CHAPTER 1

Objects

CHAPTER OBJECTIVES

In this chapter you will:

- Be introduced to Alice and Java and how they're used in this book.
- Create virtual worlds in Alice.
- Call methods on objects to make them behave in particular ways (such as moving and turning).
- Explore the set of methods that are available to all Alice objects.
- Set and modify the properties of an object.
- Create new objects from predefined classes.
- Cause multiple animation actions to occur at the same time.
- Explore composite objects (which are made up of other objects) and interact with a composite's individual parts.

Welcome to *Programming with Alice and Java*. If you're concerned at all about venturing into the world of computer programming, don't be. This book is designed to teach programming with intuitive and engaging techniques. And we use two of today's most exciting technologies: Alice, in which you build your own virtual worlds, and Java, the most popular programming language in use today. Jump on in—the water's warm!

1.1 Introduction

There was a time when the people who knew how to program a computer lived in a world of their own—a complex world full of arcane symbols and mathematical terms. It was the world of the geek and the nerd. Not anymore.

Programming a computer no longer has to be a complex, arcane experience.

Computers are technical devices, certainly, but that doesn't mean that we have to be overwhelmed by their complexity. Almost everyone can use a computer these days because most programs are designed to be truly *usable*—much of the inherent complexity is managed for us. In a similar way, the techniques for programming a computer have become more accessible as well.

For you, this may be a one-time exploration into computing. Or you may have your sights set on continuing in this field. Either way, you're in the right place. Computing is one of the world's fastest growing disciplines, and the demand for good programmers is rising. But even if you're not looking for a career, the concepts we'll explore here are helpful in many other ways.

In this book we focus on *object-oriented programming* (OOP), which is the most popular approach to computer programming today. It has gained dominance because it is a natural, intuitive way to think about problems and their solutions. OOP is largely responsible for taking the world of programming away from the geeks and making it more accessible.

In object-oriented programming, we create the objects we need and tell them to perform services for us.

As the name implies, object-oriented programming is all about managing *objects*. An object can be anything—a character in an animation, a scoreboard in a game, a list of friends, whatever. In OOP, we create the objects we need and then we tell those objects to do things for us.

Two technologies that make use of an object-oriented approach are Alice and Java, the cornerstones of this book.

Alice and Java

In the first five chapters of this book, we focus on Alice, an environment designed to introduce programming concepts in an engaging manner. In the remaining chapters we focus on Java, a popular programming language in use by professionals today. Alice will help us lay the foundation, and Java will give us the freedom to explore many other possibilities.

Alice is a computer environment in which you create virtual worlds containing three-dimensional characters and objects that move and interact. Figure 1.1 shows a screen shot from an Alice virtual world. Alice was developed at Carnegie Mellon University and is named in honor of Lewis Carroll and his wonderful books *Alice in Wonderland* and *Through the Looking Glass*.

Figure 1.1

An Alice virtual world

You can use Alice to create animations in which characters play out a scene or to create games and other interactive worlds in which objects respond to mouse clicks and keyboard input. When you create an Alice world, you are the boss. You decide which objects to include in your world and how they will behave. With Alice, you're part movie director and part choreographer. You provide the instructions, like *move forward* and *turn right*, and the objects in your world carry out those instructions.

Though it won't feel like it, when you build a virtual world in Alice and dictate the way it behaves, you are programming. After all, that's what object-oriented programming is all about—telling objects what to do.

Both Alice and Java use an object-oriented approach.

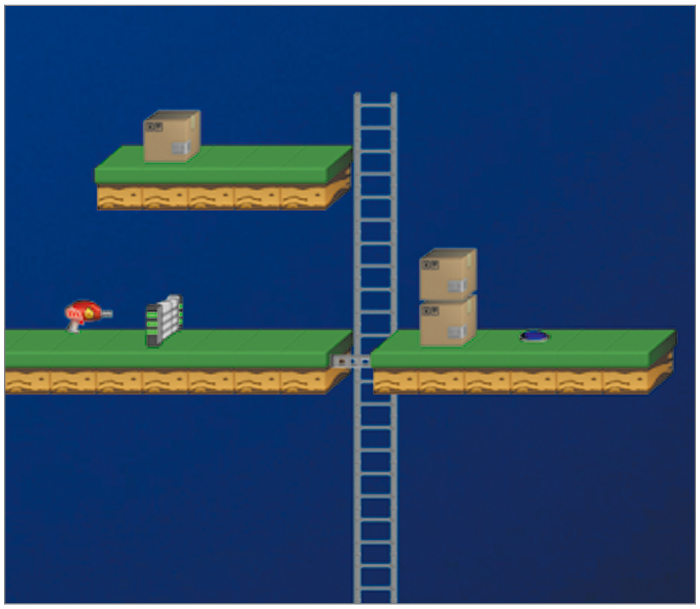
Once Alice paves the way, we'll transition into developing programs in Java, which is a *general-purpose programming language*. Java can be used to create animations and games, as we can in Alice, but it is far more versatile. You wouldn't use Alice to create a social network or help manage a student organization, but you could with Java.

Java was created by James Gosling at Sun Microsystems. It's become one of the most popular programming languages in use today.

One of the ways we'll explore the capabilities of Java is by examining a program called ThunkIt, a game in which the user helps stranded students get back to their school by solving a variety of puzzles. In each level of the game a student moves obstacles and uses various gadgets to outwit the detention robots and get one step closer to school. A screen shot of ThunkIt is shown in Figure 1.2.

Figure 1.2

ThunkIt, a Java program



Not only will we examine the code that makes the objects in ThunkIt do what they do, but you'll create game objects of your own and then design levels of the game that use them. More on that later.

Our goal in this book is not to teach Alice or Java *per se,* but rather to use them to teach fundamental programming concepts. These concepts apply not only to Alice and Java, but also to many other popular programming languages.

No matter which programming language you use, you need a development environment in which to create and execute your programs. Let's take a look at the Alice development environment.

1.2 The Alice Environment

Alice animations are made and executed within the Alice *integrated development environment* (IDE). Versions of Alice are available for both Windows and Mac OS, and can be downloaded for free from the Alice website (www.alice.org). Appendix A contains information about installing the Alice environment.

A development environment is a program used to create and run another program.

