Chapter 6: Lots-O-Gargoyles: For-loops and Arrays

Time to start adding more assets to our games. In chapter 4 we saw how to create two moving gargoyles by duplicating the code, but what if we want 100 moving objects? Certainly we don't want to duplicate all that code 100 times! If that sounds like fun to you, well, perhaps you should see a therapist. By programming intelligently using *for-loops* and *arrays* we will be able to generate lots of objects with a small amount of code. To illustrate how we start by generating the objects and explaining for-loops. Then we explain how we can store hundreds of objects inside arrays, and finally how to use these arrays to make our objects move.

6.1: Creating lots of gargoyles using For-loops

Lets start by using a for-loop to create a row of gargoyle MovieClips. Run the code in:

www.cs.du.edu/~leut/1671/flashFiles/c6_for1a.fla

In the library you will see our usual MovieClip example named "gargoyle". The code uses something called a *for-loop* to create 20 MovieClip objects and place them in a row on the screen.

```
var garg1:MovieClip ;

for (var i:Number = 0 ; i < 20 ; i++) {
        garg1 = attachMovie("gargoyle","garg"+i, i) ;
        garg1._x = i*20  + 40 ;
        garg1._y = 300 ;
        garg1._xscale = 20 ;
        garg1._yscale = 20 ;
        garg1._xscale *= -1 ;   // reverse direction of gargoyle image

}
```

When run you output as follows:

You already know most of this code. If I wrote the following code:

```
var garg1:MovieClip ;

garg1 = attachMovie("gargoyle","garg2, 2) ;
garg1._x = 2 * 20  + 40 ;
garg1._y = 300 ;
garg1._xscale = 20 ;
garg1._yscale = 20 ;
garg1._xscale *= -1 ;   // reverse direction of gargoyle image
```

then ONE gargoyle MovieClip object is created and played at location (80,300). What the for-loop does is execute the code inside of the curly braces { } 20 times, each time creating a gargoyle MovieClip object and placing it on the screen in slightly different locations. The for-loop is said to execute 20 *iterations*. The 20 gargoyles are place in locations (40,300), (60,300), (80,300) … (460,300). Hence, the gargoyles are offset only 20 pixels from each other and since that distance is less than the width of the gargoyles they overlap neighboring MovieClips images.

We will come back to the placement of the images in a minute, but first we need to explain for-loops in more detail. In general a for-loop has 4 parts:

```
for ( INITIALIZE ;  CONDITION  ;  CHANGE ) {
        STATEMENT-BLOCK
}
```

- INITIALIZE: This part sets up the initial conditions for the loop.

- CONDITION: This is a comparison or Boolean function that tests something. If the condition tested evaluates to "true", the loop is entered.

- CHANGE: This is where a change is made that eventually should cause the condition to become "false" and hence the loop to end. This change is performed after each loop iteration.

- STATEMENT-BLOCK: This is the code that is run on every iteration. This is the stuff we want to do multiple times. In the loop above it is the creation and placement of a gargoyle image.

Lets consider a simpler example:

www.cs.du.edu/~leut/1671/flashFiles/c6_for1b.fla

```
var i:Number ;

for (i = 0 ; i < 5 ; i++)
        trace("i = " + i) ;
```

When you run this example it prints out:

```
i = 0
i = 1
i = 2
i = 3
i = 4
```

Notice that inside the loop's statement-block the value of variable i changes on each iteration. In this example we have:

INITIALIZE:	i = 0
CONDITION:	i < 5
CHANGE:	i++
STATEMENT-BLOCK:	trace("i = " + i) ;

What happens when the code gets to the for-loop is it initializes variable i with the value of 0. Then it does the following:

- check if (i < 5)
- run the trace statement: prints out "i = 0"
- do the change: i++ , hence i is changed to 1
-
- check if (i < 5)
- run the trace statement: prints out "i = 1"
- do the change: i++ , hence i is changed to 2
-
- check if (i < 5)
- run the trace statement: prints out "i = 2"
- do the change: i++ , hence i is changed to 3
-
- check if (i < 5)

- run the trace statement: prints out “i = 3”
- do the change: i++ , hence i is changed to 4
-
- check if (i < 5)
- run the trace statement: prints out “i = 4”
- do the change: i++ , hence i is changed to 5

Now the check is done again, but this time, since variable i has a value >= 5, the loop is exited and the statement of the loop would be executed.

