Macromedia Director
Tutorial 4

Further Lingo

Contents

Introduction ..............................................................................................................................1
Play and Play done..................................................................................................................2
Controlling QuickTime Movies.................................................................................................3
Controlling Playback................................................................................................................3
Introducing the if, then, else script...........................................................................................6
PuppetSprites and repeat while loops.....................................................................................7
Dragging a sprite .....................................................................................................................8
Repeat while commands .........................................................................................................8
Coordinates ...........................................................................................................................12
Variables ...............................................................................................................................15
me................................................................................................................................17
Playing a movie from within a movie .....................................................................................17

Introduction

In this tutorial you will cover more Lingo commands and techniques. You will cover if, then else and repeat while constructs, variables, PuppetSprites and manipulation of QuickTime and Director movies.
Play and Play done

Play fulfils a similar task to go to.

Open the file you saved from last week’s tutorial, the button/destination movie.
Swap the go to frame “yellow” command with play “yellow”, rewind and play.

It makes no difference, it only differs when used in conjunction with the play done command.
Change the script in the last frame of yellow from go to the frame to play done.
Play the movie again.

Now when the playback head reaches the exitFrame it will see the play done command and
jump back to the frame it left when it carried out the play command.

Check what happens when you use the other buttons.
They all have the play done command as well, why?

Yellow goes back to the frame it left (frame 5), the others go back to frame 1.
This is because there is no play command on the other colour's sprites. If Director sees a
play done command without having seen a play command it always jumps the movie back to
frame 1.

In answer to the previous question, why did all the destinations change when you changed
yellow's, this is because you changed a behaviour that was assigned to all the scripted
exitFrames. If you just wanted to change yellow you would have had to delete the old
scripted frame and write a new script. Return the script to a go to the frame command.

It’s better to use the go to command unless there is a specific need for a play/play done
because it is less prone to bugs.
Controlling QuickTime Movies

Import a QuickTime movie into your button/destination movie and place it on the Score at frame 40 so there are 5 frames of this sprite (You can find a QuickTime movie file on the SMS web site in Study materials Week 6, or download your own from the Internet if you wish – try to make sure it's not too big i.e. <1 Mb). To avoid confusion from this point the tutorial will refer to a QuickTime movie as the video and the Director movie as the movie.

Click on frame 40 on the timeline to bring the playback head to this point and play the movie.

The video reaches frame 45 and stops (if Loop Playback is selected in the Control menu the movie will return to the home page). You could keep the video playing to the end by expanding the number of frames on the Score. This method is inefficient and not very useful when constructing an interactive movie.

To keep video or sound playing until it has finished you need to put some Lingo in the last frame on the Score to make the sprite either loop or, more efficiently, go to the frame.

On frame 45 add the script

```lingo
on exitFrame
    go loop
end
```

Add a marker to frame 40. You do not need to refer to the marker in the script because a go loop command will always take the playback head to the first marker it finds on its return journey. If it doesn't find a marker it will loop from frame 1.

You will see that although the playback head is looping around the sprite the video is playing continuously until it reaches the end and then it stops. This loop instruction tells Director to stay looping in this location. Director tells QuickTime to play the movie. To get Director to tell QuickTime to loop the movie you would still need to check the loop box in the Properties Inspector under the QuickTime tab.

For the best way of scripting video change the go loop command to a go to the frame command and ditch the marker, try this out.

Controlling playback

The object of this exercise is to demonstrate how Lingo can control QuickTime movies. For the purposes of this demonstration select Loop in the Property Inspector and OK the window. Click on frame 40 and let it play.

Close all windows and open the Message window.
Type in

```lingo
set the movieRate of sprite 1 to 0
```

Macromedia Director Tutorial 4
press return and hey presto...the video stops.

**MovieRate** is the running speed of a QuickTime movie, a running speed of zero stops the video.

Delete the 0, type 1 and press return, the video starts again. A movieRate of 1 is the normal playback rate of the video.

Change the movieRate to make the video run at double speed.

The video can be made to run backwards by using minus numbers.

So you can use Lingo to play, stop, fast forward and rewind a video.

Before continuing be sure to return the QTmovie to a movieRate of 1. Reopen the Properties Inspector and unloop the video. Use the **exchange cast member** tool to replace the yellow destination with the video clip.

