

AZENKOUD Younès et DAMON Régis
IMAC1 Janvier 2007

Rapport de projet C

OTHELLO



Table des matières

I – PRÉSENTATION DU PROJET

1.1 Avant-propos

1.2 Le jeu Othello

1.2.1 Un peu d'histoire

1.2.2 Règles du jeu

1.3 Cahier des charges

1.3.1 Graphisme primaire

1.3.2 Gestion des parties

1.3.3 Déroulement d'une partie

1.3.4 Le programme

II – IMPLÉMENTATION DU JEU

2.1 Notre code

2.2 Notre graphisme

III – INTELLIGENCE ARTIFICIELLE

3.1 Jouer contre l'ordinateur

3.2 Réflexion sur une direction possible

V – CONCLUSION

1.1 Avant-propos

Notre projet consistait en la réalisation d'un jeu de stratégie, l'Othello, bien connu du grand public. Nous devions le réaliser en mode graphique avec la libMlv, bibliothèque complète fournie par le professeur.

L'ensemble du projet a été réalisé en langage C, performant pour ce type de jeu.

La création de notre jeu s'est donc effectuée en deux grandes parties : dans un premier temps, la gestion technique des différents mécanismes de base d'Othello. Ensuite, nous devions fournir la possibilité de jouer seul contre l'ordinateur. Cela impliquait donc de mettre en place une intelligence artificielle.

1.2 Le jeu Othello

1.2.1 Un peu d'histoire

Le jeu d'Othello nous vient du Japon, il a été inventé au début des années 70 par Goro Hasegawa. En fait, l'ancêtre de ce jeu est un jeu anglais mis au point une centaine d'années auparavant : Reversi. Le nom même d'Othello est un hommage à ses origines. Il n'y a que très peu de différences entre Othello et l'original, mais lui, devient un énorme succès au Japon et envahit en une dizaine d'années le monde entier!

1.2.2 Règles du jeu

Il s'agit d'un jeu de plateau qui se joue à deux, le but du jeu est de posséder, au terme de la partie, plus de pions que son adversaire. Le plateau est composé de huit colonnes et de huit lignes, soit un total de soixante-quatre cases. Chaque pion possède une face noire et une face blanche.

Cette particularité vient du fait que tout au long de la partie, les pions ne vont cesser d'être retournés, faisant alternativement croître et décroître le nombre de pions appartenant à chaque joueur.

La pose d'un pion entraîne un retournement des pions de l'autre couleur, lorsque ceux-ci se retrouvent pris entre deux pions adverses. Ceci peut s'effectuer dans toutes les directions, y compris en diagonale. Il est à noter qu'un joueur n'a le droit de poser un pion que si la pose de ce pion entraîne effectivement un retournement de pions adverses.

Au début de la partie, deux pions de chaque couleur sont positionnés au centre du plateau, de manière systématique. C'est ce qui permet au premier joueur de poser un pion.

Si à un moment donné, un des joueurs ne peut pas poser de pion, c'est-à-dire s'il ne peut pas retourner de pions à l'adversaire, il doit passer son tour; jusqu'à ce qu'il puisse enfin jouer de nouveau: en effet, les coups permis doivent forcément entraîner

la prise de pion(s) de l'adversaire. Si à un moment donné, aucun des deux joueurs ne peut jouer, la partie est finie et le compte des pions peut s'effectuer.

1.3 Cahier des charges

1.3.1 Graphisme primaire

Le premier travail à effectuer est d'élaborer une interface graphique comprenant les éléments de base d'Othello, c'est-à-dire le plateau et les pions. Nous souhaitons également ajouter un compteur affichant en permanence le nombre de pions blancs et le nombre de pions noirs.

De plus les joueurs sont informés, par le biais d'un message présent sur la page de jeu, de l'état de la partie (rejouer ou passer son tour).

1.3.2 Gestion des parties

Au démarrage, le joueur doit pouvoir choisir entre une partie à deux joueurs ou une partie contre l'ordinateur. De plus, on souhaitera sûrement plus tard que l'utilisateur puisse choisir la difficulté du jeu s'il choisit de jouer contre l'ordinateur.

Le jeu devra disposer d'une barre de menu offrant les services de base tels que nouvelle partie, nombre de pions restants, joueur qui doit jouer, aide, enregistrement, chargement, quitter, etc.

