Gobelins
2011

Aymeric LAMBOLEY

Récursivité
Structure de données
Optimisation AS3

Introduction

Pour faire chauffer les esprits :

```
var a:uint, b:uint;

a = 5;
b = a++;

trace(a + "-" + b);
```

```
var i:uint = 10;

while (i-- >= 0) {
    trace(i);
}
```

La priorité des opérateurs.

```
var a:uint, b:uint;

a = 5;

b = calcul(a++);

function calcul(t:uint):uint { return t; }

trace(a + "-" + b);
```

```
var i:uint = 10;

while (--i >= 0) {
    trace(i);
}
```

Introduction

```
var a:uint = 0;

calcul();

function calcul():void {
        a = 5;
}

trace(a);
```

```
<?php
$a = 0;

calcul();

function calcul() {
        $a = 5;
}

echo $a;
?>
```

Introduction

2 solutions :

```php
<?php
$a = 0;

$a = calcul();

function calcul() {

	$b = 5;
	return $b;
}

echo $a;
?>
```

```php
<?php
$a = 0;

calcul(& $a);

function calcul($b) {

	$b = 5;
}

echo $a;
?>
```

```flash
var a:uint = 0;

calcul(a);

function calcul(b:uint) {
	b = 5;
}

trace(a);
```

La notion de référence : valeur qui permet l'accès en lecture et/ou écriture à une donnée située soit en mémoire principale soit ailleurs. Une référence n'est pas la donnée elle-même mais seulement une information de localisation. En fonction des langages, les références sont utilisées de façon explicite ou implicite.

Introduction

L'échange entre 2 variables.

```
var a:uint = 5;
var b:uint = 4;

calcul();

function calcul():void {

        var c:uint = a;
        a = b;
        b = c;
}
trace(a + " - " + b);
```

```
<?php
$a = 5;
$b = 4;

calcul(& $a,  & $b);

function calcul($a, $b) {

        $c = $a;
        $a = $b;
        $b = $c;
}

echo $a . " - " . $b;
?>
```

Un return des valeurs dans un tableau est une solution plus courante.

Sommaire

- La récursivité
- Les arbres binaires
- Les listes chaînées
- Optimisation en AS3

La récursivité

Définition : une fonction ou plus généralement un algorithme qui contient un appel à elle-même est dite <u>récursive</u>.
Attention, la programmation récursive est beaucoup plus gourmande en ressource que la programmation itérative.

Exemple : le factoriel.
Mémo : 4! = 4 * 3 * 2 * 1

De manière itérative :

```
var factoriel:uint = 5;
var resultat:uint = 1;

for (var i:uint = 1; i < factoriel + 1; i++) {
      resultat = resultat * i;
}

trace(resultat);
```

De manière récursive :

```
var factoriel:uint = 5;
var resultat:uint = 0;
resultat = calcul(factoriel);

function calcul(factoriel:uint):uint {
    if (factoriel == 1) return 1;
    else return factoriel*calcul(factoriel-1);
}

trace(resultat);
```

La récursivité

Exemple de cas d'utilisation :

La tour d'Hanoï : http://www.pracht.org/dokuwiki/cours2007/poo2/tp/tp6

Création d'un arbre : http://www.shinedraw.com/flash-vs-silverlight-gallery/

La récursivité est utilisé dans des algorithmes d'Intelligence Artificielle tel le MinMax(ex: jeux de dame) et son amélioration l'alpha-beta ainsi que le A* (pathfinding).

Comment construire un algorithme récursif ?

Etape 1 : Identifier les bons paramètres

```
function calcul(factoriel:uint):uint
```

