After finishing this chapter, you should be able to:

- Provide a brief definition of each the following terms: American Standard Code for Information Interchange (ASCII), AND, Boolean algebra, Boolean function, Boolean logic, collate, collating sequence, logical comparison operators, NOT, OR, proximity function, and Unicode.
- Describe the function of the Boolean operations AND, OR, and NOT, and show how they are used in Alice branching and looping instructions.
- List and describe the function of the six logical comparison operators, and show how they are used to form Boolean conditions in Alice.
- Describe what a Boolean function is and how such functions are used in Alice.
- Create new methods in Alice that use Boolean conditions involving random numbers, Boolean functions, and method parameters.
Chapter 5 • Boolean Logic in Programming

BOOLEAN LOGIC

Branching and looping routines both contain conditions that are either true or false. In 1854, George Boole, the first Professor of Mathematics at Queen’s College in Cork, Ireland, published a book titled “An investigation into the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities.” Boole outlined a system of logic and a corresponding algebraic language dealing with true and false values. Today that type of logic is called Boolean logic, and his language is called Boolean algebra. The conditions that exist in branching and looping routines are a form of Boolean logic.

NOTE  Boolean Logic is the basis for all modern digital electronic technology. The howstuffworks.com Website has an article about how computers implement Boolean logic at http://computer.howstuffworks.com/boolean.htm. In 1858 Boole’s original book was republished as “An Investigation of the Laws of Thought”. Copies of the 1973 reprint of this edition can still be found in many bookstores and online. The complete text of an 1848 paper by Boole titled the Calculus of Logic, is available on the Web at: http://www.maths.tcd.ie/pub/HistMath/People/Boole/CalcLogic. The site is maintained by the University of Dublin's Trinity College School of Mathematics. They also maintain links to information about George Boole at http://www.maths.tcd.ie/pub/HistMath/People/Boole.

Boolean logic is a form of mathematics in which the only values used are true and false. There are three basic operations in Boolean logic – AND, OR, and NOT, as described in Figure 5-1.

FIGURE 5-1: The Boolean AND, OR and NOT operations

<table>
<thead>
<tr>
<th>AND</th>
<th>OR</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>true and true = true</td>
<td>true or true = true</td>
<td>not true = false</td>
</tr>
<tr>
<td>true and false = false</td>
<td>true or false = true</td>
<td>not false = true</td>
</tr>
<tr>
<td>false and true = false</td>
<td>false or true = true</td>
<td></td>
</tr>
<tr>
<td>false and false = false</td>
<td>false or false = false</td>
<td></td>
</tr>
</tbody>
</table>

The AND and OR operations are binary operations, meaning that they need two operands. Basically, when two values are combined in the AND operation, the result is true only if both values are true. Otherwise, the result is false. In the OR operation, if either value is true, then the result is true.

The NOT operation is a unary operation, which means that it works on only one operand. It simply reverses the true or false value of its operand. In other words, NOT true yields false; NOT false yields true.
Comparing Values

The use of AND, OR, and NOT should be common sense. Statements like “today is Monday AND this is March” can be evaluated for their true or false value; yet people sometimes run into trouble converting the informality of human language into the formality needed for algorithms. Consider the following conversation:

**BOSS:** Give me a list of all our offices in Pennsylvania and New Jersey.

**PROGRAMMER:** Let me get this straight — you want a list of our offices located in either Pennsylvania or located in New Jersey, right?”

**BOSS:** Yes, isn’t that what I just said?

The programmer, who has experience converting the informality of human language into a formalized computer programming language, knows what would happen if the condition (state = “PA” AND state = “NJ”) were to be used to create the list. If each office is located in only one state, then both conditions can not possibly be true. What should the programmer do, give the boss a blank sheet of paper? Tell the boss the request is nonsense according to the rules of Boolean logic? No, neither reply is acceptable; the programmer’s only acceptable response was the one that was used to clarify the boss’s request.

Boolean expressions can become long and complex with many nested AND, OR, and NOT clauses layered together. Professional programmers often use Boolean algebra and other formal tools when dealing with the layered complexities of Boolean logic. For the purpose of this text, it won’t be necessary to learn Boolean algebra, which is usually included in courses with titles like “Discrete Mathematics” or “Computer Math and Logic.” However, do note that computer science students are usually required to complete several such courses.

**COMPARING VALUES**

Often the Boolean conditions in branching and looping routines are based on expressions that compare values. Consider the following warning message, which might be found in the documentation for a modern automobile:

*The passenger-side air bag may cause injury to children who are under the age of 12 or who weigh less than 48 pounds. They should not sit in the front passenger seat of this car.*

The condition in this warning might be expressed in pseudo-code like this:

```
IF (age < 12 OR weight < 48)
THEN do not sit in the front passenger seat
```

Two items, each with a true or false value, are joined by the OR operation. The two items are each comparisons of values. The condition “age < 12” is either true or false, as is the condition “weight < 48”. When the algorithm is executed, the true or false value of each of
these conditions is determined, then the overall true or false value is determined using the rules for the OR operation.

