Making Your World Leverly -- plot tokens and other magic

The goal of this tutorial is to build on the last tutorial (Making Your World Interesting) by making the world you've built even better. This tutorial assumes that you've finished the previous three Make Your World tutorials.

In this tutorial, we will get the lever animation down, create this "right order" we've been babbling about, and make some treasure!

The player is a wandering adventurer who happens upon a tower in the wilderness. Inside the tower, the player finds a friendly Guardian who tells the player about the magic levers. If they are pulled in the right order, the vault opens and the treasure may be obtained.

Getting Started:

1) Open ScriptEase.

2) Open LeverMagic3 and save it as “LeverMagic4”.

Animating Placeables:

Levers don't play nice:

3) You may have noticed that we've been avoiding the levers. Only one even moves and it doesn't affect the world at all. You may have surmised that this is because levers don't play nice. You may have surmised correctly. Let's get started, and you'll see why levers are tricky as we go.

4) Expand the “When Lever#1 is used” Encounter.

5) First, we'll delete that Action that is trying to animate the lever by right-clicking on the “Then, Animate...” Action, and choosing “Delete” from the menu.

6) Next, by right-clicking, we'll Add a Definition → Get Variables → Define Local Integer as Target’s integer labelled Label.

Intuitively, because the lever has no sense of whether it is in an “on” (the game thinks of it as “activated”) or “off” (“deactivated”) state, we must keep track of this ourselves. We do this with an integer which will change (to reflect the change in the lever's state) each time the lever is pulled. For example, when the integer is “1”, that means the lever is in the “activated” or “on” state. When the player uses the lever in the game, the integer gets changed to “0”, and the lever changes state accordingly. If this doesn't make sense, don't worry. It will become clearer as we go along.

7) “Define Local Integer” is a definition block, represented by a blue D. You can expand
the definition block to view the list of definitions. Now there is only one definition. In
the Definition you've just created → Description tab → Label → change it to
“LeverToggle”.

8) Under the “Target” tab → dropdown menu → The Placeable (Lever#1). As you've
probably guessed, we can also choose “Lever #1” from the Module Blueprint → Pick...
option. There is a reason for choosing “The Placeable (Lever#1)” over a module
blueprint, but we'll go into that when we start animating the other levers.

9) Under the “Label” tab → Constant → put “toggle” into the textbox. The Label is just
a name for the integer, for the game to remember.

10) Again, by right-clicking on the encounter, Add a Definition → Binary → Comparing
Objects or Values → Define Same Integer as whether First Integer in the same as Second
Integer.

11) Under the “Description” tab → Label → change it to “Activated”.

12) Under the “First Integer” tab → Select Integer → choose “LeverToggle” from the
dropdown.

13) Under the “Second Integer” tab → Constant → put “1” in the text box. At this point,
your module should look like the screenshot below.
**Note:** We’ve just created a special type of definition, a *binary* definition called “Activated”. A binary definition is a definition with only two possible values, true (yes) and false (no). So “Activated” is either true or false. A binary definition is **required** for a condition or a conditional action, and now we are going to create some conditional actions.

14) Right-clicking the encounter, we’ll Add an Action → Conditional Action → Do the following actions if *Condition* is true.

15) Under the “Condition” tab → Select Binary → choose “Activated” from the dropdown.
16) Now repeat steps 14 and 15, but choose “Do the following actions if Condition is false”. This is to catch both situations: when the lever is “Activated” and when it is not “Activated”. Check with the following screenshot.

17) Under the “Do the following actions if Activated is true” Action, we need to toggle the integer value (from 1 to 0) and we need to animate the lever. First we'll toggle by Add an Action → Action Atom → Variable Assignment → Assign Integer Value to Target’s variable labelled Label
18) Under the “Target” tab → dropdown menu → The Placeable (Lever#1).

19) Under the “Label” tab → Constant → put “toggle” in the textbox. Leave the Integer Value as “0”. Remember “toggle” is the label we gave to the integer in the definition “LevelToggle”.

20) To animate the lever, simply right-click “Then, Do the following actions if Activated is true”, Add an Action → Action Atom → Animations → Animate The Placeable to Animation.

21) Under the “The Placeable” tab → Select Placeable Object → The Placeable (Lever#1), and under the “Animation” tab → Constant → Deactivate.

22) Now that we've finished scripting the lever's actions when the “Activated” binary is true (when the lever is already in the “on” position), we need to do similar things for when “Activated” is false. Under the “Do the following actions if Activated is false” Action, we need to toggle the integer's value (from 0 to 1) and we need to animate the lever. First we'll toggle by Add an Action → Action Atom → Variable Assignment → Assign Integer Value to Target’s variable labelled Label

Repeat steps 18 to 21 except that in step 19, we'll assign “1” to The Placeable's integer labeled “toggle”, and in step 21, we'll animate The Placeable to “Activate”.

23) Your module should now look like the screenshot below.

