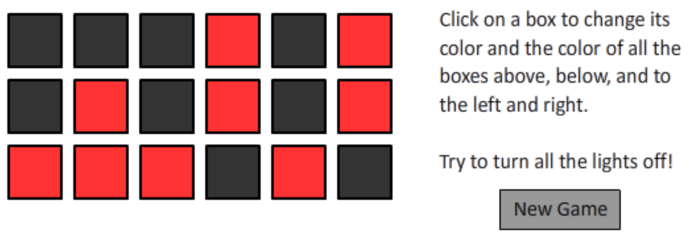
Flash Tutorial: Creating Listeners Dynamically

You should already be familiar with adding listeners to objects using Flash. These listeners are functions that take an event as an argument, and "handle" the event by doing something.

Sometimes we are creating a Flash program that needs to have several listeners that are very similar, but not identical. One example is the "Lights Out" game:

http://webspace.ship.edu/jehamb/flash/lightsout.html


Click on a box to change its color and the color of all the boxes above, below, and to the left and right.
Try to turn all the lights off!
New Game

This program draws a randomized grid of squares that the user can click on. Each square is a movie clip that needs to have a listener to handle the user clicking on it. The tricky part is that the listener needs to change not only the color of the square that was clicked on, but also the colors of the adjacent squares. In effect, the listener needs more than just a link to the object that was clicked on. It needs to know which squares are adjacent to that object as well.

However, the general structure of these listeners should all be the same. That is, once the program knows *which* square was clicked on, it will handle this click in pretty much the same way for each square. So we really don't want to manually write essentially the same listener a bunch of times. Instead, we'll have our Flash program define the listeners on the fly.

Before we get started, the only things initially on the stage are `btnNewGame` (a button) and `tbDoneText` (a dynamic text box that is initially blank). The first thing we need to do is create setup procedures to create an initial game for the user, and erase the current game and create a new game when the user clicks the "New Game" button.

```
function setupGame():void {
        addGameBoard();
        drawBoxes();
}

function newGame(evt:MouseEvent):void {
        removeGameBoard();
        setupGame();
}

// add a listener for the "New Game" button
btnNewGame.addEventListener(MouseEvent.CLICK,newGame);
```

Next we need to write functions to create and erase our game board. The game board will be at most 6 squares by 6 squares, so we need to know how big each square will be.

```
var boxHeight:int = 40;
var boxWidth:int = 40;
var boxXSpace:int = 10;
var boxYSpace:int = 10;

function addGameBoard():void {

	// create a board to display all of the square buttons on
	var gb:Sprite = new Sprite();
	gb.graphics.beginFill(0xFFFFFF);

	// since the entire grid is at most 6x6, make enough room
	gb.graphics.drawRect(0,0,5*boxWidth+6*boxXSpace,5*boxHeight+6*boxYSpace);
	gb.graphics.endFill();
	gb.name = "gameBoard";
	gb.x = 10;
	gb.y = 10;
	this.addChild(gb);
}

function removeGameBoard():void {
	// remove the game board and all of the buttons on it
	var gb:Sprite = this.getChildByName("gameBoard") as Sprite;
	gb.graphics.clear();
	this.removeChild(gb);
}
```

The next section of code is the `drawBoxes` function, which draws the grid of boxes on the game board, adds the dynamically created event listeners, and randomizes the board. Let's look at this one section at a time. Most of the first part is setup. We define the "on" and "off" colors for our squares, randomly determine the number of rows and columns, and create an array to store the true/false states that represent the lights being on or off.

```
var onColor = 0xFF3333;
var offColor = 0x333333;
var numberOfRows:int;
var numberOfColumns:int;
var states:Array;

function drawBoxes():void {

	// create the squares on the game board
	var gb:Sprite = this.getChildByName("gameBoard");
	var square:Sprite;

	// create 3 to 6 rows and 3 to 6 columns
	numberOfRows = Math.floor(Math.random()*4+3);
	numberOfColumns = Math.floor(Math.random()*4+3);

	// create the states array, which has the same number of
	// "rows" and "columns" as our grid of squares
	states = new Array(numberOfRows);
```

