Chapter 2: ActionScript Programming Introduction

Knowing how to draw pretty pictures and make tweens is lots of fun, but it is not programming. Programming allows one to have control over so much more. In the flash environment programming increases flexibility, allows for interactivity needed to make games, and makes many things possible that cannot be done using only the drawing package and timeline. In the broader view, just like mathematics is the "language of science", programming is the "language of computer games". Applications such as text editors, drawing packages, or 3D modeling tools are created by programmers. Under that nice graphical user interface (GUI), and not visible to the user, are piles of code, just like under the hood of your car is a sophisticated engine. Learning how to code is the first step to learning how to make games.

2.1: Simple Output

Most programs you write need some way to get information in from the user and give results back to the user. This is called **input** and **output.** There are several ways to do input/output in ActionScript, but for now we will focus only on the **trace** command to produce output. Our first task is to write a program that says hello to the user. Nowadays computer programming is most often done using an Integrated Development Environment (IDE). An IDE is an application that makes it easier to write, compile, and debug code. Flash has a built in IDE, you are already using it! Do the following:

1. Open a new document
2. Left click on the first frame
3. Click on the arrow button in the upper right part of the Properties Panel (it is just below the question mark). This opens a window for typing in ActionScript code.
4. In this new window add the following line of code:

   trace("Hello World") ;
5. Hit ctrl+enter to run the program

When you run the program an output window is created and the words "Hello World" appear in that window. If for some reason your Properties Panel is not visible, just go to the Windows sub-menu and select properties. Note that the first frame in the timeline now has an "a" inside. This is to signify that the frame has actionscript code embedded within it.

As we saw in chapter 1, one can also publish flash movies and then run them inside of a browser. However, the trace command DOES NOT work when a movie is published, it is just ignored. The trace command is simply there to provide an easy way to do output

while developing code. The trace command only works with ctrl+enter. In the next chapter we will see how to do input/output in published movies.

2.2: Moving Ball Example

Okay, so you may not find the "hello world" program as amusing as I do. Time to move on to something a tad more visually interesting. Not a whole lot more interesting mind you, but we have to start somewhere. Assuming you have never programmed before, you may not understand everything in this chapter. That is okay, you are not supposed to. You are supposed to get a reasonable idea of what is going one, get a bunch of questions in your head, and be ready for the following sections where we will presumably answer most of your questions.

First, lets just get the mechanics of putting an object on the stage with ActionScript and then moving that object. 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
4. Convert the circle to a symbol (Modify -> Convert to symbol). In the menu that pops up change the name to "ball" and then next in the "linkage" section check the "Export for ActionScript" checkbox. When you check the "export for ActionScript" checkbox it highlights another option, namely the "Identifier". It should already fill this box in with "ball", but if not type in "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 theBall:MovieClip  = attachMovie("ball", "b1", 1);
theBall._x = 10 ;
theBall._y = 200 ;

onEnterFrame = function () {
theBall._x = theBall._x + 4 ;
}
```

Now test the movie (ctrl-enter).

CODE EXAMPLE (from web site): 9/14, c2_movingBall

You should see your ball slowly moving to the right until it disappears. You have to type everything in exactly. If you do not see the ball moving, just go ahead and download file: c2_movingBall.fla and open it in flash, then hit ctrl+enter.

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 on the name "ball" the ball appears in a new window. To close that window click on the tab named "Scene 1" just below the timeline. 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.

Steps 1 - 6 set up the ball-moving clip in the library, but without the code you put in frame 1 duing step 7 nothing will be seen on the screen. Lets look at the code line by line and explain what it does:

```
var theBall:MovieClip  = attachMovie("ball", "b1", 1);
```

This creates an "instance" of the ball movie clip inside our ActionScript program. The word "var" is short for variable, a place in a program where we hold information. In this case the information stored in the variable "theBall" is an ActionScript MovieClip, hence, the type of the variable is "MovieClip".

```
theBall._x = 10 ;
theBall._y = 200 ;
```

These two lines set the initial location of the ball held in the variable "theBall" to coordinates (10,200).