The symbol “<” in the above example stands for “is less than.” There are six such logical comparison operators used in Boolean logic: equals, is not equal to, is less than, is greater than, is less than or equal to, is greater than or equal to. Figure 5-2 shows the symbols most commonly used for these operators.

**FIGURE 5-2: The six logical comparison operators**

<table>
<thead>
<tr>
<th>Condition</th>
<th>In Mathematics</th>
<th>In Computer Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>A equals B</td>
<td>A ≠ B</td>
<td>A = B or A == B</td>
</tr>
<tr>
<td>A is not equal to B</td>
<td>A = B</td>
<td>A&lt;&gt; B or A != B</td>
</tr>
<tr>
<td>A is less than B</td>
<td>A &lt; B</td>
<td>A &lt; B</td>
</tr>
<tr>
<td>A is greater than B</td>
<td>A &gt; B</td>
<td>A &gt; B</td>
</tr>
<tr>
<td>A is less than or equal to B</td>
<td>A ≤ B</td>
<td>A &lt;= B</td>
</tr>
<tr>
<td>A is greater than or equal to B</td>
<td>A ≥ B</td>
<td>A &gt;= B</td>
</tr>
</tbody>
</table>

Notice that several of the computer programming symbols, such as “<>” for “not equals,” are composed of two characters. This doubling of characters is required because modern computer keyboards do not include a single symbol for these comparison operators; this is in contrast to standard algebra, in which the symbol “≠” is often used.

**STRING COMPARISONS**

It’s clear that numbers can be compared according to their value, but what about other data types? Well, other data types, such as character strings, each have their own rules for logical comparisons. Character strings, for example, are compared according to the place in a collating sequence for each character of the string.

Let’s see what all this means. To **collate** means to put a set of items in order. A **collating sequence** is a list that shows the correct order to be used when collating a set of items. The English language alphabet is a collating sequence. It shows us that when putting things in alphabetic order, A comes before B, B comes before C, and so on. Modern computers most often use one of two codes to represent characters in the computer — either the **American Standard Code for Information Interchange (ASCII)**, or a newer code called **Unicode**.

There are other, older codes, but today they are rarely used. These codes can also be used as collating sequences for character string values, just as the English language alphabet can.
Comparing Values

The ASCII code is a set of characters used in computer programming based on the English language. It includes letters, numeric digits, and some “hidden” characters, such as the enter key and the escape key. Each character is given a numeric value, and the binary equivalents of those numeric values are used to store characters in the computer. Unicode is a much larger code, which includes characters for other alphabets, such as Greek, Hebrew, Arabic, and the Cyrillic alphabet used for Russian and Eastern European languages. The ASCII code is now actually a subset of the longer Unicode. For more information on the ASCII code, see [www.webopedia.com/TERM/A/ASCII.html](http://www.webopedia.com/TERM/A/ASCII.html). For more information on Unicode, see [http://www.unicode.org/standard/WhatIsUnicode.html](http://www.unicode.org/standard/WhatIsUnicode.html).

Both ASCII and Unicode are similar to the English language alphabet, except for two characteristics. First, they include characters, like the dollar sign, the decimal point, and the digits 0 through 9, that are not part of the English language alphabet; and second, in both codes, all of the capital letters come before the lowercase letters. In English the words apple, Ball, and cherry are in alphabetic order. According to the ASCII code, the order would be Ball, apple, and cherry because a capital B comes before a small A. Figure 5-3 shows a portion of the ASCII collating sequence.

![FIGURE 5-3: A portion of the ASCII code](image)

Comparisons using character strings are often used when searching and sorting data stored on a computer. While this is very important in data processing and database management, comparisons using character strings in Alice are rare.

BOOLEAN FUNCTIONS

The true and false values in computer programming can also come from Boolean functions. A **Boolean function** is a function that returns a true or false value instead of a numeric value. The *ask a user a yes or no question* function used in the last chapter and shown in Figure 5-4 is an example of a Boolean function. When a method using the function is played, a question will appear on the screen. If the user answers yes, then the function returns the value *true*. If the user answers no, then the function returns the value *false*.
Figure 5-5 shows some spatial relation functions on a seaplane’s functions tab in Alice. Each of these functions is a Boolean function that returns a true or false value.

