Fluency with Alice

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

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

Chapter 2. Quiet on the Set - Action!

The way we left *act1* in Chapter 1, **peter** and **robin** turn to face the camera when *act1* starts. Then, **peter** flaps his wings and falls over backwards with some sound effects. See **Figure 2-1**.

The director realizes that this is not what the script calls for. It seems that **peter** is a bit premature in his excitement since **robin** has not yet approached, much less noticed him. First we need to remove the instructions that cause this premature behavior in **peter**.

Figure 2-1 In Chapter 1 act1, peter prematurely falls.

Cure peter's premature action: Launch Alice and open the World that you created in Chapter 1. [If you like, you can download the authors' end of Chapter 1 file, that we have renamed **AWB2-1** and which can be found at: http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/] Be sure that the *act1* method is active by clicking on its tab in the **Method Editor** panel. If you don't see its tab in the **Method Editor** panel, just click the **world** object in the **Object Tree** panel, click the methods tab and then click the edit button next to *act1*.

Delete the Do together control that created **peter's** premature actions by dragging it up to the Trash can icon  at the top of the screen. See **Figure 2-2**. {**Video demo**: Video 2-1}.

After you trash the Do together control, click the Undo button to restore it. Then trash it again, for the last time. We have you do this twice because we want demonstrate that the Undo button is your friend! Get used to using it.

> *Alternative*: You could disable the Do together control instead of trashing it. This is done by right clicking on it and selecting disable. It will remain available for use later when it could be enabled, but it will take up space until then.

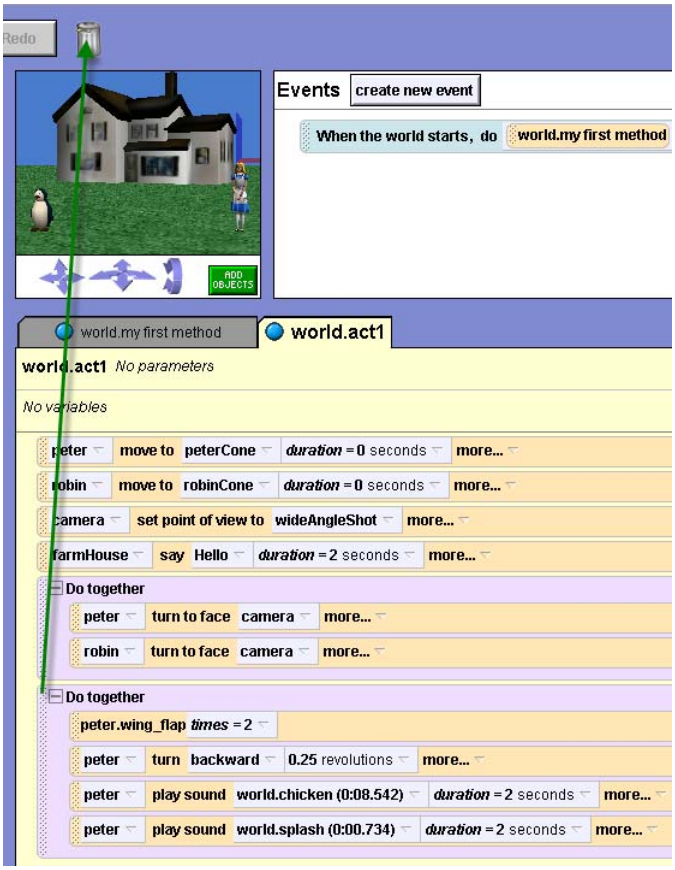


Figure 2-2 Delete (i.e., trash) the Do together control.

2.1: Robin takes an interest

Now that we have eliminated peter's premature excitement, it's back to our main story which we will finish in this chapter. By the end of this chapter, your Alice world will look like this video when played. **{Video Demo:** Video 2-2.}

We know that **robin** is interested in getting closer to **peter** so she turns to face him and begins to walk toward him in 1 meter steps. To make this happen, we need to add some more instructions to *act1*.

STEP 1 - We are still working with the *act1* method. In the **Object Tree** panel, click on **robin**. In the **robin's Details** panel methods tab, scroll down to **robin** *turn to face* method and drag and drop it underneath the Do together control (NOT inside of it). Select: **peter**/the entire peter. See **Figure 2-3**. Click Play to see the result.

