

# Algorithmique & Programmation

## Cas d'étude : jeu de Nim

[yann.secq@univ-lille.fr](mailto:yann.secq@univ-lille.fr)

ABDELKADER Omar, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEQUINIOU Erwann, MARSHALL-BRETON Christopher, REKIK Yosra, SECQ Yann, SOW Younoussa, SUDHEENDRAN Megha, SU Yue

# Le jeu de Nim

- Jeu à information complète à 2 joueurs
- N allumettes disposées face aux joueurs, chacun en retire entre 1 à 3 par tour, celui prenant la dernière perd
- Comment décomposer la description de ce jeu pour faciliter sa programmation ?

# Le jeu de Nim : analyse

- Jeu à information complète à 2 joueurs
- $n$  allumettes, chacun en tire entre 1 et 3 par tour, celui prenant la dernière perd

## ● Quelles données ?

- Allumettes ? Joueurs ?
- Quels usages ? Quels types ? Variation ou pas ?

## ● Quels traitements ?

- Notion de tour de jeu
- Notion de fin de gain d'une partie

# Données : les allumettes

- **Description**

- On dispose de  $n$  allumettes au début du jeu
- On retire 1 à 3 allumettes par tour de jeu
- En fin de partie, il ne reste plus aucune allumette

- **Analyse**

- On doit représenter un nombre d'allumettes
- Ce nombre varie entre  $n$  et 0 (tous deux inclus)
- Variable de type `int` (ou `short`, ou `byte` ...)

# Données : les joueurs

- **Description**

- C'est un jeu à deux joueuses
- Pas d'information sur leur dénomination
- Nécessaire pour savoir qui doit jouer quand

- **Analyse**

- On peut représenter le nom des joueurs : deux variables de type `String`
- Cela ne peut être qu'à l'un des deux de jouer : une variable `boolean` est suffisante
- Quelles opérations sur les joueurs/joueuses ? **Changer le joueur actif !**
- Regardons comment les traitements diffèrent en fonction du choix de représentation des joueurs

# Changer de joueur

```
class JeuDeNim extends Program {  
  
    String changer(String joueurCourant, String j1, String j2) {  
        String prochain = j1; // hypothèse par défaut  
        if (equals(joueurCourant, j1)) {  
            prochain = j2;  
        }  
        return prochain;  
    }  
  
    boolean changer(boolean joueurCourant) {  
        return !joueurCourant;  
    }  
}
```

OU

- **Quelles données ?**

- Allumettes ? Joueurs ?  
**Joueur courant !**

Seulement si l'on souhaite autre chose que les noms générique « Joueur1 » et « Joueur 2 »

Et une convention du type si true alors joueur1 sinon joueur2

```
class JeuDeNim extends Program {

    void test_changer() {
        assertEquals(true,  changer(false));
        assertEquals(false, changer(true));
    }

    boolean changer(boolean joueurActuel) {
        return !joueurActuel;
    }

    void algorithm() {
        final int NB_INITIAL_ALLUMETTES = 13;
        int allumettes = NB_INITIAL_ALLUMETTES;
        boolean joueurCourant = true; // J1 par convention
        // à compléter
    }
}
```

# Le jeu de Nim : analyse

- Jeu à information complète à 2 joueurs
- $n$  allumettes, chacun en tire entre 1 et 3 par tour, celui prenant la dernière perd

## • Quelles données ?

- Allumettes ? Joueurs ?
- Quels usages ? Quels types ? Variation ou pas ?

## • Quels traitements ?

- Notion de tour de jeu
- Notion de fin de gain d'une partie

# Traitement : boucle de jeu

- **Description**

- Le joueur 1 débute et doit retirer de 1 à 3 allumettes
- On change de joueur
- Si il reste des allumettes, on recommence
- Ensuite, il faudra déterminer le gagnant

- **Analyse**

- Peut-on connaître le nombre de tours de jeu ? A minima  $13 / 3 = 4 + 1$ , a maxima  $13/1 = 13$
- On a un intervalle mais on ne connaît pas le nombre d'itérations, cela dépend des stratégies des joueurs
- Boucle à évènement : mais jusqu'à quand reste-t-on dans la boucle ?
- Tant qu'il reste des allumettes ...