Etape 2 : Le cas général

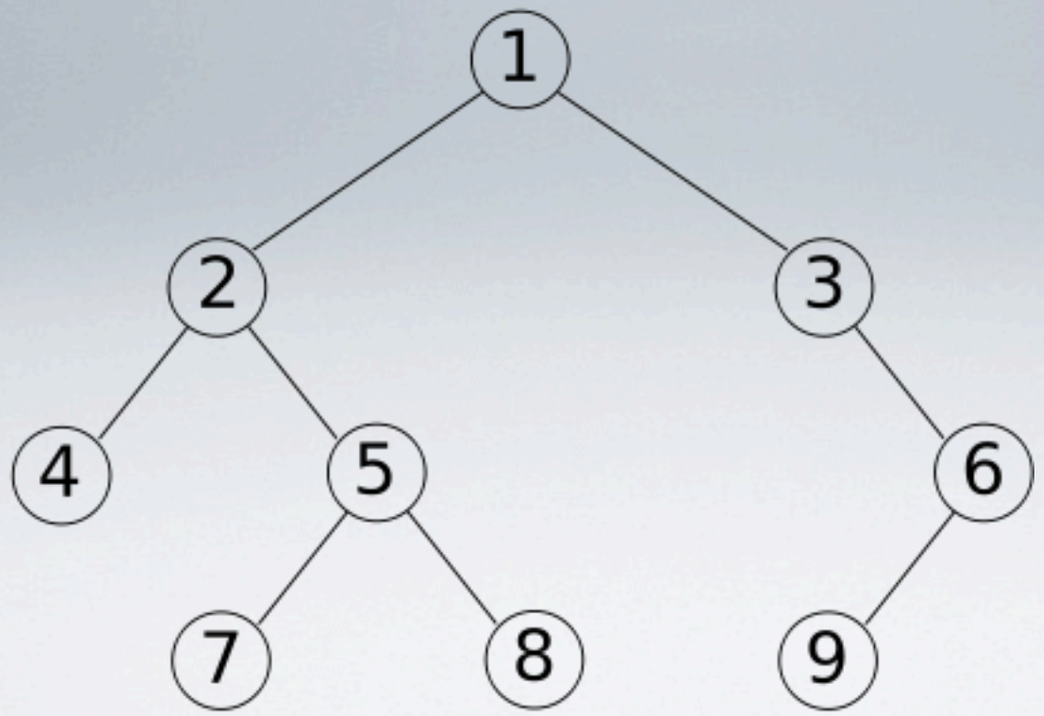
```
return factoriel*calcul(factoriel-1)
```

Etape 3 : L'exception

```
if (factoriel == 1) return 1;
```

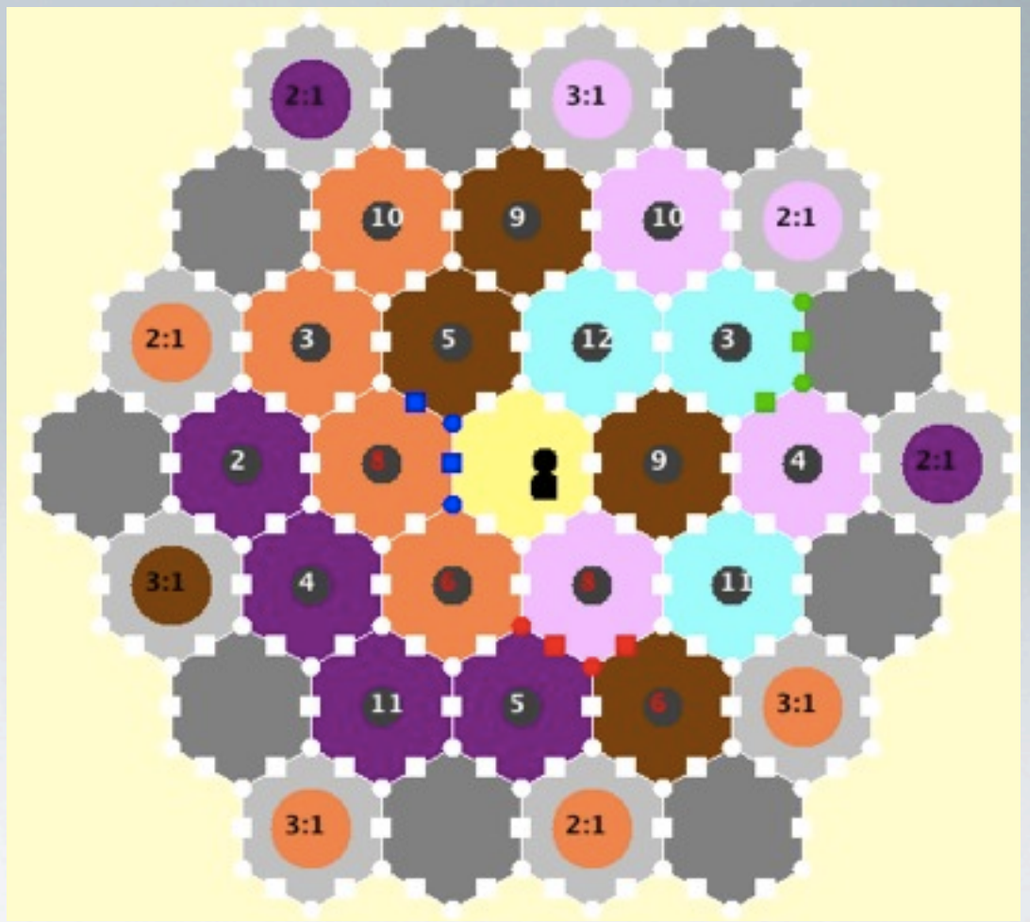
Arbre binaire

Définition : structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément est appelé nœud, le nœud initial étant appelé *racine*. Dans un arbre binaire, chaque élément possède au plus deux éléments fils au niveau inférieur, habituellement appelés *gauche* et *droit*. Du point de vue de ces éléments fils, l'élément dont ils sont issus au niveau supérieur est appelé *père*. Un nœud n'ayant aucun fils est appelé *feuille*.