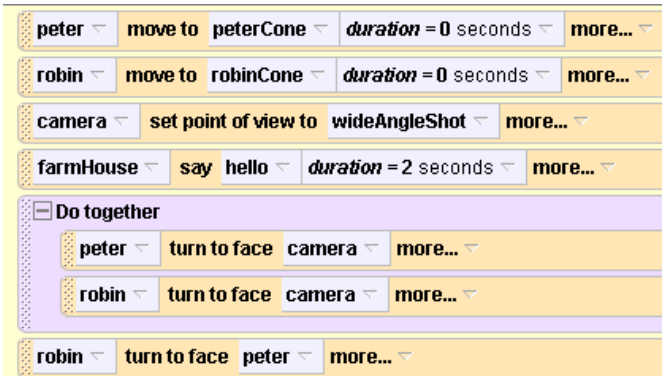


Figure 2-3 robin turns to face peter.

STEP 2 - Now let's set up the messages to tell **robin** to approach **peter**. Drag the While control from the bottom of the *act1* **Method Editor** panel and drop it under (NOT inside) the **robin** *turn to face* **peter** direction. Select true from the drop-down list. See **Figure 2-4**. {**Video demo**: Video 2-3}

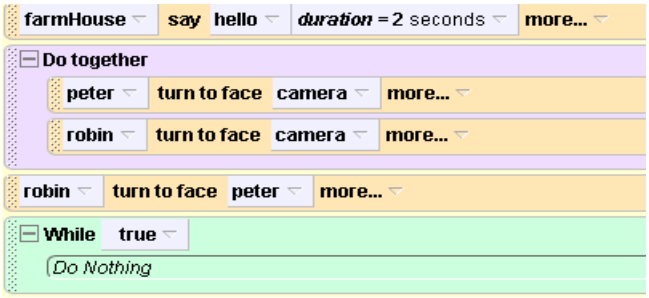


Figure 2-4 While control in place.

We want the While control to make **robin** walk forward 1 meter at a time as long as she is more than 2 meters away from **peter**. So, let's place an instruction to this effect in the *Do Nothing* area of the While control.

STEP 3 - In the **Object Tree** panel, click on **robin** and then in the methods tab of the **Details** panel drag and drop the **robin** *move* method into *Do Nothing* area of the While control. Select forward/1 meter. See **Figure 2-5.** {**Video demo**: Video 2-4}

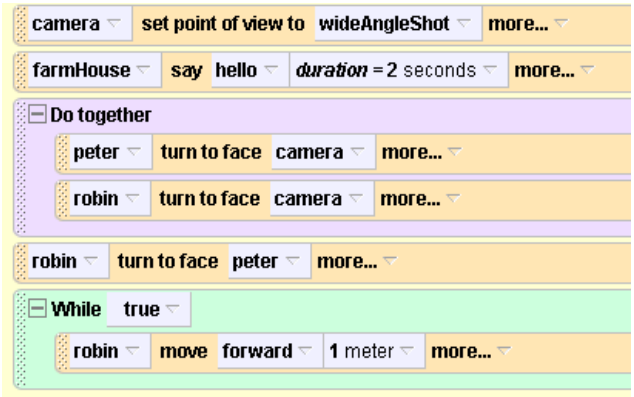


Figure 2-5 While control taking shape.

Click on the Play button in the upper left corner of the screen and watch what happens in the World Running… window that opens. Ooops! Notice that **robin** begins to move toward **peter**, and keeps on moving right off the screen. In our virtual world, **robin** is moving still. In the World Running… window, click on the Stop button to halt **robin's** movement and close the window.

The problem is that we did not tell **robin** when to stop moving. There is no conditional test to see how close she is to **peter** - only the value true in the While control.

The reason **robin** moves forward forever is that we did not provide "finiteness" to our directions to **robin**. We told **robin** to turn to face **peter** and to move forward 1 meter - but we did not tell **robin** when to stop moving. We need to provide finiteness by telling **robin** to stop when she is within proximity of **peter**, let's say 2 meters. In other words, we need to change our directions to **robin**. Currently she is moving 1 meter forward an infinite number of times (called an infinite loop).

We need to make a change in the While control. We need **robin** to move forward 1 meter only until she is within a certain proximity, 2 meters, of **peter**. Fortunately, Alice provides built-in functions that can be used to determine information about an object such as how close it is to another object. We will use a proximity function that will return a value of true as long as **robin** is more than 2 meters away from **peter**.

STEP 4 - Click on **robin** in the **Object Tree** panel and then click on the functions tab in **robin's Details** panel. Notice that the first item in the list is proximity. If there is a + sign to the left of proximity, click on the + sign to expand proximity. See **Figure 2-6**.