1.3.3 Déroulement d'une partie

Le jeu se déroulera en utilisant la souris. Le joueur posera virtuellement un pion en cliquant sur une case du plateau. C'est tout ce qu'a à faire le joueur! Le reste est géré automatiquement par l'ordinateur, notamment le retournement des pions, la mise à jour des points, le changement de joueur et la fin de partie.

Du point de vue de l'ordinateur, il devra être capable de placer ses pions dans des cases valides, et la pertinence de ceux-ci dépendra de son niveau.

1.3.4 Le programme

L'ensemble de notre projet devra pouvoir s'installer puis se lancer à l'aide d'un simple exécutable, il pourra aussi être mise en ligne sur internet par la suite.

Notre exécutable pourra bien sûr se sophistication plus tard, notamment au niveau graphique d'une part mais également au niveau des fonctionnalités mêmes du jeu, qui pour le moment ont été réduites à un niveau de conceptualisation et d'abstraction basique.

2.1 Notre code

Le jeu d'Othello se compose essentiellement d'un plateau de jeu et de pions. Pour représenter cela, nous avons naturellement opté pour un tableau à deux dimensions, composé de huit lignes et huit colonnes. Au départ, ce tableau est initialisé à 0 par défaut ; ensuite, lorsqu'un joueur pose un pion sur le plateau, la case correspondante du tableau passe à 1 ou à 2 selon la couleur du joueur.

La pose d'un pion se fait à l'aide de la souris : le curseur de la souris représente un pion de la couleur du joueur actuel. Lorsque l'on clique à un endroit du plateau, les coordonnées de la souris sont converties afin de correspondre aux cases du tableau. Si la case était jouable, le pion est déposé.

A chaque tour, il n'y a qu'un certain nombre de cases qui sont jouables. Pour assurer la validité d'un coup il a fallu mettre au point une fonction particulière. Tout d'abord, une case est jouable si elle n'est pas déjà occupée par pion.

Ensuite il faut que le placement de ce pion entraîne au moins un retournement de pions adverses. Pour cela nous vérifions dans toutes les directions si un pion adverse occupe une case juste à côté. Si cela est vérifié, il faut aussi qu'il y ait un pion ami dans le prolongement de la direction ; en faisant attention à ce qu'il n'y ait pas de cases vides entre les deux.

Une fois cette fonction établie, la gestion du retournement des pions est aisée. La fonction qui s'occupe de retourner les pions suit le même cheminement, le retournement étant simplifié par l'implémentation du plateau : pour changer la couleur d'un pion, il suffit de changer le chiffre dans notre tableau.

Lorsqu'un pion est posé, on vérifie si le joueur adverse peut jouer sur au moins une case avant de lui donner la main. Si ce n'est pas le cas, il doit passer son tour. Lorsqu'il n'y a plus de case libre, parfois même avant, aucun des deux joueurs ne peut avoir la main : c'est ce qui définit la fin de la partie.

On fait alors une dernière fois le décompte des pions pour chacun des deux joueurs, celui qui a le plus grand nombre de pion est déclaré vainqueur.

A tout moment, le joueur a la possibilité d'enregistrer la partie en cours ou d'en charger une précédemment sauvegardée. On utilise pour cela un système de fichiers. Afin de sauvegarder une partie, il est nécessaire de récupérer l'état du plateau au moment de la sauvegarde et le joueur qui doit effectuer le prochain coup. On enregistre ainsi dans des fichiers le tableau courant et la couleur du joueur. Pour restaurer une partie, il suffit de lire ces fichiers et de spécifier le tableau et la couleur lus comme tableau courant et couleur du joueur courant.