As shown in Figure 1.3, when you start the Alice environment, it presents a window that allows you to specify what you want to do initially. Depending on which tab you pick, you can choose to:

- run a tutorial,
- open a world you've had open recently (if any),
- start a new world using an existing template,
- explore one of several example worlds, or
- open an Alice world that was previously stored on your computer.

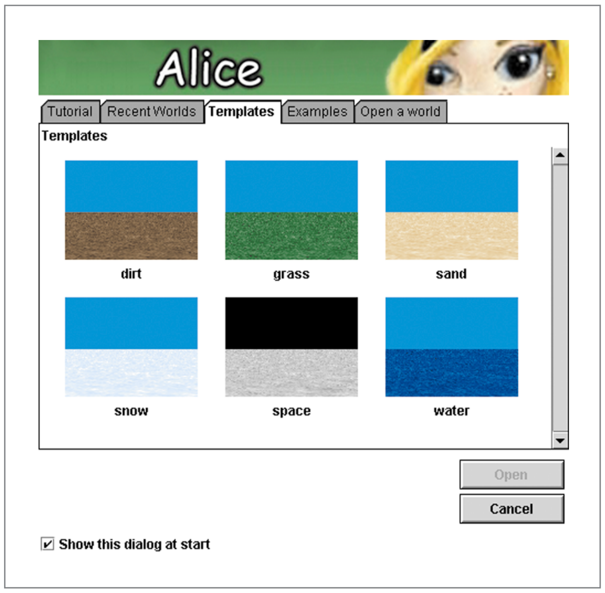


Figure 1.3 The Alice welcome screen

The tutorials are a good place to start, and we encourage you to use them to get acquainted with the Alice environment and its capabilities. Likewise, feel free to open and explore the various example worlds provided. Play around with them. Have fun. Don't worry if you don't understand everything you see in the tutorials and examples—we cover the key topics carefully in this book. Just don't be afraid to explore and experiment.

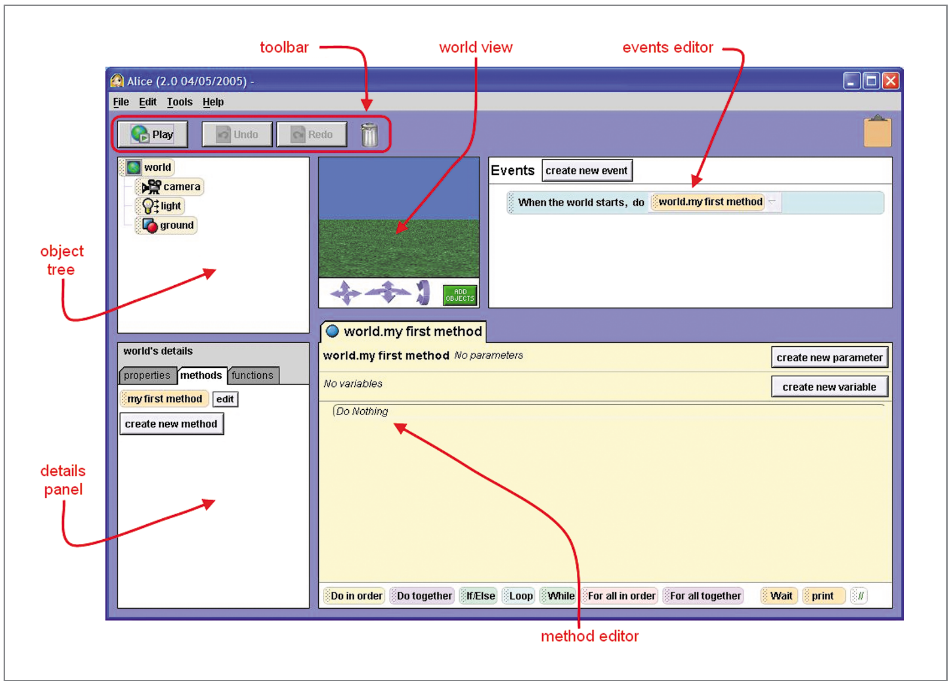
Programming is a participation sport! The more you play, the better you'll get. Explore and experiment!

The primary Alice window contains several distinct areas, as shown in Figure 1.4. The *toolbar* includes a button to play the current animation, which brings up a separate window in which the animation is displayed and controlled. The *world view* shows the virtual world as it initially appears to the camera, and has controls for adjusting the camera's initial point of view. The *object tree* lists all objects in the world, allowing you to select a particular object as you develop an animation.

The *details panel* provides information about the particular object currently selected in the object tree. The *method editor* is where you make changes to the code that dictates what your animation does. Finally, the *events editor* is where you specify what will happen when particular events occur.

Figure 1.4

The Alice development environment



We'll explore all of these aspects of the Alice environment over time as appropriate. For now, just begin to get a feel for the layout of the environment.

Appendix A contains the details of using the Alice environment to accomplish particular tasks. Use it as needed as you progress through the chapters.

1.3 Objects in Alice

Let's start by exploring a simple Alice world called `SpinningCubes` (stored in the file SpinningCubes.a2w). This world contains two cubes, one red and one blue, as shown in Figure 1.5. When you play the animation, first the blue cube spins in one direction, then the red cube spins in the opposite direction. Try it! We encourage you to keep the Alice environment open while you're reading this book, experimenting with our sample worlds as you go along.