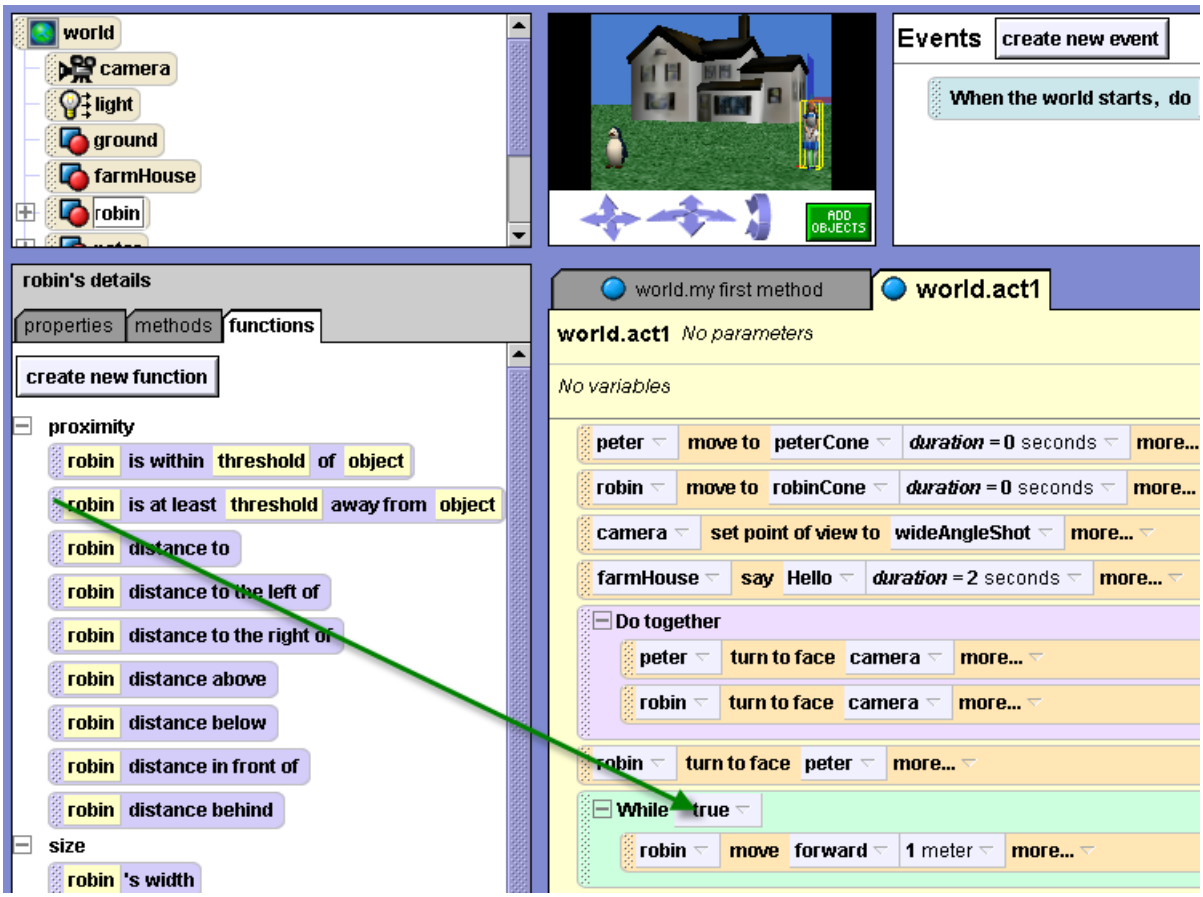


Figure 2-6 robin's functions are showing. [Move blue panel border to the right if needed.]

STEP 5 - Drag the proximity function *robin is at least threshold away from object* and drop it on the value true. In the pop-up window, choose: is at least/2 meters/away from/**peter**/the entire **peter** as shown in **Figure 2-6**. The resulting instructions are shown in **Figure 2-7**.

