Chapter 2

Methods

A Guide to this Instructor's Manual:

We have designed this Instructor's Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings you will find: lecture notes that summarize the section, Teacher Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared towards quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor's Manual, our Instructor's Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

There are two different reasons for building your own methods. One reason is to divide your story into smaller pieces to keep it more manageable and organized. A second reason is to provide an object with a behavior it needs but does not already have. In this chapter, you will examine both approaches. Note that the motivations, thought processes, and circumstances are quite different for these two approaches.

Objectives

- Build scene methods to help organize a story into scenes and shots
- Build class methods to elicit desirable behaviors from objects
- Use markers to reposition the camera for different shots within a scene
- Understand how an object's position, orientation, and point of view are determined

Teaching Tips

2.1 Scene and Shot Methods

1. Discuss the difference between scenes and shots and introduce the term **Divide and Conquer**.

2. Note that scenes and shots provide a logical and convenient way to break a big film project down into smaller, more manageable pieces. This approach can be used to program in Alice.

2.1.1 Running Example: Methods for Scenes

1. Use this section to demonstrate the divide and conquer approach.

2. Using Figures 2-1 and 2-2, show students how to create a new method called **`doScene1()`**. Note that method names should usually be a verb or verb phrase and should also be descriptive.

3. One way to make sure that a method is working is to select the **`camera`** in the **object selector** and make it do a barrel roll by sending it a **`roll()`** message within **`doScene1()`**. Use Figure 2-3 to aid the discussion.

4. Note that every message must be sent to an object. Illustrate the steps involved in sending our **Scene** object (**`this`**) the **`doScene1()`** message. Use Figures 2-4 to 2-6 to aid the discussion.

5. Inside each new **`doScene`** method, we can send the camera another **`roll()`** message, or a **`turn()`** message, or some other message to generate a different visual effect.

2.1.2 Running Example: Methods for Shots

1. Students should be aware of the following rule of thumb: If you must use the scroll bar to view all the statements in a scene method, divide it into two or more **shot methods**. Long methods tend to be complicated and error prone, and their length can make it harder to find errors.

2. Explain that Scene 3 of our running example is complicated; according to our design, building it will require five different shots. Using an approach similar to what we did in the last section, illustrate the steps involved in creating the following methods: **`doShot3a, doShot3b, doShot3c, doShot3d`**, and **`doShot3e`**. Use Figures 2-7 to 2-9 to aid the discussion.

3. Introduce the term **framework**.

4. Note that a different, faster way to edit methods that we have defined is to right-click an invocation of that method in the editing area and then choose its name from the context menu that appears.

5. Use Figure 2-10 to illustrate a structure diagram.

6. Note that scene and shot methods reflect the structure of the story we are telling and therefore belong to the program we are building. As such, they are properly stored in the **`Scene`** class, since it represents our program as a whole.

2.1.3 Enabling and Disabling Methods

1. When we have laid out our story's structure and defined methods for each one of its scenes and shots, we can work on specific scenes or shots in isolation by disabling all the parts we are not working on. Illustrate this concept by disabling the **`showTitleScreen()`** message in the **`myFirstMethod()`** method. Use Figure 2-11 to aid the discussion.

2.1.4 Running Example: Scene 1

1. Introduce the user story for Scene 1.

2. Note that introductory scene of a user story is very important, because it has to introduce the story's characters to the viewer.

3. Encourage students to create an algorithm from the user story.

4. Use the techniques presented in Section 1.3.2 to show students how to build the set shown in Figure 2-12.

5. Next, you will need to make the left tree appear stunted, position Ann offscreen, and animate the scene.

6. Discuss the issues involved in animating the scene - the camera needs to zoom in at the beginning; Ann is supposed to walk on-screen near the beginning of the scene and off-screen at the end; and Tim does not have a method to cast a spell that will make a tree grow.

7. Introduce the term **comments**. Use Figure 2-13 to aid the discussion.

2.2 Alice Tip: Using Markers

1. Pose the following question to students: How do we move the camera from one scene to another, or from one position to another within a scene?

