`Then you should say what you mean,' the March Hare went on.

`I do,' Alice hastily replied; `at least--at least I mean what I say--that's the same thing, you know.'

`Not the same thing a bit!' said the Hatter. `You might just as well say that "I see what I eat" is the same thing as "I eat what I see"!'

2 Programming in Alice: Program Design and Implementation

In this chapter, we begin an introduction to programming. A program is a set of instructions that tell the computer what to do. Each instruction is an action to be performed. Writing a program to animate 3D objects in a virtual world is naturally all about objects and the actions objects can perform. From a practical viewpoint, writing a program is somewhat like working with word problems in math. In word problems, we first read the word problem (a description of the situation) and decide how to go about solving the problem (what steps need to be done). Then, we solve the problem (write a solution) and finally test our answer to make sure it is correct. Similarly, in writing an animation program we first read a scenario (a description of the story, game, or simulation – often called the problem statement) and decide how to go about creating the animation (design a storyboard). Then, we write the program code (implementation) and finally test the program by running the animation.

As in Alice's conversation with the March Hare (see above), you must say exactly what you mean when you write a program. The best way to write a program is to begin by reading a scenario (the description of the story, game, or simulation) and then design a list of actions for the program.

Section 1 of this chapter begins with scenarios and storyboards as a methodology for designing programs. Visual storyboards were chosen because they are the design tool used by professional animators in animation film studios. Textual storyboards were chosen because they provide an algorithmic (step-by-step) structure. The lines of text in a textual storyboard are similar to pseudocode – a loose version of the instructions that will eventually become program code.

Section 2 presents the basics of creating a simple program in Alice. The idea is to use a storyboard as a guide for writing the program (list of instructions) in Alice's mouse-based editor. We can focus on a step-by-step solution because Alice will automatically take care of all the details of syntax (statement structure and punctuation). In an animation, some actions must take place in sequence and other actions simultaneously. This means the program code must be structured to tell Alice which actions to *Do in order* and which actions to *Do together*.

2-1 Scenarios and Storyboards

Creating a computer program that animates objects in a virtual world is a four-step process: read the scenario (a description of the problem or task), design (plan ahead), implementation (write the program), and test (see if it works). This section introduces the first two steps.

Reading the scenario and designing a plan of action are important steps in constructing programs for animation. A design is a "plan ahead" strategy and takes practice to master. While the programs presented in the first few chapters of this text are reasonably clear-cut, we think it is advisable to start building good designs early on. Then, when programs begin to get more complicated, the time invested in learning how to design good program solutions will pay great dividends.

Read the scenario

Before we can discuss how to create a design, it is first necessary to know what problem is going to be solved or what task is going to be performed! A scenario is a problem (or task) statement that describes the overall animation in terms of what problem is to be solved or what lesson is to be taught. (Many computer scientists use the term requirements specification. In Alice, the term scenario is easier to relate to the world scene, objects, and actions.) Cartoons and feature-length animated films begin with a scenario created by professional writers. Sometimes, a scenario is called "the story." As used here, in addition to the traditional meaning of "story," a "story" can be a lesson to teach, a game to play, or a simulation.

In an Alice world, a scenario gives all necessary details for setting up the initial scene and then planning a sequence of instructions for the animation. That is, a scenario provides answers the following questions:

1. What story is to be told?

2. What objects are needed? Some objects will play leading roles in the story while other objects will be used to provide background scenery.

3. What actions are to take place? The actions in the story will eventually become the instructions in the program.

Scenario example

You have recently been sitting at home, having missed another day of classes because a winter snowstorm dropped 2 feet of snow on the ground. You see some children outdoors creating snowpeople. You are daydreaming about a dance you recently attended. Being a very creative person, your imagination gets carried away and the two scenes blend together: Several snowpeople are outdoors, on a snow-covered landscape. A snow song is playing. A snowman is trying to meet a snowwoman who is talking with a friend (another snowwoman.) The snowman tries and tries to get her attention. He calls out "Ahem" and blinks his eyes at her. She turns to

look at the snowman and blushes. But, alas, she is not interested in meeting him. She gives him a cold shoulder and turns back to talk with her friend. He gives up and turns away.

From this scenario, we have answers to questions:

What story is to be told? This scenario tells a sad story about a snowman's unsuccessful attempt to get the attention of a snowwoman in a winter scene.

What objects are to be used? The objects are snowpeople and the background scenery should depict a winter scene.

What actions are to take place? The actions include the snowman trying to attract the attention of the snowwoman, the snowwoman blushing but not being interested, and the snowman turning away.

Design

A storyboard is the design approach we will use to create a solution to a problem, or plan a list of actions to perform a task, as specified in the scenario. At Pixar, Disney, and other major animation studios, animators break down a long scenario into sequences of many short scenarios. For each scenario, a storyboard is created to depict the sequence of scenes. The storyboard may consist of dozens of scene sketches, drawn by animation artists or generated by computer animation specialists using computer software. This approach of designing a solution to a problem by breaking it down into sub-problems is not unique to computer programmers and animators. Playwrights, for example, break their plays down into individual acts, and the acts into individual scenes! Engineers break down complicated systems (e.g., jet airplanes and claw hammers) or devices (e.g., microcircuits) into component parts to make the problem more manageable.

Figure 2-1-1 illustrates an example of a storyboard in Pixar's ….

Figure 2-1-1. Example from Pixar (to be inserted here)

Visual storyboards[1]

A visual storyboard breaks down a scenario into a sequence of major scenes with transitions between scenes. Each sketch represents a scene (state) of the animation – sort of a snapshot of the scene. Each snapshot is associated with objects in certain positions, colors, sizes, and poses. When one or more transitions (changes) occur in the animation, the transition leads to the next scene (state).

The snapshots are numbered in sequence and labeled with necessary information. For short animations, the breakdown might be presented on one large sheet of paper. For more complex designs, a separate sheet of drawing paper might used for each scene to allow the animation artist to easily rearrange or discard scenes without starting over.

A technique we can use to create a visual storyboard is borrowed from professional animators -- a sequence of hand-drawn scenes. A visual storyboard template is shown in Figure 2-1-2. Each snapshot is labeled with a **Scene Number** and contains a sketch or picture showing where the objects are in the scene. The **Description** tells what action is occurring. If sound is appropriate in the animation, the description will include a list of sounds that will be played during the scene. If a comic-book style is desired, text may be included to show the words or phrases that will be displayed in a comic-book-style text bubble. Sound and/or text are used only if needed.

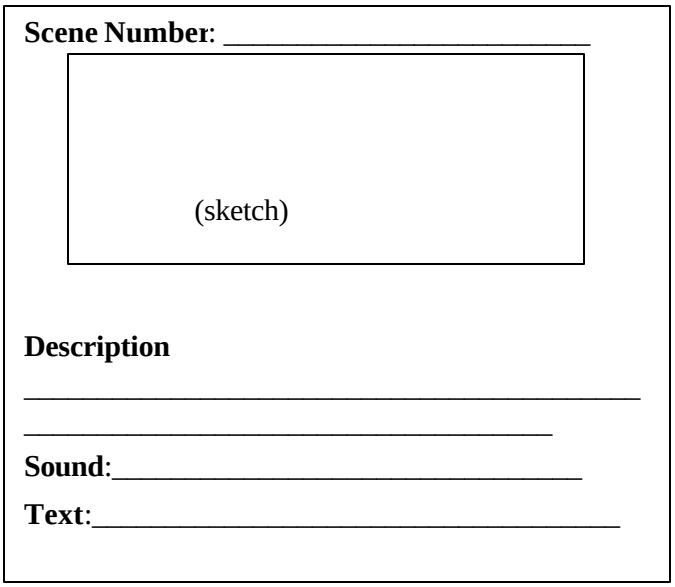

Figure 2-1-2. Storyboard template

For our purposes, preparing storyboard sketches is not intended to be a highly artistic task. Simple circles, squares, and lines can be used to represent the objects that will appear in the scene. If necessary, shapes can be labeled with the name of the object or color-coded.

To illustrate the creation of a storyboard, the sample scenario for the snowpeople party (as previously described) will be used. The sketch in Figure 2-1-3 shows a simple drawing of a scene

[1] "Storyboard" is an animation term. In other areas of computer science, the term "state-transition-diagram" is used to refer to what we will describe here as a storyboard.

where the snowman blinks his eyes at the snowwoman of his dreams. Circles were used to create the snowman and snowwoman. Diagonal lines were drawn to create a background of mountains in the distance. The grey squiggly lines were put in to represent the surface of the snow-covered ground. Using simple figures, hand-sketched storyboards are quick and easy to create.

