8 La vidéo interactive

Introduction

Tout l'intérêt de l'intégration de la vidéo dans Flash, qu'elle soit standard, composite ou en haute définition, est qu'en plus d'offrir un espace confortable pour son affichage, vous pouvez y associer des comportements. Dans ce chapitre, nous allons voir comment interagir avec la vidéo aussi bien à travers la navigation (sur actions de l'utilisateur) qu'à travers des événements (sur des actions exécutées par la vidéo durant son déroulement). En plus de toute cette interactivité, nous allons aborder la manière de réaliser une boucle et comment, dans le contexte de contenus SWF imbriqués, interrompre réellement le flux vidéo. À l'issue du chapitre, vous serez en mesure de créer des interfaces vidéo interactives riches, en ligne, mais aussi pour des systèmes hors ligne.

Dans ce chapitre, nous utilisons la bande démo de la société gKaster, généreusement mise à disposition pour cette présentation (**www.gKaster.com**).

Contrôles de base de la vidéo

Les premières actions nécessaires pour la gestion d'une vidéo sont les contrôles de lecture. Dans cette première section, nous abordons les actions qui permettent de lire, arrêter, accélérer et rembobiner un flux vidéo, mais aussi modifier le volume sonore d'une vidéo associée à un composant `FLVPlayBack`.

Exemples > ch8_videoInteractive_1.fla

Dans le document "ch8_videoInteractive_1.fla", sur la scène, un composant vidéo joue la bande démo de la société gKaster. Au-dessous, une console contient différents boutons qui contrôlent le flux vidéo (voir Figure 8.1). Dans cette console, chaque symbole est disposé sur un calque séparé et possède un nom d'occurrence lui permettant de recevoir une action. Le symbole `audio_mc`, lui, contient deux `MovieClip` qui affichent chacun un état activé ou non activé, pour le contrôle du son. Ils possèdent également leur propre nom d'occurrence.

Dans la fenêtre de scénario de la scène principale, au-dessus du calque `fond_mc`, apparaissent la console, le composant et un calque `actions` (voir Figure 8.2).

Figure 8.1
Aperçu après publication.

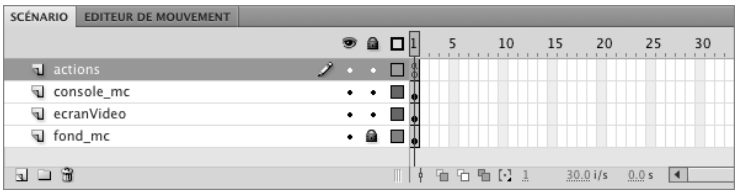


Figure 8.2
Aperçu du scénario de la scène principale.

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

```
// lecture automatique
ecranVideo.autoPlay=true;

// mise en cache
ecranVideo.bufferTime=3000;

// lire
console_mc.lire_btn.addEventListener(MouseEvent.CLICK,jouerVideo);
function jouerVideo (evt:MouseEvent) {
   ecranVideo.play();
}

// arrêter
console_mc.stop_btn.addEventListener(MouseEvent.CLICK,stopperVideo);
function stopperVideo (evt:MouseEvent) {
   ecranVideo.stop();
}

// suite
console_mc.suite_btn.addEventListener(MouseEvent.CLICK,suiteVideo);
function suiteVideo (evt:MouseEvent) {
   ecranVideo.seek(ecranVideo.playheadTime+2)
}
```

```
// retour
console_mc.retour_btn.addEventListener(MouseEvent.CLICK,retourVideo);
function retourVideo (evt:MouseEvent) {
   ecranVideo.seek(ecranVideo.playheadTime-2)
}

// Audio
console_mc.audio_mc.audioOff_mc.visible=false;
console_mc.audio_mc.audioOff_mc.buttonMode=true;

console_mc.audio_mc.addEventListener(MouseEvent.CLICK,audioVideo);
function audioVideo (evt:MouseEvent) {
   if (ecranVideo.volume>0) {
      ecranVideo.volume=0;
      evt.target.visible=false;
      console_mc.audio_mc.audioOff_mc.visible=true;
  } else {
      ecranVideo.volume=1;
      evt.target.visible=false;
      console_mc.audio_mc.audioOn_mc.visible=true;
   }
}
```

Voici le descriptif détaillé des différentes fonctionnalités rassemblées dans cette console.

Lecture automatique

La première ligne spécifie si le composant nommé `ecranVideo` doit jouer automatiquement dès l'affichage de l'animation (`true`) où s'il doit attendre une action de l'utilisateur (`false`). L'action est activée sur `true`. La vidéo joue donc automatiquement dès la publication :

```
// lecture automatique
ecranVideo.autoPlay=true;
```

Mise en cache

Une seconde action affecte directement le signal vidéo contrôlé par le composant vidéo. C'est la mise en cache. La propriété bufferTime permet d'indiquer le nombre de millisecondes à attendre avant d'exécuter la lecture du flux vidéo. Cette indication permet de réduire les risques de rupture de flux pour les connexions faibles. Ici, la valeur portée à 3 000 désigne une attente de trois secondes avant le démarrage de la vidéo :

```
// mise en cache
ecranVideo.bufferTime=3000;
```

La valeur assignée par défaut à la propriété `bufferTime` est 5 000. Elle ne peut pas être modifiée depuis l'Inspecteur de composants.

Lire la vidéo

Plus loin, un écouteur est attaché au symbole bouton `lire_btn` et indique de lire le flux vidéo avec l'action `play()` :

```
// lire
console_mc.lire_btn.addEventListener(MouseEvent.CLICK,jouerVideo);
function jouerVideo (evt:MouseEvent) {
   ecranVideo.play();
}
```

Arrêt de la vidéo

À la suite, un écouteur est attaché au symbole bouton `stop_btn` et indique de stopper la progression du flux avec l'action `stop()` :

```
// arrêter
console_mc.stop_btn.addEventListener(MouseEvent.CLICK,stopperVideo);
function stopperVideo (evt:MouseEvent) {
   ecranVideo.stop();
}
```

`stop` interrompt également la mise en cache. Pour arrêter et reprendre la lecture de la vidéo sans interrompre sa mise en cache, utilisez `pause()`.

Accélérer la vidéo

Il est possible d'accélérer la lecture de la vidéo en positionnant la tête de lecture de la vidéo à une image plus éloignée que l'image courante. Pour effectuer ce calcul, nous affectons d'abord la propriété `playheadTime` au composant vidéo pour permettre de capturer la position courante de la tête de lecture. Puis, nous augmentons la valeur, obtenue par cette propriété, de quelques images ou secondes. Nous passons ensuite directement cette valeur en paramètre de la méthode `seek()` qui permet de repositionner la tête de lecture dans la vidéo :

```
// suite
console_mc.suite_btn.addEventListener(MouseEvent.CLICK,suiteVideo);
function suiteVideo (evt:MouseEvent) {
   ecranVideo.seek(ecranVideo.playheadTime+2)
}
```

Les valeurs que nous passons en paramètre sont des nombres. La méthode `seek()` renvoie la tête de lecture à une position qui correspond toujours à l'image-clé suivante la plus proche dans le flux vidéo. La précision du ciblage dépend donc directement du nombre d'images-clés encodées dans le flux vidéo.

La vidéo que nous traitons a été encodée avec un nombre d'images-clés défini sur Automatique. Cette valeur, dans Adobe Media Encoder, spécifie en réalité un intervalle de deux secondes entre chaque image-clé. Pour obtenir une navigation plus précise que par pas de deux secondes, il convient donc de revenir sur la compression en modifiant cette valeur à un intervalle plus serré. Rappelez-vous cependant que plus un intervalle est serré, plus l'encodeur ajoute des images-clés. Or, chaque image-clé augmente le poids et, à poids égal, à taux

de compression égal, à débit égal, l'ajout d'images-clés réduit la place disponible pour encoder les autres images. Cela détériore donc l'ensemble de la vidéo. Un paradoxe que l'on peut résoudre très simplement : en augmentant les valeurs des réglages de compression et de débit.

Le pas d'incrémentation utilisé dans notre exemple est 2 (deux secondes). La tête de lecture est donc déplacée à l'image-clé la plus proche à deux secondes d'intervalle après la position courante du flux vidéo et reprend la lecture de la vidéo à partir de cette image.

Dans notre exemple, le repositionnement appelle donc toujours la deuxième image-clé située après la position courante de la tête de lecture et poursuit la lecture de la vidéo à partir de cette image, car c'est toujours l'image-clé suivante, la plus proche de l'image invoquée, qui est affichée.

Notez qu'une valeur de recherche élevée et une vidéo avec des plans assez longs peuvent contribuer à masquer le manque de précision induit par la méthode `seek()` et par un type de compression impliquant un faible nombre d'images-clés.

Rembobiner la vidéo

De la même manière que nous pouvons accélérer la vidéo, nous la rembobinons avec la méthode `seek()` et la propriété `playheadTime`. Mais, ici, nous retranchons la valeur à la position courante du flux vidéo de sorte à revenir deux seconde en arrière, à chaque clic sur le bouton retour :

```
// retour
console_mc.retour_btn.addEventListener(MouseEvent.CLICK,retourVideo);
function retourVideo (evt:MouseEvent) {
   ecranVideo.seek(ecranVideo.playheadTime-2)
}
```

Modifier le volume audio

La méthode `volume()` permet de modifier le volume sonore global de la vidéo. La valeur à passer en paramètre est 1 pour un volume normal et 0 pour un son muet. Les valeurs décimales intermédiaires permettent de nuancer le volume.