{**Video demo**: Video 2-5}

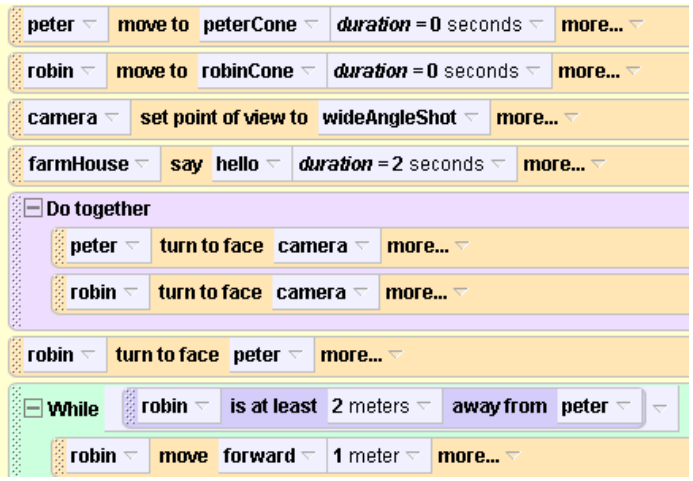


Figure 2-7 Using the proximity function in the While control.

The While control has been completed. As long as **robin** is at least 2 meters away from **peter**, the conditional test returns a value of true and the **robin** *move forward 1 meter* instruction is executed. As soon as the condition returns a false, **robin** stops moving.

To see if it works, click Play. You should now observe that **robin** stops 2 meters from **peter**. See **Figure 2-8**.

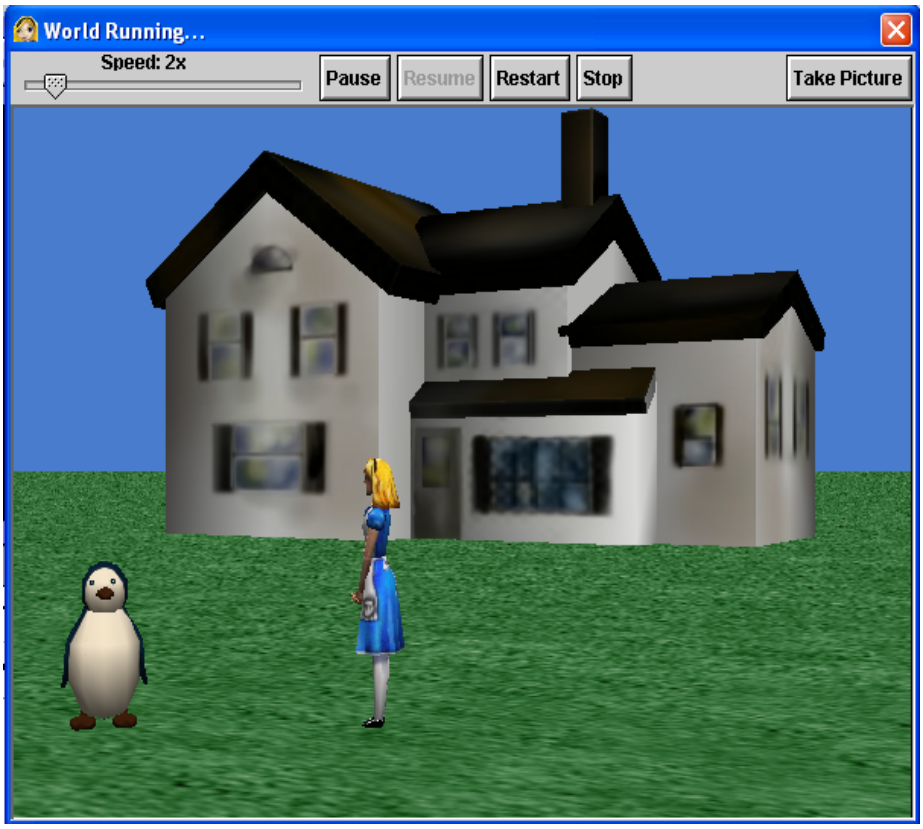


Figure 2-8. robin stops 2 meters from peter.

2.2: Peter reacts to Robin

Continuing on with our story, it is **peter's** turn to react to **robin.** When **robin** is within 2 meters of **peter**, she stops. **peter** turns to face **robin** and is startled by her being so close. In surprise, **peter** does three things at the same time: jumps up and down twice; flaps his wings twice; and spins around twice. As the Director, you can take some license with the script!

STEP 6 - With **robin 2** meters away, **peter** turns to face her. In the **Object Tree** panel in the upper left of your screen, click on **peter** and then on the methods tab in **peter's Details** panel. Scroll down until you see the **peter** *turn to face* method. Drag and drop it underneath (NOT INSIDE) the While control. Select: **robin**/the entire robin. See **Figure 2-9**.