Voici donc notre code détaillé, étapes par étapes, où nous justifions les fonctions et variables employées (dans ce code, tout est dans un même fichier. Bien sûr, pour alléger l'écriture et surtout avoir la possibilité de gérer chaque partition séparément, nous préférons diviser ce fichier en plusieurs fichiers externes, ce qui facilitera notre compilation.):



```

/*****
/*      Bibliothèques de fonctions nécessaires.      */
*****/

#include <stdio.h>

#include <MlvTypeC.h>
#include <MlvProcC.h>
#include <MlvX.h>

/*****
/*      Définition des Constantes.      */
*****/

/* Définition du rayon d'un pion. */
/* De ce rayon vont découler les proportions de tous les autres éléments! */

#define RAYON 20

/* Définition de la taille d'une case. */
/* La taille de la case comprend la taille du pion plus 3/10ième de son */
/* diamètre de chaque côté. */

#define CASE (2*RAYON + 3*(2*RAYON/10))

/* Définition du décalage du plateau de jeu. */
/* Pour l'instant, en l'absence d'autres éléments l'échiquier est décalé de la */
/* taille d'une case. */

#define BORDER CASE

/* Définition du nombre de lignes et de colonnes. */

#define LIGNES 8
```



```

/*****
/* Définition des FONCTIONS GRAPHIQUES globales. */
*****/

/* DRAWBOARD */
/* Affiche le plateau en intégralité selon ce qui est contenu dans le tableau */
/* plateau. */

void drawBoard(Colones * plateau, byte tour, MlvStruct *x_var)
{
    int i, j, x, y;

    for( i=0 ; i < COLONNES ; i++ )
    {
        for( j=0 ; j < LIGNES ; j++ )
        {
            x = BORDER+j*CASE+j;
            y = BORDER+i*CASE+i;

            draw_rectangle(x,y,CASE,CASE,"black",x_var);

            switch(plateau[i][j][tour])
            {
                case BLANC :
                    draw_filled_circle(x+CASE/2, y+CASE/2, RAYON,
"white", x_var);
                    break;
                case NOIR :
                    draw_filled_circle(x+CASE/2, y+CASE/2, RAYON,
"black", x_var);
                    break;
                default :
                    break;
            }
        }
    }
}

```

```

/* DRAWTOKEN */
/* Dessine le pion de la couleur du joueur au point (X,Y) coordonnées du centre */
/* du pion. */

void drawToken(int x, int y, byte joueur, MlvStruct *x_var)
{
    if( joueur == BLANC)
    {
        draw_filled_circle(x+3, y+3, RAYON, "black", x_var);
        draw_filled_circle(x, y, RAYON, "white", x_var);
    }
}

```

```

else
{
    if( joueur == NOIR)
    {
        draw_filled_circle(x+2, y+2, RAYON, "white", x_var);
        draw_filled_circle(x, y, RAYON, "black", x_var);
    }
}
}

```

```

/* DRAWCELL */
/* Redessine uniquement la case [X,Y] avec le pion joueur à l'interieur. */

```

```

void drawCell(int x, int y, byte joueur, MlvStruct *x_var)
{
    x = BORDER+x*CASE+x;
    y = BORDER+y*CASE+y;
    drawToken(x+CASE/2, y+CASE/2, joueur, x_var);
    display_area(x,y,CASE,CASE,x_var);
}

```

```

/*****
/* Définition des FONCTIONS globales. */
*****/

```

```

/* NEWROUND */
/* Fonction qui crée et initialise le plateau de jeu. */
/* Attention! nombre pairs obligatoires pour iLignes et iColonnes. */
/* FAIRE LES TESTS MEMOIRE! */

```

```

Colonnes * newRound()
{
    int i, j;

    Colonnes * plateau = calloc( COLONNES , sizeof( plateau ) );

    for( i=0 ; i < COLONNES ; i++ )
    {

        plateau[i] = calloc( LIGNES , sizeof( plateau[i] ) );

        for( j=0 ; j < LIGNES ; j++ )
        {

            plateau[i][j] = calloc( 1, sizeof( plateau[i][j] ) ); /* Premier tour de la
partie. */

            if( ( ( i == ( ( COLONNES / 2 ) - 1 ) ) && ( j == ( ( LIGNES / 2 ) - 1 ) ) ) ||
( ( i == ( COLONNES / 2 ) ) && ( j == ( LIGNES / 2 ) ) ) ) )

```

```

        {
            plateau[i][j][0] = BLANC;
        }
        else
        {
            if( (( i == ( ( COLONNES / 2 ) - 1 ) ) && ( j == ( LIGNES / 2 ) ) )
|| ( ( i == ( COLONNES / 2 ) ) && ( j == ( ( LIGNES / 2 ) - 1 ) ) ) ) )
            {
                plateau[i][j][0] = NOIR;
            }
            else
            {
                plateau[i][j][0] = VIDE;
            }
        }
    }
}

return plateau;
}

```

```

/* OTHERPLAYER */
/* Fonction qui passe la main au joueur adverse une fois le coup joué. */

