le maximum.

Toutefois, de cette façon, on n'anticipe aucun coup et puisque, rappelons-le, nous sommes un ordinateur, nous sommes plutôt doué pour le calcul. On va donc maintenant maintenant anticiper un certain nombre de coups afin d'éviter par exemple de jouer un coup qui retourne beaucoup de pions mais qui permet à l'adversaire de les reprendre aussitôt.

Pour cela, on va implémenter l'algorithme dit de "min-max". Cet algorithme simule un certain nombre de coups successifs en envisageant toutes les possibilitées et utilise à chaque étape l'idée suivante :

- Si c'est à l'ordinateur de jouer, il jouera le coup qu'il considère le meilleur, donc celui qui mène au plateau qui a le meilleur score. Dans ce cas la note du plateau est le maximum des notes des plateaux obtenus après un coup de l'ordinateur.
- Si c'est à l'adversaire de jouer, on suppose qu'il va jouer de la façon la plus défavorable à l'ordinateur, donc la note du plateau est le minimum des notes des plateaux obtenus après un coup de l'adversaire.

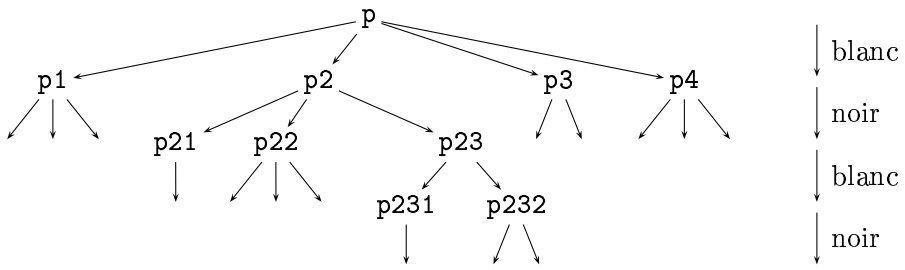
Dans les deux cas, on déduit la note d'un plateau à partir de la note des plateux obtenus après un coup. Ces notes sont elles-mêmes calculées *récursivement* de la même façon. Pour limiter la quantité de calculs, on décide de limiter la recherche en s'arrêtant après avoir simulé un certain nombre de coups de suite. Une fois atteint ce nombre de coups, la note attribuée au plateau est calculée avecla fonction « `évalue` » définie précédemment.

16. Écrire deux fonctions

    `évalue_min : plateau → joueur → int → int`
    `évalue_max : plateau → joueur → int → int`

    telles que « `évalue_max pl j n` » évalue la note d'un plateau pour le joueur « `j` » dans le cas où c'est à lui de jouer, et « `évalue_min pl j n` » évalue cette note si c'est à l'adversaire de jouer. Dans les deux cas, « `n` » est le nombre de coups de suite à simuler.

Grâce aux fonctions précédentes, on sait maintenant évaluer un plateau de jeu de façon plus intelligente qu'en ne faisant que compter les pions. On peut donc en déduire une fonction qui calcule le meilleur coup à jouer dans une situation donnée.



On suppose que c'est au joueur blanc de jouer sur le plateau « `p` », il a quatre coups possibles qui mènent au plateaux « `p1` » à « `p4` », puis sur « `p2` » le joueur noir a trois coups possibles qui mènent à des configurations « `p21` » à « `p23` », puis c'est au joueur blanc de jouer, et ainsi de suite ...

FIG. 5: Exemple d'arbre de coups possibles.

17. Écrire une fonction

    ```
    meilleur_coup : plateau → joueur → int → position
    ```

    telle que « `meilleur_coup pl j n` » renvoie le coup du joueur « `j` » qui mène au plateau qui a la meilleure note possible, en simulant « `n` » coups de suite.
18. Modifier le programme de la partie précédente pour qu'il permette à un joueur de jouer contre l'ordinateur.

Annexe

Pour l'interface graphique du jeu, nous vous fournissons deux fonctions :

- La fonction

  ```
  Ig.affiche_plateau : (int * int) list → (int * int) list →
      unit
  ```

  qui prend 2 listes de positions l_n et l_b qui sont respectivement la liste des pions noirs et la liste des pions blancs présents sur le plateau, et affiche l'interface de jeu.
- La fonction

  ```
  Ig.lire_coup : unit → (int * int)
  ```

  qui attend que l'on clique sur une des cases du plateau de l'interface de jeu et renvoie la case où l'on a cliqué.

Pour utiliser ces fonctions dans votre programme, vous devez, lors de la compilation de votre programme, le lier à l'interface graphique. Si par exemple le fichier qui contient votre programme s'appelle « `othello.ml` », vous devez le compiler avec la commande

```
ocamlc -o othello graphics.cma ig.cmo othello.ml
```

(ce qui aura pour effet de donner le nom « `othello` » à votre programme).