Fluency with Alice

Workbook to Accompany Snyder's *Fluency with Information Technology, 4th Edition*

by Robert Seidman, Philip Funk, James Isaak, Lundy Lewis

Chapter 3. The Rest of the Story

In this Chapter we introduce you to several critical steps in directing – getting the right screen play, understanding where you want to take the story, and identifying key recurring elements of the story. Unlike a movie or play, and more like a video game, we also introduce the idea that the viewer may have an opportunity to influence the outcome of the story. Finally, we consider some level of random occurrences that can make the story both more credible and a little different for each viewing. Along the way, we also introduce more of the Alice environment and show you how to create your own reusable methods.

3.1: The Whole Story

"`Would you tell me, please, which way I ought to go from here?'
`That depends a good deal on where you want to get to,' said the Cat.
`I don't much care where--' said Alice.
'Then it doesn't matter which way you go,' said the Cat." – Alice in Wonderland – Lewis Carroll

Tech Talk – In the I.T. world, the story board is presented as "requirements" often in the form of "use cases" – how someone will use the system. For virtual worlds, the movie industry model works fairly well since your objective is to tell a story – albeit for advertising, educational or simply entertainment purposes.

When we last left our hero, Peter, and heroine, Robin, what was this story we were telling? Was it a drama, a murder mystery, a comedy, a romance? What is the overall "story arc," where will it end, what is the climactic moment? Is there conflict or tension? In the movie, game, and advertising businesses, this overall picture is called a "story board." Often it is a series of sketched images that reflect key incidents in the story.

What we would like for our story is a "romance." Peter finds Robin attractive – he would like to be friends, or more, but he is a penguin.

When we left our story, Robin is embarrassed, perhaps backing away. Peter needs to gather his courage, approach Robin, let her know he understands her reaction, and - if all goes well - the happy pair can wander off into the sunset. To add a bit of tension, we can introduce a threat, say, a fire breathing dragon presenting a challenge to Peter. And, let's do something different and fun. Let's let the movie have two endings where the viewer provides some input and the movie unfolds differently based on the user's choice. Read on.

We display our story in the form of a story board as shown in **Figure 3-1**. The story begins with the sequence as played out in Act 1. Then, as **peter** seeks to respond to **robin**, a dragon appears, shoots fire at **peter**, and challenges him for **robin's** affections. Decision time: are **robin** and **peter** to be a couple? If so, **peter** conquers the dragon and they walk happily into the sunset; if not – the dragon conquers **peter** and claims **robin** as his prize.

The story board

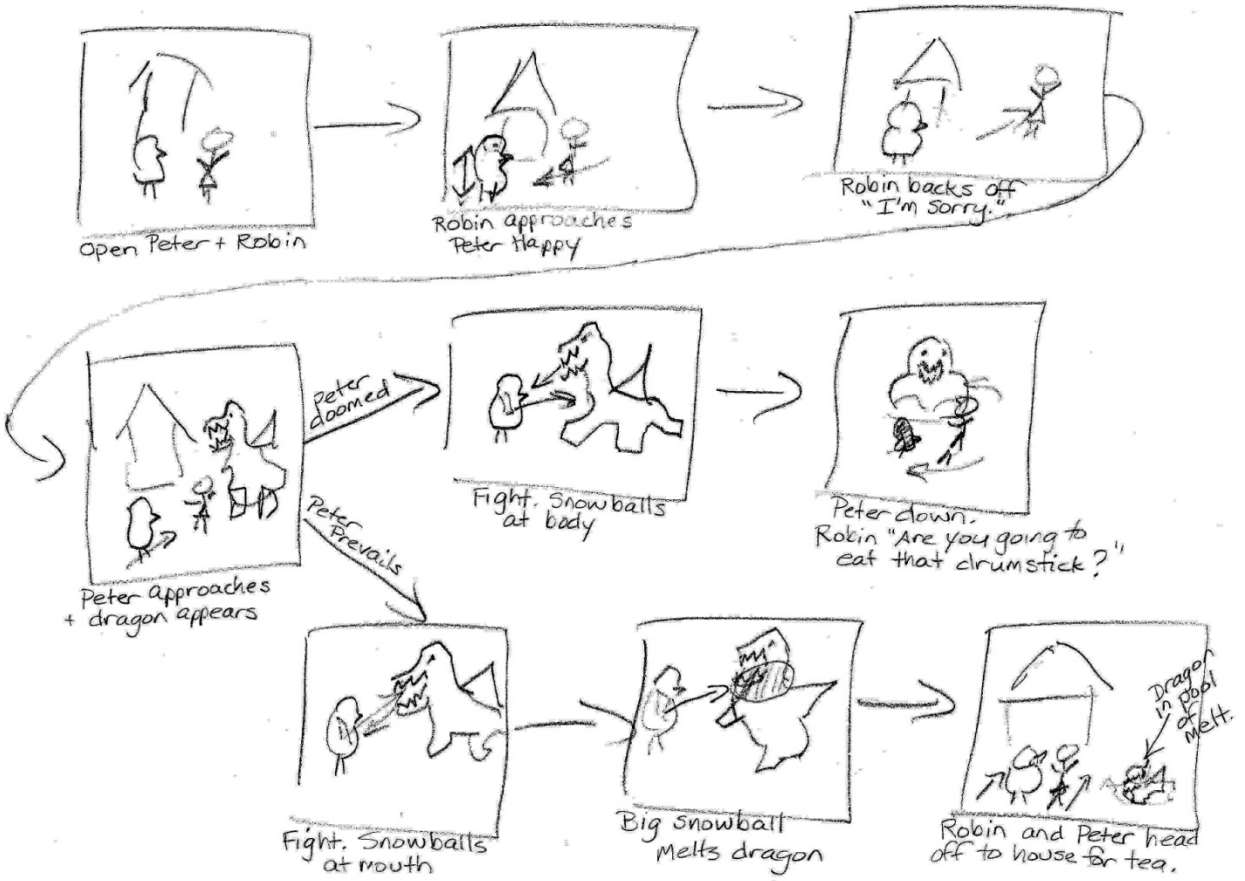


Figure 3-1 Story board.

Here is a video of the *act2* animation that you will produce in this chapter. {**Video demo**: Video 3-1}

Here are two video animations of the combined *act1* (from Chapters 1 & 2) and *act2* (from this chapter) that you will produce by the end of this chapter. We show you two versions because they differ depending upon user choice: {**Video demos**: Video 3-2a & Video 3-2b}

Let's begin by making a new copy of your work this far. Open your Alice world file as of the end of Chapter 2 in this workbook. This is where we last left **peter** and **robin**. Save this world with a new name such as "Peter and Robin Chapter 3". **See Figure 3-2a.** In this way, you retain the copy of your work thus far under the name you saved it at the end of Chapter 2. [Alternatively, you can download the Workbook authors' end of Chapter 2 Alice file, renamed **AWB3-1.a2w and can be found at: http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/**. Good to then save it with a new name.]

When Alice opens any world file, you see the **world**.*my first method* in the **Method Editor** panel. By default the **world** object's **Details** panel shows. See **Figure 3-2b** for the **AWB3-1.a2w** file. To open the *act2* method in the **Method Editor** panel, click on the method's edit tile in the **Details** panel. See arrows in **Figure 3-2b**.

Figure 3-2a Save world file with a new name.

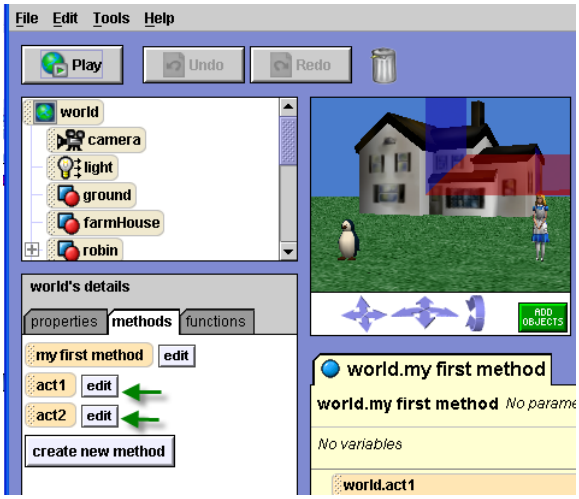


Figure 3-2b Default Alice file opening.

3.2: Set the Stage for Act 2

When we begin Act 2 we want our actors to be in the same positions they were in at the end of Act 1. This would be no problem if we run the Alice method *act1* every time we run *act2*. However, you, the Director, will want to be able to run *act2* with the characters in their proper starting places without first running *act1* each time.

That's obviously no problem as far as **farmHouse** is concerned – it never moved. The **peter** object did move in the sense that he jumped up and down, turned around, and flapped his wings. Still, he ended in his original position in the world, turned to face **robin**. The **robin** object, on the other hand, moved around quite a bit. First, she turned to face **peter** and then moved toward **peter** as long as she was at least two meters away from **peter**. Then, she moved backward one meter. It is unlikely that **robin** ended up in her original starting position. At the end of the *act1* method we also changed the position of the camera and dimmed the lights. We will need to restore these as well.

Alice concept – In the Introduction to this Workbook, we mentioned that Alice is a 3D environment. Each object has a position in this world environment. Each object's position is specified by its location relative to the center of the world. This concept will be explored in more depth later in this chapter and in Chapter 4.

To make sure that our actors are in their proper places at the beginning of Act 2, you need to create the *act2* instructions shown in **Figure 3-3** so that the characters can take their places for the start of *act2*. Read on for "How to copy" and watch the video which shows how to copy a few of the *act1* instructions to *act2*.

How to copy: You can actually copy/paste all of the instructions shown in **Figure 3-3** from *act1* right into *act2*. Here's how: Take a look at the upper-right-hand corner of your Alice window. Notice the Clipboard icon  . To copy an Alice instruction or control, drag it to the Clipboard and drop it in. To take something from the Clipboard and paste it into the **Method Editor** panel, simply drag the Clipboard into the location you want it to go. The Clipboard does not move but its contents do. {**Video demo**: Video 3-3} Save your world often during this process.

Tip: You can have more than one Clipboard in your Alice GUI. Here's how: **Edit / Preferences / Seldom Used** tab **/ number of clipboards / OK** .

Set the *duration* for all but the last instruction to 0 seconds so we do not see the actors take their places but we do see the light brighten at the beginning of *act2*. [Hint: you might want to try running the instructions first without changing the durations to 0 to see how the actors take their positions in the world.]

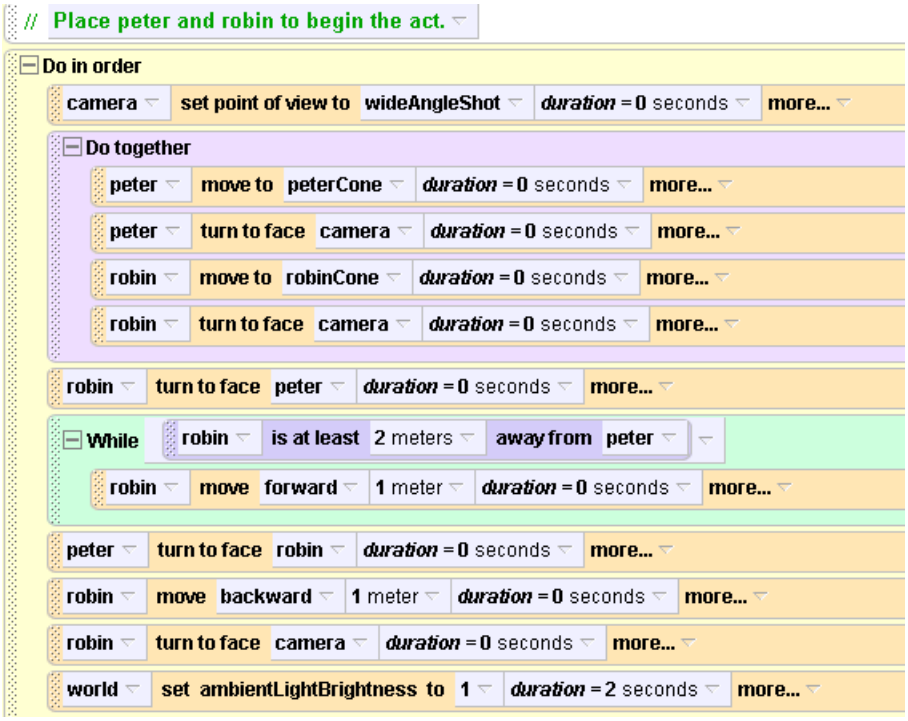


Figure 3-3 Position the actors at the beginning of act2.

Before you click Play to try out *act2*, it is best to disable *act1* so that it does not run. In the **Method Editor** panel, click on the **world**.*my first method* tab. Right click on the *act1* tile and select disable. Also, enable *act2* in the same way. You will see this: 

Then click Play. If you see prolonged movements, then you did not use *duration*=0 as your instruction parameters.

Authors' file AWB3-2.a2w to this point in the chapter can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Transition between acts

Perhaps we should include a transition between *act1* and *act2*. In a real play, there are intermissions between acts to give the actors and the audience a break. We will include a notice to the viewer that Act 1 has ended.

Create and position an end-of-Act 1 message

In Chapter 1, we announced the beginning of our story by having the **farmHouse** object say "Hello". This time, we will create a text object by using Alice 3D Text. To do so, in the **World View** panel, click on the ADD OBJECTS button. In the Local Gallery, scroll to the right until you see the Create 3D Text tile. See **Figure 3-4**. Click on the tile.

In the pop-up window that opens, enter the text that you want to display at the end of *act1*. You may also use the Font drop-down menu to select the font you want to use, such as Baskerville Old Face. **See Figure 3-5**.

Figure 3-4 3D Text.

Figure 3-5 Adding 3D Text.

After you enter the text that you want to display, click on the OK button to close the pop-up window and then click on the DONE button on the right side of the screen to exit the ADD OBJECTS window. The text you entered will be displayed in the **World View** panel. Notice that a new object with the same name as the text you entered has been added to the **Object Tree** panel. Rename this object using the standard naming convention discussed in Chapters 1 & 2: begin the name with a lower case letter; capitalize the first letter of all following words; remove spaces. In this example, we renamed the object to **endOfAct1**. See **Figure 3-6**.

Figure 3-6 Renamed 3D Text object.

Notice in **Figure 3-6** that in the **World View** panel the **endOfAct1** text object appears between the **peter** and **robin** objects with **peter** slightly in front of and **robin** slightly behind the text.

When objects are added to a world, Alice places them in a default position that is not always where you expect or want them to be. One way of observing more detail about the placement of objects in a world is through the quad view capability in Alice.

To view a world in quad view, click on the ADD OBJECTS button and then on the quad view button in the upper right corner of the screen. In addition to the normal world view, you should also see a view of objects in your world from the top, right, and front. Your screen should look something like **Figure 3-7.**

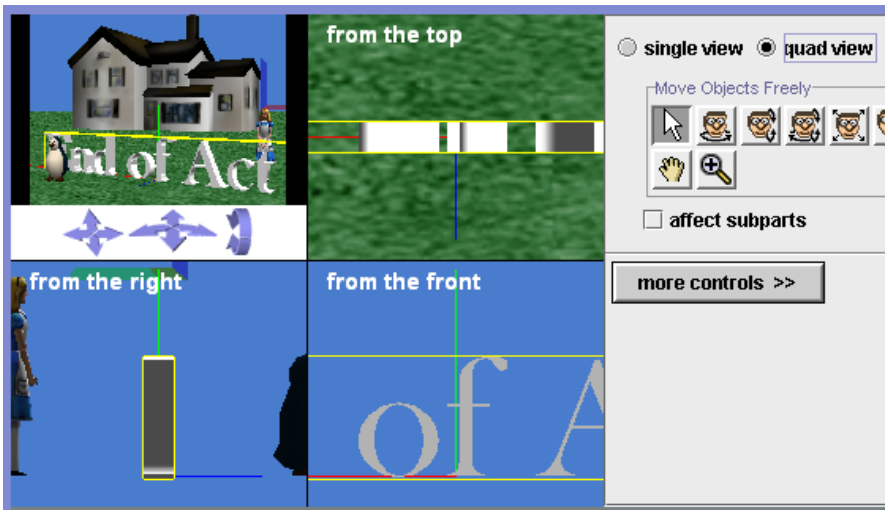


Figure 3-7 Quad view

Two helpful controls in quad view are the "hand" control that lets you scroll any of the views independent of the others, and the "zoom" control that lets you zoom in and out on any of the four quad views. Click on one of the two controls to use it in a quad view window. Practice scrolling around the different views to locate and view all objects in your world. The white up-arrow lets you move objects in any view. The other controls also apply to objects.

