Copyright 2005 by Scott T. Leutenegger

Notes 15: Depths and Layers

Remember the syntax for attachMovie: attachMovie(string,string,number) ? All that has been said about this up to now is that the first argument must be the linkage-name of the MovieClip in the library, the second must be a unique string, and the third a unique number. In class I vaguely stated that the depth number is a depth, sort of like a layer level but not quite. Now you get the full story as best as I know it.

Inside flash every object instance is given a unique depth and unique identifier. Those instances can be created within layers using the flash GUI, or, they can be created and attached dynamically from within actionscript (with commands like attachMovie, duplicateMovie, createTextField, and createEmptyMovieClip). Instances with a greater depth are drawn on top of instances with lower depth. Thus, the instance with the highest depth will always be visible if located on the screen, whereas a lower depth instance could be fully or partially covered by another instance. The instances created within the layers using the drawing tools are given depths within the range –16,384 to –1. Instances created dynamically from actionscript are given depths 0 to 1,048,575. There is actually another range of number, greater than 1,048,575, that are used for temporary storage of MovieClips but you probably will never need to use these depths, instances here do NOT show up on the timeline and you cannot remove them from within actionscript.

Instances within the timeline layers are numbered with the lowest layer getting the lowest depths, hence, placing your background image in the lowest layer will guarantee that dynamic objects will appear on top of the background. (Well, actually, you CAN specify negative depths when creating dynamic MovieClips, but that usually does not make sense, stick to positive depth numbers and you will be fine.).

Flash takes care of the depth numbering of the instances you create in the levels using the drawing tool. We will say no more about them for now and focus on the instances we dynamically create from within actionscript.

Consider the code in our first example. This example makes concrete the effect MovieClip depth has on visibility.

a_depthExample_1.fla

var frameCount:Number = 0 ; // used to keep running frame number count

// following attaches MC at depth 0
_root.attachMovie("redRect", "rr", 0) ;
_root.rr._width = _root.rr._height = 100 ;
_root.rr._x = Stage.width / 2 ;
_root.rr._y = Stage.height / 2 ;

// following attaches MC at depth 1
_root.rr.attachMovie("yellowRect", "yr", 1) ;
_root.yr._width = _root.yr._height = 100 ;
_root.yr._x = Stage.width / 2 + 50 ;
_root.yr._y = Stage.height / 2 ;

// following attaches MC at depth 2
_root.rr.attachMovie("greenRect", "gr", 2) ;
_root.gr._width = _root.gr._height = 100 ;
_root.gr._x = Stage.width / 2 ;
_root.gr._y = Stage.height / 2 + 50 ;

// following attaches MC at depth 3
_root.rr.attachMovie("blueRect", "br", 3) ;
_root.br._width = _root.br._height = 100 ;
_root.br._x = Stage.width / 2 + 50 ;
_root.br._y = Stage.height / 2 + 50 ;

onEnterFrame = function () {

    frameCount++ ;
    if (frameCount > 50 ) {
        frameCount = 0 ;
        // following swaps the deptsh of MCs rr and gr
        _root.rr.swapDepths(br) ;
    }
}

Run the above (F12) and see what happens. Four movie clips are placed on the screen. The four are colored red, yellow, green, and blue. They are overlapping and assigned depths of red==0, yellow==1, green==2, and blue==3. Every 50 frames the depths of the red and blue MovieClips (rectangles) are swapped using the swapDepths command.

Note, the command:

    _root.attachMovie("redRect", "rr", 0) ;

could also be written as either of:

    attachMovie("redRect", "rr", 0) ;
    var tempMC:MovieClip = attachMovie("redRect", "rr", 0) ;
with the same affect. The only difference between the first two is the _root is made explicit in the first statement, and the only difference between the first and third as that in the third the variable tempMC is created as assigned the movieClip IN ADDITION to the MovieClip being assigned to _root.

Notice that we use the internal names to access the MovieClip properties. Commands like:

```javascript
_root.rr._x = Stage.width / 2 ;
```

are the same as we have done before, only we are explicitly using the internal MovieClip name, “rr”, instead of assigning the MovieClip to a temporary MovieClip variable and accessing the MovieClip through that variable.

To see another example of this (as well as directly accessing depths), look at a_depthExample_2.fla. Note, to see the trace output, run this with ctrl-Enter.

