Archived:Basic Dynamics in Flash Lite

Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

We do not recommend Flash Lite development on current Nokia devices, and all Flash Lite articles on this wiki have been archived. Flash Lite has been removed from all Nokia Asha and recent Series 40 devices and has limited support on Symbian. Specific information for Nokia Belle is available in Flash Lite on Nokia Browser for Symbian. Specific information for OLD Series 40 and Symbian devices is available in the Flash Lite Developers Library.

This code snippet covers the logic and programming involved in creating simple animations dynamically in Flash Lite.

Introduction

There are several Physics engines in Flash that help in emulating the 2D or 3D behaviour of objects using Actionscript. Few of the popular physics engines are Sandy 3D and Papervision (for Flash). However, the above two engines are designed primarily for 3D design and hence, require lot of rendering to be done. In simple words, the complexity involved is high and might bog down the mobile phone. In this article, simple 2D animations are presented along with their associated code logic.

Linear Movement

In order to move an object dynamically using Actionscript, one needs to convert the moving object into a MovieClip. This is done because only MovieClips can be referenced from the code and not simple Graphical elements. Once, we create a MovieClip instantiate the same by giving it an apt name. You shall refer to this object only using this name, in the code. In this case, I have drawn a simple circle and converted it into a MovieClip. I have instantiated the converted object as my_mc. In order to move the object horizontally, use the following code.

```
my_mc.onEnterFrame = function()
{
this._x += 1;
}
```

Note that this has to be added to main timeline and the stage needs to be populated with this object. When you compile and run the code you shall observe that the circle moves as its x co-ordinate is incremented.

Stage wrap

If you want the stage walls to be wrapped up, then modify the code as follows

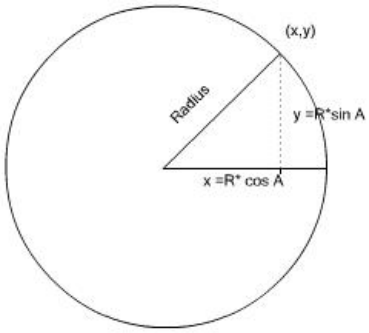
```
my_mc.onEnterFrame = function()
{
if ( this._x < Stage.width) { this._x += 1; }
else { this._x = 0; }

}
```

If you wish to increase the speed of movement, change the incremental value.

Circular movement

Moving an object over a circular path is slightly more complex and is explained below. We need to understand the polar coordinate system to do this. From basic geometry, we learn that for a circle, any point on its locus follows the equation

(C) Copyright Nokia 2014. All rights reserved.



X = R cos (A); Y = R sin (A) ;

Where A is the angle made by the point at its center and R is the radius of the circle. With this understanding, we shall now animate an object to move under a circular path.

```
var angle = 0;
onEnterFrame = function ()
{
var xPos = Math.cos( angle * Math.PI/180 ) * Radius;
var yPos = Math.sin( angle * Math.PI/180 ) * Radius;
obj_mc._x = xPos + xCenter;
obj_mc._y = yPos + yCenter;
angle += 2;
if( angle >= 360 ) angle = 0;
}
```

Let us assume that Radius denotes the radius of the circular path and xCenter and yCenter denote the co-ordinate values of the center. The object which moves on a circular path is obj_mc as per above code. In this case, unlike the previous, we change the angle made by the object. Before supplying the angle to cos function we convert it into radians.

Other Curves

From the above two examples, we can understand that objects can be moved around any locus, once we know the equation of that curve. Curves likes ellipses or asteroids have complex equations, but the logic is the same.

Collision Reaction on walls

Most leisure games that are played today, say Paddle Tennis or even Snooker, involve collision of an object against the surface on the wall and then bouncing back. The same effect can be achieved by doing a small brush-up of our high school Physics.

We must remember that at the point of collision the angle of incidence equals the angle of reflection(non-inclined wall - which has slope in multiples of 90 degrees). The angles or slope in our case is change in Y coordinate and the X coordinate. The slope is preserved, but the direction is changed. We can simply negate the suitable axis. For example, a collision on the rightmost edge (for an object from its left) of the Stage results in negation of X velocity but the Y velocity component remains the same. This is an ideal scenario where friction is assumed to be null. The code below demonstrates the motion of an object as it collides with the 4 walls or Stage boundaries. Place a movieClip named mc and place on the code on the Frame.

```
xVel = yVel=4;
onEnterFrame = function () {
 mc._x += xVel;
 mc._y += yVel;
 if (mc._x>=Stage.width-mc._width) {
  xVel = -xVel;
 }
```


(C) Copyright Nokia 2014. All rights reserved.

```
 if (mc._x<=0) {
  xVel = -xVel;
 }
 if (mc._y>=Stage.height - mc._height) {
  yVel *= -1;
 }
 if (mc._y<=0) {
  yVel *= -1;
 }
};
```

BOUNCING BALLS

This is also an extrapolated scenario of the previous which involves collision against the walls (or Floor – to be specific). As the balls bounces on the floor, it retires a little, thereby loosing some energy. Hence as the ball goes upward after a bounce the height reached by the ball reduces. This happens until the retires completely (looses all energy) and lies flat on the ground. In this case, friction needs to be considered.

The Movie Clip (mc) is placed on the Stage and is made to free-fall as the code begins to run. The variables FRICTION_COEFF and BOUNCE_LEVEL need to be a fraction less than 1. To get practical and realistic effects, set BOUNCE_LEVEL less than FRICTION_COEFF. The variable, dir, is used to indicate the direction of motion of the object. If dir equals 1 it is descending, else raising. The variable VERY_LOW_VALUE needs to be very low as such as .25. There is no physical reference to this variable, but is used to delete the computation involved once the ball stagnates on the ground.

```
var SPEED = 10;
yVel=SPEED;
ht = Stage.height-mc._y;
dir = 1;
onEnterFrame = function () {
 if (yVel<VERY_LOW_VALUE) {
  delete onEnterFrame;
 }
 if (dir == 1) {
  mc._y += yVel;
  if (Stage.height<=(mc._y+mc._height)) {
   dir = 0;
  }
 } else {
  dec = yVel*.FRICTION_COEFF;
  mc._y -= dec;
  if ((Stage.height-mc._y>=BOUNCE_LEVEL*ht)) {
   dir = 1;
   yVel = dec;
   ht = ht*BOUNCE_LEVEL;
  }
 }
};
```

As you have learnt the rudimentary level of Game Physics you are now set to create a simple game like Paddle tennis.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.