4 La programmation de squelettes

Introduction

Un squelette est une structure hiérarchique qui relie, avec un lien de parenté, différents symboles de la scène ou différentes parties d'une même forme graphique continue. En déployant l'ossature du squelette, les symboles ou les parties de la forme graphique suivent la progression du mouvement de la structure qui le compose.

Pour en programmer l'animation, nous utilisons la classe `ik` (*inverse kinematic* en anglais, pour cinématique inverse). Cette classe, à ce jour, ne permet pas de réaliser directement un squelette en programmation, mais uniquement, de l'animer. Les exemples proposés possèdent donc déjà un squelette, mais nous aborderons également la manière de construire cet objet manuellement pour la programmation.

Une structure articulée autour d'un squelette permet, entre autre, de reconstituer l'ossature d'un personnage et de l'animer. Elle permet aussi de distribuer les contenus d'un site sous la forme de dispositifs hiérarchisés, comme un menu ou une arborescence. Mais, si cette armature est associée à une forme graphique, cela nous donne aussi accès à des interpolations de forme accessibles en programmation.

Dans ce chapitre, nous réalisons dans un premier temps une animation mécanique d'un droïde dont les mouvements réagissent à la position du pointeur. Nous abordons ensuite l'animation de squelettes de formes organiques avec des formes graphiques vectorielles, pour animer, par exemple, le mouvement d'un végétal selon la position d'un objet animé (une abeille). Nous présentons ensuite comment gérer une animation programmée en mode interactif, afin que l'utilisateur puisse lui-même modifier le positionnement du squelette en déplaçant chaque segment de la structure manuellement.

Les squelettes, lorsqu'ils sont déployés dans des SWF imbriqués, ne s'exécutent plus. Nous présentons donc aussi une solution pour l'importation de squelettes en mode Exécution, à l'intérieur de documents Flash imbriqués.

Programmer un mouvement mécanique

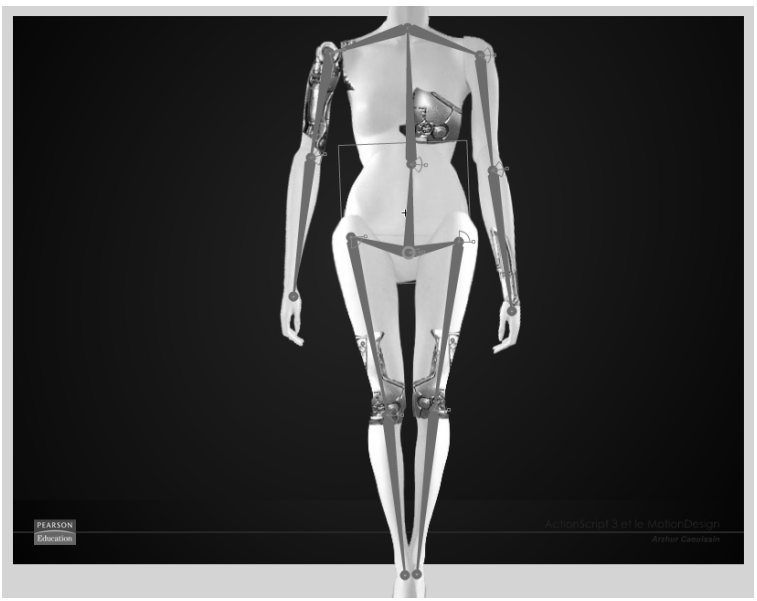
L'animation de squelettes à partir de symboles permet de mettre en mouvement des structures mécaniques qui font s'articuler des objets entre eux, et où aucun objet ne se mélange jamais aux autres. Ce type de structure à base de symboles convient à l'animation de robots, de cycles de marche mécaniques, de grues ou de systèmes de navigation, entre autres.

Dans cet exemple, un squelette est déjà en place dans le scénario. Il représente le corps d'un droïde et se compose de 15 symboles chacun relié à l'autre par un segment, `IKBone` (voir Figure 4.1). Chaque segment possède deux extrémités de jointure ou liaisons (`IKJoint`) placées respectivement en tête (`headJoint`) et en queue (`tailJoint`). C'est à partir de ces

axes de rotation et de placement que nous déterminons les mouvements à travers des interpolations cinématiques (`IKMover`). La spécificité de ces interpolations est de répercuter sur chaque symbole faisant partie de la hiérarchie, de nouvelles valeurs de positionnement, selon les contraintes de mouvement définies dans l'ossature lors de sa construction. L'ensemble du squelette est, quant à lui, inclus dans une armature générale (`IKArmature`) dont on peut déterminer le mode d'affichage avec une méthode (`IKManager`) et l'afficher, soit pour l'animation, soit pour l'exécution.

Heure de création et exécution. *La terminologie des modes d'affichage des squelettes peut surprendre. Nous observons en effet, dans l'Inspecteur de propriétés, les deux options* `Heure de création` *et* `Exécution` *littéralement traduites des termes anglais* *AuthorTime* *et* *RunTime*. *Comprenez, en ces termes, Gestion de l'animation dans l'interface auteur (*AuthorTime, *ou Heure de création) d'une part et Gestion de l'animation à la publication (*RunTime, *ou Exécution) d'autre part. Nous devrions ainsi plutôt parler de l'option "Animation" pour le premier et d'"Exécution" pour le second.*

Figure 4.1
Aperçu du document.

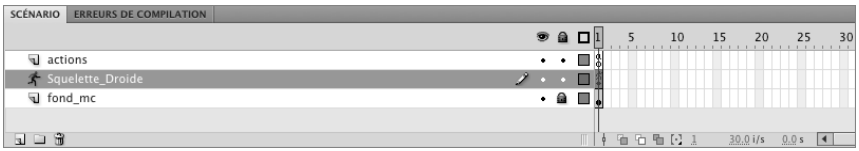


Exemples > ch4_ProgrammationDeSquelettes_1.fla

Le document que nous utilisons présente la structure suivante : dans la scène, au-dessus du calque `fond_mc`, apparaît un calque nommé `squelette_Droide` (voir Figure 4.2). Chaque symbole possède un nom d'occurrence qui définit clairement la partie du corps à laquelle il se rattache et le désigne en tant qu'objet structurel d'un squelette (`ikNode_bassin`, `ikNode_torax`, `ikNode_cou`, `ikNode_brasD`, `ikNode_avantBrasD`, `ikNode_mainD`, etc.). De même, chaque segment qui relie ces symboles entre eux possède son propre nom d'occurrence qui identifie la partie de l'ossature à laquelle il est rattaché et le désigne en tant qu'objet structurel de liaison (`ikBone_colonneBas`, `ikBone_colonneHaut`, `ikBone_epauleD`,

`ikBone_humerusD`, `ikBone_cubitusD`, etc.). À travers ces identifiants, nous distinguons bien les symboles (*Nodes*) de leurs jonctions (*Bones*). Le squelette, lui, possède également son propre nom d'occurrence `Squelette_Droide`.

