Appendice 3

Outils géométriques : Rectangle et Point.

Les outils de géométrie peuvent sembler bien rudimentaire et austère mais il peuvent s'avérer d'une grande utilité lorsqu'il s'agit de manipuler des coordonnées et grandeur. Ils possèdent aussi des méthodes très intéressantes, comme certains tests permettant de vérifier si deux rectangles sont en intersection tout comme le hitTest d'un display Object. Des calcul de distance entre deux points sont possibles aussi. Nous allons nous attacher à décrire certains aspects qui pourront simplifier notre programmation.

L'objet Rectangle.

L'objet Rectangle du package flash.geom est un objet relativement simple, mais qui trouve son utilité dans quelques situations incontournables : Le scrolling de texte ou d'image, le passage en plein écran d'une portion de l'application, très utile en vidéo, d'autres application emploient aussi cet objet… enfin la simplicité de ce type est telle que les accès mémoire seront optimisés si l'on demande à notre application d'aller chercher des informations dans un objet léger, plutôt que dans un héritier de DisplayObject beaucoup plus lourd à manipuler pour le processeur. Nous allons faire connaissance avec cet objet qui peut nous rendre bien d'autres services…

La signature du constructeur de l'objet se présente comme cela :

```
Rectangle (coordonnees_X :Number = 0, coordonnees_Y :Number = 0, largeur :Number = 0 , hauteur :Number=0) : Rectangle
```

Tous les paramètres contiennent une valeur par défaut je peux par conséquent initialiser cet objet sans lui donner de valeur.
On l'instancie de cette manière :

```
import flash.geom.Rectangle ;

var rect :Rectangle = new Rectangle() ;
```

Il peut être parfois utile de pouvoir stocker en mémoire les dimensions et coordonnées initiales d'un objet pour par exemple le replacer après une interpolation.

```
rect.x=20 ;
rect.y=20 ;
rect.width =100 ;
rect.height =100 ;
```

Nous avons paramétré cet objet pour qu'il décrive un carré de 100 pixel sur 100 pixels situé à 20 pixels en x et y du bord de l'espace de coordonnées. Cela peut être une bonne alternative de donner un objet Rectangle en paramètre lors d'un redimensionnement d'une image chargée plutôt que les dimensions du container. La fonction gagnera en souplesse. De plus comme nous allons le voir ici il est extrêmement facile d'initialiser un objet Rectangle à partir d'un DisplayObject celui possédant deux fonctions dédiées pour le faire à notre place :

récupérer les limites d'un DisplayObject

```
getBounds (espace_reference : DisplayObject):Rectangle
getrect (espace_reference : DisplayObject) :Rectangle
```

Tout DisplayObject renvoie avec ces fonctions un objet rectangle instruit avec la taille et les coordonnées de l'objet interrogé par rapport à l'espace de référence passé en paramètre de la fonction.

Exemple créons deux container de base et observons les variantes de nos deux fonctions :

```
var container:MovieClip=new MovieClip();
container.graphics.beginFill(0x000000,0.5);
container.graphics.lineStyle(10,0xCCFF33,0.5);
container.graphics.drawRect(0,0,200,280);
container.x=100;
container.y=100;
addChild(container);

var contenu:MovieClip=new MovieClip();
contenu.graphics.beginFill(0x000000,0.5);
contenu.graphics.drawRect(0,0,70,20);
contenu.x=50;
contenu.y=50;
container.addChild(contenu);


var rect_cont:Rectangle = container.getRect(stage)
trace("rectangle container (par getRect) : "+rect_cont.toString());
// renvoie : rectangle container (par getRect) : (x=100, y=100, w=200, h=280)
// ne prend pas en compte la ligne de décoration.

var rect_bounds:Rectangle = container.getBounds(stage)
trace("rectangle container (par getBounds) : "+rect_bounds.toString());
// renvoie : rectangle container (par getBounds) : (x=95, y=95, w=210, h=290)
// la ligne chevauche de 5 pixel l'intérieur et l'exterieur du graphique
// get bounds le prend en compte en montre qu'il fait 10 pixel de plus de chaque
coté
// et que le chevauchement le décale de cinq pixels en y et x.

var rect_local:Rectangle = contenu.getBounds(container)
trace("rectangle contenu (local) : "+rect_local.toString());
// renvoie rectangle contenu (local) : (x=50, y=50, w=70, h=20)
// Les coordonnées sont ici passées en coordonnées locales.

var rect_global:Rectangle = contenu.getBounds(stage)
trace("rectangle contenu (global) : "+rect_global.toString());
// renvoie rectangle contenu (global) : (x=150, y=150, w=70, h=20)
// Les coordonnées sont ici passées en coordonnées globales contenu se
// situe bien à ces coordonnées par rapport à la scène.
```

Un Rectangle obtenu à partir d'un DisplayObject propose certaines propriétés qui peuvent nous aider dans un calcul de placement :

```
var rect_global:Rectangle = contenu.getBounds(stage)
// renvoie rectangle contenu (global) : (x=150, y=150, w=70, h=20)
```

Si nous interrogeons les propriétés suivantes :

rect_global.right : nous donnera une valeur correspondante aux coordonnés x du coté droit de l'objet. Soit ici 220.

rect_global.left : nous donnera une valeur correspondante aux coordonnés x du coté gauche de l'objet. Soit ici 150. (équivalent à contenu.x si son parent est le même que l'espace mis en paramètre dans la fonction getBounds).

rect_global.left : nous donnera une valeur correspondante aux coordonnés y du bas de l'objet. Soit ici 170.

rect_global.bottomRight : nous renvoie un objet Point contenant les coordonnées x et y du de l'angle en bas à droite de « contenu ». Nous allons voir par la suite en quoi cet objet Point peut s'avérer utile.

