BOLLET Jacqueline
GRENIER Thierry

1ère Année
Février 2000

Projet d'Algorithmique
OTHELLO

Projet encadré par Léon Mugwaneza

INTRODUCTION

Le but de ce projet était de faire un jeu de stratégie d'information parfaite et de somme nulle. Notre choix s'est porté sur le jeu d'Othello. Ce jeu était inconnu de l'un de nous et mal maîtrisé de l'autre. Nous étions intrigué d'en savoir un peu plus surtout que nous disposions d'une version de ce jeu sur Game Boy sur laquelle nous avons appris les premières bases. Nous avons peu après trouver un site proposant le jeu qui a consolidé notre choix.

SOMMAIRE

I – Présentation du programme

1) Le principe du jeu, fonctionnement général

But du jeu :

Avoir plus de pions de sa couleur a la fin du jeu que son adversaire.

Fin du jeu :

Le jeu est fini quand aucun des deux joueurs ne peut jouer.

Présentation du jeu :

Le jeu se présente sous la forme d'un damier (8*8) sur lequel on peut poser des pions qui ont une face blanche et une face noire.

Le point de départ est constitué par le remplissage du carré central (2*2) par des pions de couleurs alternatives : en haut un noir puis un blanc, et en bas, un blanc puis un noir.

Déroulement de la partie :

Le joueur, dont c'est le tour, doit poser un pion de sa couleur sur l'othelier. Ce faisant, il doit encadrer au moins un pion de la couleur adverse avec le pion pose et un autre déjà en jeu de sa couleur. Si cela lui est impossible (par exemple quand les joueurs ont entièrement rempli l'othellier), il doit passer son tour.

Tous les pions adversaires entourés sont alors retournés et prennent la couleur du joueur. L'encadrement se fait toujours sur une ligne droite et ce quelle que soit sa direction. Si plusieurs pions adverses séparent le pion posé et un autre pion de la couleur en jeu, ils sont tous retournés. Une position favorable à un coup peut donc s'avérer fatale au coup suivant !

2) Mode d'emploi et fonctionnalités (options fournies)

Le jeu est fourni sous la forme d'un fichier compressé (Bollet_Grenier.tar.gz) qui contient les sources de notre programme.
Il faut d'abord le décompresser à l'aide de la commande « tar –xvvzf Bollet_Grenier.tar.gz » ce qui aura pour effet de créer un répertoire Bollet_Grenier.
Il faut ensuite aller dans le répertoire créé (cd Bollet_Grenier).
A présent seules les sources sont présentes. Pour les compiler on tape « make ».

Attention : à ce stade le programme Othello ne peut être exécuté : il faut d'abord lancer le programme Ecriture. Il écrira les fichiers heur2 et heur34 qui sont nécessaires au lancement des niveaux de jeu 2, 3 et 4.

Il ne reste plus alors qu'à lancer Othello.

Remarque : Pour conserver de l'espace disque, il peut être utile de lancer « make clean » qui effacera les fichiers objets (les exécutables, les sources et les fichiers d'heuristiques seront conservés).

Pour ne conserver que les sources, la commande « make propre » permet d'effacer tout le reste.

Fonctionnalités :

Les types de jeu sont :
- Joueur humain contre joueur humain
- Joueur humain contre ordinateur (quatre niveaux proposés)
- Ordinateur contre ordinateur (pour un apprentissage des méthodes de jeu et comparer les différents niveaux).

Pour y accéder, un choix vous est proposé en début de partie : les joueurs sont désignés par
0 pour un joueur humain
1 pour l'ordinateur de niveau 1 (niveau le plus faible)
2 pour l'ordinateur de niveau 2 (niveau intermédiaire)
3 pour l'ordinateur de niveau 3 (niveau difficile)
4 pour l'ordinateur de niveau 4 (niveau très difficile). (voir l'Annexe sur les niveaux de jeu)
Le premier joueur aura les X, le second les O.

Option conseil :

Celle-ci vous est proposée juste après avoir choisit les joueurs (si, bien entendu, un des joueurs au moins est un humain). Il suffit répondre 1 pour l'activer ou 0 pour ne pas l'activer.
A tout moment de la partie le joueur pourra demander à l'ordinateur de niveau 4 quel pion il poserait. Il suffit de taper 0 lors de la demande de l'abscisse du pion à poser. Ce conseil n'engage à rien : le joueur peut jouer en n'importe quel endroit quel que soit le résultat du conseil.
Remarque : dans un souci d'équité entre joueurs, il n'est pas possible d'activer l'option conseil pour seulement un seul joueur en cas de deux joueurs humains : les deux joueurs ont accès à la fonction conseil si elle est activée.

Option déjouer :

Elle vous permet de défaire le dernier coup que vous ayez joué. Ceci vous permet de revenir en arrière au cas où vous vous seriez trompé en entrant les coordonnées du pion à jouer.
De même que l'option conseil, elle est accessible à tout moment de la partie en tapant 9 lors de la demande de l'abscisse du pion à poser.

II – L'avancée du projet

La progression dans notre projet s'est faite pas à pas et de manière constante ; aucune de nos étapes n'a été celle qui a fait de ce projet ce qu'il est et aucune n'a été inutile.

Les étapes seront expliquées plus en détail dans la prochaine partie. Il s'agit ici plus d'un calendrier de la fin de chacune des étapes.

Le 4 décembre 1999 : Recherches sur Internet (fonctions d'évaluation, heuristiques, etc.).

Le 13 décembre 1999 : Arbitre du jeu et affichage de l'othellier.

Le 16 décembre 1999 : Définition de l'interface joueur.

Le 17 décembre 1999 : Jeu joueur-joueur (arbitre2).

Le 5 janvier 2000 : Ordinateur à un coup de profondeur avec heuristique partout égale à 1 (arbitre4).
Définition de l'interface ordinateur.
Possibilité de choisir entre les jeux joueur-joueur, joueur-ordinateur ou ordinateur-ordinateur. (arbitre4).

Le 16 janvier 2000 : Première approche des tableaux d'heuristique (arbitre6).