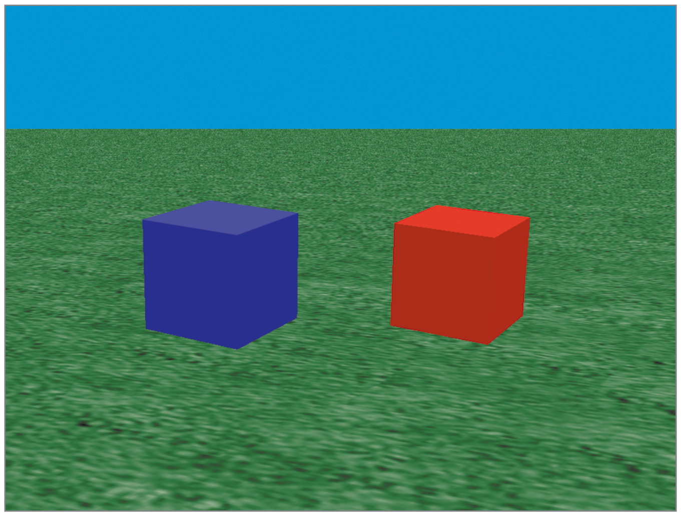
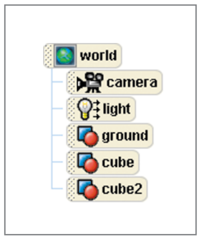


Figure 1.5

The SpinningCubes world

The two cubes in this animation are objects, as is everything in an Alice world. The objects contained in an Alice world are listed in the object tree, as shown in Figure 1.6. The `SpinningCubes` world contains objects that represent the camera, the light source, the ground, and the two cubes.



world
camera
light
ground
cube
cube2

Figure 1.6

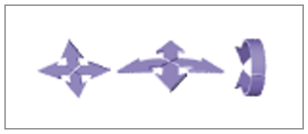
The object tree for the `SpinningCubes` world

All Alice worlds have a camera and a light source. The camera represents the point of view of the person watching the animation. Most worlds also have some kind of ground surface as well. The templates in the Alice welcome screen provide several basic ground surfaces from which to choose. The `SpinningCubes` world makes use of a ground surface covered in grass.

The camera's initial point of view of the world can be set using the camera controls, shown in Figure 1.7 and located under the world view window in the Alice environment. The camera controls may take some getting used to. The first control shifts the camera up, down, right, or left. The second control moves the camera forward or backward in the world, or rotates the camera right or left. The third control pivots the camera's view up or down. Experiment with the controls to get a feel for how they can be used to get a particular point of view on the world.

Figure 1.7

The camera controls



Once you use the camera controls to set the initial point of view, the camera will keep that orientation throughout the animation unless you dictate otherwise. As we'll see in later examples, we can set it up so that the camera's orientation changes as the animation unfolds.

TRY THIS!

1. Using the camera controls, change the camera viewpoint in `SpinningCubes` so that the blue cube is in the foreground and the red cube is behind it in the background.
2. Change the camera viewpoint in `SpinningCubes` so that it looks down on the cubes from above.

Calling Methods

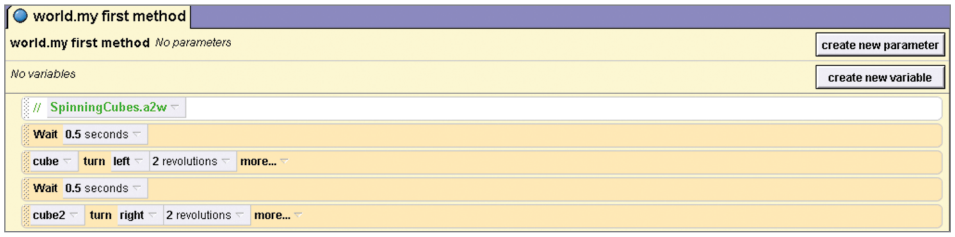
A *method* is a set of statements that can be *called* (or *invoked*) whenever we want those statements to be executed. Every object has methods that define that object's potential behavior. For example, most Alice objects have a method called `turn` that, when called, will rotate the object. Similarly, the `move` method will move an object in a particular direction when it is called.

We get an object to do something by calling one of its methods.

The `world` object in every Alice animation has a method called `my first method` that is executed whenever the animation is played. This method often calls methods in other objects. The `my first method` method for `SpinningCubes` is shown in Figure 1.8.

Figure 1.8

The `my first method` method for the `SpinningCubes` world



```
world.my first method
world.my first method  No parameters          create new parameter
No variables                                  create new variable
  // SpinningCubes.a2w
  Wait 0.5 seconds
  cube turn left 2 revolutions more...
  Wait 0.5 seconds
  cube2 turn right 2 revolutions more...
```

In this example, the first line in `my first method` is a *comment*, in green type and beginning with two slash marks (`//`). Comments are included for the human reader and do not have any effect on the animation. We typically include a comment at the beginning of `my first method` to indicate the file name of the world.

The `SpinningCubes` animation makes use of the `Wait` statement, which is used to pause the animation for a particular period of time. In most of our examples we pause at the beginning of the animation just to let the human viewer see the initial state of the world before any action begins. In this example we also use a second `Wait` statement to pause in between the spinning of the two cubes. The `Wait` statement is one of several *control statements* listed below the method editor, as shown in Figure 1.9. We explore the rest of these statements at appropriate points in the next few chapters.

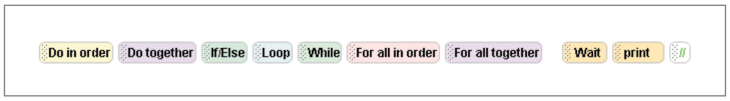


Figure 1.9
The control statements list

To make the cubes spin, we call the `turn` method of each cube. In object-oriented terms, we say we are *sending a message* to an object to request that it perform a particular service for us. In the `SpinningCubes` example, we initially ask `cube` to turn. Then, after a brief pause, we ask `cube2` to turn. Figure 1.10 shows the elements of a method call.

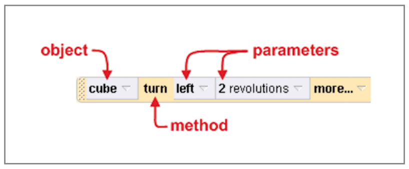


Figure 1.10
Calling a method of an object

Methods can accept *parameters*, which provide additional information to the method. When the `turn` method is called, we use parameters to indicate which direction to turn and how many revolutions to turn. These values can be changed using the drop-down menus in the statement. The menu labeled `more...` lets you access additional parameters, such as the statement's duration (how long it takes to execute the turn).

A method's parameters provide additional information that tailors its behavior.

TRY THIS!

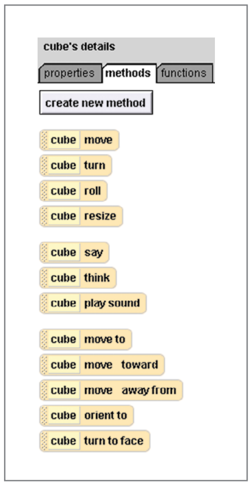
3. Modify `SpinningCubes` so that the pause between the cubes spinning is one second.
4. Modify `SpinningCubes` so that both cubes turn to the right and `cube2` turns only one revolution.
5. Modify `SpinningCubes` so that `cube` completes its turn in half a second and cube2 completes its turn in three seconds.