---

**A few points about playing video:**
The ideal size for video is quarter screen - 320x240 pixels.
When stretching a video cast it should never be more than doubled, if you want full screen video you should be working with a **640x480 Stage size**. If you are not don't worry these are only demonstrations. The optimum running time for video on a **CD-ROM** is **15 fps**. A video will run best if it is **Direct to Stage**. **Direct to Stage** means that QuickTime places the video directly on the Stage bypassing Director’s own system for processing images. This means that you can't put sprites on top of a video.

On a PC this is the only way to play a QuickTime movie.

Ok, given all of the above, if you incorporate a full screen video Direct to Stage into your button/destination movie the Return button will be invisible.

You need another way of returning to the home page, a way that returns from the video as soon as it has finished.

This requires a bit of thought. You know that a **play/play done** command will carry out this task, you also know that you need a **go to the frame** command to keep the movie playing. Both the **play done** and the **go to the frame** commands need to go in the same place, on the exitFrame of the last frame of video. How do you stop them conflicting?

You need to set up a script that says;

(if the movie is playing) **go to the frame**
(if the movie has stopped playing) **play done**
Introducing the if, then, else script

Lingo has the ability to ask Director to make decisions. It uses the following pattern:

```
on (an event)
if (something happens) then (do this)
else (do that)
end if
```

Using an if, then, else script

Now you will use the if... then... else... script to make the video play until it has finished and then return to the home page. The script starts to shape up as below;

```
on exitFrame
if (the video is playing) then
    go to the frame (keep playing until the end)
else (the video has finished)
    play done (return to the home page)
end if
end
```

What is the Lingo phrase for **the video is playing**?

The finished script is;

```
on exitFrame
    if the movieRate of sprite 1 = 1 then
        go to the frame
    else
        play done
    end if
end
```

This script needs to go in the last frame of video. Remember you need to clear the existing script and write a new one, editing the existing script will effect the behavior of all the other exitFrames.
Contract the Return button so it starts under red rather than yellow. Edit the script so the yellow button reads play "yellow" rather than go to "yellow" so the playback head returns to the last frame of the home page. Try this out.

**PuppetSprites and repeat while loops**

It is possible to make a sprite respond to the user, for example letting the user drag it about. When you drag a file into the wastebasket or move a window around on your monitor you, the user, are controlling the position of the file/window. If you are playing a game of space invaders and moving a spaceship across the screen with your mouse the same principle applies. The spaceship is the sprite, you are the user and the sprite has been programmed to follow the position of the mouse.

**Lighting a firework**

The object of this exercise is to make a movie where the user has to drag a match across the Stage to a firework, when the match touches the firework it goes off.

**Step 1.** Draw the sky, match, unlit firework, and the lit firework, write the instructions and create the animation for the firework exploding. (see Cast below for hints)

**Step 2.** Lay out the movie in the Score so that frames 1-23 of sprite 1 is the sky, frame 1 of sprite 2 is the unlit firework, frame 1 of sprite 3 is the match and frame 1 of sprite 4 is the instructions. The animation starts at frame 3 which is labelled bang. (See score below)
Step 3. Add the programming.

On frame 1 write a go to the frame command. The rest of the script involves allowing the user to move the match and making the animation play when the match touches the firework. This script needs to be looked at in stages.

The first stage is to make the match follow the cursor
For this you need the match to come under the user's control.

Dragging a sprite

The movie requires that the match follows the position of the cursor when the user clicks down on the match and drags it.

So far you have used the default mouseUp event for interacting with sprites. This time the event we want is mouseDown.

Remember the loc of sprite argument from the lingo tutorial, you asked the Message window to put the loc of sprite and it gave you the sprite's co-ordinates.

You know that Lingo can interact with the location of a sprite's co-ordinates, it can also interact with a mouse's co-ordinates.

The Lingo phrase to make the sprite follow the mouse is as follows;

on mouseDown
  set the locV of sprite 3 to the mouseV
  set the locH of sprite 3 to the mouseH
end

where locV and locH are the vertical and horizontal co-ordinates of the sprite and mouseV and mouseH are the vertical and horizontal co-ordinates of the mouse cursor. Play the movie and try this.

Problem - each time you click down the sprite jumps so that its registration point meets the position of the cursor this is because you have only told it to move to the cursor's location once, it is not being dragged.

You need to tell Director to keep moving the sprite for all the time the mouse is held down, so you need Director to keep asking itself - "is the mouse down? If so I'll keep carrying out the command until the mouse is no longer down".

