Chapter 5: Buttons and Text I/O

So far we have learned to manipulate MovieClips on the screen, but really don't have a way to get text input from a user. For example, we might ask the player how many gargoyles should be created, or ask them for their name for a high score list. Or, how do you play hangman without a way for the player to enter their guesses. In general, computer programming languages need a way to enter and display text information for names, address, id numbers, and the all important credit card number. The process of supporting entering and displaying text information is call *text input/output*, or text I/O for short. I/O can be to/from a screen, to/from a file on disk, or to/from other devices such as sensors. In flash, after entering text a user often pushes a button to signify that input is complete. The use of buttons in Flash/Actionscript is so common that there are built in button classes. We cover these first, and then cover how to use text boxes for input and output.

5.1: Buttons

Actionscript "buttons" provide another from of user interactivity. Mostly buttons are used for start/stop/modifying animation flow or to enable text input. Buttons are another built in Actionscript class, hence check out the class definition in the help manual actionscript dictionary. We will mostly be using the event handler Button.onPress( ).

Lets create our first button! Follow these steps:

- Open a new flash document, save it as some name
- Go to: Window -> Other Panels -> Common Libraries -> Buttons
- Open the "arcade buttons", and drag one of them onto the stage
- Go ahead and publish (F12), and click. It clicks! But it does not do anything else
- Back on the flash stage, select the button, and then give the instance a name in the Properties panel. Say "btn_first". You can now use "btn_first" as an identifier (i.e. a variable name) for this button inside of Actionscript code.
- Create a new layer for actionscript, and in the first frame add:

```
btn_first.onPress = function() {
        trace("ouch, you pressed me!") ;
}
```

- Run the code by ctrl-enter so you can see the trace, move the output window to the side so you can keep pressing the button and see the output

If you want, you can download a file I created as above:

www.cs.du.edu/~leut/1671/flashFiles/c5_button1.fla

Look at the code above. It is just like our mouseHanlder.onMoveMouse() function. We have put some code inside of a function and made it the *onPress* method of the button named *btn_first.* This is another example of an event handler. So for we have seen the onEnterFrame event handler, the mouse handlers, and now this button handler. They all use the same framework: assign a function to execute the code when the event occurs. If you want you can go and add more lines of code into the and then run the code. For example, change the function so it looks like:

```
btn_first.onPress = function() {
        trace("ouch, you pressed me!") ;
        trace("_xmouse is at " + _xmouse) ;
        trace("btn_first._x == " + btn_first._x) ;
}
```

If you preview this code and click on the mouse it now prints out three statements. Notice that the two locations printed out are at about the same x-coordinate. This makes sense because when you click on the mouse button the *xmouse* mouse location must be within the button area for the click to occur. Try clicking on one side of the button and then the other. You will see that xmouse moves, as it should, but both are still close to the btn_first._x. What can you glean about the button's registration from clicking on the left side of the button and the right side of the button using the code above? Answer: the registration's x-coordinate is in the center.

Buttons have other useful event handlers such as onRelease and onRollOver. Check out the following code:

www.cs.du.edu/~leut/1671/flashFiles/c5_button2.fla

```
btn_red1.onPress = function() {trace("pushed btn_red1") ; } ;
btn_red1.onRelease = function() {trace("released btn_red1") ; };
btn_red1.onRollOver = function() {trace("rolledOver  btn_red1") ; } ;

btn_red2.onPress = function() {trace("pushed btn_red2") ; } ;
btn_red2.onRelease = function() {trace("released btn_red2") ; };
btn_red2.onRollOver = function() {trace("rolled  over btn_red2") ;} ;
```

Here we have two buttons. Load the file and preview it with ctrl-enter to see the trace command outputs. Again, you may need to move the output window to the side to keep seeing the output and the buttons at the same time.