Le 20 janvier 2000 : Rapport d'avancement et cela nous a permis de mettre à plat nos idées, faire des prévisions.

Le 2 février 2000 : Chargement de tableaux d'heuristique différents suivant le niveau choisi pour l'ordinateur => Niveau 2 (arbitre9).

Le 3 février 2000 : Compilation séparée

Le 5 février 2000 : Ordinateur avec plusieurs coups de profondeur : algorithme Min-Max

Le 6 février 2000 : Ordinateur avec plusieurs coups de profondeur : algorithme Alpha-Béta

Le 6 février 2000 : Fonction d'évaluation du gain (selon un tableau d'heuristique statique)

Le 7 février 2000 : Fonction d'évaluation du gain (selon le tableau et le nombre de coups joués) => Niveau 3

Le 8 février 2000 : Déjouer les coups.

Le 9 février 2000 : Demander à l'ordinateur quel coup il jouerait à notre place

Le 10 février 2000 : Tableau d'heuristique dynamique => Niveau 4

Le 20 février 2000 : Tests et améliorations.

Nota : les dates entre le 2 février et le 10 février se suivent car elles correspondent au moment de l'implémentation. L'algorithme était déjà en place, il suffisait surtout de vérifier qu'il faisait bien ce qu'on lui demandait de faire.

Cependant, nous n'avons pas ajouter deux options auxquelles nous avions pensé :

1) Gestion d'un temps limite
2) Graphisme

Ceci pour deux raisons : le temps limite n'avait pas lieu d'être puisque le temps de réflexion de l'ordinateur est assez faible pour ne pas le limiter, et cette option pour le jeu à deux joueurs humains n'est pas très utile (les humains peuvent toujours s'imposer cette limite eux même s'ils le désirent : utilisation d'une montre ou d'un chronomètre).

De plus, l'affichage tel que nous l'avons conçu est assez visuel. Il impose peu de charge au processeur et il n'est donc pas nécessaire de l'alourdir. Enfin, nous n'avons pas non plus vraiment eu le temps pour en réaliser une avec des outils que nous ne connaissions pas comme xwindows.

III - Les étapes

1) Pré rapport

Depuis le début du projet jusqu'à la rédaction du pré rapport, tout notre travail s'est concentré sur la mise en place d'une idée (très imprécise mais globale) de l'algorithme que nous allions mettre en place.

Nous avons réfléchi aux options et fonctionnalités que nous pourrions implémenter. Pour ce faire, nous nous sommes beaucoup inspirés de recherches de jeux déjà programmés (sur Internet et sur console de jeux) pour voir ce qui pouvait être proposé et ce que nous allions effectivement retenir.

De plus, ces recherches nous ont fourni différentes méthodes pour l'algorithme de jeu ; ainsi nous nous sommes aperçu que la profondeur de la recherche dans l'arbre Min-Max n'était pas forcément la seule différence que nous pouvions mettre entre nos niveaux. Nous avons récupérer plusieurs tableaux d'heuristique, dont certains étaient modifiés au cours du jeu.

Enfin, nous avons trouvé un document expliquant différentes tactiques de jeu qui appellent des stratégies de positionnement, mais aussi de parité de coups, de mobilité et de maximisation. Cependant, certaines de ces tactiques impliquent de connaître la position exacte de chaque pièce pour la fonction d'évaluation, nous ne les avons donc pas implémentées.

Remarque : L'ordinateur de niveau 3 et 4 réalise quand même plusieurs de ces stratégies qui découlent en fait des heuristiques.

2) Première mise en place de la décomposition générale de l'algorithme

Nous avons alors tenté de décrire comment l'algorithme général allait fonctionner.

Nous avons pensé à la mise en place d'au moins trois modules : l'arbitre, l'humain et l'ordinateur. L'humain devait contenir l'interface graphique. L'arbitre devait contenir le module de vérification.

Avec la première idée sur le jeu que nous avions défini pour le pré rapport, nous avons posé les premières idées de la « structure » générale du jeu. Nous avons pensé à différentes fonctions qui devraient être faites ainsi qu'à différentes variables et tableaux.

3) Mise en place de l'Arbitre

Pour concrétiser nos pensées et partir d'une base plus ancrée, nous avons écrit l'algorithme de l'arbitre.

Ceci nous a permis de mieux fixer nos idées et de faire des choix définitifs et d'autres que l'on savait temporaires.

Les problèmes ont été alors multiples :

- Nous avions remarqué que les algorithmes vus sur Internet recherchaient si la pose d'un pion était possible ou non via un tableau de directions. Nous avons cependant pensé qu'il serait plus « intuitif » d'utiliser une double boucle sur la direction en ordonnées puis en abscisse. Cette solution plus claire se paie au prix d'un test supplémentaire (direction 0,0) mais nous avons préféré la garder.

- La fonction qui vérifie si un pion peut être posé à une case est, du moins sur le principe global, la même que celle qui joue un pion. Nous avons alors décider d'implémenter un fonction que nous savions provisoire « VerifieModifieSiPasF » qui faisait les deux. Cette solution nous a permis, entre autres, de reporter le problème des différences entre les fonctions verifie_coup, joue et joue_ordi.

Nous avons donc écrit sur papier les fonctions init, verifiemofidiesipasF, question_coup, traiter_coup, gestion_score, main, mais aussi Demander_coup. Le but était de pouvoir faire jouer un humain contre un autre humain. Il nous a donc fallu aussi nous occuper de l'interface. Comme nous savions que nous n'allions certainement pas la développer pour en faire une « vraie » interface graphique, nous en avons implementé une qui répondait surtout aux besoins de savoir où en est le jeu et à combien s'élèvent les scores.

Il s'est avéré que nous avons gardé cette interface « de fortune » qui, somme toute, est assez efficace et agréable à la pratique.

Nous avons alors traduit tout cela directement sur les machines et sommes parvenus à faire une première version qui, si elle ne répondait pas aux attentes finales, avait le mérite de nous confirmer la bonne avancée de nos travaux : nous n'étions pas partis sur de mauvaises bases.

