13 Les systèmes de navigation avancés

Introduction

Les boutons proposés par défaut dans Flash offrent des fonctionnalités souvent en deçà des besoins réels attendus en production, Par exemple, il est particulièrement difficile de contrôler, par ActionScript, des contenus distribués au sein de ce type d'objet. Ils ne proposent pas non plus d'état visité comme le ferait un simple hyperlien en HTML. En utilisant des symboles de type MovieClip, il est possible de réaliser des systèmes de navigation plus personnalisables, plus souples à mettre en forme, et plus originaux.

Dans ce chapitre, nous allons aborder l'utilisation de symboles de type MovieClip à travers différents dispositifs de navigation avancés.

Boutons MovieClip fixes avec état activé et visité

Dans cette première section, nous vous proposons de réaliser un menu avec différents boutons disposant, pour chacun d'eux, les états survolé (`over`), sortie (`out`), appuyé (`down`), relevé (`up`) et visité (`visited`).

ActionScript 3 nous permet de centraliser les interactions au sein d'un même gestionnaire grâce aux propriétés `target` et `currentTarget`. Nous les utilisons dans cet exemple pour optimiser notre code.

Nous présentons ici trois déclinaisons du même menu avec quelques variantes de codage. La première version exécute tous les états. La deuxième version, aussi, mais elle utilise une structure conditionnelle de type `Switch`, plus souple pour des menus longs. La troisième enfin aborde l'ergonomie en traitant les boutons en tant qu'éléments actifs et non plus simplement visités. Lorsqu'une nouvelle rubrique est affichée, les boutons des autres entrées sont restaurés.

Exemples > ch13_systemesNavigation_1.fla
Exemples > ch13_systemesNavigation_1b.fla
Exemples > ch13_systemesNavigation_1c.fla

Les boutons visités

Dans le document "ch13_systemesNavigation_1.fla", la scène principale affiche un symbole MovieClip de nom d'occurrence `menu_mc` (voir Figure 13.1).

À l'intérieur de ce symbole sont répartis, vers les calques, trois boutons respectivement nommés `bouton1_mc`, `bouton2_mc` et `bouton3_mc`. Sont également visibles : un filet et quelques formes graphiques ajoutées pour l'habillage (voir Figure 13.2).

Figure 13.1
Aperçu de la scène principale.

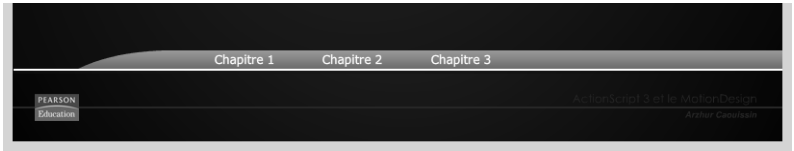
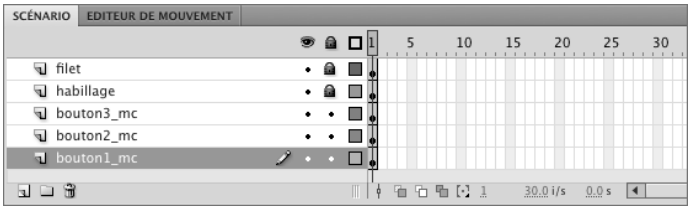
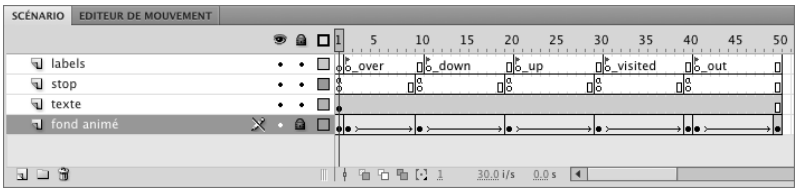


Figure 13.2
Aperçu du scénario du MovieClip `menu_mc`.



Dans le scénario de chacun de ces MovieClip, nous distinguons quatre calques dont certains affichent des interpolations (voir Figure 13.3).

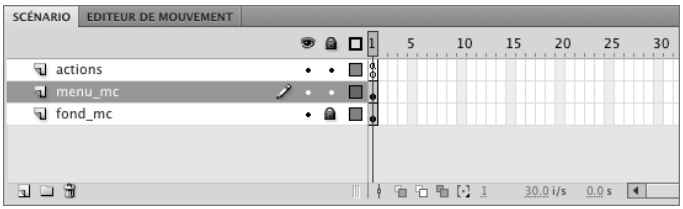
Figure 13.3
Aperçu du scénario de chaque bouton MovieClip.



Le calque du bas contient une série d'animations placées les unes à la suite des autres. Le calque nommé `texte` affiche le libellé de chaque bouton. Le calque `stop` interrompt la tête de lecture avant chaque nouvelle animation, par la commande `stop()`. Enfin, le calque `labels` distribue une série d'étiquettes (*labels*) qui nous permettent d'identifier chaque animation en fonction de son rôle. Ces étiquettes permettent aussi de piloter par ActionScript le placement de la tête de lecture à ces endroits précis, lors du changement d'état survenu pour le bouton (état survolé, sortie, état appuyé, état relâché et état visité).

Dans la scène principale, au-dessus du calque `fond_mc` et `menu_mc`, apparaît le calque `Actions` (voir Figure 13.4).

Figure 13.4
Aperçu du scénario de la scène principale.



Dans le calque `Actions`, nous pouvons lire le code suivant :

```
//---------------------- actions
//over
```

```
menu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
function over (evt:MouseEvent) {
   evt.target.gotoAndPlay("_over");
}
//down
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,down);
function down (evt:MouseEvent) {
   evt.target.gotoAndPlay("_down");
   evt.target.etatVisit=true;
   //
   if (evt.target.name=="bouton1_mc") {
      trace("action 1");
   }
   if (evt.target.name=="bouton2_mc") {
      trace("action 2");
   }
   if (evt.target.name=="bouton3_mc") {
      trace("action 3");
   }
}
//out
menu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
   if (evt.target.etatVisit==true) {
      evt.target.gotoAndPlay("_visited");
   } else {
      evt.target.gotoAndPlay("_out");
   }
}
//up
menu_mc.addEventListener(MouseEvent.MOUSE_UP,up);
function up (evt:MouseEvent) {
   evt.target.gotoAndPlay("_up");
}
```

Pour appréhender le mécanisme du menu, nous devons d'abord comprendre que tous les boutons sont rassemblés à l'intérieur d'un symbole MovieClip nommé `menu_mc`. ActionScript 3 nous permet de cibler, grâce à la propriété `target`, chacun des objets qui capture l'événement, dans un même conteneur. Nous pouvons donc automatiser le processus d'interactivité avec plusieurs boutons réunis dans un clip, sans avoir à recréer de gestionnaire d'événements pour chacun d'entre eux.

Mais, nous souhaitons aussi exécuter un nombre d'événements parfois contradictoires. Par exemple, nous souhaitons qu'un bouton survolé affiche, en sortie, l'état par défaut. Mais, nous aimerions aussi que, si ce bouton était déjà activé, que ce soit un autre état qui apparaisse : l'état visité. Pour ce faire, nous utilisons une variable et une structure conditionnelle, qui, selon l'objet activé, va exécuter l'une ou l'autre des interpolations et ainsi permettre de résoudre cette pseudo-contradiction.