These and other Boolean functions can be used in any place that a true or false value can be used, such as in an *If* instruction or a *While* instruction, as shown in the `seaplane.fly` method in Figure 5-6. The `seaplane is above water` function in the *While* instruction in this example will return a value of *true*, and the loop will be executed as long as the seaplane is above the water. Within the loop there is an *If* instruction to cause the seaplane to wave its wings whenever it is close to the island. This instruction uses a Boolean function that will return a value of *true* if the seaplane is within 10 meters of the island.
Alice also has comparison functions that can be found on the world’s functions tab; in addition, with the proper presentation, the AND, OR, and NOT functions can be used to build more complex Boolean expressions. Both concepts are shown in Figure 5-7.
With the functions in Figure 5-7, you can build expressions like these:

- If (age < 12 OR height <= 54)
- While (aliceLiddel distance to whiteRabbit <=10)
- While NOT (target = windmill) OR NOT (target = gazebo)
- If NOT (seaplane is in front of camera) AND (time < 10) OR (time > 30)
- While numberOfGuesses <= log(range)

The following tutorials will each provide experience with simple uses of Boolean functions in Alice.

**TUTORIAL 5A—BRANCHING WITH RANDOM NUMBERS**

In this exercise, you will modify the jump user choice world from Tutorial 4A to have the computer randomly select who will jump instead of asking for user input. Alice has a world level function that will return a random number in a specified range. The numbers that it returns are six digit numbers, greater than or equal to the lowest number specified and less than the highest number specified. In this exercise, for example, you will ask Alice to return a number between 0 and 3. Alice will return numbers such as 1.83475, 0.41257, 2.89175, and so on. All of the numbers it returns will be greater than or equal to 0, but less than 3.
In this exercise, if the random number is less than 1, Alice will jump. If it is greater than or equal to 1 but less than 2, then the White Rabbit will jump. If it is greater than or equal to 2 but less than 3, then the Cheshire Cat will jump. Figure 5-8 shows the expected results of the program. The logic is very similar to the user input program in Tutorial 4A, with nested If/Else instructions to determine which character will jump.

You will add a variable to the program to hold the random number. Figure 5-9 shows what the code for the method will look like after you do this. You can refer to this image to guide you through the next several steps.
1. Open the **jump user choice** world saved as part of Tutorial 4A, or follow the steps in Tutorial 4A to create the world, or load the world from the CD that comes with this book.

2. Click the **create a new variable** button on the right side of *world.my first method* to add a variable to hold the random number. Name the variable **determinant** (because it will determine who jumps), select **Number** as the type, and then click **OK**. You will now see a variable tile for **determinant** appear at the top of the method.

3. Next, you need to tell the computer to set the value of **determinant** to a random number. Drag the **variable** tile down into *world.my first method* as a new first instruction before the **If/Else** instructions. When you do this, a set value dialog box will appear. Set the value of **determinant** to 1. The 1 is only a place holder; we will change it to a random number in the next step.

4. Now you need to tell the computer to set the value of the variable **determinant** to a random number instead of a 1. To do this, select the **world** tile in the Object tree, and then click the **functions** tab in the Details area. Scroll through the list of functions and find the function titled **random number**. Drag and drop a copy of this function into the **set value** tile in place of the number 1. The **set value** tile should now look like the **determinant set value to random number** tile near the top of the method shown in Figure 5-9.

5. You now need to tell the computer what range of values to use when picking a random number. To do so, click **more** in the blue **random number** tile, and set the **minimum** value to 0. Click **more** again, and this time set the **maximum** value to 3. Now the **set value** tile in your method should look like it does in Figure 5-9.
6. The nested *If/Else* instructions still contain conditions based on the user question. You need to replace these conditions following the word *If* in each of the nested *If/Else* instructions with conditions based on the random number. To do so, drag the *determinant* variable tile from the top of the method down into the first *If/Else* instruction in place of the blue *ask user yes or no* condition tile. When you do this, a short menu will appear. Select *determinant* < and 1 as the value. Now the first *If/Else* instruction tile should look like it does in Figure 5-9.

7. Do the same thing for the second *If/Else* instruction, but this time, choose 2 as the value. When you are done, the second *If/Else* instruction should be *if determinant* < 2.

8. The method should now be ready to select a random number and make one of the three characters jump based on the value of the random number. Save the world as *triple jump random*, and you are ready to test the world.

You need a testing plan for your modified Alice world. If things work correctly, then one of the three characters should jump each time the world is run. Which character jumps depends on the value of the random number that the computer picks for the variable you named *determinant*.

To test the program properly, you would need to run it several hundred or several thousand times, and keep track of how many times each character jumps. Over a long period of time, we would expect each character to jump about one third of the time the program runs. This will be left as an exercise at the end of the chapter. For now, test the program to make sure that the same character does not jump all the time, and that each of the three characters jumps at least part of the time. Play the world, and then use the restart button several times to replay the world. Because the numbers are random and not part of a pattern, the same character may jump several times in a row, but over a larger number of trial runs, each of the characters should jump at least once.