# Traitements : analyse à gros grain

```
class JeuDeNim extends Program {  
  
    // Déclaration et définition des fonctions  
    // et des fonctions de test correspondantes !  
  
    void algorithm() {  
        // déclaration / initialisation des variables  
        TantQue il_y_a_des_allumettes ← allumettes > 0  
            afficher le_tas_d_allumettes  
            allumettesARetirer = demander(joueurCourant);  
            allumettes = allumettes - allumettesARetirer  
            joueurCourant = changer(joueurCourant)  
            feliciter le gagnant  
    }  
}
```

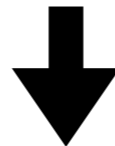
# Traitements : analyse à gros grain

```
class JeuDeNim extends Program {  
  
    // Déclaration et définition des fonctions  
    // et des fonctions de test correspondantes !  
  
    void _algorithm() {  
        // déclaration / initialisation des variables  
        TantQue (allumettes > 0)  
            afficher le_tas_d_allumettes  
            allumettesARetirer = demander(joueurCourant);  
            allumettes = allumettes - allumettesARetirer  
            joueurCourant = changer(joueurCourant)  
            feliciter le gagnant  
    }  
}
```

*fonctions à définir ...*

# Affiner les appels de fonction

```
TantQue (allumettes > 0)
  afficher le_tas_d_allumettes
  allumettesARetirer = demander(joueurCourant);
  allumettes = allumettes - allumettesARetirer
  joueurCourant = changer(joueurCourant)
feliciter le gagnant
```



```
While (allumettes > 0)
  afficher(allumettes)
  allumettesARetirer = demander(joueurCourant);
  allumettes = allumettes - allumettesARetirer
  joueurCourant = changer(joueurCourant)
feliciter(joueurCourant)
```

# Définir les signatures

```
TantQue (allumettes > 0)
  afficher(allumettes)
  allumettesARetirer = demander(joueurCourant);
  allumettes = allumettes - allumettesARetirer
  joueurCourant = changer(joueurCourant)
feliciter(joueurCourant)
```

```
void afficher(int les_allumettes)
int demander(boolean joueurActuel)
boolean changer(boolean joueurActuel)
void feliciter(boolean joueurActuel)
```

Nous pouvons maintenant écrire les corps de fonction **en**  
définissant en même temps les tests correspondants afin  
de **progresser par courtes itérations**

# Définir les corps de fonction

```
void afficher(int les_allumettes)
    for (int item=0; item < les_allumettes; item=item+1) {
        print("|");
    }
    println();
}
```

*Mais comment tester une procédure ? :(*

# Définir les corps de fonction

```
void test_affichage()  
    assertEquals("\n", affichage(0));  
    assertEquals("|||||\n", affichage(5));  
    assertEquals("|||||||||||||\n", affichage(13));  
}
```

*Fonction maintenant testable et testée :)*

```
String affichage(int lesAllumettes)  
    String resultat = "";  
    for (int item=0; item < lesAllumettes; item=item+1) {  
        resultat = resultat + "|";  
    }  
    resultat = resultat + "\n";  
    return resultat;  
}
```

← *Code correspondant au retour à la ligne !*

# Définir les corps de fonction

```
int demander(boolean joueurActuel)
    int nbAllumettes = 0;
    String joueur = "Joueur 1";
    if (!joueurActuel) {
        joueur = "Joueur 2";
    }
    do {
        print(joueur+" : nb d'allumettes ? « );
        nbAllumettes = readInt();
    } while (nbAllumettes <= 0 || nbAllumettes > 3);
    return nbAllumettes;
}
```

*Complicé de tester (automatiquement) une fonction de saisie ...*

# Définir les corps de fonction

```
void test_feliciter()  
    assertEquals("Bravo Joueur 1", feliciter(true));  
    assertEquals("Bravo Joueur 2", feliciter(false));  
}  
  
String feliciter(boolean joueurActuel)  
    int numero = 1;  
    if (joueurActuel == false) {  
        numero = 2;  
    }  
    return "Bravo Joueur " + numero;  
}
```

