Notes 3: Actionscript to control symbol locations

Okay, you now know enough actionscript to shoot yourself in the foot, especially if you don’t use types. REMEMBER to always declare vars and specify data types. But the code is just sorta what you are used to, time to use it for something fun.

Lets combine flash and actionscript to move some stuff around on the screen. Do the following step by step and then we will explain:

1) Create a new document
2) Draw a filled circle
3) Select the circle in convert to a symbol by write clicking on it
4) Choose MovieClip (WAIT!). ALSO, in this menu, within the “linkage” section check the “Export for ActionScript” checkbox. When you do this it highlight another option, nameline the identifier. Type in “ball” in this spot. Also, name the symbol (top of the menu) “ball”.
5) Click okay to close this menu
6) Delete the ball from the screen.
7) Left click in the first frame and open the actionscript window and add the following:

```
var tempBall:MovieClip = attachMovie("ball", "b1", 1);
tempBall._x = 10;
tempBall._y = 200;

onEnterFrame = function () {
    tempBall._x += 3;
    tempBall._y -= 3;
}
```

8) Now test the move (ctrl-enter)

You should see your ball slowly drifting up and to the right until it disappears. If not, go ahead and download my file a2_1.fla.

Here is the explanation. When you convert an object into a symbol, the symbol is place in a “library”. To see this, hit ctrl-L inside of your .fla file to open the library. You will see an object named “ball”. If you click it the ball appears in a new window. To close that window click on the tab named “Scene 1” just below the timeline. So, the library knows of the symbol as “ball”. But for actionscript to be able to access this symbol, it must be “linked”. This is done by the sequence of steps you did in number (4) above. The code in number (7) above simply makes a copy of this symbol (the symbol is of type MovieClip) and attaches it to the local variable “tempBall”. MovieClips have many properties associated with them, two are the x and y locations of the move clip on the stage. The statements tempBall.x = 10 ; and tempBall.y = 200 just set the (x,y) coordinates of the MovieClip to {10,200}. 

NOTE: The coordinate system has an origin (0,0) in the top left corner and coordinate x increases when going right, but coordinate y increases when going DOWN.
So far all we have done is make a copy of the MoveClip and placed it on the screen. The last four lines are what actually make the ball move.

This code using the MovieClip method: attachMovie. The general form is:

```text
myMovieClip.attachMovie(idName, newName, depth)
```

where idName is the linkage name of the movieClip symbol, newName is a unique instance name within the movie, and depth is the equivalent of a layer level. What is important to remember when using attachMovie: a) idName must be the linkage name; b) newName must be unique for every instance you make; and c) depth should also be unique.

The other big new concept in the above code is:

```text
onEnterFrame = function() {…}.
```

This is your first example of event-driven programming. In event drive programming, and event handler is executed every time an event occurs. In this case the event is entering the frame and the handler is the function specified. Pause for a moment and think about a flash animation without any actionscript. Make it really simple, just two frames with a ball where the ball is in a slightly different position in the second frame. When you play such an animation you see the ball going back and forth (check out file backForth.fla). What is really happening is the animation is played from beginning to end and then repeated indefinitely. So, frame 1 is played, then frame 2. Animation ends and is restarted: frame 1, frame 2, end. Frame 1, frame 2, end. And so on. This means that frame 1 is entered, then frame 2 is entered, then frame 1 is entered, and so on. So, what happens if an animation has only one frame? Yep, frame 1 is entered, then frame 1, then frame 1, and so on. The actionscript command, “onEnterFrame” is a command to set up an event handler, the event is entering the frame. So, each time the frame is entered, which we now know is again and again, the code inside the function is executed. Each time the code is executed the _x location of the MovieClip is incremented and the _y location is decremented, resulting in the ball moving.

How fast does the ball move? The answer is as quickly as the frame is entered. In flash this is a property you can set. The default parameter value is 12 frames per second. You can change this in the Properties panel of the main document. Click someplace outside the stage to get the main properties panel. About in the center-top you will see a frame rate box. Go ahead and change this to 24 or 36 and re-run (ctrl-enter) the animation.

Note, if you change the frame rate to 36, but change the increment/decrement to be 1 pixel instead of 3, it is the same as my original a1_1.fla file (or at least it looks the same). It is important to set the frame rate appropriately. If the frame rate is set too high, say 500, and the animation is run on a slower computer, then the computer will not be able to execute all the code (either the automatically generated flash code from drawing and
moving objects/tweens/sounds or your actionscript code) before it is required to move on to the next frame. What does flash do? It just starts dropping frames! This can be quite a problem, for example you might have a frame that deals with the collision reaction from two balls hitting each other. If that frame is dropped the balls might just go right through each other instead of bouncing because the “bounce” code is never executed.