**TUTORIAL 5B—THE NERVOUS PENGUIN**

In this exercise, you will create a world with two penguins. The first penguin will be controlled by the user and may move around freely in the world. The second penguin, the nervous penguin, will flap its wings and jump up and down whenever the first penguin gets close to it. A Boolean *proximity function* returns a value based on the distance from one object to another. In this tutorial, a proximity function will be used to cause the nervous penguin to react.
A proximity function returns a value of true or false, but a numeric proximity function returns the distance between two objects.

1. Start a new Alice world using the snow template.
2. Click the green ADD OBJECTS button to enter Scene Editor mode, and then add a penguin to the world from the Animals folder in the Local Gallery.
3. Move the penguin to the left side of the screen and push it back away from the viewer.
4. Add a second penguin to the world. Alice will give it the name penguin2. Right-click the penguin2 tile in the Object tree and rename it as nervousPenguin.
5. Move nervousPenguin to the right side of the world window. The two penguins should now be positioned something like those in Figure 5-10.

6. Now the scene is ready. Click the large green DONE button to exit, and save the world with the name nervous penguin.

The world will need one method and several events. The method will be the nervous penguin’s reaction to the other penguin getting too close. Figure 5-11 shows the method you will need to create.
1. Select the `nervousPenguin` tile in the Object tree and the methods tab in the Details area.

2. Click the `create new method` button. When the create new method window appears, type the name `react`, and then click the `OK` button.

3. The nervous penguin should jump up and down and flap its wings when the other penguin gets too close. These two actions—jumping and flapping—should happen at the same time. Drag a `Do together` tile from the bottom of the Editor area and drop it in the new method in place of `Do Nothing`.

4. Drag a `nervousPenguin jump times` tile from the methods tab and drop it in the `Do together` instruction in place of `Do Nothing`. When the short menu appears, select `2` as the number of times.

5. Drag a `wing_flap times` tile from the methods tab and drop it in the `Do together` instruction below the `nervousPenguin jump times = 2` instruction. When the short menu appears, select `2` as the number of times. The new method is now complete and should resemble Figure 5-11.

Now it’s time to create several simple events that will be needed for this world. You will create methods to provide user controls for the moving penguin, to animate the moving penguin, and to cause the nervous penguin to react when the other penguin is near. It would also probably be a good idea to create a method to let the user point the camera at the moving penguin in case it moves off camera. Figures 5-12 shows these new events.
A built-in event can be used to provide control of the moving penguin. A second event can be created to use the `penguin walk move_time` method to animate the penguin. The default event can be modified to serve as this second event.

1. Click the **create new event** button in the Events area. When the menu of events types appears, select **Let the arrow keys move <subject>** as the event type.

2. A new arrow key event will appear in the Events area, as shown in Figure 5-13. Click **camera** in the event tile and select **penguin, the entire penguin**. The event is now set to let the user move the penguin with the arrow keys.

3. Right-click the **When the world starts, do world.myfirst method** default event. When a short menu appears, select **change to, While the world is running**.
4. Select the **penguin** tile in the Object tree and the methods tab in the Details area. Drag a copy of the **penguin walk move_time** method and drop it in place of **None** following **During**: in the new event. Select 1 from the short **move_time** menu that appears. The event should resemble Figure 5-14. The penguin will now shuffle its feet and waddle even when standing in place.

The first two events are complete. The user can now move the penguin with the arrow keys, and the penguin will waddle and move its feet while the world is running, even when standing in place.

Next, you will add an event to allow the user to point the camera at the penguin in case it moves off camera.

1. Click the **create new event** button in the Events area. When the menu of events types appears, select **When a key is typed** as the event type. A new **When any key is typed do Nothing** event tile will appear in the Events area.
2. Click any key and select **Space** from the menu that appears. Your actions will make the spacebar the trigger key for this event.
3. Select the **camera** tile in the Object tree and the **methods** tab in the Details area. Drag a copy of the **camera point at** method tile from the methods tab and drop it in place of **Nothing** in the new event. Select **penguin, the entire penguin** from the menu that appears. The event to allow the user to point the camera at the moving penguin is now complete, as shown in Figure 5-15.

The last event that is needed is the event to cause the nervous penguin to react whenever the moving penguin gets too close. The event trigger will have a Boolean function that will return a value of **true** if the first penguin is within 2 meters of the nervous penguin. The
event handler will be the `nervousPenguin.react` method. Figure 5-16 shows what the finished event will look like.

