Dames / Prises multiples / Evaluation min-max

Tristan Colombo 2002

Tout d'abord il va falloir créer 2 nouveaux fichiers : min_max.c et min_max.h et modifier le makefile et dames.c pour qu'ils acceptent ces nouveaux fichiers.

```
/* min_max.h */
#ifndef MIN_MAX_H

#define MIN_MAX_H
#include <stdio.h>
#include "jeu_dames.h"
#define PROFMAX 3
#define MAXI    20000

int evaluation(Tdamier dm);
…

#endif
```

```
/* min_max.c */
#include "min_max.h"

int evaluation(Tdamier dm)
{
…
}
```

```
/* dames.c */
…
#include « min_max.h »
…
```

```
/* Makefile */
OBJECTS = dames.o jeu_dames.o min_max.o

all: Programme

CFLAGS = -g -Wall -O2 `gtk-config --cflags`

%.o: %.c
	gcc -c $(CFLAGS) $< -o $@

Programme: $(OBJECTS)
	gcc $(OBJECTS) -o dames `gtk-config --libs` -g

clean:
	@rm -f *o dames

dames.o: dames.c jeu_dames.c

jeu_dames.o: jeu_dames.c

min_max.o: min_max.c
```

1. Evaluation / Parcours d'arbre de jeu min_max

1.1 Fonction d'évaluation :

Cette fonction va renvoyer une variable entière « valeur », initialisée à 0 puis, on incrémente « valeur » en parcourant tout le damier. Si l'on tombe sur :

- un pion noir → -100
- une dame noire → -1000
- un pion blanc → +100
- une dame blanche → +1000

1.2 Choix du coup – Fonction min-max

Tout d'abord, vous allez avoir besoin de deux fonctions : une fonction qui effectue la copie du damier dans un autre tableau et une fonction qui libère la mémoire occupée par la liste des coups lorsque l'on n'utilise plus celle-ci. Voici les en-tête de ces fonctions :

```
void copier_damier(Tdamier dm, Tdamier dmc)
/* Effectue une copie du damier dm dans dmc */

void disposer_liste(Tliste l)
/* Vide la liste l */
{
  if (l!=NULL)
  {
    disposer_liste(l->multiple);
    disposer_liste(l->suiv);
    free(l);
  }
}
```

Il y a également une modification à apporter à la fonction « ajoute_tete_liste » : le champ multiple doit être initialisé à NULL :

```
Tmp->multiple=NULL;
```

Voici l'algorithme de la fonction min_max :

```
int min_max(Tcoup cp, Tdamier dm, Tcouleur camp, int pr)
{
  copier_damier(dm, dmc)
  execute_coup(cp, dmc)
  coups_possibles3(dm, adversaire(camp), &l)
  Si l == NULL
  {
    Si camp == BLANC
      Return MAXI
    Sinon
      Return -MAXI
  }
  Sinon
  {
    Si pr == PROFMAX
      Return evaluation(dmc)
    Sinon
    {
      Si adversaire(camp) == NOIR
      {
        /* Minimisation */
        min ← MAXI
        c ← l
        Tant que c ≠ NULL Faire
        {
          coup.depart ← c->depart
          coup.arrivee ← c->arrivee;
         m ← min_max(coup, dmc, adversaire(camp), pr+1)
         if m < min
           min ← m
         c ← c->suiv
        }
        Return min
      }
      Sinon
      {
        /* Maximisation */
        max ← -MAXI;
        c ← l;
```

```
        Tant que c ≠ NULL Faire
        {
          coup.depart ← c->depart
          coup.arrivee ← c->arrivee
          m ← min_max(coup, dmc, adversaire(camp), pr+1)
          if m > max
            max ← m
          c ← c->suiv
        }
        Return max
      }
    }
    disposer_liste(l)
    l=NULL
  }
}
```