Intuitively, what have we done? Look at the encounter. We are saying that:
When someone uses Lever #1,
   If this Lever has already been activated,
       Then deactivate this Lever.
   If this Lever has not been activated,
       Then activate this Lever.

“Save and Compile” your module and make sure that the lever changes position each time you “use” it. Now, let's make the other levers work.
Getting the other levers going:

24) We still haven't got Levers 2 and 3 doing anything, but now that Lever #1 is properly configured, it is quite simple to get the others going as well. First, right-click on the “When Lever#1 is used” Encounter, and choose “Copy” from the menu.

25) Right-click on the “LeverMagic4.mod” folder, and choose “Paste”. Repeat this to paste a second copy of “When Lever#1 is used”.

26) Select the first copy, under “The Placeable” tab → Module Blueprint → Pick... →
Lever#2. Repeat these steps for the second copy, changing the Placeable to reflect the use of Lever#3.

27) Now save and compile your module, and try out your newly animated levers. “So simple?” you ask? Yes. By constructing the first lever properly (by using variable names like “The Placeable” instead of a specific object), we made it easy to reuse encounters. This is the “reason” mentioned way back in step 8.
Plot Tokens:

Making tokens:

28) Plot tokens are sort of revisiting the notion of “state” that we discussed when animating the levers. We're going to use them so that the scripts know if the player has pulled the levers in the correct order. If you've forgotten, the order that the Guardian has prescribed is 1, 2, then 3. Let's create some plot tokens to help us enforce this order.

29) In SE, and choose “Plot Token Builder” from under the “Tools” menu at the top of the screen. The location of the “Tool” menu is illustrated in the screenshot below.
30) In the “Plot Token Builder” window that opens, **right-click** in the big white box that's in the centre of the window. Choose “New Plot Token” from the menu.

31) Your new plot token will be automatically selected (indicated by a light blue highlighting of your plot token). In the “Token Name” box, put in “lever1token”.

32) Repeat that steps 30 and 31 to create “lever2token”. They are automatically saved so you don’t need to worry about saving them.

33) Left-click the 'X' in the top right corner of the “Plot Token Builder” window to close it.

Getting tokens:

34) Expand the “When Lever#1 is used” Encounter, right-click and Add an Action → Action Atom → Plot → Assign **Plot Token** to **Recipient**

35) In the Action you've just created, under the “Plot Token” tab → Constant → lever1token, and under the “Recipient” tab → dropdown menu → **User**. Your module should now look like the screenshot below.

With this action, when the player uses Lever #1, a plot token called “lever1token” will be assigned to the player. This plot token is like a ticket marking the player for touching Lever #1.
36) Expand the “When Lever#2 is used” Encounter, right-click and Add a Definition → Binary → Plot → Define Owns as whether Owner owns plot token The Token. Under the “Description” tab → Label → change to Owns1, under the Owner tab → dropdown menu → User, and under “The Token” tab → Constant → lever1token.

37) Right-click the encounter and Add an Action → Conditional Action → Do...if true. Under the “Condition” tab → Select Binary → Owns1.

38) Right-click on the “Then, Do the following actions if Owns1 is true” Action and Add an Action → Action Atom → Plot → Assign Plot Token to Recipient. Under the “Plot Token” tab → Constant → lever2token and under the “Recipient” tab → dropdown menu → User. Your module should look like this screenshot:
Intuitively, we are saying that:
When the player uses Lever #2,
    If the player has already used Lever #1 (thus owns “lever1token”),
    Then assign “lever2token” to the player.

Opening doors:

39) We’ve made sure that the player has earned their two plot tokens, but we need something to happen when they pull that third lever. We need to open the door. This
will require us to make sure that the player has both the previous tokens.

40) Expand the “When Lever#3 is used” Encounter, right-click and Add a Definition → Binary → Plot → Define Owns as whether Owner owns plot token The Token. Under the “Description” tab → Label → change to Owns2, under the Owner tab → dropdown menu → User, and under “The Token” tab → Constant → lever2token.

41) Right-click the “When Lever#3 is used” encounter and Add an Action → Conditional Action → Do...if true. Under the “Condition” tab → Select Binary → Owns2.

42) Right-click the “Do the following actions if Owns2 is true” action and Add an Action → Action Encounter → Destroyed Object is destroyed with visual effect Visual Effect nearby. Under the “Destroyed Object” tab → Module Blueprint → Pick... → Vault Door (the Vault Door is under the Doors category on the icons row at the top), and under the “Visual Effect” tab → Constant → Lightning I. Your module should look like this screenshot.
43) “Save and Compile” your module and check out your handiwork.

Treasure:

44) Now we're at the real reason why you would create a module in the first place: to
make the best treasure possible and give it to your characters. Use your imagination. Do whatever you like but make sure to place it INSIDE the vault.

That’s all folks. If you're cagey, you may have noticed that there are a few holes in our module: you don't HAVE to talk to the guardian to get into the vault, pulling all the levers in a random order will eventually open the door, etc. These are left for you to tackle on your own as you continue to explore the world of NWN, Aurora, and SE.