UE INF5011 Programmation 3

Programmation Fonctionnelle et Symbolique

Projet

À remettre le vendredi 13 Décembre 2013

Jeu d'*Othello*

Othello est un jeu de stratégie sur othellier (tablier de 8×8) opposant deux joueurs, ayant chacun une couleur : noir ou blanc.
Chacun à leur tour, les joueurs posent un pion (s'il le peut) suivant les règles décrites ci-dessous, jusqu'à ce qu'aucun des deux joueurs ne puisse plus jouer. Celui ayant le plus grand nombre de pions de sa couleur à la fin de la partie est alors déclaré vainqueur.
Le tablier est numéroté. Par convention on utilisera les lettres `a` à `h` pour l'abscisse et les chiffres `1` à `8` pour l'ordonnée.
Initialement, deux pions noirs sont placés en `d5` et `e4`, et deux pions blancs sont placés en `d4` et `e5`.
Lorsqu'un joueur joue, il *capture* tous les pions de l'adversaire qui se retrouvent en ligne directe entre le pion qui vient d'être joué, et un pion de sa couleur déjà présent. Les segments valides sont horizontaux, verticaux et/ou diagonaux. Tous les pions ainsi entourés prennent alors la couleur du joueur en train de jouer. Il n'y a pas d'effet de cascade, c'est-à-dire que l'on ne considère que les alignements formés par le pion nouvellement posé, et pas les captures qu'il produit.
Un joueur peut jouer s'il peut capturer au moins un pion adverse.
Le joueur noir commence la partie.

Vous pouvez vous référer à la page wikipedia de Reversi pour de plus amples détails : `https://en.wikipedia.org/wiki/Reversi`Reversi – Wikipedia.

1 *Othello* interactif

Dans cette première partie, il est demandé de réaliser une interface en mode texte pour le jeu d'*Othello*. On demandera aux joueurs, chacun leur tour, la position où ils désirent jouer (entrée sous forme d'une chaîne `"c4"` ou `"b6"` sans les guillemets). S'il s'agit d'une position valide, on affichera le nouveau plateau de jeu et on passera au joueur suivant. Une position invalide devra être signalée au joueur de manière appropriée.

Modalités

Le point d'entrée du programme sera une fonction `othello ()`, ne prenant aucun paramètre et se terminant avec la partie.
Vous rendrez pour cette partie un fichier (ou un ensemble de fichiers) `.lisp`.

2 Stratégies non interactives

On souhaite un programme pour faire jouer l'ordinateur, soit contre un joueur humain, soit contre un autre programme.

2.1 Stratégie aléatoire

Pour mettre en place la possibilité d'avoir des stratégies non interactives, commencer par implémenter une stratégie aléatoire : le programme joue à une position aléatoire parmi les positions valides.

2.2 Intelligence artificielle

On souhaite dans un second temps de réaliser une **intelligence artificielle** pour le jeu.
Vu sa complexité, l'implémentation d'un algorithme de type *minimax* est fortement conseillée.
Le principe de ce dernier est simple : étant donné un plateau de jeu, il part du constat que le meilleur coup à jouer est celui qui amène à la victoire en fin de partie. On suppose que l'adversaire joue lui aussi parfaitement. On peut alors parcourir un arbre virtuel qui a pour racine le plateau de jeu actuel, avec pour fils l'ensemble des coups possibles sur un tel plateau de jeu, chacun de ces fils aura pour fils l'ensemble des coups possibles de l'adversaire sur un tel plateau, et ainsi de suite.
Bien évidement, un tel arbre croit exponentiellement, et l'espace de recherche est bien trop grand pour être parcouru en entier. Il va donc falloir fixer une profondeur maximale à explorer. Et dans ce cas, les feuilles de l'arbre ne sont plus uniquement des situations de fin de partie, mais peuvent également être des situations intermédiaires.
C'est là qu'il faut user d'ingéniosité en réalisant une heuristique permettant d'attribuer un score à chaque situation du jeu ; elle attribuera un score maximal en cas de fin de partie avec gain, minimal en cas de fin de partie avec perte, et un score situé entre ces deux bornes à toute autre situation de jeu. Pour ce faire, on peut constater, par exemple (et il y a beaucoup d'autres critères possibles), qu'il est plus intéressant d'avoir des pions près du bord (un pion au bord du plateau ne peut être capturé que depuis ce même bord) et encore plus dans les coins (un pion dans un coin ne peut plus être capturé). Les règles utilisées par l'heuristique devront apparaître clairement dans le code.
Une fois l'heuristique replacée dans le contexte du `minimax`, on peut alors évaluer toutes les feuilles intermédiaires (les plateaux à une profondeur maximale), et choisir le coup qui amène à la situation apportant un maximum de points.

2.3 Tournoi

Une petite partie de l'évaluation du projet se fera en faisant jouer les stratégies les unes contre les autres à l'aide d'un programme appelé `tournoi` écrit par les enseignants.
Le programme `tournoi` vérifiera que les coups proposés par une stratégie sont des coups valables, et les transmettra à la stratégie adverse.
Par conséquent, il est très important de respecter les contraintes sur les entrées-sorties, c'est-à-dire ne rien écrire d'autre que les coups que vous désirez jouer (sans même des retours à la ligne). En revanche, votre programme devrait être robuste en lecture, et prendre en compte toute forme d'espace blanc dans les coups joués par l'adversaire [1].
De plus, les IA devront répondre en moins d'une seconde [2].
Une IA ne respectant pas les formats d'entrée-sortie, les règles du jeu, ou ne répondant pas dans le temps imparti sera automatiquement disqualifiée.

Modalités

Pour cette partie, on demande **un seul** fichier `.lisp` (qui pourra être obtenu par concaténation d'un ensemble de fichiers `.lisp` convenablement commentés).
Le point d'entrée du programme sera une fonction nommée `main-standalone (first)` prenant en paramètre un unique booléen précisant le fait qu'on joue en premier (noir) ou en second (blanc).
Si on joue en premier, alors la fonction devra commencer par écrire sur la sortie standard la position jouée, sinon elle devra lire la position jouée par l'adversaire sur l'entrée standard.
La fonction `main-standalone` doit se terminer lorsque la partie est terminée.
Les lectures et les écritures doivent se faire caractère par caractère. Il est interdit d'écrire autre chose que la position jouée sur la sortie standard.

3 Modalités générales

1. Le travail doit être réalisé en trinômes constitué au sein d'un même groupe de TD.
2. L'ensemble des fichiers associés au projet doit être géré avec un outil de gestion de révision (`svn`, `git`, ...).
3. Chaque enseignant de TD proposera sa solution pour la remise du projet (archive, mail, soutenance,...).
4. La plus grande partie de l'évaluation portera sur la qualité et l'organisation du code.

[1] C'est un principe général de programmation : rigoriste lors de l'écriture, laxiste lors de la lecture.
[2] On considérera les machines utilisées en TD comme référence de performance.

Informations complémentaires

On pourra utiliser les fonctions lisp `read-char` et `write-char` pour respectivement la lecture des coordonnées et l'écriture des coordonnées.
Les fonctions lisp `char-code` et `code-char` peuvent être utilisées pour respectivement la conversion d'un caractère vers l'entier `ASCII` correspondant et d'un entier `ASCII` vers le caractère associé. On rappelle qu'en `ASCII` les codes des lettres lettres sont contigus : le code de `b` est celui de `a + 1`, celui de `c` est celui de `a + 2`, etc...