4) Création de « temp.c »

Contrairement à son nom, le fichier « temp.c » n'est pas, ni se veut, un programme. Il pose en fait de manière écrite et facilement accessible à nous deux en même temps toutes les bases que nous avions prévues pour la suite.

C'est à ce moment que nous avons pensé à nos structures définitives : les constantes Maxret, Maxcoup, Depth, les structures pion, coup et les variables Historique, histcur, HeurA et HeurB sont apparues.

Les constantes Maxret et Maxcoup ont pu facilement être calculées. En revanche pour Depth, nous espérions faire des niveaux qui joueraient plus sur autre chose que la profondeur pour se différencier. Nous espérions alors une profondeur de sept coups.

Remarque : Il s'avérera que mettre plus de profondeur aux niveaux meilleurs n'est pas inutile et que le niveau 1 avec trop de profondeur est trop difficile à battre. On choisira donc bien plus tard une variable Depth initialisée à 2 et une profondeur de (Depth + Niveau).

Nous avons pensé beaucoup plus en détails aux différences entre les niveaux, ainsi qu'à l'algorithme d'Alpha_Beta. Nous avons aussi vu la nécessité de la fonction déjouer et de détermination (que nous avions alors appelée mere_alpha_beta).

Remarque : le tableau Historique et la variable histcur sont apparues pour l'option déjouer mais aussi pour mettre en place des bases pour enregistrer une partie. Cependant, nous n'avons pas réfléchit au problème de l'enregistrement de partie c'est pourquoi nous ne l'avons pas implémenté.

Nous avons ajouté ce fichier à ce rapport car il permet de bien situer notre avancée à ce moment-là.

« Temp.c » nous a conduit à écrire plusieurs versions successives de « arbitre.c ». Bien que nous appelions ces programmes « arbitreX.c », ces derniers étaient plus que le simple arbitre mais le programme restait assez court et explicite pour le développement. Nous n'avons pas estimer nécessaire de le diviser en module tout de suite.

5) Mise en place d'un premier joueur ordinateur

C'est «arbitre6.c » qui aura eu tout ce qu'on voulait implémenter avant de s'occuper réellement des niveaux tels que nous les avions définis.

Il a un joueur ordinateur qui joue le pion qui lui permettra de retourner le maximum de pion à ce coup (c'est-à-dire une version simplifiée du niveau 1 prévu). Nous savions qu'il s'agissait en fait du même joueur que les niveaux 1 et 2 finaux à ceci près que son tableau d'heuristique (la fonction init_heur1 a été créée à ce moment) était tout à 1, et la profondeur de sa recherche était de 1. Cette simplification nos a permis d'avancer prudemment avant de nous lancer dans la mise en place des vrai niveaux. Le but était surtout de résoudre les éventuels problèmes du à la mise en place d'un ordinateur comme les problèmes d'affichage (quand afficher son coup ?).

Cette version permettait déjà de choisir les jeux Humain contre Humain, Ordinateur contre Humain et Ordinateur contre Ordinateur (nous savions qu'il allait falloir tester les niveaux des ordinateurs).

6) Rédaction du Rapport d'avancement

Le rapport a été pour nous l'occasion idéale de mettre la suite au propre : en effet, nous venions de mettre au point le premier joueur ordinateur et, les vacances approchant, il fallait déterminer ce que chacun allait faire pendant ce temps. Il nous a donc permis de mettre au clair un plan de ce qu'il nous restait à faire : le découpage final en module (même si nous n'avions pas prévu le module Ecrire) et les dernières fonctions à implémenter.

En fait, nous avons autant rédigé ce rapport pour nous que pour le client. Notre rédaction s'en est faite ressentir car nous avons surtout voulu mettre le point sur ce que nous avions déjà réalisé à l'intérieur de ce que nous nous étions fixé de faire.

Quelques modifications ont alors été effectuées pour repartir sur les bases prévues : nous avons notamment ôté la fonction VerifieModifieSiPasF pour la remplacer par les fonctions Joue, Joue_Ordi et Verifie_Coup.

7) Vacances de fin janvier

Nous avions prévu de revenir de vacances en ayant fait :

- Les algorithmes Min_Max et Alpha_Beta.

Une esquisse avait été faite au moment de « temp.c » mais ils n'étaient pas encore programmés. De plus, cette version n'était pas correcte et nous n'avions pas pensé à certains problèmes comme celui de la réaction en cas de fin de partie.

- Les tableaux d'heuristiques avec leur chargement.

Nous venions enfin de décider de les charger depuis un fichier pour pouvoir les modifier à volonté lors des tests futurs. Il restait encore à faire les fonctions qui permettraient d'écrire et de lire dans ces fichiers.

Durant cette même période, nous avons aussi ajouter la fonction Nouvelle_partie et pensé plus précisément à la fonction d'évaluation. Nous avons alors pensé à une fonction arctangente pour donner l'importance relative entre les coups et l'heuristique.

8) Compilation séparée

A ce moment, nous avions plusieurs versions d'arbitre9.c (une chacun puisque nous étions séparés), qui devenait conséquente. Il nous a donc fallu mettre en œuvre les modules définis.

En effectuant ceci, nous nous sommes aperçu de la nécessité de rajouter un module Global.c pour bien séparer les variables et structures globales, ainsi que Ecrire.c qui permettrait de changer à loisirs nos tableaux d'heuristique sans tout recompiler. Nous avons vu aussi que pour éviter que des fonctions soient définies plusieurs fois (via les #include par exemple), il fallait ajouter une condition dans le fichiers.h (#ifndef).

Nous avons d'abord lancé le compilateur sans options, puis en ajoutant le flag –Wall ce qui nous a permis de fixer quelques petites erreurs.

Remarque : La version que nous avions alors fournie ne comportait pas encore les fonctions Min_Max ni Alpha_Beta car celles-ci n'étaient pas tout à fait prêtes : les cas où un joueur passe son tour étaient mal gérés. Elles l'ont été très peu de temps après.

9) Mise en place des dernières fonctions

Cette étape a été très rapide car nous avions déjà une bonne idée de ce que nous voulions faire. Nous avons alors ajouté les options conseil et déjouer. Nous avons aussi déterminé les fonctions d'évaluation et de changement d'heuristique. Nous avons pensé à une fonction exponentielle entre l'heuristique et le nombre de pions retournés.