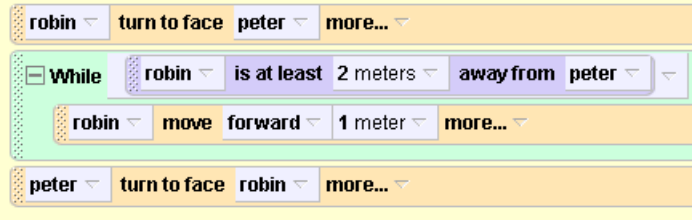


Figure 2-9 peter turns to face robin when she is within 2 meters.

STEPS 7, 8 & 9 are shown in: {**Video demo**: Video 2-6}

STEP 7 - Since **peter** is going to be doing several things together, we might as well drag the Do together control into the **Method Editor** panel and drop it underneath the **peter** *turn to face* **robin** direction. See **Figure 2-10**.

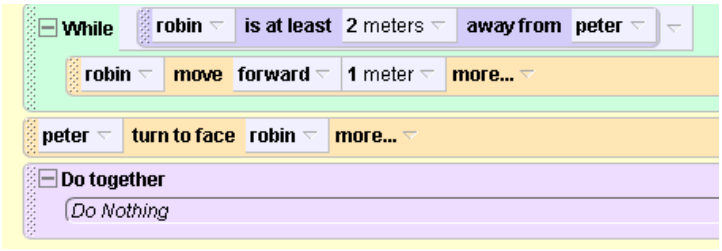


Figure 2-10. Do together for peter's reactions to robin's proximity.

STEP 8 - In the **Object Tree** panel, click on **peter** and then look in the **peter's Details** panel in the methods tab. Drag the method *jump times* and drop it into the *Do Nothing* part of the most recent Do together control. Select: *2*. Do the same for **peter's** method *wing flap* and select: **2**. See **Figure 1-11.**

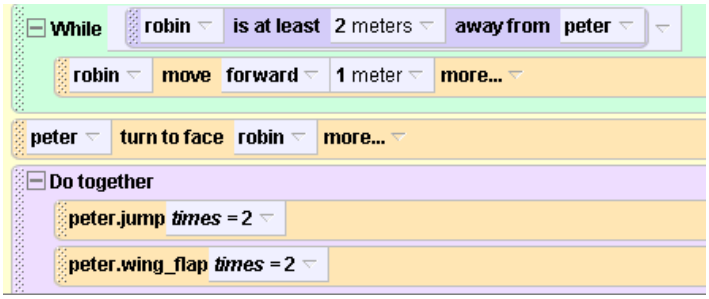


Figure 2-11. Do together for peter's reaction is complete.

Click Play to see the story so far. It looks like things are working as planned, but it's difficult to see **peter's** wings flap. Let's add some additional behavior to **peter** so we can clearly see his wings flap.

STEP 9 - In the **Object Tree** panel, click on **peter** and then on the method tab. Find the method *turn* and drag and drop it under the **peter.***wing_flap* message in the Do together control. Select: left/2 revolutions. Click **more...** and set the duration for two seconds. See **Figure 2-12**.

Click Play. You should now see **peter's** wings flap as he jumps and turns.

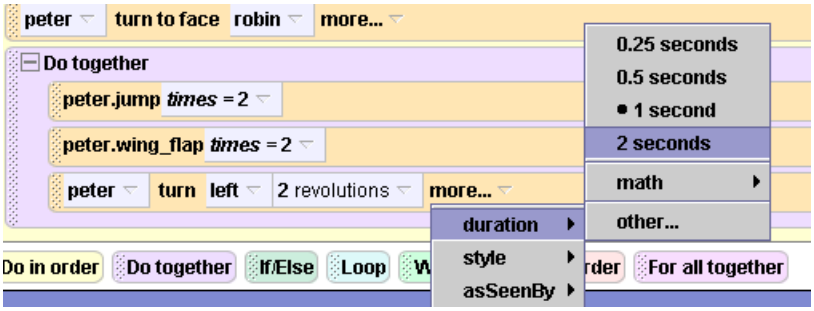


Figure 2-12 Completed Do together control for peter's reaction to robin.

2.3: Robin is startled by Peter's reaction

We want **robin** to be startled by **peter's** reaction to her movement toward him. She backs up 1 meter and turns to face the **camera**. The **camera** moves in for a close-up of **robin's** face as she blurts out the words "Oh my goodness! Sorry about that."