To add a control statement (such as `Wait`) to a method, simply drag it up from the list into the method editor. Comments can also be added in this way. As you drag, a green line appears to indicate where you are adding the new statement. Depending on the statement, it may prompt you to set certain parameter values when you add it to the method. Consult Appendix A if needed for help with these environment operations.

The available methods for an object are listed in the details panel when that object is selected in the object tree. Figure 1.11 shows part of the methods tab of the details panel for one of the cube objects. To add a new call to a method, drag the method name from the details panel to the method editor.

Figure 1.11

Some of the methods available for `cube`


cube's details

properties | methods | functions

create new method

cube move
cube turn
cube roll
cube resize

cube say
cube think
cube play sound

cube move to
cube move toward
cube move away from
cube orient to
cube turn to face

All Alice objects have a set of built-in methods that we can use. We can also add our own.

The built-in methods for a cube object will be found in almost all other Alice objects as well. They provide several basic movement operations, including some that move relative to other objects or at a particular speed. The built-in methods also include `say` and `think`, which produce speech bubbles above the object, as in a comic strip. Another method, `play sound`, allows an object to play a sound file. A few sound effects are built into the Alice environment, but you can import any .wav or .mp3 sound file to use in your animation.

The full list of built-in methods are described in Appendix B. Feel free to experiment with these methods—we'll see many of them in use in upcoming examples. You'll learn how to add your own methods to an object in Chapter 2.

TRY THIS!

6. Make additional calls to the `turn` method in `SpinningCubes` so that each cube spins in both directions, first right then left.

7. Modify `SpinningCubes` so that the cubes float up into the air 1 meter after they spin.

8. Modify `SpinningCubes` so that `cube` makes a "pop" sound after it spins and `cube2` makes a "thud" sound after it spins.

In Alice, special methods called *functions* are used to retrieve key information about an object, such as how close it is to another object. The available functions for an object are listed in a separate tab of the details panel. We'll make use of functions in later examples as well.

Properties

In addition to methods, which represent an object's potential behaviors, an object also has *properties*, which describe its state of being at any point in time. For example, the `color` of an Alice object is one of its properties. In the `SpinningCubes` world, `cube` is blue and `cube2` is red. The values of properties can be changed as needed.

An object's properties describe its current state, such as its color and opacity.

The properties of an object are listed in another tab in the details panel. Some standard properties are `opacity` (how much you can see through an object) and `fillingStyle` (whether an object is solid or represented as a wire frame).

The value of a property can be changed directly in the details panel using the corresponding drop-down menu. This sets the initial state of the object. A property value can also be changed during an animation using a method call. To change a property value using a method call, drag the property from the details panel into the method editor. This adds a call to a `set` method for that property.

Some object properties are not shown in the properties tab. For instance, an object's position within the world is a property of that object, but position doesn't show up in the properties list. Instead, Alice provides various methods (`move`, `turn`, `roll`, etc.) that change the object's position in a smooth, animated manner. The size of an object is another hidden property that can be changed using the `resize` method.

TRY THIS!

9. In `SpinningCubes`, use the properties drop-down menu for `fillingStyle` to show `cube` as a wire frame.
10. Modify `SpinningCubes` so that the color of `cube2` changes to yellow after both cubes finish turning.
11. Modify `SpinningCubes` so that the size of `cube` shrinks by half after it turns and the size of `cube2` doubles after it turns.

The techniques for setting the initial position and size of an object are discussed in the next section.

1.4 Alice Classes

An object is created from a class. In Alice, classes are organized into galleries.

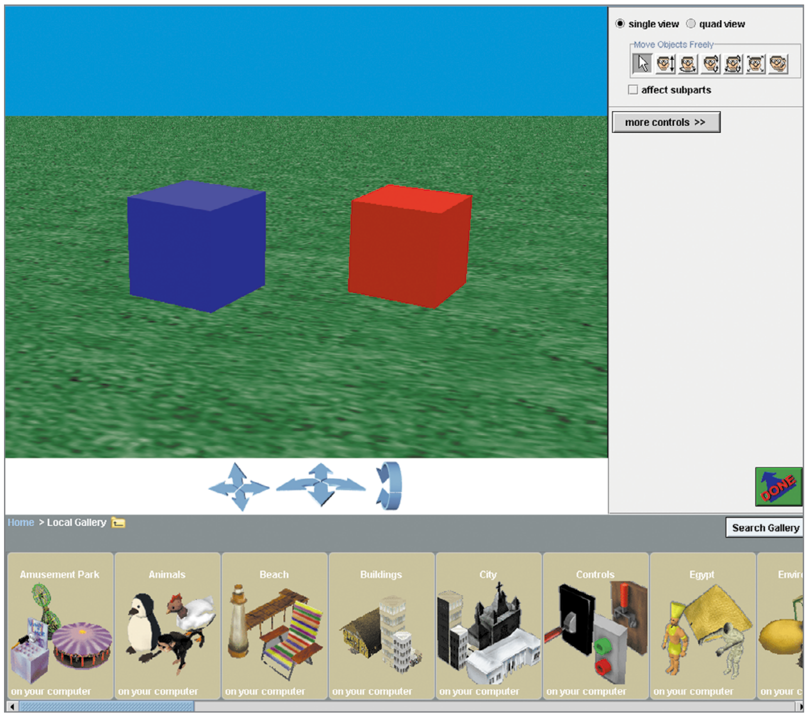
An object is created from a *class*, which serves as the blueprint, or pattern, from which all similar objects are created. For example, the two cube objects in the `SpinningCubes` world were created from a class called `Cube`. The class of an object determines the methods and properties the object will have.

The classes we use to create Alice animations are organized into *galleries*. The Alice environment has several local (built-in) galleries, and you can access several more galleries through the Web. The local galleries are generally a subset of those you can find on the Web. There may be a delay in accessing the web galleries depending on your network connection. We use classes from both sets of galleries in this book.

