# Partie 2

# Animer des éléments : L'utilisation de TweenMax.

## Introduction

L'un des premiers intérêts de flash est la possibilité de créer des animations utilisant des effets réalistes ou étonnants, la limite étant celle de l'imagination du créateur et des capacités des machines contemporaines. Mais, autant il est simple de créer une animation via le scénario, autant l'utilisation des classes d'animations (Animator et Tween) peuvent s'avérer fastidieuse et surtout peuvent présenter des bugs important, voire dramatique pour le bon fonctionnement de l'animation : Il peut en effet arriver que les animations via la classe Tween peuvent s'arrêter avant leur terme… Par ailleurs la classe Tween ne peut être utilisée que dans un environnement gérée par l'interface auteur ( par opposition avec la création via le kit de développement flex ).

Pour cette raison nous allons donc utiliser une applications d'animation externe et open source extrêmement pratique et simple d'utilisation : TweenMax de la société Greensocks.
TweenMax est une classe d'animation permettant d'agir sur les propriétés principales d'un objet d'affichage, mais qui possède aussi des propriétés avancées permettant d'animer filtres, volume sonore, proposant aussi une alternative simple à des animations complexes (interpolations en courbes, interpolations de couleurs, modifications du texte). Autre intérêt de cette classe : il en existe des versions pour as2 et as3.
D'autres classes d'interpolations sont aussi disponibles en *open source*, telle Tweener de caurina. A noter d'ailleurs que cette dernière possède des distribution dans la langage javascript permettant ainsi des effets d'interpolation directement dans le html…

Je fais le choix de vous faire découvrir TweenMax pour différentes raison : D'une par la classe est développée fréquemment et la dernière version est très récente. Et en particulier : La classe contient quelques propriétés et méthodes spéciales n'existant pas dans Tweener. Enfin TweenMax semble être plus performante en terme de rendu (plus fluide). Notamment dans l'utilisation de nombreuses interpolations.
Enfin une fois habitué à TweenMax, vous n'aurez aucune difficulté à vous adapter à Tweener si besoin en est. Les deux systèmes sont particulièrement similaires dans leur utilisation. Mais certaines fonctionnalités avancées comme les actions sur un champ texte seront payantes chez greensock et on trouvera une fonctionnalité (simple) sur le texte chez caurina qui elle sera incluse dans le paquetage. Il vous faut en vertu de cela pratiques une veille technique afin de pouvoir utiliser le moteur approprié selon vos besoins.

Nous allons maintenant en découvrir les différents aspects du fonctionnement. Avant cela nous allons nous intéresser à un type d'objet particulier : les « classes » statiques.

### Les classe statiques

Le terme statique est un attribut qui peut être placé devant une fonction ou une propriété. Il correspond au mot clé « static ».
Cet attribut signifie, s'il s'agit d'une propriété, que la valeur qu'elle contient sera partagée par l'ensemble des processus qui l'utiliseront. Pour une propriété normale, une propriété donnée ne sera consultable qu'une fois l'objet instancié. Et il ne sera disponible que dans l'instance en

question. Si la propriété est statique elle sera disponible pour chaque instance de la classe quel que soit son ordre d'apparition.

**A quoi ça sert ?**

C'est la seule possibilité de conserver une ou des valeurs quelles que soit l'endroit du code où on les utilise sans avoir à les transmettre d'objet en objet ou via le système évènementiel. Cela se révèle pratique, par exemple pour conserver les informations d'un xml externe pour une galerie photo. D'une manière générale, dès qu'il est question de stocker une données ou un ensemble de données et de la ou les rendre accessibles, de rendre les modifications quel que soit l'objet qui l'emploie.

La classe TweenMax offre deux possibilités pour la création d'interpolation : Il est possible soit de créer une instance soit d'employer les méthodes statiques. En l'occurrence, pour appeler l'une de ses fonctions ou paramètre on utilisera le nom de la classe suivi de la méthode demandée :

```
TweenMax.to (objet,3,{x : 0});
```

Soit nous pouvons récupérer l'instance de l'interpolation et l'utiliser plusieurs fois au cours de l'animation. Ainsi il est possible d'utiliser ces trois notations selon son besoin de contrôle sur l'interpolation et ses habitudes de programmation :

```
var myTween:TweenMax = new TweenMax(mc, 1, {x:100});
TweenMax.to(mc, 1, {x:100});
var myTween:TweenMax = TweenMax.to(mc, 1, {x:100});
```

Nous avons donc la possibilité d'adapter la complexité du code en fonction des besoins réels, ce qui peut représenter un gain de temps non négligeable en production.
On utilisera des instances par exemple dans le cas ou notre animation aurait besoin d'être modifiée en cours d'interpolation par un choix laissé à l'utilisateur par exemple.

```
myTween.updateTo({x :mouseX,y :mouseY},true);
```

Avant d'utiliser TweenMax nous devons passer par son installation au sein de notre environnement de programmation.