![Figure 5-16 The event to cause the nervous penguin to react](image)

1. Click the create new event button in the Events area. When the menu of events types appears, select **When something is true** as the event type. A new **When None is true** event tile will appear in the Events area.
2. Select the penguin tile in the Object tree and the functions tab in the Details area.
3. Find the `penguin is within threshold of object` tile, as shown in Figure 5-16. Click and drag the `penguin is within threshold of object` tile and drop a copy of it in the new event in place of *None* following the word *When*. Select **2 meters**, **nervousPenguin**, and **entire nervousPenguin** from the menus that appear. The trigger for the event is now complete.
4. Select the `nervousPenguin` tile in the Object tree and the methods tab in the Details area; drag a copy of the `nervousPenguin react` method and drop it in place of *None* following **Begin**: in the new event.
5. The final event is in place. Save the world again before continuing.

The nervous penguin world is complete and ready for testing. You should be able to move the penguin around using the arrow keys. Be careful—it moves rather quickly. You should be able to point the camera at the penguin using the space bar. The nervous penguin should jump up and down and flap its wings whenever the first penguin gets within two meters of it.

**TUTORIAL 5C—A SENTINEL SAILING LOOP**

In this exercise you will create a world with a sailboat and several objects. When the world starts, you will be able to click an object, and the sailboat will sail to that object. You will create a `sail to` method that uses a sentinel loop to make the sailboat sail toward the island. (A
sentinel loop is one that needs user input to execute.) The method will use a Boolean condition with a method parameter in the condition. The loop will have the following logic:

```
WHILE (NOT (the sailboat is within 5 meters of the [object]))
{
    turn to face [object]
    move forward 1 meter
}
```

The brackets “{“ and “}” are used to mark the beginning and end of the block of code within the loop. When this code is executed, the sailboat will turn to face a target object, then move 1 meter toward the object. It will continue to do so until it is within 5 meters of the object. You will write a sail to method that will work with any object and that will accept the target object as an input parameter, much like the way the primitive move method accepts direction and amount as parameters.

You will create your world in two steps. First, you will place an island in the world and hardcode the island as the target for the sail to method. (To hardcode a value in a program means that the programmer puts a specific value in a program instead of a variable or parameter.) Once this method works, you will add an object parameter called target and put it in the method in place of island.

**THE SAIL TO ISLAND WORLD**

Let’s start by setting the scene for the new Alice world. First, you will need a water world with a sailboat in it, and it will be necessary to adjust the camera.

1. Start a new Alice world with a water template.
2. Click the green ADD OBJECTS button to enter Scene Editor mode, and then add a sailboat to the world from the Vehicles folder in the Local Gallery. Your screen should resemble the left side of Figure 5-17.
3. The camera is too close to the sailboat, creating a tight shot in which the boat fills most of the frame. Using the blue camera control arrows at the bottom of the world window, move the camera back from the sailboat and up a little bit so that the world window looks more like the image on the right side of Figure 5-17. You might also want to move the sailboat a little toward the side of the window.
Next, you need to add an island to the world. It’s up to you where to put the island, but it should not be too close to the sailboat, and it should not be directly in front of the sailboat.

1. Add an **island** to the world from the Environments Gallery.
2. Position the island so that it is away from the sailboat and not directly in front of it. You can use Figure 5-18 as a guide.
3. Now click the large green **DONE** button to exit, and save the world with the name **sail to island**.

You are ready to create the new method. In this first version of the world, the sailboat will sail to the island when the world starts. You will need to create a method that is invoked by an event with **when the world starts** as its trigger. The default event does this, but it uses the method `world.my first method` as its event handler. The new method for the default event will be `sailboat.sail to`. Notice that it will be a method for the `sailboat` object and not a world-level method as `world.my first method` is. You can create the new method, change the event handler in the default event, and then delete `world.my first method`. You can delete the method because it won’t be used further in your world.

1. Select the **sailboat** tile in the Object tree and the **methods** tab in the Details area.
2. Click the **create new method** button. When the create new method dialog box appears, type the name **sail to**, and click **OK**. The new method will
appear in the Editor area, and a **sailto** method tile will appear in the Details area.

3. Drag the **sail to** tile from the Details area and drop it in the default event in place of **world.my first method**.

4. Now **world.my first method** is no longer needed. Select the **world** tile in the Object tree and the **methods** tab in the Details area. Click and drag the **world.my first method** tile to the trash can to delete it.

Now it’s time to begin writing the code for the new method. It should contain a **While** loop that matches the pseudo-code shown at the beginning of this tutorial.

Even though we are learning about the sentinel loop in this tutorial, you may have noticed that the process can also be done with events and without a **sail to** method. To add to the fun, you will learn in Chapter 7 how to create a recursive solution to replace the loop in the **sail to** method.

1. Select the **sailboat** tile in the Object tree and the **methods** tab in the Details area.

2. Drag a **While** tile from the bottom of the Editor area and drop it into the **sailboat.sailto** method in the editor area. Select **true** from the short menu that appears.

