Algorithmes itératifs

Licence 1 MASS

Introduction à Java et à l'algorithmique

Sébastien Verel
verel@i3s.unice.fr
www.i3s.unice.fr/∼verel

Équipe ScoBi - Université de Nice Sophia-Antipolis

6 février 2012

Travail de la semaine passée

- Faire une recherche bibliographique sur le cours
- Installer processing (fini ?)
- Travailler à partir des objectifs du cours
- Travailler les exemples du cours
- Travailler les exercices des TP 01 et 02
- Créer des variations d'exemples et d'exercices
- Explorer les exemples de Processing

Petit exemple avec switch et le clavier

Comment marche le clavier ?

- keyPressed : vrai lorsqu'une touche est enfoncée
- key : contient la valeur de la dernière touche enfoncée (de type char)
- keyCode : donne le code de la dernière touche enfoncée, utilisée pour détecter les touches spéciales :
  prend les valeurs UP, DOWN, LEFT, ENTER, RETURN, ESC, RIGHT, etc.

Petit exemple avec switch et le clavier

```
void setup() {
}

void draw() {
  if (keyPressed)
     print(key);
}
```

cf. key

Problème : Controler la fréquence d'acquisition

Petit exemple avec switch et le clavier

```
void setup() {
  frameRate(5);
}

void draw() {
  if (keyPressed)
     print(key);
}
```

Petit exemple avec switch et le clavier (ex. choix)

```
void draw() {
  if (keyPressed) {
    switch (key) {
      case 'a' :
        println("classique et délicieuse");
        break;
      case 'b' :
        println("choix de gourmet");
        break;
      case 'c' :
        println("choix de saison");
        break;
      default :
        println("Pardon ? Je n'ai pas compris.");
    }
  }
}
```

Avec majuscules

```
void draw() {
  if (keyPressed) {
    switch (key) {
      case 'A' : case 'a' :
        println("classique et délicieuse");
        break;
      case 'B' : case 'b' :
        println("choix de gourmet");
        break;
      case 'C' : case 'c' :
        println("choix de saison");
        break;
      default :
        println("Pardon ? Je n'ai pas compris.");
    }
  }
}
```

Objectifs de la séance 3

1. Ecrire un algorithme avec une itération de type "pour"
2. Ecrire un algorithme avec des itérations imbriquées de type "pour"
3. Ecrire un algorithme avec des l'itérations de type "tant que"
4. Contrôler les événements clavier avec Processing
5. Ecrire un algorithme correct d'au moins 8 lignes

Question principale du jour :

Comment écrire de longs algorithmes sans avoir tout à réécrire ?

Plan

1. Problématique
2. Schéma itératif à nombre déterminé d'itérations
3. Schéma itératif avec condition d'arrêt
4. Itérations multiples

Exemple

Question :

Ecrire un algorithme demandant la saisie de 12 nombres au clavier et calculant la moyenne de ces 12 nombres.

Exemple

Algorithme moyenne() : réel
début
variable m : réel
lire(a)
lire(b)
lire(c)
lire(d)
lire(e)
lire(f)
lire(g) lire(h)
lire(i) lire(j)
lire(k) lire(l)
$m \leftarrow (a + b + c + d + e + f + g + h + i + j + k + l)/12$
retourner m
fin

Exemple

Algorithme moyenne() : réel
début
variable n0, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11 : réel
variable m : réel
lire(n0)
lire(n1)
lire(n2)
lire(n3)
lire(n4)
lire(n5)
lire(n6) lire(n7)
lire(n8) lire(n9)
lire(n10) lire(n11)
$m \leftarrow (n0 + n1 + n2 + n3 + n4 + n5 + n6 + n7 + n8 + n9 + n10 + n11)/12$
retourner m
fin

Exemple

Ce n'est pas tenable

- Passage à l'échelle :
  technique impraticable lorsque plus de 1000 notes.
- Evolution du code :
  il faut écrire un nouvel algorithme lorsqu'on veut modifier le nombre de notes.

Exemple

- instruction "lire" s'effectue 12 fois de suite,

$\Rightarrow$ répéter 12 fois "lire"

Il faudrait quelque chose qui ressemble à :

```
repeter 12 fois
  lire(a)
fin repeter

m  <- ????
```

Seulement, l'affectation dans la variable *m* n'est plus bien définie....

Vers une solution

- Calculer partiellement la somme au fur et à mesure que les nombres sont lus.