Pressing the `Add Objects` button, found next to the camera controls under the world view window, produces a window such as the one in Figure 1.12. The available class galleries are displayed along the bottom. (Local galleries are displayed by default.)

Figure 1.12

Accessing the class galleries and the object positioning controls



Clicking a gallery will display the classes available in that gallery. For example, clicking the **Beach** gallery provides access to the `BeachChair` and `Lighthouse` classes, among others. The `Cube` class used in the `SpinningCubes` world is found in the **Shapes** gallery. Take some time to become familiar with the various classes available in the galleries.

To add an object to an Alice world, drag the appropriate class into the world view window. Once added, you can use the controls on the right side to position, orient, resize, and copy the object as you see fit. When you're finished adding and positioning objects, press the `Done` button. Remember that Appendix A contains additional details about using the various environment controls.

TRY THIS!

12. Add a third cube to the `SpinningCubes` world and adjust the camera's viewpoint so you can see all three cubes. Adjust the new cube's size and orientation to be roughly equal to the others. Set its color to magenta in the properties tab of the details panel. Modify `my first method` so that the new cube spins similar to the others.
13. Add an `Anvil` object from the **Objects** gallery to the `SpinningCubes` world so that it looks like it's sitting on the red cube. Add a `StopSign` object from the **Roads and Signs** gallery between the cubes.

1.5 Do Together and Do In Order

Unless we indicate otherwise, the statements in a method are executed in order, one after the other. In animations, however, we often want two or more things to happen at the same time. The Alice control statements (listed below the method editor) include a statement called `Do together` that allows us to do two or more things simultaneously. A `Do together` statement contains other statements, indicating that those statements should all be executed at the same time.

Let's look at an example. The `Blimps` world contains two blimps floating in the sky, as shown in Figure 1.13. The blimps were created using the `Blimp` class found in the **Vehicles** gallery. When the animation is played, both blimps move through the sky at the same time in different directions.

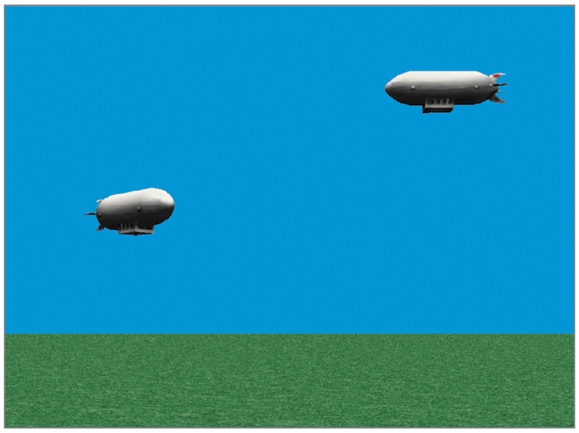
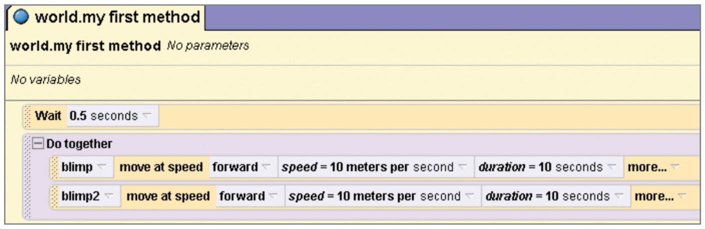


Figure 1.13
The `Blimps` world

The `my first method` method for `Blimps` is shown in Figure 1.14. It uses a `Do together` statement, which contains two statements that move the two blimps. If those two statements were not contained in a `Do together` statement, one blimp would move, and when it was finished, the other blimp would move. By putting both statements in the `Do together` statement, the blimps move at the same time. Note that in this example the blimps are controlled using the `move at speed` method, which allows us to define how fast an object moves.



```
world.my first method
world.my first method  No parameters
No variables
Wait 0.5 seconds
Do together
    blimp  move at speed  forward  speed = 10 meters per second  duration = 10 seconds  more...
    blimp2  move at speed  forward  speed = 10 meters per second  duration = 10 seconds  more...
```

Figure 1.14
Using the `Do together` statement

The `Do in order` statement is essentially the opposite of the `Do together` statement. It forces the statements it contains to be executed in order, one after another. The `Do in order` statement is needed when you want to perform some statements sequentially within a `Do together` statement.

In an example world called `Bugs`, two bugs are shown scurrying around the ground near a tree, as depicted in Figure 1.15. The bug objects are created from the `Ladybug` class in the **Animals** gallery and the tree is created from the `HappyTree` class in the **Nature** gallery.

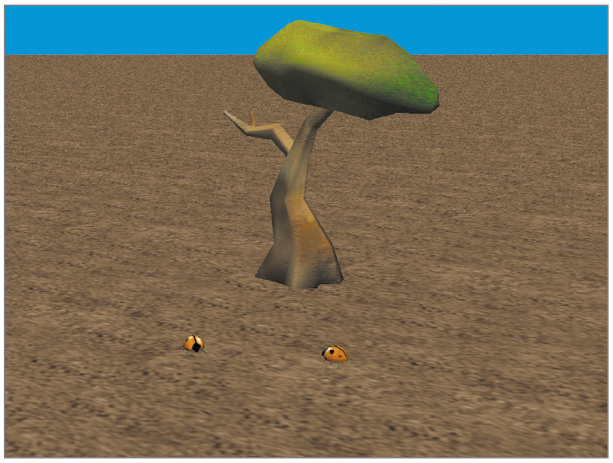
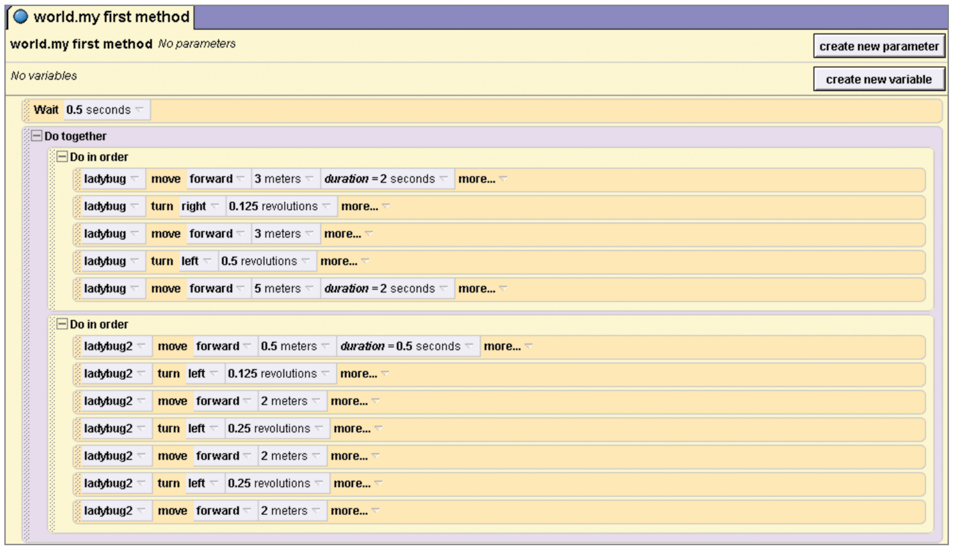