3. Next, select the **functions** tab in the Details area and find the **sailboat is within threshold of object** tile, as shown in Figure 5-19. This is a Boolean proximity function. Drag a copy of the tile and drop it in the **While** instruction in place of **true**. Select **5 meters** as the distance and **island, the entire island** as the object.

4. The **While** tile now says **While sailboat is within 5 meters of island**.

   However, you want just the opposite. Select the **world** tile in the Object tree and find the **not** tile on the functions tab. Drag a copy of the tile to the **While** tile, and drop it in place of the condition **sailboat is within 5 meters of island**. Now the condition should be as required, shown in Figure 5-20.
The `sailboat.sail to` method now has the correct loop structure and condition in place. Two instructions need to be placed in the event. One to turn the sailboat and one to move the sailboat. These should be executed together, so they will need to be inside a `Do together` tile.

1. Drag a `Do together` tile from the bottom of the Editor area and drop it into the `While` instruction in place of `Do Nothing`.
2. Select the `sailboat` tile in the Object tree and the `methods` tile in the Details area.
3. Drag a copy of the `sailboat turn to face` method tile and drop it in the `Do together` tile in place of `Do Nothing`. Select `island, the entire island` as the object; click `more` and change the style to `abruptly`.
4. Drag a copy of the `sailboat move` method tile and drop it in the `Do together` tile just below the `sailboat turn to face` tile. Select forward as the direction and `2 meters` as the amount; then click `more` and change the style to `abruptly`.

**THE SAIL TO ANY OBJECT WORLD**

Next you will use the sail to island world as the basis for a new world that will allow the user to sail the boat to any object in the world. Several additional objects will be added to the world, an event will be added to allow the user to select a target object, and the `sail to` method will be modified to include a target variable instead of the island. First the world will be saved with a new name so that the sail to island world will be preserved, then several objects will be added to the world.

1. Click **File** on the menu bar, click **Save World As**, and then save a copy of the world with the new name `sail to object`.
2. Click the green **ADD OBJECTS** button to enter Scene Editor mode, and using the blue camera control arrows, pull the camera back a little farther and up a bit more so that you can see more open water. As you complete the following steps, note that it may be necessary to adjust the camera further.
3. Add several objects to the world. As you place the objects in the world, you should move them away from each other, positioning them as if they were on the edge of a large circle. Figure 5-21 shows a world with the following objects:
4. When you are happy with the collection of objects in your world, click the large green **DONE** button to return to Scene Editor mode.

Now you are ready to modify the `sailboat.sail to` method and add a new event. You will also need to delete the default event, which will actually prevent the world from playing with the changes to be made. Figure 5-22 shows the `sail to` method with the modifications you will make. The default event should be deleted first.
1. Right-click the default event in the Events area and select delete from the menu that appears. The event will be deleted.

2. Select the sailboat tile in the Object tree and the methods tab in the Details area; then click the edit button next to the sail to method tile. You should be able to see the sail to method in the Editor area.

3. Click the create new parameter button to add a target parameter to the method. When the create new parameter window appears, type the name target, make sure that Object is selected as the data type, and then click the OK button. A tile for the target object parameter will appear near the top of the method.

4. Drag the target object parameter tile and drop it in the While instruction in place of island.

5. Drag the target object parameter tile again and drop a copy of it in the sailboat turn to face island instruction in place of island. Select target from the short menu that appears.

6. Now you can add the new event. Click the create new event button in the Events area and select When the mouse is clicked on something from the menu of event types that appears. A new event similar to Figure 5-23 will appear.
7. Click and drag the sail to target tile from the methods tab and drop a copy of it in the new event in place of the word Nothing. When the target menu appears, select expressions and then object under mouse cursor. Your new event should now look like Figure 5-24; the modifications to the world are now complete.

8. Save the world before continuing.

You can now test your new world. The sailboat should sail to whatever object you select with the mouse. If you click the water, it will sail toward the center point of the world. If you click the sky background, then the world will stop playing, and Alice will show you an error message telling you that the target value must not be null. This means that there is no value for the target parameter due to the fact that you clicked on nothing.

The sailboat moves rather slowly from object to object in the sail to any object world. You can use the speed slider control in the playing world window to speed things up a bit as you test your new world.

