DEVOIR DE PROGRAMMATION LOGIQUE

Rapport du devoir de swi-prolog

COULON Anthony
ROLLAND Cyrille
18/12/2009

Ingénieurs sup Galilée

INDEX

INTRODUCTION

Dans le cadre des études d'ingénieur en deuxième année de l'école Sup Galilée à l'université paris 13, il nous a été demandé d'effectuer un devoir de programmation logique.

Ce devoir a pour but de mettre en pratique l'ensemble des éléments de programmation logique étudiés en cours. Pour cela le devoir porte sur la création d'un jeu Othello en swi-prolog. Ce devoir se divise en deux parties principales :

- La première étant la création d'un programme qui joue aléatoirement des coups autorisés. Le but de cette partie est de nous familiariser avec les règles du jeu tout en exploitant les capacités de swi-prolog.
- La deuxième partie consiste en la réalisation d'un programme plus évolué doté d'une méthode heuristique. Cela lui permettra de jouer des coups plus intelligents qui vont le mener vers une victoire plus probable que le programme aléatoire.

Ce projet à été entièrement codé par COULON Anthony et ROLLAND Cyrille, il se compose de 3 répertoires nommés respectivement « aleatoire », « heuristique », « arbitre » et de notre rapport. Ce rapport a pour but de présenter l'ensemble des fonctionnalités implémentées afin de résoudre les différents problèmes.

I. Présentation du jeu Othello :

1. Le jeu :

Le jeu Othello se compose généralement d'un plateau de 64 cases. On y distingue deux joueurs, le premier possède les pions blancs et le deuxième possède les pions noirs. Comme indiqué dans l'énoncé de notre projet nous nous limiterons à un plateau de 36 cases.

Pour chaque joueur, le but du jeu est de posséder le maximum de pions de sa couleur en fin de partie. Une partie est terminée lorsqu'il n'y a plus de coup jouable, c'est-à-dire lorsque les joueurs ne peuvent plus poser de pion.

Un coup est dit jouable si et seulement s'il permet de prendre en sandwich des pions appartenant au joueur adverse. Un coup joué transforme les pions pris en sandwich en la couleur jouée. Les joueurs passent leur tour uniquement s'ils ne peuvent pas jouer.

Exemple de coup jouable :

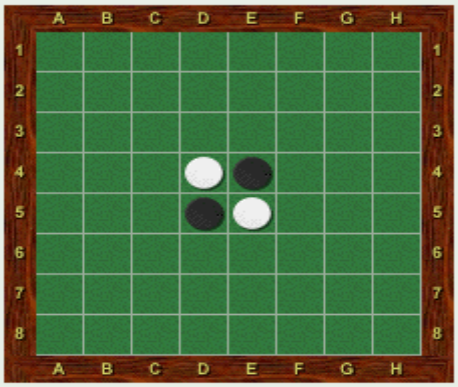

fig. 1 : position de départ

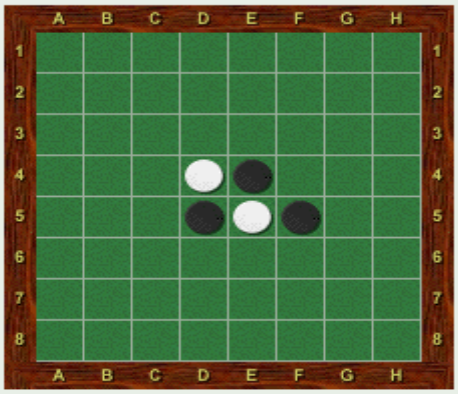

fig. 2 : Noir joue f5...

2. Le jeu d'un point de vue informatique :

La description précédente du jeu Othello nous permet de constater que l'implémentation d'un programme prolog devra posséder les fonctionnalités suivantes :

- Trouver les coups jouables (légaux).
- Jouer un coup sur le plateau.
- Déterminer si la partie est terminée.