Note that the change condition is “ i++ “. Because incrementing a variable is such a common operation in loops there is a special increment operator: ++. This statement is means exactly the same as: i += 1 ; and this is exactly the same as: i = i + 1 ;

In the above example go ahead and change 5 to 20, you will see that the loop is then executed 20 times. We can modify the loop in other ways. Consider the following code:

www.cs.du.edu/~leut/1671/flashFiles/c6_for1c.fla

```
var i:Number ;

//  make the for-loop counter increment by two
for (i = 0 ; i < 20 ; i += 2) {
        trace("i = " + i) ;
}
trace("finished loop1") ;

// have the loop counter start at 20 and be decremented by 1 until it is 10
for (i = 20 ; i >= 10 ; i--)

trace("finished loop2") ;

// loop from -10 to 5
for (var j:Number = -10 ; j <= 5 ; j++)
        trace("j = " + j) ;
trace("finished loop3") ;
```

There are three loops here. Go ahead run the code and look at the output. In the first loop the change condition is to increment variable i by 2, hence the values of i for which the statement block is executed are {0,2,4,6,8,10,12,14,16,18}. When i becomes 20 the loop is exited. In the second loop i is initialized to 20 and then the change condition is to decrement 1 from it. Hence, the values of i for which the statement block is executed are {20,19,18,17,16,15,14,13,12,11,10}. Note 10 is include because the condition is >= , not just >. In the third loop the initialization sets j to -10, the check condition is j <= 5, and

the change is to increment variable j by 1, hence, the values of variable j for which the statement block is executed are {-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5}.

Reconsider the code in:

www.cs.du.edu/~leut/1671/flashFiles/c6_for1a.fla

```
var garg1:MovieClip ;

for (var i:Number = 0 ; i < 20 ; i++) {
        garg1 = attachMovie("gargoyle","garg"+i, i) ;
        garg1._x = i*20  + 40 ;
        garg1._y = 300 ;
        garg1._xscale = 20 ;
        garg1._yscale = 20 ;
        garg1._xscale *= -1 ;   // reverse direction of gargoyle image

}
```

The variable i changes on each iteration, it takes on values from 0 to 19. The statement-block is composed of 6 statements. The first statement creates a new gargoyle MovieClip instance.[1] The last three statements set the scale of the images. The second and third statements determine the location of the gargoyles. All of them are place at y-coordinates of 300. The x-coordinate is calculated as (i*20 + 40). Since the value of variable i changes on each loop iteration, the gargoyles are placed at x-coordinates 40, 60, 80, … 440, and 460.

If you change the third second statement in the block to:

```
        garg1._x = i*10  + 40 ;
```

then the images are only separated by 10 pixels and hence are more tightly superimposed resulting in:

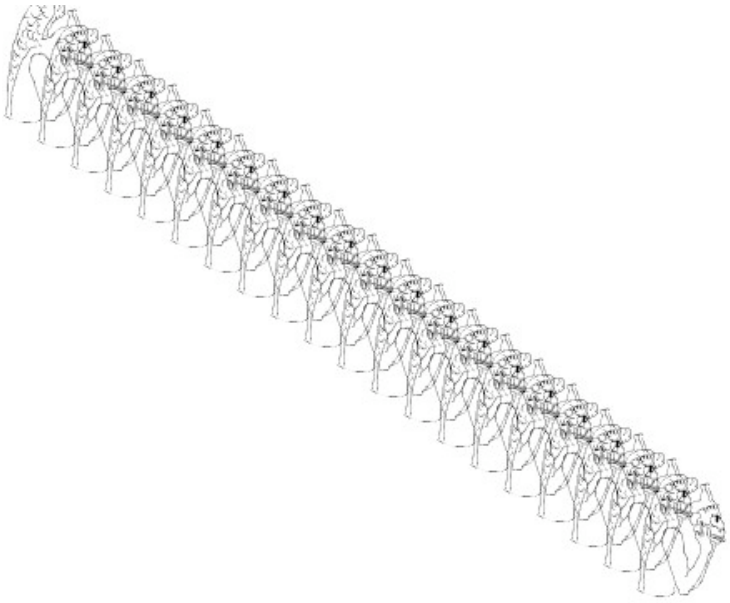
[1] Note the sort of odd parameters inside the call to the attachMovie method. At this point in time you don't really need to understand them, but here is a bit more of an explanation for those who are curious. The first parameter is the name of the library object. The second needs to be a unique name to be used inside Flash/AS for representing the object. To get unique names we simply include the loop iteration number in the name. The syntax "garg"+i does string concatenation since the left operand is a string, thus, the internal names will be "garg0", "garg1", up to "garg19". The third parameter can be thought of as the "depth" or "layer number" of the MovieClip. Only one object can occupy each "depth", hence, these need to be unique and again we just give iteration counter.