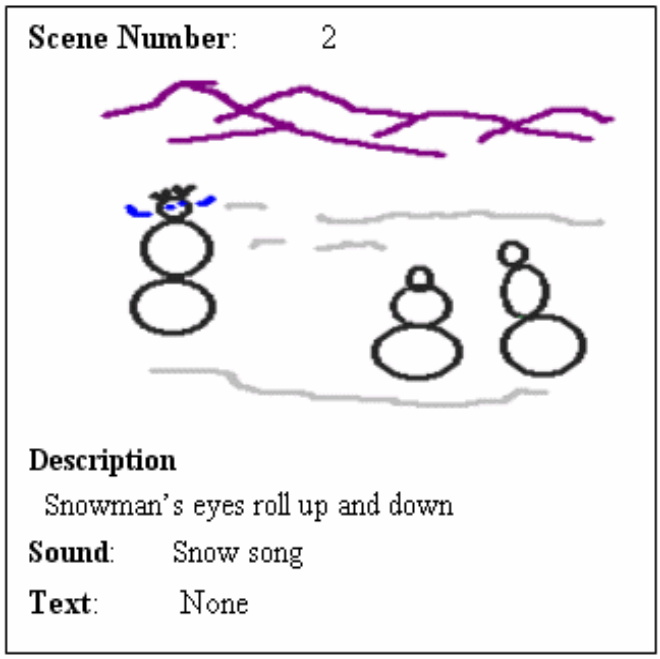

Figure 2-1-3. Hand-sketched visual storyboard

For illustrations in this book, we borrow a technique from professional animators to create visual storyboards. We use Alice's scene editor to add objects to a world and then patiently arrange the objects in various poses. As each successive scene is created, a screen capture is made and copied to a document. The screen captures shown in Figure 2-1-4 illustrate screen captures in a storyboard for the beginning of the snowpeople animation. Naturally, screen captures for a storyboard are fancier than hand-drawn sketches. But, hand-sketched drawings are much faster and easier to put together.

Textual storyboards

While professional animation artists use visual storyboards as part of their project development process, not everyone has the patience to make dozens of sketches. A textual storyboard is a good alternative to visual storyboards. A textual storyboard looks something like a "to-do list." While sketches and screen captures in storyboards provide a visual representation of the sequence of scenes, a textual storyboard allows us to prepare a planned structure for writing program code. To take advantage of each of these strengths, both visual and textual storyboards are used throughout this book.

Scene Number: 1

Description
The initial scene: the snowman is interested in meeting the snowwoman (wearing the read hat)
Sound: Snow song
Text: none

Scene Number: 2

Description:
The snowman tries to get the snowwoman's attention
Sound: Snow song
Text: Ahem….

Scene Number: 3

Description:
The snowman blinks his eyes.
The snow woman notices the snowman.
Sound: Snow song
Text: none

Figure 2-1-4. Screen captures of storyboard scenes

Textual storyboard example

A textual storyboard for the snowpeople animation is shown below. An important point that should be mentioned about textual storyboards is that a textual storyboard may summarize several scenes from a visual storyboard. For instance, the textual storyboard shown here summarizes scene number 1, scene number 2, and scene number 3 from the visual storyboard in Figure 2-1-4. (This storyboard represents only the first few actions. The storyboard will be completed in the next section.)

Do the following steps in order
snowman turns to face the snowwoman
snowman calls out to the snowwoman

Do the following steps together
snowman blinks his eyes at the snowwoman
snowwoman turns to see who is calling her.
etc.

The lines of text in a textual storyboard provide an ordered list of actions. The lines are written in an outline format and indentation makes the storyboard easy to read. Notice that two lines in the textual storyboard are in italics. These lines organize the actions – some actions are to be done in order (one at a time), others are to be done together (at the same time). The first two actions are performed in order (the snowman turns to face the snowwoman and then says "Ahem"). The third step is actually two actions performed simultaneously (the snowman blinks his eyes at the same time as the snowwoman turns around to see who has called out).

In computing terminology, a textual storyboard is called an algorithm – a list of actions to perform a task or solve a problem. The actions in a textual storyboard are very close to (but not quite) actual program code and so are often known as pseudocode.

Evaluate and revise

Once a storyboard has been designed, it is a good idea to take an objective look at the design to decide what might be changed. Evaluate the storyboard by answering these questions:

- Does the action flow from scene to scene, as the story unfolds?
- Do any transitions need to be added between scenes to blend one scene to the next?
- Did you overlook some essential part of the story?
- Is there something about the story that should be changed?

The important idea is that the storyboard is not final. We should be willing to review our plans and modify them, if necessary. In creating a program design, we go through the same kinds of cycles as an artist who has an idea to paint on a canvas. The painter often sketches a preliminary version of what the painting will look like. This is a way of translating the abstract idea into a more concrete vision. Then, the painter looks at the preliminary version and may change it several times before actually applying oils to the canvas. Likewise, an engineer who designs a bridge or an airplane (or anything else) goes through many design-modify-redesign phases before the final product is constructed. All creative people go through these design-modify-create cycles.

2-2 A First Program

In the previous section, you learned how to carefully read a scenario and design an animation to carry out a task, play a game, or create a simulation. Now, you are ready to look at how an animation program can be written. This step in building an animation is called implementation. We recommend that you read this section while sitting at a computer: start up Alice and repeat the steps shown in the example in this section.

What is a program?

As you know, a program is a list of instructions (actions) to accomplish a task. You can think of an Alice program as being somewhat like a script for a theatrical play. A theatrical script tells a story by describing the actions to be taken and the words to be delivered by actors on stage. In a similar manner, an Alice program prescribes the actions to be taken and the sound and text to be used by objects in a virtual world.

Create an initial scene

An ancient Chinese proverb is "The longest journey begins with a single step." Let's begin our journey by implementing the snowpeople animation described in the section 2-1. Recall that a snowman is interested in meeting a snowwoman. He tries to get her attention, but she isn't interested in meeting him, so she turns away. Then, he gives up. The first step of the solution is to create the initial scene. For a new world, a snow scene template is selected and then a snowman and two snowwomen (found in the People collection in the local gallery) are added. The initial scene is shown in Figure 2-2-1.

Figure 2-2-1. Snowpeople initial scene

Program code editor

Once the initial scene has been set up, the instructions that make up the program code must be written. Alice provides a program code editor -- the large yellow pane at the lower right of the main Alice window, as shown in Figure 2-2-2. The instructions for a program are entered in the editor. (From now on, we are just going to refer to the program code editor as "the editor".)

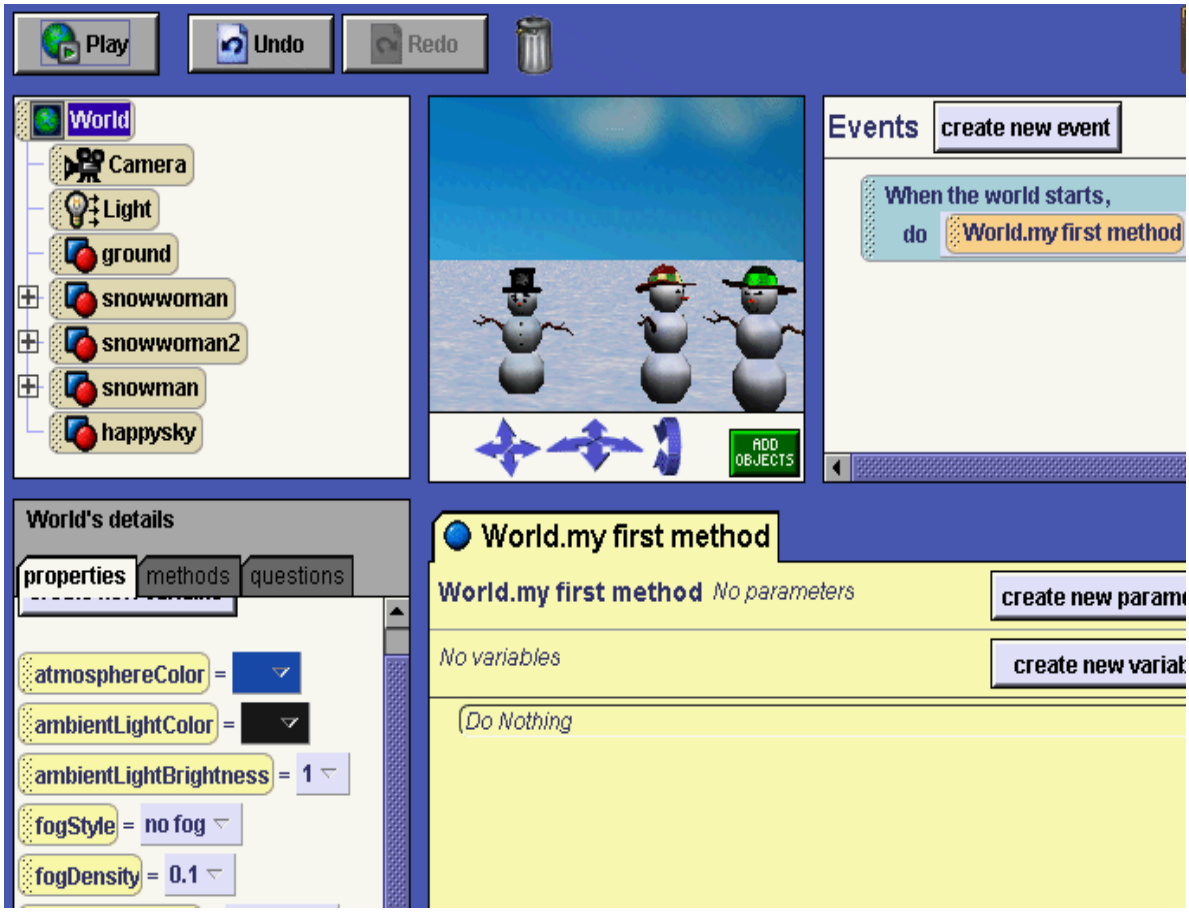

Figure 2-2-2. Program code editor (the large yellow pane)