You may also notice that the sailboat will sail through objects to get to its target. To correct this you would need to create methods for collision detection and avoidance to your world, which is rather time consuming; as such, it is not part of this tutorial.
This chapter consisted of a discussion of Boolean logic, including the comparison of values and Boolean functions, followed by three tutorials. The discussion of Boolean logic included the following:

- The process of developing methods for objects is mostly a process of developing algorithms because each method is an algorithm.
- Branching and looping routines both contain conditions that are either true or false. These conditions are a form of Boolean logic.
- Boolean logic is a form of mathematics in which the only values used are true and false.
- There are three basic operations in Boolean logic: AND, OR, and NOT. The NOT operation is a unary operation, which means that it works on only one operand. It simply reverses the true or false value of its operand: NOT true yields false, and NOT false yields true.
- People sometimes run into trouble converting the informality of human language into the formality needed for algorithms.
- Often the Boolean conditions in branching and looping routines are based on expressions that compare values.
- There are six logical comparison operators used in Boolean logic: equals, is not equal to, is less than, is greater than, is less than or equal to, is greater than or equal to.
- Numbers can be compared according to their value, but other data types, such as character strings, have their own rules for logical comparisons.
- Character strings are compared according to the place in a collating sequence for each character of the string. To collate means to put a set of items in order. A collating sequence is a list that shows the correct order to be used when collating a set of items.
- Modern computers most often use one of two codes to represent characters in the computer — either the American Standard Code for Information Interchange (ASCII), or a newer code called Unicode. These codes can also be used as collating sequences for character string values, just as the English language alphabet can.
- The ASCII code is a set of characters used in computer programming based on the English language. It includes letters, numeric digits, and some “hidden” characters, such as the enter key and the escape key.
- Unicode is a much larger code, which includes characters for other alphabets, such as Greek, Hebrew, Arabic, and the Cyrillic alphabet used for Russian and Eastern European languages. The ASCII code is now actually a subset of the longer Unicode.
- A Boolean function is a function that returns a true or false value instead of a numeric value.
Chapter 5 • Boolean Logic in Programming

- Boolean functions can be used in any place that a true or false value can be used, such as in an *If* instruction or a *While* instruction.

- Alice also has comparison functions that can be found on the world’s function tab: the AND, OR, and NOT functions that can be used to build more complex Boolean expressions.

In Tutorial 5A you modified the jump user choice world from Tutorial 4A to have the computer randomly select who will jump instead of asking for user input. You created Boolean conditions that involve random numbers.

In Tutorial 5B you created a *nervous penguin* world with a Boolean function used in an event trigger.

In Tutorial 5C you created two worlds, one to make a sailboat sail to an Island and one to make a sailboat sail to any object. Each world contained a *sail to* method that used a sentinel loop with a Boolean condition in the loop’s *While* instruction.

**REVIEW QUESTIONS**

1. Define the following terms:
   - American Standard Code for Information Interchange (ASCII)
   - AND
   - Boolean algebra
   - Boolean function
   - Boolean logic
   - collate
   - collating sequence
   - logical comparison operators
   - NOT
   - OR
   - Proximity function
   - Unicode

2. Truth tables are often used to describe the result of Boolean expressions. Fill in the correct values in each of the following truth tables. For each cell in the table, apply the operation for the table to the values at the top of the row and at the beginning of the column for that cell, just as you would fill in values on a multiplication table.

<table>
<thead>
<tr>
<th>AND</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. The following three Boolean operations are composite operations, which means they can be written as combinations of AND, OR, and NOT. All three of these are important in designing the logic circuits inside computer chips. Create truth tables similar to those in question 2 for each of these operations.
   a. XOR – A OR B, but not both.
   b. NOR – NOT (A or B)
   c. NAND – NOT (A and B)
4. Parenthesis can be used in Boolean logic to clarify the order of operations, just as in ordinary arithmetic. Assuming that the height of a palm tree is 5 meters, the height of a lighthouse is 25 meters, and the height of a beach house is 10 meters, rewrite the following expression exactly as it is three times, but with parenthesis in different places each time. Determine whether each of the new expressions is true or false.

Height of the beach house is less than the height of the lighthouse and not height of the lighthouse is less than the height of the tree or height of the beach house equals the height of the tree.

5. Boolean conditions are important for library searches search engines. Usually a library search engine has fields such as title, subject, author, year, , and publisher. A typical search condition would be something like (author = “Lewis Carrol” and year < 1865). Write Boolean conditions for searches to find books about each of the following:
   a. Martin Luther King, written between 1968 and 1978
   b. About Martin Luther King, written before 1968, but not by Martin Luther King
   c. Microsoft Excel, published by Course Technology or accounting, published by Delmar, after 1998
   d. Careers in nursing or health care or medicine

6. Boolean conditions are also important in Internet searches. Write Boolean conditions for searches to find web pages about each of the following:
   a. John D. Rockefeller and Ida Tarbell
   b. Abraham Lincoln and Jefferson Davis, but not schools or parks
   c. North Dakota or South Dakota but not Dakota Jones or Dakota Smith
   d. Cooking a turkey, but not in a microwave or deep fat fryer
   e. George Boole and either Charles Babbage or Herman Hollerith

7. Which of the following does not contain a valid Boolean expression, and why?
   a. If (distance to alice < distance to caterpillar OR queen)
   b. If (count > 10 AND count < 20)
   c. While (count < 5 AND >1)
   d. While height < count OR height < 12.5
   e. If (message = “Hello, World.” OR “Hello, World!”)

