

Tutorial 3: Use Procedures and Arguments, Add Rotation and Randomization

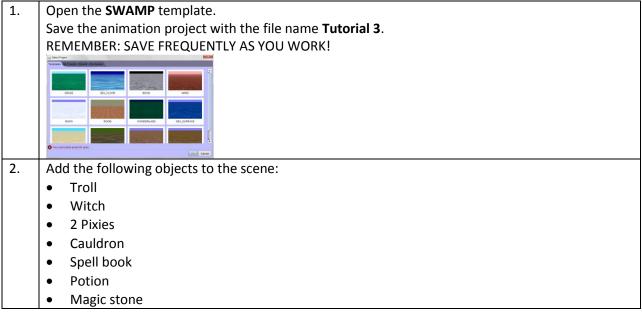
In this tutorial, you will practice using procedures and arguments to create an				
animation. You will practice rotating an object's subparts and invoking random				
movement to make an object's movements less predictable.				
Use procedures to move objects				
 Add Java programming procedures to the Code editor 				
 Demonstrate how procedure values can be altered 				
Create programming comments				
 Reorder, edit, delete, copy, and disable programming statements 				
Test and debug an animation				
 Add a control statement to the Code editor 				
Use random numbers to randomize motion				
Beginner: This tutorial is appropriate for someone who has used Alice 3 to:				
Add multiple objects to a scene				
Code a simple programming instruction				
 Use precise positioning and drag-and-drop positioning 				
 Use a one-shot procedure to precisely position an object in a scene 				
 Position the sub-parts of an object in the Scene editor 				
1 hour				
This tutorial was built using Alice 3.1.81.				

Part 1: Define the Scenario

Review the scenario and corresponding animation. In this tutorial, you are going to create an animation from the scenario defined below.

Scenario	Animation	
A witch casts a spell on a troll.	A witch casts a spell on a troll using one of her	
	potions. The troll doubles in size.	

Part 2: Add Objects to the Scene



Copyright © 2014, Oracle and/or its affiliates. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.



3. Position the objects in the scene as follows. Notice that some objects were resized, and the spell book is in the witch's hand. When you are finished, your scene should look like this:

 Image: Comparison of the scene structure

 Image: Comparison of the scene structure

Part 3: Program Objects to Move

1.	Click the Edit Code button to go to the Code editor.			
2.	Select the Witch's Mouth from the instance menu.			
3.	Drag the turn procedure into myFirstMethod. Select arguments: FORWARD \rightarrow 0.125 .			
4.	Drag the turn procedure into myFirstMethod.			
	Select arguments: BACKWARD \rightarrow 0.125 .			
5.	Run the animation Run. Observe how the witch's mouth moves like she is talking.			
6.	Drag a Do In Order do in order control statement into myFirstMethod. The Do In Order tile is located at the bottom of the Code editor: do in order count_while_for each in_ff_ do together each in_together variable_assign //comment			
7.	Drag the turn procedures inside of the Do In Order.			
	This will make it easier to copy this statement so your witch can talk a few times.			
	Your programming statement should look like this:			

2



	declare procedure myFirstMethod declare procedure myFirstMethod do norder do in order do in order for in order for in order for in order in (file witch) getMouth) turn [FORWARD], 50 128 add detail					
	((his witz) getMouth) turn [BACKWARD] , ≝0123 add detall					
8.	Copy the Do In Order statement and paste it below. To do this:					
	Right-click on the Do In Order .					
	Select Copy to Clipboard.					
	do in order					
	This witch entitleuth turn EFORWARD , 50 125 add detail V is Enabled turn (BACKWARD , 50 125 add detail					
	Then click on the clipboard and drag the object from the clipboard to myFirstMethod . OR					
	Click the Control key (CTRL) on your keyboard, then click on the Do In Order and drag it below.					
	This will also create a copy.					
	After you copy and paste, your statements should look like this:					
	Scene initializeEventListeners myFirstMethod					
	declare procedure myFirstMethod					
	do in order do in order					
	(this witch "getMouth "turn [FORWARD ", =0.125 " add detail					
	(Chis.witch) getMouth Turn EBACKWARD , =0.126 add detail					
	do in order					
	(his witch) "getMouth" turn EACKWARD , =0.125" add detail					
9.	In the Instance menu, select the witch 🕅 this.witch					
10.	Drag a say procedure into the first Do In Order, above the turn procedures.					
	Select Custom TextString					
11	Enter text: Cast a spell					
11.	Drag a say procedure into the second Do In Order, above the turn procedures.					
	Select Custom TextString Enter text: TROLL RESIZE!					
12.	Your programming statements should look like this:					
12.	declare procedure myFirstMethod					
	do in order					
	(this witch) say (Cast a speil) add detail					
	(<u>(this witch</u>) getMouth turn [EORWARD], <u>E0125</u> add detail ((this witch) getMouth turn [BACKWARD], <u>E0125</u> add detail					
	l do in order					
	do in order (mis.witch" say .(TROLL RESIZET) add detail					
	do in order this witch'' say /(TROLL RESIZE)) i add detail'' this witch'' getMouth '' turn ''FORWARD'', E0.129 i add detail''					
	do in order (mis.witch" say .(TROLL RESIZET) add detail					
13.	do in order this witch'' say /(TROLL RESIZET)'' add detail'' this witch''' getMouth '' turn [FORWARD '', 50,125)'' add detail''					
13.	do in order (mis.witch' say /(TROLL RESIZEF)* add detail*) (mis.witch' getMouth* turn [FORWARD*, =0.128* add detail*) (mis.witch' getMouth* turn [BACKWARD*, =0.128* add detail*)					
13.	do in order (his witch' sey /TROLL RESIZE) add detail (his witch' getMouth) turn (FORWARD) (his witch' getMouth) turn (FORWARD) (his witch' getMouth) (his witch' getMouth)					
13.	Run the animation Run You will notice that the witch talks first, then moves her mouth. You want her to move her mouth					
	Run the animation Run. You will notice that the witch talks first, then moves her mouth. You want her to move her mouth at the same time she is saying something.					