Instead of just printing out messages, you can use the button event handlers to do interesting things. Check out and run:

www.cs.du.edu/~leut/1671/flashFiles/c5_button3.fla

```
btn_left.onPress = function() { mc_penguin._x -= 10 ;} ;
btn_right.onPress = function() { mc_penguin._x += 10 ; } ;
btn_DecSize.onPress = function() {
  mc_penguin._xscale -= 10 ;
  mc_penguin._yscale -= 10 ;
} ;
btn_IncSize.onPress = function() {
  mc_penguin._xscale += 10 ;
  mc_penguin._yscale += 10 ;
} ;
```

Before trying to understand the code, run the animation first. Now lets talk about the layers in the flash file. To get a quick idea of which layer is which, click on the top eyeball to hide all layers, then click on each layers eyeball one at a time to reveal just that layer. The top layer contains the penguin image saved as a MovieClip and names mc_penguin (select the penguin and the check out the properties panel for the name). The next layer contains the text found on the buttons. The next layer contains the four buttons. If you select each one separately and look at the property panel you will see their names. Finally, the bottom layer contains the actionscript code above. There are four functions defined above and the code for each function is enclosed in its own set of curly braces. The four functions are for the onPress event handler for each of the four different buttons. The first two, btn_left and btn_right, just change the *mc_pengiun* MovieClip _x property to move the penguin. The next two have two statements in each function to modify both the _xscale and _yscale properties of the MovieClip.

5.2: Text I/O

In a moment we will see how to use buttons to control text input, but first we need to introduce the Flash text boxes. I/O is done in action script through text boxes created by the text tool (the button with the big letter "A" in the tool panel). With the text tool we can create three types of text boxes:

- Static: just used to put words on the screen, there is no way to modify them from AS.

- Dynamic: used to write text to the screen from action script. The box is associated with an AS variable via the properties panel.

- Input: used to get input from the user. Again the text box is associated with an AS variable via the properties panel.

Using a static textbox it is pretty straightforward, just select it, draw a box on the screen, and type in the text you want. You modify the text properties, like font and size, via the Properties panel.

Moving on to the Dynamic textbox, lets start by creating a program that prints out the frame number every frame:

- Create a new document

- Click on the text tool

- In the properties panel select "dynamic text" (top left of the panel)

- Modify the font (size and name) if you wish

- On the stage click and hold down and drag out a box for the text

- IMPORTANT: now associate an AS variable with the box. In the Var field of the properties panel type in "outbox". This is just the variable name that you can use to assign strings to the box from inside your AS code.

- Create a new layer for the AS code

- In frame one add the following AS code:

```
var frameNum:Number = 0 ;
onEnterFrame = function() {
        frameNum++ ;
        outbox = "frame " + frameNum ;
}
```

- Publish and run the code

Note, if you had problems, this example can be found in:

www.cs.du.edu/~leut/1671/flashFiles/c5_io1.fla

To do input we need to use the input text option for the text box, and again assign a variable to the box via the properties panel. Download and open the following .fla:

www.cs.du.edu/~leut/1671/flashFiles/c5_io2.fla

Inside of flash, select each of the three boxes, one after the other, and look at the properties panel for each. The word "Input" is in a static text box. The box to the right is input text, and the bottom box is dynamic text. Also notice that *the variable field for both the input box and the dynamic box is "inString"*. Now publish/run the program. Click on the input box (next to the word input) and type something in. As you type, each character it is echoed in the box below. This is because every change made in the input box automatically changes the content of the action script variable name "inString". You never even declared this variable in AS, that is okay, it is done automatically. The dynamic text box is also tied to the variable "inString", thus, any change to variable "inString" is automatically sent to the dynamic text box associated with it. Granted, this example is sort of weird, but it shows how the variable associated with input boxes is automatically updated as the user types into the box, and how the dynamic text box associated with a variable is automatically updated as the variable changes.

Now lets do something a bit more normal: ask the user for their name, and output a welcome message. The following is contained in:

www.cs.du.edu/~leut/1671/flashFiles/c5_io3.fla

```
btn_enter.onPress = function() {
        outString = "Welcome " + inString ;
}
```

Look at the .fla file. You will see there are three text boxes (one each of static, input, dynamic) and a button. Two of the text boxes do not have the borders set to on, so they show up in flash as dashed-lines. When the button is pressed the value of variable outString, which is bound to the dynamic text box (you can see this by looking at the properties panel), is assigned the string "welcome" concatenated with the string entered by the user. The idea is the user fills in the input box, pushes the button, and gets the desired welcome message.