World.my first method

As seen in Figure 2-2-2, the tab for the editing area is labeled *World.my first method*. A method is a segment of program code (a small set of instructions) that defines how to perform a specific task. Alice automatically uses the name *World.my first method* for the first editing pane. Actually, any name can be made-up and used for a method name. But, we will just use the name *World.my first method* for this example. The snowpeople scenario is simple enough to be programmed in just one method, *World.my first method*. When the **Play** button is pressed, Alice will execute *World.my first method* by carrying out the instructions that we write there.

What instructions are needed?

Let's take another look at the storyboard presented in the previous section, reproduced below.

> *Do the following steps in order*
>
> snowman turns to face the snowwoman
> snowman calls out to the snowwoman
>
> *Do the following steps together*
>
> snowman blinks eyes at the snowwoman
> snowwoman turns to see who is calling her.
>
> etc.

Actually, this storyboard is incomplete. The scenario (in the previous section) described a sequence of actions: (a) the snowman tries and tries to get the snowwoman's attention by calling out to her, (b) the snowman blinks his eyes at the snowwoman as she looks to see who is calling her; (c) the snowwoman blushes but isn't interested, so she turns away from the snowman, and (d) the snowman gives up. Only actions (a) and (b) were outlined. Let's complete the textual storyboard by adding the remaining actions, as shown below.

Do in order
- snowman looks at the snowwoman
- snowman calls out to the snowwoman
- *Do together*
  - snowman blinks his eyes at the snowwoman
  - snowwoman turns to see who is calling her.
- *Do together*
  - snowwoman blushes (her head turns red)
  - snowwoman turns back to her friends
- snowwoman's face turns back to white
- snowman turns away (gives up)

Translating a storyboard to program code

To translate a storyboard to program code, begin with the first step of the storyboard and translate it to an instruction. Then, translate the second step to an instruction, then the third, and so forth and so on until the entire storyboard has been translated to instructions. The instructions used in program code use the same built-in methods you learned in the Getting Started exercises in Appendix A. To display the snowman's available methods, click the snowman object in the Object tree and then the methods tab in the details area, as seen in Figure 2-2-3.

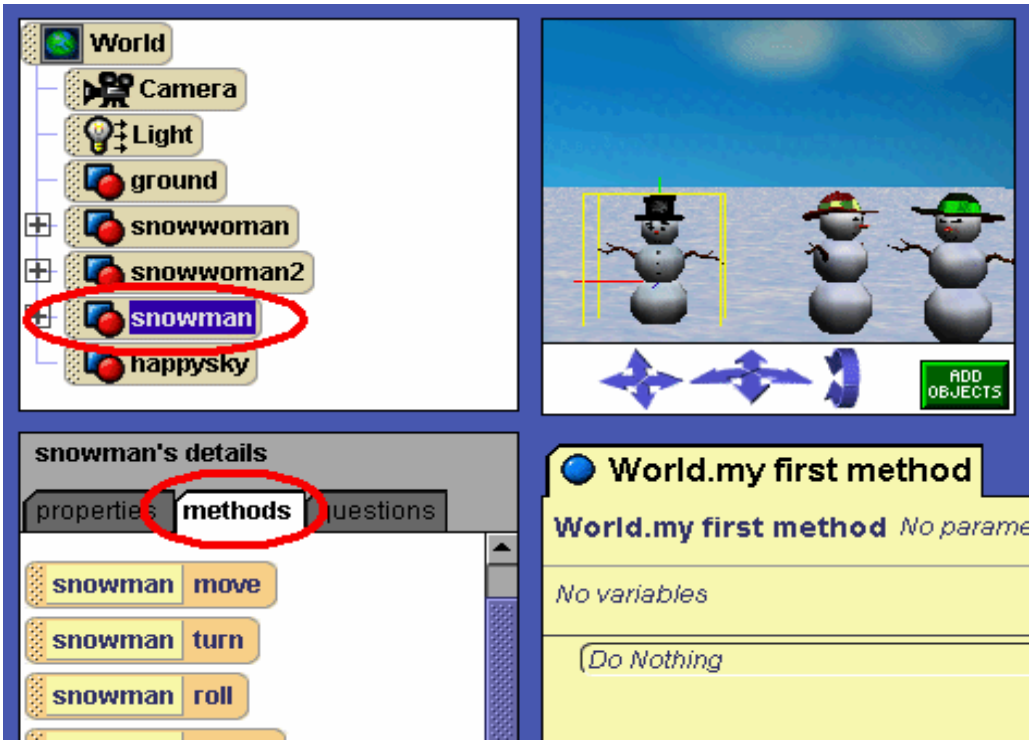

Figure 2-2-3. Built-in methods for writing program code

In our example, we want to translate the storyboard to program code. We begin with the first step in the storyboard, making the snowman turn to look at the snowwoman. One of the snowman's methods is *turn to face* – we can use this method to make the snowman turn to look towards the snowwoman. The next step is to have the snowman call out to the snowwoman. The snowman's *say* method can be used to make the snowman call out "Ahem" to the snowwoman.

Sequential versus simultaneous actions

From our storyboard, it is clear that the first two actions must occur in a specific sequence – the snowman turns to look at the snowwoman and then the snowman calls out "Ahem" to the snowwoman. We can tell Alice to *Do* these instructions *in order*. But, other actions occur simultaneously (at the same time). For example, the snowman blinks his eyes at the snowwoman at the same time as the snowwoman turns to face the snowman. To have these actions occur at the same time, Alice must be told to *Do* these actions *together*. *Do in order* and *Do together* are part of the Alice language. We call them control statements because we use these statements to tell Alice how to carry out the instructions in a program.

Do in order

To tell Alice to do instructions in sequential order, a *Do in order* block is dragged into the editor, as seen in Figure 2-2-4.

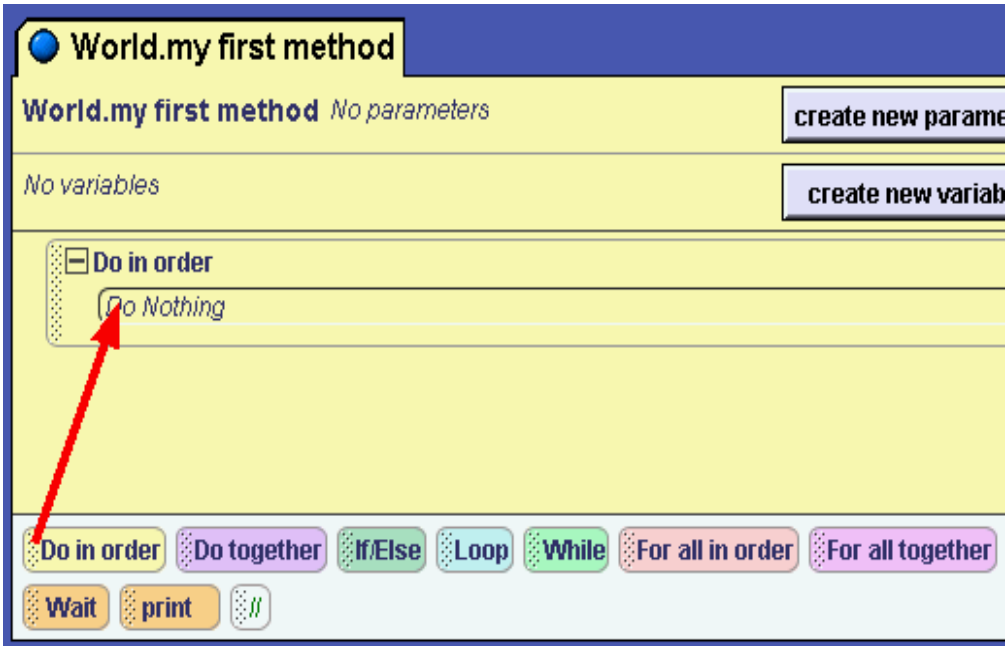

Figure 2-2-4. Dragging a *Do in order* tile into the editor

The first two instructions can now be placed within the *Do in order* block. The snowman is selected in the Object tree. Then, in the snowman's methods, the *turn to face* instruction is selected and dragged into the *Do in order*, as shown in Figure 2-2-5. The *turn to face* instruction requires an argument, namely which object the snowman should turn to face. (An argument is an item of information that must be supplied so Alice can execute the action.) In this example, the snowwoman is selected as the object that the snowman will *turn to face.*

