CGT 353 Lecture 10

ActionScript 101

Introduction

- ActionScript has come a long way...
  - AS 1.0 based on **JavaScript...**
  - AS 2.0 trending more toward an OOP language (C#, Java, C++, etc)...
  - AS 3.0 really like an OOP language (C#, Java, C++, etc)...

- Fully functioning OOP scripting language...
- Superior to the procedural/event driven approach...
- Allows you to provide both action-oriented instructions (*do this*) and logic-oriented instructions (*analyze this before doing that)*

For This Class

- You will learn the most current version, AS 3.0
- However, you will learn some AS 2.0 as we go along
- AS 2.0 is much different from 3.0...so you should be able to recognize it

The Early 1.0 Event Model

```
on (press) {
    // set the cost of Ron's lunch
    var lunchCost: Number = 4.25;

    // set the local sales tax percentage
     var taxPercentage: Number = .06;

    // determine the dollar amount of tax
    var totalTax: Number = lunchCost * taxPercentage

    // figure total cost of sale
    var totalCost: Number = lunchCost + totalTax;

    // write message
    myTextfield.text = "The total cost of your purchase is " + TotalTax;

    // make the dollar sign movie clip pop up
    dollarSign.gotoAndPlay (10);
}
```

ActionScript Terminology

- **Events -** things that occur during playback of a movie that trigger the execution of a particular script
- **Event handlers** - events to which objects can respond
- **Actions -** any line that instructs Flash to do, set, create, change, load, or delete something. Comprise most of the lines within curly braces{}and are separated by semicolons
  - **Ex. dollarSign.gotoAndPlay (10);**

ActionScript Terminology

- **Curly Braces {} -** anything that falls between them signifies an action or actions the script must perform
- As a result of this - {do this}
- **Semicolons -** used to separate multiple actions
- **Dot syntax** - specific type of coding syntax
  - Ex. _root.country.state.city.Lafayette

ActionScript Terminology

- **Note:** Because ActionScript is object-oriented, most tasks are accomplished by:

- changing a property of an object

- telling and object to do something by invoking a method

- Ex. chair._height = 50; (property change)
  - chair.play(); (method invocation)

ActionScript Terminology

- **Classes -** templates used to create objects in the application that share the same properties and methods.
- **Packages** – groups of classes that perform specific functions that can be imported as needed.
- **Objects -** a group of functions and properties that adhere to a specific class
  - Ex. Movieclip objects, string objects, color objects, and sound objects

ActionScript Terminology

- **Keywords -** words reserved for specific purposes within ActionScript syntax.
  - Includes **on, break, case, continue, delete, do, else, for, function, if, in, instanceOf, new, return, switch, this, typeOf, var, void, while, and with**

- **Instances -** individual objects based on a class
  - Ex. Individual instances of the same symbol

ActionScript Terminology

- **Methods -** predefined routines that perform a specific task for a particular class or object
- **Properties -** characteristics or attributes of an object
- **Commands -** code words that perform a specific preset function in an environment
- **Arguments (or parameters) -** optional bits of information sent to methods and functions

ActionScript Terminology

- **Variables** - containers for data defined by scope (length of existence) and data type (string, expression, etc)
- **Global Variables** - variables that are active and accessible anywhere in the movie/ defined using the _global keyword
- **Local Variables** - variables accessible only within a specific function/ defined using the *var* keyword
- **Timeline Variables** - general variables not defined with *var or _global* that exist as long as the timeline they are in continues to play

ActionScript Terminology

- **String** - any data element that consists of text as opposed to a numeral which are actual numbers
  - **Note:** Quotation marks are used to denote textural data in a script, so "3" would be taken as a string in Flash while 3 would be treated as a number
- **Operators -** programming elements that perform calculations, comparisons, or assignments

ActionScript Terminology

- **Expressions** - combination of code statements that can include variables, properties, functions, methods, and operators that must be evaluated

- **Array** - special type of variable that can store multiple values

- **Comments** - script lines preceded by two forward slash marks (//) used to insert descriptive notes into the code

Coding Strategies

- Back in the old days, timeline scripting was all you could do...
- With 3.0 (and 2.0, but to a lesser extent) ActionScript is driven with **classes.**
- We will continue to use timeline scripting as the basis for our development.