5.3: Number Guessing Game

Now that we can do input and output, we can write a simple number guessing game. But first we need to be able to generate random numbers. Computer algorithms exist that create a sequence of pseudo-random numbers that look truly random. There is a lot of theory behind these algorithms that we will not explore. Thankfully we do not need to, instead all we need to do is use the Math.random() function built in ActionScript. The Math class is a top-level class, meaning you do not need to use a construct to invoke its methods. It includes methods such as *ceil(), floor(), min(), max(), log(), random()* and a

bunch more including trigonometric functions. We will only consider the following three methods for now:

1. random ( ) This method generates random decimal point numbers between 0 and 1.0. Each number is equally likely to occur. This is known as an *uniform* distribution.
2. floor ( ) This method takes decimal number and truncates it to the closest integer. As an example: 2.00543 is converted in 2, and 99.99943 is converted into 99.
3. ceil( ) This method also converts a decimal point number to an integer but does so by rounding up, not down. Examples: 2.00543 is converted to 3, and 99.99942 is converted into 100.

We show examples of using these three methods in file:

www.cs.du.edu/~leut/1671/flashFiles/c5_math1.fla

```
var n1:Number = Math.random() ;
trace("n1 = " + n1 ) ;

var n2:Number = n1 * 10 ;
trace("n2 = " + n2) ;

var n3:Number = Math.floor(n2) ;
trace("n3 = " + n3) ;

var n4:Number = Math.ceil(n2) ;
trace("n4 = " + n4) ;

var n5:Number = Math.floor(  Math.random() * 10 ) ;
trace("n5 = " + n5) ;

n5 = Math.floor( Math.random() * 10) ;
trace("next random num = " + n5) ;
n5 = Math.floor( Math.random() * 10) ;
trace("next random num = " + n5) ;
n5 = Math.floor( Math.random() * 10) ;
trace("next random num = " + n5) ;
n5 = Math.floor( Math.random() * 10) ;
trace("next random num = " + n5) ;
```

Each time this program is run you get different output do to the calls to Math.random(). The first trace statement prints out a random number between 0.0 and 1.0. The second takes this number and multiplies it by 10 to print out a random number between 0.0 and 10.0. If we want random integers we can create these by taking the floor or ceiling of these numbers. If taking the floor the random numbers will be 0 .. 9, if taking the ceiling

the random numbers will be 1 .. 10. Go ahead and play with this file to create other random numbers.

Lets say you want to generate random integers between 0 and 4 inclusive. You could just call Math.random(), multiply by 5, and then take the floor:

```
var num1:Number = Math.floor (  Math.random() * 4 )  ;
```

What if you want numbers between 10 and 14 inclusive? Do the same as above but add 10:

```
var num1:Number = Math.floor (  Math.random() * 4 ) + 10  ;
```

Assume you want random integers between A and B inclusive. Can you come up with a statement to do so using A and B as variables?

Okay, lets take our random number generation knowledge to create a number guessing game:

www.cs.du.edu/~leut/1671/flashFiles/c5_RandomGuess1.fla

```
var theNum:Number = Math.floor( Math.random() * 3 ) ;
theNum += 1 ;    // so it is 1, 2, or 3; not 0, 1, or 2

btn_enter.onPress = function() {
        if (inString == theNum)
           outString = "You got it!  Exit and play again" ;
        else
                outString = "Nope, guess again" ;
}
```

The game is pretty simple. The program generates a random number between 1 and 3. Static text asks the player to guess a number between 1 and 3 inclusive. When the player presses the button, the btn_enter.onPress() function is called and the code in that function compares the contents of variable inString with the generated random number. If the two are equal the code puts “You got it!” into variable “outString”, which is the variable name of the dynamic text box. Other wise the code puts “Nope, guess again” into outString and presumably the player keeps guessing.

Exercises:

1. Write code to generate uniformly distributed random integers whose values are between 20 and 39 inclusive.
2. Same but for values between 21 and 40 inclusive.
3. Save as above except for values between -49 and 0 inclusive.
4. Modify the code in www.cs.du.edu/~leut/1671/flashFiles/c5_RandomGuess1.fla so that if the player guesses correctly, the code generates a congratulatory message, but then generates a new random number and challenges the user to guess again.

5.4: Timeline Manipulation and Game Phases

We can change the normal sequential timeline execution with a few action script commands: gotoAndPlay( ), gotoAndStop ( ), and stop ( ). For games this is necessary to enable the flow from start screen, to game screen, to end screen.

First, load up the following .fla file and preview using ctrl-Enter so you can see the trace() command outputs.

www.cs.du.edu/~leut/1671/flashFiles/c5_goto1.fla

I set the frame rate to 1 fps so it just plays frame 1, 2, 3, 4. Then the animation ends, and then restarts, so it just keeps counting 1 to 4 over and over. By using ActionScript you can change the order that frames are visited.