2. Note that because the **`camera`** is an Alice object, most of the standard Alice messages can be sent to it. So we could use a set of simultaneous **`move()`**, **`turn()`**, and other motion-related messages to position the camera at the beginning of each scene or shot method.

2.2.1 Markers

1. Every Alice object has a **position** and an **orientation** in the 3D world. To save the position and orientation of an object, Alice provides special invisible objects called **markers**.

2. Use the information on page 52 to discuss the steps involved in using markers to save the position and orientation of an object.

3. The following topics should also be discussed:
   - **Camera Markers**: Illustrate the steps on page 52 by using them to position the **`camera`** for the first scene of our running example. Use Figures 2-14 to 2-16 to aid the discussion.
   - **Object Markers**: Just as we have used the **scene editor** to create a marker for positioning the camera at the beginning of a scene, we can also use the **scene editor** to create a marker for positioning characters using the **Add Object Marker. . . .** button under the **Object Marker**s area. Illustrate the steps involved.

2.2.2 Scene 2: The Ogres

1. Introduce the user story for scene 2 and encourage students to create the algorithm.

2. Use the storyboards to build a set for this scene. Note that there are two approaches:
   - Use the blue-arrow controls to move the camera somewhere in our world that cannot be seen from Scene 1 and build the scene there.

- For smaller projects, another approach is to right-click the camera in the **object tree** and use the **procedures** choice from the context menu to send the camera a **`turn()`** message, giving it an angle just large enough to ensure that everything from Scene 1 is off-camera. Use Figure 2-17 to aid the discussion.

3. Point out that regardless of the approach used, once we have our camera positioned appropriately for a new scene, we can drop a marker there and then use the **`moveAndOrientTo()`** message at the beginning of that scene's method to position the camera appropriately.

4. Use Figure 2-18 to discuss the steps involved in building the set for Scene 2.

5. Illustrate the steps involved in animating the scene. Use Figures 2-19 to 2-21 to aid the discussion.

Quick Quiz 1

1. True or False: A method name should usually be a verb or verb phrase.
   Answer: True

2. True or False: Method names should begin with an uppercase letter and contain no spaces.
   Answer: False

3. In Alice 3, methods that return no values are called _____.
   Answer: procedures

4. To save the position and orientation of an object, Alice provides special invisible objects called _____.
   Answer: markers

2.3 Class Methods for Object Behaviors

1. Explain that an alternative to the scene method is the **class method**, which is used to define a behavior for a given type of object.

2.3.1 Beginning Scene 3

1. Introduce the user story for scene 3 and illustrate the steps involved in setting up the scene. Note that we have to (1) get the ogres into the right positions for them to emerge from the ground, and (2) hide Tim. Use Figure 2-22 to aid the discussion.

2. Illustrate the steps involved in animating the scene. Use Figure 2-23 to aid the discussion.

2.3.2 Building a Class Method

1. Methods are stored within classes, so to send a message to an object, we must create a method in that object's class. In this case, our object is **`ogre3`**, whose class is Ogre. Illustrate the steps involved in creating a method in class **`Ogre`**. Use Figures 2-24 and 2-25 to aid the discussion.

2. Illustrate the steps involved in creating a new procedural method. Use Figure 2-26 to aid the discussion.

3. In order for our ogre to make a "shush" gesture, we need to make his shoulder joint rotate ahead a little less than a quarter of a rotation, make his elbow joint rotate ahead about a quarter of a rotation, and then make all but one of his finger joints rotate ahead a quarter of a rotation. Then, after a short delay, we should undo those actions. Use Figure 2-27 to aid the discussion.