```

```

byte otherPlayer(byte joueur)
{
    if( joueur == BLANC)
    {
        return NOIR;
    }
    else
    {
        if( joueur == NOIR)
        {
            return BLANC;
        }
        else
        {
            return VIDE;
        }
    }
}

```

```

/* APPLYRULES */
/* Fonction qui actualise la manche TOUR une fois qu'un pion est placé à la case [X;Y] */
/* par le joueur. */
/* OPTIMISER LE CODE AVEC DES BYTE COMME DANS LES BOUCLES FOR PAR EXEMPLE!!! */

```

```

/*****
/* DEBUG */
/* printf("Tours : %d ; joueur : %d ; test : %d ; Xcase : %d ; Ycase : %d ; pion : %d ;\n",
tour,joueur,otherPlayer( plateau[yCase+i][xCase+j][tour] )
,xCase+j,yCase+i,plateau[yCase+i][xCase+j][tour]);*/
/* printf("Tours : %d ; joueur : %d ; Xcase : %d ; Ycase : %d ; XEnd : %d ; j : %d ; YEnd :
%d ; i : %d ; TCOunt : %d ; pion : %d ;\n",
tour,joueur,xCase,yCase,XEnd,j,YEnd,i,Tcount,plateau[yCase+i][xCase+j][tour]);*/
*****/

```

```

void applyRules(Colonne *plateau, int xCase, int yCase, byte tour, byte joueur, MlvStruct
*x_var)

```

```

{
    int i, j;
    int XEnd, YEnd;

    for( i=-1 ; i<2 ; i++)
    {
        for( j=-1 ; j<2 ; j++)
        {
            if( yCase+i >= 0 && yCase+i < COLONNES && xCase+j >= 0 &&
xCase+j < LIGNES)
            {
                printf(" %d %d \n", i,j);
                if( otherPlayer( plateau[yCase+i][xCase+j][tour] ) == joueur ) /*
si VRAI cela veut dire que le pion adjacent est de couleur opposée! */
                {
                    byte Tcount = 1;
                    XEnd = xCase + ( j * (Tcount + 1) );
                    YEnd = yCase + ( i * (Tcount + 1) );

                    printf("XEnd : %d ; Yend : %d tab : %d joueur :
%d\n",XEnd, YEnd, plateau[YEnd][XEnd][tour], joueur);

                    while( ( otherPlayer( plateau[YEnd][XEnd][tour] ) ==
joueur ) )
                    {
                        Tcount++;
                        XEnd = xCase + ( j * (Tcount + 1) );
                        YEnd = yCase + ( i * (Tcount + 1) );
                        printf("XEnd : %d ; Yend : %d tab : %d joueur :
%d\n",XEnd, YEnd, plateau[YEnd][XEnd][tour], joueur);
                    }

                    if( plateau[YEnd][XEnd][tour] == joueur ) /* Pion de
même couleur qui boucle la rangée. */
                    {
                        printf("Boucle 2 bonne\n");
                        while( XEnd != xCase || YEnd != yCase)
                        {
                            XEnd -= j;
                            YEnd -= i;

```



```

{
    *x -= BORDER+LIGNES;
    *y -= BORDER+COLONNES;

    if( (*x > 0) && (*y > 0) && (*x < LIGNES*CASE+LIGNES) && (*y <
COLONNES*CASE+COLONNES) )
    {
        *x /= CASE;
        *y /= CASE;
    }
    else
    {
        *x = -1;
        *y = -1;
    }
}

```

```

/*****/
/* PROGRAMME PRINCIPAL. */
/*****/

```

```

int main(void)
{

```

```

/*****/
/* Définition des VARIABLES LOCALES. */
/*****/

```

```

/* VARIABLES relatives à la fenêtre D'AFFICHAGE. */

```

```

char    *display_name    =    NULL;
int     width            =    WIDTH;
int     height           =    HEIGHT;
int     option;
MlvType x_var;

```

```

/* VARIABLES relatives au FONCTIONNEMENT du jeu. */

```

```

int     x,y;              /* Coordonnées d'un clic.*/
byte    tour              =    0;          /* Numéro du tour. */
byte    joueur           =    BLANC;      /* Pion qui commence. */

```

```

    Colonne
    • plateau = newRound();          /* Création du plateau. */