Alice Buttons – the "hand" control looks like this  and the "zoom" control looks like this  .

When you are finished, click the DONE button to exit the ADD OBJECTS screen. Don't worry about restoring anything. When you next open the world the quad views will be restored.

Explore – This is a good time to use what we learned earlier about an object's position in a world. In the Object Tree panel, click on several of the objects in your world and then on the property tab in their Details panel. Notice the values of the position in each object's *pointOfView* property (scroll down to find it). In particular, note the position of the endOfAct1 3D text object. Its position will probably be in the center of the world. We will discuss the *pointOfView* property later in this chapter and more in Chapter 4.

Obviously we do not want the **endOfAct1** 3D Text object "on stage" when our story begins. Let's move it about 10 meters off stage to its left until we need it. In the **Object Tree** panel, right-click on the **endOfAct1** object and then on methods/**endOfAct1**/*move/left/amount/10 meters*.

Notice that the 3D Text disappears from the **World View** panel. If you like, experiment with returning the 3D Text to the **World View** panel either by clicking the Undo button or by right-clicking on **endOfAct1** in the **Object Tree** panel and then selecting methods/**endOfAct1**/*move/right/amount/10 meters*. Notice that the 3D Text returns to the stage. Move it off screen again.

Now that we have the **endOfAct1** 3D Text created and safely stored off-screen, we need to add instructions to display the text at the end of *act1*. We could add the needed instructions either to the **world**.*act1* method or to **world**.*my first method*. Let's let the **world**.*act1* method handle the details of Act 1 and **world**.*my first method* handle transition between *act1* and *act2*.

To display the 3D Text correctly, we will need to do several things at the same time. This is a little tricky, but it does provide additional examples of controlling the location of objects in a world.

Click on the **world**.*my first method* tab. Let's begin revising the **world**.*my first method* by dragging a comment and a Do together control immediately under the **world**.*act1* instruction as shown in **Figure 3-8**.

Good idea to save your world.

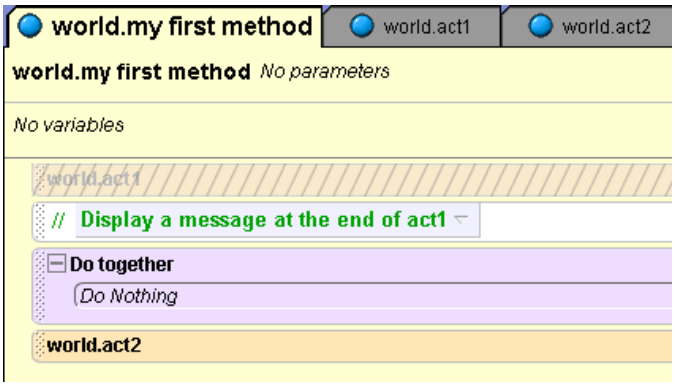


Figure 3-8 Comment and Do together control added in my first method.

Let's try this out. First, let's see what will happen if we simply return the **endOfAct1** 3D Text to its original position in the world. We can do this by creating an instruction for the 3D text that moves it 10 meters to the right as shown in **Figure 3-9**.

If you haven't already done so, edit the **world.***my first method* and temporarily disable *act2* by right-clicking on *act2* and selecting disable in the drop-down list. Next enable *act1*, which was previously disabled. Right-click on *act1* and select enable in the drop-down list.

Then, drag and drop the **endOfAct1** *move right* 10 meters method into the new Do together control. See **Figure 3-9**.

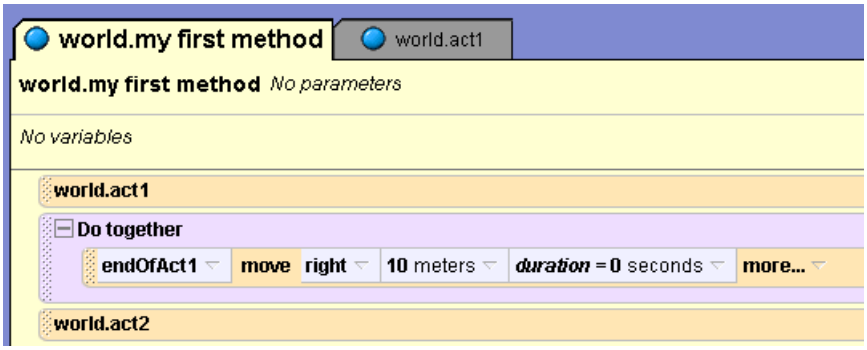


Figure 3-9 Ready to Play

After making these changes, Play the world to see the results. **Ooops!** What happened to the "End of Act 1" 3D Text?

Actually, you should not see the **endOfAct1** 3D Text. The reason is that the **camera** has moved to a close-up view of **robin's** face and the **endOfAct1** 3D text when returned to its original position is not in view of the **camera**. We need to make some position adjustments so the "End of Act 1" 3D Text will display.

But first, please take the time to read the **Object PointOfView = Position + Orientation** information on the next page. It will help you understand an important concept.

Object PointOfView = Position + Orientation

Objects have a property called *pointOfView* that consists of two aspects: position & orientation.

Position is the object's location in the **world** relative to the **world's** origin (or central point). Notice that **robin** and **peter** are at different locations (i.e., different positions relative to the **world's** origin). You can see the **world's** axes by clicking on **ground** in the **Object Tree** panel.

Consider this model airplane. Alice has a number of methods that can alter the airplane's position. One is the *move* method. This method can move the airplane straight up, straight down, forward, backward, slide the airplane to the left and slide it to the right. Try this out with **peter**: Right-click on peter in the **Object Tree** panel; Select methods; Select *move*; Select *up*; Select *½ meter*. Notice what happens to **peter**. Then click the Undo button to restore **peter**. Do the same for *move down*, *move left*, *move right*, *move forward*, *move backward* using the Undo button to restore **peter** each time.

Orientation is the way the object is facing at its position. Let's think of orientation using a model plane example. Suppose that you are holding the model airplane in your hand. Tip the nose down. Then tip it up. This is accomplished by the Alice *turn forward* method and the *turn backward* method, respectively. Try it out with **peter** in the same way you did with the *move* method, above. Use the Undo button to restore **peter**.

With the airplane in your hand, you could swivel the plane left or right. This is accomplished by the Alice *turn left* method and *turn right* method. Try it out with **peter** in the same way you did with the *move* method, above. Use the Undo button to restore **peter**.

You can tip the plane's wings up and down by rolling the plane. The Alice *roll* method will do this. Try *roll left* and *roll right* on **peter** in the same way you did with the *move* method, above. Use the Undo button to restore **peter**.

The *roll* method and the *turn* method serve to orient the plane (or **peter** if you've been playing with him). Notice that the plane does not change its physical location (i.e., position) in your world. It has only tipped its nose, rolled its wings, and swiveled its body. In other words, it has maintained the same position, but has changed its orientation and thus has changed its point of view (i.e., POV=position + orientation).

Continuing on, we know that **robin** has moved to a new position in the world as identified by her *pointOfView*. Let's begin by setting the **endOfAct1's** *pointOfView* to be the same as **robin's**.

In the **Object Tree** panel, click on **endOfAct1** to select it and then click on the properties tab in **endOfAct1's Details** panel.

In the properties tab, scroll down to find the *pointOfView* property and drag it into the **world**.*my first method* edit panel right under the **endOfAct1** move instruction. In the drop-down list that opens, select the default value. Notice that what is created is a message to **endOfAct1** to set its *pointOfView* to the default position of all zeros. The instructions are shown in **Figure 3-10**.

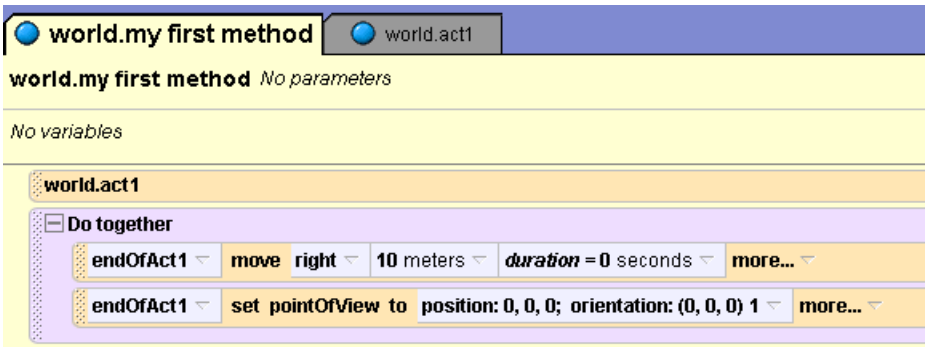


Figure 3-10 Setting point of view with default parameters.

We now need to change the default **endOfAct1** *pointOfView* property values to be the same as **robin's**. Click on **robin** in the **Object Tree** panel and then on the properties tab in the **robin's Details** panel. Scroll down to find **robin's** *pointOfView* property and drag it into position in the instruction as shown in **Figure 3-11**.

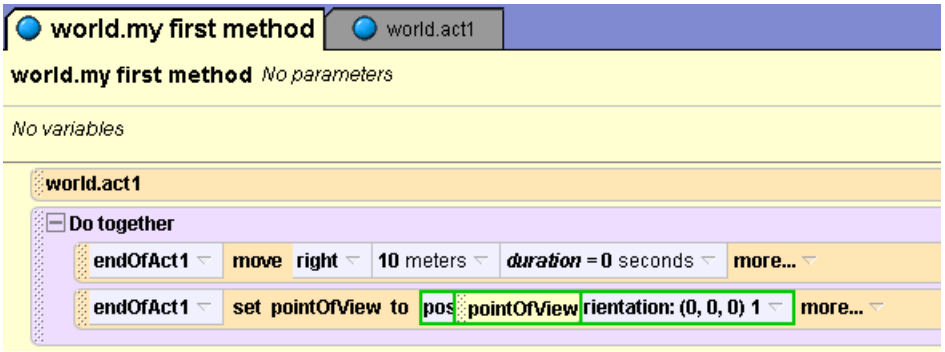


Figure 3-11 endOfAct1 point of view set to robin's Point of view.

The completed instruction will look like:

As discussed previously, an object's position in a world is represented by a set of numeric values that indicate a location (i.e., a point) in the world. We need to move the position of the 3D Text higher in the world so it is approximately the same level as **robin's** face. To do this, select **endOfAct1** in the **Object Tree** panel, select the methods tab in **endOfAct1's** details, and drag the method *move* into the **world**.*my first method* as shown in **Figure 3-12**. In the drop-down list, select up and enter 1.5 meters. If you do not see 1.5, then select other to use a keypad.

Make sure you set all the *durations* of all instructions to zero as shown in **Figure 3-12.** Play the world to see the results shown in **Figure 3-13.**

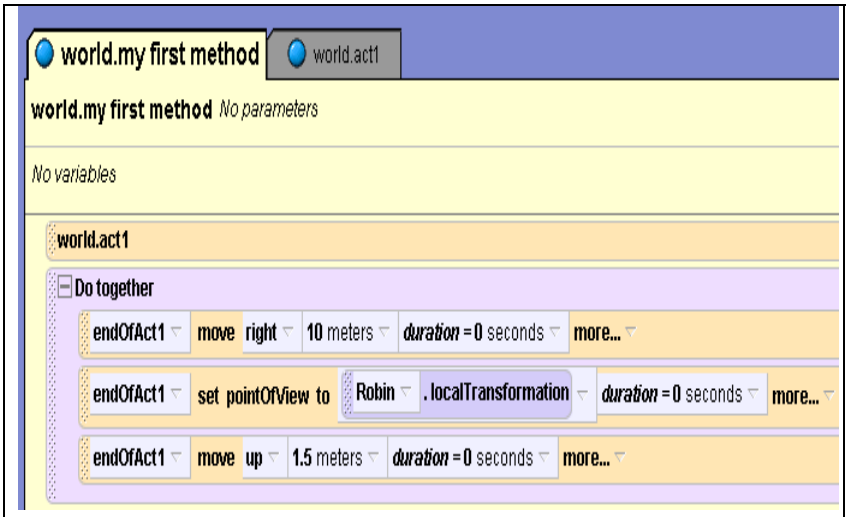


Figure 3-12 Instructions to display endOfAct1 message.

Figure 3-13 Results.

Part of the 3D Text appears, but it is obviously too large for us to see it in its entirety. Let's reduce it to 20% of its original size. Select **endOfAct1** in the **Object Tree** panel. Select the methods tab and drag the **endOfAct1** *resize* method into the **world**.*my first method* after (**not inside**) the Do together control. Select other and using the keypad, set the *resize* parameter to 0.1. Then set the instruction *duration* to 0 seconds. See **Figure 3-14**.

Finally, because part of the 3D Text may appear behind **robin's** head (depending on **robin's** positioning, your result may be a little different), we want to make **robin** invisible. To do this, click on **robin** in the **Object Tree** panel and select properties in the **robin's Details** panel. Drag the *isShowing* property into the end of the **world**.*my first method* and select *false* from the drop-down list. See **Figure 3-14**.

We also want to make sure the **endOfAct1** 3D Text displays long enough for the viewer to read it before continuing to *act2*. To do this, drag a Wait control from the bottom of the **Method Editor** panel to just under the instruction to make **robin** invisible and set the *duration* to 5 seconds. The final result is shown in **Figure 3-14**.

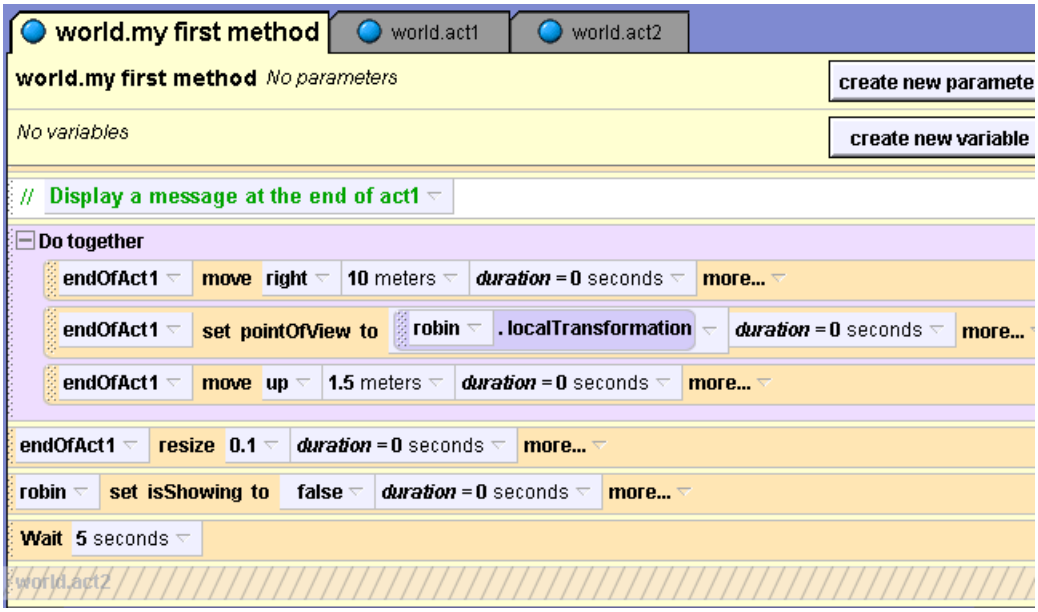


Figure 3-14 Display end of Act 1 message.

