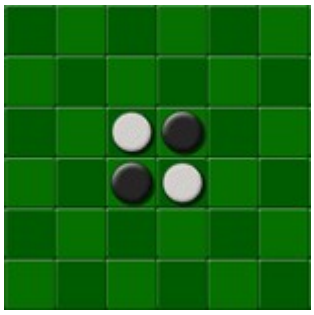
Informatique 2eme année
2009/2010

# Othello en PROLOG

V. BEAUFUME
V. BOYI-BANGA
M. K. KOCHBATI

# 1- Principe d'Othello

Le Jeu d'Othello ,étudié dans cet exemple est un jeu a deux joueurs, sur un damier 6x6 avec des pions bicolores (une face blanche, une face noire). Chaque joueur est lié a une couleur et cherche a obtenir le plus de pions possible.
Le tableau initial est le suivant :



A chaque tour un joueur joue en posant un pion de sa couleur sur une case du damier. Le coup ne peut être joué si:

- Le pion est posé sur une case vide
- Des pions de la couleurs adverse sont pris en alignement entre deux pions de la couleur alliée.

Si un joueur ne peut pas jouer il passe son tour.
La partie termine quand:

- Les deux joueurs ont passé chacun leur tour
- Le tableau est complètement rempli

A ce moment le joueur qui remporte la partie est celui qui possede le plus de pions de sa couleur.

# 2- Représentation Choisie

## *Le Plateau P*

Nous avons choisi de représenter le tableau de jeu par une liste de taille 36.La liste sera de la forme [X11,X12,X13,X14,X15,X16,
X21,X22,X23,X24,X25,X26,
X31,X32,X33,X34,X35,X36,
X41,X42,X43,X44,X45,X46,
X51,X52,X53,X54,X55,X56,
X61,X62,X63,X64,X65,X66 ]
ou chacune des variables correspond a une case du tableau de jeu. Cette représentation permet un accès facile aux données de jeu grâces aux fonctions nth0 et nth1 pré-codées dans « swi-prolog ».
Chaque case du plateau peut avoir trois valeurs :

- 0 si cette case est vide
- (-1) si le pion est noir
- 1 si le pion est blanc

Ces valeurs permettent de passer de l'une a l'autre en une simple opération.
Le fait que l'on ne laissent pas les valeurs vides ne pose pas de problème car lorsque l on joue un coup on crée une liste de taille 36 ne contenant que des variables que l'on renverra alors.

*Heuristique utilisée*

L'heuristique utilisée est un algorithme d'alpha-bêta auquel on applique notre fonction d'évaluation. La fonction d'évaluation utilisée est une fonction simple qui calcule le nombre de pions alliées – le nombre de pions adverses. Cette heuristique renvoie de très bon résultat quand on augmente la profondeur a laquelle on l'utilise.

# 3- Description des fonctionnalités

Dans cette partie nous décrirons le codes et expliqueront les fonctionnalités.

*> couleur(+P,-V,+X,+Y,+C)*

P plateau
V valeur de la couleur (-1 noir 1 blanc ou 0 vide)
X position de la colonne (1 a 6)
Y position de la ligne (1 a 6)
C compteur a initialisé a 1

Cette fonction renvoie la valeur de la liste du plateau de jeu a la position (X,Y). On effectue également un test pour vérifier que la case que l'on veut obtenir existe bien dans un plateau 6x6.

*> coup_valide(+P,+C,+X,+Y)*

P plateau
C couleur du coup (-1 noir 1 blanc)
X position de la colonne (1 a 6)
Y position de la ligne (1 a 6)

Cette fonction teste si la case (X,Y) du plateau P est un coup valide pour la couleur C.
On teste tout d'abord si la case (X,Y) est une case vide. Puis on teste pour toutes les directions possible si une case adjacente appartient a l'adversaire. Si c'est le cas on lance la fonction de boucle correspondante (qui correspond a la direction dans laquelle un coup peut être possible). Si une de ces fonctions de boucle termine alors coup_valide termine.