Now lets modify the frame play order by using the gotoAndPlay() command. Load of the following .fla file and again preview by using ctrl-enter.

www.cs.du.edu/~leut/1671/flashFiles/c5_goto2.fla

The file has four frames, they contain the following ActionScript code.

Frame 1:

```
trace("Inside frame 1") ;
gotoAndPlay(3) ;
```

Frame 2:

```
trace("inside frame 2") ;
gotoAndPlay(4) ;
```

Frame 3:

```
trace("inside frame 3") ;
gotoAndPlay(2) ;
```

Frame 4:

```
trace("inside frame 4") ;
stop() ;
```

When the code is run you see that the frames are visited 1, 3, 2, 4, and then execution stops. The command gotoAndPlay( ) causes the execution of flash to jump to the frame specified as an argument. The argument can be a frame number as used here, or a frame label in quotes. Frame labels are created in the property panel when a frame is selected. As a result, when frame 1 is played, it prints the trace command but then jumps, immediately, not waiting for the frame duration to expire, to frame 3. The code in frame 3 is immediately played and then execution jumps to frame 2, whereupon the trace command is executed and execution jumps to frame 4. All of this happens without waiting for the frame duration (determined by the frame rate) to expire. Then, we hit the end of the animation, wait for the frame duration to expire, and then the whole animation starts over again at frame 1. So, every second you see four lines followed by a blank line printed out showing the execution of frame 1, 3, 2, 4.

Consider what happens if you uncomment the stop( ) command on the last line of frame 4. This is what is done in:

www.cs.du.edu/~leut/1671/flashFiles/c5_goto3.fla

The code is identical to a_goto_2.fla except frame 4 has a stop( ) command. The stop( ) command stops the execution of the MovieClip. Note, the main timeline is simply the _root MovieClip, thus the main animation is stopped. Try moving the stop to frame 3. Yep, it plays frame 1, 3, and then stops.

Now, lets add to frame 1 an onEnterFrame event handler:

www.cs.du.edu/~leut/1671/flashFiles/c5_goto4.fla

```
onEnterFrame = function() {
        trace(" ") ;
        trace("inside _root.onEnterFrame()") ;
}
```

Run the code with ctrl-enter to see the trace command output. After the trace output from frames 1, 3, 2, and 4, the code prints out "inside _root.onEnterFrame()". The visiting of frame 1, 3, 2, 4, only occurs once, as we expect with the stop() command, but the onEnterFrame event handler keeps getting executed every second! This is because the stop( ) command stops the execution of the main MovieClip( ), but, it does not cancels any event handlers. This is important since often you may put game code in an event handler, get to the end of the game, or so you think, and the event handling code still goes on! This can be fixed by deleting (i.e. canceling) the event handler when you stop the animation. Check out:

www.cs.du.edu/~leut/1671/flashFiles/c5_goto5.fla

In the 4th frame our code now says:

```
trace("inside frame 4") ;
delete _root.onEnterFrame ;
stop() ;
```

If you load and run it, you see the onEnterFrame handler is never called, because it is deleted.

It may seem like we just took a long digression from games, but it was an important one. Most games have different types of screens. For example a start screen, a game screen, and then and end screen. The start screen might explain the rules, or ask the user for input like the difficulty level they wish to play. Perhaps there are additional screens for different levels. The way to do this in flash is with gotoAndPlay( ). Also, when a game is over, you need to be able to stop the game. Lets put it all together with a modification to our number guessing game.

Lets put it all together in one game. Download and look at each frame of:

www.cs.du.edu/~leut/1671/flashFiles/c5_RandomGuess2.fla

This file is a complete game with three phases. In phase one the player is asked to enter lower and upper bounds for the range of numbers. Then, when the press the button they are moved into phase 2 (frame 2). In phase two the player guesses numbers. If incorrect they are prompted for another guess. If correct, then the game moves to phase 3 where

the player is congratulated for their clever success and asked if they want to play again. If they want to play again control moves back to phase 1. A few things to note about this game:

- Transition between frames is done by the gotoAndPlay( ) call
- Each phase (frame) has a different background color to help you detect phase changes.
- Dynamic and input text boxes are reset to empty strings when a frame is entered to clear entries from the previous game

Exercises:

1. Modify one of the bouncing gargoyle animations to consist of two phases. In the first phase the user is asked to set the gargoyle velocities. Then, when they click done, the second phase is entered and the gargoyles bounce with those velocities.