Dans le programme, nous créons un premier gestionnaire d'événements pour la définition de l'état survolé :

```
//over
menu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
```

```
function over (evt:MouseEvent) {
   evt.target.gotoAndPlay("_over");
}
```

C'est bien le symbole `menu_mc` qui reçoit l'écouteur. Mais c'est bien l'occurrence de bouton survolée qui exécute la fonction (`evt.target`). Dans cette fonction, nous lisons l'interpolation placée à l'étiquette nommée `_over` lorsque le pointeur survole le bouton (`MOUSE_OVER`).

À la suite, un autre gestionnaire exécute une fonction lorsque le pointeur est enfoncé sur une occurrence de bouton (MOUSE_DOWN) :

```
//down
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,down);
function down (evt:MouseEvent) {
   evt.target.gotoAndPlay("_down");
   evt.target.etatVisit=true;
   //
   if (evt.target.name=="bouton1_mc") {
      trace("action 1");
   }
   if (evt.target.name=="bouton2_mc") {
      trace("action 2");
   }
   if (evt.target.name=="bouton3_mc") {
      trace("action 3");
   }
}
```

Dans cette fonction, nous distinguons plusieurs actions. D'abord, l'animation qui correspond à l'état `_down` est lancée :

```
evt.target.gotoAndPlay("_down");
```

Ensuite, nous créons une propriété `etatVisit`, pour chaque bouton activé, que nous passons sur `true`. Cette propriété nous sert à vérifier si l'objet a été cliqué ou non. Selon sa valeur, nous définissons plus loin le type d'état à afficher en sortie ("`_visited`" ou "`_sortie`").

Nous ajoutons ensuite des conditions qui permettent de lancer telle ou telle action, selon le bouton cliqué. Nous vérifions, par exemple, que le bouton cliqué se nomme `bouton1_mc`. Dans ce cas, un texte en rapport s'affiche dans la fenêtre de sortie.

C'est la définition de différentes structures conditionnelles qui nous épargne d'avoir à créer un nouveau gestionnaire d'événements propre à chaque bouton. L'ensemble des actions du menu est centralisé dans chacune de ces conditions.

Ensuite, un gestionnaire affiche les actions lorsque le pointeur sort des boutons (`MOUSE_OUT`) :

```
//out
menu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
   if (evt.target.etatVisit==true) {
      evt.target.gotoAndPlay("_visited");
   } else {
      evt.target.gotoAndPlay("_out");
   }
}
```

Dans cette fonction, nous définissons deux actions. La première active l'interpolation située à l'étiquette `_visited` si, et uniquement si, la valeur de la propriété `etatVisit` du bouton cliqué est passée sur `true`, c'est-à-dire, si le bouton, ou la rubrique associée à ce bouton, a bien été visitée.

Si le bouton pour lequel le pointeur sort n'est pas le bouton cliqué, la seconde action appelle l'interpolation de sortie simple (`_out`).

La structure *Switch... case*

Vous trouverez également, dans les exercices du livre, le fichier "ch13_systemesNavigation_1b.fla". Ce document propose une version alternative de la structure conditionnelle. La fonction intitulée `down` emploie ici une instruction `switch… case`, au lieu de `if` :

```
//down
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,down);
function down (evt:MouseEvent) {
   evt.target.gotoAndPlay("_down");
   evt.target.etatVisit=true;
   //
   switch (evt.target.name) {
      case "bouton1_mc":
         trace("action 1");
         break;
      case "bouton2_mc":
         trace("action 2");
         break;
      case "bouton3_mc":
         trace("action 3");
         break;
      default:
         trace("ni 0, 1, ou 2");
   }
}
```

Le principe de l'instruction `switch` est similaire à une structure `if`, à la différence que nous vérifions une seule fois l'objet de la la condition (`switch (evt.target.name)`). Selon la valeur identifiée après l'attribut `case`, nous spécifions ensuite les actions à exécuter. Les actions sont placées après les deux points et se terminent toujours par l'instruction `break`. Une instruction `default` permet d'exécuter une autre action lorsque la condition n'est vérifiée par aucun attribut `case`.

Cette structure est particulièrement adaptée pour les systèmes dont la condition à vérifier est toujours du même type, comme vérifier une valeur portée par une variable, et y associer un comportement en fonction du résultat obtenu.

Les boutons activés

Un troisième document, nommé "ch13_systemesNavigation_1c.fla", propose un système de navigation pour lequel chaque lien affiche l'état actif lorsque le bouton vient d'être enfoncé. Mais cet état est désormais restauré dès qu'un autre bouton reprend le focus.

Dans ce document, nous pouvons lire le code suivant :

```
//--------------------- initialisation
var actif:*=null;

//--------------------- actions menu
//down
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,down);
function down(evt:MouseEvent) {
   if (evt.target!=actif) {
      //up
      if (actif!=null) {
         actif.gotoAndPlay("_up");
      }
      //visited
      actif=evt.target;
      actif.gotoAndPlay("_visited");
   }
   switch (evt.target.name) {
      case "bouton1_mc":
         trace("action 1");
         break;
      case "bouton2_mc":
         trace("action 2");
         break;
      case "bouton3_mc":
         trace("action 3");
         break;
      default:
         trace("ni 0, 1, ou 2");
   }
}
//over
menu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
function over(evt:MouseEvent) {
   if (evt.target!=actif) {
      evt.target.gotoAndPlay("_over");
   }
}
//out
menu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out(evt:MouseEvent) {
   if (evt.target!=actif) {
      evt.target.gotoAndPlay("_out");
   }
}
```

Dans ce programme, nous initialisons d'abord une variable "interrupteur" que nous avons nommée actif :

```
//---------------------- initialisation
var actif:*=null;
```

Le type étoile (*) désigne tout type, sans en privilégier un en particulier. La valeur null assure qu'elle ne permettra aucune action tant que nous ne l'aurons pas modifiée.

La fonction down commence par modifier cette valeur lorsqu'un bouton est enfoncé (`actif=evt.target`). Les autres fonctions ainsi que les structures conditionnelles de celle-ci, vérifient alors que le bouton cliqué n'est pas celui préalablement activé. Selon la valeur détectée, elles lancent ou non les interpolations d'entrée (`_over`) ou de sortie (`_out`) des boutons du menu.

Menus avec des symboles boutons. *Il est possible de réaliser dynamiquement un menu à partir de symboles boutons et de cibler les contenus qui y sont distribués. Nous n'abordons pas cet aspect ici. Ces techniques sont décrites en détail au Chapitre 7 de l'ouvrage de Thibault Imbert, Pratique d'ActionScript 3, aux éditions Pearson.*

À retenir

- *Les boutons réalisés en MovieClip offrent plus de souplesse de manipulation et de valeur ajoutée créative, qu'un bouton classique de la classe* `Button`*. Il est notamment possible d'y introduire de nombreuses animations et de les contrôler facilement par ActionScript.*
- *Pour créer un état visité sur un bouton cliqué, nous devons ajouter une condition qui vérifie si le bouton activé est celui enregistré. Dans ce cas, nous jouons l'animation en rapport. Le cas échéant, nous spécifions une autre interpolation.*
- *En ActionScript 3, grâce à la propriété* `target`*, il est facile de centraliser les actions de tout un menu au travers d'un seul gestionnaire d'événements. Pour distinguer néanmoins les actions de chaque bouton, nous utilisons des structures conditionnelles.*
- *Une structure conditionnelle de type* `Swich` *est plus adaptée qu'une structure* `if`*, lorsque la condition à vérifier est toujours la même.*

Boutons MovieClip animés et interfaçables

La création de boutons classiques consiste à limiter la surface d'affichage à leur libellé. Les boutons de type interfaçables sont des MovieClip qui exécutent une animation et transforme la surface du bouton en fenêtre de contenu. Les éléments distribués dans cette fenêtre deviennent à leurs tours interactifs, mais ne doivent pas entrer en conflit avec les actions placées justement sur ce conteneur (voir Figure 13.5).