Ce qui devrait ressembler à quelque chose comme ceci :

```
repeter 12 fois
  lire(a)
  recalculer la somme partielle
fin repeter

moyenne  <-  somme / 12
```

Vers une solution : encore mieux

- Calculer partiellement la somme au fur et à mesure que les nombres sont lus
- Définir le nombre de notes comme paramètre

n est un paramètre défini à l'extérieur de l'agorithme :

```
repeter n fois
  lire(a)
  ajouter a à la somme partielle
fin repeter

moyenne  <-  somme / n
```

$\longrightarrow$ Plus besoin de changer l'agorithme pour changer de nombre de notes.

Une autre catastrophe

Demander une réponse Oui ou Non

```
Algorithme  SaisirReponse() : caractere
début
  variable c : caractere
  écrire("Donner votre réponse (O/N)")
  lire(c)
  si c ≠' O' ET c ≠' N' alors
    écrire("Erreur, veuillez taper O ou N")
    lire(c)
    si c ≠' O' ET c ≠' N' alors
      écrire("Erreur, veuillez taper O ou N")
      lire(c)
      si c ≠' O' ET c ≠' N' alors
        écrire("Erreur, veuillez taper O ou N")
        lire(c)
        si c ≠' O' ET c ≠' N' alors
          écrire("Erreur, veuillez taper O ou N")
          lire(c)
          si c ≠' O' ET c ≠' N' alors
            écrire("Erreur, veuillez taper O ou N")
            lire(c)
          fin si
        fin si
      fin si
    fin si
  fin si
  retourner c
fin
```

Une autre catastrophe

Demander une réponse Oui ou Non

Ce n'est pas tenable

- Correction :
  Il existe toujours des situations où le nombre de tests est insuffisant
- Lisibilité :
  l'algorithme n'est pas facile à lire (surtout s'il y a 12 tests !)

Une autre catastrophe

Vers une solution

- instructions "écrire/lire" se répètent un certain de nombre de fois

⇒ répéter "écrire/lire" tant que la condition reste valide.

```
tant que c != 'O' ET c != 'N'
  écrire ...
  lire(c)
fin tant que
```

Après cette itération, on est sûr que la variable *c* contient un caractère qui est 'O' ou 'N'

Autres problèmes de la vie

- "tant que je pourrais, je mange des crèpes."
- "Je vais ranger jusqu'à ce qu'elle vienne."
- "Continue de compter tant que je ne suis pas caché."
- "Ecouter les 4 premiers morceaux."
- "Bougez tant que la température est inférieure à 16 degrés !"
- "Dormir jusqu'à ce que le réveil sonne"

$\longrightarrow$ Phrases exprimants l'**itération** d'une suite d'opérations dans la mesure où une condition n'est pas vérifiée

Autre exemple

Recherche d'un mot dans un dictionnaire

Algorithme recherchelter(*cible* : mot) : liste de mots
début
  Lire premier mot du dictionnaire
  tant que mot lu n'est pas le mot cible **faire**
    Lire le mot suivant
  fin tant que
  liste de mots : définition du mot lu
fin

- Itération de la commande "lire"
- Arrêt de l'itération lorsque le "mot lu est le mot cible"

Quand utiliser un schéma itératif ?

La question qu'il faut se poser

- Est-ce qu'il existe une suite d'opérations similaires (modulo certains paramètres) qui se répète ?

Si oui, deux types de schéma itératif :

- schéma itératif à nombre déterminé d'itérations
- schéma itératif à condition d'arrêt

Quand utiliser ce schéma itératif ?

Les questions qu'il faut se poser

- Est-ce qu'il existe une suite d'opérations similaires (modulo certains paramètres) qui se répète ?
- Est-ce que le nombre d'itérations est déterminé "à l'avance", *i.e.* déterminé avant le début du schéma itératif ?

Si oui aux deux questions :
$\longrightarrow$ utiliser un schéma itératif à nombre déterminé d'itérations
$\longrightarrow$ appelé aussi souvent "boucle pour".

Nombre déterminé d'itérations avant le schéma itératif

pour *var* **de** *a* **à** *b* **faire**
  morceau d'algorithme
fin pour

- *a* et *b* : nombres entiers correspondants aux bornes inférieure et supérieure du schéma.
- *var* : variable de type entier qui décrit toutes les valeurs entre *a* et *b*

Fonctionnement :

- au début de la 1ère itération, *var* a pour valeur *a* (de type entier)
- à la fin de chaque itération, la variable a est augmentée de 1
- une nouvelle itération est effectuée tant que la valeur de *a* est inférieure ou égale à la borne supérieure *b*

Nombre d'itérations déterminé avant le schéma itératif

pour *var* **de** *a* **à** *b* **faire**
  morceau d'algorithme
fin pour

Combien d'itérations effectuées ?