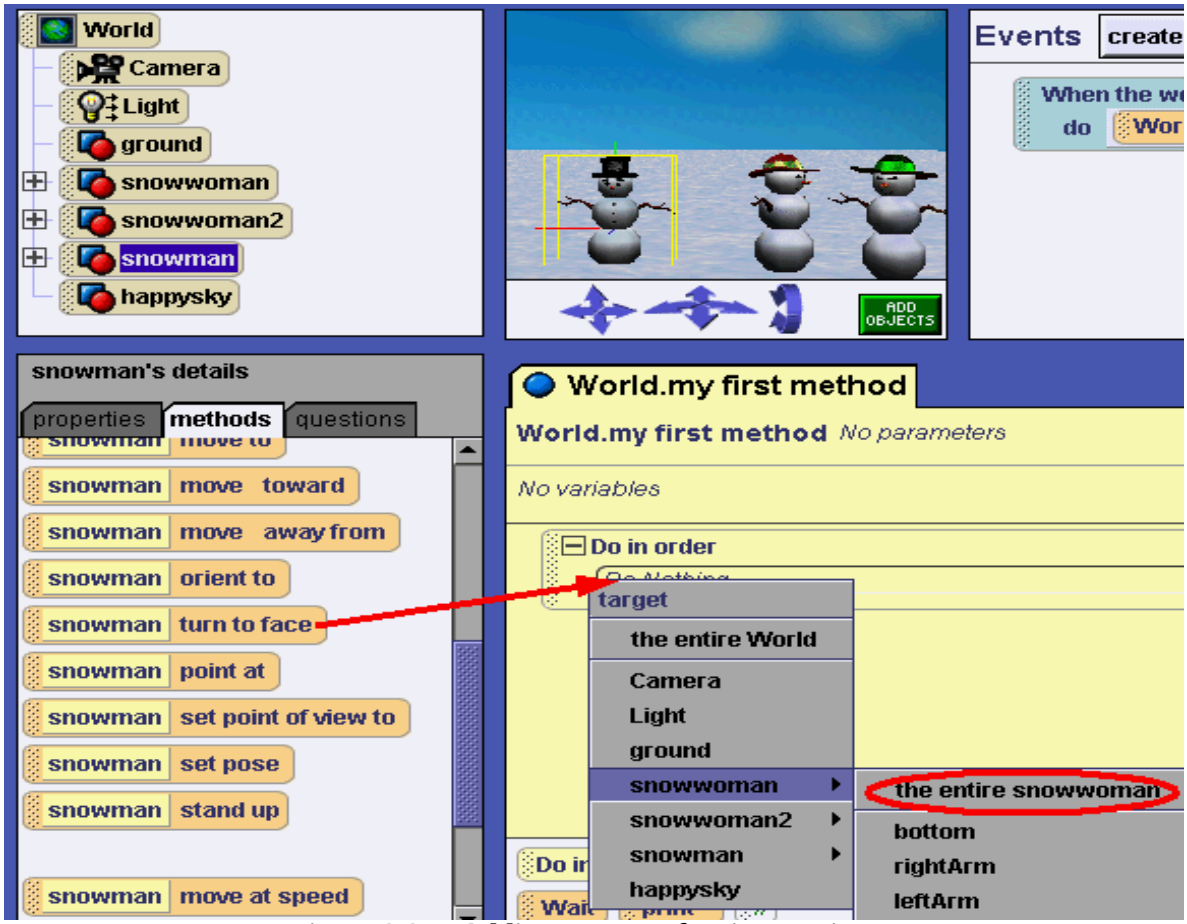

Figure 2-2-5. Adding a *turn to face* instruction

The resulting instruction is shown in Figure 2-2-6.

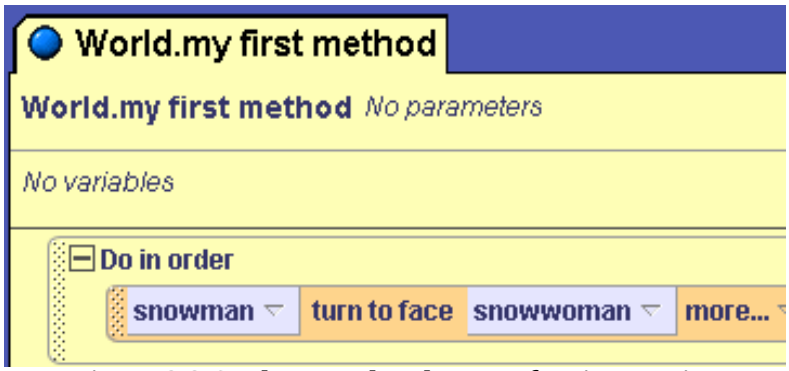

Figure 2-2-6. The completed *turn to face* instruction

In the second instruction, the snowman will *say* "Ahem" to the snowwoman. A snowman *say* method tile is dragged into the editor and "Ahem" is entered as the string of text to be displayed, as illustrated in Figure 2-2-7. The resulting code is shown in Figure 2-2-8. When this program is run (it is perfectly fine to try out the effect of just one or two Alice instructions by clicking on the **Play** button), the snowman will turn to face the snowwoman and then he will say "Ahem."

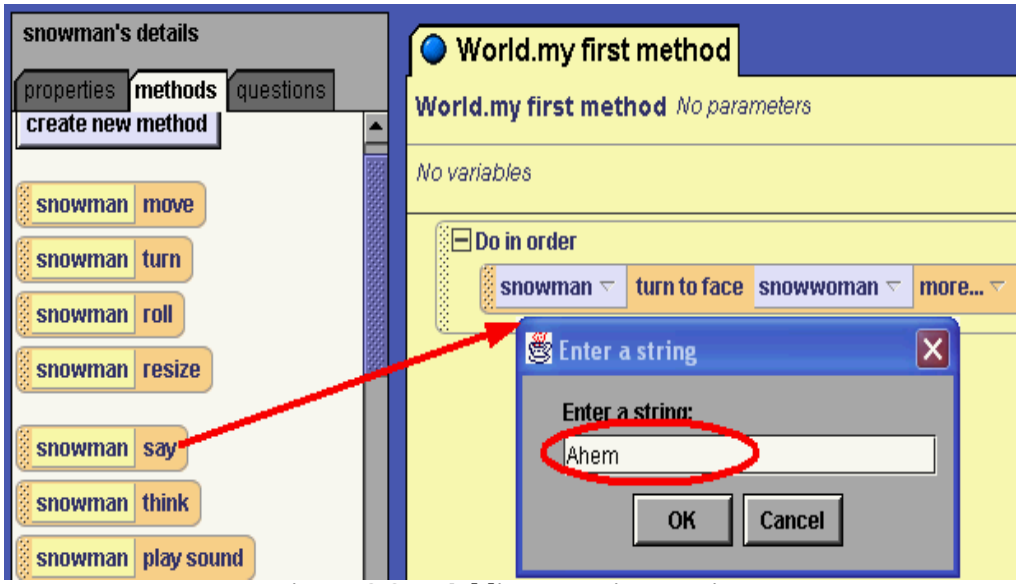

Figure 2-2-7. Adding a *say* instruction

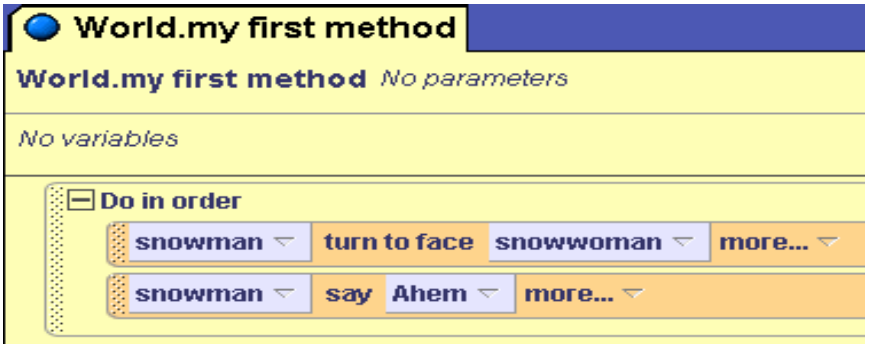

Figure 2-2-8. Resulting program code

Do together

The third step in the storyboard requires two things to occur at once: the snowman blinking his eyes at the snowwoman as the snowwoman turns her head to see who called out to her. A *Do together* tile is dragged into the *Do in order*, as shown in Figure 2-2-9. Notice the horizontal green line in Figure 2-2-9. The green line indicates where the *Do together* instruction will be dropped.