La difficulté de cet exemple repose sur la capacité à rendre les actions des éléments ajoutés, dans le conteneur MovieClip, indépendantes des actions du conteneur lui-même. Si le bouton qui déploie cette fenêtre possède ses propres actions d'ouverture et de fermeture.

Figure 13.5
Bouton interfacé.

Il devient effectivement plus compliqué d'ajouter, dans un objet déjà interactif, d'autres objets également interactifs.

Afin de permettre de placer des symboles cliquables à l'intérieur d'autres symboles déjà cliquables, nous devons considérer que :

- Les objets placés au-dessus des autres sont toujours prioritaires sur les actions à exécuter.
- Les objets doivent pouvoir être identifiés au moment précis où des actions leur sont attribuées. Ou bien, il n'est pas possible de cibler un objet qui n'apparaît pas dans la scène au moment où celui-ci est invoqué.
- En ActionScript 3, l'ensemble des objets enfants d'un conteneur remonte au conteneur principal les interactions éventuellement interceptées.

Pour ajouter des objets interactifs dans un MovieClip, lui-même interactif, nous procédons de l'une des deux manières suivantes. Soit nous plaçons tous les objets interactifs sur la première image du scénario de leur conteneur, quitte à les rendre invisible (propriété `visible=false`) afin qu'ils n'apparaissent pas au lancement de l'application et, qu'ils ne soient pas cliquables. Soit nous les isolons sur une image du scénario du conteneur, là où effectivement ils doivent apparaître, mais nous sommes alors contraints de placer les actions de ces objets à ce même emplacement, dans le scénario.

Dans cette section, nous abordons les deux solutions.

Différence entre les propriétés alpha et visible.
La propriété `alpha` *(*`nomDuSymbole_mc.alpha=0`*) rend les objets transparents mais toujours actifs. La propriété visible (*`nomDuSymbole_mc.visible=false`*) les rend invisibles et inactifs. L'alpha se définit par une valeur décimale comprise entre 0 et 1. La visibilité se définit par une valeur booléenne (*`true` *ou* `false`*).*

Méthode avec les actions dans le scénario du lien

Nous présentons d'abord la solution qui consiste à placer les actions des liens imbriqués, dans les liens eux-mêmes. Cette technique est la plus simple à appréhender. Elle permet de placer les objets dans le scénario uniquement lorsqu'ils doivent être actifs. Elle convient parfaitement pour des actions localisées, comme la lecture du scénario à partir d'un objet ajouté localement, sur une image-clé isolée, par exemple.

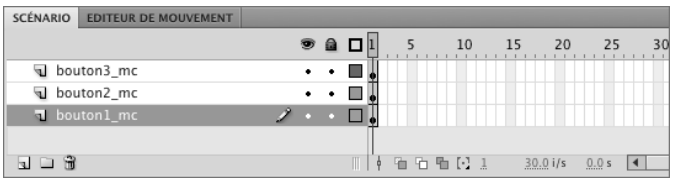
Exemples > ch13_systemesNavigation_2.fla
Exemples > ch13_systemesNavigation_2b.fla

Dans le document ch13_systemesNavigation_2.fla, nous pouvons voir une interface où des liens, en forme de poids, sont positionnés au-dessus de la partie gauche du décor. Un MovieClip canalise ces objets et porte le nom d'occurrence `menu_mc` (voir Figure 13.6).

Figure 13.6
Aperçu de la scène principale.

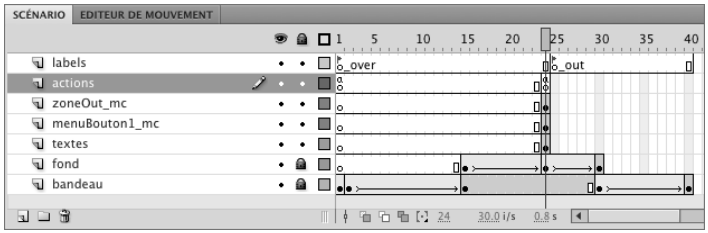
À l'intérieur de ce menu, trois symboles de type `MovieClip`, respectivement nommés `bouton1_mc`, `bouton2_mc` et `bouton3_mc` sont répartis distinctement vers les calques (voir Figure 13.7).

Figure 13.7
Scénario à l'intérieur du menu.



À l'intérieur du symbole `bouton1_mc`, nous accédons maintenant à une animation où la surface du lien s'étend jusqu'à dessiner une fenêtre. Cette animation prend fin à l'image 24 où la fenêtre est totalement déployée. Le reste de l'animation représente la fermeture de cette fenêtre. La lecture de cette animation est contrôlée par des actions. Un `stop`, placé à l'image 1 et 24, interrompt la lecture. C'est uniquement, lorsque la tête de lecture atteint l'image 24, que de nouveaux boutons apparaissent à l'écran (voir Figure 13.8).

Figure 13.8
Scénario à l'intérieur du bouton 1.

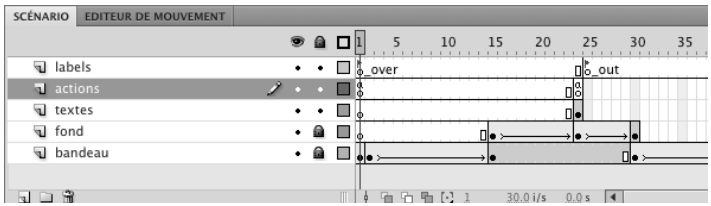


Dans la fenêtre de scénario du `bouton 1`, un autre objet apparaît également à l'image 24, un movieClip transparent nommé `zoneOut_mc` et de propriété alpha à 0 %. C'est lui qui permet d'activer la fermeture de la fenêtre, lorsque le pointeur le dépasse.

Afin d'activer l'ouverture animée de cette fenêtre, nous avons placé des actions sur la scène principale, en direction de l'objet `bouton1_mc` lui-même. Pour que les liens situés localement à l'image 24 du symbole `bouton1_mc` soient actifs, nous avons placé d'autres actions directement dans ce scénario, à l'image 24.

Les autres MovieClip boutons de ce document ne contiennent pas de liens isolés. Ils ont été mis en forme uniquement pour illustrer le mécanisme, dans un système plus global, avec plusieurs entrées de menu. Leur scénario ne possède pas les calques `menuBouton1_mc` ni `zoneOut_mc` (voir Figure 13.9). Mais le mécanisme adopté pour le premier bouton pourrait y être reproduit.

Figure 13.9
Scénario à l'intérieur des boutons 2 et 3.