Le "morceau d'algorithme" (les commandes entre "POUR" et "FIN POUR") est répété pour les valeurs entières entre a à b comprises.

- si $a \leq b$, alors $b - a + 1$ itérations sont effectuées,
- si $a > b$, alors aucune itération n'est effectuée.

Exemple très simple

Dessiner *n* carrés sur la diagonale

Algorithme carres(n : entier) : rien
début
variable *i* : entier
  pour *i* **de** 1 **à** *n* **faire**
    rectangle(10 * i, 10 * i, 10, 10)
  fin pour
fin

Remarque : le nombre d'itérations est déterminé avant le début de l'itération "POUR"

Que produisent les algorithmes suivants ?

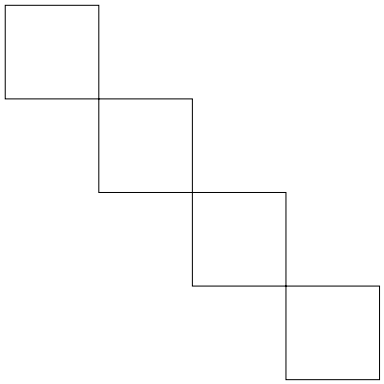
carres(4)
carres(0)

Exemple très simple

Dessiner n carrés sur la diagonale

Que produisent les algorithmes suivants ?

carres(4)



et attention

carres(0)

Retour sur le calcul de la moyenne

Algorithme moyenne() : réel
début
variable a, m : réel
n, i : entier
$n \leftarrow 12$
$m \leftarrow 0$
pour i **de** 1 **à** n **faire**
écrire("taper la note ", i)
lire(a)
$m \leftarrow m + a$
fin pour
$m \leftarrow m/n$
retourner m
fin

Execution de l'algorithme par : moyenne()

Retour sur le calcul de la moyenne

Le nombre de note peut être donner en paramètre de l'algorithme :

Algorithme moyenne(n : entier) : réel
début
variable a, m : réel
i : entier
$m \leftarrow 0$
pour i **de** 1 **à** n **faire**
écrire("taper la note ", i)
lire(a)
$m \leftarrow m + a$
fin pour
$m \leftarrow m/n$
retourner m
fin

Calcul de la moyenne de 12 notes : moyenne(12)

Utilisation d'un accumulateur

- calcul de la moyenne : mémorisation d'un calcul partiel
- nombreux d'algorithmes : mémorisation un résultat partiel

accumulateur

variable qui permet de mémoriser un résultat partiel.

ATTENTION !

Il faut toujours affecter une valeur initiale (initialiser) à l'accumulateur avant le commencement du calcul.

Cas limites

Algorithme carres0( ) : rien
début
variable *i* : entier
  pour *i* **de** 1 **à** 1 **faire**
    rectangle(10 * i, 10 * i, 10, 10)
  fin pour
fin

- Une seule itération lorsque les bornes sont identiques.

Cas limites

Algorithme carresNeg() : rien
début
  variable *i* : entier
  pour *i* **de** 5 **à** 2 **faire**
    rectangle(10 * i, 10 * i, 10, 10)
  fin pour
fin

- Aucune itération lorsque la seconde borne est strictement inférieure à la première

En Java : le plus classique

```
for(int var = a ; var <= b ; var = var + 1) {
   "morceau de programme"
}
```

ou encore :

```
int var ;
for(var = a ; var <= b ; var = var + 1) {
   "morceau de programme"
}
```

ou encore pour les plus feinéants :

```
for(int var = a ; var <= b ; var++) {
   "morceau de programme"
}
```

En Java : encore plus classique

Répétition *n* fois :

```
for(int i = 0 ; i < n ; i++) {
   "morceau de programme"
}
```

Attention à la synthaxe

Avez-vous bien noté où sont placés les ; et les { } ?
Comment sont disposés les ( ) ?
Comment sont séparés les éléments ?