Dans notre exemple, dans le MovieClip `console_mc`, nous pouvons identifier le symbole `audio_mc`. Ce dernier contient lui-même deux autres clips dont `audioOff_mc` qui affiche un trait rouge et `audioOn_mc` qui reste neutre.

Nous spécifions ici qu'en cliquant sur chaque symbole contenu dans `audio_mc` (`evt.target`), nous modifions le volume audio de la vidéo. Au premier clic, le son devient muet, et au suivant, il redevient normal, et ainsi de suite. Une condition permet de vérifier si le volume est déjà muet ou non et inverse la valeur selon le résultat.

Pour que le dispositif soit plus ergonomique, nous ajoutons un contrôle de visibilité sur chacun des boutons de sorte que l'un disparaît à chaque fois qu'on l'active, et laisse alors l'autre prendre sa place :

```
// Audio
console_mc.audio_mc.audioOff_mc.visible=false;
console_mc.audio_mc.audioOff_mc.buttonMode=true;

console_mc.audio_mc.addEventListener(MouseEvent.CLICK,audioVideo);
function audioVideo (evt:MouseEvent) {
   if (ecranVideo.volume>0) {
      ecranVideo.volume=0;
      evt.target.visible=false;
      console_mc.audio_mc.audioOff_mc.visible=true;
} else {
      ecranVideo.volume=1;
      evt.target.visible=false;
      console_mc.audio_mc.audioOn_mc.visible=true;
   }
}
```

Augmenter et diminuer progressivement le son

Exemples > ch8_videoInteractive_2.fla

Nous avons vu qu'il est possible de modifier une valeur en l'incrémentant au sein d'un gestionnaire de type `Event.ENTER_FRAME` (voir Chapitre 1). Vous pouvez donc aussi modifier le son, sur action utilisateur, en appelant un écouteur qui active la modification de l'audio tant que celui-ci n'atteint pas une certaine valeur. Nous utilisons pour ce faire un gestionnaire de type `Event.ENTER_FRAME` et des structures conditionnelles. Nous obtenons ceci :

```
// Audio
console_mc.audio_mc.audioOff_mc.visible=false;
console_mc.audio_mc.audioOff_mc.buttonMode=true;

console_mc.audio_mc.audioOn_mc.addEventListener(MouseEvent.CLICK,fonctionBaisser);
function fonctionBaisser (evt:MouseEvent) {
   addEventListener(Event.ENTER_FRAME,reduireAudio);
}
//
function reduireAudio (evt:Event) {
   if (ecranVideo.volume>0) {
      ecranVideo.volume-=0.01;
      trace(ecranVideo.volume)
      if (ecranVideo.volume<=0.01) {
         ecranVideo.volume=0;
             removeEventListener(Event.ENTER_FRAME,reduireAudio);
             console_mc.audio_mc.audioOn_mc.visible=false;
             console_mc.audio_mc.audioOff_mc.visible=true;
      }
   }
}
```

```
console_mc.audio_mc.audioOff_mc.addEventListener(MouseEvent.CLICK,fonctionMonter);
function fonctionMonter (evt:MouseEvent) {
   addEventListener(Event.ENTER_FRAME,monterAudio);
}
//
function monterAudio (evt:Event) {
   if (ecranVideo.volume<1) {
      ecranVideo.volume+=0.01;
      trace(ecranVideo.volume)
      if (ecranVideo.volume>=0.99) {
         ecranVideo.volume=1;
         removeEventListener(Event.ENTER_FRAME,monterAudio);
         console_mc.audio_mc.audioOn_mc.visible=true;
         console_mc.audio_mc.audioOff_mc.visible=false;
      }
   }
}
```

Dans cet exemple, chaque bouton exécute une fonction qui lui est propre. Dans cette fonction, le gestionnaire `Event.ENTER_FRAME` appelle une autre fonction. C'est alors que la valeur de l'audio est, soit augmentée, soit diminuée, selon la fonction qui est exécutée (suivant le bouton qui est cliqué). Une fois que la valeur est intégralement renversée, alors, une instruction interrompt la fonction.

Pour modifier la vitesse de progression du volume, il suffit de modifier la valeur du pas d'incrémentation, ici spécifiée à `0.01`.

À retenir

- Il est possible de personnaliser une console de contrôle vidéo en associant à des clips des fonctions qui affectent les propriétés du composant vidéo en cours d'exécution.
- Il est possible d'accélérer ou de rembobiner une vidéo en utilisant la propriété `playheadTime` et `seek()`. Mais cette technique offre une précision relative au nombre d'images-clés encodées dans le flux vidéo.
- Il est possible de modifier le son d'un flux audio en contrôlant la propriété `volume`. Une variation progressive de l'audio peut être effectuée grâce à un gestionnaire de type `Event.ENTER_FRAME`.

Chapitrage vidéo

La navigation au sein d'une vidéo est très simple à mettre en place. Il suffit d'utiliser la méthode `seek()` que nous venons déjà de rencontrer. Nous spécifions alors, en paramètre de cette méthode, le timecode à atteindre. Le timecode représente la position des images d'une vidéo et s'exprime en secondes. Un timecode de 12 désigne à la tête de lecture d'atteinte l'image située à la seconde 12 de la vidéo.

Naturellement, comme évoqué plus haut, nous devons aussi considérer que le flux vidéo dispose d'un nombre confortable d'images-clés ou alors, nous acceptons que le ciblage fluctue plus ou moins à une ou deux secondes près.

Pour définir un timecode précisément, considérons l'exemple d'une vidéo d'une cadence de 25 ips pour laquelle une image vaut 4 centièmes de secondes (100 centièmes de seconde / 25 images = 0,04 seconde). Pour cette vidéo, l'expression suivante appelle la soixante-septième seconde :

```
ecranVideo.seek(67) ;
```

Mais, l'expression suivante appelle la troisième image de la soixante-septième seconde (soit 67 secondes + 3 × 0,04). Si l'image appelée n'est pas une image-clé, comme vu précédemment, c'est l'image-clé suivante la plus proche qui est affichée et c'est à partir de cette image-clé que va se prolonger la vidéo :

```
ecranVideo.seek(67.12) ;
```

Dans l'exemple suivant, nous utilisons la méthode `seek()` à travers une série de vignettes afin de créer un système de chapitrage. Mais, nous allons plus loin que dans la section précédente en optimisant ici le code et en rassemblant d'abord toutes les conditions dans une seule et même fonction. À l'intérieur de cette fonction, nous ajoutons aussi une propriété qui permet de charger éventuellement une autre vidéo, en lieu et place de la vidéo active. Nous combinons donc deux méthodes : `seek()` et `source()`.

Exemples > ch8_videoInteractive_3.fla

Dans le document "ch8_videoInteractive_3.fla", sur la scène principale, se trouve un composant qui exécute directement la vidéo à la publication du document. En dessous, un menu est composé de plusieurs vignettes. Chacune de ces vignettes est isolée dans un MovieClip et possède un nom d'occurrence (voir Figure 8.3).

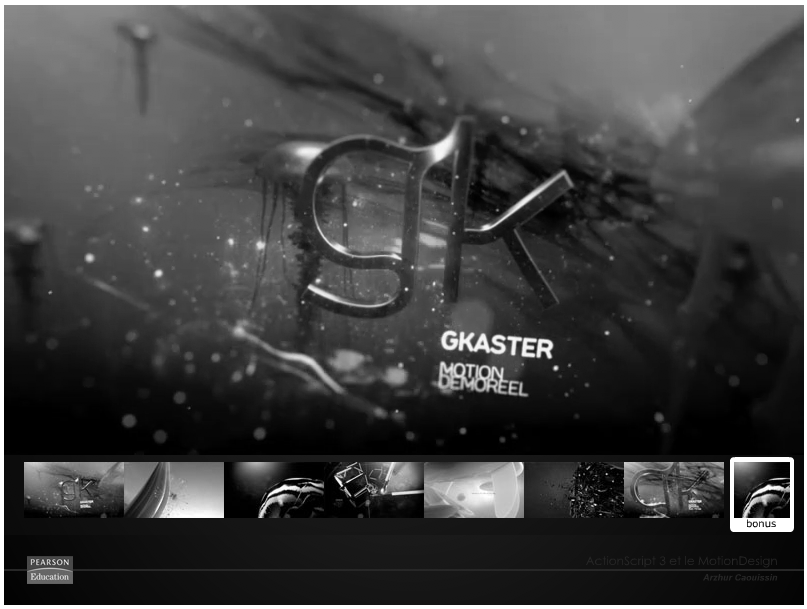
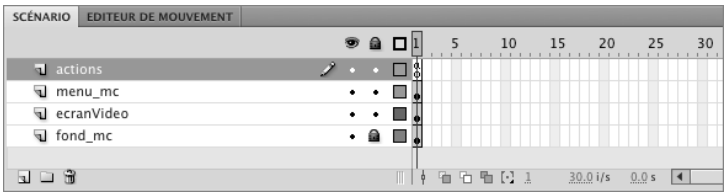


Figure 8.3
Aperçu du document après publication.

Dans le scénario, au-dessus de calque `fond_mc`, nous identifions le composant vidéo, le menu et un calque `actions` (voir Figure 8.4).

Figure 8.4
Aperçu du scénario de la scène principale.