Figure 4.2
Aperçu du scénario de la scène principale.



Dans le calque nommé `actions`, nous pouvons lire le code suivant :

```
//--------------------------- initialisation
import fl.ik.*;

//--------------------------- actions
// définition du squelette

IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Droide");
➥ squelette.registerElements(stage);

var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0);
var segment_colonneHaut:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0);
var segment_epauleD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(0);
var segment_humerusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(0).getChildAt(0);
var segment_cubitusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(0).getChildAt(0).getChildAt(0);
var segment_epauleG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(1);
var segment_humerusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(1).getChildAt(0);
var segment_cubitusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(1).getChildAt(0).getChildAt(0);
var segment_illiaqueG:IKJoint=squelette.rootJoint.getChildAt(1);
var segment_femurG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0);
var segment_tibiaG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0).
➥ getChildAt(0);
var segment_illiaqueD:IKJoint=squelette.rootJoint.getChildAt(2);
var segment_femurD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0);
var segment_tibiaD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0).
➥ getChildAt(0);

trace(squelette.rootJoint.getChildAt(0).getChildAt(0).name)

// animation du squelette

var mouvement1:IKMover= new
➥ IKMover(segment_humerusD,segment_humerusD.position);
```

```
stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
   var arrivee:Point=new Point(mouseX,mouseY);
   mouvement1.moveTo(arrivee);
}
```

Le langage

Construire un squelette pour la programmation

Avant de programmer l'animation d'un squelette, il convient de s'assurer qu'il est préalablement bien structuré. Tout d'abord, nous utilisons la classe ik qui permet de déplacer les têtes de chaque segment. Il faut donc bien comprendre où se positionnent ces têtes et comment les placer judicieusement pour garantir la cohérence de l'animation. Pour construire une armature pour la programmation, procédez comme suit :

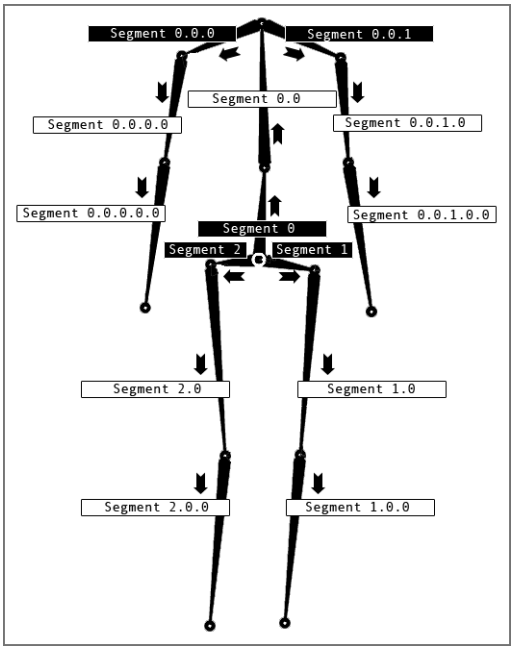
1. Placez un certain nombre de symboles de type MovieClip sur un même calque.
2. Repositionnez éventuellement les centres de transformation de chaque symbole (grâce au petit rond blanc que l'on peut déplacer avec l'outil de transformation libre). C'est en effet sur ces centres que les segments seront magnétisés et c'est leur emplacement qui détermine la cohérence de l'animation. Pour visualiser et déplacer un centre de transformation, utilisez l'outil de transformation ou activez le raccourci Q, puis glisser-déposez le petit cercle blanc à l'emplacement souhaité. Puis, réactivez l'outil de sélection en appuyant sur la touche V. Pour faciliter le positionnement des centres, pensez à désactiver au besoin les options d'accrochage du menu Affichage.
3. Si les symboles qui composent votre sujet à animer reposent sur des images bitmap (PSD, PNG, JPEG, Gif), activez le lissage des bitmaps pour éviter l'apparition du crénelage lorsque les symboles pivoteront (voir Chapitre 11).
4. Placez les segments sur les symboles. Dans la barre d'outils, utilisez l'outil Segment, en forme d'os, ou le raccourci X. Puis, cliquez sur le centre de transformation du symbole père, glissez et relâchez le bouton de la souris sur le centre de transformation du symbole enfant.
5. Reproduisez la manipulation autant de fois que vous possédez de symboles à relier entre eux, en partant toujours de l'extrémité située en queue d'un segment, puis poursuivez la hiérarchie vers les autres symboles.
6. Ajoutez un nom d'occurrence pour chaque objet créé. Pour cela, cliquez sur les segments un à un, sur les têtes de segments une à une, et sur le calque du squelette lui-même, pour leur attribuer à tous des noms d'occurrence distinctifs. Flash en propose par défaut (`IkNode_1` pour les têtes de segment, `IkBoneName1` pour les segments et `Squelette_1` pour le calque du squelette). Vous pouvez conserver ces noms ou les adapter à votre perception.
7. Activez ou non les contraintes de mouvement et de rotation. Pour ce faire, sélectionnez chaque segment avec l'outil de sélection (flèche noire ou raccourci V), et depuis l'Inspecteur de propriétés, activez les différentes options de liaison (`rotation`, `translationX` et `translationY`). Attention, les valeurs indiquent une rotation relative. Pour faciliter les manipulations, orientez chaque symbole sur l'axe médian de l'espace de rotation que vous souhaitez lui réserver. En activant l'option de rotation, Flash distribue automatiquement une zone de 45° (paramétrable) de part et d'autre de la position courante de l'objet.
8. Le squelette doit être activé pour le mode Exécution. Pour cela, cliquez sur le calque du squelette. Puis, dans l'Inspecteur de propriétés, dans le menu Type de la catégorie Options, sélectionnez Exécution.