Arbre binaire

Cas d'utilisation concret :
Recherche de chemin le plus long dans une carte avec des hexagones. Création d'un arbre binaire



Arbre binaire

Exemples de fonctions :

```
public class Node {
    public var $data : Number;
    public var $left : Node;
    public var $right : Node;

    public function Node(newData:Number)
    {
        $left = null;
        $right = null;
        $data = newData;
      }
}
```

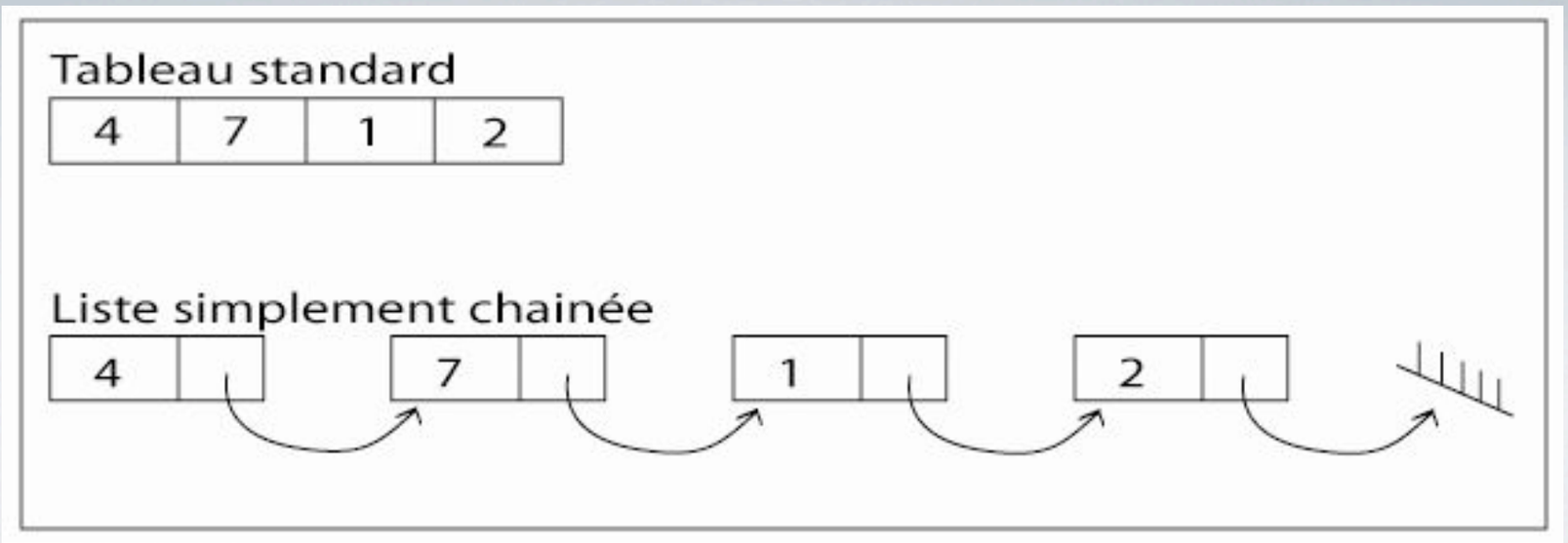
```
private function insert(node:Node,
data:Number):Node {
    if (node==null) {
        node = new Node(data);
    } else {
            if (data<=node.$data) node.$left = insert
            (node.$left, data);
            else node.$right = insertRecurs(node.$right,
            data);
    }
    return node;
}

private function size(node:Node):Number {
    if (node==null) return(0);
    else return (size(node.$left)+1+size(node.$right));
}
```

Liste chaînée

Définition : structure de données représentant une collection ordonnée et de taille arbitraire d'éléments de même type.

L'accès aux éléments d'une liste se fait de manière séquentielle : chaque élément permet l'accès au suivant (contrairement au cas du tableau dans lequel l'accès se fait de manière absolue, par adressage direct de chaque cellule dudit tableau).

Un élément a donc une référence vers son suivant.