En Java : plus général

```
for(int var = a ; var <= b ; variation de var) {
   "morceau de programme"
}
```

La *variation de var* est une affectation qui change la valeur de *var*

Exemple :

```
for(int x = 1; x < width - 10 ; x = x + 10) {
  rect(x, x, 10, 10);
}
```

Exemple multiplicatif :

```
for(int x = 1; x < width - 10 ; x = x * 2) {
  rect(x, x, 10, 10);
}
```

Exemples en Java

- carres
- carresClassiques
- carresSeries

Quand utiliser ce schéma itératif ?

Les questions qu'il faut se poser

- Est-ce qu'il existe une suite d'opérations (modulo certains paramètres) qui se répéte ?
- Est-ce que le nombre d'itérations est déterminé à l'avance, *i.e.* déterminé au moins avant le début du schéma itératif ?
- Est-ce que l'arrêt de l'itération s'exprime sous forme d'un test ?

Si oui, non, oui :
$\longrightarrow$ utiliser un schéma itératif avec condition d'arrêt
$\longrightarrow$ appelé aussi souvent boucle "tant que".

Itération conditionnnelle

tant que b **faire**
morceau d'algorithme
fin tant que

- *b* est une expression booléenne dont la valeur est soit *Vrai* soit *Faux* (voir cours précédent)

Le "morceau d'algorithme" est répété tant que l'expression booléenne est VRAI :

- si *b* est VRAI, alors "morceau d'algorithme" est exécuté
- si *b* est FAUX, alors "morceau d'algorithme" n'est pas exécuté, l'algorithme continue après "fin tant que"

Attention à la synthaxe

Vérifier ...

Itération conditionnnelle (exécution au moins une fois)

faire
morceau d'algorithme
tant que (b)

- *b* est une expression booléenne dont la valeur est soit *Vrai* soit *Faux* (voir cours précédent)

Le "morceau d'algorithme" est répété tant que l'expression booléenne est VRAI :

- si *b* est VRAI, alors "morceau d'algorithme" est exécuté
- si *b* est FAUX, alors "morceau d'algorithme" n'est pas exécuté, l'algorithme continue après "fin tant que"

Exemple

Petit jeu où l'on doit deviner un nombre mystère

Algorithme deviner(n : entier) : rien
début
variable a : entier
  $a \leftarrow n - 1$
  tant que $a \neq n$ **faire**
    écrire("Proposer un nombre ")
    lire(a)
  fin tant que
  écrire("vous avez trouvé")
fin

Retour sur l'algorithme demandant une réponse Oui ou Non

Algorithme saisirDemande() : caractere
début
variable c : caractere
  écrire("Donner votre réponse (O/N)")
  lire(c)
  tant que $c \neq' O'$ ET $c \neq' N'$ **faire**
    écrire("Erreur, veuillez taper O ou N")
    lire(c)
  fin tant que
  retourner c
fin

Retour sur l'algorithme demandant une réponse Oui ou Non

Algorithme saisirDemande() : caractere
début
variable c : caractere
  faire
  écrire("Veuillez taper O ou N")
  lire(c)
  tant que $(c \neq' O' \text{ ET } c \neq' N')$
  retourner c
fin

Pourquoi connecteur ET ?

$c \neq' O'$	$c \neq' N'$	Faire une itération
Faux	Vrai	
Vrai	Vrai	
Faux	Faux	
Vrai	Faux	

Pourquoi connecteur ET ?

$c \neq' O'$	$c \neq' N'$	Faire une itération
Faux	Vrai	Non
Vrai	Vrai	Oui
Faux	Faux	Non
Vrai	Faux	Non

- Correspond au connecteur logique ET

Condition avec connecteurs logiques

Problème : "Continue de compter tant que je ne suis pas caché ou alors tant que tu ne dépasse pas 100"

```
Algorithme  cacher() : rien
début
variable  a : réel
          n : entier
          reponse : caractère
          cache : booléen
  n ← 0
  cache ← FAUX

  tant que n < 100 ?? NON cache faire
    n ← n + 1
    écrire(n)

    écrire("Es-tu caché (O/N) ?");
    lire(reponse)
    si reponse = O alors
      cache ← VRAI
    fin si
  fin tant que
  écrire("J'arrive !")
fin
```

Condition avec connecteurs logiques

cache	$n < 100$	Non cache	Faire une itération
Faux	Vrai		
Vrai	Vrai		
Faux	Faux		
Vrai	Faux		

Condition avec connecteurs logiques

cache	$n < 100$	Non cache	Faire une itération
Faux	Vrai	Vrai	Oui
Vrai	Vrai	Faux	Non
Faux	Faux	Vrai	Non
Vrai	Faux	Faux	Non