Now we want to actually create the squares themselves. We will have two nested loops. At each step, we will draw the square, add it as a child to our game board, set its state to “off,” and create a listener especially for that square. Finally, we randomize the game board.

```
        for (var i:int = 0; i < numberOfRows; i++) { // i = row number
                states[i] = new Array(numberOfColumns);
                for (var j:int = 0; j < numberOfColumns; j++) { // j = column number

                        // create a square, which is by default "off"
                        square = new Sprite();
                        square.graphics.lineStyle(2, 0x000000);
                        square.graphics.beginFill(offColor);
                        square.graphics.drawRect(0,0,boxWidth,boxHeight);
                        square.graphics.endFill();
                        states[i][j] = false;
                        gb.addChild(square);

                        // position the square in the correct location
                        square.x = (j+1)*boxXSpace + j*boxWidth;
                        square.y = (i+1)*boxYSpace + i*boxHeight;
                        square.name = "square" + String(i) + String(j);

                        // add a dynamically created listener for this square
                        square.addEventListener(MouseEvent.CLICK,createListener(i,j));
                }
        }
        randomizeGameBoard();
}
```

There are two big pieces of code that we still need: `createListener`, which should be a function that takes in two arguments (row and column numbers), and returns a *listener function*. The second piece is the randomizer, but that turns out to be fairly simple, and we’ll do that last. It turns out that `createListener` is very simple:

```
function createListener(a:int, b:int):Function {

        // dynamically create a listener for the square in position (a,b)
        // that switches the box at (a,b) and the adjacent boxes, and
        // then checks to see if you are done

        var foo:Function = function (evt:MouseEvent):void {
                switchAt(a,b);
                if (checkDone()) { tbDoneText.text = "Good job!"; }
        }
        return foo;
}
```

Notice that `createListener` returns an object of type `Function`. Be careful to distinguish `function` (which is a keyword used to create functions) and `Function` (which is a data type). We define a local variable foo that is equal to the listener we want. However, this is a listener that has access to the variables a and b. When this listener is called, it will call the switchAt function (which we'll see very soon) using the values that a and b had when the listener was created.

The function switchAt should switch the state of a given square, and also the squares adjacent to it. For this we use a helper function called invertState, which inverts a single square.

```
function invertState(row:int, col:int):void {

        // lookup the square located at (row, col)
        var gb:Sprite = this.getChildByName("gameBoard");
        var thisClip:Sprite = gb.getChildByName(("square" + row) + col);

        // if the square is off, turn it on, and vice versa
        if (states[row][col]) {
                thisClip.graphics.beginFill(offColor);
                thisClip.graphics.drawRect(0,0,boxWidth,boxHeight);
                thisClip.graphics.endFill();
        } else {
                thisClip.graphics.beginFill(onColor);
                thisClip.graphics.drawRect(0,0,boxWidth,boxHeight);
                thisClip.graphics.endFill();
        }

        // reverse the state of the square
        states[row][col] = !states[row][col];
}

function switchAt(row:int, col:int):void {

        // switch the square that was clicked on
        invertState(row,col);

        // switch the squares adjacent to the clicked square
        if (row > 0) { invertState(row-1,col); }
        if (row < numberOfRows - 1) { invertState(row+1,col); }
        if (col > 0) { invertState(row,col-1); }
        if (col < numberOfColumns - 1) { invertState(row,col+1); }
}
```

Notice that the if statements in the switchAt function are required so that we don't go outside of the bounds of our grid of squares.

Finally, we need to check to see if we have won the game (by turning off all of the lights), randomize the game board, and then set up the game when the program initially starts.

```
function checkDone():Boolean {

	// check to see if all the lights are off
	var bool:Boolean = true;

	for (i = 0; i < numberOfRows; i++) {
		for (j = 0; j < numberOfColumns; j++) {
			bool = bool && !states[i][j];
		}
	}
	return(bool);
}

function randomizeGameBoard():void {

	// randomly click the game board a random number of times
	// this ensures the game is solvable
	var r:int = Math.floor(Math.random()*5+5);
	var row:int, col:int, i:int;

	for (i = 0; i < r; i++) {
		row = Math.floor(Math.random() * numberOfRows);
		col = Math.floor(Math.random() * numberOfColumns);
		switchAt(row,col);
	}
}

setupGame();
```