Play the world to make sure everything works right. If you can't see the entire 3D Text, then you can change the resize value to something less than 0.1.

Make *act2* active in the **Method Editor** panel. You can do this by clicking **world** in the **Object Tree** panel and then by clicking on its methods tab in the **Details** panel. Click on the edit button to the right of the *act2* method tile.

There are two additional adjustments that we need to make. We ended *act1* with **robin** invisible and the **endOfAct1** 3D Text message displayed. We want to reverse this situation at the beginning of *act2*. To do so, add a Do in order control to the very beginning of the *act2* method. Drag the existing **camera**.*set point of view to* method into it and then add two other instructions shown in **Figure 3-15**. Be sure all *duration*=0 seconds and add the comment. {**Video Demo**: Video 3-4}

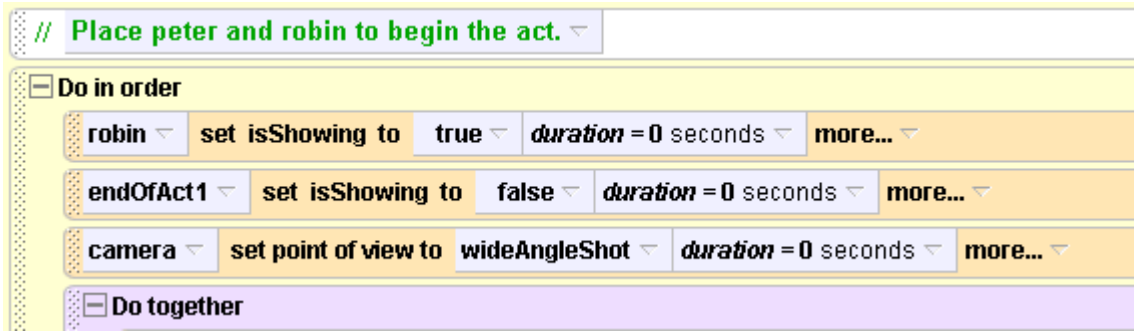


Figure 3-15 New instructions to clean up our act2.

When you are satisfied that everything is working properly, this would be a good time to save your world (pun intended).

Authors' file to this point is **AWB3-3.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

3.3: The Confrontation

"Though this be madness, yet there is method in't." – Hamlet, Act 2, Scene 2 – William Shakespeare

Now that *act2* is set up we are ready to start working on the Act 2 story parts. If you have not already done so, click on the **world**.*my first method* tab and then right-click on the *act1* instruction and select disable. Right-click on the *act2* instruction and choose enable.

To begin, select **world** in the **Object Tree** panel and then the methods tab in the **world's Details** panel. Click on the edit button next to *act2* in the list of **world's** methods. The *act2* method allows us to sequence our story in logical units. It also allows us to "rehearse" Act 2 without having to do all of Act 1. Segmenting a large thing (e.g., book into chapters, play into acts, etc.) allows us to concentrate on smaller pieces, one at a time.

Now we need to add our dragon. Fortunately, Alice has one. Click on the ADD OBJECTS button and scroll to the right in the Local Gallery until you find the Medieval folder. Click on the folder to open it and find the Class Dragon tile. Click on Class Dragon and then on the Add instance to world button. Click the DONE button.

The result should be rather interesting. It looks like the **dragon** was added in the middle of the **farmHouse**. This is because Alice objects have default positions in Alice worlds and it looks like the creators of the FarmHouse class and the Dragon class decided to put them in approximately the same position. Move **dragon** out of **farmHouse** by right-clicking on **dragon** in the **Object Tree** panel and in the drop-down list select methods/**dragon** *move to/robinCone*. Also, reduce the size of **dragon** by right-clicking on **dragon** in the **Object Tree** panel and then selecting methods/**dragon** *resize/amount ½ (half as big).*

That makes **dragon** a more reasonable size, but **dragon** appears half above and half below the ground. The reason is that each Alice object has a center point. For some objects, such as **peter** and **robin**, this center point is at their feet. For other objects, such as **dragon**, this center point is close to the middle of the object.

To see the difference, try the following experiment. First, right click on **peter** in the **Object Tree** panel and in the drop-down list, select methods/**peter** *turn/ direction forward/ ¼ revolution*. When you see the results, click on Undo to return **peter** to his original position. Now try the same thing with **dragon**. Select **dragon**, right-click on **dragon**, and then select methods/**dragon** *turn/ direction forward/ ¼ revolution*. Notice how **peter** and **dragon** turn about different axes because their center points are located in different places. Click on the Undo button to return **dragon** to its original position.

We still have to move **dragon** so that he (or she or it) is not partially underground. The easiest way to do this is to go to the ADD OBJECTS panel (if you are not still there). See **Figure 3-16**. Look at the "Move Objects Freely" list of buttons shown in **Figure 3-17**. Click on the Move Objects Up and Down with the Mouse button (first face from the left). In the **World View** panel drag the **dragon** up slowly until its feet are at the same level as **robin's** and **peter's** feet. Then, click on the white arrow (see **Figure 3-17**) and drag the **dragon** back until it is standing slightly behind **robin**.

The result should look like **Figure 3-18**. Click the DONE button to exit the ADD OBJECTS screen. Save file.

Authors' version of the file to this point is **AWB3-4.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

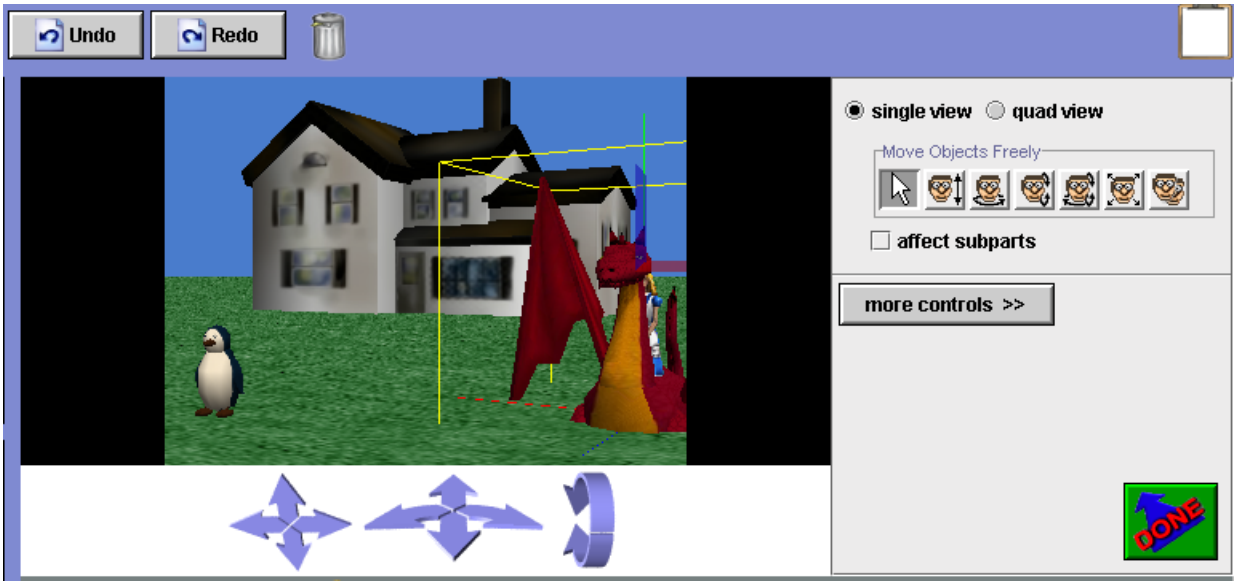


Figure 3-16 This is the world view.

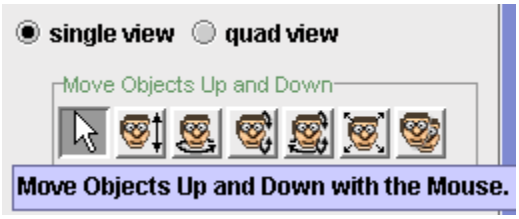


Figure 3-17 Move Objects Up and Down control.

Tech Talk – The center of the dragon. Each object has its own enclosing box and reference point. This is visible when you select the object in the Object Tree panel.

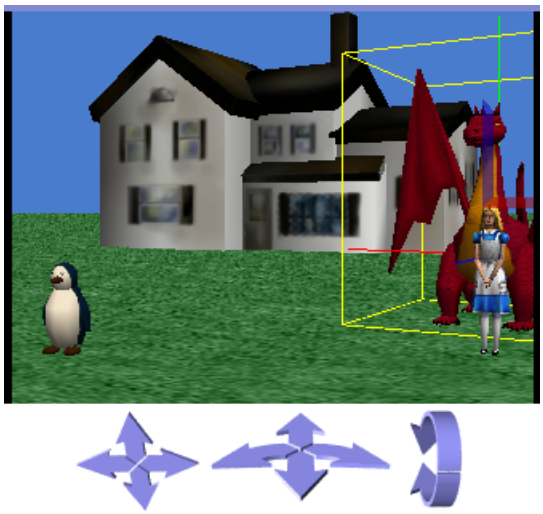


Figure 3-18 Everyone in correct positions.

At this point, we have positioned the **dragon** on the stage. But, since we don't want the **dragon** visible in *act1* we need to make him disappear. If you can toss a big sheet over him it would help, or just right-click on the **dragon** object in the **Object Tree** panel. Select methods/*set opacity* to/0 (0% invisible). Our **dragon** disappears.

Finally, in the **Object Tree** panel, rename **dragon** to **ollie**. Save your world.

As a reminder, we will want to have the **ollie** appear in the scene when **peter** approaches **robin**. An excerpt from our story board is shown in **Figure 3-19**.

Authors' version of the file to this point is **AWB3-5.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

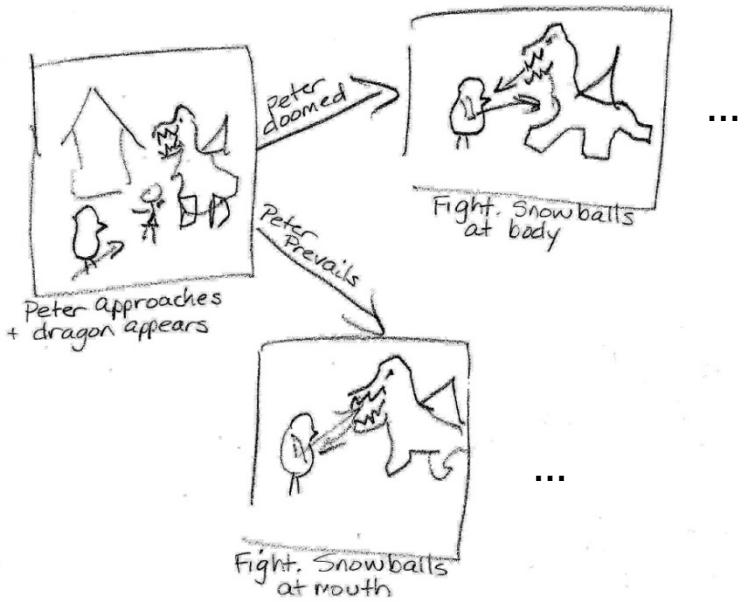


Figure 3-19 Story board for the dragon confrontation.

We are ready to continue work on *act2*. Make sure that the *act2* tab is active in the **Method Editor** panel.

At the beginning of *act2* we can have **ollie** materialize as **peter** is starting to make his move. First, make sure **peter** is facing **robin** by adding an instruction to have **peter** turn to face **robin**.

Since we know that **peter** was facing **robin** at the end of *act1*, we make the *duration* 0 seconds. Then add a Do together control so we can have **peter** walk toward **robin** and have **ollie** materialize at the same time. **peter** actually has a *walking* method, so drag the *walking* method into the Do together control as shown in **Figure 3-20** and set the number of walking steps to 2. In the Do together control, have **ollie's** *opacity* property change to 1. If you set the *duration* of the *opacity* change to 4 seconds, the transition works nicely. Note the comment is placed at the top of this new set of instructions. This helps us to identify blocks of instructions.

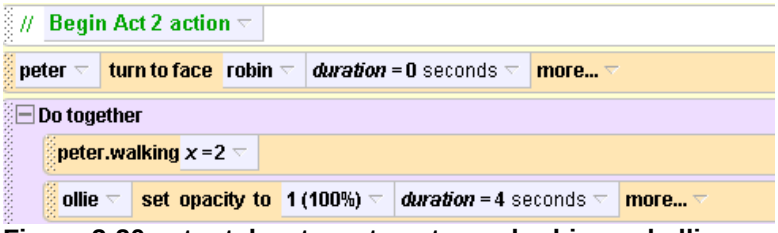


Figure 3-20 peter takes two steps toward robin, and ollie appears.

Drag a new Do in order control inside our Do together control right under 

, so that we can have **peter** talk as he walks before **ollie** becomes visible. See **Figure 3-21** to see the placement of this Do in order control.

Here is what you do to create the instructions shown in **Figure 3-21**. From the bottom of the screen, drag a Wait control for 1 second into this new Do in order block of instructions. Then, **peter** says "Robin, I ...", then another wait for 1 second, and then a **peter** says "Look Out!" at about the time that **ollie** materializes.

Note that if we did not use the Do in order control for **peter's** warning, the two statements from **peter** would happen simultaneously and we don't want that. The 'wait' instructions provide for dramatic tension (or at least we hope they do).

Figure 3-21 shows the updated instructions. Click Play to try it out. Save your world.

Note: You may have already disabled the *act1* method so you can concentrate on *act2* when you click on the Play button. Just know that the appearance of the **endOfAct1** sign will look a little strange. Of course it will be fine when both acts are enabled.

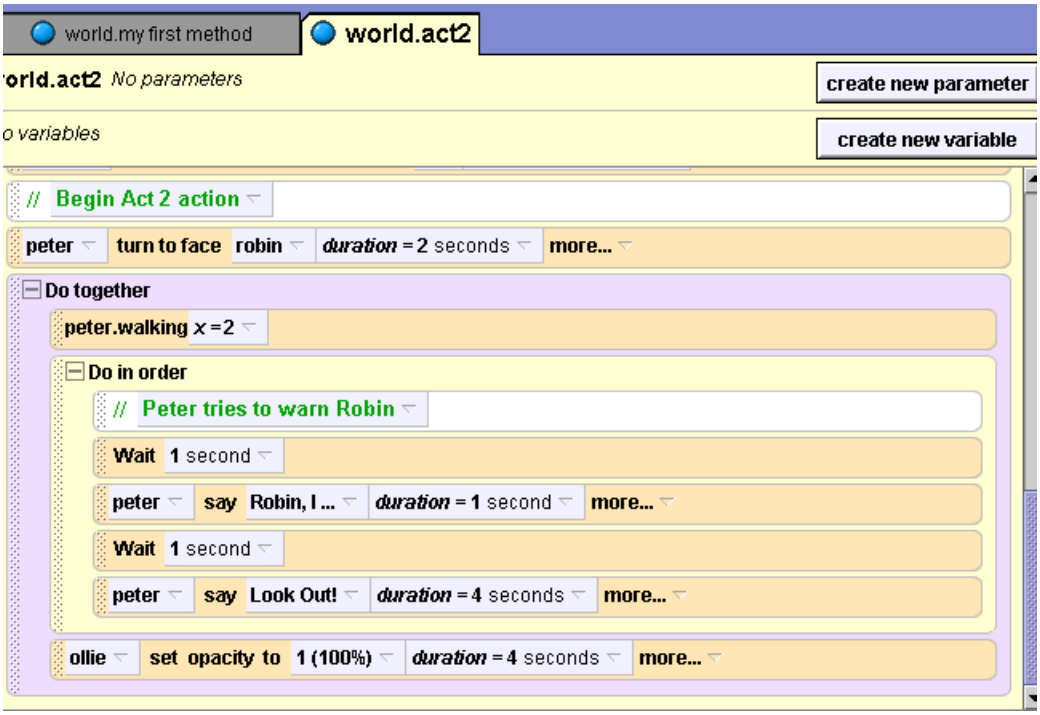


Figure 3-21 peter approaches robin, speaks, and ollie appears.