Les actions du calque de la scène principale sont les suivantes :

```
//---------------------- initialisation
var boutonActif:String;

//---------------------- actions
//over
menu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
function over (evt:MouseEvent) {
   if (evt.target.currentFrame==1) {
      evt.target.gotoAndPlay("_over");
   }
```

```
}
//down
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,down);
function down (evt:MouseEvent) {
   boutonActif=evt.target.name;
   //
   if (evt.target.name=="bouton1_mc") {
      trace("action 1");
   }
   if (evt.target.name=="bouton2_mc") {
      trace("action 2");
   }
   if (evt.target.name=="bouton3_mc") {
      trace("action 3");
   }
}
//out
menu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
   if (evt.target.name!="bouton1_mc") {
      if (evt.target.currentFrame==24) {
         evt.target.gotoAndPlay("_out");
      }
   }
}
```

Les actions localisées dans le symbole `bouton1_mc`, à l'image 24, sont les suivantes :

```
stop();
//
menuBouton1_mc.addEventListener(MouseEvent.CLICK,actionsMenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
   if (evt.target.name=="lien1_mc") {
      trace("action 1 du menu bouton 1");
   }
   if (evt.target.name=="lien2_mc") {
      trace("action 2 du menu bouton 1");
   }
   if (evt.target.name=="lien3_mc") {
      trace("action 3 du menu bouton 1");
   }
}
//
zoneOut_mc.addEventListener(MouseEvent.MOUSE_OUT,sortirBouton1);
function sortirBouton1 (evt:MouseEvent) {
   play();
}
```

Dans la fenêtre d'actions de la scène principale, nous associons une fonction sur le conteneur `menu_mc` qui accueille les trois boutons :

```
//---------------------- actions
//over
menu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
function over (evt:MouseEvent) {
   if (evt.target.currentFrame==1) {
      evt.target.gotoAndPlay("_over");
   }
}
```

Dans cette fonction, nous activons la lecture du scénario du bouton cliqué, uniquement si l'image courante du bouton cliqué est l'image 1. Ceci évite que l'animation soit relancée intempestivement lorsque l'utilisateur navigue au sein de la fenêtre, si elle est déployée.

Plus bas, dans la fonction associée à l'événement `MOUSE_DOWN`, nous reprenons le même type d'actions que celles présentées dans la section précédente.

En revanche, dans le gestionnaire `MOUSE_OUT`, nous précisons que l'action de restauration de l'état du bouton ne peut avoir lieu que si le nom du bouton actif est différent de celui qui contient d'autres liens. Autrement dit, l'action de restauration n'apparaît que pour les liens simples. L'action de fermeture de la fenêtre, dans notre exemple, est contrôlée directement à l'image 24, dans le scénario du MovieClip `bouton` :

```
//out
menu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
   if (evt.target.name!="bouton1_mc") {
      if (evt.target.currentFrame==24) {
         evt.target.gotoAndPlay("_out");
      }
   }
}
```

Dans le scénario du MovieClip, à l'image 24, nous affectons d'abord un écouteur sur l'enveloppe `menuBouton1_mc` qui rassemble les trois sous-entrées locales. Pour chaque lien de ce conteneur, nous définissons une action `trace` spécifique :

```
menuBouton1_mc.addEventListener(MouseEvent.CLICK,actionsMenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
   if (evt.target.name=="lien1_mc") {
      trace("action 1 du menu bouton 1");
   }
   if (evt.target.name=="lien2_mc") {
      trace("action 2 du menu bouton 1");
   }
   if (evt.target.name=="lien3_mc") {
      trace("action 3 du menu bouton 1");
   }
}
```

Il n'y a pas de contradiction avec le conteneur principal, car ces objets sont situés au premier plan, et le gestionnaire cible bien un conteneur différent du conteneur principal.

Plus bas, une autre action est ajoutée et contrôle la fermeture de la fenêtre :

```
//
zoneOut_mc.addEventListener(MouseEvent.MOUSE_OUT,sortirBouton1);
function sortirBouton1 (evt:MouseEvent) {
   play();
}
```

L'action cible tout simplement la forme transparente située au pourtour de la fenêtre. Lorsque cette zone apparaît dans la scène, le pointeur est déjà sur la partie centrale. L'événement `MOUSE_OUT` permet donc d'activer la fermeture animée de la fenêtre, uniquement si l'utilisateur sort de la zone transparente, par l'extérieur.

Des instructions de neutralisation de MovieClip et de boutons sont proposées en fin de chapitre. Ils offrent une autre alternative à la gestion de contenus imbriqués.

Dans le document "ch13_systemesNavigation_2b.fla", une déclinaison de cet exemple est proposée, à partir de l'instruction `switch`. Ce qui donne, pour la fonction `down` de la scène principale :

```
//down
function down (evt:MouseEvent) {
   boutonActif=evt.target.name;
   //
   switch (evt.target.name) {
      case "bouton1_mc":
         trace("action 1");
         break;
      case "bouton2_mc":
         trace("action 2");
         break;
      case "bouton3_mc":
         trace("action 3");
         break;
      default:
         trace("ni 0, 1, ou 2");
   }
}
```

Pour la fonction placée à l'intérieur du symbole :

```
menuBouton1_mc.addEventListener(MouseEvent.CLICK,actionsMenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
   switch (evt.target.name) {
      case "lien1_mc":
         trace("action 1 du menu bouton 1");
         break;
      case "lien2_mc":
         trace("action 2 du menu bouton 1");
         break;
      case "lien3_mc":
         trace("action 3 du menu bouton 1");
         break;
      default:
         trace("aucun");
   }
}
```

Méthode avec les actions sur la scène principale

Une déclinaison du précédent document est proposée dans le fichier "ch13_systemesNavigation_2c.fla". Ce document prend en charge la gestion des liens imbriqués en plaçant le code directement sur la scène principale, ce qui rend plus facile la liaison des données entre fonctions et simplifie, entre autres, la récupération de valeurs, l'activation d'autres fonctions, la centralisation du code et sa maintenance.

Exemples > ch13_systemesNavigation_2c.fla

Dans ce document, nous observons que l'intérieur du MovieClip `bouton1_mc` affiche déjà les objets `zoneOut_mc` et `menuBouton1_mc`, dès la première image (voir Figure 13.10).

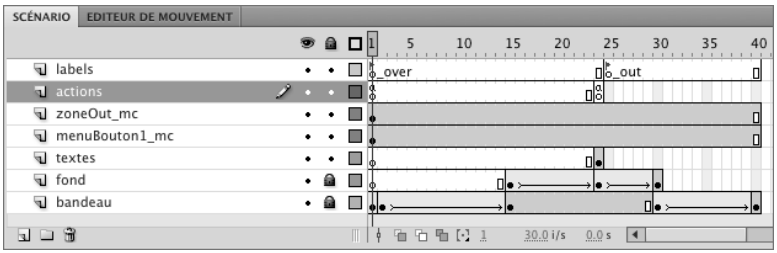


Figure 13.10
Scénario à l'intérieur du bouton 1.

D'autre part, aucune action spécifique, en dehors des stops, n'apparaît non plus à l'image 24. Toutes les actions sont centralisées à l'image 1 du scénario de la scène principale.

Dans le code de la scène principale, nous pouvons lire ceci :

```
//---------------------- initialisation
var boutonActif:String;

//---------------------- actions
//over
menu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
function over (evt:MouseEvent) {
   if (evt.target.currentFrame==1) {
      evt.target.gotoAndPlay("_over");
   }
}
//down
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,down);
function down (evt:MouseEvent) {
   boutonActif=evt.target.name;
   //
   if (evt.target.name=="bouton1_mc") {
      trace("action 1");
   }
   if (evt.target.name=="bouton2_mc") {
      trace("action 2");
   }
   if (evt.target.name=="bouton3_mc") {
      trace("action 3");
   }
}
//out
menu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
   if (evt.target.name!="bouton1_mc") {
      if (evt.target.currentFrame==24) {
         evt.target.gotoAndPlay("_out");
      }
```

```
    }
}

// liens 1, 2 et 3, dans bouton 1
menu_mc.bouton1_mc.menuBouton1_mc.addEventListener(MouseEvent.CLICK,actions-
➥MenuBouton1);
function actionsMenuBouton1 (evt:MouseEvent) {
    if (evt.target.name=="lien1_mc") {
        trace("action 1 du menu bouton 1");
    }
    if (evt.target.name=="lien2_mc") {
        trace("action 2 du menu bouton 1");
    }
    if (evt.target.name=="lien3_mc") {
        trace("action 3 du menu bouton 1");
    }
}
menu_mc.bouton1_mc.menuBouton1_mc.visible=false;

// sortie bouton 1
menu_mc.bouton1_mc.zoneOut_mc.addEventListener(MouseEvent.MOUSE_OUT,sortirBouton1);
function sortirBouton1 (evt:MouseEvent) {
    menu_mc.bouton1_mc.play();
}
menu_mc.bouton1_mc.zoneOut_mc.visible=false;

// gestion de l'affichage des liens dans le bouton 1
addEventListener(Event.ENTER_FRAME,afficherLiens);
function afficherLiens (evt:Event) {
    if (menu_mc.bouton1_mc.currentFrame==24) {
        menu_mc.bouton1_mc.menuBouton1_mc.visible=true;
        menu_mc.bouton1_mc.zoneOut_mc.visible=true;
    } else {
        menu_mc.bouton1_mc.menuBouton1_mc.visible=false;
        menu_mc.bouton1_mc.zoneOut_mc.visible=false;
    }
}
```