Définition du squelette

Pour comprendre la manière dont nous interagissons en ActionScript avec les éléments du squelette, nous devons au préalable bien saisir la manière dont celui-ci a été construit. Dans notre exemple, nous disposons d'une armature composée de 15 segments. Certains d'entre eux sont rattachés au même nœud de liaison. Chaque branche qui se divise, dans la hiérarchie descendante de l'armature, n'est pas considérée comme un ensemble indépendant, mais bien comme faisant partie d'un même ensemble. Ce qui distingue une division par rapport à une autre est son ordre d'apparition dans la liste d'affichage du squelette (ce qui équivaut par défaut à l'ordre de création des nœuds), ou à son nom d'occurrence. Pour contrôler une branche plutôt qu'une autre, nous ciblerons donc celle-ci par son ordre d'affichage (`getChildAt(0)`) ou par son nom d'occurrence (`getNodeByName("nomDuNoeud")`).

Ainsi, avant de programmer des actions, il est souhaitable d'identifier clairement la hiérarchie dont nous disposons. Pour notre exemple, nous avons réalisé un schéma qui reprend l'ordre de chaque segment de liaison tel qu'identifié depuis la liste d'affichage (voir Figure 4.3) (nous présentons la manière d'obtenir l'ordre d'affichage de chaque segment dans la lecture du code qui suit).

Figure 4.3
Aperçu de la structure du squelette.



Dans notre exemple à nouveau, relevons que nous avons commencé la construction du squelette à partir du bassin, en progressant d'abord vers le cou, puis la jambe gauche et enfin vers la jambe droite. Le bassin se divise donc, dès le départ, en trois branches distinctes que nous pouvons identifier par la branche 0 (cou), la branche 1 (jambe gauche) et la branche 2 (jambe droite). Le cou, à son tour, se divise en deux parties, une pour chaque bras.

Nous obtenons donc les subdivisions suivantes : avec la branche 0 qui correspond au bras droit, et 1 qui correspond au bras gauche. Mais, dans le contexte hiérarchique qui part du bassin, et se prolonge par le thorax avant d'atteindre les épaules, nous pouvons identifier la branche du bras droit comme liaison 0.0.0 et celle du bras gauche comme liaison 0.0.1. Et ainsi de suite. En somme, à chaque progression vers un sous-niveau de l'armature, nous ajoutons une valeur.

Création de squelettes humains. Le livre *L'art du bluff avec Flash CS4,* de Chris Georgennes, aux éditions Pearson, propose d'ajouter un segment à l'aplomb du bassin afin de le relier au sol. Pour de plus amples renseignements sur la construction de squelettes d'animation, notamment humains, reportez-vous à cet ouvrage.

Le code affiché est composé de trois parties. D'abord, nous définissons les classes et les variables à utiliser (initialisation). Ensuite, nous structurons les éléments destinés à être animés (identification des éléments animables). Enfin, nous réalisons l'animation. Celle-ci se caractérise d'abord par le contrôle du mouvement des liaisons du squelette (animation du squelette), puis par la mise en relation de ces mouvements avec le pointeur à travers la fonction.

Nous commençons par importer les classes requises pour l'animation du squelette :

```
//--------------------------- initialisation
import fl.ik.*;
```

La classe `ik` est utilisée pour la gestion des mouvements de liaison du squelette. L'astérisque sous-entend l'importation des sous-classes `IKArmature` (définition du squelette), `IKBone` (définition des segments qui contiennent les liaisons), `IKEvent` (définition des événements pour les fonctions), `IKJoint` (définition des liaisons à animer), `IKManager` (définition du statut du squelette) et `IKMover` (définition des animations en tant que telles).

Puis, nous activons le squelette pour pouvoir, ensuite, en animer la composante :

```
//--------------------------- actions
// définition du squelette

IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Droide");
squelette.registerElements(stage);
```

Dans ces premières lignes, grâce à la sous-classe `IKManager` qui permet de statuer sur le comportement du squelette, nous affectons le contrôle du squelette à la scène courante. Mais cette instruction est optionnelle car l'API de Flash le gère automatiquement :

```
IKManager.setStage(stage);
```

Puis, nous convertissons le squelette de la scène en objet `IKArmature` programmable. Pour cela, nous instancions un nouvel objet `IKArmature` en reprenant, en paramètre de la méthode `getArmatureByName()`, le nom d'occurrence du squelette que nous avons pu identifier depuis l'Inspecteur de propriétés directement sur la scène principale :

```
var squelette:IKArmature= IKManager.getArmatureByName("Squelette_Droide");
```

Pour cibler le squelette, nous utilisons ici `getArmatureByName` puisque nous en connaissons le nom. Mais nous aurions aussi bien pu le cibler par son ordre d'affichage, dans la liste d'affichage des squelettes indépendante de la liste d'affichage de la scène, avec la méthode `getArmatureAt()`.

Nous mémorisons ensuite toute l'ossature du squelette présente sur la scène de sorte à en autoriser le contrôle avec la méthode `registerElements()` :

```
squelette.registerElements(stage);
```

Une fois que nous avons instancié la structure globale, nous pouvons à présent entrer dans les maillons du squelette et déclarer chacune des liaisons dont nous souhaitons pouvoir contrôler la position :

```
var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0);
```

Pour déclarer la première liaison, nous créons un nouvel objet `IKJoint`. Nous ciblons le premier nœud de liaison en reprenant d'abord le nom du squelette véhiculé par la variable squelette, puis, ciblons le nœud racine avec `rootJoint`, c'est-à-dire le centre de transformation du bassin bas, puis, la première branche qui remonte la colonne vertébrale avec `getChildAt(0)` (voir Figure 4.4).

