

// Line 4 declares a variable that will be used to hold the random number that is "spun."

// Line 8 stops the Tween animation from playing. Unlike a Timer object, a Tween object immediately starts to "play" when it is created.

// Line 10 has the Tween object listen for the MOTION_FINISHED event so the final result of the spin can be displayed only after the spin has completed.

// Lines 15-21 are the meat of the program, executed when the `mcArrow` object is clicked. Here we pick our random number from 1 to 8, display the text "Spinning..", set the beginning and ending values for the Tween object, and let it start playing.

// On line 23, we make the arrow "unclickable" while the Tween is playing.

// Lines 26-29 define the two things to do when the spinning motion have finished: display the result of the spin and make the arrow clickable again so another spin can be made.

```
4 var rn:Number;
5
6 var tw:Tween = new Tween(mcArrow, "rotation",
    Strong.easeOut, 0, 360, 3, true);
7
8 tw.stop();
9
10 tw.addEventListener(
    TweenEvent.MOTION_FINISH, spinStopped);
11
12 mcArrow.addEventListener(MouseEvent.CLICK, spin);
13
14 function spin(mevt:MouseEvent):void {
15     rn = Math.ceil(8*Math.random());
16
17     txtCount.text = "Spinning..";
18
19     tw.begin = mcArrow.rotation;
20     tw.finish = 720 + rn*45;
21     tw.start();
22
23     mcArrow.removeEventListener(
        MouseEvent.CLICK, spin);
24 }
25
26 function spinStopped(twevt:TweenEvent):void {
27     txtCount.text = String(rn);
28     mcArrow.addEventListener(
        MouseEvent.CLICK, spin);
29 }
```

See **GameSpinner.fla** in **Chapter5** of the download folder
http://flashandmath.com/appletsbook_files
for the complete source code for this example.

THE "REPEAT EFFECT" FOR BUTTONS

From our first examples we have been frustrated that clicking and holding a button with the mouse does not produce the same effect as pressing and holding a key on the keyboard. We will see here that we can emulate the expected behavior for mouse clicks on buttons through a clever use of animation. In this case, rather than the animation directly making something move on the screen, the animation repeatedly calls a function previously set up to handle a mouse click. In the example below, we manage this animation with the ENTER_FRAME event, but the example could be easily modified to use a Timer object instead.

We will add this functionality to an example that has already been developed, so there is no benefit to showing a separate screen shot here. It is already shown in **Figure 6** on page 51 of **Chapter 3**. However, it is instructive to try the interface at the following link before thinking about how to manage this interaction.

For a finished version of this applet, see
<http://www.flashandmath.com/appletsbook/CircleFillRepeat.html>

Open the file **CircleFillSprite.fla** from **Chapter3** of the download folder, and rename it to something suitable like **myRepeat.fla**. Use the following code to replace the last twenty lines or so of the script already written in the **Actions** panel.

A significant difference here is the use of the `MOUSE_DOWN` event (instead of `CLICK`) to trigger the button. In addition, this event is triggering the start of an animation with calls to the `changeAngle` function each time the `ENTER_FRAME` event is detected.

```
btnLess.addEventListener(
    MouseEvent.CLICK, startDecrease);

function startDecrease(evt:MouseEvent):void {
    degChange = -Math.abs(degChange);
    stage.addEventListener(
        Event.ENTER_FRAME, changeAngle);
}

btnMore.addEventListener(
    MouseEvent.CLICK, startIncrease);

function startIncrease(evt:MouseEvent):void {
    degChange = Math.abs(degChange);
    stage.addEventListener(
        Event.ENTER_FRAME, changeAngle);
}

function changeAngle(evt:Event):void {
    degree = degree + degChange;
    if (degree > 360) {
        degree = degree - 360;
    }
    if (degree < 0) {
        degree = degree + 360;
    }
    updateArrow(degree);
}
```

// Note that the value of `degChange` is made negative when the `startDecrease` function is called, and positive when the `startIncrease` function is called. We use the absolute value function `Math.abs(...)` to make this easy to manage.

// Each time `ENTER_FRAME` is detected, the `changeAngle` function makes the appropriate change to degree and calls the function `updateArrow` to update the graphics on screen.

// The **MOUSE _ UP** and **MOUSE _ OUT** events on either button will call the **stopAngle** function, which simply removes the listener for the **ENTER _ FRAME** event, effectively halting the animation.

```
}  
  
btnMore.addEventListener(  
    MouseEvent.MOUSE _ UP, stopAngle);  
  
btnMore.addEventListener(  
    MouseEvent.MOUSE _ OUT, stopAngle);  
  
btnLess.addEventListener(  
    MouseEvent.MOUSE _ UP, stopAngle);  
  
btnLess.addEventListener(  
    MouseEvent.MOUSE _ OUT, stopAngle);  
  
function stopAngle(mevt:MouseEvent):void {  
    stage.removeEventListener(  
        Event.ENTER _ FRAME, changeAngle);  
}  
}
```

See **CircleFillRepeat.fla** in **Chapter5** of the download folder http://flashandmath.com/appletsbook_files for the complete source code for this example.

ADDITIONAL RESOURCES

Downloads for this chapter

To see the source code for the examples in this chapter, look in **Chapter5** of the download folder (http://flashandmath.com/appletsbook_files) for the following:

- ▶ The file **BounceEnterFrame.fla** illustrates the quick method of using the **ENTER_FRAME** event to achieving animation with ActionScript code.
- ▶ The file **BounceTimer.fla** repeats the previous effect using a **Timer** object. We include a mechanism by which the user can provide an animation speed while the application is running in order to highlight a key difference between a **Timer**-based animation and an **ENTER_FRAME** animation.
- ▶ The file **GameSpinner.fla** shows how to use a **Tween** object to achieve the motion of a spinning arrow — it starts off as uniform motion and then “eases out” of the animation as it nears a target number. This file also uses the **TweenEvent** class to detect when the **Tween** animation has finished.