*> c_x_xx(+P,+C,+X,+Y)*

P plateau
C couleur du coup (-1 noir 1 blanc)
X position de la colonne (1 a 6)
Y position de la ligne (1 a 6)

Ces fonctions (c_h_d , c_h_g , c_v_h , c_v_b , c_d_db , c_d_dh , c_d_gh , c_d_gb )sont les fonctions de boucles décrites précédemment.
Ces fonctions échouent si :

- Lors du parcours on arrive a une case vide
- Lors du parcours on sort du tableau

Ces fonctions terminent quand le coup permet de prendre en sandwich une ligne

*> tout_coup_legaux(+C,+P,-R)*

P Plateau
C Couleur
R Liste des coups valides

Cette fonction calcule les coups possibles pour la couleur C a partir du plateau P.
On utilise pour cela la fonction findall( ...) pour obtenir toutes les valeurs pour lesquels lister_coup est valide.
La liste renvoyée est de la forme [[X1,Y1],[X2,Y2],[X3,Y3],....] ou Xi et Yi correspondent au coordonnées d'un coup possible.

*> lister_coup(+P,+C,+[X,Y])*

P Plateau
L liste de renvoi
[X,Y] position du coup

Cette fonction termine si coup_valide(P,C,X,Y) termine.
On a pour cela énuméré toutes les cases possible ou coup_valide pourrait terminer.

*> joue_coup(+C,+P,+[X,Y],-NP)*

X position de la colonne (1 a 6)
Y position de la ligne (1 a 6)
C Couleur
P plateau
NP nouveau plateau renvoyé

Cette fonction permet de modifier le damier après avoir joue un coup.
Tout d'abord on teste si le coup est valide pour la case [X,Y]. Si c'estle cas on crée une liste vide de taille 36 "NP" dans laquelle on rajoute le pion de la couleur C a la position [X,Y].
On appelle alors retourne_case(P,C,[X,Y],NP) qui va ajouter les case retourné au tableau NP , puis on appelle completer(P,NP) qui va remplacer toutes les variables de NP (cases que 'on n a pas modifié lors du retournement des pions) par les valeurs contenues dans les cases correspondantes de P

*> retourne_case(+P,+C,+[X,Y],-NP)*

X position de la colonne (1 a 6)
Y position de la ligne (1 a 6)
C Couleur
P plateau
NP nouveau plateau renvoyé

Cette fonction va modifier NP de telle sorte a ce que tout les pions qui sont retourné par le coup [X,Y] testé valide soit ajouter au plateau NP.
On va pour cela tester si un coup est valide dans une direction (en utilisant les boucles utilisés avec coup_valide) et on va compléter le tableau en utilisant d'autres fonctions de boucles basés sur le même principe que précédemment mais qui vont modifiés NP.
On ferra attention a bien tester toutes les directions.

*> modifier_x_xx(-NP,+P,+C,+X,+Y)*

X position de la colonne (1 a 6)
Y position de la ligne (1 a 6)
C Couleur

P plateau
NP nouveau plateau renvoyé

Ces fonctions (modifier_h_d , modifier_h_g , modifier_v_h , modifier_v_b , modifier_d_db ,modifier_d_dh ,modifier_d_gh , modifier_d_gb )sont les fonctions de boucles décrites précédemment dans retourner_case.
Elles retournent toutes les cases jusqu'à atteindre la case de couleur alliée

*> completer(+P,-NP)*
P plateau
NP nouveau plateau

Cette fonction va remplacer les cases ou aucune valeurs ne sont assignés dans NP par les valeurs de P correspondantes.
On a pour cela énuméré toutes les cases possible ou il pourrait y avoir une case vide (non assigné).