To explore more ways to control image locations look at file:

www.cs.du.edu/~leut/1671/flashFiles/c6_for1d.fla

```
var garg1:MovieClip ;

for (var i:Number = 0 ; i < 20 ; i++) {
        garg1 = attachMovie("gargoyle","garg"+i, i) ;
        garg1._x = i * 20  + 40 ;
        garg1._y = i * 15  + 40 ;
        garg1._xscale = 20 ;
        garg1._yscale = 20 ;
        garg1._xscale *= -1 ;   // reverse direction of gargoyle image

}
```

Here we change the y-coordinate of each created MovieClip as well as the x-coordinate. The result is:

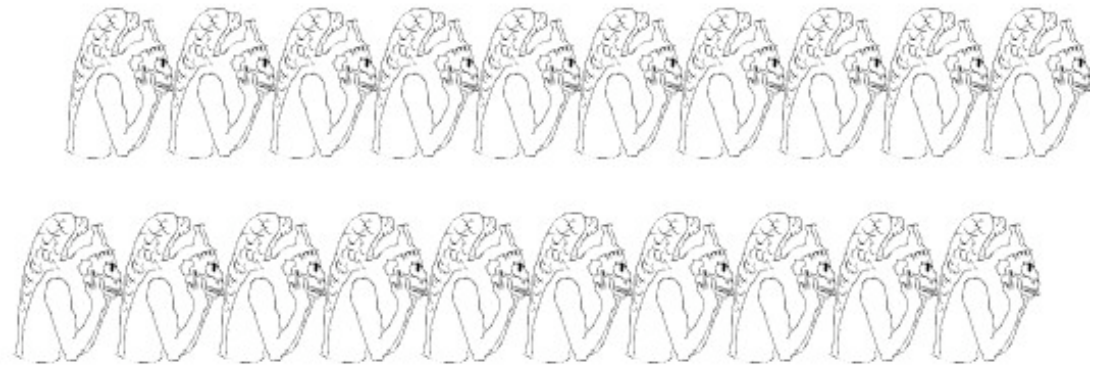


In the next file we created two rows of gargoyles:

www.cs.du.edu/~leut/1671/flashFiles/c6_for1e.fla

```
var garg1:MovieClip ;

for (var i:Number = 0 ; i < 20 ; i++) {
        garg1 = attachMovie("gargoyle","garg"+i, i) ;
        garg1._x = i * 25  + 40 ;
        if ( (i % 2) == 1)    // if i is odd, not the % sign as an operator means "modulo"
           garg1._y = 100 ;
        else
                garg1._y = 200 ;
        garg1._xscale = 20 ;
        garg1._yscale = 20 ;
        garg1._xscale *= -1 ;   // reverse direction of gargoyle image

}
```

Here is a new operator, the modulo (%) operator. This does not calculate a percentage, but rather the remainder. A % B returns the remainder of A divided by B. So, if A is even, then A % 2 equals 0, and if A is odd, A % 2 equals 1. Thus, in this example, every other iteration a gargoyle is placed at y-coordinate 200, and on the intervening iterations the gargoyles is placed at y-coordinate 100. The result is output that looks like:



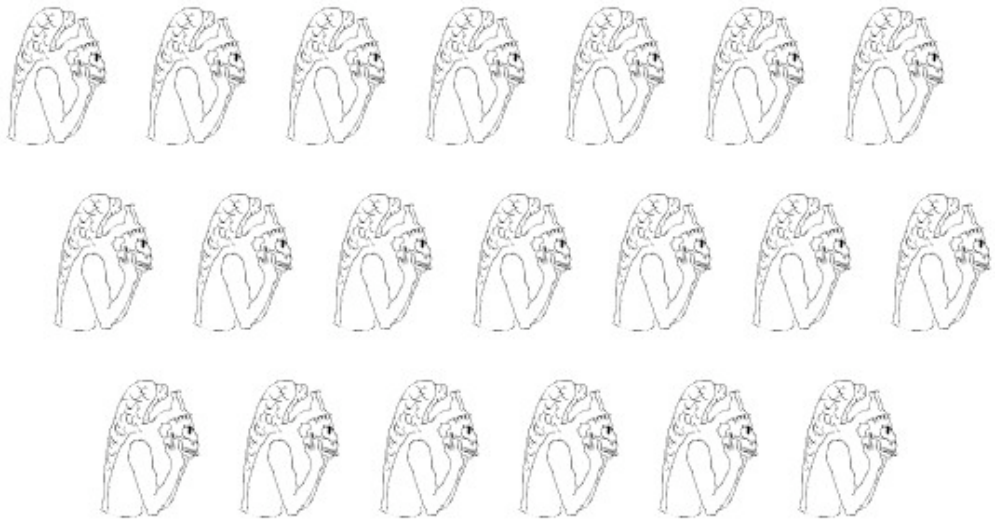
As our last example, consider:

www.cs.du.edu/~leut/1671/flashFiles/c6_for1f.fla

```
var garg1:MovieClip ;

for (var i:Number = 0 ; i < 20 ; i++) {
        garg1 = attachMovie("gargoyle","garg"+i, i) ;
        garg1._x = i * 25  + 40 ;
        if ( (i % 3) == 0)    garg1._y = 100 ;
```

```
            if ( (i % 3) == 1)    garg1._y = 200 ;
            if ( (i % 3) == 2)    garg1._y = 300 ;

            garg1._xscale = 20 ;
            garg1._yscale = 20 ;
            garg1._xscale *= -1 ;   // reverse direction of gargoyle image

        }
```

In this case our output is three rows of staggered gargoyles:



Exercises:

1. Write code using two for-loops to create an image that looks something like the following image. Note, you need to use the _rotation MovieClip property to do this:

6.2: Attempts to Get Them All Moving

Sure, drawing lots of gargoyles is fun. But getting these creatures to move about would be even better. Lets start with a first attempt:

www.cs.du.edu/1671/flashFiles/c6_for2.fla

```
var garg1:MovieClip ;
garg1 = attachMovie("gargoyle","garg1",1) ;
garg1._x = 100 ;
garg1._y = 100 ;
garg1._xscale = 20 ;
garg1._yscale = 20 ;
garg1._xscale *= -1 ;   // reverse direction of gargoyle image

var garg2:MovieClip ;
garg2 = attachMovie("gargoyle","garg2",2) ;
garg2._x = 100 ;
garg2._y = 300 ;
garg2._xscale = 20 ;
garg2._yscale = 20 ;
garg2._xscale *= -1 ;   // reverse direction of gargoyle image



// make the _x coordinate move over 10 pixels 20 times
//  NOTE: YOU DO NOT see the gargoyle move, only the end position, why?
// comment the two code lines below out to see the original gargoyle position
for (var i:Number = 0 ; i < 20 ; i++)
        garg2._x += 10 ;
```

This code places two gargoyle MovieClips on the stage, one at (100,100) and the other at (100,300). Then, it uses a for-loop to move the second gargoyle across the screen 10 pixels to the right 20 times. You might be tempted to think that you would see the gargoyle move, but you don't. Instead you just see that the second gargoyle is 200 pixels to the right of the first. The entire for-loop is processed during the playing of the frame, what you see on the screen is actually the result of running any AS code in the frame. Once the code is finished running the resultant images are rendered. In order to see the gargoyle moving you need to use the onEnterFrame() function/event as before. Here is an attempt to put 20 gargoyles on the stage and move them back and forth left/right:

www.cs.du.edu/~leut/1671/flashFiles/c6_for3.fla

```
var theGarg:MovieClip ;

// first place 20 gargoyles on the stage in a diagonal line
```

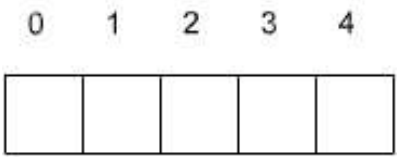
```
for (var i:Number = 0 ; i < 20 ; i++) {
        theGarg = attachMovie("gargoyle","garg"+i, i) ;
        theGarg._x = i * 20  + 40 ;
        theGarg._y = i * 15  + 40 ;
        theGarg._xscale = 20 ;
        theGarg._yscale = 20 ;
        theGarg._xscale *= -1 ;   // reverse direction of gargoyle image

}

var vx:Number = 5 ;

onEnterFrame = function() {
        theGarg._x += vx ;

        if ( (theGarg._x >= Stage.width)  || (theGarg._x <= 0) ) {
           vx *= -1 ;
                theGarg._xscale *= -1 ;
        }

}
```