```

```

/*****/
/* CREATION de la FENETRE. */
/*****/

create_window (display_name,width,height,&x_var);

name_window ("Othello",&x_var);

/*****/
/* GRAPHISME global de la FENETRE. */
/*****/

draw_filled_rectangle(10 , 10, WIDTH-20, HEIGHT-20, "DarkSeaGreen",
&x_var);

drawBoard(plateau, tour, &x_var);

/*****/
/* AFFICHAGE du GRAPHISME global de la FENETRE. */
/*****/

display_window(&x_var);

/*****/
/* LANCEMENT du JEU. */
/*****/

while(1)
{

/* On attend un clic de l'utilisateur, le programme est alors stoppé ici. */
mouse_wait (&x, &y, &x_var);

/* Une fois le clic enregistré, on incrémente le tour, la partie avance. */
tour++;

/* On traduit les coordonnées du clic en terme de numéro de case. */
seekCase(&x,&y);

/* On réactualise le plateau de jeu. */

refreshBoard( plateau, x, y, tour, &joueur, &x_var);

}

```

```
    return 1;  
}
```



Une fois le programme principal écrit, des améliorations peuvent être envisagées :

- afficher le nombre de tours joués.
- afficher le nombre de pions dont dispose chaque joueur.
- un indicateur élégant affiche si c'est aux blancs ou aux noirs de jouer.
- un clic dans la zone "aide" affiche un cercle sur les cases possibles.
- un clic dans la zone "aide" affiche sur les cases possibles, le nombre de pions dont disposera le joueur s'il s'y positionne.
- un clic dans la zone "nouvelle partie" crée une nouvelle partie.
- un clic dans la zone "annulation" permet de revenir une fois en arrière.
- on peut revenir en arrière jusqu'au début de la partie.
- on crée un mode "contre l'ordinateur" où l'intelligence artificielle se positionne simplement sur la case qui lui rapporte immédiatement le plus de pions.
- l'intelligence artificielle se positionne sur la case qui engendre le plus de pions à un coup en avance, puis à n coups en avance.

Nous en faisons le détail dans notre soutenance orale.

2.2 Notre graphisme

Basique, sans faire non plus dans le minimalisme. Une interface agréable, une interactivité conviviale simple. Le jeu correspond à toute catégorie de joueurs et personne n'est exclu! On pourra plus tard, en utilisant la 3D, obtenir un rendu plus soigné, plus futuriste.

3.1 Jouer contre l'ordinateur

Lorsque l'on choisit de jouer contre l'ordinateur, les principes de base ne sont évidemment pas changés. La seule chose est que l'on ne doit pas simplement s'occuper de la validité et de la cohérence du déroulement de la partie, nous devons également permettre à notre programme de répondre "intelligemment" à une situation donnée. En clair, lorsque le joueur pose un pion, l'ordinateur doit à son tour poser un pion, cela se fait en fonction des possibilités qui s'offrent. Avant toute chose, il fallait donc répertorier tous les coups possibles selon la situation actuelle. Nous avons opté pour un tableau global, partant du principe qu'en général il n'y a qu'entre cinq et dix coups possibles. La première case de notre tableau contiendra le nombre de coups jouables. Ensuite chaque case contiendra les coordonnées du coup sous la forme d'un simple réel : la ligne représentant la partie entière, la colonne étant la partie décimale.

3.2 Réflexion sur une direction possible

Maintenant que nous avons à disposition l'ensemble des coups jouables par l'ordinateur, il faut décider d'un moyen de choisir parmi ceux-là. Nous avons opté pour l'utilisation d'un algorithme de probabilités minimales et maximales alternées (simulation aléatoire à un rang fixé). Revenons d'abord un peu sur cet algorithme. En règle générale, on va d'autant mieux répondre à un problème donné qu'on a un horizon de vision important. Choisir directement une direction par rapport à ce qu'elle apporte à court terme est une très mauvaise stratégie. Cela est encore plus vrai dans notre cas du jeu Othello.

Si je cherche à maximiser tel ou tel paramètre, en l'occurrence mon nombre de pions, je peut être trompé par les résultats à court terme. En effet, il se peut qu'au tour d'après, mon adversaire ait l'opportunité, à cause de mon coup précédent, de minimiser ce paramètre de manière beaucoup plus importante que je ne l'ai maximisé.

Ainsi, plus je m'intéresserai en profondeur au problème, plus je pourrai effectuer de choix pertinents.