Dans ce document, nous avons déplacé les actions de l'image 24, du symbole MovieClip `bouton1_mc`, à l'intérieur de la fenêtre actions de la scène principale. Afin que ces actions s'exécutent correctement, nous avons placé les objets ciblés sur la première image du scénario `bouton1`. Ainsi, ils peuvent être identifiés par la tête de lecture, au moment où les actions de la scène principale sont interprétées. Mais, pour que ces objets n'apparaissent pas au démarrage, nous avons ajouté des contrôles de visibilité, en fonction de l'image active de ce scénario.

Le même document, nommé "ch13_systemesNavigation_2d.fla", décliné avec des instructions `switch`, est également disponible dans les fichiers des exemples du livre.

À retenir

- Des liens peuvent être placés localement dans un objet déjà interactif. Pour cela, ils doivent être placés au premier plan et leurs actions doivent être situées localement au niveau du lien.
- Une approche alternative existe. Il est possible de placer les actions sur la scène principale, mais avec des conditions qui neutralisent d'abord l'affichage de l'objet par défaut.

Console de navigation en miniature

Si vous utilisez la fenêtre de navigation de Photoshop ou d'Illustrator, vous êtes sûrement familier avec la navigation dans une fenêtre réduite. Une fenêtre de navigation réduite reprend, en miniature, un aperçu de l'ensemble de la scène et permet à l'utilisateur de s'y mouvoir en Y déplaçant un simple rectangle. Ce rectangle matérialise la zone visible à l'écran.

Dans cette section, nous allons voir comment mettre en place ce dispositif de navigation. Nous utilisons pour cela une carte du monde, agrandie à l'échelle de 200 % par rapport à sa taille initiale. Cette carte déborde donc largement de la zone visible par l'utilisateur. C'est en déplaçant le rectangle rouge de la fenêtre de navigation, que l'on fera bouger le contenu, pour accéder aux informations que celui-ci propose (voir Figure 13.11).

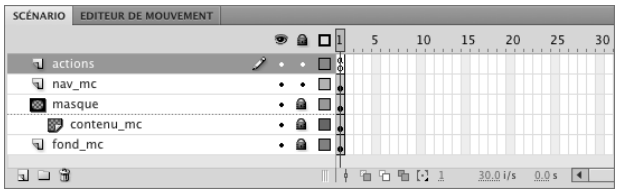
Figure 13.11
Aperçu du document publié.

Exemples > ch13_systemesNavigation_3.fla

Dans le document "ch13_systemesNavigation_3.fla", sur la scène principale (voir Figure 13.12), au-dessus du calque `fond_mc`, nous visualisons le calque `contenu_mc`

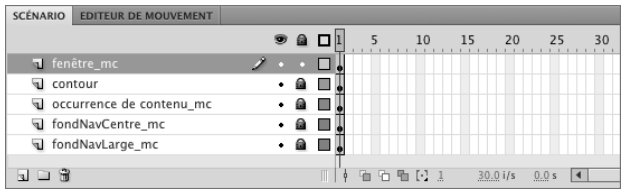
masqué par un rectangle de dimensions équivalentes à celles de la scène, moins la barre d'informations.

Figure 13.12
Fenêtre de scénario de la scène principale.



À l'intérieur du calque `nav_mc`, figurent plusieurs symboles, dont le rectangle rouge que nous utilisons comme outil de déplacement et qui porte le nom d'occurrence `fenetre_mc` (voir Figure 13.13). En dessous, une copie de l'occurrence `contenu_mc` est affichée en proportions réduites, elle illustre l'ensemble du contenu disponible à la navigation, et situe la position courante du rectangle par rapport au contenu affiché dans cette zone. En arrière plan de ces calques, le symbole `fondNavCentre_mc` représente la zone limite de déplacement dont nous reprenons les coordonnées dans le code. L'objet `fondNavLarge_mc` et le calque contour servent uniquement la décoration.

Figure 13.13
Fenêtre de scénario du symbole `nav_mc`.



Dans la fenêtre Actions de la scène principale, nous pouvons lire le code suivant :

```
//------------------------ initialisation
var origineX:Number=nav_mc.fondNavCentre_mc.x;
var origineY:Number=nav_mc.fondNavCentre_mc.y;
var largeurRect:Number=(nav_mc.fondNavCentre_mc.width)-
➥(nav_mc.fenêtre_mc.width);
var hauteurRect:Number=(nav_mc.fondNavCentre_mc.height)-
➥(nav_mc.fenêtre_mc.height);
var zoneDeplacement:Rectangle=new Rectangle(origineX, origineY, largeurRect,
➥hauteurRect);
//
var echelleMouvement:Number=(contenu_mc.width/nav_mc.fondNavCentre_mc.width);

//------------------------ actions
// déplacement du rectangle
nav_mc.fenêtre_mc.addEventListener(MouseEvent.MOUSE_DOWN,deplacerRectangle);
function deplacerRectangle (evt:MouseEvent) {
   addEventListener(Event.ENTER_FRAME, deplacerContenu);
   nav_mc.fenêtre_mc.startDrag(false,zoneDeplacement);
}
addEventListener(MouseEvent.MOUSE_UP,relacherRectangle);
function relacherRectangle (evt:MouseEvent) {
```

```
    removeEventListener(Event.ENTER_FRAME, deplacerContenu);
    stopDrag();
}

// déplacement du contenu
function deplacerContenu (evt:Event) {
    contenu_mc.x=nav_mc.fenêtre_mc.x*-echelleMouvement;
    contenu_mc.y=nav_mc.fenêtre_mc.y*-echelleMouvement;
}
```

Le programme se déroule en trois étapes. Nous définissons d'abord la zone de déplacement. Nous activons ensuite le mouvement. Puis nous lions le contenu de la scène à l'outil déployé et mobile.

Tout d'abord, dans la partie initialisation, nous spécifions, au travers des quatre premières variables, les limites de la zone de déplacement :

```
var origineX:Number=nav_mc.fondNavCentre_mc.x;
var origineY:Number=nav_mc.fondNavCentre_mc.y;
var largeurRect:Number=(nav_mc.fondNavCentre_mc.width)-
➥(nav_mc.fenêtre_mc.width);
var hauteurRect:Number=(nav_mc.fondNavCentre_mc.height)-
➥(nav_mc.fenêtre_mc.height);
var zoneDeplacement:Rectangle=new Rectangle(origineX, origineY, largeurRect,
➥hauteurRect);
```

Nous reprenons la position courante et les dimensions de la zone de déplacement contenue dans le symbole `nav_mc`, qui porte le nom d'occurrence `fondNavCentre_mc`. Mais pour définir la largeur utile de ce déplacement, nous devons également soustraire la largeur et la hauteur du rectangle mobile, comme nous l'avions fait pour l'ascenseur au Chapitre 2, avec les interpolations `TweenMax`.