Figure 1.15
The `Bugs` world

The movement of the bugs is accomplished using various calls to their `move` and `turn` methods. We want both bugs to move at the same time, but we want each step for a bug (move, then turn, then move, etc.) to be executed in order. The method that accomplishes this coordinated movement is shown in Figure 1.16.



```
world.my first method
world.my first method  No parameters                                   create new parameter
No variables                                                           create new variable
Wait 0.5 seconds
Do together
  Do in order
    ladybug move forward 3 meters duration = 2 seconds more...
    ladybug turn right 0.125 revolutions more...
    ladybug move forward 3 meters more...
    ladybug turn left 0.5 revolutions more...
    ladybug move forward 5 meters duration = 2 seconds more...
  Do in order
    ladybug2 move forward 0.5 meters duration = 0.5 seconds more...
    ladybug2 turn left 0.125 revolutions more...
    ladybug2 move forward 2 meters more...
    ladybug2 turn left 0.25 revolutions more...
    ladybug2 move forward 2 meters more...
    ladybug2 turn left 0.25 revolutions more...
    ladybug2 move forward 2 meters more...
```

Figure 1.16

Using the `Do in order` statement

The two `Do in order` statements are executed at the same time, one controlling the movement of one bug and the other controlling the movement of the other bug. Within each `Do in order` statement, the individual movements of a bug are executed sequentially.

By using a thoughtful combination of `Do together` and `Do in order` statements, it's possible to create interesting animation effects.

TRY THIS!

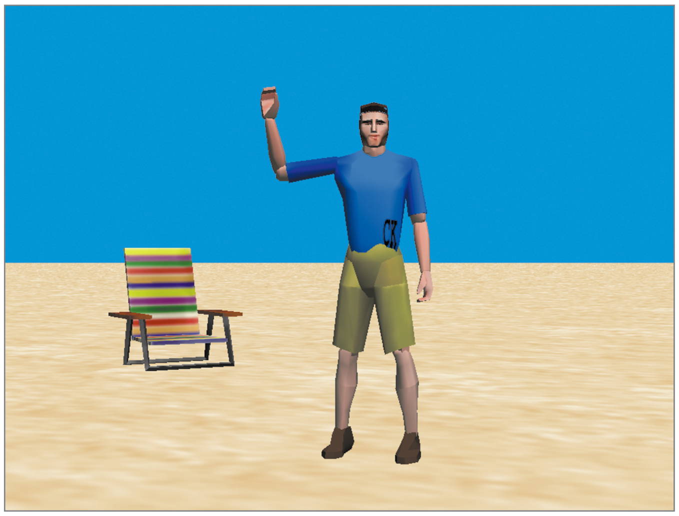
14. Modify `SpinningCubes` so that both cubes spin at the same time.
15. Add a third blimp to the `Blimps` world that moves half as fast as the others.
16. Add a third `Ladybug` object to the `Bugs` world that moves in its own pattern.

1.6 Composite Objects

A *composite object* is an object that contains other objects. Many objects in the Alice galleries are composite objects. Let's look at an example. The `SurferWave` world shows a surfer on the beach, as shown in Figure 1.17. When the animation is played, the surfer turns his head (as if noticing the viewer), turns his upper body to face the viewer, and then waves his hand.

Figure 1.17

The SurferWave world



The surfer was created from the RandomGuy2 class in the **People** gallery. We modified the object's name from the default (randomGuy2) to something more appropriate for this example (surfer) in the object tree. The beach chair was created from the BeachChair class in the **Beach** gallery.

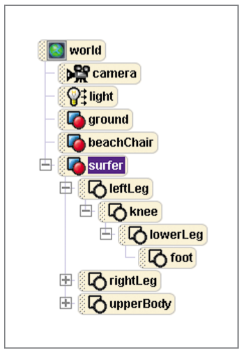
A composite object is made up of other objects. We can control the whole object or any of its parts.

The surfer is a composite object. It is made up of the left leg, the right leg, and the upper body objects. Each of these parts is itself a composite object. The upper body, for instance, is made up of the left arm, the right arm, and the head. The arms and legs can be further decomposed.

A composite object has a plus sign next to it in the object tree. Clicking the plus sign expands the tree and displays its component objects, as shown in Figure 1.18. When expanded, the plus sign changes to a minus sign. When the minus sign is clicked, that section of the tree is hidden again.

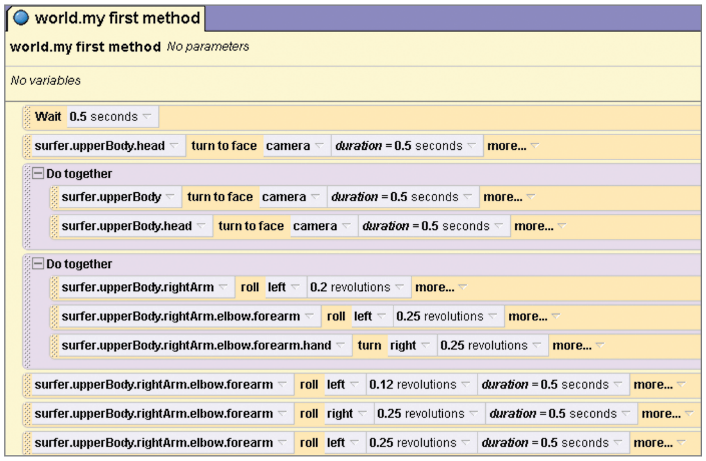
Figure 1.18

Viewing the parts of the surfer composite object



The `Ladybug` objects from the `Bugs` world in the previous section are also composite objects. Each leg of a bug can be moved independently, as can the wings and even the antennae. In the `Bugs` world example, we simply moved the entire bug, but composite objects give you the ability to refine the animations to the level you choose.