Nous avons aussi ajouter les fonctions de modification de l'heuristique. Cette opération pouvait être très compliquée ou plus simple. Nous avons choisi de tenir en compte que le cas ou un joueur prend un coin pour faire nos modifications.

Nous sommes alors parvenu à faire une version qui répondait à tout ce que nous nous étions fixé. A ce stade, il nous restait deux possibilités : soit ajouter de nouvelles fonctionnalités (comme celle d'enregistrer une partie, problème auquel nous avons réfléchi mais qui nous semblait dur à réaliser car nous n'y avions pas pensé dans le détail auparavant), soit d'améliorer ce qui existait déjà. C'est à ce moment que nous vous avons demandé conseil sur la marche à suivre. Nous avons donc opté pour la seconde solution.

10) Tests

Comme attendu, la fonction d'Evaluation nous a posé de gros problèmes à optimiser. Nous nous y sommes repris à plusieurs fois avant d'en trouver une qui soit convenable. Nous avons alors effectué de nombreux tests. Nous sommes arrivé à la conclusion suivante : les 20 premiers coups, l'heuristique est plus importante que les pions gagnés. Les 10 derniers coups, le nombre de pions gagnés devient beaucoup plus important. Entre temps, l'importance des heuristiques décroît de manière exponentielle. La forme de l'exponentielle a été choisie pour « améliorer » les résultats obtenus. Elle ressemble beaucoup à la forme en arctangente à laquelle nous avions pensé initialement.

Remarque : le coefficient coeff a été calculé pour que la valeur du gain en heuristique soit aussi important que le nombre de pions retournés quand l'exponentielle est égale à 0,5. Nous l'avons donc divisé par 2,5 : l'heuristique la plus importante était de l'ordre de 50, alors que le nombre de pions retournés au maximum tourne autour de 20 (20 * 2,5 = 50).

Nous nous sommes aperçu que l'algorithme Alpha_Beta se comportait mal en cas de fin de partie. Nous avons donc rajouter la condition qui rend un gain très fort dans ce cas (+5000 si la partie est gagnée, -5000 si elle est perdue).

Nous avons ensuite changé à plusieurs reprises nos tableaux d'heuristique, les faisant jouer l'un contre l'autre. Nous avons nettement augmenter l'heuristique des coins pour les niveaux 3 et 4 car ceux-ci sont toujours menacés tard dans la partie.

Nous avons aussi tenté de changer la profondeur de recherche (en supprimant la constante Depth) mais celle pensée initialement s'est avérée plus convaincante. Nous avons remarqué que la constante Depth pouvait être mise à trois (d'où sept coups de profondeur pour le niveau 4), en gardant un temps de calcul raisonnable, mais la différence de jeu n'a pas été trouvée assez convainquante pour justifier cette perte de temps. D'une manière générale, nous avons toujours préféré la vitesse de jeu et de calcul à un niveau de jeu trop élevé. C'est aussi une des raisons pour laquelle la fonction de modification des heuristiques est si simple.

Enfin, en testant notre jeu contre un autre du commerce, nous avons pensé à changer le gain dans le cas ou un joueur devait passer. Après quelques essais, nous en sommes arrivé à la constatation que ceci n'a de l'importance que si on est peu profond dans l'arbre de recherche : en effet, le joueur voyant qu'il va passer, évitera que ceci lui arrive. En revanche, si le prochain coup conduit à cette position, cela peut être très avantageux.

III - Structures de données

Nous avons modéliser l'othellier par un tableau (10*10) qui présente un « garde-fou » pour que l'ordinateur puisse tester en dehors de l'othellier physique (8*8) sans rajouter de tests.

Le premier joueur (noir sur le vrai jeu d'Othello) est ici représenté par les X, avec une couleur 1 et une référence 0. De la même manière le second joueur (blanc) possède les O, la couleur –1, et la référence 1. Ceci nous permet de passer facilement d'un joueur à l'autre. A chacun de ces joueurs peut être associé (si besoin est) un tableau d'heuristique. Il s'agit, pour l'ordinateur, d'une indication des cases stratégiques à jouer. Chaque tableau est associé à un niveau. Pour passer d'un tableau à l'autre, on utilise des fonctions très simples (ChoixHeuris et Choix_heur).

Les pions sont représentés par leur abscisse x et leur ordonnée y. Un coup est caractérisé par le joueur C, le pion ppose qu'il vient de poser, le nombre de pion nbret qu'il retourne alors et la position de ceux-ci. Chaque coup est enregistré dans un tableau Historique qui permet de revenir aux coups précédents.

Nous avons pensé depuis le début faire une fonction d'évaluation qui ne tient en compte que les changements dus à la pose d'un pion. Une autre solution aurait consisté à considérer toujours l'othellier en entier. Notre solution permet d'avoir un fonction d'évaluation qui, au pire fait Maxret (20) opérations, tandis que la seconde en aurait toujours fait 64. Sachant que cette fonction est appelée autant de fois Alpha_Beta, cette solution nous semblait s'imposer d'elle même.

Voici le détail des structures et variables globales utilisées :

- Définition A = 1 (joueur noir : premier joueur)
  B = -1 (joueur blanc : second joueur)
- Définition d'une structure Pion : Entier x : abscisse du pion
  Entier y : ordonnée du pion
- Définition champ de donnée joueur : entier qui détermine qui joue entre le joueur blanc et le joueur noir
- Définition matrice 10*10 qui désigne un plateau.
- Définition pointeur sur plateau : Plateau
- Définition Maxret = 20 : nombre maximum de pions retournés en un coup
- Définition Maxcoup = 60 : nombre de coups maximum (il y a 64 cases et 4 pions posés initialement)
- Définition Depth = 2 : constante du backtracking.
- Définition d'un entier joueur

- Définition Structure Coup : Entier nbret : nombre de pions retournés
  Pion ppose : le pion posé à ce coup (utilise la structure Pion)
  joueur C : le joueur noir ou blanc (utilise la définition de joueur)
  Tableau de Pion pionret de taille Maxret : contient les pions retournés à ce coup

- Tableau M par forme Plateau :
  Les premières et dernières lignes et colonnes sont un « garde-fou » pour le joueur ordinateur et pour l'arbitre.