Dans le calque `actions`, nous lisons le code suivant :

```
var cheminVideo:String;
var timeCode:Number=0;

menu_mc.addEventListener(MouseEvent.CLICK,timeCode1);
function timeCode1 (evt:MouseEvent) {
   if (evt.target.name=="lien1_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=0.2;
   }
   if (evt.target.name=="lien2_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=10;
   }
   if (evt.target.name=="lien3_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=19;
   }
   if (evt.target.name=="lien4_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=45;
   }
   if (evt.target.name=="lien5_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=66;
   }
   if (evt.target.name=="lien6_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=92;
   }
   if (evt.target.name=="lien7_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=119.04;
   }
   if (evt.target.name=="bonus_btn") {
      cheminVideo="gKaster/gKaster-amusement.f4v";
      timeCode=0;
   }
   ecranVideo.source=cheminVideo;
   ecranVideo.seek(timeCode);
}
```

Nous déclarons en premier lieu deux variables :

```
var cheminVideo:String;
var timeCode:Number=0;
```

La première désigne une chaîne de caractères qui véhiculera le chemin d'un fichier vidéo, disponible dans notre projet. Chaque lien cliqué va ainsi pouvoir renseigner cette valeur pour cibler le fichier qui lui est propre. Plus loin dans le code, une instruction reprend la valeur alors renseignée pour appeler le fichier correspondant avec la méthode `source()`.

La deuxième variable est un nombre et initialise les timecodes avant qu'ils ne soient éventuellement modifiés à travers la fonction.

Ensuite, un écouteur est attaché au menu et non aux vignettes elles-mêmes. Cela permet d'introduire des conditions qui définissent, selon l'objet du menu qui est cliqué (`evt.target.name`), telle ou telle instruction (voir Chapitre 5 pour en savoir plus sur la propriété `target`) :

```
if (evt.target.name=="lien1_btn") {
      cheminVideo="gKaster/gKaster-demoreel.f4v";
      timeCode=0.2;
   }
```

La condition vérifie que le nom de l'objet cliqué correspond bien à celui spécifié entre parenthèses. Lorsque la valeur est vérifiée, la variable `cheminVideo` est renseignée, ainsi que le timecode. Une condition est crée pour chaque bouton.

En fin de programme, les deux variables sont utilisées pour activer le chapitrage :

```
ecranVideo.source=cheminVideo;
ecranVideo.seek(timeCode);
```

Vous remarquez le dernier bouton `bonus`, qui se distingue des autres en cela qu'il appelle un fichier différent. Notez que nous exécutons le programme localement, et donc, que les vidéos sont chargées instantanément. Nous pourrions donc spécifier un timecode pour cette nouvelle vidéo. Mais, n'oubliez pas que l'on ne peut atteindre une image d'un fichier vidéo que si l'image appelée est déjà chargée. Nous ne recommandons donc pas de spécifier une autre valeur que 0 lorsqu'une nouvelle vidéo est appelée. Le seul moyen de permettre d'atteindre directement une image d'une vidéo non chargée est d'utiliser la technologie Flash Media Server, qui autorise la diffusion en vrai *streaming* (en continu).

Nous remarquons ici que le chapitrage peut appeler indifféremment des séquences dans un même flux vidéo (avec `seek`) que plusieurs fichiers vidéo distincts (avec source). Notez que la création de flux séparés (avec `source`) offre une plus grande souplesse dans la navigation, car l'ensemble de la vidéo n'a alors pas besoin d'être chargée pour permettre d'accéder à d'autres chapitres. Les références appelées à chaque requête (avec `source`) sont toujours de nouveaux flux vidéo, indépendants, qui se substituent à la vidéo en cours de lecture. Que les chapitres soient constitués de vidéos distinctes ou contenues dans une seul signal vidéo, nous utilisons un seul, même et unique, composant.

Adobe propose une technologie serveur adaptée à la gestion de flux vidéo haute définition en continu. Cette technologie se nomme Flash Media Server. Vous trouverez des informations sur son utilisation ainsi qu'un serveur de test d'hébergement à l'adresse suivante : **http://www.streamedia.eu/#news_fr_5.html**.

Chapitrage vidéo avec les points de repère

Il est également possible de créer un système de chapitrage à partir de points de repère. Nous utilisons alors la méthode `seekToNavCuePoint()`. Cette technique est plus précise car il est possible de contrôler la position des images-clés. Reportez-vous à la section "Synchroniser des actions avec les points de repère" pour en savoir plus sur cette méthode.

À retenir

- Pour créer un système de chapitrage simple, nous utilisons la méthode `seek()`. Mais le nombre d'images-clés encodées dans le flux vidéo détermine la précision du ciblage.
- Le chapitrage peut appeler aussi bien des séquences dans un même flux vidéo que plusieurs fichiers vidéo distincts. Mais, étant donné que le ciblage ne permet d'atteindre que les flux déjà chargés ou qui démarrent, l'option avec des fichiers séparés demeure la plus confortable pour l'utilisateur.

Sous-titrage vidéo

Flash met à disposition un composant qui simplifie la gestion des sous-titres de la vidéo. Vous devez pour cela utiliser d'abord un composant `FLVPlayBack` pour y charger une vidéo, puis, créer un fichier XML qui contient le texte pour les sous-titres en respectant un format bien défini. Enfin, vous devez ajouter sur la scène un composant `FLVPlayBackCaptioning` qui se charge de placer le texte du fichier XML dans le champ de texte de votre choix.

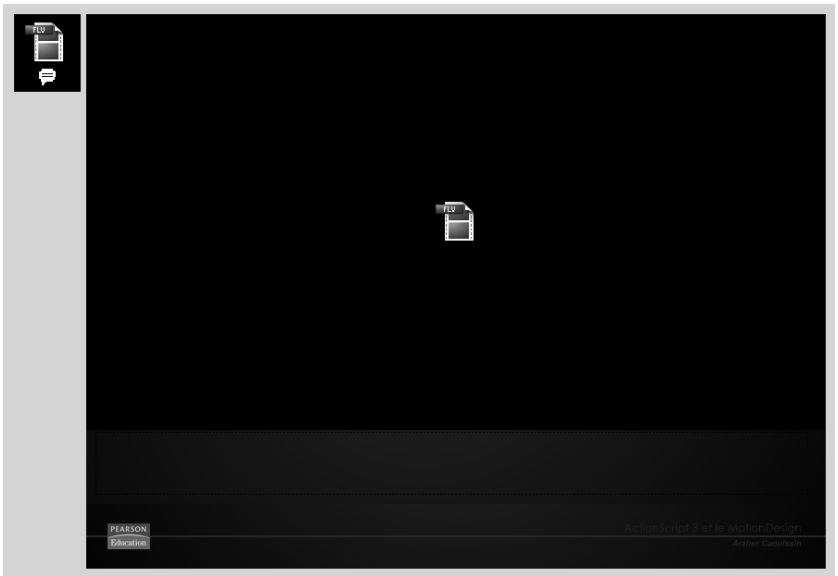
Dans cet exemple, nous ajoutons des sous-titres qui accompagnent une création autour d'un poème de Dan Andersson – écrivain suédois – (voir Figure 8.5). Les textes sont stockés dans un fichier XML.

Figure 8.5
Aperçu du document après publication.

Exemples > ch8_videoInteractive_4.fla

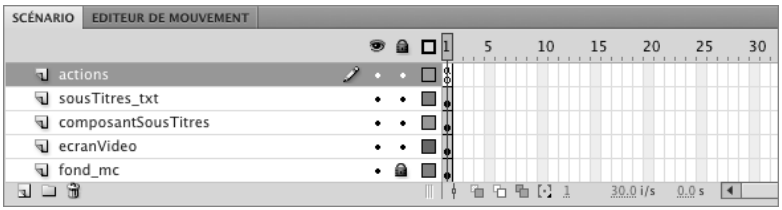
Dans le document "ch8_videoInteractive_4.fla", la scène principale affiche un composant vidéo, un texte dynamique déjà formaté avec une typo embarquée (voir Chapitre 15 pour en savoir plus sur la typographie) et un composant `FLVPlayBackCaptioning` situé hors champ (voir Figure 8.6). Chaque objet possède un nom d'occurrence. La vidéo appelée se nomme "gkaster-c19.f4v".

Figure 8.6
Aperçu de la scène principale.



Dans la fenêtre de scénario, au-dessus du calque `fond_mc`, les trois objets sont clairement répartis vers des calques distincts (voir Figure 8.7).

Figure 8.7
Fenêtre de scénario de la scène principale.