STEP 10 - In the **Object Tree** panel, click on **robin** and then in the methods tab of the **robin's Detail** panel and drag **robin** *move* method underneath the last Do together control (not inside of it) and drop it. Choose: backward/1 meter.

Now we need **robin** to turn to face the **camera** which is still in wide-angle mode.

STEP 11 - Click on **robin** in the **Object Tree** panel and in the methods tab in the **robin's Details** panel find the *turn to face* method. Drag it under the last instruction in the **Method Editor** panel and select: **camera**. See **Figure 2-13.** Click on Play.

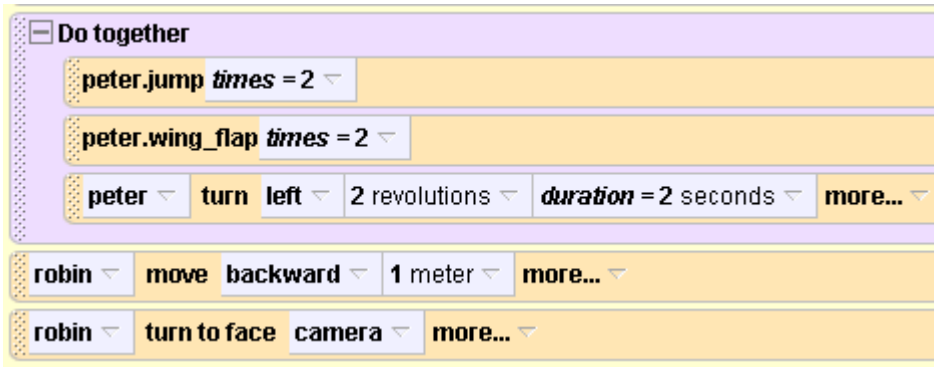


Figure 2-13 robin backs up 1 meter and turns to face the camera.

The **camera** now needs to pull in tight for a close-up of **robin's** face.

STEP 12 - In the **Object Tree** panel, click on **camera** and in the **camera's Details** panel in the methods tab find the **camera** *set point of view to* method. Drag and drop it under the last instruction in the **Method Editor** panel and in the pop-up window select: Dummy Objects/**robinCloseUp**. See **Figure 2-14.**

Click on Play. **Ooops!** We have a problem. The problem is that the camera is not pointing at **robin**. Time to debug the instructions.

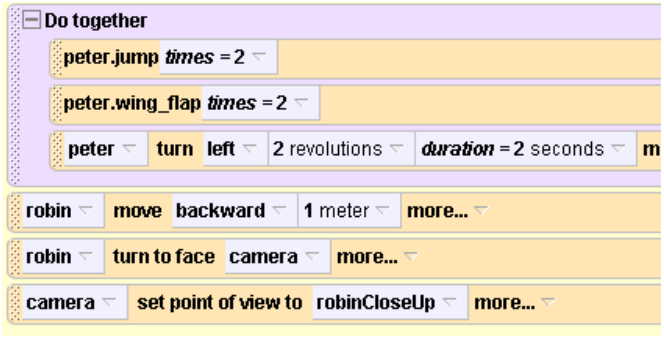


Figure 2-14. Camera needs to do a close-up of robin's face.

> *Debug protocol*: (a) What are the symptoms of the problem? (b) Why does the problem occur? (c) How can we correct or avoid the problem?

Debug the instructions

Recall that the Dummy Object/**robinCloseUp** was set up for the location of the **robinCone** which was where **robin** started the scene. So naturally, the **camera** did a close-up of that area, but **robin** had moved from her cone to be close to **peter**.

STEPS 13, 14 & 15 are shown in: {**Video demo:** Video 2-7}

STEP 13 - Delete the last *set point of view to* instruction to the **camera** by dragging it to the trash can at the top of the window. [You can also right-click on the instruction and select delete.].

STEP 14 -In the **Object Tree** panel, click on **camera**. In the **Details** panel find the method **camera** *move toward*. Drag and drop it under the last instruction in the **Method Editor** panel. In the pop-up window, choose: 2 meters/**robin**/**neck**/**head**/the entire head. See **Figure 2-15**. Click on Play.

> *Reality Check*: Computers don't always do what we want them to do, but always do what we direct them to do.