Plus bas, nous définissons une variable pour déterminer le cœfficient de vitesse du mouvement. Il s'agit du rapport d'échelle entre la taille du contenu de la scène principale et les dimensions de la zone de déplacement :

```
var echelleMouvement:Number=(contenu_mc.width/nav_mc.fondNavCentre_mc.width);
```

Dans la deuxième partie, nous activons en premier la fonction qui recalcule la position des objets (deplacerContenu). Puis, à la ligne, nous activons le contrôle du mouvement de la forme rectangulaire rouge et introduisons, en paramètre de la méthode `startDrag`, la référence au rectangle préalablement désigné :

```
//------------------------ actions
// déplacement du rectangle
nav_mc.fenêtre_mc.addEventListener(MouseEvent.MOUSE_DOWN,deplacerRectangle);
function deplacerRectangle (evt:MouseEvent) {
    addEventListener(Event.ENTER_FRAME, deplacerContenu);
    nav_mc.fenêtre_mc.startDrag(false,zoneDeplacement);
}
addEventListener(MouseEvent.MOUSE_UP,relacherRectangle);
function relacherRectangle (evt:MouseEvent) {
    removeEventListener(Event.ENTER_FRAME, deplacerContenu);
    stopDrag();
}
```

Cette forme est entièrement cliquable car nous y avons introduit un symbole qui couvre toute la surface mais avec un alpha à 0 %. Le paramètre `false` indique que l'objet ne se repositionne pas en fonction du clic souris, à l'inverse, l'usage de `true` aurait centré le rectangle en fonction de ce clic.

La fonction `relacherRectangle` interrompt le déplacement du symbole et la fonction `deplacerContenu`, gérée par un `ENTER_FRAME`.

Enfin, nous ajoutons un écouteur qui exécute le repositionnement en reprenant le cœfficient calculé en amont, et repositionne le symbole `contenu_mc`, en X et en Y, en fonction de la position du rectangle rouge :

```
// déplacement du contenu
addEventListener(Event.ENTER_FRAME, deplacerContenu);
function deplacerContenu (evt:Event) {
   contenu_mc.x=nav_mc.fenêtre_mc.x*-echelleMouvement;
   contenu_mc.y=nav_mc.fenêtre_mc.y*-echelleMouvement;
}
```

À retenir

- *La partie active d'un bouton ou de tout type de symbole doit toujours être matérialisée. Pour déplacer une forme qui n'affiche pas de fond, nous plaçons, à l'intérieur du symbole utilisé pour le déplacement, une forme pleine mais transparente.*
- *Pour déterminer le rapport d'échelle des dimensions entre un contenu et la scène courante, il suffit de diviser les dimensions de la première par la seconde.*

Menu déroulant à repositionnement automatique

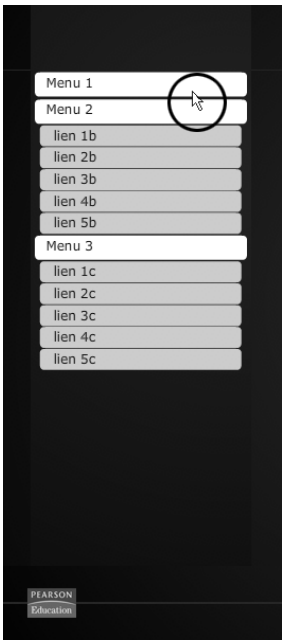
La création d'un menu déroulant ne pose pas de difficulté particulière dès lors que l'on sait utiliser les masques, le scénario, quelques transitions animées et les conditions en ActionScript. Mais, si nous voulons que les menus s'affichent en colonne, par exemple, les uns au-dessus des autres, le mécanisme peut se compliquer. En effet, pour ouvrir un menu et afficher ses sous-entrées, nous devons alors chasser les menus du dessous, encore refermés, à mesure que les sous-entrées du menu précédent descendent. Et inversement à la fermeture.

Pour réaliser ce dispositif, nous créons des sous-menus directement à l'intérieur des MovieClip boutons qui représentent les entrées principales de chaque menu. Puis, nous utilisons des transitions de type `TweenMax` pour activer le défilement des sous-menus, sur clic de l'utilisateur.

Nous gérons aussi le repositionnement des entrées principales de menu, avec un gestionnaire de type `ENTER_FRAME`. C'est dans ce gestionnaire que nous indiquons, à chaque entrée principale, de se positionner en fonction de la position Y du dernier élément du menu précédent. Ainsi, quelle que soit la position des sous-entrées de chaque menu, les entrées principales et leurs sous-menus respectifs, enfants, suivent les mouvements de chaque sous-menu.

Dans cette section, nous utilisons un menu à trois entrées, qui affichent, chacune, cinq sous-entrées. Les symboles utilisés sont tous différents. Vous pouvez donc rapidement personnaliser le fichier pour vous l'approprier et y placer de vraies fonctionnalités (voir Figure 13.14).

Figure 13.14
Aperçu du document publié.

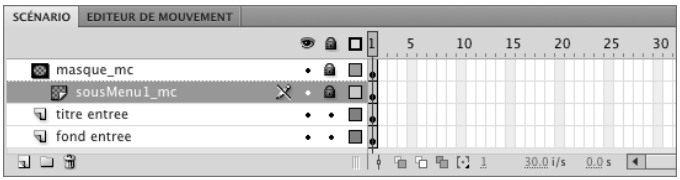


Exemples > ch13_systemesNavigation_4.fla

Dans la scène principale du document "ch13_systemesNavigation_4.fla", nous pouvons voir un symbole `menu_mc` qui contient trois entrées principales de menu, respectivement nommées `entree1_mc`, `entree2_mc` et `entree3_mc`.

Dans le scénario de chaque bouton `MovieClip` d'entrée, nous remarquons la présence d'un masque qui rend invisibles les sous-entrées appelées par les scripts. Si nous déverrouillons les calques, nous constatons que ces sous-entrées sont positionnées, au pixel près, au-dessus de la zone d'affichage, matérialisée par le masque (voir Figures 13.15 et 13.16). Dans les scripts, nous déterminerons le positionnement des éléments en fonction de leur position courante. Il est donc important que les éléments ne soient pas placés n'importe où.

Figure 13.15
Aperçu du scénario de la scène d'un sous-menu



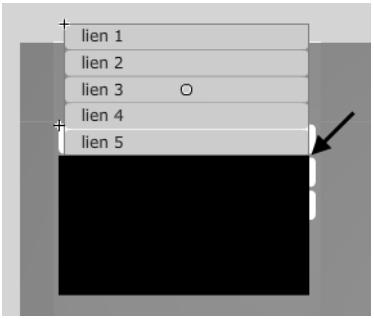


Figure 13.16
Aperçu du sous-menu et de son masque.

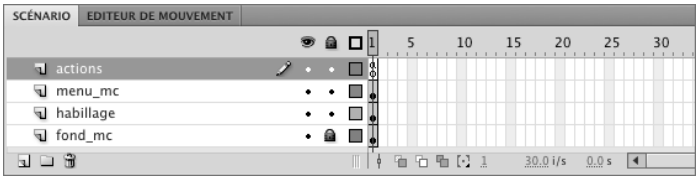


Figure 13.17
Aperçu du scénario de la scène principale.

