Written by Billy D. Spelchan for www.BlazingGames.com

Chapter 4

Action Script Basics

Contents

While it is quite possible to create a Flash animation without any Action Script, if you want any interactivity in your movie you are going to have to use Action Script. This chapter takes a brief look at the basics of Action Script.

- What is Action Script? - A look at what Action Script is.
- Comments - How commenting works.
- Variables - What variables are and how to set them.
- Using Classes (Objects) - Using built in classes.
- Math - Everybody's favourite subject.
- Functions - Making commonly used code reusable.
- The IF statement - Decision making.
- Writing code that will only run once - Creating a block of code that only runs once.

What is Action Script?

Action Script is the scripting language that is used by Flash. A scripting language is required in order to add non-linear elements to a Flash movie. Non-linear elements are generally things that are set into motion by things that the user does to interact with the Flash movie. The best example of this would be the user clicking on a button. The Flash movie has no idea when the user is going to click on the button. Likewise, if there is more than one button, the Flash program is going to need to know what to do based on what button is pressed.

Action Script, is based on the ECMA-262 standard. This standard is better known to people as JavaScript. JavaScript is loosely based on the Java language, which itself is based on the C++ language which is an object oriented version of the C language. In other words, understanding Action Script will make it easier to learn real programming languages like Java or C++.

Action Script is a very powerful scripting language. In fact, you can pretty much do anything in Action Script that you could do in a real programming language. The big difference is that Action Script is interpreted, and as such runs slower. Likewise, there are some restrictions on how long a script tied to a single frame can run and Action Script has no support for threading.

Games require interaction in order to be playable. For this reason, all games are going to require some Action Script. Still, in many cases you can get away with a minimal amount of simple Action Script.

Comments

Comments are simply blocks of text that Action Script ignores. While this may not sound useful, the purpose of a comment is to provide a way for the programmer to leave information about how the script works so that when he or another programmer comes back to the script at a future date they will be able to understand what is going on. There are two types of comments. Single line and multi-line comments.

Single line comments start with a double slash (//). Everything on the line after the slashes is the comment. Multi-line comments start with a slash immediately followed by an asterisk (/*). The comment ends with an asterisk followed by a slash. Anything between the start of the comment and the end of the comment is considered to be a comment.

Here is some examples:

```
// single line comment here
/* A multi-line comment here.
 * Note that the asterisk on this line is not necessary.
 * Asterisks are only included here to make the comment look nice!
 */
/* A multi-line comment on a single line is allowed */
x = x /* they can also be in the middle of a line of code */ + 4;
```

Variables

A variable is essentially a name that refers to a bit of information. The information that the variable can hold can be changed (hence, that is why it is called a variable). Action Script doesn't care what information the variable contains, though some programming languages do require that you have specific types of variables for holding specific types of information.

You set a value to a variable by using the equals sign and can do various operations on the variable. Variables can be set to numeric values, String values, and can even be a function or instance of an object. We will discuss these things in further detail in later sections and chapters. First, here is a bit of sample code that shows various variables being assigned values.

```
x = 11;
y = 7;
text = "Strings need to be enclosed in quotes";
function_variable = function() { /*code here*/ };
sound_object = new Sound();
```

Using Classes (Objects)

Flash is an Object Oriented scripting language. What this means is that the language knows how to create objects, known as classes. Classes are essentially a combination of variables combined with the functions used to manipulate those variables. While there are a lot more features of object oriented programming than this, there is no real need to cover them at this time.

You use a class by creating an instance of it. The variables in every instance are unique to that instance. You can, in theory, have as many instances of a class as you wish (in reality you have a limited amount of memory so you are limited to the number of instances). In fact, your Flash movie is itself an instance of the MovieClip class.

To create an instance of a movie clip you can quite often just use Flash to drag the object (be it dynamic text, a movie clip, or a component) onto the current frame and give it an instance name using the properties panel. Some classes are not visual in nature so you have to manually create them using the new operator as follows.

```
variable_name = new ClassName();
```

Objects do take up memory. This memory will continue to be used until all references to an object have been removed. To remove an object (or at least one of the reference to the object) you use the delete keyword. All variables that are assigned to that object need to be deleted (or changed to refer to something else) before the object will be removed from memory.

```
delete variable_name;
```

To access a variable or function that belongs to an object, you use periods. An exception to this is if you are writing code within the class. In that case, it is assumed you are referring to the class. This is why you can use MovieClip functions in your code without specifying what object you are referring to. Here are some examples.

```
otherMovieClip.gotoAndPlay(1);
       Math.cos(angle);
    otherMovieClip._x = 23;
    otherMovieClip._y = 32;
```

Math

When using variables that use numbers in Flash, it is possible to do mathematical operations on the variables. Math in Flash uses algebraic order of operations (meaning that numbers will be multiplied or divided before they are added or subtracted unless brackets are used to group operations). Adding and subtracting uses the plus and minus sign. Multiplication is done by using the asterisk (*) symbol and dividing is done by using the slash (/) symbol. You can also get the remainder of a division by using the percent symbol.