So, what is the right frame rate? There is no easy answer, but the default that I see on the web for games seems to be 24 frames per second. So, I suggest we all use 24 fps. Remember, you have no control over the speed of the computer that someone uses to view your animation/game, hence you need to design for lame machines, not the latest and greatest muscle PC.

Okay, I think that was a fascinating discussion, but I must confess one ball moving across the screen just doesn’t rock my boat. How about two? Try out the following code (available on the web site as a2_2.fla).

```actionscript
var tempBall:MovieClip  = attachMovie("ball", "b1", 1);
tempBall._x = 10 ;
tempBall._y = 200 ;
var tempBall2:MovieClip  = attachMovie("ball", "b2", 2);
tempBall2._x = 100 ;
tempBall2._y = 300 ;

onEnterFrame = function () {
    tempBall._x += 3 ;
    tempBall._y -= 3 ;
    tempBall2._x += 3 ;
    tempBall2._y -= 3 ;
}
```

How about 5 balls? Sure, you can do the following (which is found in a2_3.fla):

```actionscript
var tempBall1:MovieClip ;
var tempBall2:MovieClip ;
var tempBall3:MovieClip ;
var tempBall4:MovieClip ;
var tempBall5:MovieClip ;
tempBall1 = attachMovie("ball", "b1", 1);
tempBall2 = attachMovie("ball", "b2", 2);
tempBall3 = attachMovie("ball", "b3", 3);
tempBall4 = attachMovie("ball", "b4", 4);
tempBall5 = attachMovie("ball", "b5", 5);

tempBall1._x = 10 ;  tempBall1._y = 50 ;
tempBall2._x = 20 ;  tempBall2._y = 100 ;
tempBall3._x = 30 ;  tempBall3._y = 150 ;
tempBall4._x = 40 ;  tempBall4._y = 200 ;
tempBall5._x = 50 ;  tempBall5._y = 250 ;

onEnterFrame = function () {
```
```
Now you have five balls moving to the right. Notice that the five balls move off the screen to the right an never come back. The reason is that one every frame entry we keep incrementing the x location but our screen only goes up to some upper bound (when run on the web it would be the minimum of browser window size and the flash stage size). One common technique is to have objects go off the right (or top) and reappear on the left (or bottom). The following function, found in a2_4.fla, does this:

```
onEnterFrame = function () {
    tempBall1._x += 5 ;
    tempBall2._x += 5 ;
    tempBall3._x += 5 ;
    tempBall4._x += 5 ;
    tempBall5._x += 5 ;

    // The next 5 lines see if the ball is "going off the screen to the right", and
    // if so move them back to location 0. This is a boundary condition example and
    // we will learn to handle this situation more elegantly later.
    if (tempBall1._x >= 400) tempBall1._x = 0 ;
    if (tempBall2._x >= 400) tempBall2._x = 0 ;
    if (tempBall3._x >= 400) tempBall3._x = 0 ;
    if (tempBall4._x >= 400) tempBall4._x = 0 ;
    if (tempBall5._x >= 400) tempBall5._x = 0 ;

}
```

Slap that in your code and try it.

It is rather simplistic to arbitrarily pick x = 400 as the boundary. The normal way is to use the built-in Stage members. The stage is simply an object with members and methods. Two important members are Stage.width and Stage.height. Using Stage.width the code, as fund in a2_5.fla, is:

```
onEnterFrame = function () {
    tempBall1._x += 5 ;
    tempBall2._x += 5 ;
    tempBall3._x += 5 ;
    tempBall4._x += 5 ;
    tempBall5._x += 5 ;

    if (tempBall1._x >= Stage.width) tempBall1._x = 0 ;
    if (tempBall2._x >= Stage.width) tempBall2._x = 0 ;
    if (tempBall3._x >= Stage.width) tempBall3._x = 0 ;
    if (tempBall4._x >= Stage.width) tempBall4._x = 0 ;
    if (tempBall5._x >= Stage.width) tempBall5._x = 0 ;

}
```
Now the balls use up the entire stage width. Note, if you use ctrl-enter to test the movie it does not seem to work. For some reason the test mode does not respect the stage boundaries. Instead, publish the page and preview it in browser. Now it works as you expect.

If your objects are moving to the left you can use 0 as your boundary. Try the following, found in a2_6.fla, (with just two balls to make the code smaller):