Dans le scénario de la scène principale (voir Figure 13.17), au-dessus des calques de décoration et du menu, nous accédons aux actions qui affichent le code suivant :

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

var initEntree1Y:Number=menu_mc.entree1_mc.y;
var initEntree2Y:Number=menu_mc.entree2_mc.y;
var initEntree3Y:Number=menu_mc.entree3_mc.y;

var initSousMenu1Y:Number=menu_mc.entree1_mc.sousMenu_mc.y;
var initSousMenu2Y:Number=menu_mc.entree2_mc.sousMenu_mc.y;
var initSousMenu3Y:Number=menu_mc.entree3_mc.sousMenu_mc.y;

//-------------------- actions
// activation des sous-menus
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,ouvrirMenu);
function ouvrirMenu(evt:MouseEvent) {
   if (evt.target.name=="entree1_mc") {
      if (menu_mc.entree1_mc.sousMenu_mc.y<=initSousMenu1Y) {
         TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
         y:initSousMenu1Y + menu_mc.entree1_mc.sousMenu_mc.height,
         ease:Strong.easeInOut,
         onStartListener : placeEcouteur,
         onCompleteListener :retireEcouteur,
         overwrite:3
         });
      } else {
```

```
            TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
            });
         }
      }
      if (evt.target.name=="entree2_mc") {
         if (menu_mc.entree2_mc.sousMenu_mc.y<=initSousMenu2Y) {
            TweenMax.to(menu_mc.entree2_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y + menu_mc.entree2_mc.sousMenu_mc.height,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
         });
         } else {
            TweenMax.to(menu_mc.entree2_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
            });
         }
      }
      if (evt.target.name=="entree3_mc") {
         if (menu_mc.entree3_mc.sousMenu_mc.y<=initSousMenu3Y) {
            TweenMax.to(menu_mc.entree3_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y + menu_mc.entree3_mc.sousMenu_mc.height,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
            });
         } else {
            TweenMax.to(menu_mc.entree3_mc.sousMenu_mc, 0.5, {
            y:initSousMenu1Y,
            ease:Strong.easeInOut,
            onStartListener : placeEcouteur,
            onCompleteListener :retireEcouteur,
            overwrite:3
            });
         }
      }
trace(evt.target.name);
}

// suivi des entrées
function placeEcouteur(Evt:TweenEvent){
   addEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}
function retireEcouteur(Evt:TweenEvent){
   removeEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}
```

```
function replacerEntreesMenu(evt:Event) {
   trace("fonction active")
   menu_mc.entree2_mc.y=initEntree2Y+(menu_mc.entree1_mc.sousMenu_mc.
   ➥ y-initSousMenu1Y);
   menu_mc.entree3_mc.y=initEntree3Y+(menu_mc.entree1_mc.sousMenu_mc.
   ➥ y-initSousMenu1Y)+(menu_mc.entree2_mc.sousMenu_mc.y-initSousMenu2Y);
}
```

Notre code se définit en trois parties. La première importe les classes et enregistre la position courante des éléments. La deuxième active le déroulement des sous-menus. La troisième et dernière partie repositionne les entrées en fonction de la position courante des sous-menus.

Dans la première partie, l'initialisation, après l'importation des classes, nous stockons les valeurs de position en Y, de chaque sous-menu et de chaque entrée de menu :

```
var initEntree1Y:Number=menu_mc.entree1_mc.y;
var initEntree2Y:Number=menu_mc.entree2_mc.y;
var initEntree3Y:Number=menu_mc.entree3_mc.y;

var initSousMenu1Y:Number=menu_mc.entree1_mc.sousMenu_mc.y;
var initSousMenu2Y:Number=menu_mc.entree2_mc.sousMenu_mc.y;
var initSousMenu3Y:Number=menu_mc.entree3_mc.sousMenu_mc.y;
```

Dans les actions, lorsque l'utilisateur active une des entrées du menu (`evt.target`), nous ajoutons des actions selon le nom de l'entrée activée :

```
//-------------------- actions
// activation des sous-menus
menu_mc.addEventListener(MouseEvent.MOUSE_DOWN,ouvrirMenu);
function ouvrirMenu(evt:MouseEvent) {
   if (evt.target.name=="entree1_mc") {
      if (menu_mc.entree1_mc.sousMenu_mc.y<=initSousMenu1Y) {
         TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
         y:initSousMenu1Y + menu_mc.entree1_mc.sousMenu_mc.height,
         ease:Strong.easeInOut,
         onStartListener : placeEcouteur,
         onCompleteListener :retireEcouteur,
         overwrite:3
         });
      } else {
         TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
         y:initSousMenu1Y,
         ease:Strong.easeInOut,
         onStartListener : placeEcouteur,
         onCompleteListener :retireEcouteur,
         overwrite:3
         });
      }
   }
// déclinaison des instructions pour les entrées de menu 2 et 3
trace(evt.target.name);
}
```

À l'intérieur de chaque condition, une autre condition vérifie si la position courante du sous-menu concerné est en position affichée ou repliée. La position repliée est définie en reprenant une des valeurs enregistrées en amont. Si la valeur enregistrée est identique à la position actuelle, alors, le sous-menu n'a pas bougé. Il est donc ramassé. Ceci s'inverse dès que le sous-menu est activé, celui-ci est déplacé et sa position ne répond plus à la condition qui alors exécute une autre instruction.

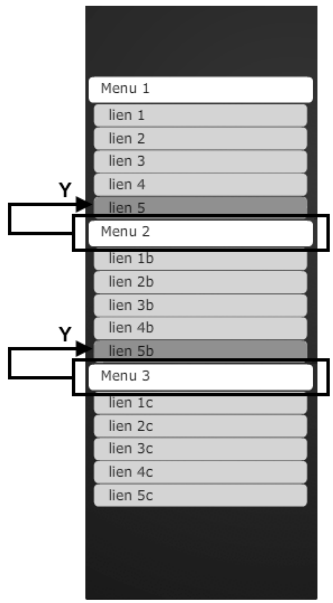
La position des sous-menus, lorsqu'ils sont affichés, augmente de l'équivalent de leur hauteur respective (d'où la nécessité de les placer précisément à la limite de la zone d'affichage, dans chaque entrée de menu) :

```
TweenMax.to(menu_mc.entree1_mc.sousMenu_mc, 0.5, {
        y:initSousMenu1Y + menu_mc.entree1_mc.sousMenu_mc.height,
        ease:Strong.easeInOut,
        onStartListener : placeEcouteur,
        onCompleteListener :retireEcouteur,
        overwrite:3
        });
```

Si la condition est vérifiée, si le menu est fermé donc, une transition de type `TweenMax` le déplace de l'équivalent de sa hauteur à laquelle nous ajoutons (par sécurité), sa position enregistrée. Sinon, il le ramène uniquement à sa position préalablement enregistrée (`initSousMenu1Y`), donc, en position replié (voir Figure 13.18).

Figure 13.18

Mécanisme de repositionnement des entrées principales du menu.



Au sein des interpolations, d'autres propriétés sont utilisées. Chacune se termine en invoquant directement, grâce aux propriétés d'écouteur `onStartListener` et `onCompleteListener`, abordées au Chapitre 2, deux autres fonctions. La première (`placeEcouteur`) active le repositionnement continu des objets au démarrage de l'interpolation. La seconde (`retire-Ecouteur`) interrompt cette fonction une fois l'interpolation achevée :