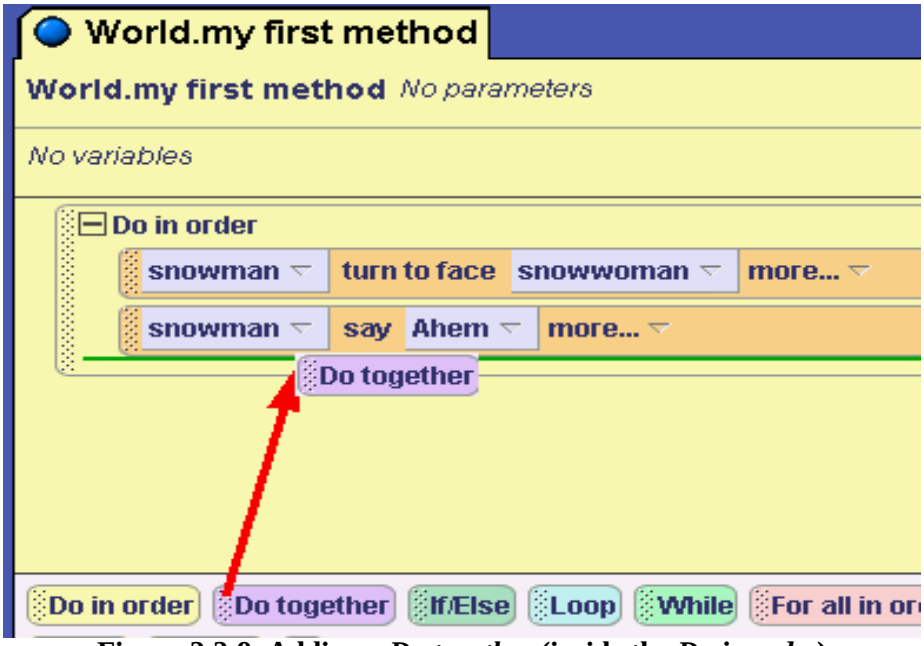

Figure 2-2-9. Adding a *Do together* (inside the *Do in order*)

The result of this modification, illustrated in Figure 2-2-10, is that the *Do together* block is nested within the *Do in order* block. Nesting means that one program statement is written inside another. Note that nesting the *Do together* inside the *Do in order* just happens to be the best way to animate this example. A *Do together* does not have to be inside a *Do in order*. These two coding blocks can work together or can work separately in many different combinations.

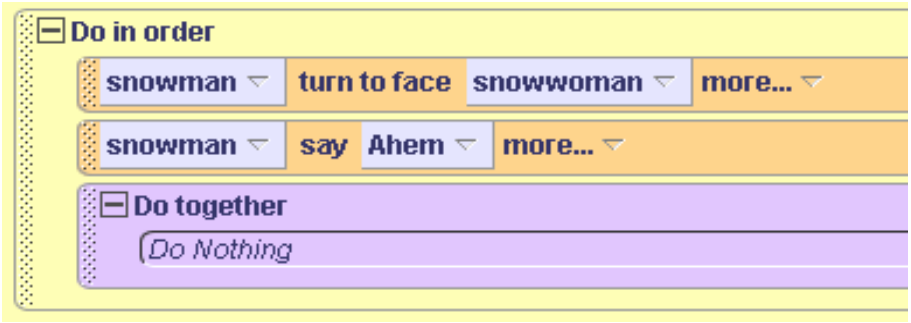

Figure 2-2-10. *Do together* nested within a *Do in order*

Now, instructions can be dragged into the *Do together* block to simultaneously turn the snowwoman's head and blink the snowman's eyes. How can we have the snowwoman turn her head to face the snowman? Clicking on the + to the left of the snowwoman in the object tree causes the snowwoman's parts to be displayed in the object tree. Then clicking on the snowwoman's head in the object tree allows access to instructions for moving her head, as illustrated in Figure 2-2-11.

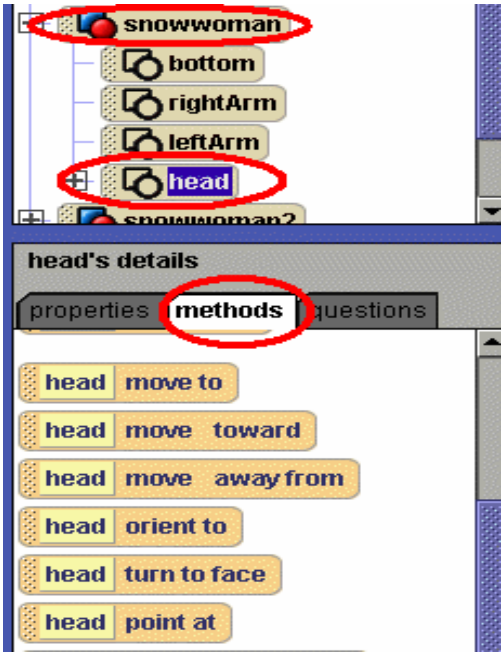

Figure 2-2-11. Accessing the methods for the snowwoman's head

A *turn to face* instruction for the snowwoman's head is added to the *Do together* block, as seen in Figure 2-2-12.

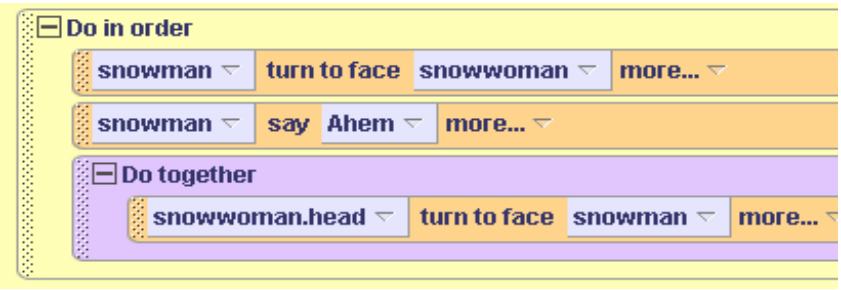

Figure 2-2-12. Code to turn the snowwoman's head

Creating instructions to make the snowman's eyes blink is more complicated. A click on the + to the left of the snowman in the Object tree allows access to the subparts of the snowman. Then, a click on the + to the left of the snowman's head allows access to the snowman's eyes, as shown in Figure 2-2-13.

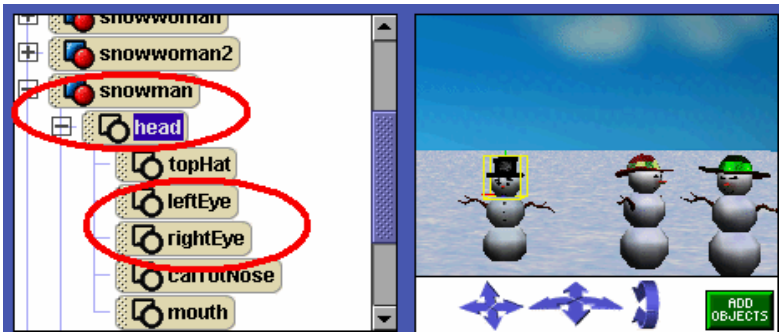

Figure 2-2-13. Accessing the snowman's eyes

It is now possible to drag instructions into the editor to move the snowman's eyes up and then down, as shown in Figure 2-2-14. Note that popup menus allow you to select arguments for the direction and the amount of movement. When *other* is selected as the amount, a number pad (looks like a calculator) pops up on the screen. We chose 0.04 meters, clicking the buttons on the number pad to make our selection. How did we know to use *0.04* meters as the distance? Well, we didn't. We just tried several different distance values until we finally found one that worked to give the best effect. This is an example of a trial and error strategy. While we always recommend good planning strategies, sometimes trial and error is useful.