*Nous pouvons maintenant écrire l'ensemble du programme principal*

```

class JeuDeNim extends Program {

    void test_changer() {...}
    boolean changer(boolean joueurActuel) {...}
    void test_affichage() {...}
    String affichage(int les_allumettes) {...}
    int demander(boolean joueurActuel) {...}
    void test_feliciter() {...}
    String feliciter(boolean joueurActuel) {...}

    void _algorithm() {
        final int NB_INITIAL_ALLUMETTES = 13;
        int allumettes = NB_INITIAL_ALLUMETTES;
        boolean joueurCourant = true; // J1 par convention
        while (allumettes > 0) {
            print(affichage(allumettes));
            allumettesARetirer = demander(joueurCourant);
            allumettes = allumettes - allumettesARetirer;
            joueurCourant = changer(joueurCourant)
        }
        feliciter(joueurCourant)
    }
}

```

```

class JeuDeNim extends Program {

    boolean changer(boolean joueurActuel) {...}
    String affichage(int les_allumettes) {...}
    int demander(boolean joueurActuel) {...}
    String feliciter(boolean joueurActuel) {...}

    void _algorithm() {
        final int NB_INITIAL_ALLUMETTES = 13;
        int allumettes = NB_INITIAL_ALLUMETTES;
        boolean joueurCourant = true; // J1 par convention
        while (allumettes > 0) {
            print(affichage(allumettes));
            allumettesARetirer = demander(joueurCourant);
            allumettes = allumettes - allumettesARetirer;
            joueurCourant = changer(joueurCourant);
        }
        feliciter(joueurCourant);
    }
}

```

***Est-ce fini ?***

# Synthèse du cas d'étude

## Méthode pour analyser et proposer un algorithme pertinent

- **Identifier les données importantes**, leur type, leur nature (variable/constant)
- **Réfléchir à traitement si plusieurs types sont possibles pour représenter une donnée**
- Établir un brouillon de l'algorithme principal, mélangeant pseudo-code et code *ijava* pour avoir une vue globale et un point de départ
- **Identifier les signatures** des fonctions appelées dans `void algorithm`
- **Écrire les corps de fonctions et les fonctions de tests associées**
- **Progresser par de petites itérations** en testant au fur et à mesure

**On peut se tromper** et corriger en cours de route : **ce n'est pas grave**

# One more thing

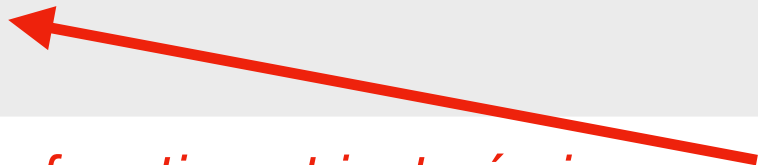
```
int demander(boolean joueurActuel)
{
    int nbAllumettes = 0;
    String joueur = "Joueur 1";
    if (!joueurActuel) {
        joueur = "Joueur 2";
    }
    do {
        print(joueur+" : nb d'allumettes ? « );
        nbAllumettes = readInt();
    } while (nbAllumettes <= 0 || nbAllumettes > 3);
    return nbAllumettes;
}
```

```
String feliciter(boolean joueurActuel)
{
    int numero = 1;
    if (joueurActuel == false) {
        numero = 2;
    }
    return "Bravo Joueur " + numero;
}
```

```
int numero(boolean jActuel) {  
    int numero = 1;  
    if (joueurActuel == false) {  
        numero = 2;  
    }  
    return numero;  
}
```

```
int demander(boolean joueurActuel)  
    int nbAllumettes = 0;  
    do {  
        print(numero(joueurActuel) + " : nb d'allumettes ?");  
        nbAllumettes = readInt();  
    } while (nbAllumettes <= 0 || nbAllumettes > 3);  
    return nbAllumettes;  
}
```

```
String feliciter(boolean joueurActuel)  
    return "Bravo Joueur " + numero(joueurActuel);  
}
```



*Voir même supprimer cette fonction et juste écrire ceci dans l'algorithme principal*  
*println("Bravo Joueur " + numero(!joueurCourant))*