Dans le calque `actions`, nous accédons au code suivant :

```
composantSousTitres.flvPlayback=ecranVideo;
composantSousTitres.source="gKaster/sousTitres.xml";
composantSousTitres.autoLayout=false;
composantSousTitres.captionTargetName="sousTitres_txt";
```

Dans le répertoire "gKaster" des exemples du livre, se trouve un document XML qui contient les sous-titres et repose sur la structure suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<tt xml:lang="fr" xmlns="http://www.w3.org/2006/04/ttaf1" xmlns:tts="http://www.w3.org
➥ /2006/04/ttaf1#styling">
   <head>
   </head>
   <body>
   <div xml:lang="fr">
      <p begin="00:00:00.00" dur="00:00:03.00">gKaster <span tts:fontFamily="Verdana"
      tts:fontSize="+12">C19</span></p>
      <p begin="00:00:09.00" dur="00:00:03.00">Le soleil se lève à l'horizon,</p>
      <p begin="00:00:12.45" dur="00:00:03.30">regarde, au bord du mont du village de
      paille.</p>
      <p begin="00:00:17.00" dur="00:00:02.30">Sur le fleuve fragile, dégoulinent,</p>
      <p begin="00:00:20.30" dur="00:00:02.30">marchent, les hommes en silence.</p>
      <p begin="00:00:25.00" dur="00:00:02.30">Sous le ciel gris du matin frais,</p>
      <p begin="00:00:28.30" dur="00:00:04.00">des pas lourds foulent le sol jonché
      des roses...</p>
      <p begin="00:00:36.00" dur="00:00:03.00">Des têtes s'y plient comme à la
      prière,</p>
      <p begin="00:00:40.00" dur="00:00:05.00">loin des terres arrides s'est fait
      porter feu le poête.</p>
      <p begin="00:00:52.00" dur="00:00:05.00">Au travers de la clairière verdoyante,
      sous la rosée,</p>
      <p begin="00:01:00.30" dur="00:00:04.00">il a terminé ses années de douleur.</p>
      <p begin="00:01:06.00" dur="00:00:06.00">Mais quand le cercueil s'avance, noir,
      à travers la forêt verte du printemps</p>
      <p begin="00:01:13.00" dur="00:00:05.00">Le silence traverse la nature qui
      s'éveille.</p>
      <p begin="00:01:18.00" dur="00:00:04.00">Et là, le vent d'ouest s'arrête, en se
      demandant :</p>
      <p begin="00:01:22.30" dur="00:00:05.00">Qui foule ces roses d'un pas si
      lourd ?</p>
      <p begin="00:01:32.00" dur="00:00:06.00">Allez doucement, c'est peut-être une
      fleur qui vient de mourir.</p>
      <p begin="00:01:45.00" dur="00:00:06.00">Si j'étais Houragan, je
      l'accompagnerais jusqu'au bout du chemin.</p>
   </div>
   </body>
</tt>
```

Pour mettre en place un système de sous-titrage, il suffit de glisser-déposer une occurrence du composant `FLVPlayBackCaptioning` depuis la fenêtre des composants (Fenêtre > Composants) sur la même scène que celle où figure déjà le composant de lecture du flux vidéo `FLVPlayBack`. Puis, il faut lui attribuer un nom d'occurrence. Dans notre exemple, nous le nommons `composantSousTitres`.

Pour lier l'objet `composantSousTitres` à la vidéo, dans le code, nous spécifions :

```
composantSousTitres.flvPlayback=ecranVideo;
```

Pour lui permettre d'identifier l'emplacement du fichier XML, nous ajoutons également :

```
composantSousTitres.source="gKaster/sousTitres.xml";
```

Nous avons la possibilité de magnétiser le champ de texte dynamique sur la vidéo (`true`) ou non (`false`). Pour que le champ de texte reste à sa position actuelle dans la scène, nous écrivons :

```
composantSousTitres.autoLayout=false;
```

Enfin, pour que le texte s'affiche dans un champ dynamique de notre propre création, nous associons, à une dernière commande, le nom d'occurrence du texte dynamique placé sur la scène :

```
composantSousTitres.captionTargetName="sousTitres_txt";
```

À défaut de spécifier un champ de texte dynamique pour accueillir les légendes, le composant les affichera directement sur la vidéo dans une zone rectangulaire de fond noir.

L'ensemble de ces instructions reprend les paramètres du composant `FLVPlayBackCaptioning` également disponibles et paramétrables depuis l'Inspecteur de composants.

Dans le fichier XML (de type texte TT – *Timed Text* – mais au format XML), nous pouvons lire une structure proche d'une page HTML. Ce document répond en fait à une organisation normalisée par le W3C (**http://www.w3.org/AudioVideo/TT/**). Tout en respectant la syntaxe propre à ce formatage, il nous suffit d'ajouter autant de ligne que de sous-titres doivent apparaître, et ensuite de renseigner chacune des propriétés.

Le fichier XML que nous utilisons repose sur le mécanisme suivant. Pour chaque nouveau sous-titre, le composant requiert une balise `<p></p>` :

```
<p begin="00:00:00.00" dur="00:00:03.00">gKaster
   <span tts:fontFamily="Verdana" tts:fontSize="+12">C19</span></p>
```

Dans cette balise, quatre attributs peuvent être renseignés : `begin`, `dur`, `end` et `style`. Dans notre exemple, les deux premiers seulement sont utilisés.

- L'attribut `begin` sert à définir le timecode à partir duquel le titre doit apparaître.
- L'attribut `dur` désigne la durée de ce sous-titre.
- L'attribut `end`, qui se définit de la même manière que les deux précédents, désigne le timecode de fin du sous-titre. Il n'a pas lieu d'être si l'on connaît déjà la durée.
- L'attribut `style`, optionnel, permet de gérer le formatage des textes à partir de styles HTML de base.

La gestion de l'affichage du sous-titre est précise, contrairement à la navigation utilisée avec la méthode `seek()`, car c'est le fichier XML qui détermine le moment où les textes doivent apparaître et non les images-clés de la vidéo.

Le dernier attribut, `style`, permet d'appliquer un style dont nous pouvons définir le formatage dans l'en-tête du document `<head></head>`, en amont. Pour ce faire, nous utilisons la syntaxe suivante :

```
<head>
   <styling>
      <style id="style1" tts:fontSize="20"/>
   </styling>
</head>
<body>
   <div xml:lang="en">
      <p begin="00:00:00.00" dur="00:00:03.00" style="style1">gKaster C19</p>
   </div>
</body>
```

Dans notre exemple, nous n'utiliserons pas les styles. En publiant le document Flash, la vidéo est lue et des légendes se succèdent les unes à la suite des autres, selon le timecode défini dans le fichier XML.

Vidéo

Le composant `CaptionButton`

Vous pouvez glisser-déposer le composant `CaptionButton` dans la même scène que le composant `FLVPlayBack Captionning`. Il permet à l'utilisateur de désactiver et réactiver l'affichage des sous-titres. Ce composant, comme les boutons de contrôle de la vidéo, peut être personnalisé en double-cliquant dessus jusqu'à atteindre les objets graphiques qui le composent.

Étendre les formatages du document XML pour les sous-titres

Des options de formatage sont accessibles pour les données contenues dans le fichier XML des sous-titres. Ces formatages répondent à une norme précise et standardisée par le W3C. Vous trouverez le descriptif détaillé de ces options et les balises à employer à l'adresse suivante : **http://help.adobe.com/fr_FR/ActionScript/3.0_UsingComponentsAS3/WS5b3ccc516d4fbf351e63e3d118a9c65b32-7ee5.html**.

À retenir

- Il est possible de déployer simplement un système de sous-titrage grâce à l'utilisation du composant `FLVPlayBackCaptioning`.
- La synchronisation avec le flux vidéo est précise car c'est le code qui gère l'affichage et non les images-clés de la vidéo.
- Des styles de formatage peuvent être appliqués aux textes des sous-titres, depuis Flash ou à partir de formatages définis dans le fichier XML.

Boucle vidéo

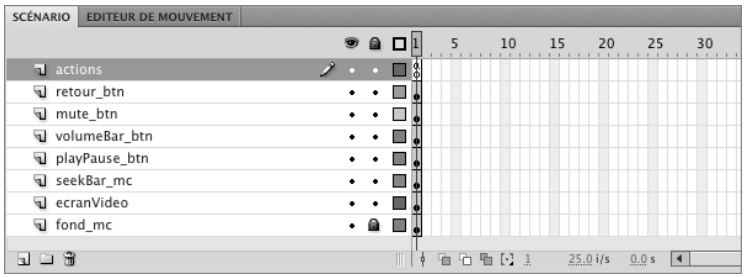
Par défaut, une vidéo gérée avec un composant `FLVPlayBack` est lue de bout en bout sans boucler sur elle-même. Il peut être intéressant de permettre de relancer la vidéo pour créer un flux ininterrompu. Pour cela, nous utilisons une classe qui détecte le comportement de la vidéo. Cette classe se nomme `videoEvent`.

Dans cet exemple, nous utilisons une séquence vidéo associée à une barre de progression, ceci afin de nous permettre, lors de la publication, de tester l'effet de boucle plus rapidement, en déplaçant simplement la tête de lecture.

Exemples > ch8_videoInteractive_5.fla

Dans le document "ch8_videoInteractive_5.fla", la scène principale affiche un composant vidéo nommé `ecranVideo` et des occurrences de composants de contrôle réparties vers des calques distincts (voir Figure 8.8).

Figure 8.8
Aperçu du scénario de la scène principale.



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

```
import fl.video.VideoEvent;
//
ecranVideo.addEventListener(VideoEvent.COMPLETE,boucleVideo);
function boucleVideo(evt:VideoEvent) {
   ecranVideo.seek(0);
   ecranVideo.play();
}
```

D'abord, nous importons la classe `videoEvent` qui permet de détecter le comportement du flux vidéo. Puis, nous créons un écouteur que nous attachons au composant `FLVPlayBack` placé sur la scène. Le gestionnaire d'événements fait référence à la classe importée et invoque la propriété `COMPLETE` afin de détecter le moment où la vidéo est terminée.

Lorsque la vidéo est achevée, la fonction `boucleVideo`, appelée par l'écouteur, exécute deux instructions. La première, `seek(0)`, replace la tête de lecture à l'image 0. La seconde, `play()`, indique de reprendre la lecture.

À retenir

- Il est possible de générer une boucle vidéo à l'aide de la classe `videoEvent`. Il faut pour cela replacer la tête de lecture à l'image 0 et relancer la lecture de la vidéo.
- En détectant la fin de lecture de la vidéo, vous pouvez aussi associer d'autres actions, comme l'enchaînement avec d'autres flux vidéo ou l'affichage d'un autre contenu.

Vidéo

Enchaîner plusieurs vidéos à la suite

En détectant la fin de lecture d'un flux vidéo, vous êtes en mesure de placer d'autres instructions à la place du repositionnement de la tête de lecture comme nous l'avons fait ici. Par exemple, vous pouvez très bien invoquer un autre fichier vidéo en vue de créer un enchaînement de plusieurs vidéos.

Pour appeler une autre vidéo, utilisez l'instruction :