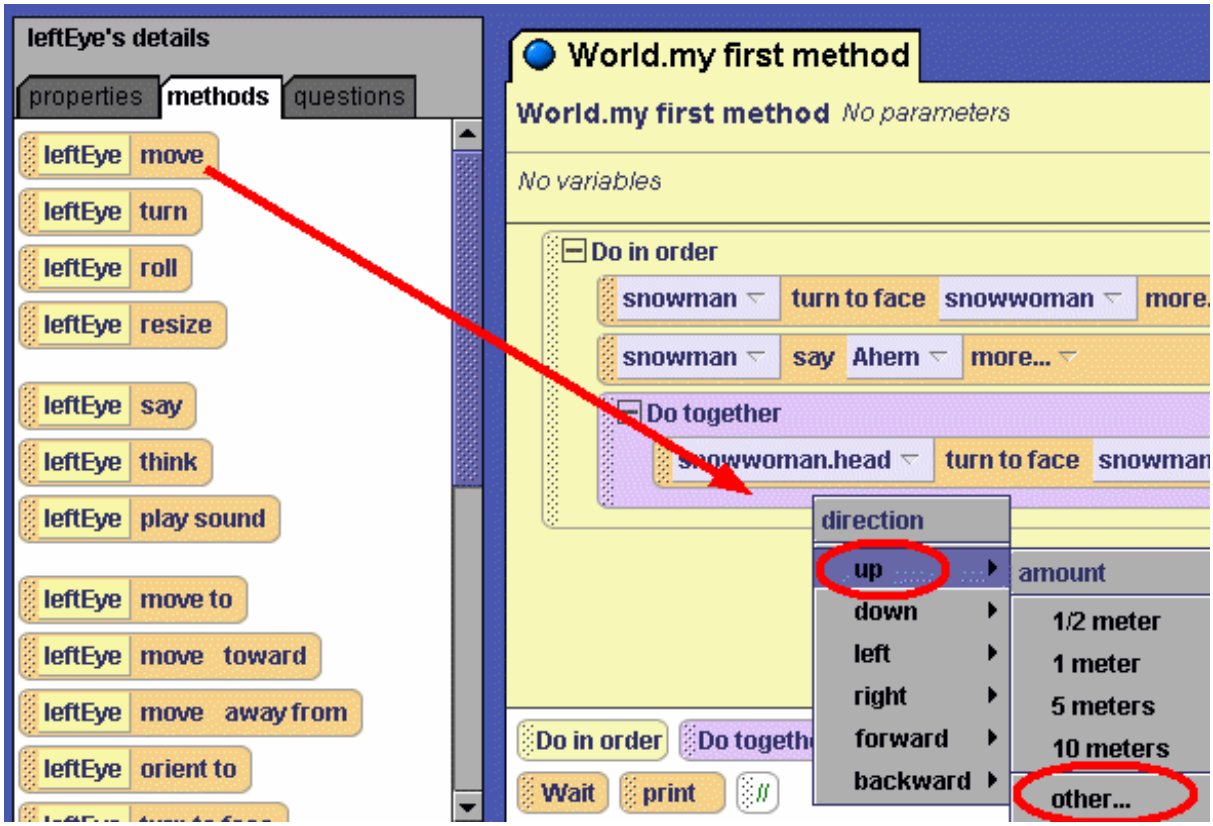

Figure 2-2-14. Dragging the *leftEye move* tile into the editor

Instructions are added to the editor for moving the *leftEye up*, *leftEye down*, *rightEye up*, and *rightEye down*, as seen in Figure 2-2-15.

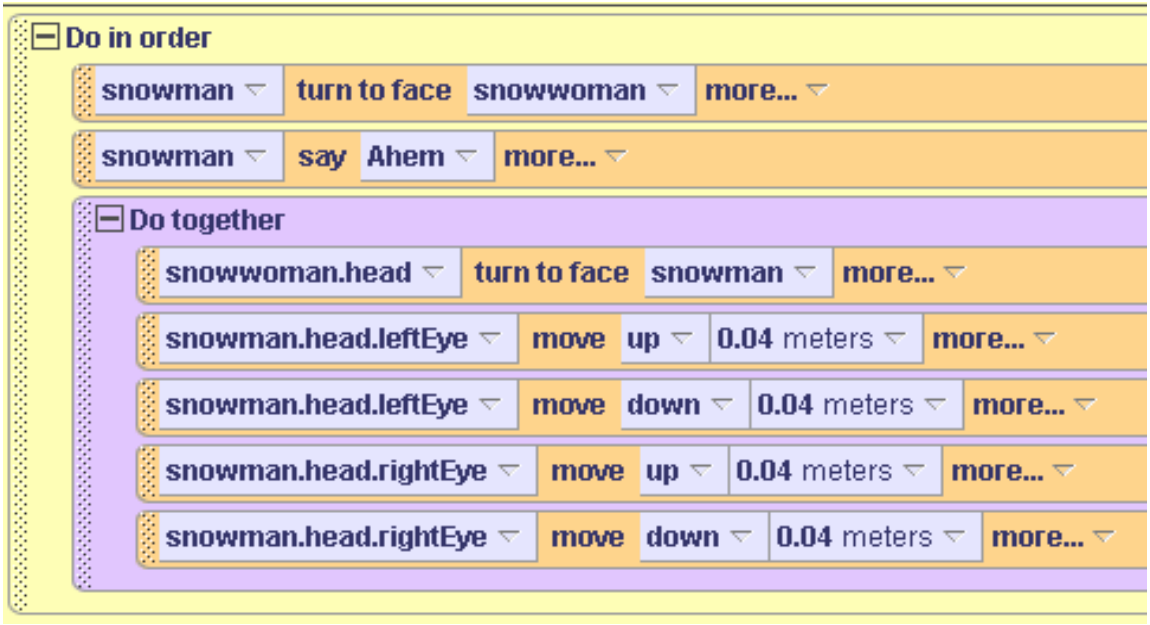

Figure 2-2-15. Program code has been added to make the snowman blink his eyes

Bugs

You will recall that the four steps in creating an animation program are: read, design, implement, and test. Now that several lines of code have been written (implemented), it is a good idea to test it to see if what you have written thus far works the way you thought it would. You do not have to wait until the entire program is completed. To test the instructions written thus far, the **Play** button is clicked. The snowman turns to face the snowwoman, the snowman says "Ahem," the snowwoman turns her head to face the snowman, but the snowman's eyes don't move up and down. In fact, they do not appear to move at all!

The reason the eyes do not move is the program has a bug. (Errors in computer programs are generally referred to as bugs. When we remove bugs from a program, we debug the program.) The problem is, in the code shown above, the snowman eye movement instructions are written inside a *Do together*. Of course, if the eyes are moving both up and down at the same time, they effectively cancel each other out, and the snowman's eyes do not move at all! To fix this problem, it is necessary to place the snowman's left eye movement instructions within a *Do in order* block and also the snowman's right eye movement instructions within a *Do in order* block, as illustrated in Figure 2-2-16.

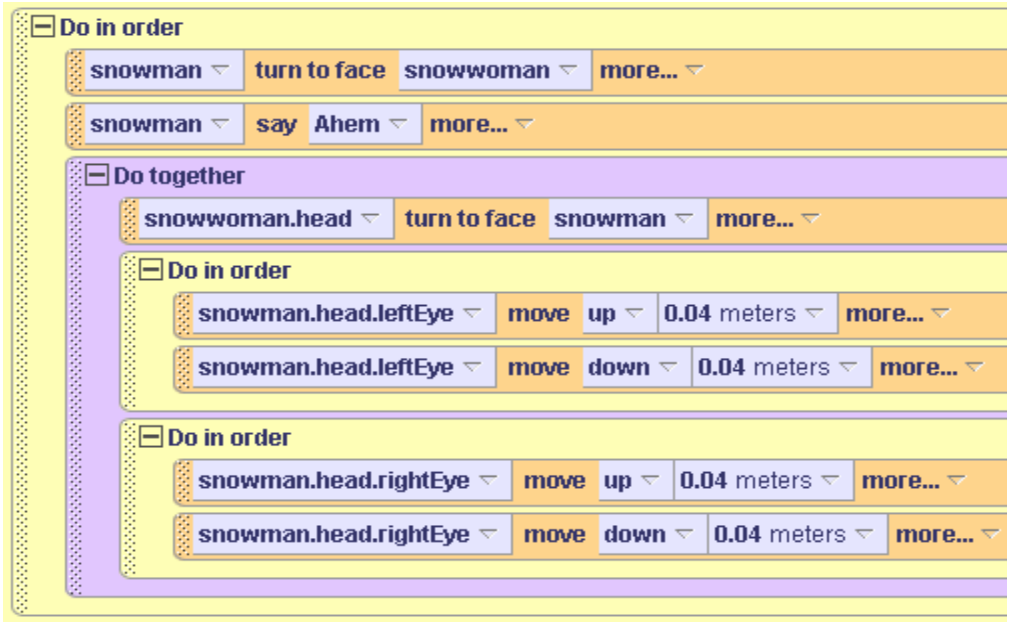

Figure 2-2-16. Revised instructions for blinking the snowman's eyes

Now, the snowman's eyes move up and down! There is one other useful observation to make. Animation instructions, by default, require one second to run. Normally, within a *Do together* block, each of the instructions should take the same amount of time. Since it takes one second for the snowwoman to turn her head, moving the snowman's eyes up and down should also take one second. However, there are two steps in moving his eyes (up and then down). Each step in moving his eyes should require ½ a second. To change the duration of an instruction, click on *more…*(at the far right of the instruction where the duration is to be changed), select the duration menu item, and select *0.5 seconds*, as shown in Figure 2-2-17.