Liste chaînée

Les listes chaînées permettent de bien meilleur performance que les tableaux. Toutefois, elles nécessitent de manipuler un grand nombre de données pour que leurs performances soient mises en avant.

Exemple : les moteurs de particules et la 3D.
http://www.minuit4.fr/2009/05/flash/passer-par-des-linkedlist-pour-gagner-en-performances/

Liste chaînée

Implémentation en AS3 :

```
public class NumberBox {
    public var value:uint = 0;
    public var next:NumberBox = null;
}
```

Afficher les valeurs de la liste

```
const list:NumberBox = create(1000);
var current:NumberBox = list;

while(current) {
    trace(current.value);
    current = current.next;
}
```

Création d'une liste chaînée avec numValues éléments

```
function create(numValues:int):NumberBox {

    if(1 >numValues) return null;

    const start:NumberBox = new NumberBox();
    var current:NumberBox = start;

    while(--numValues != 0)
        current = current.next = new NumberBox();

    return start; //On retourne le 1er élément de la liste
}
```

Optimisation en AS3

Différentes optimisations en AS3, attention à la lisibilité du code !

Le hitTestObject est une opération coûteuse en ressource, si un élément du test a toujours une position statique et une forme rectangulaire, envisagez de vérifier la collision avec les coordonnées x et y. Exemple : le casse-brique.

Lors d'une boucle for ou while (plus rapide), si la taille d'un tableau est utilisé pour déterminer une limite d'itération, il est préférable de créer une variable avec la taille du tableau au préalable et de l'utiliser. Cela évite de recalculer sa taille à chaque itération.

```
for (var i:uint = 0; i < tab.length; i++) {
    trace(i);
}
```

```
var tailleTab:uint = tab.length;
for (var i:uint = 0; i < tailleTab; i++) {
    trace(i);
}
```

Stocker les propriétés getter en tant que variable local si elles sont utilisées plusieurs fois dans une méthode :

```
var n:Graphics = sprite.graphics;
n.clear();    n.beginFill(0x000000);    n.drawRect(0,0,10,10);    n.endFill();
```

Optimisation en AS3

La déclaration de variables sur une seule ligne est plus rapide : var a:uint = 5, b:uint = 6.

La division est beaucoup plus lente que la multiplication.
var a:uint = 5; a = 5 / 5; a = 5 * 0.2;

Les opérateurs de bit à bit sont les plus rapides
var val1:uint = 4 * 2; var val2:uint = 4 * 4; var val3:uint = 4 / 2; var val4:uint = 4 / 4;
var val1:uint = 4 << 1; var val2:uint = 4 << 2; var val3:uint = 4 >> 1; var val4:uint = 4 >> 2;
x = int(1.232); x = 1.232 >> 0;

Pensez à enlever vos objets de la liste d'affichage et les mettre à nul. Supprimez les écouteurs non utilisés, arrêtez les Timer, déchargez les swf chargés, fermez les connexions ouvertes et arrêtez les sons et vidéos.

L'opérateur new est également très gourmand. Ne l'utilisez pas si ce n'est pas nécessaire :
var tab:Array = [];

Optimisation en AS3

Utiliser l'opérateur XOR pour l'échange de variable permet d'éviter de créer une variable temporaire locale.

```
var t:int = a;
a = b;
b = t;
```

```
a ^= b;
b ^= a;
a ^= b;
```

Les fonctions mathématiques d'Adobe sont relativement lentes. Oubliez les Math.abs(), Math.ceil () et Math.floor().

```
i = x < 0 ? -x : x;      i = 5.123 << 0;      i = (5.123 << 0) + 1;
```

Utilisez en priorité des useWeakReference = true pour vos écouteurs, ceci permet le nettoyage de votre écouteur par le Garbage Collector.

Evitez tant que possible la propagation d'un événement vers ses parents : bubbles = false;

Conclusion

Il existe de nombreuses méthodes pour optimiser son code. Toutefois il est très important de suivre l'évolution du langage et de mettre à jour ses connaissances. A chaque mise à jour majeure, les performances varient.

Il est très important d'optimiser son code notamment si le travail porte sur un framework qui sera réutilisé dans diverses applications.

Attention à garder un code lisible par une personne novice.

Pour les personnes souhaitant approfondir davantage, regardez l'Object Pooling et les Tables de hachage.

Liens

http://www.gskinner.com/talks/resource-management/

http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/

http://www.danielhai.com/blog/?p=55

http://lab.polygonal.de/2007/05/10/bitwise-gems-fast-integer-math/

http://www.stephencalenderblog.com/?p=109

http://wiki.joa-ebert.com/index.php/Main_Page