Ollie threatens Peter

Now for a little fire. We need **ollie's** mouth to open and flame to emerge. Dragons are objects made up of pieces (i.e., components) which are also objects. These components are capable of separate actions. If you select **ollie** in the **Object Tree** panel and click on the **+** sign, you expand **ollie** to show his parts. If you further expand **ollie's neck**, you find the **head**, and if you expand the **head** you find the **jaw**. In other words, the **jaw** is attached to the **head**, which is attached to the **neck**, which is attached to the **ollie**. See **Figure 3-22**. Note that the **jaw** object has properties, methods and functions of its own as can be seen in its **Details** panel.

To demonstrate this concept, select **ollie** in the **Object Tree** panel. Temporarily set **ollie's** *opacity* to 100% by selecting the *opacity* property in **ollie's Details** panel and selecting 1 (100%) so he is visible. Then, right-click on **ollie.neck.head** in the **Object Tree** panel and in the drop-down list, select methods **ollie.neck.head** *turn/right/2 revolutions.*

Experiment with other methods to see how you can move parts of an object individually. Don't forget to put things back together when you are finished. Undo can help.

Begin creating instructions to have **ollie** breathe fire by dragging a new Do together control to the end of the *act2* method. Then, select the methods tab for the **ollie.neck.head.jaw** (**Figure 3-22**) and drag the *turn forward* method into the new Do together block. If you experiment a bit, you will find that turning the jaw forward about 0.1 revolutions you will get a nice, threatening open jaw. See **Figure 3-23**.

If you go the wrong way, or go too far, odd but predictable things happen. You can also experiment with the methods directly in the ADD OBJECTS view and use "Undo" to get things back to where you started. It helps when experimenting like this to have **ollie** visible which you just did by adjusting **ollie's** *opacity* property.

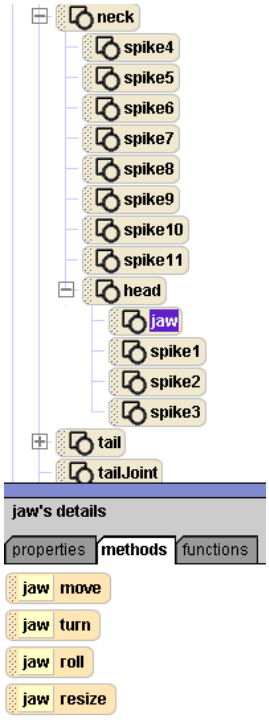


Figure 3-22 ollie's jaw methods.

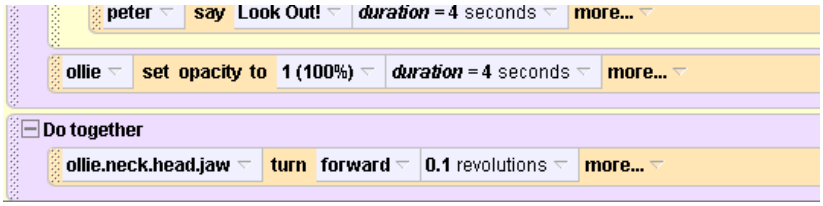


Figure 3-23 New Do together control with ollie's jaw instruction.

Authors' version of the file to this point is **AWB3-6.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Finding fire

Now we need to find some fire, add it to our world, and connect it to **ollie's** jaw. The Alice Web Gallery (NOT the Local Gallery) has a Special Effects category with a FireAnim class that looks interesting. To find it, click ADD OBJECTS which opens the Gallery screen. You are going to need an internet connection for this.

Then click on the up-arrow to the right of Local Gallery. click on the Web Gallery folder.

Scroll to the right in the categories until you get to Special Effects. Open the Special Effects category and find the Class FireAnim. Click on the FireAnim class and Add an instance to the world. Click the DONE button. Notice that a new event has been added automatically to the Events panel. That is because *flame_flicker* is a method that **FireAnim** owns and wants started when the **world**

Tech Talk – Reminder. Setting point of view does two things: it moves to the position of an object, and also takes on the orientation of that object (the way it is facing).

Tech Talk – Vehicle is a powerful property for an object. Vehicle property ties an object to another host object so it moves with the host. E.g., something in a hand wants to have the hand as the vehicle so it moves along with the hand. We will use vehicle later in this Chapter and again in Chapter 4.

starts.

Then, in the **Object Tree** panel, right-click on the **FireAnim** object and use the *set point of view to* method to set its point of view to **ollie's** jaw. To get to the **jaw**, you will have to click through: **ollie/neck/head**. You will see the flame move to a spot above **robin** which is where the invisible **ollie** is standing. You could make **ollie** temporarily visible by setting opacity to .5. Do this by right-clicking on **ollie** in the **Object Tree** panel and selecting the appropriate method.

Note that this is not quite the right positioning, and there is too much fire for starters. In the **Object Tree** panel, right-click **FireAnim** and select the method to resize it by one-half. Do this twice and you have a more manageable starting size for the flame.

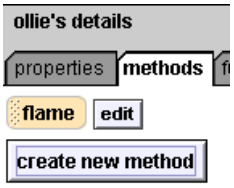
Next, turn the flame so it is more in parallel to **ollie's** mouth by right-clicking on the **FireAnim** object in the **Object Tree** panel and selecting methods/*turn/backward/.2 revolutions*. Right-click again to move it down one-half meter by selecting methods/*move/down/ ½ meter*. Note that the "down" direction is from the **FireAnim** object's point of view. The result is that the flame should move slightly ahead of **ollie's** face. Again, you might want to experiment with different settings to see what happens.

In the **Object Tree** panel, right-click on **FireAnim** and select the methods/*set vehicle to/ollie/neck/head/jaw*. Now the fire will stay with the **jaw** as things move. Also, methods/*set opacity to 0 (invisible)* so the fire will not display until we want it to.

Authors' version of the file to this point is **AWB3-7.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/. [Note that **ollie** is 50% visible in this file. Later, we will need **ollie** to be invisible again.]

Ollie breathes fire at Peter

We want **ollie** to be able to "breathe fire" more than once. This is a good reason to create a new method for our dragon. Click **ollie** in the **Object Tree** panel and then click the methods tab in **ollie's Details** panel. Click the create new method button and create a new method called "*flame.*" This is a user created method and as such will appear above the create new method button in the methods tab of the **ollie's Details** panel as shown here:



Be sure that the **world**.*act2* tab is active. Drag to the Clipboard found at the upper right corner of the screen, the last Do together control we started that contains the **ollie** jaw opening (i.e., **jaw** *turn forward* .1 *revolutions*.) instruction. Then, switch back to the **flame** method tab and drag the contents of the Clipboard there. {**Video demo**: Video 3-5.}

Note: if you drag a Clipboard with user created methods, including things like the penguin walk method, back into the **Method Editor** panel, Alice 2.0 shows an error. We have structured our examples to avoid this error, but you may encounter it as you try more sophisticated actions. Anything appearing above the create new method button in an object's **Details** panel is not permitted to be copied back into the **Method Editor** panel from the Alice clipboard.

Add to our *flame* method instructions for the **FireAnim** object. Set *opacity* to 1 (from **Details** panel properties tab). *move away from* (from the methods tab) **ollie's** jaw by .25 meters. You may need to click the down-

arrow to the right of meters and then click *other* to use the keypad to enter .25. Resize up by two (from the methods tab). The instructions should look like **Figure 3-24.**

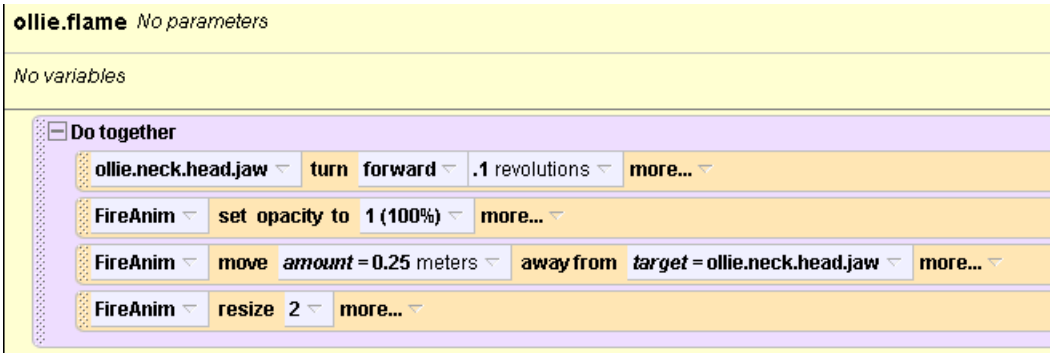


Figure 3-24 Initial flame method.

Be sure that the *act2* method tab is active. Delete the last Do together control by dragging it to the Trash can. Click on **ollie** in the **Object Tree** panel. Drag and drop our new **ollie**.*flame* method into the end of *act2*. {**Video demo**: Video 3-6}

Play the world and see what happens. **ollie's** mouth should open and the flame should appear.

Authors' version of the file to this point is **AWB3-8.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Of course the fire should not last forever. Make the *flame* method active in the **Method Editor** panel and copy its Do together control to the Clipboard. Then, paste it back into the *flame* method and reverse each of the directions involved: jaw turn backward, *opacity* back to 0, move .25 meters toward the jaw, and resize the fire by one half. See **Figure 3-25**. {**Video demo**: Video 3-7}

Try Playing the world again. Notice that **ollie** breathes fire only once.

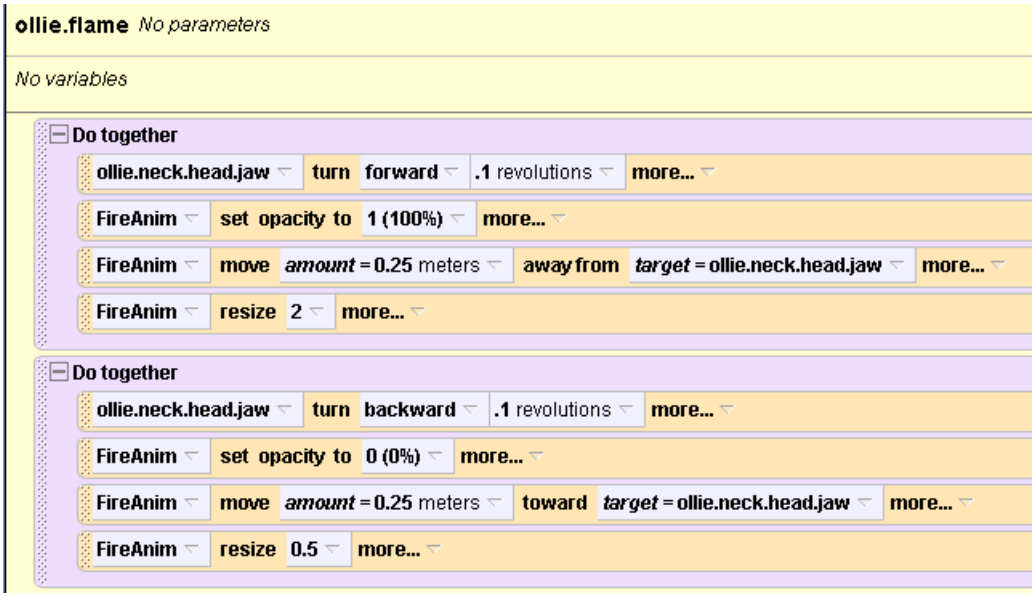


Figure 3-25 Flame appears and disappears.

Let's also revise the *act2* method to move **robin** from between **ollie** and **peter** – out of the line of fire so to speak. Click on the *act2* tab. At the bottom of *act2* and just under the **ollie**.*flame* instruction, drag in a Do together control. In the Do together control block, place an instruction that tells **robin** to move out of the way – perhaps a couple of meters to her left. Then, have **ollie** turn to face **peter**. Finally, tell **ollie** *flame* again but put the instruction outside of the Do together control. The end of *act2* should now look like **Figure 3-26**.

Play the world again to make sure everything works.

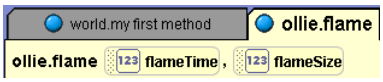
Figure 3-26 In the act2 method, robin moves and ollie faces peter.

Authors' version of the file to this point is **AWB3-9.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Director needs to control the size and speed of Ollie's flame

If we are going to have **peter** and **ollie** resolve their differences, we need to make sure that **ollie's** flame can match the task. As the director, you want to have control over the flame shot forth by **ollie** (i.e., the time it is displayed and its size). This can be accomplished by adding "parameters" to our flame method. For re-usable methods, having the right parameters gives the director useful controls.

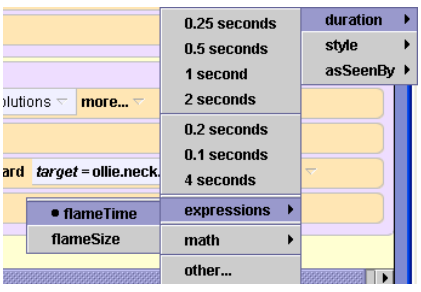
Select **ollie** from the **Object Tree** panel, and then select the methods tab in the **ollie's Details** panel. Edit the *flame* method. In the **Method Editor** panel, click the create new parameter button and add a parameter called "*flameTime.*" Make sure the Number radio button is selected. Click OK. Add a second parameter called "*flameSize.*" Make sure the Number radio button is selected. Click OK. You will see these new parameters at the top of the *flame* method in the **Method Editor** panel as shown in the picture below.



These parameters will allow you to indicate how long the flame should last and how large it should get when you request the *flame* method in *act2* (or from other uses of our dragon).

Increase flame size. Make the *flame* method active. Focus on the top Do together control in the *flame* method. Drag and drop the *flameSize* parameter tile (it looks like this 

) down to the "amount" value in the **FireAnim** move instruction. This changes the move amount to the *flameSize* parameter value (which has not been set yet). See **Figure 3-28**. [We are skipping by **Figure 3-27** for a bit.]



In that same instruction, click on more…. Select *duration*/expressions and choose *flameTime* as shown in the picture to the right.

Then drag the *flameSize* parameter tile to the 2 part of the **FireAnim** *resize* instruction. The flameSize parameter will now serve as the resize factor. It will be set to a value later on.

Also set the *duration* value of all instructions to the *flameTime* value. See the top Do together control in **Figure 3-28**.

Note that what we are doing to our **FireAnim** object is placing parameters in the *flame* method for that object. These parameters specify the *flameTime duration* parameter and the *flameSize amount* parameter. Set the parameters once to a value and their value carries through the method wherever the parameter is present in an instruction. That's the utility and efficiency of parameters.

Reduce flame size. Now focus on the bottom Do together control in the *flame* method. It is the part of the method that has the flame disappear and the dragon's jaw close. Leave the resize instruction alone for now. For the other instructions, drag and drop the parameters in as you did in the top Do together control. See **Figure 3-28**.

