ACTIONSCRIPT SESSION 3

Roy McElmurry

TextFormat

Syntax:

```
import flash.text.*;
var textField:TextField = new TextField();
…
var format:TextFormat = new TextFormat(font, size, color, …);
textField.setTextFormat(format);
```

- The *TextFormat* object can be used to set many other properties as well.

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/text/TextFormat.html

TextFieldAutoSize

Syntax:

```
import flash.text.*;
var textField:TextField = new TextField();
…
var format:TextFormat = new TextFormat(font, size, color, …);
textField.autoSize = TextFieldAutoSize.LEFT;
textField.setTextFormat(format);
```

- The auto size property must be set so that the *TextField* knows how it should respond to formatting
- In this case in treats the text as left justified

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/text/TextFieldAutoSize.html

For Each

Syntax:

```
var stuff:Array = {thing1, thing2, thing3, …};
for (var key:type in stuff) {
    //use key as index
    //get data with stuff[key]
}
```

- The for-each loop iterates over the indices of any iterable object, not the values
- You can then use the index to access that data at that index

More: Constants

FlashVars in HTML

Syntax:

```
<div>
    <object type="application/x-shockwave-flash" data="yourfile.swf"
                width="550" height="400" >
        <param name="yourfile" value="yourfile.swf" />
        <param name="FlashVars" value="key=value&key=value…" />
    </object>
</div>
```

- FlashVars are a way for us to load a flash movie with parameters
- The format is the same as that for HTML query strings
- You can also just add a query string to the end of the data attribute

FlashVars in AS

Syntax:

```
import flash.display.LoaderInfo;
…
var params:Object = LoaderInfo(this.root.loaderInfo).parameters;
for (var key:String in params) {
        //do stuff with the keys of the FlashVars
        //value = params[key]
}
```

- You can treat the params object as if it were an associative *Array*

Events

- Events are flags that are raised by certain objects for different purposes
- In many cases we want something to happen when this event occurs, some such events may include
  - A button being clicked
  - The user pressing a key on the keyboard
  - A picture having finished loading
  - A timer going off
- When we use *addEventListener* we are saying that we wish for the listener function to be executed when the given type of event occurs

Listener Functions

Syntax:

```
public function listener(ev:Type):void {
    //do stuff
}
```

- The name of the listener function must exactly match the function name given to *addEventListener()*
- The listener function must take a single parameter of the same type as that specified when you used *addEventListener()*
- The parameter contains valuable information about the event that one might want to use

Event

Syntax:

```
public function listener(ev:Type):void {
    //ev can be very useful
}
```

Fields	
ev.target	The object that started the event.
ev.type	Contains metadata about the sound such as artist, duration, title, etc.
ev.currentTarget	The object that is currently responding to the event

- The *Event* parameter stores valuable data
- *target* and *currentTarget* can differ when the same event is being listened for by multiple things

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/events/Event.html

Loading an Image

Syntax:

```
import flash.display.Loader;
import flash.display.Bitmap;
…
var loader:Loader = new Loader();
loader.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
loader.load("myimage.jpg");
…
public function loaded(e:Event):void {
        var img:Bitmap = new Bitmap(e.target.content.bitmapData);
        addChild(img);
}
```

Embed Resources

Syntax:

[Embed(source="**/filename.ext**")]
private var **name1**:Class;
private var **name2**:**Type** = new **name1**() as **Type**;

- Using the embed command is a shortcut for loading a resource and responding to it's load complete event
- This command must be placed in the fields area of your class
- In practice it would be better style to initialize *name2* in the constructor

Sound

Syntax:

```
import flash.media.Sound;
…
[Embed(source="/filename.mp3")]
private var name1:Class;
private var name2:Sound = new name1() as Sound;
```

Functions/Fields	
sound.play();	Plays the mp3, returns a SoundChannel object which could be used to tinker with the sound
sound.id3	Contains metadata about the sound such as artist, duration, title, etc.
sound.length	Returns the length of the mp3

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/media/Sound.html

Bitmap

Syntax:

```
import flash.display.Bitmap;
…
[Embed(source="/filename.jpg")]          //also png or gif
private var name1:Class;
private var name2:Bitmap = new name1() as Bitmap;
```

Functions	
pic.getBounds(obj:DisplayObject)	Returns a Rectangle object that defines the area of pic relative to obj
pic.hitTestObject(obj:DisplayObject);	Determines if pic is intersecting the given DisplayObject
pic.hitTestPoint(x:Number, y:Number);	Determines if pic intersects (x, y)