```
// suivi des entrées
function placeEcouteur(Evt:TweenEvent){
   addEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}
function retireEcouteur(Evt:TweenEvent){
   removeEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
}
```

Il serait dommage en effet de conserver active la fonction `replacerEntreeMenu` et de continuer de calculer le repositionnement même lorsqu'aucune entrée du menu n'est cliquée. Cette fonction qui est associée à l'événement `ENTER_FRAME` sollicite d'importantes ressources. En désactivant cette fonction lorsqu'elle n'est plus usitée, nous optimisons celles de l'utilisateur.

Un dernier paramètre (`overwrite`) permet à chaque interpolation de reprendre la main si l'utilisateur activait une entrée du menu alors qu'une autre interpolation est déjà en cours d'exécution. L'interpolation en cours est dans ce cas interrompue et permet à la nouvelle de démarrer. Cette propriété est utilisée pour éviter les conflits lorsque plusieurs interpolations ciblent les mêmes objets, ce qui est notre cas dans ce dispositif de navigation.

Propriété `overwrite` pour `TweenMax`

La propriété overwrite contrôle le comportement des autres interpolations si elles interviennent sur le même objet. La valeur attendue est un chiffre entier compris entre 0 et 3 :

- **0 (NONE).** Aucune interpolation n'est neutralisée. Cette option risque de créer des conflits si d'autres interpolations affectent simultanément le même objet avec les mêmes propriétés.
- **1 (ALL).** Cette valeur est celle activée par défaut sauf si la méthode `OverwriteManager.init()` est invoquée. Toutes les interpolations qui affectent le même objet sont neutralisées, qu'elles soient actives ou non. Par exemple :

```
TweenMax.to(mc, 1, {x:100, y:200});
TweenMax.to(mc, 1, {x:300, delay:2, overwrite:1});
   // cette interpolation écrase la précédente.
```

- **2 (AUTO).** Cette valeur est celle activée par défaut lorsque la méthode `OverwriteManager.init()` est invoquée. La valeur 2 écrase uniquement les propriétés communes avec les interpolations en cours sur les mêmes objets. Par exemple :

```
TweenMax.to(mc, 1, {x:100, y:200});
TweenMax.to(mc, 1, {x:300, overwrite:2});
   // seule la propriété x de l'interpolation précédente est écrasée.
```

- **3 (CONCURRENT).** Interrompt toutes les interpolations, uniquement ayant cours, et qui affectent les mêmes objets. Par exemple :

```
TweenMax.to(mc, 1, {x:100, y:200});
TweenMax.to(mc, 1, {x:300, delay:2, overwrite:3});
   // cette interpolation n'écrase pas la précédente, car elle intervient
   // avec un délai de 2 secondes après l'autre,
   // bien qu'elles affectent le même objet mc.
```

Les conditions sont déclinées pour chaque entrée. Une action `trace` permet également de visualiser que le ciblage des éléments fonctionne bien. Cette action vous permet aussi de personnaliser facilement ce développement en Y ajoutant vos propres actions, en rapport avec chaque lien. Il suffit alors de reprendre simplement son nom pour un ciblage dynamique, comme vu précédemment.

Dans la troisième partie, un gestionnaire `ENTER_FRAME` recalcule en la position de chaque entrée de menu :

```
addEventListener(Event.ENTER_FRAME,replacerEntreesMenu);
function replacerEntreesMenu(evt:Event) {
   menu_mc.entree2_mc.y=initEntree2Y+(menu_mc.entree1_mc.sousMenu_mc.
   ➥ y-initSousMenu1Y);
   menu_mc.entree3_mc.y=initEntree3Y+(menu_mc.entree1_mc.sousMenu_mc.
   ➥ y-initSousMenu1Y)+(menu_mc.entree2_mc.sousMenu_mc.y-initSousMenu2Y);
}
```

La première instruction détermine la position Y de la deuxième entrée. La première, demeurant immobile, il n'est donc pas utile de la calculer.

Dans cette première équation, nous spécifions que la position en Y de l'entrée 2 doit dépendre de la position en Y du bas du sous-menu de l'entrée 1. Le bas du sous-menu est calculé en additionnant sa hauteur à sa position courante en Y (voir Figure 13.18).

Le principe est le même pour l'entrée 3, à la différence que nous décalons le calcul d'une ligne, en ajoutant la position courante en Y du deuxième sous-menu.

À retenir

- *Pour réaliser des menus qui se repositionnent les uns par rapport aux autres, dynamiquement, nous devons utiliser un gestionnaire* `ENTER_FRAME` *qui définit la position de chaque entrée de menu par rapport à la position courante du sous-menu qui la précède.*
- *Le positionnement des sous-entrées de menu est important, dans la construction du document, car cela détermine leur visibilité lorsqu'elles sont déployées.*

Activer et désactiver les boutons et les MovieClip

Dans bien des cas, il est nécessaire de désactiver des liens et des MovieClip initialement associés à un écouteur d'événement. Lorsque vous chargez une rubrique par-dessus une interface déjà riche en fonctionnalités, par exemple, il peut être conflictuel de garder actifs certains éléments laissés en arrière-plan.

Dans cette section, nous présentons les différentes méthodes employées pour neutraliser un bouton ou un MovieClip. Mais aussi pour matérialiser le pointeur en forme de main sur les MovieClip et ainsi améliorer l'ergonomie de certaines mises en scène.

- Pour désactiver uniquement les états des boutons (haut, dessus et abaissé), utilisez l'instruction : `nomDuBouton.enabled=false;`
- Pour désactiver l'interactivité d'un bouton ou d'un MovieClip, utilisez : `nomDuBouton-OuClip.mouseEnabled=false;`

- Pour désactiver l'interactivité d'un bouton ou d'un MovieClip ainsi que tout élément enfant de chaque objet géré avec evt.target (précision valable pour les clips ou les Sprites) : `nomDuBoutonOuClip.mouseChildren=false;`
- Pour activer uniquement l'affichage de la main, au survol du pointeur, sur un MovieClip, choisissez l'instruction : `nomDuClip.buttonMode = true;`
- Pour désactiver l'affichage de la main sur un symbole bouton, prenez l'instruction suivante : `nomDuBouton.useHandCursor=false;`

Vous pouvez aussi neutraliser les interactions avec l'une de ces méthodes :

- Masquer un bouton ou un MovieClip à l'aide de la propriété `visible`, passée sur `false`.
- Supprimer l'objet de la liste d'affichage avec la méthode `removeChild()`.
- Supprimer l'écouteur associé à l'objet bouton ou MovieClip avec la méthode `removeEventListener`.

Attention, ces techniques ne suppriment pas les actions éventuellement en cours d'exécution, elles rendent seulement les objets interactifs indisponibles pour le lancement de nouvelles actions.

Ciblage des boutons entre SWF imbriqués. *Il est souvent utile de pouvoir neutraliser une action ou réagir en fonction d'une action lancée dans un document SWF importé. Reportez-vous au Chapitre 8 pour en savoir plus sur les techniques de ciblage d'objets dans des contenus imbriqués.*

`Target` et `currentTarget`. *La définition des actions sur les symboles boutons et MovieClip se détermine aussi dans le cadre d'éléments contenus à l'intérieur d'un objet parent et ciblés avec les propriétés* `target` *et* `currentTarget`. *Reportez-vous également au Chapitre 5 pour en savoir plus à ce sujet.*

Synthèse

Dans ce chapitre, vous avez appris à développer des systèmes de navigation sophistiqués en y intégrant des boutons avec des états visités ou activés. Vous avez appris à transformer de simples liens en zone interfaçable et interactive sans conflit avec l'environnement initial. Vous avez également vu comment gérer la superposition dynamique de menus déroulants. Vous avez enfin appris à réaliser un dispositif de navigation global concentré dans une fenêtre réduite. Vous êtes en mesure de réaliser des interfaces souples et ergonomiques.