Flash also supports special decrement and increment operators. The double minus signs represent the decrement operation and essentially means deduct one from the variable this operator is used on. There is also an increment operator which is represented by double plus signs. The operator can be placed before or after the variable. If placed before the variable, the operation is done immediately. If placed after the operator, the operation will only be performed after any other operations are done. For instance, the statement x=y++ would place the value of y into x and then increase y.

In programming languages it is valid to have a statement such as x = x + 3. This would take the value that is currently in x and increase it by 3. This is actually very common. So common, that Flash has a shortcut for doing these operations. By placing a mathematical operator (+, -, *, /) before an equals sign you tell flash to do that operation on the lefthand variable. So the statement x *= 2 would actually be resolved as x = x * 2. While this doesn't sound like that big of a time saver, if you use long variable names you will quickly come to appreciate this shortcut.

More advanced mathematical operations, such as sin and cosine, can be done by using the Math class. You can call these functions simply by using the format Math.function(). I will explain these functions as we use them, but you can also read about them in the Action Script Dictionary.

Strings, a term used for a block of text, also can use one particular math operator. That being the + operator. Adding strings, better known as concatenation, simply joins the strings together. Adding a number to a string concatenates the string equivalent of the number to the end of the string.

Functions

Functions, which are known as methods in Java, are named blocks of code that can be called by using the name of the function. This allows you to create a block of code that can be re-used without having to duplicate the code every time it is used. Functions are defined by using the function keyword. Functions can also be given parameters. Parameters are values that the function can use. You can have a function use as many parameters as you need by separating them with commas. Functions may also optionally return a value by using the return keyword.

An example of a function would be:

```
function square(number)
{
    return number * number;
}
```

This function would simply return the square of the number passed to it. To use this function, you simply need to call it, like this:

```
x = square(2);
```

The parameter passed can be a variable or it can be a constant value. A constant being any value that will never change, such as the 2 above.

It is also possible to assign a function to a variable, as you have seen in an earlier section. In the earlier section you seen a function being created in-line to be assigned to a variable. To assign an already existing function to a variable you simply uses the function's name without any parentheses and followed by a semi-colon.

```
variable_holding_square = square;
```

The IF statement

One reason for having a scripting language is to be able to control the flow of a movie. Controlling something requires making decisions. This is what the if statement does. The if statement uses the following format:

if (condition) statement to perform if condition is true
else statement to perform if condition is not true

The else part of the if statement is optional. If you wish to have more than one statement after the if or the else, you should place the block of statements into a pair of curly braces. In fact, in any situation where a single statement is required, you can use a pair of curly braces to hold a block of statements. Here is an example.

```
if (x == 1)
{
    trace("x is assigned a value of 1");
    y = 23;
}
```

The condition is simply a mathematical condition, but two equal signs are used together to indicate equals while a != combination is used for not equal. The greater than and less than symbols (<, <=, >, >=) can also be used.

Conditions can be combined to form more complex conditions. This is done by placing each condition in it's own set of brackets and then placing an and (&&) or an or (||) symbol between them.

The and (&&) operation means that both conditions must be true if the condition is to be true. If either or both conditions are false then the condition is false.

The or (||) operation just requires that one of the conditions is true, though both conditions being true is also acceptable.

Writing code that will only run once.

Sometimes your Flash movie may have code that only needs to be executed once and shouldn't be ran again if that frame happens to be shown a second time. Flash does not have a built in way of doing this, so a bit of trickery is needed to do this.

Flash has the ability to create variables on the fly. Any variable that is created does not have a value until something is assigned to it (by using the equals sign). Flash has the ability to check to see if a variable has been initialized, so all that has to be done to make sure a section of code has not already been executed is to create a unique variable for that block of code that will only be assigned a value in that block. Before running the block of code we then check to see if the code has been executed by using an if statement (explained in the next sidebar).

If the variable has been assigned a value, we immediately return from the function. Otherwise, we assign a value to that variable then proceed with the function.

```
function initOnce()
{
      if (initOnceInitialized != undefined)
            return;
      initOnceInitialized = true;
      // additional initialization code goes here
}
```