You can send messages to (that is, call methods of) an entire object or to any component part. When referring to a component part, you access it through its containing object. In this example, the entire object is referred to as `surfer`. The entire upper body of the surfer is referred to as `surfer.upperBody`. The head of the surfer is referred to as `surfer.upperBody.head`. Figure 1.19 shows the code for the `SurferWave` world.



```
world.my first method
world.my first method No parameters

No variables

Wait 0.5 seconds
surfer.upperBody.head turn to face camera duration = 0.5 seconds more...
Do together
    surfer.upperBody turn to face camera duration = 0.5 seconds more...
    surfer.upperBody.head turn to face camera duration = 0.5 seconds more...
Do together
    surfer.upperBody.rightArm roll left 0.2 revolutions more...
    surfer.upperBody.rightArm.elbow.forearm roll left 0.25 revolutions more...
    surfer.upperBody.rightArm.elbow.forearm.hand turn right 0.25 revolutions more...
surfer.upperBody.rightArm.elbow.forearm roll left 0.12 revolutions duration = 0.5 seconds more...
surfer.upperBody.rightArm.elbow.forearm roll right 0.25 revolutions duration = 0.5 seconds more...
surfer.upperBody.rightArm.elbow.forearm roll left 0.25 revolutions duration = 0.5 seconds more...
```

Figure 1.19

Manipulating the parts of a composite object

The `turn to face` method is used to turn the surfer's head toward the camera initially. The first `Do together` statement then turns the entire upper body toward the camera, and again turns the head. If the second head turn were not performed, the head would "ride" the upper body and turn to face away from the viewer.

The second `Do together` statement swings the arm up in preparation for the wave. To do this, it simultaneously rolls the right arm, rolls the right forearm, and turns the hand. The wave itself is accomplished with three rolls of the forearm.

TRY THIS!

17. Modify `SurferWave` so that the surfer says "Welcome to my world!" while he's waving.

18. Modify `SurferWave` so that the surfer's arm returns to its original position after finishing the wave.

19. Modify `SurferWave` so that the surfer moves his left hand to his hip during the wave.

6184_LEWI_CH01_p001-024.qxd 1/17/08 6:41 PM Page 18

The composite objects in the Alice galleries vary in the way they are made up. Some can be articulated down to individual fingers and others are less versatile.

Unfortunately, it is not easy to add a new class to Alice. Therefore, you can't really make your own types of objects. The process of creating a composite object is particularly tricky. It involves using additional 3D graphics software to create the pieces of the object and to define their relationships to each other, pivot points, and other details. This process is beyond the scope of this book. There is, however, a tool provided with Alice to make somewhat customized characters. We discuss that tool in the next section. In the examples in this book we constrain ourselves to using the predefined classes provided in the galleries.

1.7 More to Explore

As discussed in section 1.1, the content of this book focuses on the core ideas related to object-oriented programming. We've already introduced several in this chapter: objects, classes, methods, properties, and composite objects.

To help complete the picture, each chapter in this book ends with a section called More to Explore, in which we briefly discuss topics that you may want to look into. These issues are usually environment or language details that don't play a role in the big picture, but will help you as you develop your programs. For Alice, Appendices A and B contain further details for many of these topics.

Built-In Methods Make sure you explore the methods that are part of (almost) every Alice object by default. We've discussed a few of them in this chapter and will continue to use them as needed. Some of them have subtle but important distinctions, such as the difference between the `move` and `move at speed` methods. Appendix B contains a summary of all the built-in Alice methods.

Turn vs. Roll In a three-dimensional world, an object's orientation (the way it's facing) can be changed without changing its position. Each object has a particular pivot point around which it rotates. You can change an object's orientation by turning it right or left, turning it forward or backward (think of leaning forward or backward), or rolling it left or right (think of leaning to one side). See Figure 1.20. Keep in mind that directions are relative to an object's orientation—so changing position and orientation at the same time can cause some interesting results. For even more control, experiment with the `asSeenBy` parameter when making complicated movements. Like anything else, the more you experiment with the various combinations of methods and directions, the more familiar they will become.

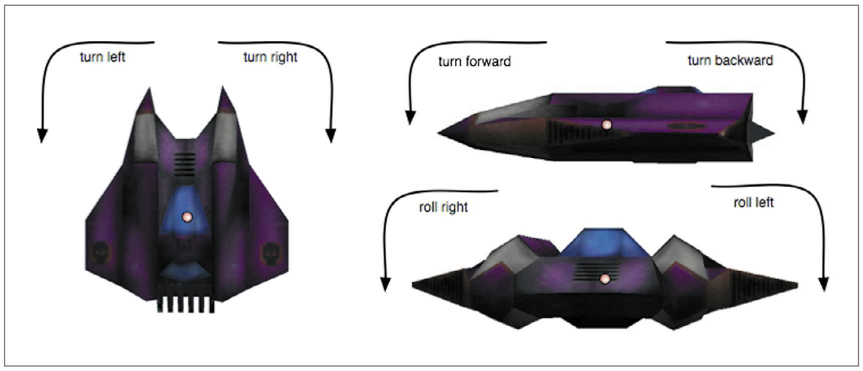
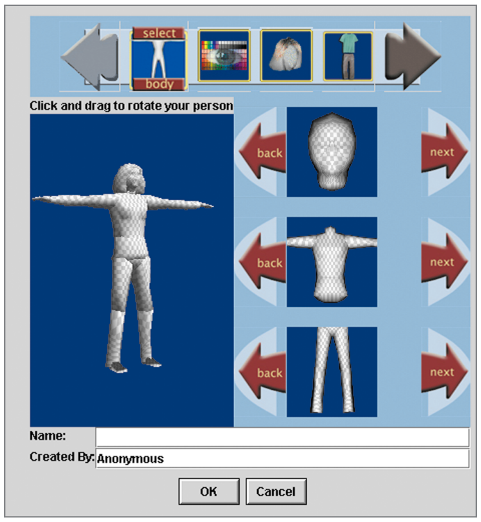


Figure 1.20

Changing an object's orientation in three dimensions

He Builder / She Builder The ability to create completely new classes of objects in Alice, using our own graphics, is not something we can tackle in this book. However, tools called `He Builder` and `She Builder` have been built into Alice to give you some control over the look of the human characters you create. You can choose skin and hair color, hair style, body type, and clothes. These tools are available at the end of the list of classes in the **People** gallery. They bring up a separate window to guide you through the character creation process, as shown in Figure 1.21.