We will now need to make a special adjustment to the resize instruction in this part of the *flame* method. Recall that in the top Do together control part of the method, we increased the size of the flame by the amount of the *flameSize* parameter. In the second part we will want to reduce the flame to its original size. We can do this by using the reciprocal of *flameSize*.

Here is how reciprocal works. Think of the amount of money you may have on your person right now. Say it is $10. If a magician wanted to double it to $20, she would wave her magic wand and multiply the original $10 by the number 2. Now you have $20 on your person. But then if the magician wanted to reduce the $20 back to the original $10, she would wave her magic wand and multiply your $20 by the reciprocal of 2. The reciprocal of 2 is 1 divided by 2 (i.e., ½). We can see that ½ times $20 = $10. Bad magician!

Back to **ollie's** *flame* method. If a *flameSize* of 2 doubles the size of the flame, then to reduce the flame to its original size we would use the reciprocal of 2 which is ½, to resize the flame. Similarly if we triple the size of the flame, we must then reduce it to 1/3 of its increased size.

We can do this by first setting the resize parameter in the bottom Do together control of the method to 1. Then, click on the pull-down menu for the resize parameter and select math, then select "1/". Then select expressions, and then select *flameSize* as shown in **Figure 3-27**. The entire revised flame method is shown in **Figure 3-28**. {**Video demo**: Video 3-8}

Now give it all a try. Click Play to see the results. Things still work fine. Save your world.

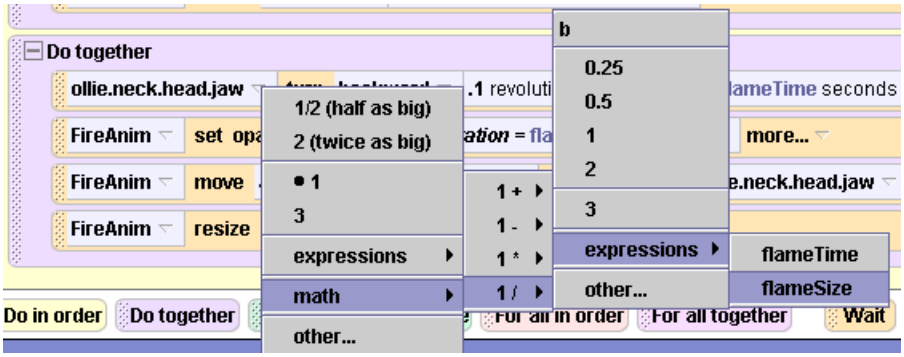


Figure 3-27 Using reciprocal of flameSize to reduce the size of the flame.

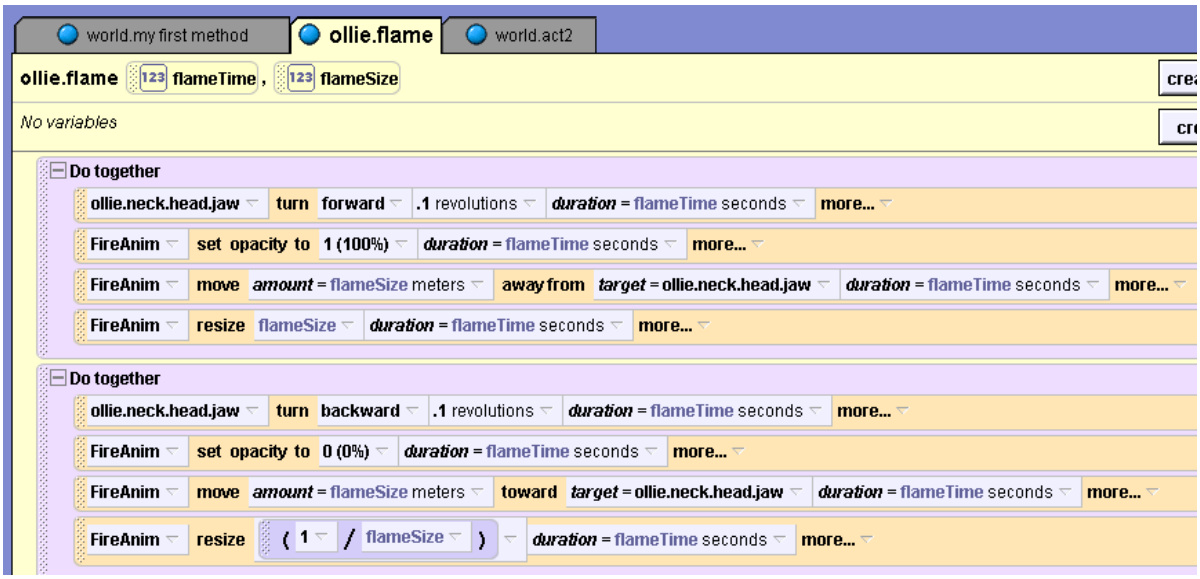


Figure 3-28. Completed flame method with flameTime and flameSize parameters.

Authors' version of the file to this point is **AWB3-10.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

This might be a good time to recap our work so far.

The **world** *my first method* method runs *act1*, displays a 3D Text when *act1* ends, and then runs *act2*. The *act1* method handles the initial interaction between **robin** and **peter**. The *act2* method contains instructions to place our actors in their proper positions to begin the act, instructions to **peter** to move toward and speak to **robin**, and instructions to have **ollie** appear and confront **peter.**

Continuing on, if we look at the *act2* method in the **Method Editor** panel, we find that our **ollie**.*flame* instruction now has both parameters set to 1. We didn't do this - Alice automatically defaulted the values of these parameters to 1. See **Figure 3-29.**

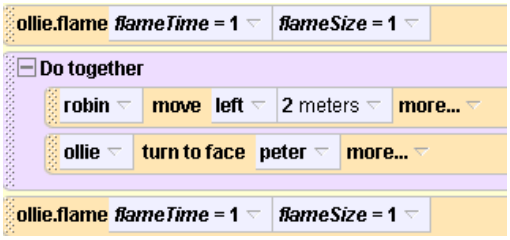


Figure 3-29 act2 calling the flame method with its parameters defaulted to 1.

Let's give that second shot of fire, the one aimed directly towards **peter**, a bit more punch. In the *act2* method, separately click on *flameTime* and *flameSize* as shown in **Figure 3-29** and set the parameters *flameTime* to 3 and *flameSize* to 4. Try other values for the parameters if you like. Click Play to see the results. It's good to experiment. Remember that Undo is your friend!

If you try other values, note that it can be useful to speed-up the play sequence with the slider at the top of the play window. Also, when you disable blocks of code, even all of *act1*, you can focus on specific interactions in order to get the directions just right for those instructions. The trick of course is to have everything in place

and set up for the action. When experimenting with parameters for the *flame* method, you can do much just watching the dragon without other objects in their places. You can make the other objects temporarily invisible by right-clicking on them in the **Object Tree** panel and selecting their opacity methods. Don't forget to make them visible again when you are done experimenting.

Here is the power of parameters Suppose that you wanted the same flame size in both of the *flame* method calls in *act2* as in **Figure 3-29**. As you can see, right now that *flameSize*=1. Make the *flame* method active in the **Method Editor** panel. Take a look at the top where the parameter tiles are shown (see **Figure 3-28)**. Try this out: Drag and drop the *flameSize* parameter tile as the first instruction in the method. Set value/expressions/other/10. Now, whenever the *flame* method is called, all instructions that contain the *flameSize* parameter will use 10 as its value. Changing this value in just one place causes the change to propagate through the entire method. Ok to Undo now. Note that you can do the same thing with the *flameTime* parameter. By the way, parameters propagate their values only within the methods where they exist. {**Video demo**: Video 3-9}

Variables are good too!

By running tests with the parameter values mentioned above, we can see that the flame goes too far (it extends into the ground under **peter**). We need a way increase the flame size without moving it too far from **ollie**, particularly not beyond **peter**. We could add another parameter, or we could compute an appropriate distance that is based on the size of the flame. We know that .25 meters worked with a resize of 2, so let's try a distance that is one half of the flame size. There is more than one way to do this, but if we use a variable within the method we can refer to it later.

In the **ollie** *flame* method, select the create new variable button, make sure the Number radio button is selected and name the new variable "*distance.*" A variable entry will appear at the top of the method, initially set to 1. Drag this variable above the top Do together control and set it to the expression *flameSize*. You can see this in **Figure 3-30**.

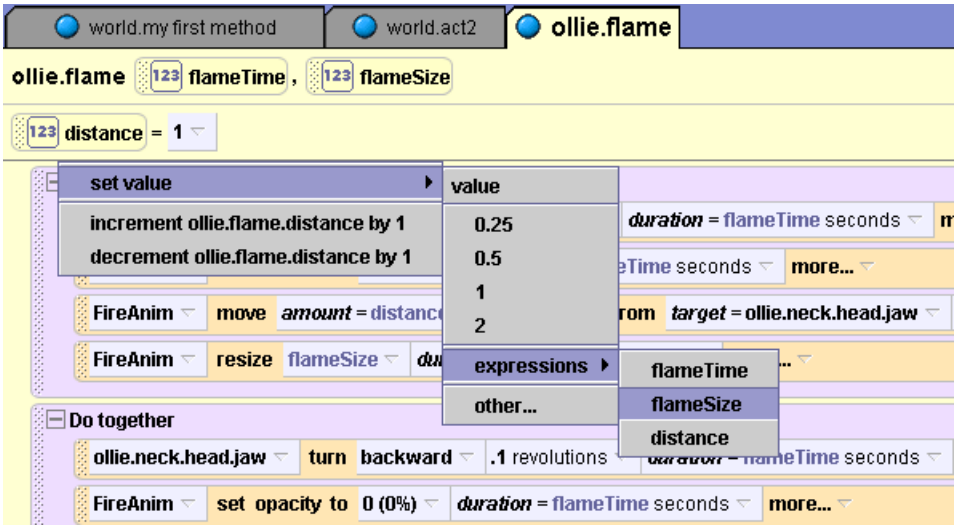
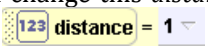


Figure 3-30 Setting the variable named distance to the parameter flameSize.

The resulting instruction is shown below.

Now click the down arrow you see in the *flameSize* parameter of the instruction shown above so you can set a value for distance. Then, select the math option with division "/" and select the value 2. Now you have *distance* set to the *flameSize* divided by 2 as shown below.

If we decide a different value for distance would work better, it is easy to change it in just one place. That's the beauty of and variables and parameters!

Now in the **FireAnim** *move* instruction in the *flame* method, select the *amount* parameter and change this from the *flameSize* to the expression *distance*. Don't forget to make the same change in the bottom Do together control part of the method - the part that makes the flame disappear. The entire *flame* method is shown in **Figure 3-31**. Click Play to give it a try. You can change this *distance* parameter value from 1 to some other value simply by changing it in one place: the 

tile at the top of the method.

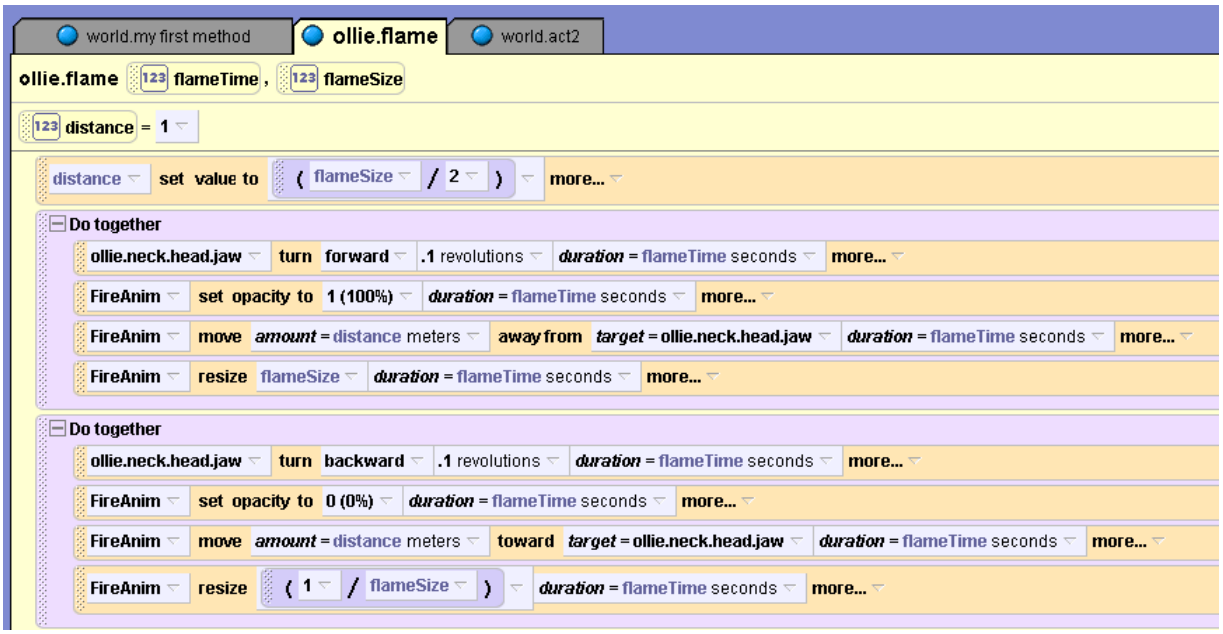


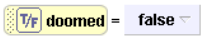
Figure 3-31 flame method with the distance variable added.

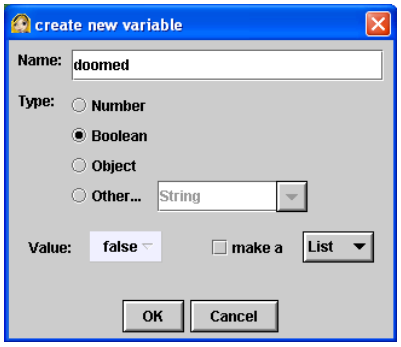
Authors' version of the file to this point is **AWB3-11.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

3.4: The audience gets to pick a winner: Ollie or Peter

Up to now, when we played *my first method*, we sat back and watched Act 2 unfold before our eyes. Of course, we created the instructions so we knew what was going to happen. But, how about letting the audience decide whether **peter** or **ollie** triumphs?!

Recall the story board. We have to decide whether **peter** or **ollie** will be victorious in the ensuing conflict. Well, actually we don't. Here is where a movie differs from the virtual world. We can engage the audience in the decision making. So let's ask the viewer if **peter** is doomed or if **robin** is his destiny. We will need the decision saved (i.e., make note of it) for reference in the rest of the story.

To have a spot to save this decision information and make it accessible everywhere, we need a variable for the **world** object. Select **world** from the **Object Tree** panel and then the properties tab in **world's** details. Note that we are creating a variable that will be owed by an object – not owed by a method as before. Then create new variable called "*doomed*" with a type Boolean. Booleans can take on only the value *true* or *false*. Select *false* as the value (we will hope for the best for **peter**). This new **world** *doomed* variable looks like: 



Be sure that *act2* is open in the **Method Editor** panel. Now, drag and drop our new *doomed* variable from the **world's** properties to the end of our instructions for *act2* setting the value to *true*. We started *doomed* with the value *false* and then changed it to what we wanted, *true*, to demonstrate that Booleans can only have a value of *true* or *false*.

In the **world's Details** panel, select the functions tab and drag and drop the "Ask user for yes or no" onto *true* located in the set value to part of the instruction we just added. In the drop-down list, select question/other/ and in the pop-up window enter the text "Is Peter Doomed?" If we decide "yes", that **peter** is doomed, the value of the *doomed* variable to be *true*. If we decide "No", that **peter** is not doomed, the value will be *false*. {**Video demo**: Video 3-10}

The end of *act2* should now look like **Figure 3-32**.