Les plateaux sont gérés sous forme de liste de 36 cases comme suit :

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

On accède aux cases grâce à leurs indices. Chaque case est soit une variable, soit possède une valeur b pour blanc ou n pour noir. Nous avons choisi cette structure car elle permet une gestion simple du plateau.

Remarque : Cette façon de procéder étant différente que celle utilisée par l'arbitre fourni, il a fallu implémenter des prédicats permettant de convertir le type utilisé par l'arbitre en liste et inversement.

II. Présentation du programme joueur aléatoire :

Le joueur aléatoire est un joueur choisissant aléatoirement un coup dans la liste des coups jouables et de le jouer. Le but de l'implémentation d'un tel joueur est de se familiariser avec les règles du jeu et de coder les fonctions de base qui seront réutilisées pour le joueur heuristique. Ce joueur a été adapté afin de fonctionner correctement avec l'arbitre qui nous a été fourni.

Vous trouverez le code correspondant à ce joueur dans le répertoire « aléatoire » de notre projet.

1. Gestion des coups réalisables :

La gestion des coups réalisables se fait par le prédicat :

tous_coups_legaux (+Couleur,+Etat, -ListeCoupsAdmissibles).

Les arguments de ce prédicat sont :

- La couleur dans laquelle on cherche les coups jouables (b pour blanc, n pour noir)
- L'état du plateau dans lequel on cherche les coups jouables
- En retour on obtient une liste des coups jouables

Afin de simplifier et d'améliorer nos algorithmes, nous avons formé une liste des coups jouables formée comme suit :

La liste des coups jouables est composée de plusieurs listes dont le premier élément est la position de la case où le joueur joue et les autres éléments sont les positions des cases changent de couleur suite à ce coup.

Exemple :

```
Si la liste des coups jouables pour noir dans la position initiale est :
                  [[14,15],[23,22],[9,15],[28,22]]
(ici si on joue en 14 alors on retournera le pion de la position 15)
```

Ce type de listes permet d'avoir un prédicat joue_coup très simple à implémenter.

2. Comment un coup est il jouer :

Un coup est joué grâce au prédicat *joue_coup(+Couleur,+Etat,+Coup,-NouvelEtat)*

Les arguments de ce prédicat sont :

- *Couleur :* la couleur du joueur
- *Etat :* Le plateau courant
- *Coup :* le coup joué sous la forme d'une liste choisie dans la liste retournée par *tous_coups_legaux*
- *NouvelEtat :* Le nouveau plateau obtenu après avoir joué le coup *Coup*

Les plateaux sont gérés sous forme de listes. Ce prédicat doit juste remplacer les éléments référencés dans la liste *Coup* par des pions de couleur *Couleur*. En effet la simplicité de ce prédicat est due aux informations contenues dans la liste *Coup* qui contient l'ensemble des indices des pions à mettre à la couleur *Couleur*.

3. Le prédicat joueur_aleatoire :

Le prédicat *joueur_aleatoire(+Couleur,+Etat,-NewEtat,-CoupJoue)* est le prédicat principal du joueur aléatoire. En effet il fait appel aux deux prédicats vus précédemment et applique également un « random » sur la liste des coups jouables afin de rendre aléatoire le type de jeu.

Les arguments de ce prédicat sont :

- La couleur du joueur
- L'état du plateau avant qu'il joue
- Le nouvel état obtenu après qu'il ait joué
- Le coup joué par ce joueur

4. Quelques exemples de coups avec le joueur aléatoire

Voici un exemple pris durant une partie entre deux joueurs aléatoire :

```
  A B C D E F
1   @
2 @ @     O
3   @ @ O
4 O @ O @   O
5 @ O   @ O
6   O   O @ @

 Le coup 17 est joue en : d2

  A B C D E F
1   @
2 @ @   @ O
3   @ @ @
4 O @ O @   O
5 @ O   @ O
6   O   O @ @

 Le coup 18 est joue en : a1

  A B C D E F
1 O @
2 @ O   @ O
3   @ O @
4 O @ O O   O
5 @ O   @ O
6   O   O @ @

 Le coup 19 est joue en : f1

  A B C D E F
1 O @       @
2 @ O   @ @
3   @ O @
4 O @ O O   O
5 @ O   @ O
6   O   O @ @
```