```actionscript
var tempBall1:MovieClip;
var tempBall2:MovieClip;

tempBall1 = attachMovie("ball", "b1", 1);
tempBall2 = attachMovie("ball", "b2", 2);

tempBall1._x = Stage.width - 20; tempBall1._y = 50;
tempBall2._x = Stage.width - 30; tempBall2._y = 100;

onEnterFrame = function () {
    tempBall1._x -= 5;
tempBall2._x -= 5;

    if (tempBall1._x <= 0) tempBall1._x = Stage.width;
    if (tempBall2._x <= 0) tempBall2._x = Stage.width;
}
```

Now the move to the left, disappear, and reappear on the right. But wait, they no longer slide off the screen to the left, the touch the left and then “poof”, they are gone. The reason is that a MoveClip._x member is the x coordinate of its lower left corner. When they went off the right side of the screen we waited until the x member was equal to Stage.width, by then the object had slide off the screen. To get the same effect off to the left we need to allow the x coordinate to go to (0 – MC._width). The following code does what we want (found in a2_7.fla):

```actionscript
onEnterFrame = function () {
    tempBall1._x -= 5;
tempBall2._x -= 5;

    if (tempBall1._x <= (0 - tempBall1._width)) tempBall1._x = Stage.width;
    if (tempBall2._x <= (0 - tempBall1._width)) tempBall2._x = Stage.width;
}
```

Usually we associate a trajectory with an object. One way to do this, and deal with boundary conditions on top and bottom, is below (and found in a2_8.fla):

```actionscript
// set up velocities for balls 1 and 2, xv1 means x velocity of ball 1
var xv1 = 6;
var yv1 = -5;
```
var xv1 = -8;
var xv2 = 4;

tempBall1 = attachMovie("ball", "b1", 1);
tempBall2 = attachMovie("ball", "b2", 2);

// put in initial locations
tempBall1._x = Stage.width - 20;  tempBall1._y = 50;
tempBall2._x = Stage.width - 30;  tempBall2._y = 100;

onEnterFrame = function () {
    // update new locations of balls 1 and 2
    tempBall1._x += xv1;
    tempBall1._y += yv1;
    tempBall2._x += xv2;
    tempBall2._y += yv2;
    // trace(tempBall1._x + " " + tempBall1._y);

    // check boundary conditions, if off screen wrap around from the other side
    if (tempBall1._x <= (0 - tempBall1._width)) tempBall1._x = Stage.width;
    if (tempBall1._y <= (0 - tempBall1._height)) tempBall1._y = Stage.height;
    if (tempBall1._x > Stage.width) tempBall1._x = 0;
    if (tempBall1._y > Stage.height) tempBall1._y = 0;
    if (tempBall2._x <= (0 - tempBall2._width)) tempBall2._x = Stage.width;
    if (tempBall2._y <= (0 - tempBall2._height)) tempBall2._y = Stage.height;
    if (tempBall2._x > Stage.width) tempBall2._x = 0;
    if (tempBall2._y > Stage.height) tempBall2._y = 0;

    // Question: what happens in the above two lines if you replace > with >=? Why?
}

Now you may want to add RANDOM motion. Actionscript provides a math library including the function: Math.random(). Math.random() returns a random number between 0 and 1.0. If you want a random integer between 1 and 200 you would want Math.ceil(Math.random() * 200).

Armed with this knowledge, you can now make more interesting random motion.

**Exercise 2.1:** open the file a2_8b.html  Now, write code to mimic this behavior. Hint, you can use a2_8b.fla as a starting point, you need to add random motion by incorporating in calls to Math.random().

**Exercise 2.2:** open the file a2_8c.html  Note how balls BOUNCE off of walls now instead of come out the other side. Write code to mimic this behavior. Hint: an object’s velocity changes when it bounces off of a wall.

Example files from this section of the notes:
a2_1.fla: attach one ball movie clip and move it around on screen
a2_2.fla: Two ball movie clips and move them
a2_3.fla: Five ball movie clips and move them
a2_4.fla: Five ball movie clips movement wrapping around
a2_5.fla: Two balls, using Stage.width
a2_6.fla: Two balls, making move off the left instead of the right
a2_7.fla: Two balls, making move off the left more smoothly
a2_8.fla: Two balls, moving up/down left/right using trajectories
a2_8b.html: Example of movement you are to produce in Exercise 1.1
a2_8c.html: Example of movement you are to produce in Exercise