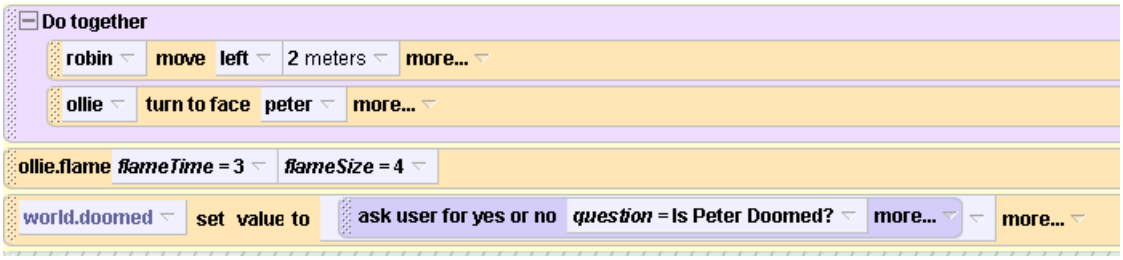


Figure 3-32 The viewer choice instruction in the act2 method.

Our story moves on. From the bottom of the screen, add an If/Else control to end of the *act2* method. Then drag and drop the **world.doomed** variable to the decision value part of that instruction (i.e., If part). Select true. See **Figure 3-33**. Drag the comment indicator () under both the If and the Else parts of this block. For each of these comments, add text indicating what the result will be under that option. You can choose your

own comments, but "Peter shall succumb" and "Peter shall prevail" are reasonable options. {**Video demo**: Video 3-11}

The If/Else instruction, with the comments, should look like **Figure 3-33**.

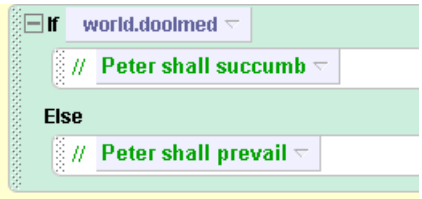


Figure 3-33 Decision time for the viewer.

Important: If the viewer selects "Yes" as an answer to the question "Is Peter Doomed?", then the value of the *doomed* variable will be *true* – making it such that the instructions we put in the "Peter shall succumb" block will be implemented. If the viewer selects "No," then the *doomed* variable's value will be *false* - making it such that the instructions we put in the "Peter shall prevail" block will be implemented. Note that the If/Else controls color is light blue. From now on, you will be placing most or the *act2* instructions inside this If/Else control so be sure the light blue border is around those instructions you enter.

Authors' version of the file to this point is **AWB3-12.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

3.4.1: Peter is toast

Important reminder: The instructions that follow (in Section 3.4.1) will be placed in the "Peter shall succumb" part of this If/Else control.

Let's focus on what happens if **peter** is doomed. You might remember from our storyboard that the certain actions were indicated. See **Figure 3-34.**

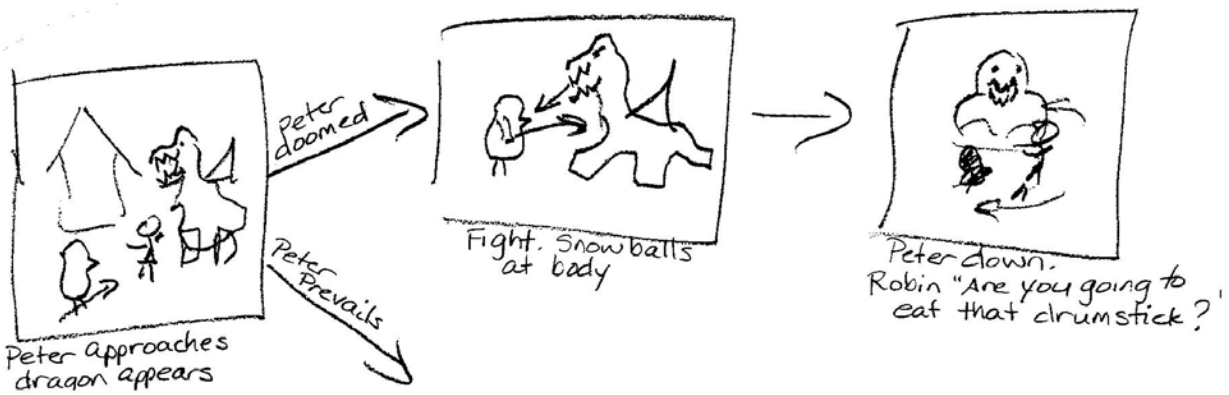


Figure 3-34 peter is doomed.

We will be looking at **Figure 3-35**. Drag and drop the Loop control up into "Peter shall succumb" block (right under the comment) and select 1 time from the drop-down list. Then, go to the **world's Details** panel and select the functions tab. Scroll down to find the **world** function *random* and drag random number into the Loop count (i.e., where you see 1 time). We do this is because we want to allow for a different kind of battle between **peter** and **ollie** each time you Play the world. Click the pull-down button following more… and set

the random number minimum to 2. Again, click the pull-down button following more… and set the maximum to 5. The instruction should look like **Figure 3-35**. This will give our story some variation as folks try it. Play your world. You will see as far as the user's decision point.

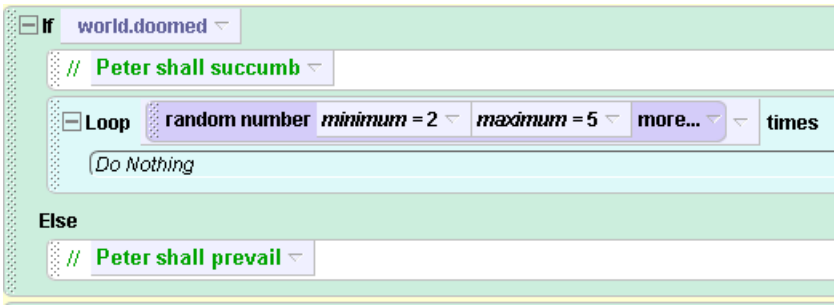


Figure 3-35 A loop with randomness to introduce variability into the story.

Since **ollie** got the first shot, let's give **peter** a response. Given that he is a penguin, and that dragons breathe fire, it seems a snowball would be a good weapon for **peter**. We can create a *tossSnowBall* method for the **peter** object. Select **peter** in the **Object Tree** panel. Click on **peter's** methods tab and create a new method named *tossSnowBall*. Add a parameter to this method called *target* and select type Object for this parameter.

We will need a snowball to toss. Go to the ADD OBJECTS view. Click the yellow up arrow until you see the Local Gallery. Find the Shapes category, and pick the Class Sphere. (We could use one from the Sports Class, but that ball has seams.) Add instance to world. Rename the sphere to **snowball**, and move it to **peter's** right wing. You may need to use the camera controls to keep the **snowball** in the **Word View** panel.

We need to resize the current snow ball. You can accomplish the following by right-clicking on **snowball** in the **Object Tree** panel. Since **snowball** is currently a bit big for **peter**, let's resize it down – cut it in half three times and we get a nice size. Then we need to make it transparent until we are ready to use it, so alter the snowball *opacity* property to 0. Finally set the **snowball** *vehicle* property to **peter/**the entire peter, so it will travel with him as he moves. Click DONE.

Now back in **peter's** *tossSnowBall* method, start by dragging the **snowball's** *opacity* property into the method and set it to 100%. Also set the *duration* for this to 0 as **peter** doesn't have time to be slow about this. Next, drag in the **snowball** *move to* method into the *tossSnowBall* method and indicate expression/*target*. Set the duration to 2. This will allow our viewers to see that the **snowball** is visible as it moves from **peter** to its target. Finally, convert the **snowball** *opacity* back to 0 so that after the **snowball** hits its target the **snowball** disappears. Now, move the **snowball** back to **peter.rightWing**, Set the *duration* of each of these last two instructions to 0. This will set the **snowball** up for future use (i.e., another toss). The *tossSnowBall* method should look like **Figure 3-36**.

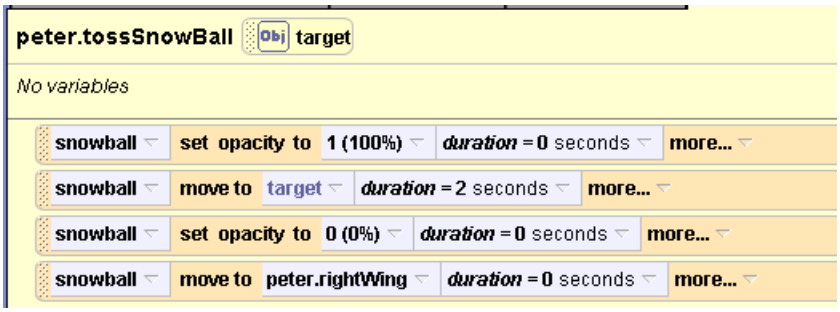


Figure 3-36 Instructions for peter to throw snowball at some to be determined target.

Back in the Peter shall succumb part of the *act2* method, direct **peter** to *tossSnowBall* at the entire **ollie**. Then sustain the **ollie**.*flame* for a *duration* of 2 and *size* of 5. The instructions should look like **Figure 3-37**. Try this out by Playing the world and answering Yes to the question, "Is Peter doomed." Notice that the random values will give a slightly different number of tosses each time. You will see either 2, 3, or 4 snowballs thrown. This is because the random numbers produced by Alice do not include the stated maximum limit (5, in our case).

During Play, you might want to speed things up by using the slider in the upper left corner of the World Running… window. It looks like:

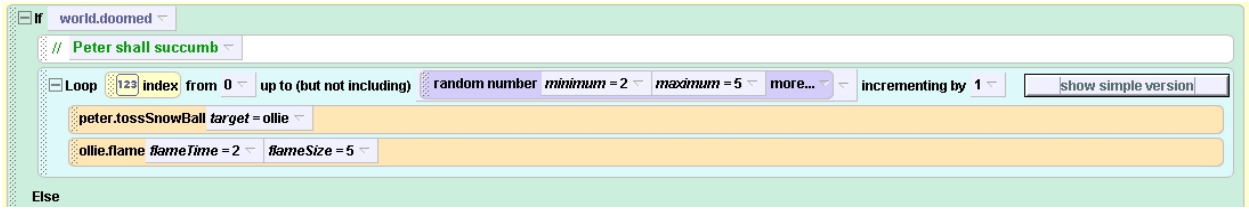
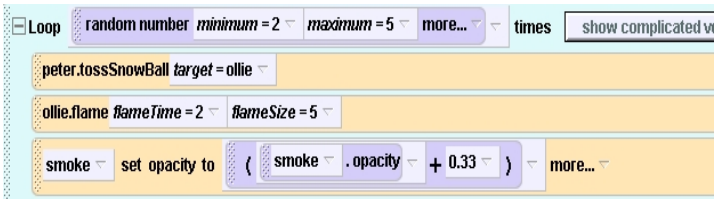


Figure 3-37 Loop control with 'complicated' version shown.

Authors' version of the file to this point is **AWB3-13.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Let's see if we can make the effects of the *flame* method more dramatic. Return to the ADD OBJECTS view and, in the Web Gallery, find the Special Effects category. Add an instance of the Class Smoke into our story and move it close to **peter**. In the **Object Tree** panel, select **smoke**. Right-click on it and select methods. As with the **snowball**, shrink (resize) **smoke** in half twice, make the vehicle **peter/**the entire peter and set the *opacity* to 0. Click DONE to exit the view.

We want to increase the intensity of **smoke** each time through the loop. Back in the *act2* method, add changes to make the **smoke** appear in our loop. Drag and drop the **smoke** *opacity* property into the bottom of the Loop control with *opacity* of 100%. Now drag *opacity* in again and drop it in the 100% spot in the instruction you just created. It will be used as a parameter for the value to set *opacity*. Click on this *opacity* parameter and the math option to add (+) .33 as shown below. {**Video demo**: Video 3-12}



Now **smoke** will become more visible with each Loop iteration because we reset its *opacity* each time.

Tech Talk – The concept of using the current value of a property as an argument to change that property is very powerful. It is implicit in the resize method for Alice, but here you are doing it directly.

Also add a resize of **smoke** inside the Loop to increase by **1.3** each time. 

Finally, add a Do together control into the Loop as shown in **Figure 3-38** and move the **ollie**.*flame* and **smoke** change instructions into the block for that control. Also set the *duration* on the **smoke** opacity changes to 5 and the *flame* method will make **peter** smoke. Play it. If the **smoke** isn't visible, add an instruction telling **smoke** to *turn to face* the **camera**. Your instructions should look like **Figure 3-38.**

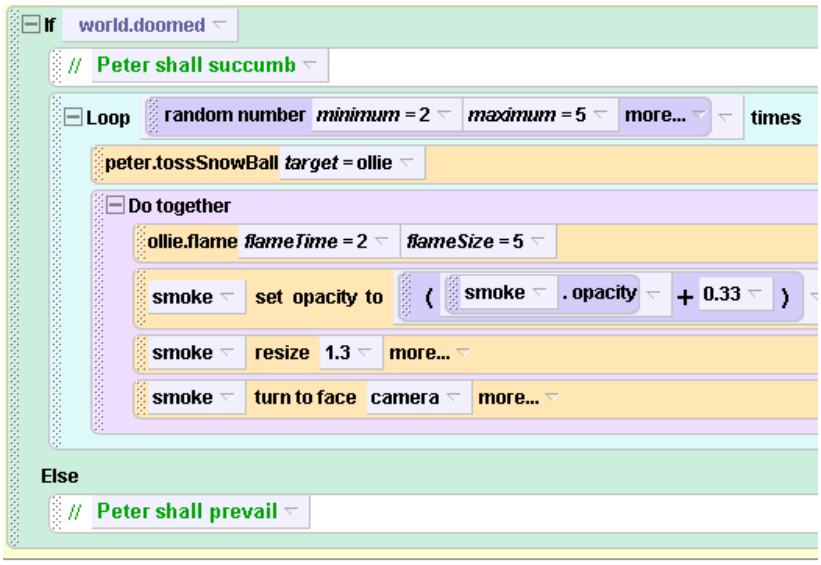


Figure 3-38 Peter gets the worst of the battle.

Flame reach: Suppose **ollie's** flame barely reaches **peter**. You can increase the distance variable in **ollie's** *flame* method from 1 to something else. And/or, you can simply move **ollie** closer to **peter** using your mouse cursor, prior to Play. That position will hold for subsequent Plays.

Morte de Peter

The last step is "morte de Peter." We are still working in the Peter shall succumb part of the *act2* method.

Under the Loop control (NOT inside of it), add a Do together control. In this, set **peter's** *color* property to black. In **peter's** method tab, drag the instruction **peter** *roll/ right/ ¼ revolution* into the Do together control. Drag and drop **peter's** *resize* method in order to resize him by ½. Finally, after the Do together control, drag in the **smoke** *opacity* property and set it to 0. These instructions will make **peter** shrink and disappear. These instructions should look like **Figure 3-39.** Save your world and click Play to see the result.

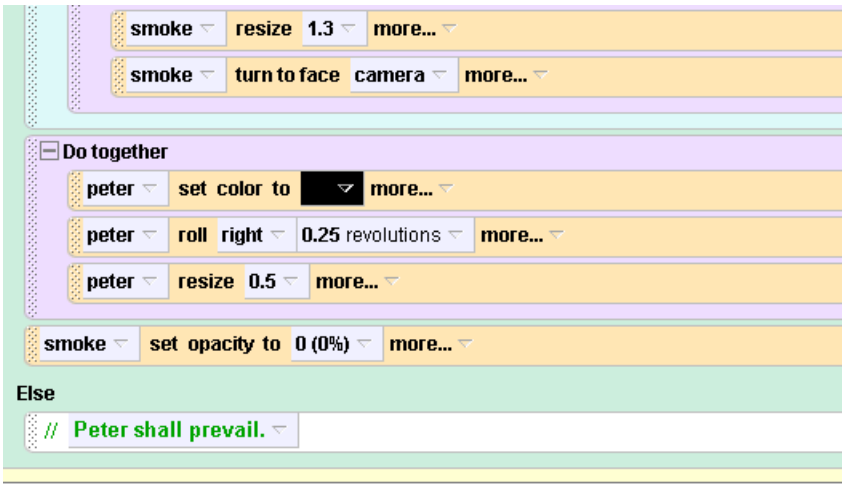


Figure 3-39 Poor Peter.