La méthode scrollRect

Tout DisplayObject offre parmi ses méthodes la méthode scrollRect qui agit un peu comme un masque mais qui possède ses propres particularités.
Pour initialiser cette méthode il faut fournir en paramètre une instance d'objet Rectangle qui fournira un cadre en dehors duquel l'objet sur lequel nous appliquons cette méthode sera invisible.
La différence avec un masque est qu'il n'est pas possible de fournir une autre forme qu'un carré.

Enfin il ne faut JAMAIS placer un objet masqué à l'aide de setMask à l'intérieur d'un conteneur auquel un scrollRect est appliqué. Cela fera planter non seulement votre swf mais aussi le logiciel dans lequel s'exécutera l'application (soit flash ou le navigateur)

Naviguer dans une image plus grande que le cadre :

```
		// déclarations de variables
		var layer_photo:MovieClip;
		// le rectangle qui servira de reférence pour la navigation dans
l'image.
		var limites_appli:Rectangle;
		// le container sur lequel nous allons appliquer la méthode scrollRect
		var support_image:MovieClip = new MovieClip();
		// vitesse de defilement du gestionnaire de mouvement
		var speed: Number = 10;

		// je verifie que l'objet stage est accessible
		if(stage)
		init()
		else
		this.addEventListener(Event.ADDED_TO_STAGE,init);
		// sinon je crée un écouteur pour m'en assurer

		function init(even:Event=null):void
		{
			// si l'ecouteur à été place sur la scène (this) je l'enleve
			if(this.hasEventListener(Event.ADDED_TO_STAGE))
			this.removeEventListener(Event.ADDED_TO_STAGE,init);

			//j'instancie la photo qui se trouve dans la bibliothèque de mon
appli
			layer_photo = new photo();
			// je crée un rectangle aux dimensions du swf
			var rect_scene : Rectangle = new
Rectangle(0,0,stage.stageWidth,stage.stageHeight);
			// je clone le rectangle pour m'en servir de référence pour la
			// navigation:
			// Pour voir le bas droit de l'image la souris devra en bas et à
			// droite de la scene
			limites_appli = rect_scene.clone();
			// je diminue les cotés du rectangle de 50 pixels chacun pour
```

```
                        // avoir une fênetre plus petite
                        rect_scene.inflate(-50,-50);

                        this.addChild(support_image);
                        // j'applique le rectangle en tant que masque.
                        support_image.scrollRect= rect_scene;

                        support_image.addChild(layer_photo);

                        // je place mon container image aux coordonnées du rectangle.
                        // le clip se trouve bien centré car la
                        // méthode rectangle.inflate() diminue ou élargit
                        // les bords par rapport au centre.
                        support_image.x = rect_scene.x;
                        support_image.y = rect_scene.y;

                        // je place un ecouteur sur le parcours du scenario par la tête
de lecture.
                        this.addEventListener(Event.ENTER_FRAME, mouvement);
                }

                function mouvement(even:Event):void
                {
                                // L'algorithme gerant le deplacement par rapport à la
souris
        layer_photo.x += (-((mouseX)*((layer_photo.width/limites_appli.width) *(1-
(limites_appli.width/ layer_photo.width))))-( layer_photo.x))/speed;
        layer_photo.y += (((mouseY)*((layer_photo.height/limites_appli.height)* (1-
(limites_appli.height/ layer_photo.height))))-( layer_photo.y))/speed;
                }
```

L'objet Point.

L'objet Point est encore plus simple que l'objet rectangle. Il s'agit simplement d'une paire de coordonnées x et y.

On l'instancie de cette manière :

```
var point_a=new Point (200,200);
```

point.x et point.y renvoient 200.

Je viens donc de définir un simple point se situant au croisement d'une abscisse et d'une ordonnée de 200 pixels chacun.

Deux propriétés interéssantes

```
Point.distance(point_a : Point , point_b :Point ) : Number
```

Cette méthode me permet de connaître la distance en pixels entre deux points. Et puisque nous pouvons, comme nous venons de le voir, récupérer facilement les coordonnées de deux points à partir d'éléments sur la scène, calculer la distance entre eux, même s'ils ne sont pas alignés devient aisé

J'obtiendrai avec ces paramètres la distance entre le coin bas-droit de l'objet 1 et le coin haut-gauche de l'objet 2

```
var distance :Number = Point.interpolate(objet_1.getBounds(stage).bottomRight,
objet_1.getBounds(stage).topLeft);
```

J'utilise ici la possibilité que m'offre as3 de chainer les différentes méthodes en une seule instruction.

```
Point.interpolate(point_a : Point , point_b :Point, ratio :Number
) : Point
```

Cette méthode me renvoie un objet Point. Ce point est celui qui se trouve entre les deux premiers entrés en paramètres, modérs par le paramètre ratio (qui est un décimal compris entre 0 et 1).

- ratio = 0 : les coordonnées seront celle du point b
- ratio = 1 le Point renvoyé sera équivalent aux coordonnées du point a.
- ratio = 0.5 le point renvoyé sera équidistant aux deux autres.

Pour un exemple de son utilisation reportez vous au dernier code du chapitre 6 sur les interpolations.