La fonction choix_coup effectue en fait le calcul pour la première profondeur de min-max puis appelle min-max :

```
void choix_coup(Tcouleur camp, Tdamier dm, Tliste l, Tcoup *mcp, int *valeur)
{
  Si camp==BLANC
  {
    /* Maximisation */
    max ← -MAXI;
    c ← l;
    coup.depart ← c->depart
    coup.arrivee ← c->arrivee
    *mcp ← coup
    Tant que c ≠ NULL Faire
    {
      coup.depart ← c->depart
      coup.arrivee ← c->arrivee
      m ← min_max(coup, dm, camp, 1)
      Si m > max
      {
        max ← m
        *mcp ← coup
      }
      c ← c->suiv
    }
    *valeur ← max
  }
  Sinon
  {
    /* Minimisation */
    min ← MAXI
    c ← l
    coup.depart ← c->depart
    coup.arrivee ← c->arrivee
    *mcp ← coup
    Tant que c ≠ NULL Faire
    {
      coup.depart ← c->depart
      coup.arrivee ← c->arrivee
      m ← min_max(coup, dm, camp, 1)
      if m < min
      {
        min ← m
        *mcp ← coup
      }
```

```
      c ← c->suiv
    }
    *valeur ← min
  }
}
```

2. Gestion des dames

2.1 Primitives de déplacement des dames :

On peut s'inspirer des fonctions de déplacement des pions (mais une dame à le droit de se déplacer en arrière …) en regardant si les cases p+1 à p+n sont vides.

Déplacement général des dames :

2.2 Primitives de prise des dames :

Là encore, comme pour le déplacement, on peut s'inspirer des fonctions sur la prise des pions. Pour tester si les cases précédant la prise sont vides nous utiliserons les fonctions de déplacement des dames définies précédemment.

3. Générateur de coups de pions – prises multiples

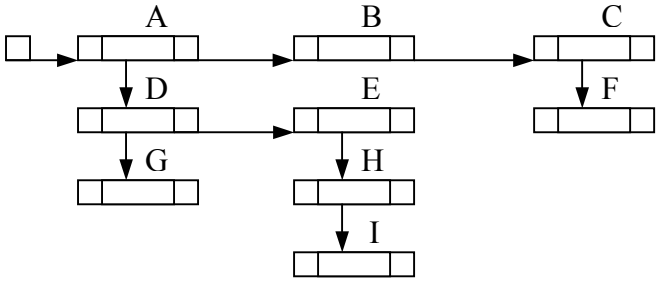
3.1 Primitives sur les listes avec coups multiples

La primitive disposer_liste a déjà été écrite précédemment pour la fonction min-max.

La primitive longueur_coup vous a été donnée en cours :

```
int longueur_coup(Tcoup cp)
{
  int longueur=1;
  Tliste l=cp.multiple;
  while (l!=NULL)
  {
    longueur++;
    l=l->multiple;
  }
  return(longueur);
}
```

Pour calculer la profondeur d'une liste l nous allons créer une fonction récursive :
Condition d'arrêt : Si l == NULL la profondeur vaut 0
Sinon on descend d'un cran et on calcule la profondeur de l->multiple. On calcule également la profondeur de l->suiv et on renvoi le maximum de ces valeur.



Sur l'exemple ci-dessus, prof(l)=max(prof(A), prof(B), prof(C)). Or prof(B)=0, prof(C)=1+prof(D)=1, et prof(A)=1+max(prof(D), prof(E)).
On a prof(D)=1+prof(G)=1 et prof(E)=1+prof(H)=1+1+prof(I)=2.
Donc prof(l)=3.
L'algorithme est donc le suivant :

```
int profondeur_liste(Tliste l)
{
  Si l == NULL
    Return 0
  Sinon
    Return max(profondeur_liste(l->multiple)+1, profondeur_liste(l->suiv))
}
```

Note : Il faut bien sûr écrire la fonction max …

3.2 Génération de coups multiples

La fonction prises_multiples calcule à partir des prises (sous forme de coup) données par l, les prises multiples possibles et les greffe sous la liste :

```
void prises_multiples(Tdamier dm, Tliste l)
{
  copier_damier(dm, dmc)
  cp.depart ← l->depart
  cp.arrivee ← l->arrivee
  execute_coup(cp, dmc)
  ltrav ← NULL
  Si dmc[l->arrivee.r][l->arrivee.c].piece == PION
    coups_possibles3(dm, dmc[l->depart.r][l->depart.c].coul, &ltrav)
  /* Sinon Si DAME … */
  l->multiple ← ltrav
  Tant que ltrav ≠ NULL Faire
  {
    prises_multiples(dmc, ltrav)
    ltrav ← ltrav->suiv
  }
}
```