```
ecranVideo.source="cheminDeLaVideo.f4v" ;
```

Pensez éventuellement y adjoindre l'instruction `play()`, comme pour notre boucle, afin de garantir le redémarrage automatique de la lecture.

Synchroniser des actions avec les points de repère

Les points de repère (ou *CuePoints*) sont des marqueurs que l'on distribue tout au long de la vidéo. Ils permettent :

- D'y associer des actions de contrôle de lecture de la vidéo pour s'y référer comme dans un système de chapitrage (repères de navigation).
- D'y placer des événements programmés en ActionScript (repères d'événements), qui planifient des actions dans le temps.

Chacun des deux procédés requiert naturellement une vidéo, mais aussi la création des images-clés qui définissent le timecode sur lequel chaque repère doit être placé.

Il existe plusieurs techniques pour la création de ces repères : la première consiste à les intégrer physiquement dans l'encodage du signal vidéo, la deuxième à les stocker dans un fichier XML appelé ensuite avec ActionScript, la troisième consiste à les définir directement dans la fenêtre d'Actions du document par Flash et la quatrième engage le remplissage manuel de la propriété cuepoints, disponible depuis l'Inspecteur de composants.

- À l'intérieur du flux vidéo, les repères sont introduits depuis After Effects, Premiere Pro ou Adobe Media Encoder. Ils sont encapsulés dans le fichier une fois celui-ci rendu, si bien que toute modification du timecode, d'un seul de ces repères, induit de procéder à un nouveau rendu. Cette méthode présenterait peu d'intérêt si elle ne permettait pas un ciblage précis pour les repères de navigation. Dans ce contexte, en effet, à chaque fois que nous introduisons un repère de navigation, nous générons aussi une image-clé. Il en résulte que cette méthode est celle que nous préférons employer pour définir des repères de navigation, car ils permettent de travailler avec une grande précision. Pour les repères d'événements, en revanche, les autres techniques restent plus confortables.
- La création d'un fichier XML peut être intéressante pour les cas où un nombre élevé de repères d'événements est enregistré et où la valeur du timecode de chacun de ces repères doit être modifiée souvent. La gestion d'un fichier XML, dans ce cas précis, peut devenir intéressante car elle épargne d'avoir à republier le document Flash pour le mettre à jour. Nous n'abordons pas cette méthode dans notre ouvrage.

- La troisième option consiste à générer les points de repère d'événements directement dans la fenêtre d'actions du document Flash. Cette technique est très souple puisque, d'abord, le fait de dissocier les repères du flux vidéo va permettre de simplifier considérablement leur mise à jour. Ensuite, une ligne de code suffit par point de repère et, placés en tête de la fenêtre d'actions, toute mise à jour devient extrêmement simple à effectuer.
- La quatrième option, qui permet d'utiliser directement le composant, reprend le principe de la troisième, mais n'offre pas la souplesse de manipulation du code où nous pouvons plus facilement établir des relations entre le nom des repères que nous avons définis et les instructions à exécuter en regard des noms ajoutés. Nous n'abordons pas non plus cette dernière option dans notre ouvrage.

Dans cette section, nous décrivons la méthode de l'encodage dans le flux vidéo, pour l'utilisation des repères de navigation avec un ciblage précis. Plus loin, nous revenons sur un document Flash pour y introduire, en ActionScript, les repères d'événements dynamiques.

Repères de navigation

Dans cet exemple, nous allons placer des repères de navigation à l'intérieur de la vidéo demoreel de la société gKaster de manière à permettre une navigation précise. Cette vidéo est initialement encodée en F4V.

Les points de repère étant une fonctionnalité ajoutée par Adobe dans le format Flash vidéo, cette fonctionnalité n'est malheureusement utilisable que pour le format FLV. En plus de définir les points de repère, nous allons donc aussi modifier le format d'encodage de la vidéo dont nous disposons. Ce qui nous permet surtout d'aborder la manière d'encoder une vidéo pour l'ajout de repères de navigation. Naturellement, dans le cadre d'un projet réel de création de contenu, nous vous recommandons de repartir de la source vidéo non compressée pour obtenir un meilleur rendu.

Pour cet exemple, nous reprenons le dispositif de chapitrage utilisé préalablement avec la méthode `seek()`. Nous y remplaçons, après l'encodage, les références `seek()`, imprécises, par une action de détection de repères de navigation, plus précise, avec la méthode `seekToNavCuePoint()`.

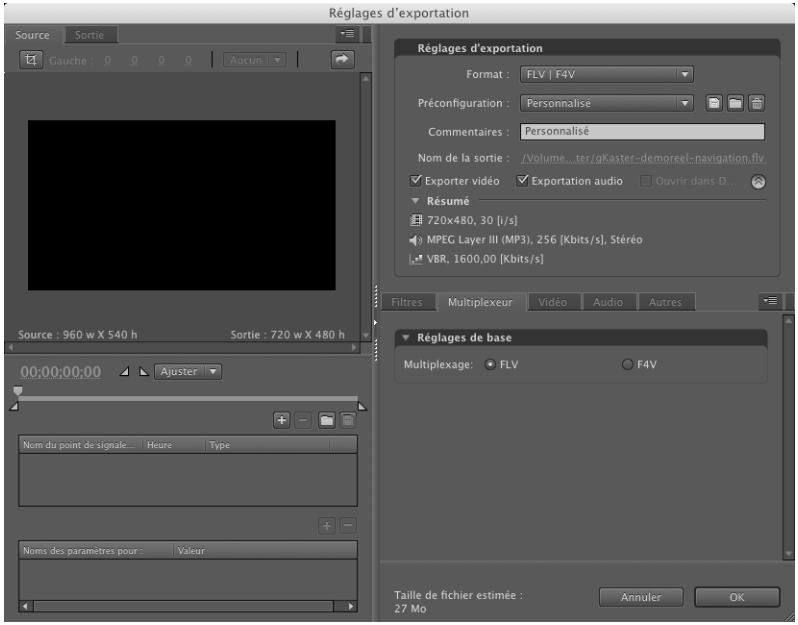
Exemples > ch8_videoInteractive_6.fla

Encoder la vidéo

Revenons d'abord sur l'encodage de la vidéo :

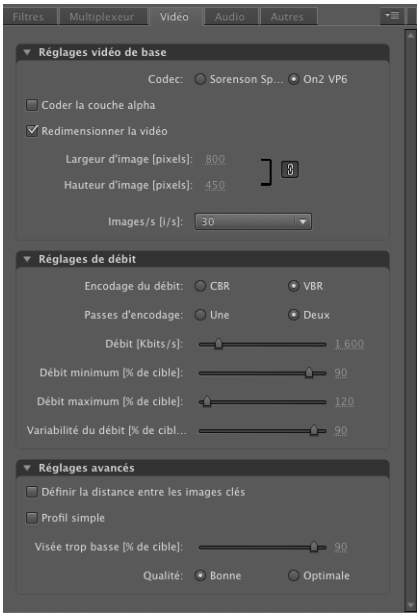
1. Lancez l'application Adobe Media Encoder.
2. Ajoutez, à la liste de rendu, la vidéo intitulée gKaster-demoreel.f4v.
3. Puis, cliquez directement sur le lien jaune de la colonne Prédéfinir, ou bien sur le bouton Réglages, pour ouvrir la fenêtre de réglages (voir Figure 8.9).

Figure 8.9
Réglages d'exportation.



4. Dans la partie de droite, sélectionnez l'onglet Multiplexeur puis activez l'option FLV.
5. Depuis l'onglet Vidéo, spécifiez la cadence de l'image. Dans notre exemple, nous maintenons une cadence d'origine à 30ips (voir Figure 8.10).

Figure 8.10
Choix de la cadence avant l'ajout de points de repère.



Dans l'encodeur, notez qu'il est important de déterminer la cadence de la vidéo à partir du champ Image, de l'onglet vidéo, avant de créer les points de repère. Les points de repère sont ici calculés en fonction du nombre d'images par seconde préalablement défini. Si nous modifions la cadence de la vidéo après avoir généré les points de repère, comme nous configurons les points de repère à partir de données invariables, nous risquons de subir un décalage, voire, de ne pouvoir atteindre certaines séquences une fois le document Flash publié. Par exemple, nous spécifions une cadence initiale à 30ips, et nous ajoutons un repère à l'image 29 de la énième seconde. Si nous modifions, après avoir ajouté le repère, la cadence de la vidéo en la ramenant par exemple à 25ips, l'image 29 qui n'existe plus ne pourra être trouvée.

Vérifiez ensuite que les dimensions de la vidéo correspondent à la surface disponible dans notre document (800 × 450 pixels).

Comment réactiver les champs de dimensionnement de l'encodeur ? *L'encodeur étant un peu instable, vous pouvez avoir besoin de réactiver les options de réglage pour les éditer. Dans le volet de gauche de la fenêtre d'encodage, cliquez alors sur l'outil de recadrage, puis, dans l'onglet Sortie, sélectionnez l'option Modifier la taille de sortie. Puis, revenez dans l'onglet Source et désactivez l'outil de recadrage.*

Ajustez enfin légèrement la compression de sorte que l'ajout d'images-clés ne réduise pas l'espace disponible pour coder les autres images, et donc, n'altère pas la qualité globale de la vidéo. Passez le débit minimum à 90.

Une fois l'échantillonnage calibré, nous pouvons ajouter les repères de navigation.

Ajout des repères de navigation

Pour placer des repères de navigation, nous devons revenir sur la partie gauche de la fenêtre d'encodage Adobe Media Encoder. Pour chaque repère, nous positionnons la tête de lecture à l'emplacement voulu. Puis, nous ajoutons un repère en cliquant, dans la partie inférieure, sur le bouton Plus.

Placez la tête de lecture au timecode 00;00;00;07 qui correspond à la septième image. Pour contrôler la position de la tête de lecture avec précision, et glissez sur le timecode sans relâcher le pointeur. Attention un simple clic rend le champs éditable. Vous pouvez également y saisir manuellement une valeur, ou utiliser les flèches droite et gauche qui incrémentent ou décrémentent le time code de l'image. Si vous saisissez une valeur, pour confirmer son entrée, cliquez ensuite dans une zone neutre de la fenêtre. N'appuyez pas sur la touche Entrée qui ferme la fenêtre.

La vidéo étant initialement cadencée à 30 ips, le timecode s'arrête à l'image 30 et poursuit à la seconde suivante. La gestion du timecode ici n'est pas similaire à celle que nous avons contrôlée par ActionScript, qui est définie en secondes uniquement.

1. Dans la partie inférieure de la fenêtre, cliquez sur le bouton plus (+). Et, depuis le menu déroulant marqué Évènement, sélectionnez l'option Navigation.
2. Une entrée est enregistrée et ajoutée à la liste.
3. À gauche, cliquez sur le nom attribué par défaut, Point de signalement, pour le renommer. Saisissez par exemple : repere1 (voir Figure 8.11).

Figure 8.11
Ajout d'un repère de navigation.

Procédez de même pour l'ensemble des repères qui représentent le début de chaque séquence animée pour lesquelles nous disposons de vignettes dans le document Flash. Par exemple, placez la tête de lecture à l'image 00;00;13;13 pour ajouter une image-clé au début de la séquence qui correspond à la deuxième vignette dans Flash. Puis cliquez à nouveau sur Plus (+), pour définir un nouveau repère (voir Figure 8.12) et ainsi de suite.

Figure 8.12
Ajout d'un repère de navigation.

Les timecodes suivants correspondent à l'ensemble des séquences identifiées dans le Flash, par des vignettes. Une vidéo déjà codée avec les repères de navigation est disponible dans le dossier "gKaster" sous le nom "gKaster-demoreel-navigation.flv".

- 00;00;00;07 = repere1
- 00;00;13;14 = repere2
- 00;00;17;27 = repere3
- 00;00;41;14 = repere4
- 00;01;05;10 = repere5
- 00;01;29;26 = repere6
- 00;01;58;07 = repere7

Confirmez l'encodage en cliquant sur OK. Choisissez le nom de sortie "gkaster-demoreel-navigation.flv" et remplacez éventuellement le document déjà encodé qui porte le même nom. Puis lancez un rendu.

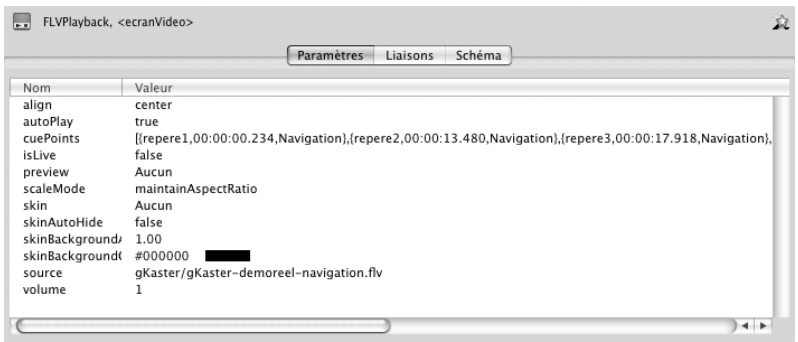
En plus des images-clés générées automatiquement toutes les deux secondes, chaque point de repère que nous avons spécifié en deviendra également une au moment de l'encodage. Nous pouvons maintenant réintégrer cette vidéo au document Flash et y saisir les instructions de navigation.

Détecter les points de repère avec ActionScript

Avant de programmer la détection des points de repère, nous devons mettre à jour le composant vidéo en y rattachant la vidéo exportée au format FLV. La vidéo étant déjà intégrée dans ce document, vous pouvez la remplacer en mettant simplement à jour la propriété source de l'Inspecteur de composants :

1. Revenez dans Flash et ouvrez le document "ch8_videoInteractive_6.fla".
2. Sélectionnez le composant vidéo.
3. Dans l'Inspecteur de composants (Fenêtre > Inspecteur de composants), modifiez la source en la cliquant deux fois. Puis, ciblez la vidéo que vous venez d'encoder et qui contient vos points de repère de navigation (voir Figure 8.13).

Figure 8.13
Liaison avec la vidéo.



Vous remarquez que l'Inspecteur de composants a détecté l'ensemble des repères que nous avons intégrés dans la vidéo, dans le paramètre `cuePoints`.

Revenez sur le calque `actions`. La fenêtre affiche le code suivant :