Timeline Scripting

- In Flash timeline scripting, there are **three basic objects** that can react to events:
  - **Frame**
  - **Button**
  - **Movie Clip**

Timeline Scripting

- These objects can respond to **2 primary events:**
  - Events related to user interaction (mouse or keyboard)
  - Events associated with timeline frames

- So you can have code execute in one of **two ways**:
  - When movie hits a certain frame - **frame action**
  - When user does something to a button or movie clip - **movie clip action**

The Correct Method for Coding ActionScript

- The old approach consisted of attaching scripts to buttons and movie clips as well as frames.
- Still good for basic, fast creations.....
- AS 2.0 approach encouraged the writing of all scripts in **single frames or in external files**
- This evolved because its hard to find code when they are scattered throughout a movie in various symbols.

Sample Code

- **1.0 Method:**

```
on (press)
{
        getURL(http://www.purdue.edu);
}
```

- **2.0 Method:**

```
mybutton.onPress = function ()
{
        getURL(http://www.purdue.edu);
}
```

Sample Code

- **3.0 Method**

```
import flash.events.MouseEvent;

newButton.addEventListener(MouseEvent.CLICK, myClick);

function myClick(event:MouseEvent):void
{
        navigateToURL(new URLRequest("http://www.purdue.edu"));
}
```

AS 3.0 Key Core Language Features

- 1st class support (as opposed to second-class?) for object-oriented constructs – classes, objects, and interfaces

- **Single-threaded execution model** (as opposed to multiple threaded model, which is faster and more flexible)

- **Optional compile-time type checking** (for data typing) – run into fewer logical errors b/c it's inherently more strict

AS 3.0 Key Core Language Features

- New dynamic features such as runtime creation of constructor functions

- **Runtime exceptions** – exception: the occurrence of some condition that changes the normal flow of execution (errors), used only for signaling error (exceptional) conditions.
  - In AS 2.0….many exceptions went silent…so you didn't catch them
  - AS 3.0 handles errors better, which helps you debug

- Direct support for XML as a built-in data type
- Namespaces for qualifying identifiers (names)
- Regular expressions

Flash Client RunTime Environments

- **Runtime environments (RTEs or runtimes)** are programs that can run ActionScript.

- Since AS is run by these portable runtimes, AS is itself portable...
  - **Adobe AIR** – standalone on desktops
  - **Flash Player** – current player is 9
  - **Flash Lite** – mobile devices

- Each runtime environment is basically the same, but with a few different custom features that deal with the capabilities and security measures of each environment

Runtime API's

- Each RTE has it's own set of functions, variables, classes, and objects called by its own name.

- Flash Player = Flash Player API

Shared Features of all API's

1. Graphic and feature display
2. Hierarchical event structure
3. Text display and input
4. Mouse and keyboard control
5. Network operations for loading external data and communicating with server side apps
6. Audio playback
7. Printing
8. Communication with external local apps
9. Programming utilities

Development Tools

- Can write AS in Notepad if you wanted...
- Adobe Flash Authoring Application (or Tool) (FAT)
- Adobe FlashBuilder - an **integrated development environment (IDE)**
- Can use AS and/or MXML (for making user interfaces)

The ActionScript Virtual Machine

- **AVM2 is ten times faster than AVM1**
- AVM1 can run AS 1.0 and 2.0
- AVM2 can run AS 1, 2, and 3.0

How it All Works

- **AS 3.0** (for us humans)

  to

- **AS ByteCode (ABC)** (for computers)

  to

- **binary container** (.swf – compiled for the runtimes)
  - Flex and FAT use the same Flash compiler but different swf compilers
- All code is compiled twice first to ABC then to machine code

Migrating to AS 3.0

- With AS 2.0…the majority of commands we used were contained in the **MovieClip API.**
- In AS 3.0, most of the commands you will use are located in the **Display API.**
- The Movieclip API is now just mostly used as display object containers and to move about timelines.