**Installation :**

Première chose à faire, récupérer la dernière version
http://www.greensock.com/tweenmax/
Le dossier contient tout le nécessaire pour l'utilisation et la documentation ainsi qu'un générateur de code, très utile pour commencer à se servir de la classe (dossier « demos »).
Vous trouverez aussi les principales en ligne sur la page ci-dessus.

**Activation des classes dans le script :**

Deux possibilités :

Une fois l'archive décompressée, nous récupérons le dossier greensock-as3 puis :

- Nous le plaçons à la racine de l'animation que nous allons créer pour une utilisation ponctuelle. Si vous utilisez flash CS3, il est nécessaire de préciser l'emplacement dans les paramètres d'as3 :
Fichier / paramètres de publication / onglet flash / paramètres d'AS3 puis cibler le dossier greensock-as3.
- Soit nous pouvons l'installer dans le logiciel de manière permanente ce qui permettra ensuite de la réutiliser au gré de nos applications.
Nous le plaçons dans un dossier à part, dans un emplacement stable puis nous allons à l'emplacement :
Modifier / Préférences / ActionScript / Paramètres d'acionScript 3 / puis cibler le dossier greensock-as3.

**Utilisation.**

Nous voici maintenant avec la possibilité d'utiliser la classe dans notre application.
Première chose a faire est d'importer les paquetages nécessaires pour des interpolations de bases :

```
import com.greensock.*;
import com.greensock.easing.*;
```

**Utilisations des paramètres de TweenMax.**

Nous allons effectuer une première interpolation à l'aide de la méthode statique dévolue TweenMax.to(objet, temps,{parametres}).

La méthode prend trois arguments minimum :

- l'objet qui va subir une interpolation
- la durée de l'interpolation en secondes
- un objet - d'où la notation entre crochets et « : » remplacant l'opérateur « = » - contenant, entre autre, les propriétés de l'objet qui doivent être interpolées et leur valeur finale. Entre autre, car cet objet peut contenir nombres d'autres paramètres comme une fonction de rappel en début, fin ou à l'exécution de l'interpolation. Il n'est pas nécessaire de placer les propriétés dans un ordre particulier.

```
TweenMax.to ( objet_a_interpoler, duree_secondes, {prop :valeur, prop2 :valeur} ) ;
```

La syntaxe de cette classe est la même qu'il s'agisse d'as2 ou as3, seules les nom des propriétés doivent être inscrites selon la version d'actionscript.
Nous créons un clip sur la scène auquel nous donnons un nom d'occurrence.
Interpolation d'une propriété, de deux… Comme on le voit, une seule interpolation permet la mise en place de multiples changements.

## Utilisation des fonctions spéciales : l'exemple des courbes.

L'un des aspects le plus intéressants de TweenMax est de pouvoir modifier des paramètres qui dans le strict cadre du langage d'adobe demanderaient des lignes et des lignes de code pour leur mise en oeuvre.

L'une d'elle en particulier va nous permettre de faire suivre à l'objet un trajet non linéaire vers son point d'arrivée, à l'aide des courbes de Bézier. Ce qui peut s'identifier à l'utilisation d'un guide dans le logiciel  adobe Flash ®.
Pour avoir accès à ces fonctions spéciales il faudra parfois les activer avant de s'en servir. Cela se fait au sein d'une fonction unique prenant un tableau en paramètre.

Pour commencer dans le moteur d'interpolation. On importe donc le répertoire des plugins :

import com.greensock.plugins.*;

et à la tête du code on active le plugin concerné (ici les trajectoires via bezier et l'ombre portée).

TweenPlugin.activate([BezierPlugin , DropShadowFilterPlugin]);

Il est possible maintenant d'utiliser ces fonctions, sans quoi un message d'erreur sera émis lors de la compilation.

### Les trajectoires

TweenMax dispose de plusieurs possibilités pour faire emprunter une trajectoire particulière. Les deux emploient les courbes de bezier. La première classiquement courbera l'axe menant de la propriété de coordonnée de départ à celle d'arrivée. Vous pouvez vous rendre à la page de TweenMax. L'exemple vous fournira un code utilisable dans votre application.

