Creating a Mobile Game

In this unit, you will learn how to build a game for use on mobile devices.

Objectives

After completing this unit, you should be able to:

- ▶Describe the benefits of using ActionScript for Animation
- ▶Animate a `MovieClip` using ActionScript
- ▶Use the `onEnterFrame` event to continually execute a statement over time
- ▶Stop and `delete` animation handlers to conserve resources

Using ActionScript to animate

If you have used Flash regularly, you're probably used to animating with the Timeline. However, you can also use ActionScript to animate content.

- Animating with ActionScript provides the following benefits:
  - File size - ActionScript helps minimize file size because it maximizes code reusability and provides ways to break a movie into a series of smaller movies.
  - Control - You gain precise control over the movie clip instances and frames in the movie, without having to repeatedly use the Property inspector and Align panel to position content. With ActionScript, numbers determine all animation.
  - Interaction - By using ActionScript to animate, you gain access to interactive features that are not available using Timeline-based animation. For example, using ActionScript, you can have a movie clip instance react to an event, such as hitting another object. It is difficult if not impossible to build a game using purely Timeline based animation.
  - Dynamic animation - By using ActionScript to animate, you can alter an animation based on information received by Adobe Flash - such as user interaction with a device - to create effects essential to game development.

Changing MovieClip properties over time

- ▶Since animating a movie clip requires continually changing its properties over time you need a way to tell Flash Lite to keep executing a statement at timed intervals.
- ▶The onEnterFrame event can be a solution

Responding to the onEnterFrame event

- ▸Events, like `onRelease`, broadcast a signal when a particular user or system event occurs, such as after a user releases the mouse.
- ▸You write event handlers to execute code at that time.
- ▸The `onEnterFrame` event broadcasts at the frame rate of the current movie. For example, if the frame rate for the current movie is 12 frames per second, `onEnterFrame` is broadcast, and its event handler called, 12 times per second.

Responding to the onEnterFrame event

An `onEnterFrame` event handler may use either syntax:

```
my_mc.onEnterFrame = function():Void
{
 // statements
}

-or-

my_mc.onEnterFrame = doEnterFrame;
function doEnterFrame():Void
{
 // statements
}
```

Animating by changing properties over time

Changing a movie clip property is like changing any other property. You use this syntax:

```
my_mc.property = value;
```

where `value` is either a literal value or an expression.

When you animate, you will commonly enter an expression, such as an value increment (`++`), so that the property will change over time as the `onEnterFrame` event fires.

For example, the following code changes the `_xscale` property for a movie clip as long as that movie clip exists.

```
my_mc.onEnterFrame = function ():Void
{
 this._xscale += 10;
 // same as: this._xscale = this._xscale + 10;
}
```

Setting properties in relation to the Stage

The width and height of the Stage can be determined through code, using the width and height properties of the static `Stage` class.

```
block_mc._width = Stage.width / 2;
block_mc._height = Stage.height / 2;
```

Walkthrough 1: Using ActionScript for Animation

In this walkthrough, you will perform the following tasks:

- ▶Add the `player MovieClip`
- ▶Create event handlers to move the `player MovieClip`
- ▶Create the methods to animate the `player MovieClip`
- ▶Stop the `player MovieClip` animation

Changing an animation interactively

- ▶When animating with ActionScript, you can change an animation while the movie runs, based on programmed conditions.
  - For example, when animating, one condition you may want to check is whether one movie clip has touched another. You can then reverse the animation's direction, make the movie clip invisible, increase the movie clip's size, and so on.
  - Any change you could make at design time could be made at run time with ActionScript.

Using the hitTest() method

- To check whether movie clips overlap, you use a movie clip method called `hitTest()`.
  - This method lets you check to see whether the movie clip has overlapped a set of coordinates or another movie clip.
  - Its syntax is as follows:

```
my_mc.hitTest(x,y)
my_mc.hitTest(target)
```

  - where `x,y` is a position on the `Stage`, and `target` is a reference to a movie clip, button or text field instance.
  - Returns `true` if the move clip passed as a reference is intersecting the bounding box of the move clip on which the method is called,
  - else `false` is returned.

Using the hitTest() method

- You can use `hitTest()` inside an `onEnterFrame` event handler to continually test whether one movie clip overlaps another. For example, the following code continually tests whether to make `oval_mc` invisible if it overlaps `circle_mc`:

```
circle_mc.onEnterFrame = function():Void {
     if (this.hitTest(oval_mc))
   {
    // increase oval_mc scale during overlap
    oval_mc._xscale = 150;
    oval_mc._yscale = 150;
   }
   else
   {
    // retore oval_mc scale if not overlapping
    oval_mc._xscale = 100;
    oval_mc._yscale = 100;
   }
}
```

Referring to movie clips dynamically

- ActionScript can refer to object properties either using dot notation or bracket notation. The following are equivalent.

```
product_mc._x = 25;
- or -
product_mc["_x"] = 25;
```

- If using bracket notation, the property name may be held within a variable.

```
var propertyName:String = "_x";
product_mc[propertyName] = 25;
```

Adobe

Referring to movie clips dynamically

- The instance name of any object may be treated as a property of its timeline, referred to using a path keyword: `_root`, `_parent`, or `this`.

```
product_mc._x = 25;
- or -
this.product_mc._x = 25;
- or -
this["product_mc"]._x = 25;
- or -
var instanceName:String = "product_mc";
this[instanceName]._x = 25;
```