- Champ scoreA et scoreB qui retient le score des joueurs (A = noir ; B = blanc).

- Tableau ref[2] d'entiers : tableau de référence pour savoir qui est un humain et qui est l'ordinateur et son niveau de jeu sachant que 0 correspond à l'humain et un entier de 1 à 4 désigne le niveau d'un ordinateur.

- Tableaux HeurA et HeurB par forme Plateau : HeurA désigne un pointeur sur HA et de même pour HeurB.

- HA désigne le tableau d'heuristique du joueur A et de même pour HB.

- Tableau Historique par forme coup de taille Maxcoup : Historique des coups déjà joués.

- Champ Histcur : Entier qui désigne le nombre de coups déjà joués.

V - Modules et fonctions

1) Structure globale de notre programme

Nous avons découpé le programme Othello en 7 modules :

- Arbitre
- Humain
- Ordi
- Action
- Verification
- Affichage
- Global

Nous aussi un programme Ecriture composé du seul module Ecrire qui créé des fichiers d'heuristique.

Arbitre est un module où est défini l'initialisation du jeu (le « main » y est défini) . C'est lui qui fait appliquer les règles : il passe la main au joueur à qui de droit, contrôle la validité d'un coup proposé, détermine si la partie est finie. Il utilise tous les modules.

Humain est un module qui demande à l'utilisateur humain où il veut jouer. Si le coup demandé n'est pas dans le plateau, un autre coup est demandé. L'Arbitre vérifiera alors la validité du coup. Il utilise le module Ordi pour la fonction de conseil.

Ordi est un module qui correspond aux joueurs gérés par l'ordinateur. Il détermine la place où va jouer l'ordinateur. Il utilise les modules Action et Verification.

Action est un module où sont définies les fonctions qui jouent et déjouent les coups.

Verification est un module qui vérifie si un joueur peut jouer. C'est lui qui indique à l' Arbitre quel joueur doit jouer ou alors si la partie est finie.

Global est le module qui contient la déclaration des variables et structures globales dont ont besoin tous les autres modules.

Affichage est un module qui gère l'affichage du plateau ainsi que celui des scores et le résultat du jeu.

2) Détail des modules

a) Arbitre

Fonction main : Entrée : rien
Sortie : Entier qui indique le bon fonctionnement du jeu.

Elle initialise les variables.
Elle boucle tant que l'utilisateur veut jouer de nouvelles parties.
Elle appelle Choisir_joueur (qui demande de quelle sorte sont les joueurs, remplit le tableau ref et initialise les tableaux d'heuristique en conséquence), Init (qui initialise le plateau). Elle initialise le premier joueur (joueur noir).

Elle contient la boucle principale :
- Affichage du plateau et des scores actuels : Affiche_jeux et Affiche_score.
- Elle appelle la fonction Verifie (qui indique qui doit jouer). Si Verifie renvoie 0, elle arrête la partie (sort de la boucle).
- Elle appelle Traiter_coup (qui demande le coup, joue et retourne le nombre de pions retournés ou 0 si l'utilisateur a voulu déjouer un coup).
- Si Traiter_coup renvoie un résultat non nul, elle appelle Gestion_scores (qui gère les scores) puis passe au joueur suivant.

Enfin, elle appelle Affiche_score_final (qui s'occupe de la fin de la partie) et appelle Nouvelle_partie (qui demande si l'utilisateur veut jouer une nouvelle partie).

Procédure Choisir_joueur : Entrée : Rien

Elle demande de quelle sorte sont les joueurs. Si un ou des joueurs sont définis comme étant des ordinateurs, elle appelle la procédure Init_heur (qui initialise les tableaux d'heuristique).
Si au moins un des joueurs est humain, elle demande si le joueur veut activer le conseil (et met à un le flag Conseil le cas échéant) et appelle Init_heur34 pour le(s) joueur(s) humain(s).

Procédure Choix_heur : Entrée : Entier référence qui désigne quel tableau on doit utiliser (0 = HeurA, 1 = HeurB)

Renvoie un pointeur sur le tableau (HeurA ou HeurB) à utiliser.

Procédure Init_heur : Entrée : Entier référence qui désigne quel tableau on doit utiliser (0 = HeurA, 1 = HeurB)

1 = Niveau 1 ; 2 = Niveau 2 ; 3 = Niveau 3 ; 4 = Niveau 4.
Appelle Choix_heur (renvoie un pointeur sur le tableau d'heuristique HeurA ou HeurB selon le cas).
Appelle Init_heur2 si l'ordinateur est de niveau 2.
Appelle Init_heur34 si l'ordinateur est de niveau 3 ou de niveau 4.

Procédure Init_heur2 : Entrée : Plateau qui est un pointeur sur le tableau d'heuristique à initialiser

Charge le fichier heur2 et initialise le tableau d'heuristique pointé avec les valeurs lues.

Procédure Init_heur34 : Entrée : Plateau qui est un pointeur sur le tableau d'heuristique à initialiser

Charge le fichier heur34 et initialise le tableau d'heuristique pointé avec les valeurs lues.

Procédure Init : Entrée : Rien

Initialise le plateau de jeu : met 0 partout dans le tableau M sauf au centre où elle met un entier qui correspond au joueur A ou B.
Initialise les scores.

Procédure Gestion_scores : Entrée : joueur C et Entier n. C est le joueur (A ou B) qui a gagné n pions.

Elle ajoute les n pions au score de C et celui est posé (+1) et retranche les n pions à l'autre joueur.

Procédure Ajuster_scores : Entrée : joueur C et Entier n. C est le joueur (A ou B) qui a gagné n pions au coup à déjouer.

Elle enlève les n pions au score de C et celui a été posé (+1) et ajoute les n pions à l'autre joueur.

Fonction Traiter_coup : Entrée : joueur C
Sortie : Entier n : nombre de pions retournés

Elle appelle HumainouOrdi pour savoir qui joue.
Si c'est un humain, elle appelle Demander_coup (appelle le joueur humain).
Si c'est un ordinateur, elle appelle Determination (appelle le joueur ordinateur).

