

MM 319 Lab Five: Message Passing

Introduction

This activity will *help* you understand message passing and the message passing hierarchy. What you see here should reinforce lecture and readings, but this activity will not provide the deeper understanding you need.

Messages

One way to think of messages is as function calls, although this is not a perfect analogy. A special type of message is used (potentially) to allow screen interaction to activate to activate some handler (function) within a script. These messages are generated by Director's runtime engine in response to events. **mouseUp** is an event you already know. Other events of interest are **mouseEnter**, **mouseLeave**, **mouseWithin**, **mouseDown**, **mouseDoubleClick**, and **mouseRightClick**. As you might guess, these events are all associated with mouse activity. Events not related to the mouse are **enterFrame** and **exitFrame**, which take place when the playback head in the score enters or leaves each frame; **startMovie** and **prepareMovie**, which are *almost* the same, and close enough for current purposes; and **idle**, which is an event that takes place whenever nothing else is going on.

Messages are sent to specific objects, and if not handled by their initial recipients, are passed on through a hierarchy. If no script in the hierarchy handles a message, it is ignored. This activity will illustrate some aspects of the message passing hierarchy. Here is a brief summary:

Mouse events are sent to first to the sprite that is affected. If there are several sprites under the mouse cursor, then the message is sent to the sprite that is "nearest" the user—the one in the highest numbered channel. If that sprite's script has no handler for that message, it goes next to the member for the sprite. If that script doesn't handle the message, it goes to the next sprite under the cursor (if there is one), otherwise the message goes to the current frame. If the frame doesn't handle the message, then it goes to the movie.

Frame events can be intercepted by the frame script or other scripts. These events seem to propagate even when an object has a handler for them. However, the frame and movie scripts are the places where you are most likely to handle these events.

Movie events are handled by the movie script. Sprites and members don't receive these messages.

The Experiment

Create a new Director file. The stage doesn't need to be too big—you'll want room to see the message window and the stage simultaneously.

Insert a text member named "TheText," a bitmap named "ThePicture" and a button named (surprise) "TheButton." Make sure the cast members are all named. Arrange the three objects on the screen so there is some overlap, as shown in Figure One, below.