Pour vérifier que l'ordre d'apparition choisi en paramètre correspond bien à l'objet que nous souhaitons rendre disponible sur la scène, nous utilisons une action `trace` en reprenant le ciblage auquel nous associons la propriété `.name` pour connaître le nom d'occurrence de l'objet :

```
trace(squelette.rootJoint.getChildAt(0).getChildAt(0).name)
```

Nous obtenons, dans la fenêtre de sortie, la réponse suivante :

```
Segment_cou
```

Nous reproduisons ensuite le principe pour tous les segments, afin d'en permettre l'animation :

```
var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0);
var segment_colonneHaut:IKJoint=squelette.rootJoint.getChildAt(0).
➥ getChildAt(0);
var segment_epauleD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(0);
var segment_humerusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(0).getChildAt(0);
var segment_cubitusD:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(0).getChildAt(0).getChildAt(0);
var segment_epauleG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(1);
var segment_humerusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(1).getChildAt(0);
var segment_cubitusG:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0).
➥ getChildAt(1).getChildAt(0).getChildAt(0);
var segment_illiaqueG:IKJoint=squelette.rootJoint.getChildAt(1);
var segment_femurG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0);
var segment_tibiaG:IKJoint=squelette.rootJoint.getChildAt(1).getChildAt(0).
➥ getChildAt(0);
var segment_illiaqueD:IKJoint=squelette.rootJoint.getChildAt(2);
var segment_femurD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0);
var segment_tibiaD:IKJoint=squelette.rootJoint.getChildAt(2).getChildAt(0)
➥ .getChildAt(0);
```

Observons que pour atteindre un maillon enfant, nous utilisons la technique de ciblage pointé en ajoutant, pour chaque nouveau segment enfant, la méthode `getChildAt()` avec le

numéro qui correspond à l'ordre d'affichage de l'enfant ciblé, conformément au schéma observé plus haut.

Il est possible de cibler uniquement un segment à partir de son identifiant, comme suit :

```
var segment1:IKBone=squelette.getBoneByName("ikBoneName1");
```

Ou à partir de son ordre d'affichage :

```
var segment1:IKBone=squelette.getBoneByName("ikBoneName1");
```

Une fois les liaisons définies, il ne reste plus qu'à animer.

Animer le squelette

La propriété rotation n'étant pas autorisée en écriture par la classe `IK`, pour animer une liaison, nous définissons donc un nouvel objet de transition qui déplace une liaison d'un point donné en X et Y à un autre point en X et Y. Pour mieux comprendre le procédé, simplifions notre exemple en nous concentrant d'abord sur le fonctionnement d'une animation de liaison :

```
// animation du squelette
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
var arrivee:Point=new Point(0,0);
mouvement1.moveTo(arrivee);
```

Nous déclarons ici, en premier, l'objet `IKMover` qui constitue le moteur de l'animation et redistribue, pour chaque segment lié à la chaîne courante, les valeurs nécessaires à leur éventuel repositionnement :

```
var mouvement1:IKMover= new IKMover(segment_humerusD,segment_humerusD.position);
```

L'animation appelle deux paramètres qui sont respectivement l'objet à animer (en l'occurrence, la liaison `segment_humerusD` définie précédemment), et les coordonnées du point de départ de l'animation de cet objet, à partir d'une valeur de type `Point(X,Y)`.

Définition du constructeur `Point()`. *Une valeur de type `Point(X,Y)` véhicule, dans une seule variable, les propriétés* x *et* y *de l'objet auquel elle est rattachée. Ce constructeur peut, pour information, être appelé par les classes suivantes :* `BitmapData`, `DisplayObject`, `DisplayObjectContainer`, `DisplacementMapFilter`, `NativeWindow`, `Matrix` *ou* `Rectangle`.

Ainsi, nous pouvons déterminer la valeur de position de départ à travers une nouvelle variable ou en reprenant la position courante de l'objet à animer. Avec une variable, nous obtenons :

```
var depart:Point=new Point(100,50) ;
var mouvement1:IKMover= new IKMover(segment_humerusD,depart);
```

En reprenant directement la propriété position de l'objet courant, nous obtenons :

```
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
```

Nous récupérons directement la position courante de l'objet à animer. Puis, nous définissons le point d'arrivée :

```
var arrivee:Point=new Point(0,0);
```

Et enfin, nous activons l'animation :

```
mouvement1.moveTo(arrivee);
```

L'animation reprend l'objet `IKMover` initié plus haut et utilise la méthode `moveTo` afin de créer une interpolation entre le point de départ défini dans l'objet `IKMover` et le point d'arrivée passé ici en paramètre.

Dans le fichier exemple de cette section, l'interpolation que nous venons de décrire, est placée dans une fonction associée à un gestionnaire d'événements de type `Event.ENTER_FRAME` afin que le calcul de la position puisse être redéfini perpétuellement. Nous plaçons, en paramètre du point d'arrivée, les valeurs qui correspondent à la position courante du pointeur avec `mouseX` et `mouseY`. Nous obtenons :

```
// animation du squelette
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
   var arrivee:Point=new Point(mouseX,mouseY);
   mouvement1.moveTo(arrivee);
}
```

Pour optimiser encore plus l'animation, vous pouvez éventuellement typer la variable arrivee en dehors de la fonction, et la mettre à jour seulement dans la fonction, comme suit :

```
// animation du squelette
var mouvement1:IKMover= new
IKMover(segment_humerusD,segment_humerusD.position);
var arrivee:Point ;
stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
   arrivee=new Point(mouseX,mouseY);
   mouvement1.moveTo(arrivee);
}
```

En modifiant le paramètre qui appelle le nom de l'objet à animer, dans l'objet `IKMover`, vous pouvez animer le squelette à partir de chacun des axes de rotation (voir Figures 4.4 et 4.5).

À retenir

- La classe `ik` permet d'animer la position des liaisons établies entre plusieurs segments d'un même squelette.
- Pour programmer l'animation d'un squelette, vous devez au préalable construire le squelette dans l'interface auteur de Flash en activant l'affichage pour l'exécution depuis l'Inspecteur de propriétés.
- Pour faciliter la gestion de l'articulation des segments, vous devez placer judicieusement les centres de transformation de chaque symbole avant la création du squelette.
- Pour faciliter la définition des liaisons à animer, vous devez clairement identifier l'ordre d'affichage de ces éléments.
- Il est possible d'associer le mouvement d'une liaison à la position courante du pointeur.