« Tweener » propose une petite animation très bien faite, dont vous pourrez vous servir pour créer vos premières trajectoires via ce moteur.
http://tweener.googlecode.com/svn/trunk/examples/bezierMaker_as3_flash9cs3.zip

La seconde possibilité propre à TweenMax : bezierThrough (littéralement « bezier à travers ») est de faire passer l'objet précisément par les coordonnées fournies.

Les courbes sont définies à travers les coordonnées des points de contrôles assemblées dans un objet, le ou les objets sont insérés dans un tableau.

```
TweenMax.to ( objet_a_interpoler, duree_secondes, {x :100,y ,200, bezier :
[{ x : 50, y : 23 }, { x : 510, y : 213 }, { x : 40, y : 13 }] } ) ;
```

Complémentairement il est possible aussi d'orienter automatiquement l'objet dans l'axe de la trajectoire grâce à orientToBezier:true . Il suffit de rajouter la commande dans les propriétés de l'animation :

```
TweenMax.to ( objet_a_interpoler, duree_secondes, {x :100,y ,200, bezier :
[{ x : 50, y : 23 }, { x : 510, y : 213 }, { x : 40, y : 13 }],
orientToBezier:true } ) ;
```

Enfin signalons la possibilité de faire suivre à l'objet une succession de point de coordonnées lors de l'interpolation avec l'objet LinePath2D. L'interpolation suivra alors une trajectoire rectiligne entre les coordonnées fournis.

## Création d'une animation complexe.

Nous savons comment créer une interpolation. Il arrivera fréquemment qu'une seule animation ne soit pas suffisante, nous aurons besoin d'une seconde qui devra s'enchaîner à une première. Il existe pour cela deux possibilités. La première est à réserver à un enchaînement d'interpolations simple. La seconde permet un contrôle avancé des interpolations et de leurs déroulements. Il est possible aussi d'utiliser les deux simultanément a différentes fins.

### Delay

L'ajout de du paramètre delay dans les propriétés de notre interpolation est la méthode la plus simple pour synchroniser  plusieurs instances. « delay » accepte une valeur en secondes. L'interpolation sera mise en attente jusqu'à ce que le delai indiqué en secondes soit atteint. On place simplement la propriété au sein de l'objet du second paramètre. Dans l'exemple suivant, la deuxième interpolation commence 2 secondes après que la première se soit achevée :

```
TweenMax.to ( objet_a_interpoler, 3, {x :100,y ,200, bezier : [{ x :
50, y : 23 }, { x : 40, y : 13 }], orientToBezier:true } ) ;
TweenMax.to ( objet_a_interpoler, 1, {x :500,y ,400, bezier : [{ x :
50, y : 23 }], orientToBezier:true, delay :5 } ) ;
```

### Utilisation des fonctions de contrôle (onComplete, onStart, onUpdate).

Les fonctions de contrôle permettent d'exécuter un bloc de code en début (onStart), ou fin (onComplete) d'animation ainsi qu'a chacun de ses rafraîchissements (onUpdate).
Le système d'évènements lié à TweenMax est plutôt hérité de l'ancienne version d'actionScript. Aussi on pourra trouver diverses écritures correspondant à un passage de fonction en paramètre.

```
TweenMax.to ( objet_a_interpoler, 1, {x :500,y ,400, bezier : [{ x : 50,
y : 23 }], orientToBezier:true,
onComplete :function() :void{trace(« interpolation terminée »)} } ) ;
```

*Conseil : nous arrivons a des paramétrages sui deviennent de plus en plus long à écrire. Pour clarifier votre code, utilisez les retour à la ligne et l'indentation, ainsi l'interpolation précédente sera bien plus digeste présentée de cette manière, nous voyons au passage que le passage à la ligne n'est pas pris en compte par le code comme une fin d'instruction puisqu'il est ici précédé d'une virgule :*

```
TweenMax.to (
            objet_a_interpoler,
            1,
            {
                  x :500,
                  y ,
                  400,
                  bezier : [{ x : 50, y : 23 }],
                  orientToBezier:true,
                  onComplete :function() :void
                  {
                        trace(« interpolation terminée »)
                  }
           }
           ) ;
```