```
onEnterFrame = function () {
   theBall._x = theBall._x + 4 ;
}
```

These three lines define a function that is executed again and again. The x-coordinate of the ball is incremented by 4 pixels each time the code is run (which is at a rate of "Frames Per Second" defined in the Properties Panel).

To help understand this example we need to consider a few concepts: variables, arithmetic statements, the coordinate system, and frame (event) driven programming. These are covered in the following sections. Come back and read this section again after going through the rest of the sections in this chapter.

2.3 Variables

Computer programs need to store information in the computers memory. A programming language, like ActionScript, has a way of associating "tags" or "names" with memory locations. These "tags" or "names" are called **variables.** A variable is simply a named location. Furthermore, variables have types associated with them that specify what type of information can be stored in the location. Consider the following analogy: you are going to tidy up your apartment/home/dorm room and to do so you decide to organize all your possessions in boxes. You can only put one item in each box,

and, you have to specify the type of items that can go into a box. Being a junk-food eating music-loving bibliophile, all of your possessions fall into 3 categories: books, snacks, and CDs. Lets say you get a box, label it "Spanish : book", and put it on your desk. You get another box, label it "computer science : book", and put it on your desk. You get 3 more and label them: "morning : snack", "afternoon : snack", and "Beatles : CD". So far there is nothing in the boxes, you simply have set out 5 boxes, labeled them, and specified what type of item can go into each box.

In Acitionscript we create a variable and specify its type with the a **variable declaration statement**:

```
var aNum:Number ;
var num2:Number ;
var aString:String ;
var st2:String ;
var theBall:MovieClip ;
```

The above five statements declare 5 variables: aNum, aString, num2, st2, and theBall. The statements give a name to each of the variables, namely "aNum", "num2", "aString", "st2", and "theBall". The statements also specify what type of data the variables can hold, namely (Number, Number, String, String, MovieClip) respectively. After the above 5 lines of code, the memory locations associated with the variables have no contents, the statements simply create a named space to hold stuff. This just like our analogy of putting 5 boxes on your desk, labeling them, and in the label specify the type of contents that can go into the box.

Lets go back to our home organization example. Say you pick up your copy of "Dos Mundos" and put it in the box labeled "spanish : book", you put your copy of "An Introduction to Computer Programming Using Games and ActionScript" into the box labeled "computer science : book", a twinkie into the box labeled "morning : snack", another twinkie into the box labeled "afternoon : snack", and your copy of The White Album into the box labeled "beatles : CD". You have now actually put items into the boxes. This is similar to the following 5 lines of code:

```
aNum = 999 ;
num2 = 3.14159 ;
aString = "Hello world" ;
st2 = "Cats rule, dogs drool" ;
theBall = attachMovie("ball", "b1", 1 );
```

These five statements put actual data into the memory locations specified by the variable declarations, just like the physical act of putting the books, snacks, and CDs into the box.

There was no reason we had to give the variables the names that we did. If we wanted to we could have named them "a", "b", "c", "d", and "e", or even something more obscure like "buzz", "hamm", "rex", "woody" and "MrPotato". BUT, you want the variable names to mean something to help you read your code. Imagine you labeled boxes for storage and put them on a shelf. You would not label them "buzz", "hamm", etc (unless

you happen to be a toy collector), nor would you label them “a”, “b”, etc. You want names that mean something.

TIP: Give variables names that are meaningful. Furthermore: I suggest variable names always start with a lower-case character.

You can give variables pretty much any name, as long as they start with a character (not a number), don’t have spaces, and are not reserved words. Reserved words are part of the programming language, words like “var”, “MovieCli”, “Number”, “if”, “else”, “for”, “while” and so on. If you want to know all the reserved words, just google “ActionScript reserved words”. And if you use one in a variable declaration, don’t worry, flash will let you know you goofed with an error message something like:

Error Scene=Scene 1, layer=Layer 1, frame=1:Line 3: Identifier expected

A tad hard to read? Yep! The key here is the last two words: Identifier expected and also the line number, this tells you at the specified line you need to give a legitimate variable name.

In order to access the data inside a variable we just refer to the variable name. Consider the following code:

```
trace(aNum) ;
trace(num2) ;
trace(aString) ;
```

These three trace statements will print out the contents of the three variables. Go ahead and type the following code into the action script panel and then run the code with ctrl+enter:

```
var aNum:Number ;
var num2:Number ;
var aString:String ;

aNum = 5 ;
num2 = 3.14159 ;
aString = “Dogs drool” ;

trace( aNum) ;
trace(num2) ;
trace(aString) ;
```

Now consider type specification. The variables “aNum” and “num2” as specified above are of type “Number”. This means that they can hold number data only. If you try to do the following you will get an error:

```
num2 = “Sloths are cool” ;
```

Go ahead, add this line to your program and hit ctrl+enter. You will NOT hurt your computer. It is okay to make mistakes, that is how we learn, and you will make lots of mistakes when learning to program so go ahead and do this one now.

When you try to run this code you get an error message that contains something like:

> "found String where Number is required"

The error message also includes the line number, hence, it is pretty easy to figure out what is wrong and how to fix it (either put a number in that variable, or change the variable type to string).

So why do we bother with types? It makes our programs easier to debug. Debugging is the process of removing errors from the code, and, if you are have statements using variables as if they were numbers, such as adding numbers together, it helps if the contents are actually numbers and not strings!

You can combine variable declaration and variable assignment in one statement. For example, consider the following two lines of code:

```
var animal1 : String = "pig" ;
var numAnimals : Number = 5 ;

trace("there are" ) ;
trace( numAnimals ) ;
trace( animal1 ) ;
```

Go ahead and slap that into the ActionScript window and see what happens. The first two lines combine both the variable declaration and assignment to the variables. In a little bit we will show how to make the output a bit prettier.

You can also assign the contents of an existing variable to another variable. Consider the following:

```
var n2:Number ;
n2 = 5 ;
var n3:Number ;
n3 = n2 ;
```

After the above four lines, there are two variables, n2 and n3, of type Number, and at the end of executing all 4 lines of code both variables are holding the value 5. The fourth line above assigns the contents of variable n2 to the variable n3.

To see examples of this in a file, check out:

CODE EXAMPLE (from web site): 9/14, c2_vars1

A closing note on variable types and ActionScript. You can skip this paragraph if you want. Before ActionScript 2.0 variables did not have types. To stick with our analogy it is sort of like a box that can grow or shrink to hold any type of object. This form of "untyped" and mutable variables have advantages, but, they make learning how to program and debug difficult. Thus, we won't get into the debate of typed versus untyped, for this book ALWAYS specify variable types (unless instructed otherwise....)

2.4 Arithmetic Statements

Storing and printing out data is useful, but there is so much more one does with variables. The first thing to do is to compute some simple arithmetic. Consider the following code:

```
var num1:Number ;
var num2:Number ;

num1 = 5 ;
num2 = 2 + 3.5 ;

trace(num1) ;
trace(num2) ;
```

The second assignment statement, "num2 = 2 + 3.5 ; ", is new to you. The way this works is everything to the right of the = sign is computed, and then assigned (or put into) the variable on the left side of the = sign. The = sign does not mean "equals", instead it means "assign the value on the right of the = sign to the variable on the left of the equal sign". As you see from the trace statement output, it does pretty much what you would expect. The + sign is called the **operator,** and the 2 and the 3.5 are called the **operands**. Go ahead and try out some arithmetic, you can us the following operators:

+	(plus)
-	(minus)
*	(times)
/	(divide)

You can also put variables on the right hand side of the = sign, in which case the current value stored in the variable is used in the calculation. Consider:

```
var num1:Number = 3 ;
var num2:Number = 5 ;
var num3:Number ;
```

```
num3 = num1 + num2 ;

trace(num1) ;
trace(num2) ;
trace(num3) ;
```

The above code will print out:

3
5
8

The assignment statement for num3 adds together the contents of num1 and num2. And there is nothing to stop you from using the same variable on the right of the = sign as the one you are assigning to on the left. For example:

```
num1 = 5 ;
num1 = num1 + 2 ;
trace(num1) ;
```

The output resulting from the above three lines is "7". It works as follows. The right hand side is evaluated first, hence the contents of variable num1 are added to 2, which gives 7. Then, the value of the right hand side is assigned to the variable on the left hand side, overwriting the current contents. Hence, the value of 5 previously contained in variable num1 is replaced with the new value from the right hand side, i.e. 7.

It is VERY common in programming to keep adding some fixed value to a variable, you will see this later in moving objects across the screen and in for loops. Actually, the astute reader will notice we already did this in our moving ball example. Because it is so common most programming languages, including ActionScript, provide a shorthand notation. Say you want to add 2 to the contents of variable num1. You could do this two ways:

```
num1 = num1 + 2 ;
```

or you could do this as:

```
num1 += 2 ;
```

The second way is just a shorthand notation doing the same thing as the first. All 4 basic arithmetic operators have the same shorthand notation as an option. Consider the following code:

```
var num1:Number ;
num1 = 4 ;
trace(num1) ;
num1 += 1 ;
trace(num1) ;
num1 *= 4 ;
```

```
trace(num1) ;
num1 /= 10 ;
trace(num1) ;
num1 -= 1 ;
trace(num1) ;
```

The output will be:

4
5
20
2
1

Do you see why?

EXERCISES:

1. What is the output resulting from the following code:

```
var n1 : Number = 5 ;
var n2: Number ;
var n3: Number = 7 ;
n2 = n1 * n3 ;
n2 *= 2 ;
trace(n2) ;
```

2. What is the output resulting from the following code:

```
var theNum : Number = 5 ;
theNum += 3 ;
trace(theNum) ;
theNum *= 3 ;
trace(theNum) ;
theNum = theNum – 20 ;
trace(theNum) ;
theNum /= 2 ;
trace(theNum) ;
```

3. What is wrong with the following code?

```
var num1 = 5 ;
var num2 : Number = 7
var num3 : Number = num1 + num2 ;
trace(“num3”) ;
```

4. Consider the code below. Add a two lines of code that calculates and prints out the average of the numbers stored in num1, num2, num3, and num4:

```
var num1:Number = 3 ;
var num2:Number = 5 ;
var num3:Number = 9 ;
var num4:Number = 15 ;
var average:Number ;
```

2.5 Simple String Manipulation

Variables can take on several types. The four major types we will deal with are Number, String, MovieClip, and user defined classes. The last two are covered later. Strings are a sequence (or string) of characters. We delimit strings with the " characters. Consider the following code:

```
var aString:String = "The cow says MOOO" ;
trace(aString) ;
```

When the code is run the informative statement above is printed in the output window. Note, the quote marks to not appear in the output, they are simply part of ActionScript code to identify all the characters in between the quotes as a string. Can you guess what the following line of code will produce:

```
var aString:String = "The cow says MOOO" ;
trace("aString") ;
```

It prints out the word "aString", without quotes around it. It does not print out "The cow says MOOO". The reason is that the trace command sees the quotes and treats the characters between the quotes as a string and prints them out. If the quotes are missing the trace command assumes the word aString is a variable name and prints out the contents of that variable.

There are many operators on strings, for now we will just cover the most useful one, the **concatenation** operator. The concatenation operator, +, just ties the two strings together, with the right hand operand immediately glued on the end of the left hand operand. Consider the following code:

```
var st1:String = "How now" ;
var st2:String = "brown cow" ;
var st3:String ;
st3 = st1 + st2 ;
trace(st3) ;
```

What gets printed out is "How nowbrown cow", without the quotes. What happens is the contents of the right operand variable, "brown cow", are appended, or glued, to the end of the contents of the left operand variable, "How now", and then assigned as the contents of variable st3. Then, the contents of variable st3 are printed. Notice that the words

“now” and “brown” are squished together. The programming language does not understand grammer nor any nuances of the English language. Variables st1 and st2 simply hold a string of characters and the + operator just concatenates them together. So, how do we make it pretty? Replace the second last line with:

```
st3 = st1 + " " + st2 ;
```

Now the variable on the left, st3, is assigned the contents of three strings appended together. A space character is appended to the end of the contents of st1 and then the contents of st2 are appended to the end of both. Of course you can stick any string of characters in between st1 and st2, not just a space. If you replace the second last line above with:

```
st3 = st1 + " you god-like " + st2 ;
```

It will print out , “How now you god-like brown cow”, again, without the quotes.

It is important to differentiate between strings and variable names. Consider the following code:

```
var num1:Number = 5.5 ;
var num2:Number = 2 ;
var st1:String = "Learning how to program rocks!" ;

trace(num1) ;
trace("num1") ;
trace(st1) ;
trace("st1") ;
```

Will print out:

5.5

num1

Learning how to program rocks!

st1

The key to understanding this is in the quotes. If the characters between the parentheses of the trace statement do not have quotes, then it is assumed to be a variable and the contents of that variable are printed out. If the characters between the parentheses are quoted, then it is treated as a string. What is really happening is the trace command always evaluates the stuff inside the parentheses as a string and print out the string. Stuff

without quotes are variables, and the contents are treated as a string. Consider the following code:

```
var num1:Number = 5.5 ;
var num2:Number = 2 ;

trace("num1 = " + num1) ;
trace("num2 = " + num2) ;
```

This produces the output:

num1 = 5.5

num2 = 2

The stuff between the parentheses is all converted into one string and printed out. The contents of variables num1 and num2 are converted into strings and appended to the string on the left hand side of the + operator, and then printed out as one string.

EXERCISES:

1. Write code to assign (5, 9, 13) to three different variables. Then calculate the average of these three variables and put that in a 4^{th} variable. Finally, print out the average.
2. Write code to create two variables of type Number named "numDogs" and "numCats". Assign the number of dogs and cats in your house to these variables. Then, write a statement or statements that produces an output line like: "There are 2 dogs and 3 cats in the house", where 2 and 3 where the numbers in your variables.

2.6 Frames and the onEnterFrame Function

Remember the frame-by-frame animation we created in section 2.2? Flash would play this animation by playing the first frame, then the second, the third, and so one. When the last frame was reached the animation started over at the beginning. The rate of transitions from frame to frame is controlled by the frame rate (fps) in the Property Panel. If the flash animation only has one frame then the player plays it once and stops.

The same thing happens when we have ActionScript code in the animation. If there is one frame in the Flash Movie, it plays once and stops. If there are two or more frames

the frames are played in order until the end, then started over again from the beginning. Create a two frame animation in flash. In frame 1 add one line of ActionScript code:

```
trace("Inside frame 1") ;
```

In frame 2 add one line of code:

```
trace("Inside frame 2") ;
```

Now run the code, with ctrl+enter, and see what happens. Go ahead and put in three frames and see what happens.

CODE EXAMPLE (from web site): 9/14, c2_twoFrames

Next, try moving a ball by putting code in many frames of a flash animation. Put the following code in frame 1:

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

The above code creates a variable, "theBall", of type MovieClip and attaches the MovieClip "ball" which is in the library. The code then positions the MovieClip at location (200,200) on the screen.

Next, create 10 more frames and put the following line of code into each frame:

```
theBall._x += 5 ;
```

As you can see, the line of code that is run in each frame moves the ball over 5 pixels. As a result, the ball moves to the right at a rate of 5 pixels per frame. When the last frame is reached the animation starts over at the beginning.

CODE EXAMPLE (from web site): 9/14, c2_movingBall_manyFrames

Again, if there is only one frame in an animation, Flash decides to NOT play the movie over and over. The reasoning from flash's point of view, nothing can change hence why restart it? But you will notice that in our moving ball example in section 2.2 that there is

only one frame yet the ball is moving. This is because of the of our declaration of the onEnterFrame function:

```
onEnterFrame = function () {
   theBall._x = theBall._x + 4 ;
}
```

A function is a chunk of code that is executed by calling the function name. We will explain functions in much more detail in chapter ZZZ. For now just realize that the code associated with the function named "onEnterFrame" is just the one line of code between the curly braces {}. We could have put more code in between the braces and all the code would be run each time the function is run.

The onEnterFrame function is a special function built into ActionScript. Declaration of this function sets up a periodic timer. Many people have a watch they can set to beep every hour, much to the chagrin of classmates, teachers, and parents. The onEnterFrame function is similar, the time between "beeps" is the time it takes to play a frame. If the frame rate is 24 Frames Per Second (FPS) then the time between calls to onEnterFrame will be $1/24^{th}$ of a second. The code inside the function is executed every 1/FPS seconds. This is an example of **event driven programming.** The event here is the expiration of the time interval, the **event handler** is the function that responds to the event, in this case it is the code defined for the onEnterFrame function.

This is a lot to swallow in the first chapter, don't worry if it does not make 100% sense, we will come back to functions and event driven programming later. Again, this should just sort-of make sense.

2.7 The Coordinate System

Graphics programming languages need a means to specify where on the screen objects are located. This is done with a coordinate system. I am sure we all remember plotting functions in high school algebra (I found it amusing when instructors would give you a set of mystery functions to plot, almost always a smiley face, I guess because it is so much easier say than plotting the Minotaur goring a victim in the labyrinth). Moving along.... the coordinate system always looked like the one shown in Figure 2.coord1:

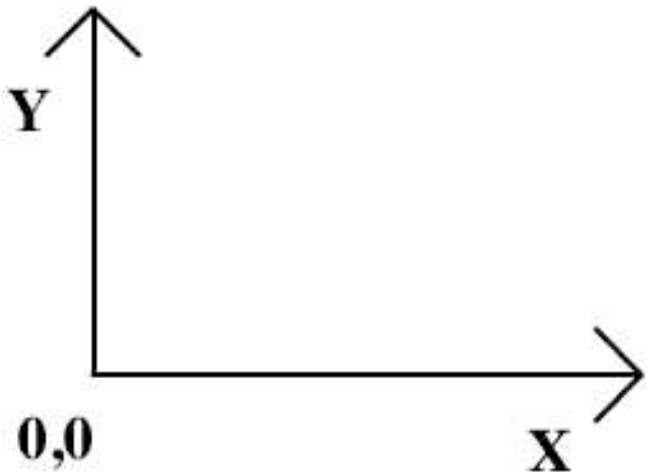

Figure 2.coord1

But in most computer graphics language, the Y-axis actually increases going DOWN as in Figure 2.coord2 below:

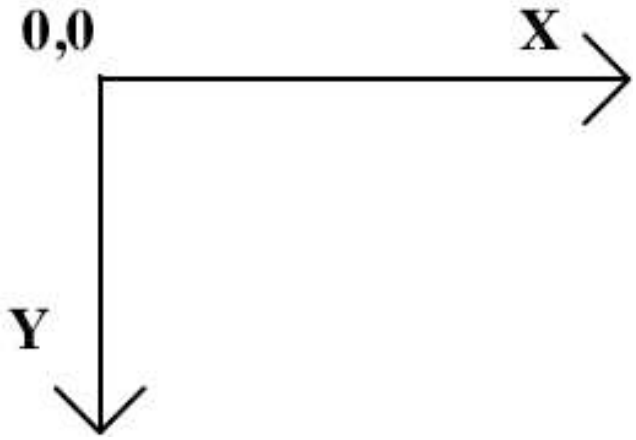

Figure 2.coord2

This difference is very important to keep straight when writing graphics code. In your head you may say, "I want to move the ball up", but in your code you have to decrease, not increase, the y-coordinate of the ball to make the ball move up.

EXERCISES:

1) Write code to move a ball from right to left. Do both with multiple frames and with the onEnterFrame approach.
2) Write code to move a ball from the lower part of the screen and up.

3) Write code to move the ball from the top down.