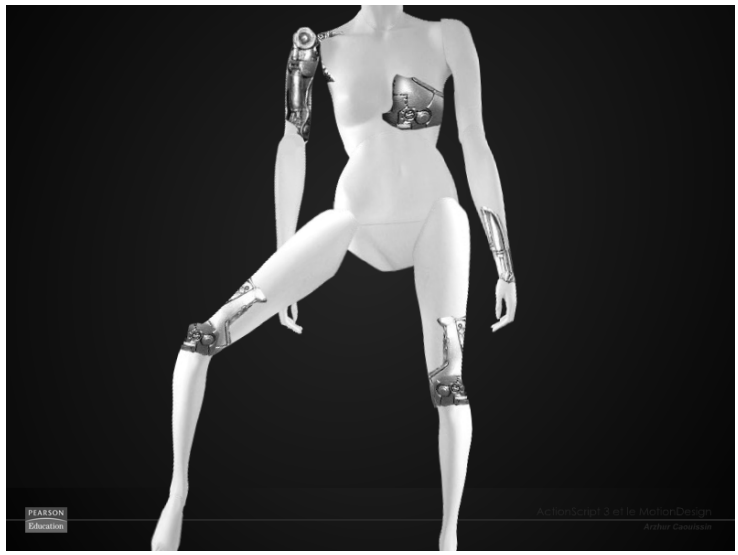


Figure 4.4
Aperçu de l'animation avec l'objet `segment_tibiaD`.

Figure 4.5
Aperçu de l'animation avec l'objet `segment_avant-BrasG`.

Programmer un mouvement organique

De la même manière que nous animons des squelettes de symboles, nous pouvons aussi programmer l'animation de squelettes pour des formes graphiques vectorielles.

L'utilisation des squelettes avec les formes vectorielles permet de réaliser, par exemple, des animations de "lipsync" (mouvement de bouche), des animations végétales, organiques, liquides ou hétérées (radiosité, vapeurs, nuages), entre autres, mais pas de formes transparentes.

Dans cette section, nous utilisons l'animation d'une plante qui se courbe au passage d'une abeille. Nous utilisons ici les paramètres de la méthode `moveTo` pour magnétiser la dernière

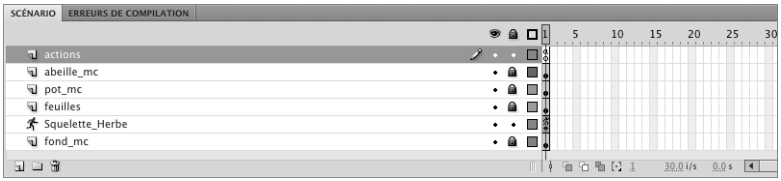
liaison de la plante sur l'abscisse du symbole abeille (voir Figure 4.6). L'abeille, elle, est animée à l'aide de la classe `Greensock TweenMax`.

Figure 4.6
Aperçu du document.

Exemples > ch4_programmationDeSquelettes_2.fla

Dans la scène de notre document, nous pouvons voir, au-dessus du calque `fond_mc`, un squelette attaché à une forme graphique nommé `Squelette_Herbe`. Au premier plan de cette armature, apparaissent quelques feuilles et un pot pour la décoration, puis un symbole de type MovieClip nommé `abeille_mc` (voir Figure 4.7).

Figure 4.7
Aperçu du scénario de la scène principale.



Dans le calque des actions, nous pouvons lire le code suivant :

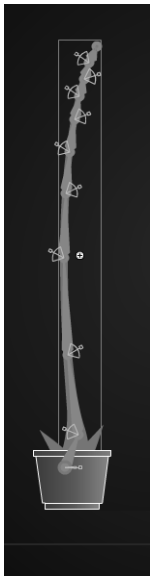
```
//--------------------------- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
//
import fl.ik.*;

//--------------------------- actions
// définition du squelette

IKManager.setStage(stage);
```

```
var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Herbe");
squelette.registerElements(stage);

var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0)
➥ .getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).
➥ getChildAt(0).getChildAt(0).getChildAt(0);

// animation du squelette

var mouvement1:IKMover= new
IKMover(segment_colonneBas,segment_colonneBas.position);

stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
   var arrivee:Point=new Point(abeille_mc.x,abeille_mc.y);
   mouvement1.moveTo(arrivee);
}

var tweenAbeille:TweenMax;
tweenAbeille=TweenMax.to(abeille_mc,5,{x:900,delay:2,ease:Strong.easeInOut});
tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation);

function suiteAnimation(evt:TweenEvent) {
   tweenAbeille=TweenMax.to(abeille_mc,8,{x:300,delay:0,ease:Back.easeInOut});
}
```

Comme dans l'exemple précédent, nous devons commencer par identifier l'ordre d'affichage des segments du squelette. Nous procédons, de même, en repérant le nom des occurrences depuis l'Inspecteur de propriétés et en déterminant le nombre de liaisons disponibles ainsi que la présence éventuelle de subdivisions multiples (voir Figure 4.8).

Figure 4.8

Aperçu de la structure du squelette nommé `Squelette_Herbe`.



Notre squelette ne comporte pas de subdivision. Il compte dix liaisons. Son nom d'occurrence est `Squelette_Herbe`. Les contraintes de rotation des liaisons ont été activées depuis l'Inspecteur de propriétés.

Ces contraintes ont plus particulièrement été marquées sur la base de l'ossature afin de renforcer l'effet d'enracinement et d'inertie pour la partie proche du pot, et donner un effet de pousse légère au sommet de la tige. Des contraintes identiques sur toute la longueur auraient donné une impression de structure entièrement molle ou totalement rigide, selon les valeurs d'angle enregistrées.

Dans les actions, nous importons d'abord les classes requises :

```
//--------------------------- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
//
impfort fl.ik.*;
```

En plus de la classe `ik` utilisée pour la cinématique inverse, nous importons la classe `TweenMax (gs)` pour animer le déplacement de l'abeille.

Plus loin, nous définissons le squelette avec `IKManager` et `IKAmrature`, comme vu dans la section précédente.

```
//--------------------------- actions
// définition du squelette

IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Herbe");
squelette.registerElements(stage);
```

Puis, comme nous l'avons vu précédemment , nous définissons le segment à animer:

```
var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0)
➥ .getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).
➥ getChildAt(0).getChildAt(0).getChildAt(0);
```

Ici, le segment que nous ciblons est le dernier de la chaîne. Elle en compte dix. Nous répétons donc dix fois le ciblage progressif jusqu'à atteindre notre cible.