4. Introduce the **`straightenOutJoints()`** message, which returns all of an object's joints back to their original positions and orientations

5. The following topics should also be discussed:
   - **Animation Style**: Note that many of Alice's standard statements provide four different **animationStyle** options under the **add detail** button: BEGIN_AND_END_ABRUPTLY, EGIN_GENTLY_AND_END_ABRUPTLY, BEGIN_ABRUPTLY_AND_END_GENTLY, and BEGIN_AND_END_GENTLY. The **animationStyle** you choose should depend on the complexity of the behavior you are animating. Discuss the guideline that should be followed.
   - **Testing**: Illustrate the steps involved in testing the method. Use Figure 2-28 to aid the discussion. Note that to make Tim materialize at the end of the shot, we set his **Opacity** property to 1.0, using the **`setOpacity()`** method. Introduce the term **abstraction**.

Teaching Tip	A class method defines a message that can be sent to *any* instance (object) of that class.

2.3.3 Building Scene 3, Shot B

1. Introduce the user story for Shot 3b.

2. Illustrate the steps involved in manually positioning the camera and marker. Use Figure 2-29 to aid the discussion.

3. Use Figure 2-30 to illustrate the steps involved in animating the shot.

2.4 More Class Methods

1. Practice creating class methods by opening up a new world. Add a couple of people to it from class **`Biped classes > Adult`**, one male and one female. Call them Alex and Bess; add an object from class **`Biped classes > StuffedTiger`** named Hobbes. Outfit the humans in exercise clothing, as shown in Figure 2-31.

2.4.1 Example 1: Bowing

1. Suppose that Scene 1 of our user story begins with Alex bowing to Hobbes, then Hobbes bows back, and finally Bess bows to the camera. Illustrate the steps involved in building a method to achieve this behavior (see Figures 2-32 to 2-36). Note that the following statements should be performed:
   - **`alex.bow();`**
   - **`hobbes.bow();`**
   - **`bess.bow();`**

2.4.2 Example 2: Signalling "OK"

1. Suppose that after doing some exercises in Scene 2, you want Scene 3 to have Bess ask, "Is everyone OK?" to which Hobbes replies, "I'm OK," and then Alex signals that he's okay using a hand gesture. Use this section to illustrate the steps involved in creating a **`signalOK()`** method that animates the "OK" hand gesture. Use Figures 2-37 to 2-41 to aid the discussion.

2.5 Alice Tip: Reusing Your Work

1. Software developers like to say work smarter, not harder. One way to work smarter is to invest time in designing your algorithms. Use examples to aid the discussion.

2. Note that another way to work smarter is to reuse work that you have already done. The idea is that if you find yourself repeating actions that you have done previously, you can reuse what you did earlier by copying it and adapting the copy to the new situation.

2.5.1 Using the Clipboard

1. Explain that the clipboard icon located in the upper-right corner of the Alice window provides a way to work smarter in some situations. For example, if you drag a statement from the editing area to the clipboard, Alice cuts that statement from the editing area.

2. The following topics should also be discussed:

- **Saving Yourself Time: Similar Operations**: Show students how to use the clipboard to save time when two algorithm steps or messages are similar. Use Figures 2-20, 2-28, 2-42, and 2-43 to aid the discussion.
- **Dividing a Big Method into Smaller Methods**: Discuss the steps involved in splitting a scene method into two or more shot methods. Use the example on page 78 to aid the discussion.

2.6 Thinking in 3D

1. Note that every object in a 3D world has the following two properties: position and orientation.

2.6.1 An Object's Position

1. Introduce the term **axis**. Note that every Alice object has its own three axes, and the point where they meet is called the object's **pivot point**. Use Figure 2-46 to aid the discussion.

2. Explain that an object's position within a three-dimensional world consists of three numeric values—*lr, ud*, and **fb**—that specify its location measured using the world's three axes.

3. The following topic should be discussed:
   - **Changing Position**: To change an object's position, Alice provides the **`move()`** method. When we drop Alice's **`move()`** method into the **editing area**, Alice displays a menu of the directions the object may move, shown in Figure 2-47.