Repeat while commands

A repeat while command will make Director keep asking a question about a condition and depending on that condition it will repeat an action until the condition changes.

In this case the condition is the mouse being down and the action is the movement of the sprite to the position of the cursor. This script is laid out below.

on mouseDown
  repeat while the mouseDown = true
set the locV of sprite 3 to the mouseV
set the locH of sprite 3 to the mouseH

end repeat

end

The condition is either true or false, in this case you want it to be true that the mouse is down in order to carry out the command. A **repeat while** loop will not work without an **end repeat** command, Director needs to see an end in sight before it embarks on a long journey. Try this extended script.

Problem - you are only seeing the end result, the match is jumping to the resting place of the cursor when you let go. You want to see the sprite at all the points in-between. You need to tell Director that you want a result at each point in time otherwise it will try and shortcut by only giving you a result after the event. This is done by adding the command **updateStage**. The new script is now;

```
on mouseDown
  repeat while the mouseDown = true
    set the locV of sprite 3 to the mouseV
    set the locH of sprite 3 to the mouseH
    updateStage
  end repeat
end
```

Try this, it works!

**The next stage is to make the jump to the firework animation when the match intersects with the firework**

When you drag a file from a floppy disk to the hard disk of your computer the file's icon intersects with the hard disk's icon or an open window and the computer's operating system tells it that on this event a file transfer must happen. This is an intersect **event handler** and a **file transfer command**.

Transferring this knowledge to the firework movie, the items that need to intersect are the match (sprite 3) and the firework (sprite 2) and the resulting command would be **go to "bang"**.

The intersect event needs to live within an **if...then...**

If (this sprite intersects with that sprite) then
(do something)
end if

So, if this event occurs Director carries out the associated command and if the event doesn't occur Director carries on with whatever else it was doing. To add this to your mouseDown script so that it reads:
Try this.

You now have a fully functional movie although there are a couple of improvements you could make.

As it stands you could put the unlit end of the match to the base of the firework and still get a result. Wouldn't it be better to only be able to use the flame being put to the wick? This is possible, it needs to be divided into two separate tasks; make the match intersect with the wick only, and then make the flame only intersect with the wick.

Part one is easy, there are two solutions, the first would have been to build the firework from two cast members; the main body and the wick and then intersect the match sprite with the wick sprite, this is really the best solution, only for the purposes of this tutorial I will show you another solution, simply because it's a very handy tip.

Open the Tool Palette from the Window menu, select the rectangle shape and then from the lines at the bottom select no line. This will enable you to draw an invisible box around the wick.

Then reopen the mouseDown script and change the intersect line from;
if sprite 3 intersects 2 then
to;

end repeat
if sprite 3 intersects 2
then
  go to "bang"
end if
end
if sprite 3 intersects (the new sprite no.) then
the new sprite being the sprite number of the invisible box, probably 5.

Now the match will only light the firework if a part of it makes contact with the wick. The invisible box trick is a handy one to know because it can be used as a quick solution to many problems.

The lighting with the wrong end of the match problem is a more complex one. The following is one possible solution.

Problem: The whole match sprite responds to the intersect command, the invisible box or separate cast trick won’t work on its own because the flame, somehow, has to be dragged around with the matchstick, however this will be the basis of the solution.

This time make the flame a separate cast member, in Paint highlight the flame and Cut it, click the + icon for a new cast member and Copy the flame here.
Drag the new flame cast onto the Stage to rebuild the match.

**Coordinates**

The next stage is to alter the script so that when the match gets moved around the flame follows in perfect harmony. You already know how to do this with a sprite and the mouse, the only difference is that one sprite needs to follow another. The flame sprite needs to be scripted so that it follows the match sprite which follows the mouse. Add the script for this as shown below.

```director
on mouseDown
  repeat while the mouseDown = true
    set the locV of sprite 3 to the mouseV
    set the locH of sprite 3 to the mouseH
    set the locV of sprite 6 to the locV of sprite 3
    set the locH of sprite 6 to the locH of sprite 3
    updateStage
  end repeat
if sprite 6 intersects 5 then
  go to "bang"
end repeat
```

The intersect command also needs to be changed so that it is the flame which intersects with the wick and not the match.
end if

end

Try this.

It's not quite what you want but it is on the right track.

When you set the vertical and horizontal location of a sprite you are referring to the vertical and horizontal co-ordinates of the registration point of a cast member. The match's registration point is on the left and the flame is to the right so on the mouseDown event you are getting your fingers burnt.