Si Demander_coup a retouné une demande pour déjouer le coup (p.x = 9), elle déjoue les coups précédents de l'ordinateur (s'ils existent) jusqu'à rencontrer un coup jouer par un humain qu'elle déjoue aussi. Pour déjouer, elle appelle les fonctions Dejouer (enlève les pions) et Ajuster_scores (recalcule les scores). Elle décrémente aussi histcur.

Elle appelle Joue (qui tente de jouer le coup donné : renvoie 0 si le coup n'est pas possible).
Si joue renvoie 0, elle redemande le coup (boucle).

Si aucun humain ne joue (ref[0] et ref[1] non nuls) elle appelle My_pause (attend que l'utilisateur appuie sur entrée avant de continuer).

Procédure My_pause : Entrée : Rien

Attend que l'utilisateur appuie sur la touche Entrée.

b) Humain

Fonction Demander_coup : Entrée : joueur C
Sortie : Pion : celui que l'humain veut poser

Elle demande un coup au joueur C.

Si le joueur renvoie 9 pour x, elle renvoie le pion (9,0) (pour que Traiter_coup puisse déjouer le coup).
Si le joueur renvoie 0 pour x, elle change la référence de ce joueur pour appeler Determination comme si c'était le joueur de niveau 4 qui jouait puis rétablit la référence (le coup que jouerait l'ordinateur s'affiche dans Determination).

Si le coup demandé est dans le plateau, elle retourne le pion. Sinon, elle redemande (boucle).

c) Ordi

Fonction Determination : Entrée : joueur C
Sortie : pion : celui que l'ordinateur va jouer

Pour chaque place p du plateau :
Elle appelle Verifie_coup (vérifie si un pion peut être posé a la place p)
Si p est possible :
Elle appelle Alpha_beta (ou Min_Max) qui donne la « valeur » du coup
Si cette « valeur » est meilleure que la meilleure déjà enregistrée, elle enregistre la place p : elle appelle la fonction Copier (qui copie un pion) pour le mettre dans pbest.

Elle rend pbest (meilleure place).

Fonction Alpha_beta : Entrée : joueur C, joueur moi, pion p, Doubles alpha, beta, Entier ncoup et Double gprec. C est le joueur pour lequel est déterminé le coup à prévoir. moi est le joueur courant (celui qui cherche le meilleur coup). p est le pion testé. alpha et beta sont les valeurs courantes d'alpha et de beta. ncoup est le nombre de coups restants à jouer. gprec est le gain précédent.
Sortie : Double : « valeur » du coup joué

Il s'agit de l'algorithme alpha_beta vu dans le cours adapté à notre jeu :

Elle appelle Joue_Ordi (simule la pose d'un pion et retourne le gain que cela rapporterait)
Elle appelle HumainouOrdi pour savoir quel est le niveau du joueur moi.
Elle appelle Verifie pour savoir qui joue après

Si le prochain joueur est le même que C et que moi est de niveau 3 ou 4, alors elle augmente le gain (car un joueur passe alors son tour).

Si on est à la fin de la partie ou à la profondeur maximale
- Elle appelle Dejoue (déjoue le coup simulé)
- Si c'est la fin de la partie retourne une très bonne valeur si la partie est gagnée et une très mauvaise sinon
- Sinon elle retourne le résultat précédent majoré du nouveau gain.

La suite consiste à rappeler Alpha_Beta pour toutes les cases possibles avec un gain précédent ajouté ou retranché avec le nouveau gain et à changer les valeurs d'alpha et de bêta en conséquence pour éviter de regarder les branches dans lesquelles il n'est pas utile de voir ce qu'il se passe.

Elle rend la « valeur » du coup.

Fonction Min_Max : Entrée : joueur C, joueur moi, pion p, Entier ncoup, Double gprec. C est le joueur pour lequel est déterminer le coup à prévoir. moi est le joueur courant (celui qui cherche le meilleur coup). p est le pion testé. ncoup est le nombre de coups déjà joués. gprec est le gain précédent.
Sortie : Double : « valeur » du coup joué

Il s'agit de l'algorithme min_max vu dans le cours adapté à notre jeu. Elle rend la « valeur » du coup.
Le principe est le même que pour Alpha_Beta

d) Action

Fonction Joue : Entrées : joueur C, pion p ; C indique le joueur qui veut poser p .
Sortie : Entier n qui est le nombre de pions retournés.
n est initialisé à 0.
Cette fonction vérifie pour chaque direction si le pion à poser retourne des pions => Boucle :
Si oui : elle retourne les pions à retourner, les enregistre dans l'historique au coup courant et incrémente n pour chacun.
A la fin de la boucle, avec n différent de 0, on enregistre le pion posé, n et le joueur C dans l'historique.
Elle appelle Modif_heur si un joueur est de niveau 4 ou l'option conseil est activée.
A la fin, on retourne n.

Fonction HumainouOrdi : Entrée : joueur C
Sortie : Entier qui désigne de quel sorte est le joueur C

Elle détermine le type (humain ou ordinateur avec son niveau) du joueur C.

Fonction Copier : Entrée : pion (à copier)
Sortie : pion (copié)
Elle copie un pion dans un autre.

Fonction ChoixHeuris : Entrée : joueur C : celui dont on cherche le tableau d'heuristique (équivalent à un plateau)
Sortie : un pointeur sur le plateau (Plateau)

Elle indique quel plateau d'heuristique utilisé pour le joueur et renvoie un pointeur sur lui.