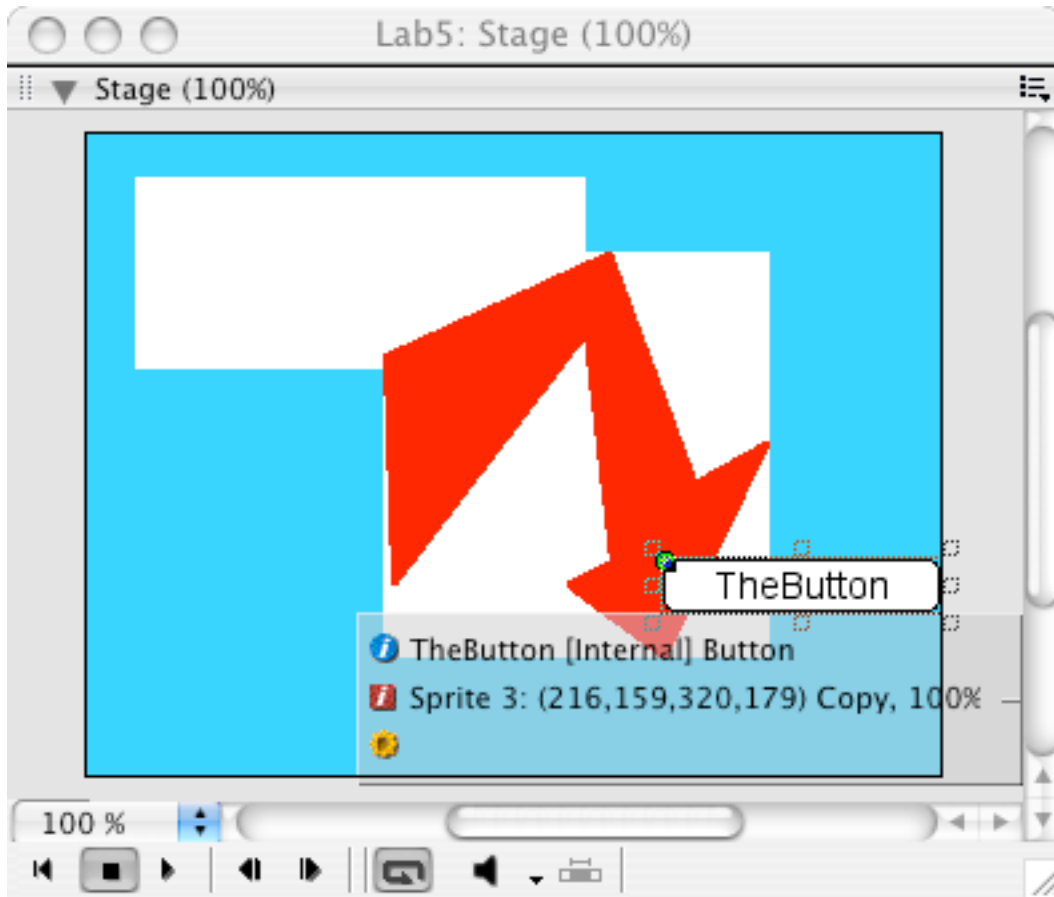


Figure One: Three Sprites on Stage

Now edit the sprite script for the text member to include these lines of code:

```
on mouseUp me
  put "Mouse up received by the text"
end

on mouseEnter me
  member("TheText").text = "I see you"
end mouseEnter

on mouseLeave me
  member("TheText").text = ""
end

on prepareMovie
  member("TheText").text = "Starting Value"
end
```

What this script tells the textbox on the screen to do is this: if the user clicks the mouse over this field, put a statement in the message window reporting this; if the mouse cursor crosses into the text box, display some text in the text box; when the cursor leaves the boundary of the text box, remove the text; and finally, when this sprite receives the prepareMovie message, put an initial text value into the text box.

Save the movie, then arrange the stage and the message window so you can see both, and play the movie. Move the mouse around the stage, try clicking in various places, and observe what happens.

The first thing you should notice is that the prepareMovie handler is not executed. This is because the sprite is not in the path for getting this message.

Another observation is that the mouse messages seem to reach the text box even where the other objects block the text window.

Now add a script for the picture sprite that looks like this:

```
on mouseUp me
  put "MouseUp received by the picture"
end
```

```
on mouseEnter me
  put "Mouse Enter received by the picture"
end
```

```
on mouseLeave me
  put "MouseLeave received by the picture"
end
```

When you save and run again, you will be able to detect when these events are received by the text sprite and when they are received by the picture. Notice that now, mouse activity in the region where the two objects overlap is handled only by the picture (assuming you inserted the picture after the text, placing it in a higher-numbered channel).

Edit the button cast member script as follows:

```
on mouseUp
  put "Button Cast Member received mouseUp"
end
```

Test the results of this change, and make a note of what this means in terms of the order of objects on the message path. Now give the button's sprite a similar script that reports that the "Button sprite" received the message, and test this. Notice that the cast member no longer receives the message.

Now add a movie script that looks like:

```
on mouseUp
  put "The movie received mouseUp"
end
```

When you run this test, try clicking over the three objects, and also in the area unoccupied by any of them. As you will see, the only time the message reaches the movie object is when nothing else has been able to intercept it. (I know this is probably getting old, but it's almost over!)

Now let's add a frame script. In the last frame in the score that is occupied, add this script:

```
on exitFrame me
  go to the frame
end
```

```
on mouseUp
  put "The frame received the message"
end
```

This tells the playback head to stay in this frame, and also tells the frame to report when it receives the mouseUp message. When you run and test this version, you'll notice that the frame prevents the message from reaching the movie script.

Conclusion

Obviously, all of these scripts could contain instructions to accomplish more than displaying feedback in the message window. The point of this activity is to begin developing an understanding of how messages are passed around, and what sorts of events we can use to trigger script.

You should begin thinking about how you'll determine where to place script to do various things. Clearly a task entirely associated with the user clicking a button can be placed in the button script. More generic jobs might go in the movie script or a frame script. If some "sub task" like confirming that a value is numeric is needed in numerous places, it can be handled in the movie script, and the other object scripts can call it.