17.	Drag the first Do In Order inside of the Do Together.				
-/.	Your programming statement should look like this:				
declare procedure myFirstMethod					
do in order do together					
	this.witch say /Cast a spell add detail				
	(this witch getMouth) turn [FORWARD], =0.125 add detail				
	(this witch)" getMouth) Turn [BACKWARD]", 20125 add detail				
18.	Run the animation.				
10	You will notice that the witch talks while moving her mouth.				
19. Repeat these steps to edit the second set of programming statements so that the with "TROLL RESIZE!" while moving her mouth.					
	When you are done, your programming statements should look like this:				
	declare procedure myFirstMethod				
	do in order I do together				
	(<u>this witch</u> say <u>if Cast a spell</u>) add detail				
	(this witch) getMouth turn [FORWARD], =0.1231 add detail				
	(<u>this witch</u>)" getMouth turn [BACKWARD], <u>[0.126</u>]" add detail"				
	do together				
	(Inis witch) say /(TROLL RESIZE) add detail do in order				
	(this witch'' getMouth) turn [FORWARD) , 50125 add detail				
	(<u>this witch</u>)" getMouth turm [BACKWARD]", <u>10128</u> " add detail"				
20.	Run the animation.				
	Next, you will program the troll to move his arms, then grow larger.				
21.	Select the Troll from the instance menu.				
22	Dues the seven second up into my First Mathed heles, the other programming statements				
22.	Drag the say procedure into myFirstMethod, below the other programming statements. Select Custom TextString				
	Enter text: Oh no!				
23.	Drag a Do Together into myFirstMethod.				
24.	Select the troll's left shoulder from the instance menu.				
25.	Drag a roll procedure into the Do Together.				
23.	Select arguments: LEFT \rightarrow 0.25.				
26.	Select the Troll's right shoulder from the instance menu.				
27.	Drag a roll procedure into the Do Together.				
27.	Select arguments: RIGHT \rightarrow 0.25 .				
28.	Select arguments: RIGHT \rightarrow 0.25. Select the Troll from the instance menu.				
29.	Drag the resize procedure into myFirstMethod.				
29.	Select argument: 2.0.				
30.	Run the animation.				
31.	<i>Next, you will randomize the size of the troll, so that each time the animation is run, the troll becomes a different size.</i>				
	In the resize procedure, click on the factor argument (currently set at 2.0).				
	וו נויב וכאב איטכבענויכ, נוונג טו נויב ומכנטו מושמווכות (בעורבוונוץ זכן מן 2.0).				



	Select the Random menu option.					
	this troll resize 2.0 add detail					
	factor. =2.0 (current value)					
	≣0.25					
	=0.6					
	≦ <u>1.0</u> ≣2.0					
	Ξ <u>10.0</u>	EnextRandomRealNumberInRange E0.0, E1.0				
	Random	EnextRandomRealNumberInRange €???), €???) ►				
32.	Select nextRandomRealNumberInRange ???, ???					
	Select 1.0 for the first value, then 10.0 for the second value.					
	This will set the resize factor of the Troll to a random number between 1 and 10 each time the					
	animation is run.					
			=0.25			
	(this.trol) / resize 2.0 add detail /		=0.29 =0.5			
	factor: 20 (current value)		=1.0			
	=0.25	20.25	=2:0			
	=0.6	= <u>0.5</u> =1.0	Custom DecimalNumber			
		=======================================	•			
	210.0 Ene	xtRandomRealNumberInRange <u>50.0</u> , <u>51.0</u> 510.0				
	Random	ktRandomRealNumberInRange 특???), 특???) > Custom Decimal				
33.	Run the animation. Continue editing the animation to make it unique!					

Part 4: Give the Animation a Unique Ending

Now that the animation is complete, you need to give the animation a unique ending. Here are some ideas to spur your thinking:

- The troll picks up the witch and tosses her around.
- The pixies cast a spell on the witch and make her very small.
- The troll, witch and pixies dance.