For Other 2.0 – 3.0 Migration Tips

- http://www.adobe.com/devnet/flash/articles/first_as3_application.html

Syntax Issues

- **Two basic programming errors:**
  - **Logical** - how the code is designed
  - **Syntactical** - how the code is written

- **NOTE: Most programming errors are syntactical!**

Biggest Syntax Issues

- **Structural Details -** semicolons, parentheses, etc.
- **Case Sensitivity Issues**
- **Comments**
- **Dot Syntax and Targeting Paths**
- **The Output Window can help, but is by no means the perfect debugging tool.**

Biggest Syntax Issues

- **Curly brackets -** used to denote logical blocks of functional code most often used to define function definitions and control structures

- The physical location of the brackets is NOT critical, but should be standardized

Biggest Syntax Issues

- ```
  _root.sampleClip.onPress = function()
  {
        gotoAndPlay (1);
        stopAllSounds ();
  }
  ```

- NOTES:
  - Curly brackets line up vertically.
  - Curly brackets also line up vertically with the first letter of the event it belongs to above
  - Contents inside the curly brackets are indented
  ```

Biggest Syntax Issues

- **Semicolons** go at the end of each statement

- **Case Sensitivity:**
  - ActionScript is, for the most part, case sensitive....
  - ActionScript 3.0 properties are too.....but AS 2.0 are NOT

Comments

- Basically a method for leaving notes within your code.
- You are expected and required from this point on to include detailed comments in your code.
- You are expected to do this because:
  - It is professional
  - It will help you keep track of what you are trying to do
  - It will help others keep track of what you are trying to do
  - Most importantly, it helps your instructor keep track of what you are trying to do

Comments

- **Put comments wherever:**
  - The code is incomplete or needs modification
  - Generates a known error or a bug
  - Doesn't follow standard conventions
  - Is difficult to understand

Dot Syntax and Targeting

- In order to control objects, you have to **target** them in order to evoke their methods, access their properties, etc

- A **target** is the way of specifying a location of an object in the movie hierarchy in order to control that object.

- Most of the **general actions in Flash** do not require targets and are automatically directed to the main timeline.

Accessing Properties and Methods

- To access a method or property of an object, you use dot syntax to target it.
  - **2.0 Ex.**
    _root.RonsClip_mc._alpha
  - **3.0 Ex.**
    root.RonsClip_mc.alpha

Controlling Movie Clips and Buttons

- Like any other objects, movie clips and buttons can be controlled.

- In order to communicate with ANY symbol, you must provide it with an **instance name.**

- It is this, and not the symbol name, which must be referenced

Absolute vs. Relative Targets

- To communicate with objects, you have to target them either:
  1. based on the current object's location in the hierarchy
  2. based on a fixed point in the hierarchy (usually the main timeline
     - The problem with absolute targeting is when you move objects, the targets become unusable.
     - As such, is it usually better to stay with relative paths.
       - **Target:** MC1a
       - **From Object**: Button A
       - **Absolute Target:** _root.MC1.MC1a (2.0)
       - **Relative Target:** MC1.MC1a

Strict Data Typing

- **Strict data typing** is the ability to declare the data type of a variable when that variable is initialized.

- **Old Method:**
  - var myname = "Ron";

- **New Method:**
  - var myname:String = "Ron";
  - var myAge:Number = 31;
  - var myObject:Object = new Object();

- **Strict Data Typing with Functions:**
  - Problems arise when you build functions that return the wrong data type