Referring to movie clips dynamically

▶An instance name or variable referred to using bracket notation may be generated dynamically.

```
// set _x for product1_mc through product10_mc
var instanceName:String = "";
for (var i:Number = 1; i <= 10; i++)
{
 instanceName = "product" + i + "_mc";
 this[instanceName]._x = 25;
}
```

Removing a MovieClip at runtime

- ▶You remove a `MovieClip` in ActionScript by calling its `unloadMovie()` method.
- ▶The `unloadMovie()` method is the opposite of the `loadMovie()` method: it directs Adobe Flash to remove the indicated movie clip from memory. Its syntax is as follows:

```
instanceName.unloadMovie();
```

Adobe

Walkthrough 2: Testing for Collisions

In this walkthrough, you will perform the following tasks:

- ►Add the `ball` `MovieClip`
- ►Add instructions to start the game
- ►Start the game
- ►Test if the `ball` hits an object
- ►Set up the game bounds
- ►Test for collision with the blocks

Stopping an animation

Until now, animations run continually. Often, you will want to stop animation after a visual effect has been achieved: for instance, when an alpha effect reaches a certain value, or when a movie clip instance is at a certain position on the `Stage`. To accomplish this, you must stop the animation when a certain condition is met.

Adobe

Testing for the end of an animation

You test for the end of an animation by using an `if` statement within an `onEnterFrame` event handler. Flash continually checks the statement for validity. The syntax could be as follows:

```
my_mc.onEnterFrame = function():Void{
 if (condition){
  // statement(s) to stop the animation
 }
}
```

Testing for the end of an animation

In the following example, the movie clip slides in from the left side of the `Stage`. It is initially off the `Stage`; it moves to the right, with its `_x` property continually incremented by units of `30` as `onEnterFrame` is executed. It stops moving when its `_x` property is greater than or equal to 0, as its `_x` property will then be set specifically to 4.

```
my_mc._x = -300;
my_mc.onEnterFrame = function():Void
{
 this._x += 30;
 if (this._x >= 0)
 {
  this._x = 4;
 }
}
```

Note: In this code, the event handler keeps firing, though the object no longer moves.

Deleting the onEnterFrame event

The `onEnterFrame` event is somewhat resource intensive, as a timed process must be created and watched inside the Flash Lite player, for each such handler. Once the visual animation is complete, there is no reason to continue running `onEnterFrame` and using CPU cycles.

To stop an `onEnterFrame` process, delete the `onEnterFrame` event.

In the previous section, you animated a movie clip with `onEnterFrame` using the following code.

```
my_mc._x=-300;
my_mc.onEnterFrame = function():Void
{
 this._x += 30;
 if (this._x >= 0)
 {
  this._x = 4;
 }
}
```

Deleting the onEnterFrame event

You can delete the `onEnterFrame` event associated with the movie clip so that it no longer consumes CPU resources, using the `delete` operator. To do this, you use the following code.

```
delete instanceName.onEnterFrame;
```

Deleting the onEnterFrame event

In this example, place the `delete` inside the conditional statement, as in the following code.

```
my_mc._x = -300;
my_mc.onEnterFrame = function():Void
{
 this._x += 0;
 if (this._x >= 0)
 {
  this._x = 4;
  // stop animation by terminating the event
  delete this.onEnterFrame;
 }
}
```

Note: In this example, `my_mc` *would remain on the* `Stage` *at* `_x`*:*`4`

Unloading the movie clip containing onEnterFrame

In the previous example, you wanted the movie clip itself to remain on screen after the animation completes. In some cases, you may want to remove the movie clip when the animation completes.

Unloading the movie clip also removes any associated `onEnterFrame` events, so their assigned event handlers will not longer fire.

Unloading the movie clip containing onEnterFrame

Changing the previous example to remove the entire movie clip after the animation is complete, the code becomes:

```
my_mc._x = -300;
my_mc.onEnterFrame = function():Void
{
 this._x += 30;
 if (this._x >= 0)
 {
  this._x = 4;
  this.unloadMovie();
 }
}
```

Note: In this example, `my_mc` *would disappear, rather than sit at* `_x:4`

Walkthrough 3: Scoring and Ending the Game

In this walkthrough, you will perform the following tasks:

- ►Add points when a block is removed
- ►Track the number of blocks
- ►End the game when all the blocks are gone

Summary

- ▸Objects may be animated by updating their visual properties within an `onEnterFrame` event handler.
- ▸The `Stage` object provides runtime access to the stage dimensions and related values.
- ▸A conditional (`if`) statement allows different code blocks to run depending on whether a value is `true` or `false`.
- ▸The `hitTest()` method - a function which may be called through any `MovieClip` - returns `true` if its target is intersecting the object or position passed as its argument.
- ▸A property may be referenced using either dot notation or bracket notation.
- ▸An instance name may be referenced as a property of its timeline.
- ▸Bracket notation allows property and instance names to be created by assigning an expression to a variable, which is then used in the brackets.

Summary

- ▶`MovieClip` objects are removed from screen by invoking the `unloadMovie()` method on the object to be removed.
- ▶Ending conditions for code-based animations are tested using a conditional (`if`) statement within an `onEnterFrame` event handler.
- ▶Once complete, an `onEnterFrame` event handler should be removed using either the `delete` keyword, or by calling `unloadMovie()` on the `MovieClip` to which it's attached.