III. Présentation du programme joueur heuristique :

Le joueur heuristique se différencie du joueur aléatoire par sa stratégie. En effet l'objectif cherché par l'implémentation d'un tel programme n'est pas uniquement de respecter les règles du jeu, comme le fait le programme joueur aléatoire, mais de jouer de façon à augmenter ses chances de victoire.

1. Gestion du joueur heuristique :

Le joueur heuristique est géré sur le même principe que « joueur_aleatoire ». Afin d'améliorer les coups joués, il utilise l'algorithme de l'alpha-beta vu en cours. Cette façon de faire nous oblige donc à implémenter une fonction d'évaluation pour mettre notre stratégie en place afin de noter les différents états obtenus par l'algorithme de l'alpha-beta.

Cependant la note attribuée à un état est le reflet de la stratégie que nous utilisons. Le joueur heuristique a été adapté pour fonctionner correctement avec le programme arbitre.

Le principal prédicat du programme joueur heuristique est :

joueur_h(+C,+Etat1,-Coup,-Value)

Les arguments de ce prédicat sont :

- La couleur des pions du joueur
- L'état courant du plateau de jeu
- Le coup à jouer d'après l'alpha-beta (au format liste vu dans la partie aléatoire)
- La valeur de l'évaluation de ce coup

2. Quelle stratégie avons-nous utilisée ?

La stratégie que nous avons utilisée se divise en deux phases principales :

1. Durant la majorité de la partie, le joueur heuristique tente de maximiser son nombre de coups jouables tout en favorisant la prise des bords et surtout des coins. Le but de cette phase est de créer un maximum de pions définitifs et de prendre les places stratégiques avant la deuxième phase. Pour cela nous avons affecté des notes aux cases situées sur les bords, la note attribuée à une case peu être différente selon la configuration du jeu.

2. La deuxième phase de notre stratégie consiste à maximiser le nombre de pions retournés tout en s'assurant que cela ne bloque pas le nombre de coups jouables. Cette phase est utilisée en fin de partie et se lance uniquement si l'ensemble des coins ont été pris.

Pourquoi avons-nous choisi cette stratégie ?

Nous avons adopté cette stratégie après l'avoir mise en concurrence avec cinq autres stratégies proches. Après différents tests et jeux effectués grâce au programme arbitre, cette stratégie a obtenu les meilleurs résultats.

Nous avons également testé cette stratégie contre notre « joueur_aléatoire » et les différents résultats sont sans appel comme vous pourrez le voir dans la partie suivante.

Des résultats sans appel ?

En effet cette heuristique a obtenu de larges victoires contre nos premières heuristiques.

3. Quelques exemples du fonctionnement du joueur heuristique :

Résultats de notre heuristique contre notre joueur aléatoire :

1. Ici heuristique en joue blanc (0)

```
  A B C D E F
1 0 0 0 0 0 0
2 0 0 0 0 0 0
3   0 0 0 0 0
4 0 @ 0 0 0 0
5   0 0 0 0 0
6 0 0 0 0 0 0

 Le coup 38 est joue en : a3

  A B C D E F
1 0 0 0 0 0 0
2 0 0 0 0 0 0
3 0 0 0 0 0 0
4 0 0 0 0 0 0
5   0 0 0 0 0
6 0 0 0 0 0 0

 Le coup 39 est joue en : passe

  A B C D E F
1 0 0 0 0 0 0
2 0 0 0 0 0 0
3 0 0 0 0 0 0
4 0 0 0 0 0 0
5   0 0 0 0 0
6 0 0 0 0 0 0
Le gagnant est blanc avec 35 contre 0 pions.
true
```