When you run the code only one gargoyle moves back and forth, the other 19 sit there. The reason is that in the onEnterFrame function you are changing the _x member of the gargoyle MovieClip that is currently held inside of variable "theGarg". Variable theGarg holds only one MovieClip. In the for-loop you create 20 MovieClip objects using attachMovie and put them in variable theGarg. On entering the first iteration variable theGarg is empty and a gargoyle MovieClip object is put in the variable. On the second iteration a new object is created and placed in variable theGarg. The current object is removed from the variable to make way for the new object, but the first object still exists. You can see it on the screen at location (40,40). At the end of the loop there are 20 objects on the screen, but only the last one is currently stored inside variable theGarg. Thus, in the onEnterFrame function, the gargoyle MovieClip object held in variable theGarg is moved back and forth across the screen, but the other 19 are not touched. So what happened to the other 19? They are still on the stage, but we did not keep them in a variable so we cannot access their properties.[2] We need a way to store each of the MovieClip objects in a place where we can access them. We could create 20 variables, but then we are back to having to duplicate our code 20 times which is what we wanted to avoid when we started this discussion. Instead we will use **arrays** to store the objects.

6.3: Arrays and Getting The Gargoyles All Moving

[2] Actually, this statement is untrue. In the second parameter of attachMovie() we specified a unique internal name for the MovieClips we created: garg0, garg1, garg2, …. up to garg19, and we can access them using these names. But, this is not good general programming style so we won't use them this way and instead show how to store them in arrays. Note, hard-core ActionScript programmers may in fact access the MovieClips using these internal names, but, we discourage you from this practice.

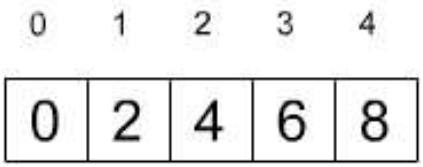
An array is a special type of variable that holds many different objects. Think of a wooden soda-bottle crate/case. It holds 24 bottles of soda but is one storage container. An array is similar in that it is one container that holds many objects, but the compartments that hold the objects can be thought of being in a single line and individual locations can be indexed as the offset from the beginning of the sequence. For example, if I have an array of five elements you could think of it as the following figure where the numbers above the elements are the element indexes (or offsets from the beginning) of the array:



Notice that array elements are number 0 to N-1, where N is the number of array elements. In the picture above the "elements" or "compartments" of the array are empty. To see how to actually create an array and put numbers into the elements, lets say I run the following code:

```
var theArray:Array ;
theArray = new Array(5) ;


theArray[0] = 0 ;
theArray[1] = 2 ;
theArray[2] = 4 ;
theArray[3] = 6 ;
theArray[4] = 8 ;
```

Then you would have an array of five elements that contain the numbers 0, 2, 4, 6, and 8. The var declaration specifies that variable "theArray" is of type Array. We now use the "new" operator to actually create an instance of an array. The second statement creates an array object and assigns it to the "theArray" variable. The next five statements put the data items (0,2,4,6,8) into elements (0,1,2,3,4) respectively. The resultant array looks like:



where the 0,2,4,6 and 8 are the array element contents.

Code that uses arrays often involves for-loops because the loops allow a natural way to iterate over the array's elements. To see an example of creating arrays, putting elements in the elements, and removing elements from the elements, check out:

www.cs.du.edu/~leut/1671/flashFiles/c6_array1.fla

```
var maxElements:Number = 10 ;
var theArray:Array ;
theArray = new Array(maxElements) ;


// (1)
// Explicitly assign elements to the array one call at a time
theArray[0] = 0 ;
theArray[1] = 2 ;
theArray[2] = 4 ;
theArray[3] = 6 ;
theArray[4] = 8 ;
theArray[5] = 10;
theArray[6] = 12;
theArray[7] = 14 ;
theArray[8] = 16 ;
theArray[9] = 18 ;

// (2)
trace("theArray[8] = " + theArray[8]) ;

// (3)
trace("\ncontents of array:") ;
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}

// (4)
for (var i:Number = 0 ; i < maxElements ; i++) {
        theArray[i] = 0 ;
}

// (5)
trace("\ncontents of array:") ;
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}

// (6)
for (var i:Number = 0 ; i < maxElements ; i++) {
        theArray[i] = i*4 ;
}

// (7)
trace("\ncontents of array:") ;
```

```
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}
```

The first chunk of code, following the commented out (1) above, puts numbers into the array one element at a time. The code following the (2) above shows how to access a single element of the array. The code following the (3) is a for-loop that prints out the contents of each array element. Notice how the counter variable in the for-loop is used as the array element index. The loop following the (4) sets the contents of each array element to zero. The loop following the (5) prints out the contents of each array element, all of the elements now contain a zero. The loop after the (6) fills each element with a value that is equal to 4 times the current for-loop counter variable. Finally, the code after the (7) again prints out the array contents.