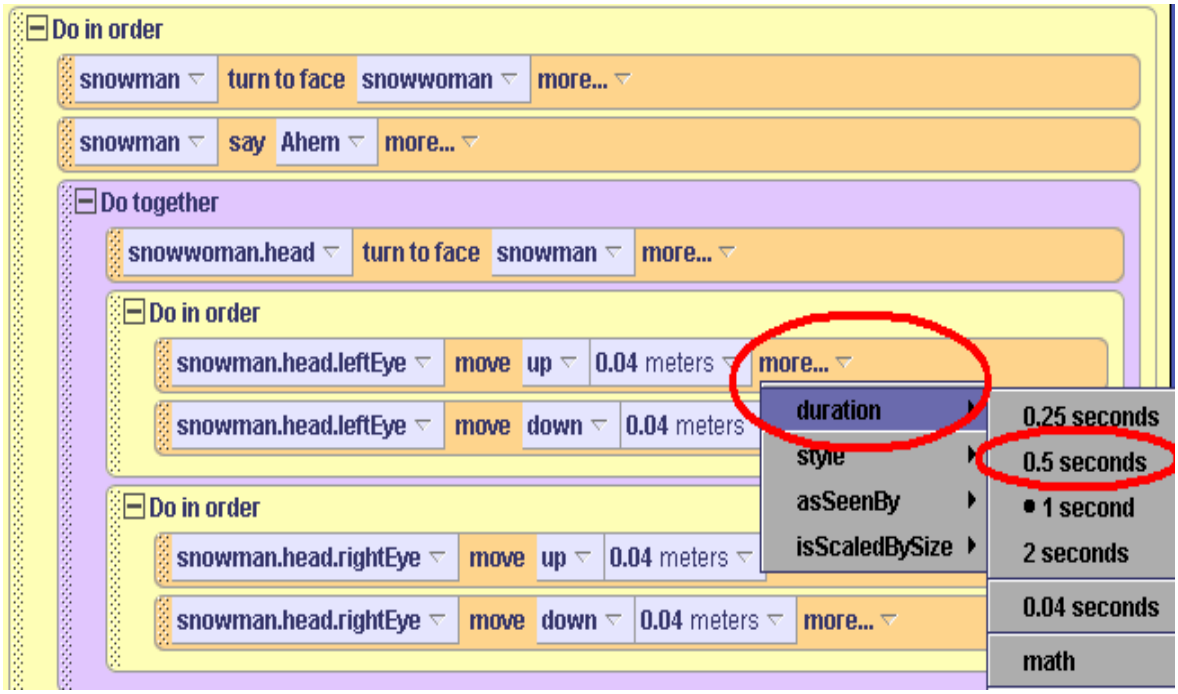

Figure 2-2-17. Changing the duration of an instruction

Using a property

We still need to complete the final two steps described in the storyboard. The fourth step requires the snowwoman to blush (her head turns red) as she turns her head back to her snow friends. Making the snowwoman's head change color is slightly different from other instructions we have used so far. To change the color of the snowwoman's head, we use the color property of the snowwoman's head. To view the list of properties of the snowwoman, select the snowwoman in the Object tree and select the properties tab in the details area (lower left of the Alice window), as shown in Figure 2-2-18.

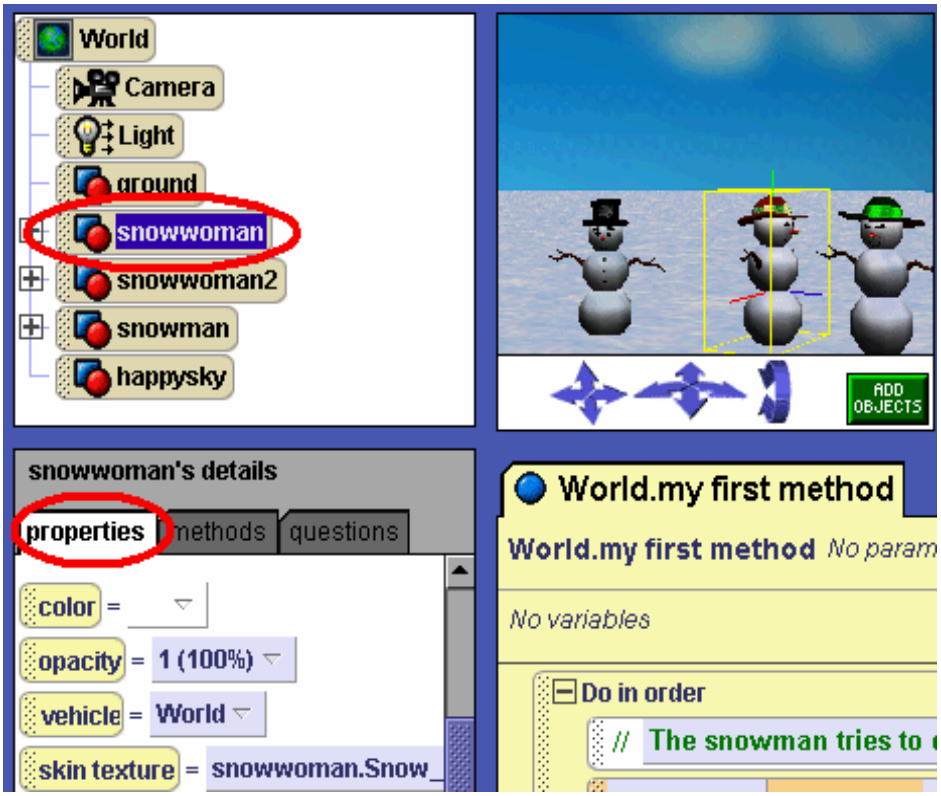

Figure 2-2-18. The properties of the snowwoman

What we want to do is change the color of the snowwoman's head when she turns to look at the snowman while the animation is running. (The technical term for "while the animation is running" is "at runtime.") Figure 2-2-19 demonstrates the steps. First, a Do together block is dragged into the editor (beneath all the instructions written thus far). Then, in the Object tree, the + beside the snowwoman is clicked to expand the tree, showing the subparts of the snowwoman. The subpart named *head* is selected. Then, the color tile in the properties list for the snowwoman's head is dragged into the *Do together* block. Finally, the color Red is selected from the popup menu of available colors.

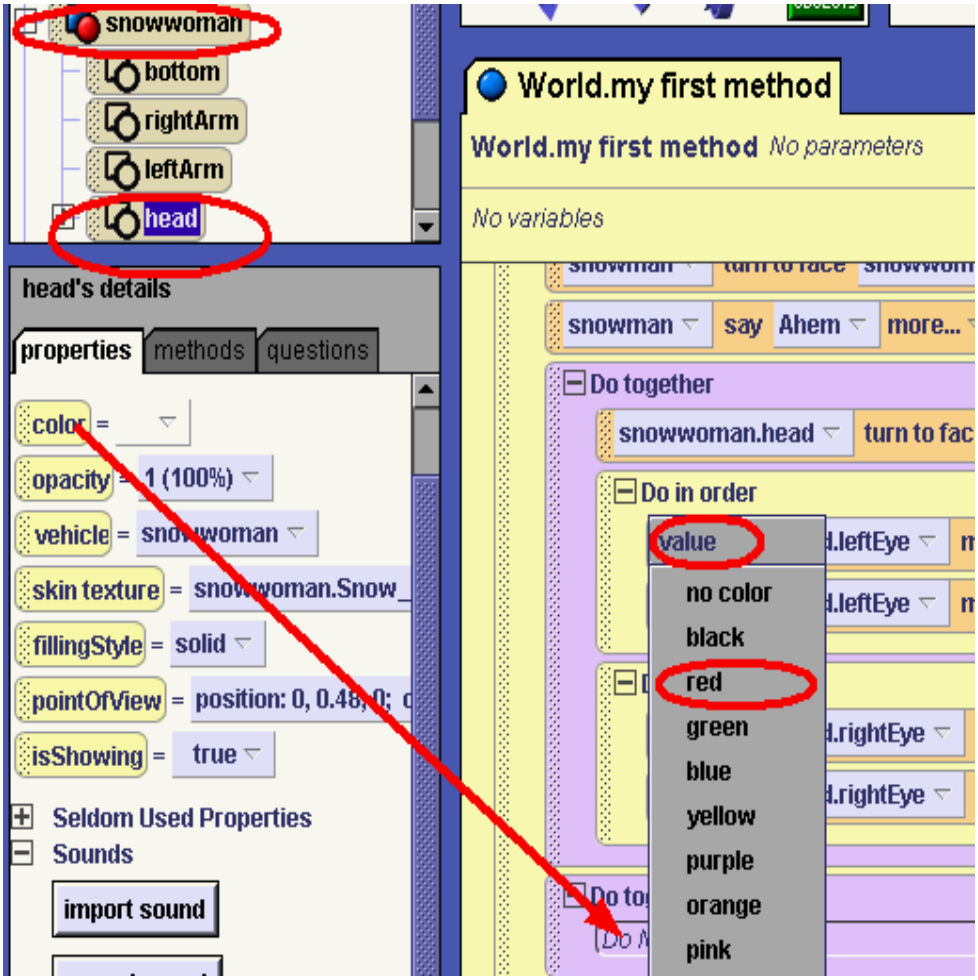

Figure 2-2-19. Changing the color of the snowwoman's head

As the snowwoman blushes, she should turn her head to look again at her friend. So, a *turn to face* instruction is added to the *Do together* block to make the snowwoman's head return to look back at snowwoman2. Finally, the last instructions are written to return snowwoman's head to a white color and then the snowman gives up (hangs his head in disappointment and turns away). The resulting code for the entire animation is listed in Figure 2-2-20.