**a_depthExample_2.fla:**

```javascript
var frameCount:Number = 0 ; // used to keep running frame number count
var tempMC:MovieClip ;

// following attaches MC at depth 0
tempMC = attachMovie("redRect", "rr", 0) ;
rr._width = rr._height = 100 ;
rr._x = Stage.width / 2 ;
rr._y = Stage.height / 2 ;

// following attaches MC at depth 1
attachMovie("yellowRect", "yr", 1) ;
yr._width = yr._height = 100 ;
yr._x = Stage.width / 2 + 50 ;
yr._y = Stage.height / 2 ;

// following attaches MC at depth 2
attachMovie("greenRect", "gr", 2) ;
gr._width = gr._height = 100 ;
gr._x = Stage.width / 2 ;
gr._y = Stage.height / 2 + 50 ;

// following attaches MC at depth 3
attachMovie("blueRect", "br", 3) ;
br._width = br._height = 100 ;
br._x = Stage.width / 2 + 50 ;
br._y = Stage.height / 2 + 50 ;
```
trace("gr is at depth " + gr.getDepth() );
trace("rr is at depth " + rr.getDepth() );
trace("yr is at depth " + yr.getDepth() );
trace("br is at depth " + br.getDepth() );
trace("theTextBox is at depth " + theTextBox.getDepth() );

trace("Next highest depth is " + _root.getNextHighestDepth() );

// following attaches MC at depth 8
attachMovie("blueRect", "br2", 8);
br2._width = br2._height = 100;
br2._x = Stage.width / 2 - 150;
br2._y = 0;

trace("Next highest depth is " + _root.getNextHighestDepth() );

onEnterFrame = function () {
    frameCount++;

    // the following show two ways we can access the movie clip
    if (frameCount == 20)
        tempMC._x += 100; // by the var we created for it
    if (frameCount == 40) {
        _root.rr._x -= 100; // by the internal name we created
        // note, in the above, you can leave off _root. if not
        // there _root. is implicit
        frameCount = 0;
    }
}

We create four MovieClips as in the proceeding example and then print out their depths using the trace command. Yep, the depths match the depths we put in as arguments to attachMovie(). Notice we also print out the depth of the instance “theTextBox” and it is –16382. Instance “theTextBox” was created as a dynamic text box in the layer and in the properties panel the “instance” box was set to “theTextBox”. This is the internal name for this instance. If you do not specify an instance name I am not certain what happens, but I conjecture either it is left undefined or the system generates some unique identifier for it.

Notice the use of function getNextHighestDepth(). This is a function that returns an integer that is one larger than the largest depth number of instances created so far. To demonstrate it more, I create an instance with depth 8, and then call getNextHighestDepth() which returns 9 as it should. What is useful here, is this function can be used to help create unique depth numbers and internal names.
So why is that important? Because we will get unexpected results if you just assign depths at random. When an instance is created at depth D, if an instance already exists with depth D, the existing instance is removed to make room for the new instance! Run the following (F12) and open up and read the code:

*a_depthExample_3.fla*

Specific comments:

1) Four MovieClips are created and given names/depths of (b0,0) (b1,1) (b2,2) and (b3,3) by the loop:

```javascript
// The balls are attached at depths 0 .. numBalls
for (var i:Number = 0 ; i < numBalls ; i++) {
  balls[i] = attachMovie("ball", "b"+i, i);
  balls[i]._xscale = 35 ;
  balls[i]._yscale = 35 ;
}
```

2) The balls have there (x,y) coordinates initialized so they are in a diagonal row.

3) On the 100th frame entry, a new MovieClip, redRect, is attached at depth 2. Because a ball exists at depth 2, the ball “b2” is removed to make way for this red rectangle.

```javascript
if (frameNum == 100) {
  _root.attachMovie("redRect", "rr", 2) ;
  // the above REPLACES the MC attached at depth 2
  _root.rr._width = _root.rr._height = 70 ;
  _root.rr._x = Stage.width / 2 ;
  _root.rr._y = 300 ;
}
```

4) On frame 200 another MovieClip is attached at depth 3, thus removing ball “b3”.

```javascript
if (frameNum == 200) {
  _root.attachMovie("yellowRect", "yr", 3) ;
  // the above REPLACES the MC attached at depth 3
  _root.yr._width = _root.yr._height = 70 ;
  _root.yr._x = Stage.width / 2 ;
  _root.yr._y = 200 ;
}
```

5) On frame 300 another MovieClip is attached at depth 0 replacing ball “b0”

6) On frame 400 another MovieClip is attached at depth 2, thus replacing the current MovieClip at that depth (the red rectangle internally name “rr”).
7) Finally, there is the usual “wall-reflecting” ball bounce code, but since there is only one set of velocities \((x_v,y_v)\), when one ball hits ALL are reflected even though they have not hit yet. Clearly we need a better way to deal with this.