Authors' version of the file to this point is **AWB3-14.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/. [The *distance* variable in **ollie's** *flame* method is now 4. Also, **ollie** starts off closer to **peter**.]

Peter is gone, but the others live

This is just about it for **peter**, but **ollie** and **robin** need to complete their parts in *act2*. There is a concept that a story should have an inevitable but surprising end, so we shall try for that.

First we need to target positions for **ollie** and **robin** at the end of this scenario – after **peter** has succumbed. We have the cones, and these provide a good tool for that purpose. Go to the start of the "Peter shall succumb" If/Else control in the *act2* method. For convenience purposes, add a Do in order control right under the "Peter shall succumb" comment and above the Loop control to contain our positioning instructions. **Figure 3-40** shows the proper placement. With *duration* of zero for each, add the following instructions into the Do in order control:

- move **peterCone** to **peter** [use the *move to* method]
- move **robinCone** to **peter** [use the *move to* method]
- move the **peterCone** toward the **farmHouse** two meters [use the *move toward* method]
- move **peterCone** up by 0.5 meters [use the *move* method]
- move the **robinCone** toward **robin** one meter [use the *move toward* method]

This "marks the stage" so to speak, for our final positions. The newly added instructions should look like **Figure 3-40.**

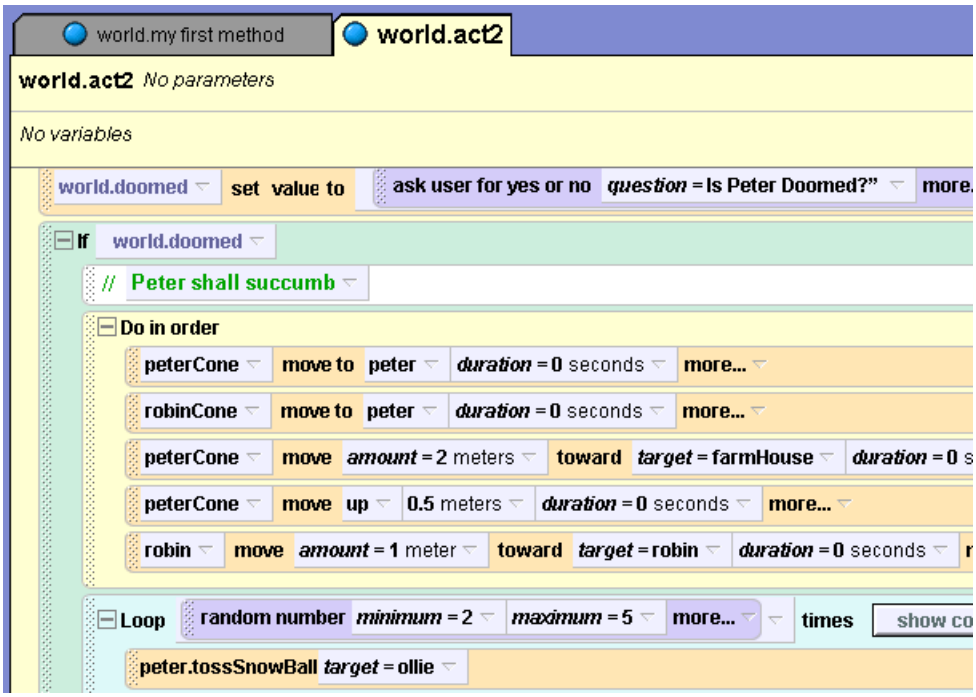


Figure 3-40 Target positions for the survivors.

To see how this all plays out, you might want to temporarily set the *opacity* of **peterCone** and **robinCone** to 100%, change the *duration* of each of the instructions that you just added to 1 second, and Play the world.

The final position of everything should look something like **Figure 3-41**. Now you can *reset opacity* of both cones to 0 (i.e., invisible).

Figure 3-41 Final positioning in the peter succumbs scenario.

We are now going to be referring to **Figure 3-42**. Add a final Do together control at the end of the Peter shall succumb block (right above the Else). And add the following instructions into it, each with a *duration* of 1 second: **ollie** *move to* **peterCone**, **ollie** *turn to face* **camera**; and **robin** *move to* **robinCone**.

As an added touch, add these instructions: **ollie.frontRightLeg.lowerLeg** *roll right ¼ turn*; and the **front.LeftLeg.lowerLeg** *roll left ¼ turn*. The effect is that **ollie** sits, contented with his work and his prize, **robin**, there by his side.

Now for a final spin. Add **ollie** saying "Well Done, I think" and finish with a **robin** saying "Did you want a drumstick?" with a *duration* of 6 seconds for this closing comment. Put them in a Do in order control.

These additional instructions are shown in **Figure 3-42.**

Authors' version of the file to this point is **AWB3-15.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

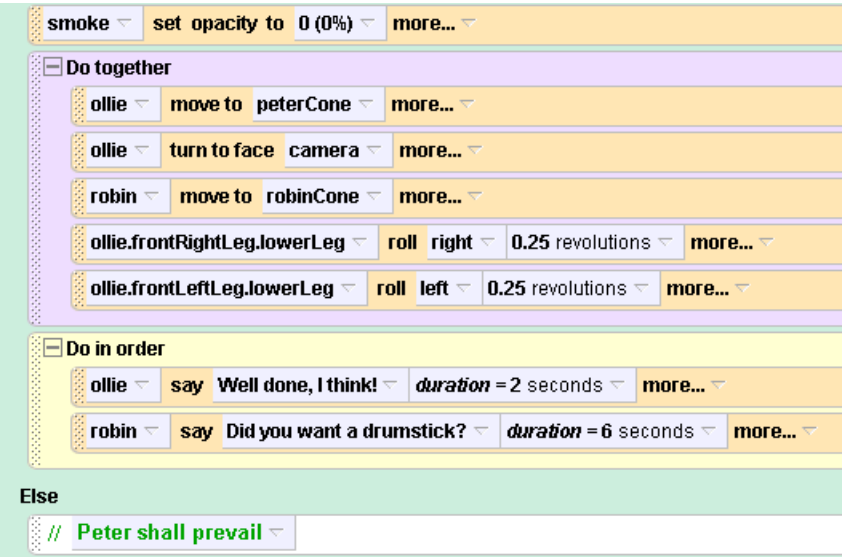


Figure 3-42 robin and ollie, side-by-side.

3.4.2: Peter prevails

The Society for the Protection of Penguins didn't like the previous ending. We must also consider the "Peter shall prevail" sequence as shown in our storyboard and repeated in **Figure 3-43.**

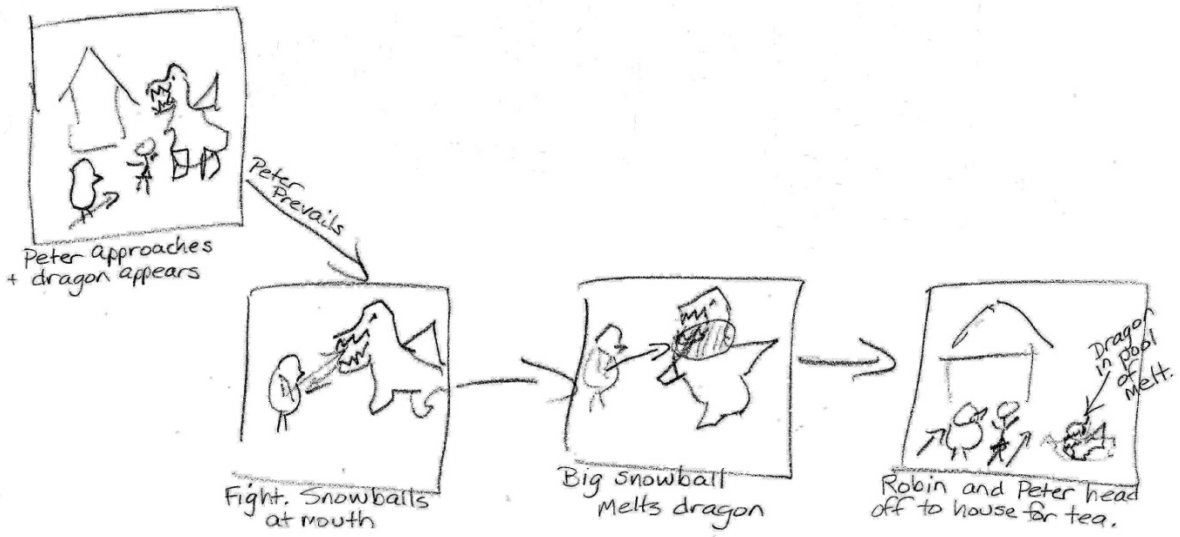


Figure 3-43 Story board for peter prevails.

This scenario handles the actions to be taken when the user's response to the "Is Peter doomed" question is "No." This will send control to the Else part of the If/Else control in the *act2* method which is the "Peter shall prevail" section. Refer **Figure 3-33** (repeated below) to see what we mean.

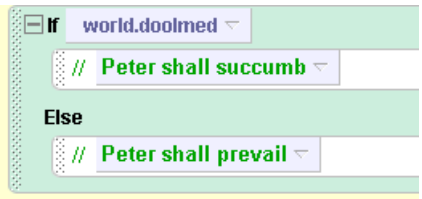


Figure 3-33 (repeated) Decision time for the viewer.

Important: From here to the end of the chapter, unless we say otherwise, we will be working in the "Peter shall prevail" Else part of this If/Else control. Be sure all instructions and controls are placed in that block. Remember that the light blue color identifies the border of the If/Else control

We can reuse some our work from the "Peter shall succumb" Loop control, shown in **Figure 3-38**, in the "Peter shall prevail" part of the If/Else decision control we added earlier. Unfortunately, a bug in Alice 2.0 prevents copying user defined methods to the Clipboard and then putting them in an additional location. If you remove the **peter** *toss* and **ollie** *flame* instructions out of the Loop control shown in **Figure 3-38**, you can drag the rest of the loop to the Clipboard. Then drag the Clipboard into the "Peter shall prevail" block. Move the two instructions back into the "Peter shall succumb" Do together control. {**Video demo**: Video 3-13}

Then, re-enter the **peter** *toss* and **ollie** *flame* instructions from scratch (no copy permitted) in the prevails part of the If/Else control. In the "Peter shall prevail" block, change the target for the **snowball** *toss* to the **ollie.neck.head.jaw.** Also, change the battle scenario so that **peter's** *tossSnowBall* throw and **ollie's** *flame* happen at the same time.

The instructions should look like **Figure 3-44**. Now the snowballs go right into the **ollie's** mouth. That could make the difference for **peter's** survival.

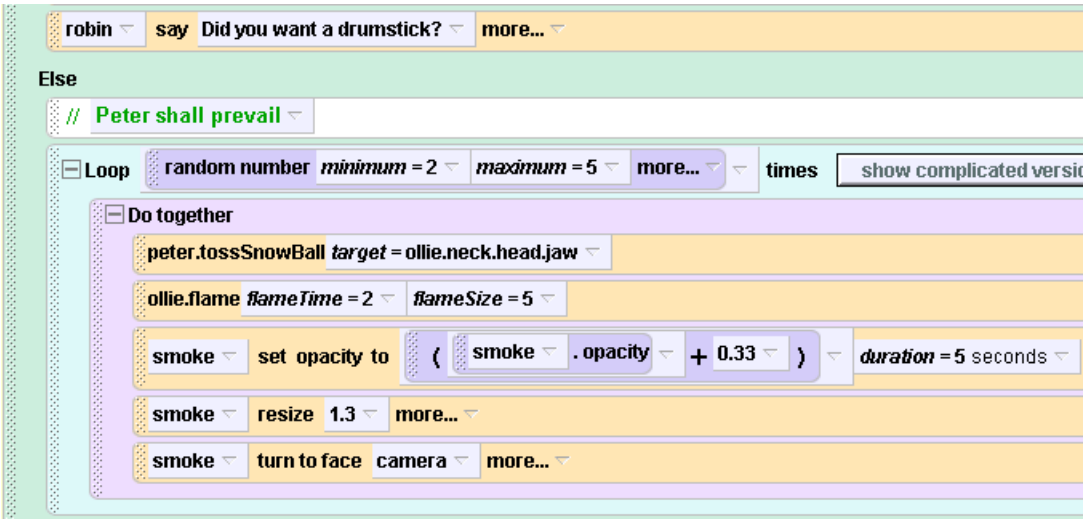


Figure 3-44 Battle action for peter shall prevail.

To dispose of his pesky dragon adversary, **peter** tosses a super large snowball at the dragon's head. This is the knockout and the last toss! Fortunately, we designed the *tossSnowBall* method with a parameter that lets us select the *target*.

We will create the instructions shown in **Figure 3-45**. Add a Do together control with **peter**.*tossSnowBall* with the target **ollie.neck.head**, along with a **snowball** *resize* 2. If you also select the more option on the *resize*, you can have *style* = *begin gently* which causes **snowball** to grow gradually. Add the comment.

The *tossSnowBall* method returned the **snowball** to **peter** and made it invisible. At this point we need to leave the **snowball** visible and at **ollie's** head. To make this adjustment, add three additional instructions, each with a *duration* of 0: **snowball** *move to* **ollie.neck.head**, **snowball** *set vehicle* to **ollie.neck.head**, and **snowball** set *opacity* to 100%. You will find the last two in **snowball** properties. Place all three instructions outside of the Do together control. The instructions are shown in **Figure 3-45.** Play and take the "No" option.

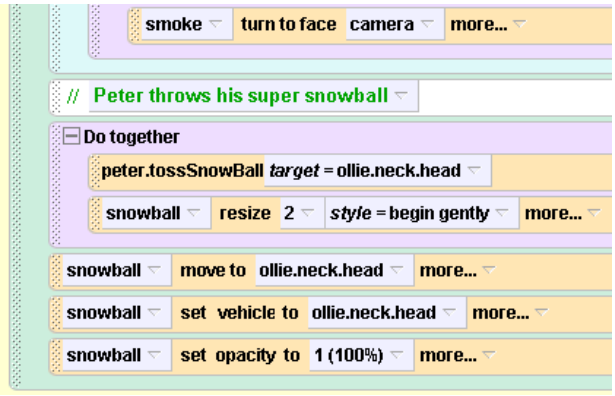


Figure 3-45 peter has a secret weapon.

Death scene for Ollie

Our death scene for **ollie** will have him shrink, the result of his fire being extinguished by **peter's** super **snowball**. Add a Do together control with 5 seconds of **ollie** thinking "Oh No, you wicked penguin, I am melting" (or other dragon's final thoughts of your choice) and a Loop control for 5 iterations. Inside the Loop control use a Do together control to have **ollie** resize by one half, resize the **snowball** by 1.1, and have **ollie** move down 0.2 meters.

Also, set the *opacity* of the **snowball** to 1. As we did in a previous part of this chapter, drag the **snowball** *opacity* property over the 1. Then click on the pull-down button to the right of **snowball**.*opacity*, select math, and then **snowball**.*opacity* ***** and set the value to 0.9 (***** is the multiplication operator). The result is that the *opacity* of the **snowball** diminishes by ten percent on each Loop control iteration. **Figure 3-46** shows the completed instructions. **Figure 3-47** shows the result of the animation.