```
var repere:String="";

menu_mc.addEventListener(MouseEvent.CLICK,atteindreRepere);
function atteindreRepere(evt:MouseEvent) {
  if (evt.target.name=="lien1_btn") {
    repere="repere1";
  }
  if (evt.target.name=="lien2_btn") {
    repere="repere2";
  }
  if (evt.target.name=="lien3_btn") {
    repere="repere3";
  }
  if (evt.target.name=="lien4_btn") {
    repere="repere4";
  }
  if (evt.target.name=="lien5_btn") {
    repere="repere5";
  }
  if (evt.target.name=="lien6_btn") {
    repere="repere6";
  }
  if (evt.target.name=="lien7_btn") {
    repere="repere7";
  }
  trace(repere)
  ecranVideo.seekToNavCuePoint(repere);
}
```

De la même manière que nous avons distribué la méthode `seek()` dans les chapitres précédents, nous utilisons ici la méthode `seekToNavCuePoint()`. Nous spécifions, en paramètre, une valeur "chaîne de caractères" qui correspond au nom du point de repère de navigation ajouté dans l'encodeur, et ce, pour chaque lien.

Publiez le document en faisant Cmd+Entrée (Mac) ou Ctrl+Entrée (Windows). La vidéo joue instantanément. En cliquant sur chacune des vignettes, le ciblage atteint précisément le timecode spécifié dans le flux vidéo.

Repères d'événements

Les repères d'événements ont pour objectif de planifier une action à l'intérieur du document Flash, à mesure de la progression de la vidéo.

Comme pour les repères de navigation, nous devons au préalable définir les repères d'événements avant d'ajouter les actions. Pour créer les repères, nous pouvons les placer dans le flux vidéo, comme vu pour la navigation ou bien les coder dynamiquement en ActionScript.

L'avantage à le coder repose sur l'idée suivante : un repère d'événements, par rapport au repère de navigation, est indépendant du flux vidéo. Si un repère de navigation doit être dans la vidéo, parce qu'il cible une image-clé précise de la vidéo, le repère d'événements, lui, vise une action, par nature indépendante de la vidéo. Les repères d'événements n'ont donc aucune raison d'être encodés dans le flux vidéo. En somme, les repères d'événements agissent un peu comme des drapeaux (*flags* ou labels), des déclencheurs lus par la tête de lecture en même temps que l'image à laquelle ils sont attachés, et ce, quelle que soit la teneur du flux vidéo. Il devient donc plus judicieux de les traiter en ActionScript, et non en dur dans la vidéo. Ainsi, nous pouvons utiliser une vidéo au format F4V, de meilleure qualité, et non se limiter uniquement au format FLV, qui est le seul à gérer les points de repère.

Dans cette section, nous allons voir comment associer à un composant qui cible une vidéo enregistrée en F4V, des repères codés en ActionScript. Dans notre exemple, les actions que nous associons aux repères permettent de repositionner un MovieClip blanc sous la vignette qui correspond à la séquence en cours de visualisation. À mesure que la vidéo progresse, la vignette est repositionnée, à l'aide d'une interpolation de type `TweenMax`, sur la vignette suivante, et ce, jusqu'à la dernière.

Exemples > ch8_videoInteractive_7.fla

Le document "ch8_videoInteractive_7.fla" contient un composant qui appelle la vidéo "gKaster-demoreel.f4v". Sous le composant, figure un menu composé de MovieClip possédant tous un nom d'occurrence. L'un d'entre eux, situé en arrière-plan, signale la progression de la lecture. Il est placé sous la première image qui matérialise la première séquence du flux vidéo (voir Figure 8.14 et 8.15).

Figure 8.14
Aperçu du document.

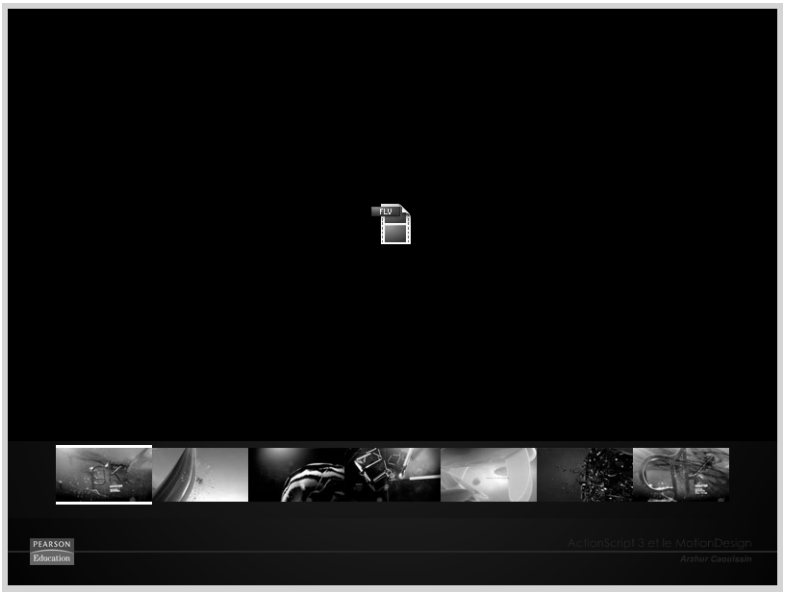
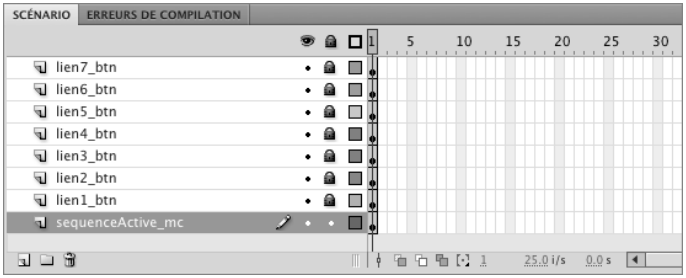
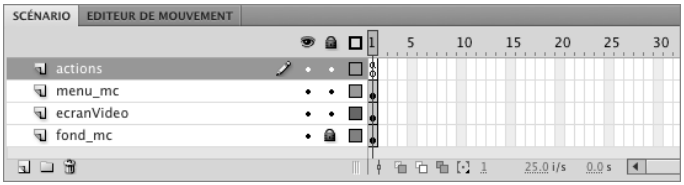


Figure 8.15
Fenêtre de scénario du symbole menu.



Dans le scénario, au-dessus du calque `fond_mc`, nous retrouvons le composant et le menu (voir Figure 8.16).

Figure 8.16
Fenêtre de scénario de la scène principale.



Dans la fenêtre Actions, apparaît le code suivant :

