TD 4 – Récursivité-Le jeu de Nim

Structures de données (IF 122)

Le jeu des allumettes (Nim ou jeu de Marienbad)

Le jeu se joue à deux. Plusieurs allumettes (représentées par des 1) sont disposées sur un certain nombres d'emplacements (classiquement 21, mais il est conseillé de tester d'abord sur des valeurs plus petites).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1			1	1	1	1				1	1				1	1		1	1	1

Au début du jeu tous les emplacements sont pris. À chaque tour vous pouvez retirer une, deux ou trois allumettes se situant dans des cases consécutives. Par exemple, si on est dans la configuration du dessus, on peux retirer les allumettes 4 et 5, ou les allumettes 18, 19 et 20.

Le but du jeu est de faire retirer la dernière allumette par son adversaire. Autrement dit, celui qui ramasse la dernière allumette perd.

Structures de données

Une position de jeu sera décrite par :

```
class Damier{
    int longueur; // taille du plateau
    boolean[] damier;
    boolean joueur; // true ordi, false humain
}
```

Et un coup à jouer par

```
class Coup {
    int position; //case ou on commence
    int nombre; //nbre d'allumettes a enlever
}
```

Exercice 1 (TD) Écrire une fonction itérative `static boolean coupPossible (Damier partie, Coup coup)` qui vérifie si le coup `coup` est possible dans la position donnée.

Exercice 2 (TP) Écrire une fonction itérative `static Damier jouerCoup (Damier partie, Coup coup)` qui retourne la nouvelle position du jeu, après le coup `coup` (on peut supposer que le coup est possible). Il s'agit, en d'autres termes, d'effacer des allumettes (mettre à `false` des cases du damier). Attention : ne pas oublier aussi de changer de main (les joueurs s'alternent).

Exercice 3 (TP) Une position est *terminale* lorsque il n'y a plus d'allumettes. Écrire une fonction `static boolean terminale(Damier partie)` qui dit si la position est terminale.

Exercice 4 (TD) Il est possible de déterminer si une position `p` du jeu est *gagnante* pour le joueur qui a la main, par un raisonnement **récursif**. En effet, si la position est terminale le joueur en question a gagné, puisqu'au tour précédent, son adversaire a enlevé la dernière allumette. En revanche, si la position n'est pas terminale, alors il faut explorer tout les coups possibles : si on trouve un coup conduisant à une position `p'` perdante (non *gagnante*...) pour son adversaire, la position `p` est *gagnante*, et vice-versa. Suivant cette idée, écrire une fonction récursive : `static boolean gagnante (Damier partie)` (ou `static boolean gagnante (boolean [] damier)`).

Exercice 5 (TD) Écrire une fonction qui calcule le meilleur coup pour l'ordinateur : `static Coup meilleur (Damier partie)`, il suffit d'adapter la procédure `gagnante` ci-dessus.

Exercice 6 (TP) Écrire un constructeur `Damier (int l, boolean j)` qui intialise une position (avec des allumettes à toutes les cases), avec un tableau de longueur `l` et le joueur `j` qui commence. On écrira aussi un constructeur pour la classe `coup`.

Exercice 7 (TP) Écrire une procédure d'affichage d'une position, qui produit quelque chose du genre :

```
----------------------------------------
|0   |1   |2   |3   |4   |5   |6   |7   |8   |9   |
----------------------------------------
|    |    | *  | *  |    |    |    |    | *  | *  |
----------------------------------------
c'est a l'humain(e) de jouer
```

Exercice 8 (TP) Écrire alors une procédure qui réalise la boucle d'interaction du jeu, c'est-à-dire la boucle où soit, la position courante est affichée et l'utilisateur est sollicité pour entrer son coup lorsque c'est à lui de jouer, soit la fonction `meilleur()` est appelée, lorsque c'est au tour de la machine. Cela jusqu'à la fin du match.

Vous pouvez aussi, écrire une procédure qui fait jouer l'ordinateur contre lui-même.