In the following example we show what happens

```
var maxElements:Number = 8 ;
var theArray:Array ;
theArray = new Array(maxElements) ;


theArray[0] = 2 ;
theArray[1] = 3 ;
theArray[2] = 5 ;
theArray[3] = 6 ;
theArray[4] = 12 ;
theArray[5] = 18;



// see what happens if you access data that has not been initialized
trace("\ncontents of array:") ;
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}

// print out the size of the array:
trace("Num elements in the array is " + theArray.length) ;

theArray[6] = 28 ;
theArray[7] = 30 ;

// see what happens if you access data that has not been initialized
trace("\ncontents of array:") ;
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}

theArray[9] = 44 ;
// print out the size of the array:
trace("Num elements in the array is " + theArray.length) ;
// see what happens if you access data that has not been initialized
```

```
trace("\ncontents of array:") ;
for (var i:Number = 0 ; i < theArray.length ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}
```

You may wonder what happens if you access an array element if you have not put anything into that elements. Look at:

www.cs.du.edu/~leut/1671/flashFiles/array2.fla

```
var maxElements:Number = 8 ;
var theArray:Array ;
theArray = new Array(maxElements) ;


theArray[0] = 2 ;
theArray[1] = 3 ;
theArray[2] = 5 ;
theArray[3] = 6 ;
theArray[4] = 12 ;
theArray[5] = 18;


// (1)
// print out the size of the array:
trace("Num elements in the array is " + theArray.length) ;

// see what happens if you access data that has not been initialized
trace("contents of array:") ;
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}


// (2)
theArray[6] = 28 ;
theArray[7] = 30 ;

trace("") ;

// (3)
// print out the size of the array:
trace("Num elements in the array is " + theArray.length) ;

trace("contents of array:") ;
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(i + ":\t" + theArray[i]) ;
}
```

In this example we create an array of 8 elements and then assign data to the first six elements. Then, after (1) , we print out the size of the array and then loop through all 8 elements and print out the content. The array class includes a member “length” which

holds the number of elements in the array, this is used in the trace statement after (1). When the elements that have no data in them are printed they show up as "undefined". The code after (2) then puts data into elements 6 and 7, and finally the code after (3) prints out all the data in the array. The output of the above code is:

```
Num elements in the array is 8
contents of array:
0:      2
1:      3
2:      5
3:      6
4:      12
5:      18
6:      undefined
7:      undefined

Num elements in the array is 8
contents of array:
0:      2
1:      3
2:      5
3:      6
4:      12
5:      18
6:      28
7:      30
```

In all of the above examples we have stored numbers in the arrays, but arrays can hold all sorts of data including Strings and MovieClips. First lets consider an example of storing Strings in an array:

www.cs.du.edu/~leut/1671/flashFiles/c6_array3.fla

```
var maxElements:Number = 4 ;
var theArray:Array ;
theArray = new Array(maxElements) ;


theArray[0] = "Roses are red" ;
theArray[1] = "Violets are blue" ;
theArray[2] = "Nothing on earth" ;
theArray[3] = "Smells as sweet as you" ;

// print out the size of the array:
trace("Num elements in the array is " + theArray.length) ;

// print out contents of the array
trace("contents of array:") ;
for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(theArray[i]) ;
}
```

```
theArray.sort() ;
trace("") ;
trace("After theArray.sort(), contents of array:") ;

for (var i:Number = 0 ; i < maxElements ; i++) {
        trace(theArray[i]) ;
}
```

The output from running this code is:

```
Num elements in the array is 4
contents of array:
Roses are red
Violets are blue
Nothing on earth
Smells as sweet as you

After theArray.sort(), contents of array:
Nothing on earth
Roses are red
Smells as sweet as you
Violets are blue
```

Notice the use of the sort() method. Sort is a built in method in the Array class. It sorts the data based on lexigraphic (alphabetic) order. If you have numbers in the array you need to add another function to sort numerically, we will discuss this later.