```
// importation des classes
import fl.video.MetadataEvent;
import gs.*;
import gs.easing.*;
import gs.events.*;

// création des points de repère
ecranVideo.addASCuePoint(0,"repere1");
ecranVideo.addASCuePoint(13.14,"repere2");
ecranVideo.addASCuePoint(17.27,"repere3");
ecranVideo.addASCuePoint(41.14,"repere4");
ecranVideo.addASCuePoint(65.10,"repere5");
ecranVideo.addASCuePoint(89.26,"repere6");
ecranVideo.addASCuePoint(118.07,"repere7");

// traitement des actions
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,ecouteRepere);
function ecouteRepere(evt:MetadataEvent) {
   if (evt.info.name=="repere1") {
      TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-300, ease:Strong.easeInOut});
   }
   if (evt.info.name=="repere2") {
      TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-200, ease:Strong.easeInOut});
   }
   if (evt.info.name=="repere3") {
      TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-100, ease:Strong.easeInOut});
```

```
    }
    if (evt.info.name=="repere4") {
       TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:0, ease:Strong.easeInOut});
    }
    if (evt.info.name=="repere5") {
       TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:100, ease:Strong.easeInOut});
    }
   if (evt.info.name=="repere6") {
       TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:200, ease:Strong.easeInOut});
    }
    if (evt.info.name=="repere7") {
       TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:300, ease:Strong.easeInOut});
    }
    trace(ecranVideo.playheadTime)
}
```

En premier, nous importons les classes nécessaires. La classe `MetadataEvent` permet de gérer les métadonnées liées à un contenu vidéo. Les classes `Greensock gs` permettent de créer les animations `TweenMax`, comme vu au Chapitre 2 :

```
// importation des classes
import fl.video.MetadataEvent;
import gs.*;
import gs.easing.*;
import gs.events.*;
```

À la suite, nous définissons les points de repère, un à un. Il y en a sept. Chacun d'entre eux est structuré ainsi :

```
ecranVideo.addASCuePoint(0,"repere1");
```

Nous ciblons d'abord le composant auquel ces repères doivent être associés avec le nom du composant vidéo (`ecranVideo`). Puis, nous ajoutons un repère ActionScript avec la méthode `addASCuePoint`, méthode à l'intérieur de laquelle deux paramètres sont attendus dont le timecode (en secondes) et son nom (une chaîne de caractères). Décliné pour chaque vignette, nous obtenons :

```
// création des points de repère
ecranVideo.addASCuePoint(0,"repere1");
ecranVideo.addASCuePoint(13.14,"repere2");
ecranVideo.addASCuePoint(17.27,"repere3");
ecranVideo.addASCuePoint(41.14,"repere4");
ecranVideo.addASCuePoint(65.10,"repere5");
ecranVideo.addASCuePoint(89.26,"repere6");
ecranVideo.addASCuePoint(118.07,"repere7");
```

Une fois les repères ajoutés, il reste à y associer des actions. Nous plaçons pour ce faire un écouteur sur le composant `ecranVideo`, composant sur lequel nous venons d'attacher les repères en ActionScript. Nous faisons directement référence aux points de repère véhiculés par cette classe, avec la propriété `CUE_POINT` :

```
// traitement des actions
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,ecouteRepere);
function ecouteRepere(evt:MetadataEvent) {
   // actions sur chaque point de repère
}
```

À l'intérieur du bloc d'instruction de la fonction, nous plaçons les actions à exécuter à chaque fois que l'écouteur détecte un point de repère, quel qu'en soit le nom, quelle qu'en soit

la nature (événement ou navigation). Pour distinguer une action d'une autre, nous utilisons une structure conditionnelle qui permet de spécifier l'action à conduire en fonction du nom de chaque point de repère. Ce qui donne :

```
// traitement des actions
ecranVideo.addEventListener(MetadataEvent.CUE_POINT,ecouteRepere);
function ecouteRepere(evt:MetadataEvent) {
   if (evt.info.name=="repere1") {
      TweenMax.to(menu_mc.sequenceActive_mc, 2, {x:-300, ease:Strong.easeInOut});
   }
}
```

Si, le nom de l'objet (`evt.info.name`), identifié par l'écouteur, correspond à une certaine valeur, alors, dans cet exemple, une interpolation de type `TweenMax` est exécutée et déplace le MovieClip blanc à une abscisse déterminée, qui correspond à la position courante de la vignette. Nous déclinons ce principe pour chacun des points de repère.

À la fin de la fonction, une instruction permet de vérifier la position du timecode au moment où l'action est exécutée, avec la méthode `playheadTime`, que nous avons déjà abordée pour gérer l'accélération et le rembobinage en début de chapitre :