Pour implémenter cela, il faut tester tous les coups possibles et cela de manière récursive, dépendant de la profondeur jusqu'où l'on veut aller. Une fois au bout des appels récursifs, on regarde la situation dans laquelle on se trouve et l'on récupère la balance des pions. En remontant, il faut tenir compte du fait que l'ordinateur cherche à avoir au final, le plus de pions de sa couleur, et son adversaire va vouloir qu'il en ait le moins possible. A la fin, nous avons gardé en mémoire le coup qui nous a le plus satisfait et nous plaçons le pion à cet endroit.

Pour pouvoir faire tout cela, ce qu'il y a de plus important est le fait de pouvoir faire et défaire les changements. Dès que l'ordinateur part dans une branche donnée, il faut ensuite pouvoir revenir dans l'état précédant tous les changements effectués, afin de repartir dans une autre branche. Nous pouvons donc mettre en mémoire le contexte de la partie dans des tableaux, avec un

nombre de tableaux égal à la profondeur de l'algorithme. Ils pourront tous être répertoriés à l'aide d'un vecteur.

Admettons que l'on ait déjà récupéré des valeurs pour une partie de l'ensemble de recherche à une profondeur p ; lors des futures recherches à une profondeur $p+1$, la récupération de certaines valeurs pourra assurer qu'il ne sert plus à rien de chercher dans cette direction.

Prenons l'exemple suivant : on est à une profondeur p où l'on recherche une valeur minimale, et que pour l'instant la plus petite valeur vaut α .

Si on rencontre à une profondeur $p+1$, où l'on cherche donc des valeurs maximales, une valeur supérieure ou égale à α , on peut arrêter les recherches dans ces branches.

En effet, quelles que soient les valeurs trouvées, aucune ne pourra être utile. Si on trouve des valeurs plus petites, comme à $p+1$ on veut maximiser, on n'en tiendra pas compte. Si on trouve des valeurs plus grandes que α , on sait qu'à p , on cherche à minimiser, donc on n'utilisera pas non plus ces résultats. Le principe pour le reste est le même sauf que la préoccupation des différents niveaux est inversée.

La mise en place de ces coupures permet de gagner énormément de temps à l'exécution car évitant des recherches inutiles.

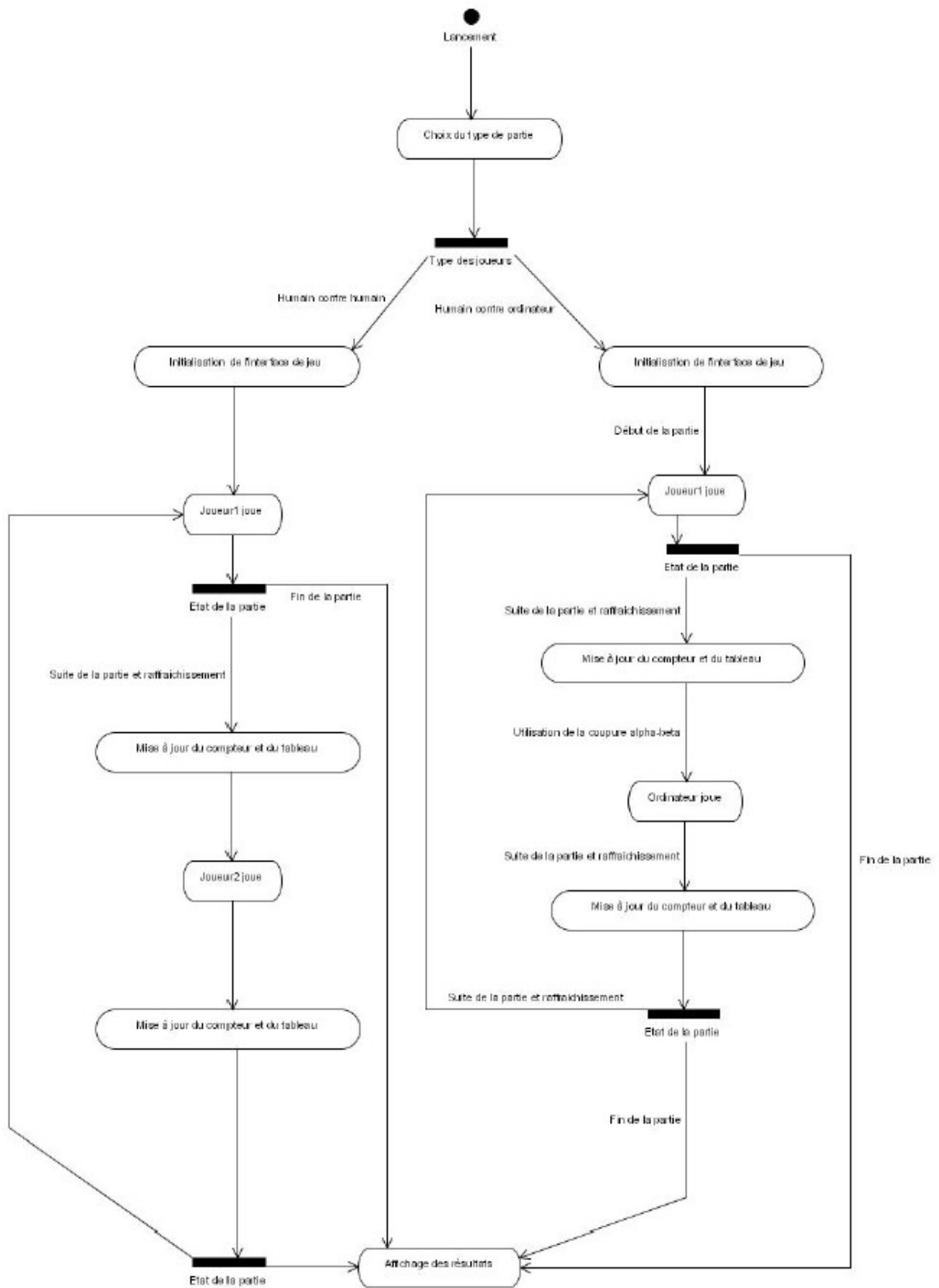
Dans tous les cas, lorsque l'on arrive au bout de la récursivité de nos fonctions, il faut évaluer le contexte dans lequel se trouve maintenant notre partie. La mise au point de la fonction d'évaluation de notre situation de jeu est un élément primordial pour le fonctionnement du programme. C'est réellement cette fonction qui va simuler l'intelligence artificielle car elle permet de porter un jugement sur une situation donnée.