▶ **Exercice 9**

La correction n'est pas a 100

```
import fr.jussieu.script.Deug;

class Damier{
    int longueur;
    boolean[] damier;
    boolean joueur; // true ordi, false humain

    Damier (int l, boolean j) {
    longueur = l;
    joueur = j;
    damier = new boolean[longueur];
    for (int  i = 0; i < longueur; i++) {
        damier[i] = true;
    }
    }

    static void affiche (Damier partie) {
    int l = partie.longueur;
    for (int  i = 0; i < l; i++) {
        Deug.print("----");
    }
    Deug.println();
    Deug.print("|");
    for (int  i = 0; i < l; i++) {

        Deug.print(i);
        if (i<10)
        Deug.print("  |");
        else
        Deug.print(" |");
    }
        Deug.println();
    for (int  i = 0; i < l; i++) {
        Deug.print("----");
    }
    Deug.println();
    Deug.print("|");
    for (int  i = 0; i < l; i++) {
        if (partie.damier[i] == false)
        Deug.print("   ");
        else
        Deug.print(" * ");
        Deug.print("|");
    }
        Deug.println();
    for (int  i = 0; i < l; i++) {
        Deug.print("----");
    }
    Deug.println();
    if (partie.joueur)
        Deug.println("c'est a l'ordinateur de jouer");
    else
        Deug.println("c'est a l'humain(e) de jouer");
```

```
 Deug.println();
 Deug.println();

 }

 static boolean terminale(Damier partie) {
 for (int i = 0; i < partie.longueur; i++) {
     if (partie.damier[i] == true)
     return false;
 }
 return true;
 }

 // recursif
 static boolean coupPossible (Damier partie, Coup coup){
 //verification de la coherence des arguments
 if ((coup.nombre > 3) || (coup.nombre < 1))
     return false;
 if  ((coup.position >= partie.longueur) || (coup.position < 0))
     return false;

 // si pas d'allumette a la position voulue
 if ( partie.damier[coup.position] == false)
     return false;
 else {
     if (coup.nombre == 1)  // si j'ai fini
     return true;
     else {
     coup = new Coup(coup.position+1, coup.nombre -1);
     return coupPossible (partie,coup ); // je recommence avec les autres allumettes
     }
 }
 }

 //iteratif
static Damier jouerCoup (Damier partie, Coup coup){
 if (!(coupPossible(partie, coup)))
     return partie;

 int l = partie.longueur;
 Damier res = new Damier(l, !(partie.joueur));
 for (int i = 0; i < l; i++) {
     res.damier[i] = partie.damier[i];
 }
 for (int i = coup.position; i < coup.position + coup.nombre; i++){
     res.damier[i] = false;
 }
 return res;
}
 static Coup  premierCoupPossible( Damier partie) {
 // on prend la premiere case pas vide

 int l = partie.longueur;
 for (int i = 0; i < l; i++) {
     if(partie.damier[i] == true) {
     return  new Coup(i,1);
```

```
    }
}
return new Coup(0,0);
}

// ne sauvegarde pas le bon coup
static boolean gagnante (Damier partie) {
if (terminale (partie))
    return true;
int l = partie.longueur;
for (int  pos = 0; pos < l; pos++) {
    for ( int nb = 1; nb <= 3; nb++) {
    Coup coup = new Coup (pos, nb);
    if (coupPossible(partie, coup)){
        Damier res = jouerCoup(partie, coup);
        if (!(gagnante(res)))
        return true;
    }
    }
}
return false;
}

//
static Coup meilleur (Damier partie) {
if (terminale (partie)) {
    return (new Coup(0,0));
}
int l = partie.longueur;
for (int  pos = 0; pos < l; pos++) {
    for ( int nb = 1; nb <= 3; nb++) {
    Coup coup = new Coup (pos, nb);
    if (coupPossible(partie, coup)){
        Damier res = jouerCoup(partie, coup);
        if (!(gagnante(res))){
        return coup;
        }
    }
    }
}
return new Coup(0,0);
}
static void jeuOrdiOrdi(Damier partie){
    Coup coup;
    while ( !(terminale(partie))) {
        affiche (partie);
    coup =meilleur(partie);
        if (coup.nombre != 0) { // il y a un meilleur coup: on le joue
    partie = jouerCoup(partie, coup);
            Deug.println( " meilleur coup = "+ coup.position+ " " + coup.nombre);
        }
        else {
            coup = premierCoupPossible( partie); // sinon on prend celui qu'on trouve
            partie = jouerCoup(partie, coup);
            Deug.println( " premier coup = "+ coup.position+ " " + coup.nombre);
```

```
            }
        }
        affiche (partie);
    }

    static void jeuHumainOrdi(Damier partie){
    Coup coup;

        while ( !(terminale(partie))) {
            affiche (partie);
        if ( partie.joueur){ //si c'est a l'ordi de jouer
        coup =meilleur(partie);
        if (coup.nombre != 0) { // il y a un meilleur coup: on le joue
            partie = jouerCoup(partie, coup);
            Deug.println( " meilleur coup = "+ coup.position+ " " + coup.nombre);
        }
        else {
            coup = premierCoupPossible( partie); // sinon on prend celui qu'on trouve
            partie = jouerCoup(partie, coup);
            Deug.println( " premier coup = "+ coup.position+ " " + coup.nombre);
        }
        }
        else { //c'est a l'humain de jouer
        int pos; int nb;
        Deug.print("Donner la position de la premiere allumette:");
        pos = Deug.readInt();
        Deug.print("Combien d'allumette:");
        nb =  Deug.readInt();
        coup = new Coup(pos, nb);
        partie = jouerCoup(partie, coup); // pas de verification
        }
    }
    affiche (partie);
    if (partie.joueur)
        Deug.println("l'ordinateur a gagne");
    else
        Deug.println("le joueur a gagne");
    }
}

class Coup {
    int position; //case ou on commence
    int nombre; //nbre d'allumettes a enlever

    Coup ( int pos, int nb) {
    position = pos;
    nombre = nb;
    }
}


class Nim{
    public static void main (String[] args) {
    int l;
    Damier partie;
    Deug.print("donner la taille du damier: ");
```

```
    l = Deug.readInt();



    Deug.print("Jeu ordi/ordi (taper 1)  ou jeu humain/ordi (taper 2) ");
    if (Deug.readInt() == 1) {
        partie = new Damier (l, true);
        Damier.jeuOrdiOrdi(partie);
    }
    else {
        Deug.print("qui commence? ordi (taper 1) humain (taper 2) ");
        if (Deug.readInt() == 1)
        partie = new Damier (l, true);
        else
        partie = new Damier (l, false);
        Damier.jeuHumainOrdi(partie);
    }
    }
}
```