2.6.2 An Object's Orientation

1. Note that when an object moves, turns, or rolls, its axes move, turn, or roll with it. For example, if we send the dolphin of Figure 2-45 the message **`turn(RIGHT, 0.25)`** and then click the dolphin, the picture changes to that shown in Figure 2-48.

2. The following topics should be discussed:
   - **Yaw**: Explain that if you compare the axes in Figure 2-45 and Figure 2-48 carefully, you will see that a **`turn(RIGHT, 0.25)`** message causes the dolphin to rotate about its UD axis. A **`turn(LEFT, 0.25)`** message causes a rotation about the same axis, but in the opposite direction. Introduce the term **yaw**.
   - **Pitch:** Note that because an object has three axes, it is possible to rotate an object around either of its other two axes. Introduce the term **pitch** and use Figure 2-50 to aid the discussion.
   - **Roll:** An object can also rotate around its FB axis. Introduce the term roll and use Figure 2-51 to aid the discussion.

Teaching Tip	An object's orientation is its combined yaw, pitch, and roll.

2.6.3 Point of View

1. Note that in Alice, an object's combined position and orientation is called that object's point of view. An object's point of view thus consists of six values: [(*lr, ud, fb*), (yaw, pitch, roll)]. Use the table on page 86 to aid the discussion.

Quick Quiz 2

1. From the editing area, you can drag any statement onto the _____ and Alice will store a copy of it there for you.
   Answer: clipboard

2. An object's exact location with respect to the world's axes is called its _____.
   Answer: position

3. In 3D terminology, an object's _____ is how much it has rotated about its UD axis from its original position.
   Answer: yaw

4. An object's _____ is how much it has rotated about its LR axis from its original position.
   Answer: pitch

Class Discussion Topics

1. How does decomposing a user story into scenes and shots help you organize the components of an Alice program?

2. What is the difference between a scene method and a class method?

3. What is the value of adding comments to a program?

4. What is meant by the assertion that an Alice object has six degrees of freedom?

Additional Projects

1. Refer to the structure diagram in Figures 2-10 for this problem. Expand the structure in Figure 2-11 to include the following elements:
   - Three Shots for Scene 4

- Three pieces for Shot 2 of Scene 2

Implement and test the revised structure.

2. Suppose that Scene 1 of our user story begins with Alex waving to Hobbes, then Hobbes waving back, and finally Bess waves to the camera. Build a method to achieve this behavior.

Additional Resources

1. Alice Developer Community: www.alice.org/community/

2. Camera Markers: http://www.alice.org/3.1/Materials/Videos/11.CameraMarkers.mp4

Key Terms

- **abstraction:** creating a new method to perform a behavior.
- **axis:** pair of opposite-facing arrows that represent a spatial dimension.
- **class method:** used to define a behavior for a given type of object.
- **class structure diagram:** hierarchical diagrams that depicts class relationships.
- **comment:** explanatory statements that are not compiled by Alice.
- **divide and conquer:** methodology for solving a "big" problem by (1) breaking it into "small" problems, (2) solving each "small" problem, and (3) combining the "small" problem solutions into a solution to the "big" problem.
- **markers:** special invisible objects.
- **framework:** structure within which you can place the animations for specific scenes.
- **inherit:** one class inherits the functions, or properties behavior of another class.
- **object method:** message used to define a complex behavior for a single object.
- **orientation:** the direction that an object is facing.
- **pitch:** the amount of an object's rotation about the LEFT-RIGHT (LR) axis.
- **point of view:** an object's position and orientation.
- **position:** an object's location in the world.
- **reusable object:** an object that can be used in multiple worlds.
- **roll:** the amount of an object's rotation about the FORWARD-BACKWARD (FB) axis.
- **scene method:** method stored in the Scene class.
- **scene:** part of a user story.
- **shot:** part of a scene that is filmed from a particular camera position.
- **subclass**: derived from a main class.
- **superclass**: a class that has been extended by another class.
- **yaw:** the amount of an object's rotation about the UP-DOWN (UD) axis.