Fonction Joue_Ordi : Entrées : joueur C, pion p, joueur moi, entier ncoup ; C indique le joueur qui poserait p (C peut soit être l'Ordi qui appelle cette fonction, soit le joueur adverse auquel on simule le coup). Le calcul de l'heuristique du coup joué est basé sur les valeurs du tableau d'heuristique du joueur (moi) appelant.
ncoup désigne le nombre de coups simulés par l'ordinateur.
Sortie : Entier g qui est la valeur du coup joué
Donnée/Résultat : (par passage par adresse) le coup joué m.
(g est initialisé à 0.)
Cette fonction vérifie pour chaque direction si le pion à poser retourne des pions => Boucle :
Si oui : elle retourne les pions à retourner, les enregistre dans le coup m et incrémente n pour chacun.
A la fin de la boucle, avec n différent de 0, on enregistre le pion posé, n et le joueur C dans le coup m.
Appelle la fonction Evaluation qui calcule le gain en fonction du coup j et du joueur moi) qui nous donne g.
Appelle la fonction Modif_Heur (qui modifie l'heuristique s'il y a un joueur de niveau 4).
A la fin, on retourne g (résultat de Evaluation).

Procédure Dejouer : Entrées : coup D, qui indique le coup à déjouer
Elle appelle Demodif_Heur (démodifie l'heuristique s'il y a lieu).
Elle retourne les pions retournés (pionret) du coup D et retire le pion posé (ppose).

Fonction Evaluation : Entrées : coup m, joueur moi, entier ncoup : On a le coup j à évaluer, le joueur moi qui va permettre de savoir quel tableau d'heuristique utiliser et ncoup qui désigne le n° du coup simulé alors.
Sortie : Double gain
On récupère le niveau du joueur (HumainouOrdi). S'il est de niveau 1 on renvoie le nombre de pions gagnés.
Sinon :
On charge le bon tableau d'heuristique (ChoixHeuris).
On ajoute les valeurs des cases des pions retournés. On fait une moyenne de ces valeurs. Puis on effectue une moyenne (g) entre cette dernière valeur et l'heuristique de la case où est posé le pion pour le coup m.

Si moins de 20 coups ont été joués, on retourne g.
Sinon :
On calcule un coefficient (coeff) qui correspond à l'importance accordée à g par rapport au nombre de pions retournés.
On retourne alors la moyenne entre g et le nombre de pions retournés, pondérée par coeff.

Fonction Modif_Heur : Entrées : joueur C, joueur moi, pion p : moi désigne le joueur auquel le tableau d'heuristique va être modifié, C celui qui joue et p le pion posé.
Sortie : entier F, indique si il y a eu modification ou non (1= oui, 0=non)

Active dans le cas d'un niveau 4 ou si l'option conseil est activée , elle modifie le tableau d'heuristique en fonction du pion p posé.
Elle appelle ChoixHeuris pour déterminer le tableau d'heuristique à modifier.
Elle appelle HumainouOrdi pour déterminer si « moi » est de niveau 4 (lors d'un conseil, l'humain est référencé avec un niveau 4).

Procédure Demodif_Heur : Entrées : joueur C, joueur moi, pion p : moi désigne le joueur auquel le tableau d'heuristique va être démodifié, C celui qui joue et p le pion posé.

Appelée dans Dejouer, elle modifie l'heuristique du joueur moi en fonction du pion p posé.
Elle appelle ChoixHeuris pour déterminer le tableau d'heuristique à modifier.
Elle appelle HumainouOrdi pour déterminer si « moi » est de niveau 4 (lors d'un conseil, l'humain est référencé avec un niveau 4).

e) Verification

Fonction Verifie : Entrée : joueur C : celui pour qui on vérifie s'il peut jouer ou non
Sortie : joueur C : c'est celui qui doit jouer (A ou B) ou 0 si la partie est finie

Appelle Question_coup (vérifie si le joueur C peut jouer : renvoie 0 s'il ne peut pas)
Si elle a reçu 0, elle change de joueur et rappelle Question_coup.
Si elle reçoit encore 0, elle renvoie 0
Sinon elle renvoie la couleur (1 ou –1) du joueur qui peut jouer.

Fonction Question_coup : Entrée : joueur C : celui pour lequel on demande s'il peut jouer
Sortie : Entier v : vrai (1) ou faux (0) selon si C peut jouer ou non

v est initialisé à faux.
Boucle :
Pour toutes les cases et tant que v est faux:
v reçoit la valeur de Verifie_coup (test si un pion peut être posé à cette case)
Renvoie v.

Fonction Verifie_coup : Entrée : joueur C, pion p : on test si le joueur C peut poser le pion p
Sortie : Entier v : oui (1) ou non (0) selon si le pion peut être posé ou non

v est initialisé à faux.
Boucle :
Pour toutes les directions et tant que v est faux
Test si p peut retourner des pions dans la direction (si les pions alignés dans la direction sont de la couleur opposée de C et si un pion de couleur C clôt l'alignement).
Si le test (v) est vrai, on sort de la boucle.
On sort aussi de la boucle quand on a épuisé toutes les directions.
On retourne v qui est égal à 1 si un des test dans la boucle a été vrai, sinon v est égal à 0.

f) Affichage

Procédure Affiche_jeux : Entrée :rien
Cette procédure affiche le plateau dans son état courant.
Elle affiche ligne par ligne avec des indications pour indiquer les coordonnés du plateau

Procédure Affiche_score : Entrée : rien
Cette procédure affiche les scores courants des 2 joueurs.

Procédure Affiche_score_final : Entrée :rien
Cette procédure affiche le résultat du jeu : Elle détermine lequel des 2 joueurs a gagné où s'il y a eu match nul.

3) Structure du programme Ecriture

Il est constitué d'un seul module : Ecrire.

Ce programme va créer 2 fichiers : heur2 et heur34 dans le répertoire courant.
(attention si un fichier au moins existe, on a alors une concaténation de fichier ; d'où l'intérêt de la commande « make propre » qui va effacer ces 2 fichiers s'ils existent dans le répertoire courant.)

Ce programme a 2 tableaux d'heuristique : Heurist2 et Heurist34 qui vont être écrit respectivement dans heur2 et heur34.

ANNEXE

NIVEAUX DE JEU DE L'ORDINATEUR

Afin de permettre à l'utilisateur de se confronter à plusieurs adversaires, nous avons choisi de lui fournir 4 niveaux de jeu. Ces niveaux sont complètement différents et permettent ainsi de voir plusieurs tactiques de jeu.

Niveau 1 (Débutant)

Ce niveau est très facile à battre. Il est conseillé pour ceux qui viennent d'apprendre à jouer à Othello.