Puis, nous animons l'objet en liant, cette fois-ci, ses coordonnées à celles de l'abeille, par l'intermédiaire d'une transition `TweenMax` :

```
// animation du squelette

var mouvement1:IKMover= new
IKMover(segment_colonneBas,segment_colonneBas.position);

stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
   var arrivee:Point=new Point(abeille_mc.x,abeille_mc.y);
   mouvement1.moveTo(arrivee);
}
var tweenAbeille:TweenMax;
tweenAbeille=TweenMax.to(abeille_mc,5,{x:900,delay:2,ease:Strong.easeInOut});
```

Lorsque l'abeille se déplacera à partir de sa position `x` actuelle (inférieure à 0) vers la position définie dans la transition, soit 900 px, la fonction associée à la classe `Event` et à l'événement `ENTER_FRAME` va permettre de repositionner le dernier segment selon la position `x`

de l'abeille en mouvement. Les segments étant reliés, et la méthode `MoveTo` répercutant les valeurs de positionnement vers les autres liaisons en fonction de leurs contraintes respectives, l'herbe haute va se courber pour suivre l'abeille dans son déplacement.

Nous terminons avec un enchaînement de transition :

```
tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation);

function suiteAnimation(evt:TweenEvent) {
   tweenAbeille=TweenMax.to(abeille_mc,8,{x:300,delay:0,ease:Back.easeInOut});
}
```

Ici, une seconde interpolation succède à la première, grâce à l'événement `COMPLETE` associé à la classe `TweenEvent`. Cette transition fait revenir, en marche arrière, l'abeille, de sorte qu'elle se place juste au-dessus de la plante. La fonction `activerMouvement1` qui demeure toujours active permet à la plante animée de continuer de suivre l'abeille jusqu'à son point arrêt. Tout nouveau mouvement que fera l'abeille par la suite provoquera un nouveau mouvement de l'herbe (voir Figure 4.9 à 4.12).

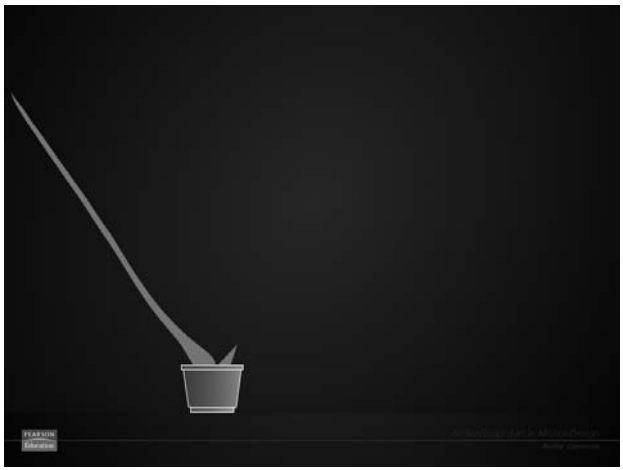


Figure 4.9
Aperçu de l'animation organique (1/4).

Figure 4.10
Aperçu de l'animation organique (2/4).

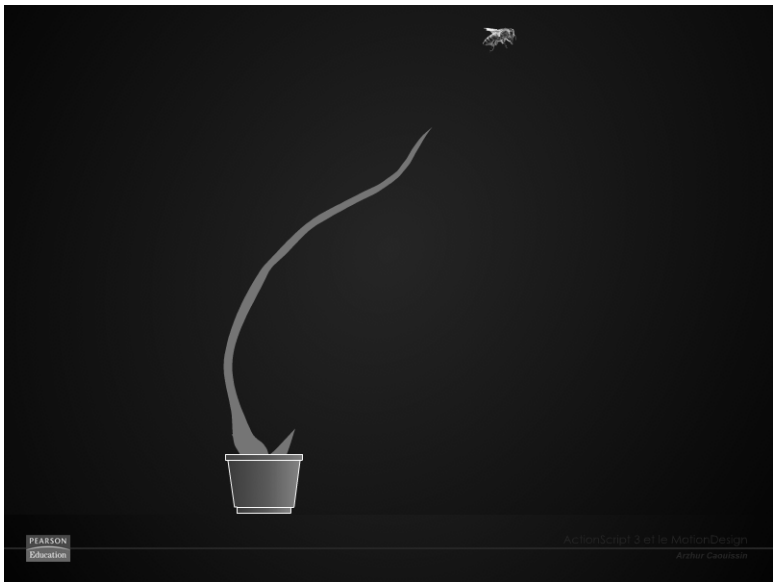


Figure 4.11
Aperçu de l'animation organique (3/4).

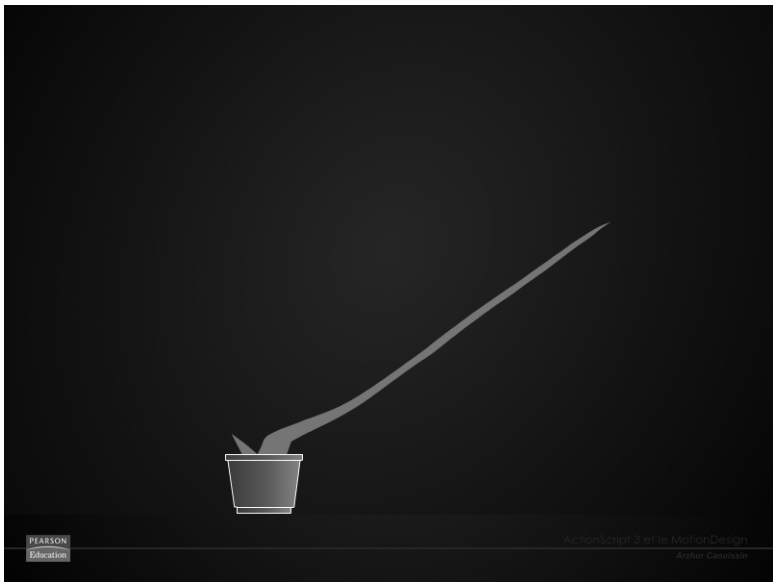


Figure 4.12
Aperçu de l'animation organique (4/4).