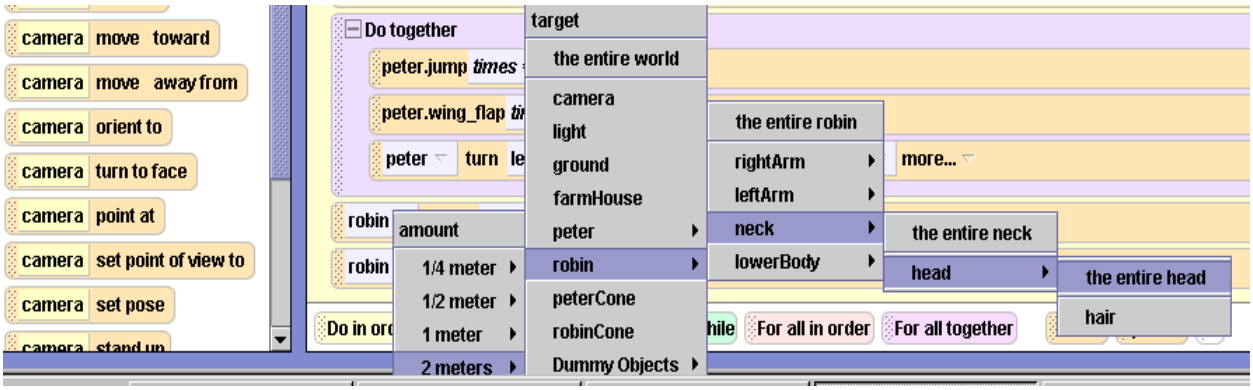


Figure 2-15 Move the camera two meters towards robin's head.

STEP 15 - But, we are not close enough to **robin's** face. So, in the **Method Editor** panel, click on the amount in the **camera** *move* direction and choose other. Click on *other* and then on 7 and then on the Okay button. [You would have used trial, error and Undo to get the 7 meters.]

Click Play. Works better now.

2.4: Robin shouts out her regrets as Act 1 ends

We want **robin** to blurt out her regrets: "Oh my goodness! Sorry about that." The lights dim and *act1* ends.

STEP 16 - **robin** needs to say something. Click on **robin** in the **Object Tree** panel. Drag and drop the **robin** *say* method underneath the last instruction in the **Method Editor** panel and select other. In the input box type: "Oh my goodness! Sorry about that." See **Figure 2-16.**

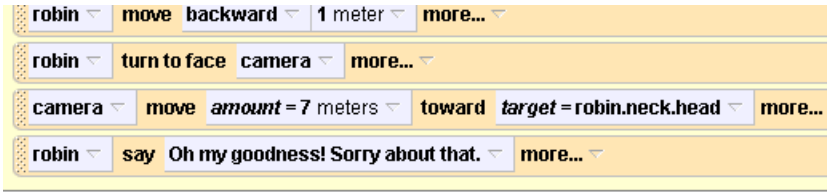


Figure 2-16 robin speaks.

Click Play. **Ooops again!** That was too fast.

STEP 17 -Remember that the default duration for methods in Alice is one second. **robin** needs her words to remain on the screen for a longer duration. In the **robin** *say* direction in the **Method Editor** panel, click on more. Choose: *duration*/other/5/Okay. See **Figure 2-17**.

Click on Play. That should be better.

Figure 2-17 robin speaks for a longer duration.

This is how the scene ends so let's dim the lights to fade out.

STEP 18 - To fade the **world's** lights, click on **world** in the **Object Tree** panel and in the **Details** panel, click on the properties tab. Drag and drop the *ambientLightBrightness* property below the last instruction in the **Method Editor** panel. Choose: .25

Click Play. Remember that the lights will not be dimmed until the display of **robin's** comments has been completed.

The dimming was kind of fast. Let's slow it down.

STEP 19 -In the *ambientLightBrightness* instruction, click on more and choose: *duration*/other/5. Click Okay button. See **Figure 2-18.**

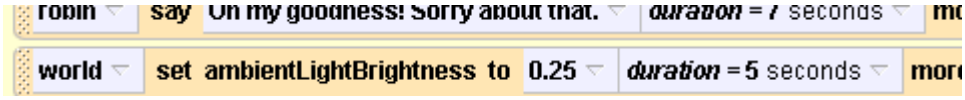


Figure 2-18 Fade the world.

Click Play but don't speed up the playback. {Animation should look like this video demo. {**Video demo**: Video 2-8}

Save your world.

The complete set of *act1* instructions is shown in **Figure 2-19.**

Authors' file AWB2-2.a2w to this point in the chapter can be found at http://media.pearsoncmg.com/aw/aw_snyder_fluency_3/alice/World_Files/.

Figure 2-19 Final list of instructions for act1.

Final words on Chapter 2

This concludes *act 1*.