Nous pouvons opter pour deux paramètres distincts : la disposition des pions et la mobilité. L'évaluation de la disposition des pions se fait par le biais d'une grille d'évaluation. Lorsque l'ordinateur possède un pion sur une case, on augmente son score d'un nombre de points dépendant de sa position. Si c'est le joueur humain qui possède un pion on diminue son score.

Pour ce qui est de la mobilité, l'évaluation fonctionne sur le principe suivant : moins le joueur humain aura de coups possibles, moins il sera dangereux. Nous diminuons donc le score de l'ordinateur du nombre de coups jouables pour l'humain dans la position finale. On multiplie ce nombre par un coefficient permettant de gérer l'importance de ce paramètre. Cette deuxième étape possède un autre avantage : cherchant à minimiser le nombre de coups jouables par l'humain, on diminue la taille de l'arbre de recherche des fonctions. En étudiant les conséquences de l'appel à cette fonction, on optimisera les possibilités du jeu.

Pour résumer, voici un diagramme expliquant le déroulement du jeu choisi:

2 chemins possibles, joueur contre joueur ou joueur contre ordinateur,



CONCLUSION

Nous obtenons au terme de notre projet, un jeu d'Othello, où tous les mécanismes intrinsèques sont bien mis en place et où l'intelligence artificielle permettra, une fois que l'on aura défini les fonctions appropriées, de jouer contre un ordinateur dont le niveau peut être défini par l'utilisateur.

Nous avons fait en sorte que le déroulement de la partie ne soit pas entaché par une attente trop longue et répétitive, l'ergonomie et l'interactivité étant restés nos principaux axes de réflexion.

La principale amélioration que nous pourrions apporter ne se situerait pas au niveau de l'algorithme de parcours des différents coups possibles, mais plutôt dans notre éventuelle fonction d'évaluation. L'idéal serait de prendre en compte si un pion est prenable ou non. En cherchant à maximiser le nombre de pions "définitifs", nous pensons que la qualité de jeu serait améliorée, et de loin!

Au final, la réalisation de notre jeu d'Othello s'est faite en concordant à la fois le formalisme et les théories liés aux systèmes intelligents, et les principes de stratégie inhérents au jeu d'Othello lui-même.

Il ne vous reste plus qu'à préparer vos stratégies, et défier vos adversaires... même nous!



Sources principales: www.developpez.com
www.wikipedia.com