À retenir

- Il est possible de programmer l'animation de formes vectorielles avec la classe `ik`.
- Une liaison d'un squelette de forme vectorielle peut suivre un objet en mouvement.
- Il est possible de combiner des animations de type `TweenMax` avec des transitions de la classe `IKMover`.

Basculer du mode animation programmée au mode exécution

La programmation d'une animation de squelette requiert, pour le squelette actif de la scène courante, que le menu Type de la catégorie Options soit défini sur Exécution. Mais, une fois que l'animation est programmée, il est possible de rendre la main à l'utilisateur pour lui autoriser de déplacer lui-même les segments des objets animés.

Dans cette section, nous présentons comment basculer du mode d'animation programmée au mode 100 % interactif de l'affichage de l'armature, avec la classe `IKManager` (voir Figure 4.13).

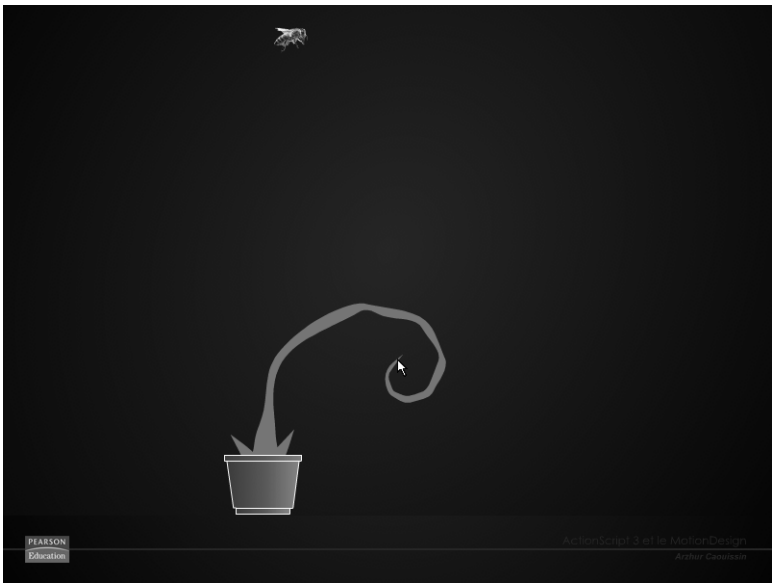


Figure 4.13
L'utilisateur prend le contrôle du positionnement des liaisons.

Exemples > ch4_programmationDeSquelettes_3.fla

Le document que nous utilisons présente la structure suivante : dans la scène, nous retrouvons les mêmes éléments que dans la section précédente.

Dans le calque des actions, nous pouvons lire le code suivant, basé sur l'exemple précédent, mais nous y avons ajouté les commandes nécessaires au basculement de la propriété :

```
//-------------------------- initialisation
import gs.*;
import gs.easing.*;
import gs.events.*;
//
import fl.ik.*;

//-------------------------- actions
// définition du squelette
```

```
IKManager.setStage(stage);

var squelette:IKArmature=IKManager.getArmatureByName("Squelette_Herbe");
squelette.registerElements(stage);

var segment_colonneBas:IKJoint=squelette.rootJoint.getChildAt(0).getChildAt(0)
➥ .getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).getChildAt(0).
➥ getChildAt(0).getChildAt(0).getChildAt(0);

// animation du squelette

var mouvement1:IKMover= new
IKMover(segment_colonneBas,segment_colonneBas.position);

stage.addEventListener(Event.ENTER_FRAME, activerMouvement1);
function activerMouvement1(evt:Event) {
  var arrivee:Point=new Point(abeille_mc.x,abeille_mc.y);
  mouvement1.moveTo(arrivee);
}

var tweenAbeille:TweenMax;
tweenAbeille=TweenMax.to(abeille_mc,5,{x:900,delay:2,ease:Strong.easeInOut});
tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation);

function suiteAnimation(evt:TweenEvent) {
  tweenAbeille=TweenMax.to(abeille_mc,8,{x:300,delay:0,ease:Back.easeInOut});
  tweenAbeille.addEventListener(TweenEvent.COMPLETE,suiteAnimation2);
}

// interrompre l'animation et basculer en mode interactif

var nombreDeBoucle:Number=1;
var dureeBoucle:Number=1000;
var boucle:Timer=new Timer(dureeBoucle,nombreDeBoucle);

function suiteAnimation2(evt:TweenEvent) {
  IKManager.trackAllArmatures(true);
  boucle.start();
  boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
}

function lancerBoucle(evt:TimerEvent) {
  stage.removeEventListener(Event.ENTER_FRAME, activerMouvement1);
}
```

Pour détecter la fin de l'animation, nous utilisons un nouvel écouteur attaché à l'objet animé, en l'occurrence, il s'agit de la seconde animation de l'abeille définie dans la fonction `suiteAnimation1`. Lorsque cette animation est achevée, nous programmons l'exécution d'une nouvelle fonction `suiteAnimation2`. Dans cette fonction, trois instructions sont ajoutées :

```
function suiteAnimation2(evt:TweenEvent) {
  IKManager.trackAllArmatures(true);
  boucle.start();
  boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
}
```

La première reprend la classe `IKManager` employée au début du code pour spécifier, à travers la propriété `trackAllArmatures`, que le déplacement de toute l'armature par l'utilisateur est désormais actif (`true`), selon les mêmes contraintes de rotation, définies initialement depuis l'Inspecteur de propriétés, que pour l'animation programmée.

Les deux suivantes activent d'abord un chronomètre de type `Timer` défini plus haut. Ces instructions associent, à ce même chronomètre, une nouvelle fonction qui interrompt l'animation de la liaison `activerMouvement1` dont la position est rattachée à celle de l'abeille, grâce à la méthode `removeEventListener` :

```
function lancerBoucle(evt:TimerEvent) {
  stage.removeEventListener(Event.ENTER_FRAME, activerMouvement1);
}
```

Notez que nous aurions pu interrompre la fonction directement dans la fonction `suiteAnimation2`, avec la même méthode `removeEventListener`. Mais, en procédant de la sorte, l'animation peut rendre un peu brutal l'arrêt du positionnement des éléments de liaison. En effet, l'animation est gérée par la classe `IKMover` et le moteur utilise un repositionnement qui comporte un algorithme d'accélération. L'animation aurait alors été interrompue au moment où le `TweenMax` s'achève, même si l'interpolation `IKMover` n'était pas terminée (du fait de l'amortissement). Afin que l'animation s'arrête de manière plus naturelle, nous avons recours à un Timer. Ceci laissera le temps à l'animation de l'abeille de terminer l'effet d'amortissement de son arrêt. Ensuite l'utilisateur pourra prendre la main.