Try the Chapter 2 **Exercises** that follow. They are meant to give you practice on what you already know how to do and to also 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 2 **Skills, Concepts and Capabilities** section to find out more about the theory and competencies you have learned and how they relate to the Snyder *Fluency with Information Technology* book.

Exercises for Chapter 2

Bravo! You have created the first act of your animated movie. and have perhaps written your first computer program. For exercises, you can tweak the program in various ways. You've likely heard the expression "Act 1 – Take 1" many times. That's what we're doing. You should store a copy of the original Act 1 in case you want to start over. And, remember to use the Undo button if needed. Have fun, and be humorous.

1. In the current scene, Peter jumps and spins after Robin has approached him. Tweak the scene so that Peter jumps and spins as Robin approaches him. (Hint: Use the Do Together control and slide Robin's and Peter's behaviors into the Do Together control.)

2. Make Robin jump and spin also as she moves back from Peter. (Hint: Using Peter's instructions as a guide, create them for Robin also.)

3. At the end of the scene, have Robin say something other than "Oh my goodness! Sorry about that." Have her say something humorous like "This dude has flipped ... literally. I'm outta here" and have her move quickly off the stage to the right.

4. Have Peter think something at the very end of the scene, perhaps "What did I do?" (Hint: "Think" is a predefined method , so select Peter's *think* methods.)

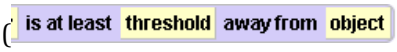
5. Try putting object sound effects into your exercises, above.

Chapter 2 Skills, Concepts. and Capabilities

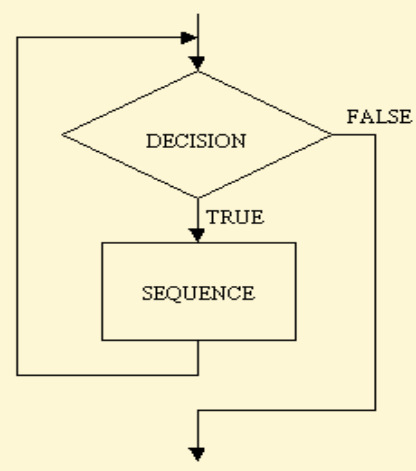
While you've been having fun making movies with Alice, you continue to learning some important things about algorithms and computer programming. Here is how what you have learned in the chapter relates to *Fluency* skills, concepts, and capabilities

As in Chapter 1, with regard to the five essential properties of algorithms, we specified what the movie would look like when completed (output), specified direction for each stage of the movie (definiteness), know that Alice can carry out your directions (effectiveness) and that the movie stopped when completed (finiteness). When you specified distance (e.g., 2 meters) and duration (e.g., 5 seconds), you were providing input data to the program/algorithm (input property of algorithms).

You learned to revise (i.e., edit) instructions by deleting (and disabling) them. This is part of the program development life-cycle.

Also, you've learned that computers execute instructions (i.e., directions) sequentially, unless told otherwise (e.g., Do together). You learned that computers can be programmed to make decisions (i.e., While control). Here, **robin** moved 1 meter closer to **peter** until she got within **2** meters. The part of that instruction that made the condition test was the *proximity* function (

) which returned the Boolean value true to the While control as long as **robin** was more than 2 meters from **peter**. When, **robin** was no longer 2 meters away from **peter**, the *proximity* function returned the Boolean value false. This signaled the end of the While control leading to the next sequential instruction to be executed. All functions return values. Without this stopping condition, recall that **robin** entered an infinite loop.

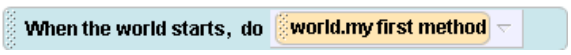


This flow chart shows the While decision flow scheme.

You learned that in Alice (and all object-oriented programming environments) the fundamental building blocks are **objects** which have their own methods, properties and functions.

Using the Alice GUI, you were able to create a set of directions that had two objects (**peter** and **robin**) interact when the starting event triggered program execution (*When the world starts* …). Also, the story directions to **robin** had a logical structure (sequential, concurrent, and a looping While control).

You learned that computers only do what you tell them to do but not always what you want them to do (e.g., Ooops!). That means that you need to detect and correct mistakes (debug) which is part of stepwise refinement of computer programs (i.e., Play, test, modify, Play, test...). Debugging programs is a vital skill to learn. Everyone makes mistakes!

You learned that an event 

was launched upon the click of the Play button and that a play or movie script can be broken down into directions and that those directions, invoked in the right sequence, can create an Alice computer animation.