Okay, we finally know everything we need to make the gargoyle objects move. All we need to do is store the MovieClip objects in an array and then loop through the array updating the MovieClip _x and _y members. Run the following example:

www.cs.du.edu/~leut/1671/flashFiles/c6_array4a.fla

```
// (1)
var maxElements:Number = 10 ;
var theArray:Array ;
theArray = new Array(maxElements) ;

var theGarg:MovieClip ;

// (2)
// create 10 gargoyle MovieClips and position them on the stage
for (var i:Number = 0 ; i < maxElements ; i++) {
        theGarg = attachMovie("gargoyle","garg"+i, i) ;
        theGarg._xscale = 20 ;
        theGarg._yscale = 20 ;
        theGarg._xscale *= -1 ;   // reverse direction of gargoyle image
        theGarg._x = Math.random() * ( Stage.width - theGarg._width) ;
        theGarg._y = Math.random() * ( Stage.height - theGarg._height) ;
```

```
            // (3)
            // store this MovieClip object in the array
            theArray[i] = theGarg ;

    }

    var g1xv:Number = 5 ;
    var g1yv:Number = 5 ;


    // move the gargoyle MovieClip currently stored in variable theGarg
    onEnterFrame = function () {

            // update locations of all gargoyles

            // (4)
            for (var i:Number = 0 ; i < maxElements ; i++) {
                    theArray[i]._x += g1xv ;
                    theArray[i]._y += g1yv  ;
            }



    }
```

When you run the code, 10 gargoyle are placed randomly on the screen and then they all move down and to the right at the same speed. A few notes about the code:

- The code after (1) creates the array
- The loop after (2) creates 10 gargoyle objects and sets their _x and _y locations to random spots on the Stage.
- Inside this loop, the statement after (3) stores each gargoyle object in the array so we can access them later
- The loop after (4), loops through each element of the array, which are the gargoyle objects, and updates their _x and _y locations by adding g1xv to them. Notice that this code is inside the onEnterFrame function hence the gargoyle objects are updated at rate FPS.

This is great! We can create lots of gargoyles now. But what if we want them to reflect off of the stage borders as before with one or two gargoyles?

www.cs.du.edu/~leut/1671/flashFiles/c6_array4b.fla

The first part of the code is the same the previous example: we create 10 gargoyle objects, set their (_x,_y) locations at random, and then put the object in an array. The difference is the onEnterFrame function. Here we not only loop through the array of gargoyle objects

updating _x and _y for each, but we also check for Stage boundary violations. If an object is located on the x (y) boundary or has gone off, we negate velocity g1xv (g1yv) to move the object back. Below is the onEnterFrame function code:

```
onEnterFrame = function () {

        // update locations of all gargoyles
        for (var i:Number = 0 ; i < maxElements ; i++) {
                theArray[i]._x += g1xv ;
                theArray[i]._y += g1yv  ;


                // check gargoyle boundaries and reverse direction/scale if needed

                // see if going off right boundary, if so negate velocity
                if ( (theArray[i]._x + (theArray[i]._width/2) ) >= Stage.width) {
                        g1xv *= -1 ;    // reverse direction of gargoyle
                        theArray[i]._xscale *= -1 ;    // reverse direction of the image
                }

                // see if going off left boundary, if so negate velocity
                if ( (theArray[i]._x - (theArray[i]._width/2)) <= 0 ) {
                        g1xv *= -1 ;    // reverse direction of gargoyle
                        theArray[i]._xscale *= -1 ;    // reverse direction of the image
                }

                // see if going off bottom boundary, if so negate y-velocity
                 ( (theArray[i]._y + (theArray[i]._height/2))  >= Stage.height) {
                        g1yv *= -1 ;    // reverse direction of gargoyle
                }

                // see if going off top boundary, if so negate y-velocity
                if ( (theArray[i]._y - (theArray[i]._height/2) ) <= 0) {
                        g1yv *= -1 ;    // reverse direction of gargoyle
                }

        }

}
```

This is a another step towards our goal, but we are not there yet. Notice that when variable g1xv or g1yv is negated, ALL objects switch directions. So only a few objects actually make it to the boundaries. The ones that make it are the top-most, bottom-most, left-most, and right-most gargoyles. We see that all gargoyles move in sync. Perhaps this is the right behavior for sheep, but a gargoyle? Somehow I don't see gargoyles as docile herd creatures. I see gargoyles as independent creatures, going wherever they wish when they wish.