À retenir

- *Vous pouvez, à l'issue d'une animation programmée, rendre la main à l'utilisateur, pour lui permettre de déplacer lui-même les liaisons du squelette de l'objet animé.*
- *Les transitions `IKMover` utilisent un repositionnement qui comporte un algorithme d'accélération et obligent, pour les interrompre, d'utiliser un chronomètre de type Timer afin d'éviter que la rupture ne soit trop sèche.*
- *Pour basculer d'un mode d'exécution à un autre, nous utilisons la classe `IKManager`.*

Activer un squelette chargé dans un SWF

Lorsqu'un document SWF qui comporte un squelette en mode Exécution est importé dans un nouveau document SWF, le squelette n'est pas actif dans l'interface de ce nouveau document. Cela s'explique par le fait que seul le document qui contient le squelette est compilé avec les objets d'affichages qui gèrent ce squelette. Même, en construisant un nouveau squelette dans le fichier appelant, le squelette importé ne réagit toujours pas. Il en résulte que tout document, dépourvu d'une action spécifique sur l'importation des squelettes, ne peut exécuter le squelette contenu dans tout document importé. Il est donc nécessaire de reconfigurer le document appelant, de sorte qu'il puisse exécuter correctement le fichier appelé.

Attention

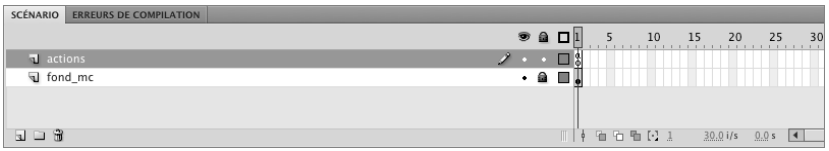
Les noms du fichier appelant. *Suite à un bogue du moteur Flash, il est possible que vous ne parveniez pas à importer un document si vous placez des caractères non ascii dans le nom du fichier. Utilisez de préférence des lettres collées, sans séparateur tiret -). Pour séparer les mots, préférez le underscore ou tiret du bas (_). Les tirets peuvent ne pas être admis dans certaines configurations.*

Dans cette section, nous importons un document SWF contenant un squelette de forme, publié en mode Exécution et le réactivons pour l'exécuter dans ce premier document.

Exemples > ch4_programmationDeSquelettes_4.fla
Exemples > ch4_programmationDeSquelettes_4a.fla

Le document que nous utilisons se nomme "ch4ProgrammationDeSquelettes4.fla". Dans la scène, rien ne figure sinon le calque `fond_mc` (voir Figure 4.14).

Figure 4.14
Aperçu du scénario de la scène principale.



Dans le calque des actions, nous pouvons lire le code suivant :

```
//---------------------- initialisation
import fl.ik.*;

//---------------------- chargement
var chemin:URLRequest = new URLRequest("ch4-programmationDeSquelettes-4a.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);

chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);

var squelette:IKArmature;

function afficher(Evt:Event){
  addChild(chargeur);
  //
  IKManager.setStage(stage);
  squelette = IKManager.getArmatureByName("Squelette_Herbe");
  squelette.registerElements(stage);
  IKManager.trackAllArmatures(true);
}
```

D'abord, nous importons la classe `ik` pour reconstituer le squelette :

```
//---------------------- initialisation
import fl.ik.*;
```

Puis, nous définissons un nouveau chargeur afin d'importer le fichier SWF contenant le squelette :

```
//---------------------- chargement
var chemin:URLRequest = new URLRequest("ch4-programmationDeSquelettes-4a.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);
chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);
```

Lorsque le chargement du document est terminé, nous commençons par définir un espace mémoire pour la création d'une armature :

```
var squelette:IKArmature;
```

Puis, nous exécutons la fonction qui affiche le contenu et réactive l'armature du document importé :

```
function afficher(Evt:Event){
  addChild(chargeur);
  //
  IKManager.setStage(stage);
  squelette = IKManager.getArmatureByName("Squelette_Herbe");
  squelette.registerElements(stage);
  IKManager.trackAllArmatures(true);
}
```

À la suite de l'instruction `addChild`, la fonction place l'affectation du moteur de squelette sur la scène courante avec `setStage()`. L'instruction `getArmatureByName()` permet de pointer sur le squelette en utilisant le nom d'occurrence utilisé dans le SWF importé. Puis, elle mémorise les propriétés du squelette sur la scène globale avec `registerElements(stage)` et l'active avec `trackAllArmatures` passé sur `true`. C'est la dernière instruction qui permet de réactiver le mode d'affichage Exécution.

Cette fois, en publiant le document SWF, nous pouvons constater que le document importé est activé. Il est donc possible de déplacer les liaisons avec le pointeur et la plante préserve les contraintes définies dans le document importé (voir Figure 4.15).

Figure 4.15
Aperçu du SWF importé et exécuté.

À retenir

- Il est possible de réactiver un squelette désactivé automatiquement, après l'avoir importé dans un nouveau document SWF. Pour cela, vous devez réactiver le squelette par programmation dans le document appelant, afin de réimplanter le moteur d'exécution du squelette exclusif à l'API de Flash, lors de la publication du nouveau document.

Synthèse

Dans ce chapitre, vous avez appris à programmer des animations à partir de squelettes créés manuellement dans l'interface auteur de Flash. Vous avez appris à créer des animations mécaniques et organiques en exploitant des structures à base de symboles ou de formes. Vous avez également appris à programmer les animations de squelette en utilisant des objets et/ou le pointeur de la souris comme guide de mouvement. Vous savez enfin activer le mode d'exécution une fois l'animation achevée, et permettre à l'utilisateur de reprendre la main sur l'armature à la fin d'une interpolation. Vous êtes en mesure à présent de réaliser des animations de structure complexes et encore une fois dynamiques.