Authors' version of the file to this point is **AWB3-16.a2w** and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

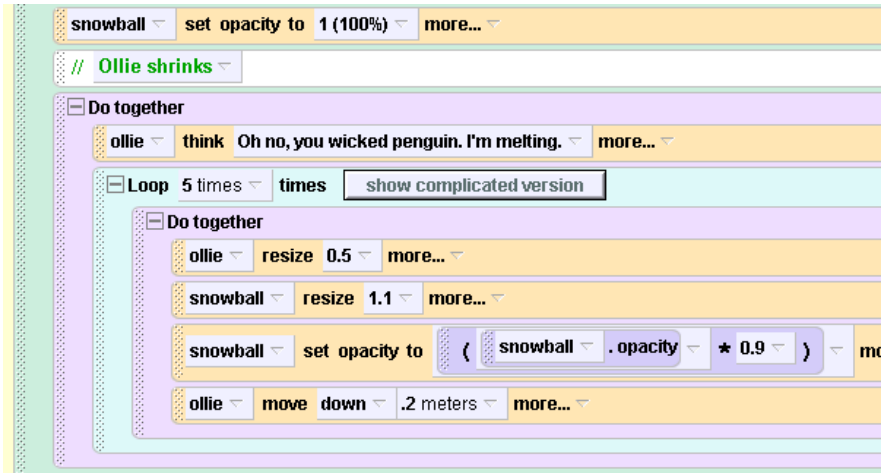


Figure 3-46 Bye-bye ollie!

Figure 3-47 The dragon is slain.

Ollie is gone, but what about Robin?

Now we need **robin** to rejoin our story. Recall that **robin** moved to her left just after **ollie** materialized and breathed fire. We need to save the location she was in prior to her move. Focus your attention on the spot in the *act2* method just below the first **ollie**.*flame* instruction, and just above the Do together control where **robin** moved away. Insert an instruction to have the **robinCone** move to **robin** with a *duration* of 0 seconds. The instruction is shown below. See **Figure 3-48** to see where the instruction is placed.

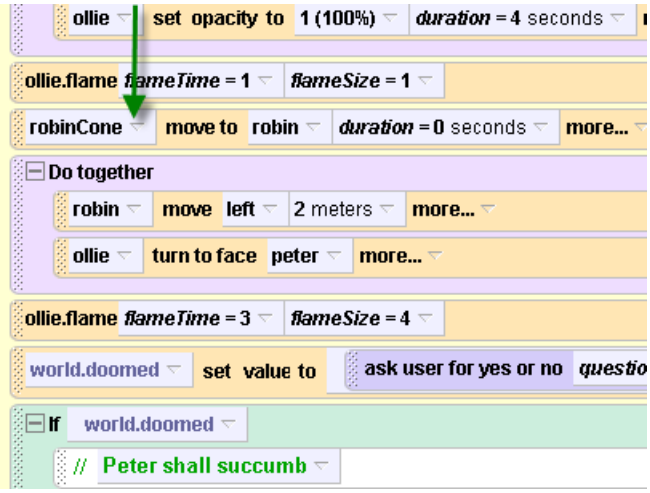


Figure 3-48 Saving robin's position with robinCone.

Focus on the end of the "Peter shall prevail" Else block in the *act2* method. Add a Do together control. Make sure you place this Do together control inside the "Peter shall prevail" block. In this Do together, add instructions to have **robin** *move to* **robinCone**, and **robin** *turn to face* **peter**. Also have the **smoke** *opacity* go to 0. Follow this Do together with **robin** saying "My Hero" (or some other admiring remark). The new instructions are shown in **Figure 3-49.**

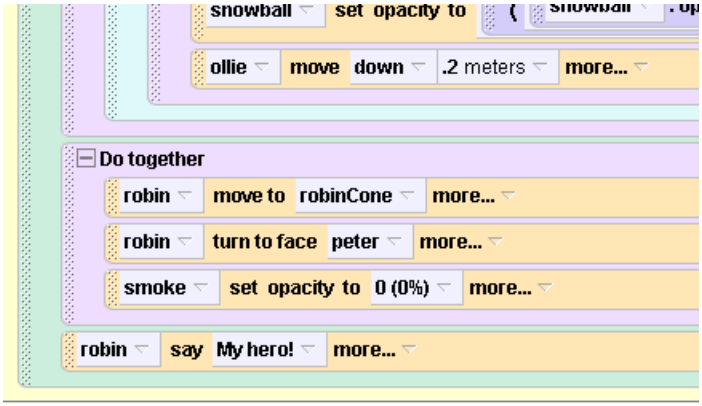


Figure 3-49 robin praising her hero.

Authors' version of the file to this point is AWB3-17.a2w and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Time for the hero and heroine to exit

For the hero and heroine exit, add a Do together control (we are still in the "Peter shall prevail" block) and place instructions in the control for both **robin** and **peter** to turn to face the **farmHouse**. Then, for **robin**, set vehicle to **peter**. This will have them move together. Add another Do together control with **peter**.*walk* with a time of 8 seconds, **robin** saying "Would you like a cup of tea?" for 8 seconds, and **peter** *move toward* the **farmHouse** for 8 meters and 8 seconds. You can see the instructions in **Figure 3-50**.

It turns out the **smoke** is something we can use to put an interesting end to our story. Still at the end of the "Peter shall prevail" block, have the **smoke** (which also has **peter** as a vehicle) move one-half meter closer to **robin**, resize it by 2, and have the **smoke** turn to face the **farmHouse** all in *duration*=0 seconds. For convenience, place these three instructions inside a Do together control. Follow this block with instructions

to set **robin's** and **peter's** *vehicle* properties to the **world** so they no longer move with **peter** with *duration* of 0 seconds.

For the final close, add one last Do together control inside the "Peter shall prevail" block. Inside that Do together control, add a Do in order control with instructions to wait 1 second and then have **peter** *jump* 2 times. Back in the Do together control, have the **light** *brightness* go to 0 with *duration*=3 seconds, the **world** *ambientLightBrightness* go to zero with *duration*=3 seconds, and the **world** *atmosphereColor* go to black with a *duration* of 3 seconds. Also, set the **smoke** *opacity* to 70% with a *duration* of 4 seconds.

If you have been working on this lengthy program with the dragon visible it is time to make **ollie's** *opacity*=0. Right-click on **ollie** in the **Object Tree** panel and select the *opacity* method. Set to 0.

This final set of instructions is shown below in **Figure 3-50**. Play the world to see the results. {**Video demo**: Video 3-14)

Authors' version of the file to this point is AWB3-18.a2w and can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Tech Talk: Sharing and showing off your Alice worlds. There are several ways to do this.

(1) Send or give your a2w file to someone. These files tend to be rather large. But, whoever receives them can Play your world and inspect your program instructions if they have the free Alice software.

(2) Create an html file (i.e., web page) showing some or all of your world's methods. In Alice, just click on File and select "Export Code for Printing." Choose the methods you want to export. The resulting file will save as a web page that you can open in a browser to share and/or print. This isn't your Alice world so it will not Play, but at least you can show and share the program code.

(3) Send or share a video of your world's resulting animation. The File/Export Movie feature does not work in Alice 2.0 but it is promised for Alice 2.2. We like the free (so far) screen capture software called Jing which can record your world's animation as a Flash file (swf). All that is needed to see your world playing is a web browser to open the Flash file. You can also post the file to a web site and send people the web link. See: http://jingproject.com.

```
    smoke set opacity to 0 (0%)
robin say My hero!
// Hero and heroine exit
Do together
    robin turn to face farmHouse
    peter turn to face farmHouse
robin set vehicle to peter duration = 0 seconds
Do together
    peter.walk move_time = 8
    robin say Would you like a cup of tea? duration = 8 seconds
    peter move amount = 8 meters toward target = farmHouse duration = 8 seconds
Do together
    smoke move amount = 0.5 meters toward target = robin duration = 0 seconds
    smoke resize 2 duration = 0 seconds
    smoke turn to face farmHouse duration = 0 seconds
robin set vehicle to world duration = 0 seconds
smoke set vehicle to world
Do together
    Do in order
        Do in order
            Wait 1 second
            peter.jump times = 2
    light set brightness to 0 duration = 3 seconds
    world set ambientLightBrightness to 0 duration = 3 seconds
    world set atmosphereColor to [color] duration = 3 seconds
    smoke set opacity to 0.7 (70%)
```

Figure 3-50 The end part of the act2 method.

3.5: Putting the entire play together (act1 + act2)

Recall that inside of **world.***my first method*, we are calling two other methods: *act1* and *act2*. For this Chapter, we disabled *act1* so we could concentrate on *act2* while we were building and testing it.

Edit **world**.*my first method.* You will see that *act1* is disabled. 

Right-click on it and select enable.

Play the world and watch the entire play unfold. Speed up the playback if you want. If you find glitches in the show, try to track them down. Debugging is a fundamental part of creating computer programs. Almost no one ever gets it right on the first try. For example, your authors found that the dragon was showing up in *act1*. We had forgotten to set its *opacity* to 0.

Here are speeded up videos of *act1* and *act2* combined. There are two user choices. Peter is doomed: {**Video demo**: Video 3-15}. Peter prevails: {**Video demo**: Video 3-16}.

Authors' version of act1 and act2 combined is AWB3-19.a2w and is at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Final Thoughts on Chapter 3

This concludes our play. All of the Chapter 3 authors' files are posted to a web site should you wish to refer to it at a later time. These files are available at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Try the Chapter 3 Exercises that follow. They are meant to give you practice and to stretch your knowledge and skills. Of course, you are encouraged to explore Alice on your own. You can't break anything, so have fun!

Also, read the Chapter 3 Skills, Concepts and Capabilities section to find out more about the competencies you have learned and how they relate to the Snyder *Fluency with Information Technology* book.

Exercises for Chapter 3

Congratulations! You have just created your first animated movie and your first complex computer program. Let's continue to tweak the program in various ways. As usual, you should make a copy of the original in case you make mistakes and want to start over, and remember to have fun and be humorous.

1. Let's have Robin actually speak her words. Record some words for **robin** and replace the written words with the spoken words. (Hint: Click on **robin's** methods in the usual way, drag **robin**.*play sound* into the **Method Editor** panel. Select record new sound… and take it from there. Remember: humor is a good thing.)

2. Add a few chickens in the background and have them move occasionally. Can you make a chicken's legs move, as if it were scratching for something to eat? That would be cool. Also, chickens come with premade sounds. Check them out.

3. Add some background music for the entire movie. (Hint: First, legally grab your favorite piece of music off the web, or you might have a good song on your computer already. Add a new instruction at the very beginning of the scene that causes the music to be played throughout the scene, but don't let the music override the speech of **peter** and **robin** from Exercise 1.)

4. When Peter does his snowball toss, it would be more realistic to have his wing move. Try to identify which motion works best to simulate wing movement (e.g., roll or turn, and for how much of a revolution).

5. Peter's final snowball grows a bit too quickly, even with the "begin gently" option. Divide this up into a few steps to control the growth of the snowball as it moves toward the dragon.

6. With each snowball entering the dragon's mouth, it would be nice if the dragon had less fire. The 'index' expression in the Loop control gives you the current number of the loop. Use this index to reduce the dragon's *flameSize* on each Loop control iteration.

7. In the *act2* method, the Peter shall succumb and Peter shall prevail blocks of instructions were quite long. Create separate methods to contain the instructions for these blocks.

8. Create an End of Play 3D Text to appear at the end of Act 2. We did this for the end of Act 1.

Group Exercises

The following two exercises are group exercises. Depending on the size of the class, form groups of three students or so, and see what you can come up with.

1. **Have a Film Festival**. Start with the original movie and do your own creative tweaking. Change it in any way you like – change behaviors, add new objects, or whatever. Just have fun. When you're done, play each modified movie to the entire class, where the class as a whole votes on which one is the best, the most creative, or the most interesting. Maybe you'll want to have several categories of awards.

2. **Write an Original Script**. Review the simple narrative of our story in Chapter 1, and also review the pictorial storyboard in Chapter 3. Each group should now brainstorm about some original script totally unrelated to Peter and Robin. Imagine your own background objects, characters, behaviors, and storyline. Each group should discuss it, argue about it, laugh about it, but eventually write up its own narrative and pictorial storyboard in a two-page document to present to the class for feedback and constructive criticism. Optionally, if you're motivated, implement your original script in an Alice program as we have done with Peter and Robin. Do you aspire to write commercials for your favorite product or cause? Here is your chance.

Chapter 3 Skills, Concepts and Capabilities

Maybe you didn't know it, but in Chapters 1, 2, and 3 you learned about the nitty-gritty details of computer programming (e.g. the concept of an algorithm as a sequence of instructions and the five essential properties of algorithms – input, output, definiteness, effectiveness, and finiteness.) You also learned that computers execute instructions sequentially unless programmed to do otherwise.

You learned that computers can be programmed to make decisions somewhat like humans do. For example, "If X is at least two meters away from Y, then X turns to face Y and moves forward." That was the While control. Another example was the If/Else control. When the **world**.*doomed* variable got to be *true*, then the block of instructions in the Peter shall succumb block was executed. Otherwise, the block of instructions in the Else section (Peter shall prevail) was executed.

This decision-making capability suggests an advanced area of I.T. called Artificial Intelligence (AI) which involves programming computers to make decisions much like intelligent humans. AI is used to program robots and to program the intelligence of characters in popular games. If you're interested, see if your school offers a course in AI. It is fun stuff and quite leading edge. It is the stuff of the future, and you've done a little of it during this course!

You learned that a computer program can be looked at as a collection of collaborating objects, each having its own special properties, behaviors, and means to communicate with others – just like collaborating individuals in an organization. This is called object-oriented programming. You learned about parameters and variables and how they can be used in instructions for flexibility and efficiency.

In addition, you learned about iterative loops that execute a set of instructions again and again. You encountered the notion of randomness which was used to determine the minimum and maximum number of snowballs that Peter could throw at the dragon. We never knew how many would be thrown: 2, 3, or 4.

And finally, you learned how to debug computer programs. As we put it earlier: Computers only do what you tell them to do but not always what you want them to do. Inevitably, sometimes **robin** won't behave the way we intended, so we have to examine the instructions we wrote for **robin's** behavior very closely to discover where we went wrong. Don't worry, the best programmers hardly ever write instructions perfectly the first time around.

In Chapter 3 you learned several other important concepts that contribute to your fluency in I.T.: the program development cycle; top-down design; how to convert ideas into computer programs; interactive software; and how to manage complexity. You got used to using the Alice GUI, we hope!

Top-down design. Think about the storyboard in Chapter 3. The story board is the very first step in the program development lifecycle. It is a high-level description of how our movie should play out. Sometimes these are called "user requirements." All software, even mundane business software, starts with high-level descriptions of how they are to be used by consumers. From there, computer programmers (aka, software engineers) gradually transform these high-level descriptions into computer programs. This is just what we did in this Chapter. It's called top-down design. And further, our design included interaction with the user. We developed the software such that the user provides input to determine how the movie plays out.

You learned a few other tricks about programming in this Chapter too: how to focus on just a portion of the instructions in order to save time while developing and debugging a program; how to handle multiple objects in 3D space; and how to use variables and parameters to modify the behaviors of objects. In particular, you might have noticed that it is difficult to think about objects in 3D space when it is from someone else's point of view. Don't worry. This is challenging for most everyone. Our discussion and your thinking about objects in 3D space are very good mental exercises that will train your brain to think clearly about other challenging matters.

We hope that you had fun with this chapter. You have acquired lots of skills and competencies to make it easier to learn even more about computer programming.