There needs to be a way of offsetting the location so the flame stays in the right place in relation to the match.

Cast your mind back to the Lingo tutorial, there was a Message window which did useful things like give you sprite locations and you also found out that Lingo can do maths.

It must, therefore, be possible to find out the locV and locH of both sprites and subtract one from the other. The difference would then be added to your script.

Say, for example, the loc of sprite 1 was (5, 12) and the loc of sprite 2 was (10, 8).

You want the two sprites to move in unison but maintain exactly the same distance between their registration points.

To find the distance subtract the two horizontal co-ordinates, 5 from 10, giving a distance of 5, and the two vertical co-ordinates, 12 from 8, giving a distance of -4.

The script would then read;

set the locV of sprite 2 to the locV of sprite 1 -4
set the locH of sprite 2 to the locH of sprite 1 +5

Then if sprite 1 was dragged around sprite 2 would follow maintaining its distance.

Open the Message window and type in put the loc of sprite 3 and press return.

Ask for the location of the flame sprite. Then ask the Message window to subtract the set of co-ordinates of the sprite you want to move from the sprite you want to follow, for example;

Reopen the mouseDown script and make the amendments. The first figure is the horizontal co-ordinate, the second is the vertical.
Problems cured.

For the sound two firework samples are available on the SMS web site. Take your pick and adjust your movie to make the sample and animation synchronise. Place the sample in the sound channel as shown below.

You could also use a rollover behaviour to turn the cursor into a closed hand when it rolls over the match.

The finished Score:

There is a very important aspect to Lingo that needs to be raised at this point.
Variables

What is a variable? Technically a variable is an area of the computer's Random Access Memory (RAM) which contains a value and is assigned a name.

Using Lingo you create a variable by coming up with a name and entering a value, which all sounds horribly vague. To try and unvague the vagueness:

A variable is a stored piece of information that can be changed when the need arises. There are two types of variable that Lingo can deal with, **global variables** and **local variables**.

A **global variable** remains intact from the moment it is created until the program is quit. A **local variable** exists only for the length of the event handler it is attached to.

This task will include the use of a simple local variable but first an example to try and shed light on murky waters.

You could create a variable by asking for an area called Fred and putting the number 3 in that area. In response the computer will call an area of its RAM Fred and insert the value 3. Then every time you ask for Fred the computer gives you the number 3.

If this were a global variable you could ask for Fred and get the number 3 from the moment the variable was created until you quit the program.

In this case you want a local variable so you can only ask for Fred and get the number 3 for the duration of a particular event handler.

To create this local variable you would need an event handler and a bit of Lingo;

```
on mouseDown
    set Fred to 3
```

would be a start.

When the mouse is held down Director will read the `set` command and will see that Lingo is asking for something to be set to 3. Director does not recognise the word Fred so it knows that it must ask the computer to make something called Fred with a value of 3, and the computer knows a request for a variable when it sees one.

You can call a variable anything you like as long as it is not already in Lingo’s vocabulary.

If Lingo recognises the word it will try and carry it out, meaning, for example, if you wanted to call your variable locV Lingo would want to set the vertical location of something to 3 and would ask me what that something was.

So you have a variable called Fred with the value 3, so what?

Remember numbers are to be avoided, you changed the `go to frame` number commands to `go to frame` name commands to make the Lingo easier to follow and to change and, therefore, less prone to bugs. You have probably also named your cast members.

So to make it easy on yourself when you read your complex script you want to be able to look at words and picture in your mind what you were referring to.

Look at your finished script.
The numbers 3 and 6 are used repeatedly.

Say, for example, you added clouds and an audience and other fireworks going off in the distance and maybe a bonfire too. This would all take place behind the main action and so you would need to shift the matchstick, flame and firework down the Score changing their sprite numbers.

Then you would have to go back and change all the sprite numbers, seven changes in this tiny little movie. What if you missed one? What if you couldn't remember which was which? Both these problems are easily overcome in this tiny movie but you need to know how to deal with big complex, realistic movies.

If you used variables in place of numbers you would solve both these problems. A variable called `match` could replace the number 3 and a variable called `flame` could replace the number 6.

After `on mouseDown` write in `set match to 3`. This creates a variable called `match` with the value 3. Then add the second variable.