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/display/Bitmap.html

Game Loop

- In all games it is necessary to have a game loop
- A game loop is some logic that is performed every so often to maintain the game environment
- Things that need to happen regularly should be in the game loop
- Ex)
  - Moving things that are not user-controlled
  - Detecting collisions
- A game loop for pong might only move the ball

Motion Vector

- We want our ball to move in a random direction
- One approach is to store the direction of the ball in a *Point* object
- The *x* and *y* fields of the *Point* will represent the direction that the ball will move in
- When using a direction vector the *Point* must be normalized to be length one, otherwise speed will vary

Point

Syntax:

```
import flash.geom.Point;
…
var name:Point = new Point(x:Number, y:Number);
```

Functions	
Point.distance(p1:Point, p2:Point);	Returns the distance between the two points
Point.polar(len:Number, angle:Number);	Returns a Point with cartesian coordinates, where angle is in radians
p1.add(p2:Point);	Add the coordinates of p2 to p1
p1.offset(dx:Number, dy:Number);	Offset p1 by dx and dy

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/geom/Point.html

Collisions

- In games it is often important to know when objects on the screen are touching
- Even the simplest games need to respond to objects colliding with either other objects or specific points
- Examples
  - Pong – ball bounce
  - Worm – eating an apple
  - Galaga – bullets hitting enemies
  - Halo – pwning newbs

HitTestObject

Syntax:

```
var s1:Shape;
var s2:Shape;
…
s1.hitTestObject(s2);
```

- If the rectangle that surrounds *s1* intersects the rectangle that contains *s2*, then *hitTest* will return *true*, otherwise it returns *false*
- This can be called on any *DisplayObject*

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/display/DisplayObject.html#hitTestObject()

HitTestPoint

Syntax:

```
var s1:Shape;
…
s1.hitTestPoint(x, y);
```

- If the rectangle that surrounds s1 overlaps or intersects the point specified
- You can supply a third optional *boolean* parameter for whether to check if the actual pixels of the object are touching instead of just the bounding box
- This can called on any *DisplayObject*

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/display/DisplayObject.html#hitTestPoint()

KeyboardEvent

Syntax:

```
import flash.events.*;
…
stage.addEventListener(KeyboardEvent.KEY_DOWN, listener);
…
public function listener(ev:KeyboardEvent):void {
      //respond to user action
}
```

- There is also a *KEY_UP* event that you can use

KeyboardEvent

Fields	
ev.charCode	The ASCII value of the character that was pressed, all modifier keys have a keyCode of 0
ev.keyCode	A numeric code for the which button on the keyboard was pressed
ev.keyLocation	Where the key is that was pressed
ev.shiftKey	Boolean for whether the shift key was being pressed
ev.altKey	Boolean for whether the alt key was being pressed
ev.ctrlKey	Boolean for whether the control key was being pressed

- *KeyboardEvent*s have tons of useful information in them

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/events/KeyboardEvent.html

Common Codes

key	keyCode	charCode
w	119	87
a	97	65
s	115	83
d	100	68
space	32	32
up	0	38
left	0	37
right	0	39
down	0	40

- The *Keyboard* class contains constants for a lot of these so that we do not have to memorize the numbers

More: http://people.uncw.edu/tompkinsj/112/flashactionscript/keycodes.htm

More: http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/ui/Keyboard.html

KeyboardEvent Issues

- The *KEY_DOWN* event is fired once and then after a pause is fired regularly as we would expect, but the delay creates for a bad user experience
- Pressing a different key will overshadow a previous key that is still being pressed meaning that holding down two keys will only register one *KEY_DOWN* event
- We can fix this issue by using the *KEY_DOWN* and *KEY_UP* events
- The idea is to key track of which keys are down and then update in the game loop

KeyboardEvent Fix

Syntax:

```
import flash.events.*;
…
var keysPressed:Array = [];
stage.addEventListener(KeyboardEvent.KEY_DOWN, downListener);
stage.addEventListener(KeyboardEvent.KEY_UP, upListener);
…
public function downListener(ev:KeyboardEvent):void {
        keysPressed.push(ev.charCode);
}
public function upListener(ev:KeyboardEvent):void {
        removeKey(keysPressed, ev.charCode);
}
public function gameLoop():void {
        for (var key:int in keysPressed) {
                //respond to key being down
        }
}
//You need to write a removeKey function as well
```

Game Libraries

- Game programming can get complex and there can be lots of subtle bugs associated with it, like the *KEY_DOWN* event issues
- There are many libraries that help us with Flash game programming
- Ex)
  - Flixel with the FlashDevelop IDE
  - FlashPunk
  - APE (Actionscript Physics Engine)