```
trace(ecranVideo.playheadTime)
```

Nous avons utilisé dans cette section une instruction pour générer dynamiquement des points de repère et permettre l'utilisation du format F4V. Mais si la vidéo était au format FLV et contenait ses propres repères, fussent de navigation, la fonction que nous avons développée fonctionnerait également. Les instructions liées à la création des points de repère en ActionScript seraient simplement inutiles. La fonction, en effet, que les repères soient inclus dans les métadonnées de la vidéo ou créés dynamiquement en ActionScript, agit de la même manière.

À retenir

- *Il est possible de créer un système de navigation précis en utilisant des points de repère de navigation encodés dans une vidéo FLV.*
- *Pour naviguer dans des points de repère encodés dans les métadonnées de la vidéo, nous utilisons la méthode* `seekToNavCuePoint();`.
- *Les points de repère encodés dans un flux F4V ne sont pas pris en compte.*
- *Il est possible de gérer des événements ActionScript sur une vidéo F4V à partir de points de repère ajoutés dynamiquement avec ActionScript.*
- *Il est possible de générer des points de repère depuis l'Inspecteur de composants, dans l'onglet paramètres, avec la propriété* `cuepoints`*, de la même manière que nous le ferions dans Adobe Media Encoder.*

Lire une vidéo en arrière

Lire une vidéo Flash en arrière avec fluidité peut être intéressant dans le cadre d'une présentation interactive graphiquement élaborée. Par exemple, un fabriquant de voitures, un

bagagiste de luxe, un joaillier, peuvent vouloir valoriser leurs créations dans une mise en scène fine réalisable uniquement en vidéo (lumières, détails, animation 3D avec radiosité et vélocité, etc.). En permettant alors à l'utilisateur de prendre la main sur le flux vidéo pour le laisser se déplacer librement autour de l'objet filmé ou modélisé, celui peut contrôler le sens de défilement de la tête de lecture dans la vidéo, comme s'il naviguait dans un espace réel.

Tant que l'utilisateur déplace la tête de lecture vers l'avant, la vidéo reste fluide. Mais, lire un flux vidéo en arrière aboutit généralement à une lecture saccadée de l'image, quelle que soit le type de programmation utilisé, ceci même avec des fichiers encodés avec 100 % d'images-clés, même en jouant avec deux fichiers dont un projeté à l'endroit et l'autre à l'envers. Le repositionnement aléatoire et de vitesse variable du pointeur, contrôlé par l'utilisateur, implique une autre solution.

La solution la plus fluide et la plus simple à déployer pour gérer une lecture vidéo avant et arrière, consiste à intégrer le flux vidéo physiquement dans le document Flash et à en contrôler la lecture à l'aide d'un gestionnaire de type `ENTER_FRAME`. Pour rendre la lecture plus fluide, nous mettons également à jour l'affichage à chaque itération.

L'intégration d'une vidéo à l'intérieur d'un document Flash appelle cependant quelques recommandations :

- La vidéo doit de préférence être encodée en FLV et avec un nombre d'images-clés intégral (1 image-clé par image). Le FLV semble en effet plus permissif à la lecture arrière sur des images-clés pleines que ne l'est un F4V.
- L'intégration d'une vidéo dans Flash est techniquement limitée à 16 000 images, mais elle doit proposer un temps de chargement honorable. Donc, pensez à placer, en amorce du document, une jauge de chargement qui avertisse l'utilisateur sur sa progression (voir Chapitre 5).
- Au-delà de 2 Mo de poids pour votre document Flash, vidéo comprise, réduisez les dimensions de la vidéo. Diminuez sa durée. Compressez davantage son débit.
- Nous ajoutons la méthode `updateAfterEvent()` pour fluidifier la lecture de la scène.

Dans cette section, nous utilisons une vidéo FLV qui représente une galaxie en mouvement. La séquence est courte et dure 179 images (soit 7,16 secondes pour un flux à 25ips). Une fois compressée avec une image-clé par image, le fichier ne pèse que 1,4 Mo. Dans cet exemple, un curseur est positionné sur l'écran pour que l'utilisateur prenne la main sur le contenu et naviguer comme bon lui semble à l'intérieur du flux vidéo.

Exemples > ch8_videoInteractive_8.fla

Dans le document "ch8_videoInteractive_8.fla", un MovieClip intitulé `vidéo_mc` contient une séquence vidéo importée. La séquence vidéo est étendue intégralement dans le scénario de ce MovieClip. Sur la scène principale, le symbole `console_mc` contient un autre MovieClip nommé `curseur_mc` (voir Figure 8.17).

Figure 8.17
Aperçu du document.

Info

Pour intégrer physiquement une vidéo dans le scénario, faites Fichier > Importer > Importer de la vidéo. Puis, cliquez sur Parcourir pour sélectionner le fichier vidéo déjà compressé au format FLV. Sélectionnez l'option Incorporer le fichier FLV dans le SWF et le diffuser dans le scénario. Puis cliquez sur Continuer. Choisissez alors de l'inclure de préférence dans un clip. Puis, confirmez jusqu'à la fermeture de la boîte de dialogue.

Dans le scénario, au-dessus du calque `fond_mc`, nous pouvons identifier les objets clairement répartis sur des calques distincts (voir Figure 8.18).

Figure 8.18
Fenêtre de scénario.

Le calque `actions` affiche le code suivant :

```
//--------------- initialisation
video_mc.stop();
var pourcentage:Number;
var positionTeteDeLecture:Number=0;
var initMouseX:Number=console_mc.curseur_mc.x;

//--------------- scroll
var zoneDeDeplacement:Rectangle = new Rectangle(10,0,780,0);

console_mc.curseur_mc.addEventListener(MouseEvent.MOUSE_DOWN, activerDeplacement);
```

```
function activerDeplacement(evt:MouseEvent) {
   addEventListener(Event.ENTER_FRAME,jouerVideo);
   console_mc.curseur_mc.startDrag(false,zoneDeDeplacement);
   evt.updateAfterEvent();
}

addEventListener(MouseEvent.MOUSE_UP, stopperDeplacement);
function stopperDeplacement(evt:MouseEvent) {
   evt.currentTarget.stopDrag();
   removeEventListener(Event.ENTER_FRAME,jouerVideo);
}

//--------------- jouer la vidéo
function jouerVideo (evt:Event) {
   pourcentage=Math.ceil((console_mc.curseur_mc.x-initMouseX)/(stage.stageWidth/100));
   positionTeteDeLecture=Math.ceil(pourcentage*(video_mc.totalFrames/100))-10;
   video_mc.gotoAndStop(positionTeteDeLecture);
}
```

Pour bien comprendre la mécanique du curseur, nous devons d'abord considérer que le clip qui accueille la vidéo comporte un certain nombre d'images (179). De même, le curseur qui se déplace dans la console est mobile sur une certaine largeur (de 10 à 780 pixels). Nous devons donc, comme nous l'avons fait pour l'ascenseur au Chapitre 2, créer une équation qui convertit la position du curseur en numéro d'image à atteindre.

Le programme démarre avec, dans les premières lignes du code, une instruction qui interrompt la lecture du clip vidéo, avec un `stop`. Nous initialisons ensuite deux variables nombre que nous utilisons plus loin pour le calcul du rapport entre la position du curseur et celle de la tête de lecture à l'intérieur du clip vidéo. La troisième variable, nommée `initMouseX` enregistre la position de départ du curseur, en X :

```
//--------------- initialisation
video_mc.stop();
var pourcentage:Number;
var positionTeteDeLecture:Number=0;
var initMouseX:Number=console_mc.curseur_mc.x;
```

Plus loin, nous définissons les actions qui permettent de gérer le défilement du curseur le long de l'axe représenté, dans notre document, par un filet blanc :

```
//--------------- scroll
var zoneDeDeplacement:Rectangle = new Rectangle(10,0,780,0);

console_mc.curseur_mc.addEventListener(MouseEvent.MOUSE_DOWN, activerDeplacement);
function activerDeplacement(evt:MouseEvent) {
   addEventListener(Event.ENTER_FRAME,jouerVideo);
   console_mc.curseur_mc.startDrag(false,zoneDeDeplacement);
   evt.updateAfterEvent();
}
addEventListener(MouseEvent.MOUSE_UP, stopperDeplacement);
function stopperDeplacement(evt:MouseEvent) {
   evt.currentTarget.stopDrag();
   removeEventListener(Event.ENTER_FRAME,jouerVideo);
}
```

La dernière commande de la fonction `activerDeplacement` utilise la méthode `updateAfterEvent()`. Cette méthode rafraîchit la scène immédiatement après que les autres

instructions ont été exécutées, à savoir, dès que le déplacement du curseur a eu lieu. Cela rend le déplacement de l'objet plus fluide que si nous limitions le rafraîchissement de l'affichage à un taux basé uniquement sur la cadence de la scène (avec `ENTER_FRAME`). De même, grâce à cette méthode, l'effet de déplacement dans la vidéo est un peu plus fluide.

Toujours dans la fonction `activerDeplacement`, une instruction appelle la fonction jouerVideo, détaillée plus bas, qui contrôle la position de la tête de lecture dans le clip vidéo.

Dans la fonction `stopperDeplacement`, nous arrêtons le déplacement du curseur (`stopDrag`) ainsi que le déplacement de la tête de lecture dans le clip vidéo (`removeEventListener`).

Plus bas, la fonction appelée au déplacement du curseur (`jouerVideo`) calcule la position de la tête de lecture, dans le clip `video_mc` :

```
//--------------- jouer la vidéo
function jouerVideo (evt:Event) {
   pourcentage=Math.ceil((console_mc.curseur_mc.x-initMouseX)/(stage.stageWidth/100));
   positionTeteDeLecture=Math.ceil(pourcentage*(video_mc.totalFrames/100))-10;
   video_mc.gotoAndStop(positionTeteDeLecture);
}
```

Dans la fonction `jouerVideo`, la valeur `pourcentage` ramène d'abord sous la forme d'un pourcentage, la position courante du curseur. Mais, au préalable, nous prenons soin de retrancher la valeur véhiculée par la variable `initMouseX`, avant la division. Cela nous permet d'initialiser le pourcentage à la valeur zéro, et non à la valeur exacte de positionnement du curseur, qui n'est pas placé à zéro au démarrage de l'application, mais à 10 pixels dans notre exemple.

La deuxième ligne du bloc d'instruction reprend cette valeur et la multiplie par un coefficient. Ce coefficient transpose le pourcentage, obtenu au-dessus, à l'échelle de la durée de la vidéo. Pour définir ce coefficient, nous prenons le nombre d'images contenues dans le clip vidéo (`totalFrames`) et le divisons par 100. Le numéro d'une image de scénario étant un chiffre entier, nous utilisons la classe `Math.ceil()` pour arrondir ce chiffre. L'image zéro n'existe pas dans un scénario. Cette méthode permet donc aussi d'obtenir une valeur supérieure à zéro et d'atteindre une image qui existe, en toute circonstance.

Une fois la valeur calculée, elle est transmise en paramètre de la méthode `goToAndStop()` qui contrôle la position de la tête de lecture dans le clip vidéo.

À retenir

- *Pour réaliser une interface où l'utilisateur prend la main sur le flux vidéo, sans sautillement ni rupture de flux, nous importons la vidéo dans le Flash en respectant les contraintes de poids qui permettent de préserver l'accessibilité du contenu.*
- *Pour que le défilement soit fluide, nous contrôlons le rafraîchissement de la scène avec la méthode* `updateAfterEvent()`.
- *Pour que le flux d'image ne saute pas, le fichier vidéo doit être encodé en FLV avec 100 % d'images-clés.*