Sa stratégie consiste à jouer dans la case qui lui permettra de gagner le plus de pions possibles (ou d'en perdre le moins possible s'il ne peut en gagner) sur l'ensemble des trois prochains coups. Toutefois, si une configuration de jeu lui permet d'être sûr de gagner la partie (sur l'ensemble des coups qu'il regarde), il la choisira.

A l'utilisation, on s'aperçoit qu'il ne parvient que très rarement (jamais ?) à ce genre de situation et s'il gagne la partie, elle se finit avec le remplissage total du plateau.

Niveau 2 (Intermédiaire)

Ce niveau est déjà nettement plus difficile à battre. Il constitue une bonne approche d'un joueur qui commence à bien savoir jouer. Cependant, il commet encore de lourdes erreurs qui vous permettront de le battre sans trop de difficultés.

Il dispose d'un tableau d'heuristique lui indiquant quelles sont les meilleures cases à jouer en début de partie. Au bout du vingtième coup, il commence à donner de moins en moins d'importance à ce tableau pour favoriser le nombre de pions gagnés. Au bout du quarantième coup, c'est le nombre de pions gagnés qui prime dans tous les cas. Lui aussi préférera les configurations gagnantes à court terme (il regarde les quatre prochains coups).

A l'utilisation, ce joueur s'avère inefficace contre de « bonnes » stratégies de jeu. En réalité, sa propre stratégie n'est pas assez étudiée pour lui permettre de gagner rapidement. Il ne termine la partie en vainqueur avant le remplissage complet de l'othellier que dans de rares situations.

Niveau 3 (Confirmé)

Ce niveau de jeu permet des parties très intéressantes si vous savez bien jouer à Othello. L'ordinateur commet de moins en moins d'erreurs et ne vous laisse pas beaucoup d'ouvertures intéressantes.

Il dispose lui aussi d'un tableau d'heuristique mais qui est mieux adapté à sa fonction d'évaluation. Comme le niveau 2, sa stratégie lui impose de joueur une première moitié de partie basée surtout sur les heuristiques et une seconde moitié surtout sur les gains. Mais il veillera plus à conserver le coins pour lui. S'il tient en compte les positions gagnantes et perdantes à court terme (cinq coups de profondeur), il se laissera aussi tenter par les positions qui poussent l'adversaire à passer son tour.

Ce joueur s'avère être du même niveau que ses programmeurs : il nous a mis en difficulté plus d'une fois et ne s'est pas laissé gagner facilement. De plus, s'il nous laisse parfois gagner plus de pions à un tour, c'est peut-être pour mieux jouer les deux coups suivants : il lui arrive de jouer deux ou trois coups en nous faisant passer entre temps. De ce fait, il termine les parties beaucoup plus rapidement et il n'est pas rare de le voir gagner une partie avant le remplissage total de l'othellier. Méfiance donc !

Niveau 4 (Expert)

C'est le joueur du plus haut niveau que nous ayons pu implémenter. S'il lui arrive encore de commettre des fautes, ce n'est que face à d'autres joueurs de très bon niveau qui l'auront poussé à la faute. Cependant, il fera tout pour rattraper sa faute.

Du point de vue de la stratégie, il ressemble beaucoup au niveau 3. La seule différence vient du fait qu'il change les valeurs de son tableau d'heuristique selon l'évolution de la partie pour mieux coller à la position du jeu. Il peut donc ainsi tenter de rattraper d'éventuelles erreurs ou s'assurer un nombre de pions minimal à la fin de la partie. Enfin, il regarde les six prochains coups, ce qui lui permet d'être rarement pris au dépourvu.

A l'utilisation, nous ne sommes pas parvenus à le vaincre. Il a imposé une cuisante défaite à un débutant (63 à 0 !). Il n'est donc pas conseillé aux joueurs qui ne connaissent pas encore les bonnes tactiques de jeu. Un joueur humain est quand même parvenu à le battre, ce qui signifie qu'il garde un niveau de jeu respectable (en effet, s'il était impossible à battre, l'intérêt de jouer contre lui serait limité !).

Cependant, en le testant contre d'autres algorithmes (dont celui de François Colonna), il s'est avéré que l'on pouvait le pousser à la faute. Il laisse alors de trop belles ouvertures à son adversaire. Cependant, même dans ces cas là, il n'a jamais vraiment été battu en conservant moins de 20 pions en fin de partie. De plus, il bat le joueur de niveau 4 de colonna mais pas celui de niveau 5 qui n'ont pourtant qu'un coup de profondeur comme différence.

Afin de mieux vous persuadez de la différence entre les niveaux, nous les avons testé les uns contre les autres. Voici un récapitulatif des résultats :

Résultats des parties Ordinateur contre Ordinateur

Joueur Blanc \ Joueur Noir	1	2	3	4
1	43 - 21	50 - 14	49 - 0	57 - 2
2	18 - 46	26 - 38	38 - 26	55 - 9
3	2 - 60	26 - 38	43 - 21	53 - 11
4	5 - 46	13 - 51	18 - 46	56 - 7

Les scores sont de la forme ScoreNoir - ScoreBlanc

Les résultats énoncés précédemment sont bien visibles : niveaux 1 et 2 sont faits pour les débutants et terminent la partie une fois le plateau rempli. Les niveaux 3 et 4, bien plus performants, tentent des fins plus rapides.

CONCLUSION

Comme nous pouvons le voir, nous avons réussi à atteindre l'objectif fixé de réaliser un jeu d'information parfaite et de somme nulle.

Par ailleurs, son implémentation nous a été bénéfique dans le sens où elle nous a permis de mettre en avant nos compétences pour nous organiser (temps et répartition des tâches), nous entendre (savoir écouter les points de vue de chacun).

D'autre part, ce projet a fait appel à notre imagination et à la rationalisation de nos idées lors de la conception (savoir concrétiser une idée avec les possibilités qui nous étaient offertes par le langage imposé : le C).

Notre impression générale est maintenant une certaine satisfaction vis à vis de ce que nous avons pu réaliser et même un peu de fierté. ☺