Then swap every 3 with the word `match` and every 6 with the word `flame`. The script should be as follows:

```director
on mouseDown
  set match to 3
  set flame to 6
  repeat while the mouseDown = true
    set the locV of sprite match to the mouseV
    set the locH of sprite match to the mouseH
    set the locV of sprite flame to the locV of sprite match - 17
    set the locH of sprite flame to the locH of sprite match + 72
    updateStage
  end repeat
  if sprite 6 intersects 5 then
    go to "bang"
  end if
end
```

The numbers 3 and 6 are used repeatedly.
end repeat

if sprite match intersects 5 then
go to "bang"
end if

end

After the variables match and flame have been set every time you write match Director sees 3 and every time you write flame Director sees 6 and carries out the instructions accordingly.

Now when you go back to your script a year later to update it you can easily see that the match and the flame have been puppetted and are programmed to stick together and follow the mouse.

If you add the changes mentioned before, the clouds, audience etc and need to change the programme to reflect this;
say the match was now sprite 8 and the flame sprite 9, before you had variables you would have had to make seven changes to your script, now you only need to make 2.

All you need to do is change the values of your variables from 3 and 6 to 8 and 9 thus eliminating five chances of making a programming error.

In this example the variable's value has been a number, this doesn't have to be the case. The value can also be what programmers call a string. A string is a letter(s), word(s) or sentence(s).

The movie is now complete. Publish it so that it fills the browser window and adjusts the movie size to the browser window size.

me

So what does the term me mean then, that appears by default in the script window after the event handler?

me is a variable. If you look at the script window, pictured above (over page), the first line of script says on mouseDown me. The me variable stores the memory address of the sprite the mouseDown handler is controlling, in this instance sprite 3, the match. This is just a convention that helps Director know which sprite is being referred to, you can delete the me and the script will work just as well.

Playing a movie from within a movie

It is possible to program Director to open and play a Director movie from within another movie. This feature can be used to create one large continuous movie from a series of smaller movies, a good technique for overcoming any file size problems. Alternatively this feature can be used to enhance interactivity expanding the resource base from which the end user can extract information.

There are a couple of ways to play a movie from within a movie. The first is to open a new window so the new movie is playing while the old one is still open in the background, the second is to jump from one movie to another. Either way this works best if all the movies you are using are saved in the same folder, make sure this is the case before starting the following exercises.
Using Lingo to play a second movie in a new window

The object of this exercise is to use a single line of lingo to open your running figure movie from within your buttons/destinations movie. (Or any other movie you have made, e.g. the tennis ball/racket movie).

Highlight your green button and open the Property Inspector by clicking on the Behavior icon.

Highlight the on mouseUp go to "green" instruction (mine is cast member 11) open the Script Window.

Delete go to "green" and type in open window "running figure" (or whatever you called your movie).

Rewind and play the movie, click on the green button and your new movie will open in a new window.

The first time you do this a window may appear asking where the new movie file is. Select the appropriate file and click open, this will not happen on subsequent plays. The original movie will remain open.
To close the movie just close the window.

If you are designing a movie specifically to be played from within another movie it is worth making its Stage smaller than the original movie's Stage so the original movie is still visible in the background.

**Using Lingo commands to play a movie within a movie**

You can jump from one movie to another using the simple Lingo commands that you are already familiar with, `go to` and `play/play done`.

**Go to movie**

This time open the blue button's Script Window and change the script from `go to "blue"` to `go to movie "firework8"` (or whatever you called this movie).

```lingo
on mouseUp me
  go to movie "firework8"
end
```

The only difference between this script and the `go to` script you have used before is the addition of the word `movie`. Now, instead of jumping to another point within the movie you are playing, this Lingo command will jump to the new movie. This time what happens is the movie you are playing is closed and the new movie opened. You will need to save the changes before trying this out. Then play the movie and click on the blue button.

The firework movie will open and play as normal. To complete the loop add a `go to movie` command to the last frame of the firework movie to take the user back to the original movie. (See below)

**Play movie**

In this case simply swap the `go to movie` command with a `play movie` command. Save the changes.
Open the firework movie and replace the `go to the movie` command with a `play done` command. Save the changes.

Close the movie and reopen the original movie. Now, when you play the movie and click on the blue destination, the movie closes and the firework movie opens, when the firework has exploded and the playback head hits the `exitFrame` it will see the `play done` command, identify that the `play` command came from the previous movie and it will hop back to the point it left in the previous movie.

**Homework**

Finish all the tutorials to date (tutorials 1-4).