- Correspond au connecteur logique ET

Condition avec connecteurs logiques

```
Algorithme  cacher() : rien
début
variable  a : réel
          n : entier
          reponse : caractère
          cache : booléen
  n ← 0
  cache ← FAUX

  tant que n < 100 et NON cache faire
    n ← n + 1
    écrire(n)

    écrire("Es-tu caché (O/N) ?") ;
    lire(reponse)
    si reponse = O alors
      cache ← VRAI
    fin si
  fin tant que
  écrire("J'arrive !")
fin
```

Equivalence entre une boucle de type "pour" et une boucle de type "tant que"''

Les schéma itératifs suivants sont équivalents :

variable i : entier

pour i **de** a **à** b **faire**
  morceau d'algorithme
fin pour

variable i : entier

$i \leftarrow a$
tant que $i \leq b$ **faire**
  morceau d'algorithme
  $i \leftarrow i + 1$
fin tant que

ATTENTION : La version "tant que" est souvent source d'erreurs : oublie initialisation, incrémentation.

Une erreur comprise

Ne pas modifier la valeur de la variable d'itération dans une boucle "pour" :

variable i : entier
  pour *i* **de** *a* **à** *b* **faire**
    morceau d'algorithme

    $i \leftarrow i + 2$
  fin pour

Revient à incrémenter la variable *i* de 3 à chaque itération

Boucle while

en Java

```
while (b) {
    morceau de programme
}
```

b est une expression booléenne qui peut être :

- true : dans ce cas le morceau de programme s'exécute, puis le test est de nouveau effectué
- false : dans ce cas le morceau de programme n'est pas exécuté

Exemple en java

```
size(200, 300);
background(255);

int i = 0;


while (i < 256) {
  stroke(i);
  line(0, i, width, i);
  i++;
}
```

Donner l'équivalence avec l'autre type de boucle

Exemple en java

```
size(512, 200);
background(255);

int i = 0;

int a = 0;
int b = width;

while (a < b) {
  stroke(i);
  line(a, i, b, i);

  a = a + 1;
  b = b - 1;
  i++;
}
```

boucles imbriquées

Boucles de type "pour" imbriquées

variable i, j : entier

pour i **de** 1 **à** n **faire**
  pour j **de** 1 **à** p **faire**
    algorithme A
  fin pour
fin pour

- si p ne dépend pas de la valeur de i alors l'algorithme A s'exécute $n \times p$.
- si p dépend de i,
  alors l'algorithme A s'exécute
  $\sum_{i=1}^{n} p(i)$

Tests multiples

Attention : succession de 2 boucles $\neq$ boucles imbriquées

variable i, j : entier

pour i **de** 1 **à** n **faire**
algorithme A
fin pour

pour j **de** 1 **à** p **faire**
algorithme B
fin pour

- l'algorithme A s'exécute n fois

Puis,

- l'algorithme B s'exécute p fois

Que vaut x à la fin de chacun des algorithmes suivants ?

Algo1 :

$x \leftarrow 0$
pour i **de** 1 **à** n **faire**
  pour j **de** 1 **à** p **faire**
    $x \leftarrow x + 1$
  fin pour
fin pour

Algo2 :

$x \leftarrow 0$
pour i **de** 1 **à** n **faire**
  pour j **de** 1 **à** i **faire**
    $x \leftarrow x + 1$
  fin pour
fin pour

Algo3 :

$x \leftarrow 0$
pour i **de** 1 **à** n **faire**
  $x \leftarrow x + 1$
fin pour

pour j **de** 1 **à** p **faire**
  $x \leftarrow x + 1$
fin pour

Boucles imbriquées

Exercice

Ecrire un programme qui donne la table de multiplication

Objectifs de la séance 3

1. Ecrire un algorithme avec une itération de type "pour"
2. Ecrire un algorithme avec des itérations imbriqées de type "pour"
3. Ecrire un algorithme avec des l'itérations de type "tant que"
4. Contrôler les événements clavier avec Processing
5. Ecrire un algorithme correct d'au moins 8 lignes

Question principale du jour :

Comment écrire de longs algorithmes sans avoir tout à réécrire ?

Travail de la semaine

- Faire une recherche bibliographique sur le cours
- Travailler à partir des objectifs du cours
- Travailler les exemples du cours, les exercices des TPs
- Créer des variations d'exemples et d'exercices
- Explorer les exemples de Processing
- Si besoin me contacter !