8. Write Boolean conditions to make While loops continue to function until the following occurs:
   a. A car door has been opened and closed 1,000 times, or the door falls off the car
   b. A penguin in an Alice world is within 5 meters of an igloo or an Eskimo, or is more than 100 meters away from the camera
   c. A sailboat is moving at a speed between 5 and 10 meters per second

9. Write the following rule as a Boolean expression:
   I before E except after C, or when sounding as A as in “neighbor” or “weigh.”
10. In most states in the US, a vehicle must stop at an intersection with a traffic light when the traffic light is red, except when making a right turn, in which case the driver must pause and can then make a right turn on red if the way is clear, unless otherwise posted. At a particular intersection there is a sign that says, “No right turn on red, 6 am to 6 pm, except Saturdays and Sundays.” Write the set of rules and conditions for this intersection as an If/Else instruction with a single Boolean expression.

EXERCISES

1. Assume that we have an Alice world similar to the nervousPenguin world, but with a strangePenguin and a motherPenguin as well as the nervousPenguin. Write code for Boolean conditions that could be used to trigger events to make the nervousPenguin react in each of the following situations:
   a. The strangePenguin is within 2 meters of the nervousPenguin and the motherPenguin is more than 2 meters away
   b. The strangePenguin is within 2 meters of the nervousPenguin or the motherPenguin is more than 2 meters away
   c. The strangePenguin is closer than the motherPenguin


3. Information about leap years can be found on the U.S. Naval Observatory’s Website at: http://aa.usno.navy.mil/faq/docs/leap_years.html. Write the rule for determining if a year is a leap year as a single Boolean expression.

4. Modify the triple jump world created in Tutorial 2C to include instructions to make whichever object is closest to the camera jump. Your world should work properly no matter where each object is positioned in the world. Test the world several times by moving the objects around in Scene Editor mode and then playing the world.

For exercises 5, 6, and 7, create a new Alice world with a grass template and five objects from the people gallery: aliceLiddel, blueBallerina, handsomePrince, maleBalletDancer, pinkBallerina

Exercises 5, 6, and 7 will refer to this world as the dancer world.

5. Using the dancer world described above, create a method to print the height of each character with the print instruction and the height function for that character. Using the information shown when the world is played, determine which of the following are true:
   a. aliceLiddel is taller than handsomePrince or shorter than blueBallerina
   b. blueBallerina is taller than pinkBallerina or shorter than pink Ballerina
   c. maleBalletDancer is shorter than handsomePrince and taller than blueBallerina
   d. aliceLiddel is (shorter than pinkBallerina and shorter than maleBalletDancer) or (taller than Blue Ballerina)
   e. aliceLiddel is (shorter than pinkBallerina) and (shorter than maleBalletDancer or taller than Blue Ballerina)
6. Using the dancer world, create a method that will allow the user to select any two of the objects, and will then cause the taller of the two objects to spin. If they are the same height, then both should spin. Hint: An object variable can be used to store which object a user has selected.

7. Using the dancer world described create a method that will allow the user to select any three of the objects, and will then cause the tallest of the three objects to spin. If more than one is the tallest, then the first of the tallest objects selected should spin.

8. Sometimes complicated Boolean expressions can be simplified, or rewritten in a different form. One rule that explains how to do this is DeMorgan’s Law. See if you can find information on the Internet about DeMorgan’s Law and then use that information to rewrite each of the following using only one NOT operation:

   NOT(today is Monday) OR NOT (today is Wednesday)

   NOT(subject = History) AND NOT (subject = Biology)

9. The logic in the following code can be implemented with a single event with a compound Boolean condition, or with two events with simple Boolean conditions. What are the advantages and disadvantages of each approach? When can separate events be combined into one event, and when must separate events be kept as separate events?

   IF (mouse is clicked on island OR mouse is clicked on pier)
   THEN sail to pier

10. Create an Alice “guessing game” world to do the following. The world should have two characters of your choice. It might be best to create a flowchart or pseudo-code to help design your world.

   a. Pick a random number X, such that 1 <= X < 100. In the instruction to pick the random number, click “more” and select “integer only = true” so that the random number tile looks like something this:

      [X set value to random number (minimum = 1 maximum= 100) integerOnly = true]

   b. Ask the user to guess the number. Alice has a world level function to ask the user for a number.

   c. Have one of the characters tell the user if the guess is too low. Have the other character tell the user if the guess is too high.

   d. Set up a sentinel loop to repeat the process while the user’s guess is not equal to the number the computer picked.

   e. Have both characters tell the user when the guess is correct and react, such as with a dance.