2. Ici heuristique joue en noir (@)

```
  A B C D E F
1 @ @ @ @ @ O
2 @ @ @ @ O
3 @ @ @ O @ @
4 @ O O O @ @
5 @ O O @ @ @
6 @ @ @ @ @ @

 Le coup 33 est joue en : f2

  A B C D E F
1 @ @ @ @ @ O
2 @ @ @ @ @ @
3 @ @ @ O @ @
4 @ O O O @ @
5 @ O O @ @ @
6 @ @ @ @ @ @

 Le coup 34 est joue en : passe

  A B C D E F
1 @ @ @ @ @ O
2 @ @ @ @ @ @
3 @ @ @ O @ @
4 @ O O O @ @
5 @ O O @ @ @
6 @ @ @ @ @ @
Le gagnant est noir avec 29 contre 7 pions.
```

On constate que le joueur heuristique obtient de large victoire face au joueur aléatoire. Il arrive même fréquemment que le joueur aléatoire soit battu avant que le plateau soit rempli comme le montre l'exemple 1.

Nous allons maintenant voir comment notre joueur se comporte face à nos anciennes heuristiques.

Exemples de coups échangés entre notre ancienne heuristique et notre heuristique finale :

Ici l'heuristique finale joue en blanc (0)

```
 Le coup 1 est joue en : b3

  A B C D E F
1
2
3   @ @ @
4     @ 0
5
6

 Le coup 2 est joue en : d2

  A B C D E F
1
2       0
3   @ @ 0
4     @ 0
5
6

 Le coup 3 est joue en : e3

  A B C D E F
1
2       0
3   @ @ @ @
4     @ 0
5
6

 Le coup 4 est joue en : b4

  A B C D E F
1
2       0
3   @ 0 @ @
4   0 0 0
5
6
```

```
  A B C D E F
1   @ 0 0 0
2     @ 0   0
3   @ 0 @ 0 0
4   @ 0 0 0 0
5   @ @ @ 0 0
6   @ @ @

 Le coup 22 est joue en : a1

  A B C D E F
1 0 0 0 0 0
2     @ 0   0
3   @ 0 @ 0 0
4   @ 0 0 0 0
5   @ @ @ 0 0
6   @ @ @

 Le coup 23 est joue en : e2

  A B C D E F
1 0 0 0 0 0
2     @ @ @ 0
3   @ 0 @ 0 0
4   @ 0 0 0 0
5   @ @ @ 0 0
6   @ @ @

 Le coup 24 est joue en : f1

  A B C D E F
1 0 0 0 0 0 0
2     @ @ 0 0
3   @ 0 0 0 0
4   @ 0 0 0 0
5   @ @ @ 0 0
6   @ @ @
```

On remarque que notre nouvelle heuristique joue plus intelligemment que l'ancienne et que tous les coups sont valides.

4. Les difficultés que nous avons rencontrées :

Nous avons réussi à terminer notre projet sans pour autant ne pas avoir rencontrés quelques soucis lors de sa réalisation. En effet lorsque nous avons reçu le programme arbitre nous avons dû développer des prédicats permettant de convertir les différents types.

De plus pour que notre programme fonctionne correctement avec l'arbitre nous avons dû réduire la profondeur de l'alpha-béta. Cependant cette réduction de la profondeur n'a pas grandement modifié le comportement de notre joueur heuristique. De plus il a fallu modifier le code de l'alpha-beta afin qu'il fonctionne avec notre code et gère le changement de joueur à chaque coup.

A ces deux principaux problèmes s'ajoutent de nombreux dysfonctionnement qu'il a fallu résoudre non sans peine dans certains cas.

Conclusion

Pour conclure nous constatons que le projet a été entièrement implémenté dans les temps et qu'il fonctionne correctement. Il se compose de deux programmes qui sont « le joueur aléatoire » et « le joueur heuristique ». Ce projet a été entièrement réalisé par COULON Anthony et ROLLAND Cyrille dans le cadre du devoir de programmation logique.