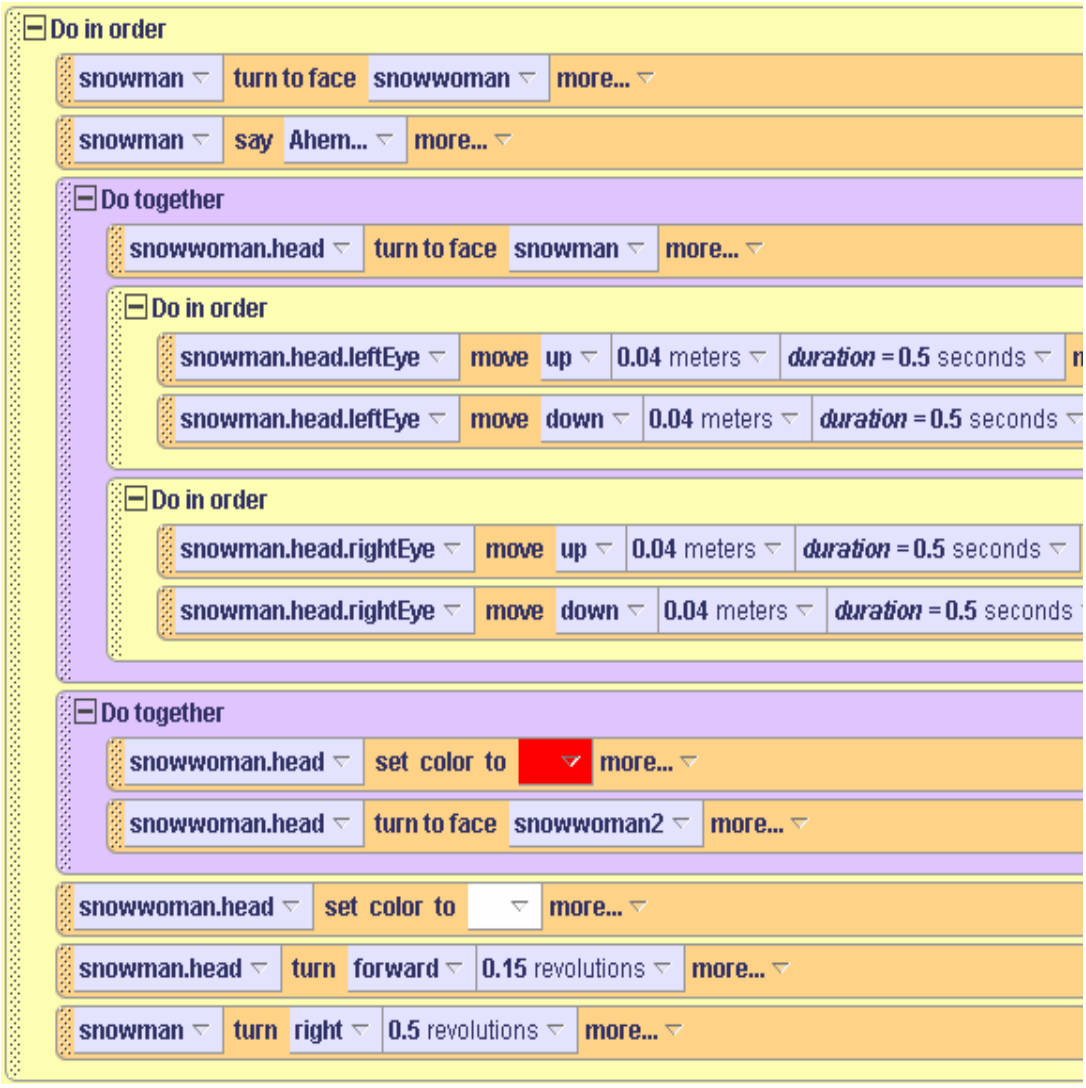

Figure 2-2-20. The program code for the entire snowpeople animation

Comments

Now that we have written our first program, it is time to look at a useful component in programs – comments. Comments are NOT instructions that cause some action to take place. This means that Alice can ignore comments when running a program. However, comments are considered good programming "style" and are extremely useful for humans who are reading a program. Comments help the human reader understand what a program does. This is particularly helpful when someone else wants to read your program code to see what you wrote and how you wrote it.

Comments in Alice are created by dragging the green // tile into a program and then writing a description of what a sequence of code is intended to do. Figure 2-2-22 illustrates *World.my first method* with a comment added. Where it is not obvious, a comment should be included at the beginning of a method to explain what the method does. This kind of comment is like writing a topic sentence in a paragraph – it summarizes what is going on. Also, small sections of several lines of code that collectively perform some action can be documented using a comment. An additional comment has been added in Figure 2-2-23. This comment explains that this small section of the code is to have the snowwoman blush and turn away.

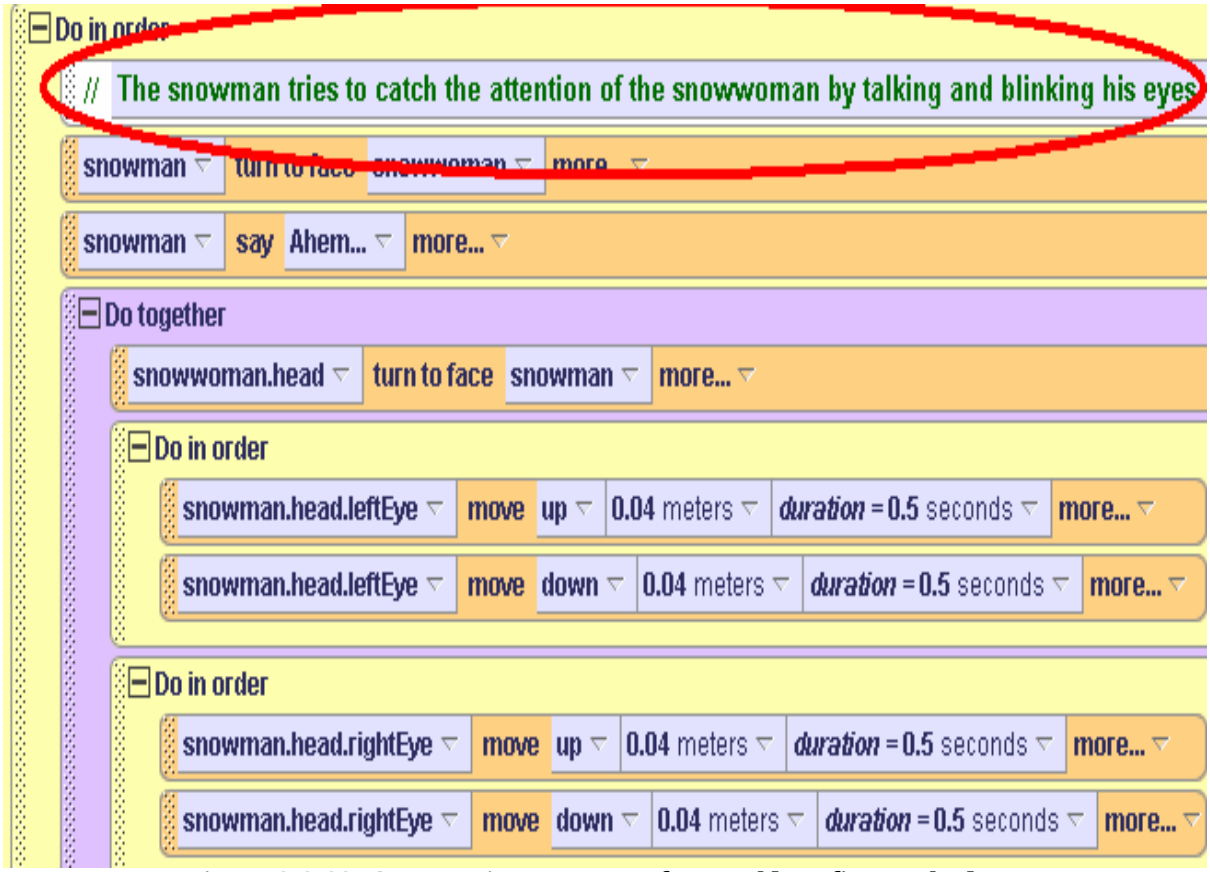

Figure 2-2-22. An overview comment for *World.my first method*

Figure 2-2-23. A comment for a small section of code