Il est aussi possible bien sur d'ecrire sa fonction en dehors des paramètres. C'est même à conseiller pour la lisibilité des paramètres de l'interpolation :

```
TweenMax.to ( objet_a_interpoler, 1, {x :500,y ,400, bezier : [{ x : 50,
y : 23 }], orientToBezier:true, onComplete : finInterpolation } ) ;
```

```
function finInterpolation() :void{trace(« interpolation terminée »)} ;
```

Dans le cadre d'une classe on peut bien sur mettre cette fonction dans le corps de celle-ci :

```
private function finInterpolation() :void{trace(« interpolation
terminée »)} ;
```

On se trouve dans ce cas dans l'impossibilité d'accéder à l'interpolation à moins qu'une variable declarée dans le corps de la classe lui soit attribuée.
Mais il existe aussi deux autre possibilités : Soit celle de pouvoir passer des paramètres rassemblés dans un tableau au travers des propriétés : onStartParams, onUpdateParams, onCompleteParams.

```
var interpolation : TweenMax  = TweenMax.to ( objet_a_interpoler, 1,
{x :500,y ,400, bezier : [{ x : 50, y : 23 }], orientToBezier:true,
onComplete : finInterpolation, onCompleteParams:[5, mc]});

function onFinishTween(param1:Number, param2:MovieClip):void {
trace("The tween has finished! param1 = " + param1 + ", and param2 = " +
param2);}
```

Soit si l'on veut rester à la même notation que celle du système évènementiel AS3 :
On utilise dans ce cas la propriété onCompleteListener ou onStartListener etc… En donnant le nom de la fonction.  Dans ce cas seulement, on retrouve la notation propre d'as3 en terme d'évènement ;
Un evenement  de type TweenEvent sera émis par l'interpolation ( penser à importer le paquetage com.greensock.events).

```
import com.greensock.*;
import com.greensock.easing.*;
import com.greensock.events.*

private function declencher():void
{
      var interpolation : TweenMax  = TweenMax.to ( objet_a_interpoler, 1,
      {x :500,y ,400, bezier : [{ x : 50, y : 23 }], orientToBezier:true,
      onCompleteListener : finInterpolation});
}

private function finInterpolation (even:TweenEvent):void
{
      var interpolation:TweenMax = even.target as TweenMax;
      // inversion du mouvement :
      interpolation.reverse();
}
```

Nous voyons donc que le moteur de greensock s'adapte très facilement aux habitudes de programmation que nous contracterons avec la pratique, et aux besoins qui peuvent s'avérer différents en terme de complexité.
Il existe bien d'autres méthodes liées à cette classe. Le présent document ne fait que guider vos premiers pas avec ces moteurs d'interpolation, il ne dispense en aucun cas d'étudier le moteur de votre choix pour en tirer le meilleur parti. En cherchant dans cette documentation vous vous apercevrez par exemple qu'il existe une méthode permettant d'arrêter toutes les interpolations, une autre permettant de les reprendre. La plupart des besoins courants seront bien souvent alimentés via une méthode ou un plugin à activer.

**Rappel de notation**

Pour nous raccourcir des lignes de code, nous allons, au long de ce chapitre, utiliser la syntaxe abrégée pour la création des tableaux et objet nécessaires.

Pour les tableaux :

```
var tab :Array = new Array() ;
tab[0] = (« une_valeur ») ;
tab[1] = (« une_autre_valeur ») ;
```

Revient à écrire :

```
var tab :Array = [ « une_valeur », « une_autre_valeur » ] ;
```

On observe que les crochets suffisent à définir un tableau. Les mêmes crochets servent aussi à définir l'index du tableau ou à en récupérer la valeur.

De même pour les objets :

```
var obj :Object = new Object() ;
obj[« x »] = 50 ;
obj[« y »] = 23 ;
```

Revient à écrire :

```
var obj :Object = { x : 50, y : 23 } ;
```

Ici on voit que les accolades suffisent à définir un objet.
Contrairement à la notation régulière, la propriété est définie sans l'aide des guillemets. On lui attribue une valeur à l'aide de l'opérateur « : ».
Les couples propriétés / valeurs sont séparés par des virgules.

Nous aurons durant le chapitres l'occasion de voir des notations telles que :

```
[{ x : 50, y : 23 },{ x : 510, y : 213 },{ x : 250, y : 123 },{ x : 40, y : 13 }]
```

Il s'agit en fait d'un tableau composé de quatre objets informant des propriétés x et y. C'est par exemple la notation que vous trouverez pour définir une courbe dans TweenMax.