*> fonct_eval(+P,+C,-V)*
P plateau
C couleur
V valeur retour

Ceci est la fonction d'évaluation qui sera utilisé dans l'algorithme de alpha-bêta.
Cette fonction calcule le nombre de pions que l'on possède moins ceux que possédé l'adversaire. Le nombre de pions possède par un joueur est calculer par la fonction nb_case(P,C,V,CP)

*> nb_case(+P,+C,-V,+CP)*
P plateau
C couleur a compter
V valeur retour
CP compteur initialise a 1

Cette fonction calcule le nombre de pions qu'un joueur associé a la couleur C possède sur le plateau P. On est pour cela obligé de parcourir tout le plateau

*> alpha_beta(+C,+PR,+P,+A,+B,-MCH,-V)*
C couleur du joueur
PR la profondeur que l'on va utiliser (dépend de la machine sur laquelle on lance l'algo)
> On utilisera ici 3 par défaut
P Plateau
A Valeur de Alpha
B Valeur de Bêta
MCH coup que l'on a choisi
V Valeur que l'algo de l'Alpha-Bêta va retourner

Cette fonction est une implémentation de l'algorithme de l'alpha-bêta en prolog que nous avons vue en cours. Cette fonction renvoie la valeur de la fonction d'évaluation si la profondeur atteinte est nulle, sinon on va créer la liste des coup possible (tout_coup_legaux) a partir de laquelle on appellera la fonction evaluate_and_choose(C,LIST,P,PR-1,-1*A,-1*B1,_,(MCH,V))) qui continuera l'algorithme de l'alpha-Bêta en choisissant le meilleur coup et en renvoyant sa valeur de la fonction

d'évaluation associée

*> evaluate_and_choose(+C,+LIST,+P,+PR,+A,+B,-CT,-CFIN)*

C couleur du coup
LIST Liste des coups possibles
P Plateau
PR profondeur
A valeur d alpha
B valeur de bêta
CT coup choisi temporaire
CFIN duet contenant le coup retenu (le meilleur) et sa valeur associé

C'est ici que l'on va tester les différentes possibilités pour chacune des profondeur de l'arbre alpha-bêta. On va jouer_coup pour chacun des coups possibles de la liste LIST avant de lancer l'alpha-bêta pour chacune des valeurs possible, en effectuant des coupe (avec la fonctionnalité coupe(C,COUP,V,PR,A,B,LIST,P,CT,CFIN) )quand nécessaire afin de gagner du temps.
On sélectionne alors la meilleure valeur (position + valeur alpha-bêta) que l'on renvoit

*> coupe(C,COUP,V,PR,A,B,LIST,P,CT,CFIN)*

C couleur du coup
V valeur actuelle retenue par l algorithme d'alpha_beta
COUP coup choisi actuel
LIST Liste des coups possibles
P Plateau
PR profondeur
A valeur d alpha
B valeur de bêta
CT meilleur coup choisi temporaire
CFIN duet contenant le coup retenu (le meilleur) et sa valeur associé

Cette fonction effectue les coupes nécessaires pour l'algorithme de l'alpha-bêta.

*> joueur_rand(+P,+C,-T,-NP)*

P plateau
C couleur
T coup joué
NP nouveau plateau après retournement des cases

Cette fonction correspond au joueurs aléatoire. On récupère d'abord tout les coups possibles on effectue alors un test sur la taille de la liste récupérées si celle ci est vide (c'est a dire que l'on doit passer) alors on renvoie le coup [0,0] et on ne modifie pas le plateau.
Sinon on choisit la case N ( qui est définit aléatoirement) de la liste des coups possibles et on joue ce coup.

*> joueur_heuristique(+P,+C,-T,-NP)*

P plateau
C couleur
T coup joué
NP nouveau plateau après retournement des cases

Cette fonction correspond au joueurs heuristique. On récupère d'abord tout les coups

possibles on effectue alors un test sur la taille de la liste récupérées si celle ci est vide (c'est a dire que l'on doit passer) alors on renvoie le coup [0,0] et on ne modifie pas le plateau.
Sinon on lance la fonctionnalité evaluate_and_choose avec la liste des coup possible et des valeurs grande pour alpha et bêta qui va nous choisir le coup optimal.
On jouera alors ce coup.