We need each object to bounce around independently at their own velocities. But does that mean we need to create separate velocity variables for each gargoyle object? No! We just create an array of x-velocities and an array of y-velocities. The velocities found in element "j" or the velocity arrays belong to the gargoyle found in element "j" of the gargoyle array. Check out the following example. We include all the code here:

www.cs.du.edu/~leut/1671/flashFiles/c6_array4c.fla

```
var maxElements:Number = 20 ;
var theArray:Array ;
theArray = new Array(maxElements) ;

// (1)
var xvArray:Array ;
xvArray = new Array(maxElements) ;

// (2)
var yvArray:Array ;
yvArray = new Array(maxElements) ;

var theGarg:MovieClip ;

// create 10 gargoyle MovieClips and position them on the stage
for (var i:Number = 0 ; i < maxElements ; i++) {
        theGarg = attachMovie("gargoyle","garg"+i, i) ;
        theGarg._xscale = 15 ;
        theGarg._yscale = 15 ;
        theGarg._xscale *= -1 ;   // reverse direction of gargoyle image
        theGarg._x = theGarg._width + Math.random() * ( Stage.width - 2*theGarg._width) ;
        theGarg._y = theGarg._height + Math.random() * ( Stage.height - 2*theGarg._height) ;
        theArray[i] = theGarg ;

        // (3)
        xvArray[i] = Math.random() * 3 ;  // set intial between 0 and 3
        yvArray[i] = (Math.random() * 6) - 3 ;  // set initial velocity between -3 and 3


}

var g1xv:Number = 5 ;
var g1yv:Number = 5 ;


// move the gargoyle MovieClip currently stored in variable theGarg
onEnterFrame = function () {

        // update locations of all gargoyles
        for (var i:Number = 0 ; i < maxElements ; i++) {

                // (4)
                theArray[i]._x += xvArray[i] ;
                theArray[i]._y += yvArray[i]   ;



                // check gargoyle boundaries and reverse direction/scale if needed

                // see if going off right boundary, if so negate velocity
                if ( (theArray[i]._x + (theArray[i]._width/2) ) >= Stage.width) {
```

```
                        // (5)
                        xvArray[i] *= -1 ;    // reverse direction of gargoyle
                        theArray[i]._xscale *= -1 ;    // reverse direction of the image
                }

                // see if going off left boundary, if so negate velocity
                if ( (theArray[i]._x - (theArray[i]._width/2)) <= 0 ) {
                        xvArray[i] *= -1 ;    // reverse direction of gargoyle
                        theArray[i]._xscale *= -1 ;    // reverse direction of the image
                }

                // see if going off bottom boundary, if so negate y-velocity
                if ( (theArray[i]._y + (theArray[i]._height/2))  >= Stage.height) {
                        yvArray[i] *= -1 ;    // reverse direction of gargoyle
                }

                // see if going off top boundary, if so negate y-velocity
                if ( (theArray[i]._y - (theArray[i]._height/2) ) <= 0) {
                        yvArray[i] *= -1 ;    // reverse direction of gargoyle
                }
        }  // end of for-loop

}  // end of onEnterFrame()
```

In the code above, after the (1) you see the code to create an array to hold the x-velocities, and after (2) is the code creating the array to hold the y-velocities. After (3) we initialize the elements of the xvArray and yvArray with random numbers. The x-velocities are initialized with values of 0 to 3, so they are all moving in the direction they are facing, and the y-velocities with values between -3 and 3. Inside the onEnterFrame function the code loops through the gargoyle array and updates the x,y locations of each gargoyle, but it uses the velocities for that particular gargoyle found in the corresponding xvArray and yvArray. You can see this in the code after (4). Finally, when an individual gargoyle hits a boundary, only that one gargoyle's x or y-velocity is negated. You can see this in the line after (5) and in the subsequent boundary checks.

Exercises:

2. Time to add some love to a gargoyles life. Write code to do roughly the same as found in: www.cs.du.edu/1671/flashFiles/heartRain.html Hint: create about 100 instances of the hear and have moving down. When a heart object goes off the bottom reset it to just above the top screen so it rains down again. I added a drift towards the right, so when they go off the right I move them to just left of the stage. Also note there is some randomness in them so the hearts are not all moving in sync.

3. Create a two phase animation. In phase one the user inputs the number of gargoyles then want and the average x and y velocities. Then, when the click a button, the animation plays according to the user's inputs.

4. Write code to generate 15 gargoyles moving randomly around the screen. When you mouse-click on a gargoyle it turns into a heart.

5. Extend the previous example so that click-on objects keep switching back and forth: gargoyles turn into hearts and hearts turn into gargoyles.