Name:
Created By: Anonymous

Figure 1.21

Character building

Capture Pose Getting a character to perform a particular movement (like the surfer waving his hand) can involve a complex combination of movements. One way to simplify this process is to maneuver a character into a particular pose using the mouse tools, and then capture that pose to use later. Once you have the character in the pose you want, you can right click on it and choose the `capture pose` menu option, or use the `capture pose` button on the properties tab. You can then use the `set pose` method in an animation, which causes the character to smoothly move into the specified pose. We use character poses in later examples to cut down on the amount of code otherwise required.

Summary of Key Concepts

- Programming a computer no longer has to be a complex, arcane experience.
- In object-oriented programming, we create the objects we need and tell them to perform services for us.
- Both Alice and Java use an object-oriented approach.
- A development environment is a program used to create and run another program.
- Programming is a participation sport! The more you play, the better you'll get. Explore and experiment!
- We get an object to do something by calling one of its methods.
- A method's parameters provide additional information that tailors its behavior.
- All Alice objects have a set of built-in methods that we can use. We can also add our own.
- An object's properties describe its current state, such as its color and opacity.
- An object is created from a class. In Alice, classes are organized into galleries.
- A composite object is made up of other objects. We can control the whole object or any of its parts.

Exercises

EX 1.1 Describe the following terms: object, class, method, parameter, and property.

EX 1.2 What's the difference between the `move` and `move at speed` methods? Compare the parameters and consult Appendix B as needed.

EX 1.3 What does it mean when we "send a message" to an object?

EX 1.4 Write a statement, as it would appear in an Alice program, that would cause an object called `dancer` to spin around three times.

EX 1.5 Write a statement that would cause an object called `tree` to grow to three times its current size.

EX 1.6 Write a statement that would cause an object called `cheerleader` to turn to face the camera.

EX 1.7 How do you get two animation steps to happen at the same time in an Alice program? Give an example.

EX 1.8 Describe the composite structure of an object created from the `Chicken` class in the **Animals** gallery.

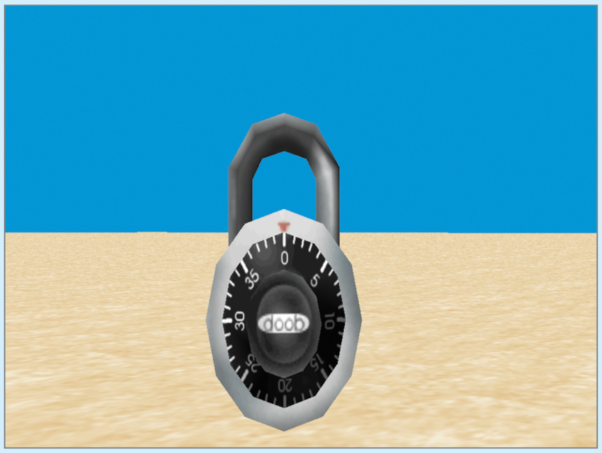
EX 1.9 Describe the composite structure of an object created from the `Barn` class in the **Farm** gallery.

EX 1.10 How do you access a particular part of a composite object? Give an example using the `Phonograph` class in the **Objects** gallery.

Programming Projects

PP 1.1 Create an Alice world in which a penguin waddles toward a hole in a frozen lake, tips over, and falls in. The `Penguin` class can be found in the **Animals** gallery and the `FrozenLake` class is in the **Environments** gallery. Use the `Circle` class (colored gray) from the **Shapes** gallery to make the hole in the ice. In addition to the standard built-in methods, the `Penguin` class comes with a few other methods that you can call to help with this animation.

PP 1.2 Create an Alice world that shows a combination lock being dialed and then the latch opening. The combination is 15-35-5 (that is, turn right to 15, then left to 35, and right again to 5). The `CombinationLock` class is in the **Objects** gallery.

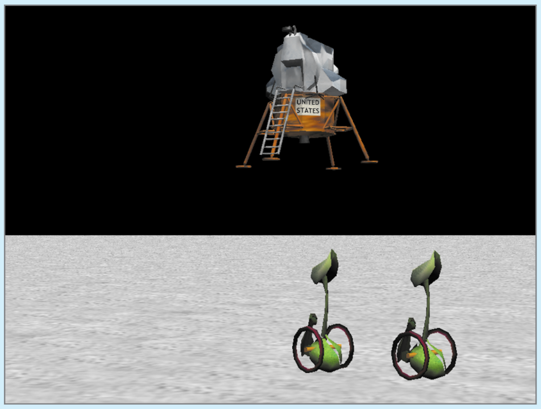


PP 1.3 Create an Alice world that shows a graveyard scene in which a casket opens and a mummy inside it sits up. Use the `Casket` and `Mummy` classes from the **Spooky** gallery, plus various others to set the mood.

PP 1.4 Using various classes from the **Vehicles** gallery, such as `Biplane`, `Blimp`, `Jet`, and `NavyJet`, create an Alice world in which several flying vehicles are moving through the air at different speeds, in different directions, and at different altitudes.

PP 1.5 Using various classes from the **Vehicles** gallery, such as `Motorboat`, `Sailboat`, and `Shakira`, create an Alice world in which several boats are moving across the water at different speeds and in different directions. Have one boat change direction at some point. Add a few colored `Sphere` objects from the **Shapes** gallery, partially submerged in the water, to represent buoys.

PP 1.6 Create an Alice world in which a `LunarLander` object from the **SciFi** gallery floats down gracefully to the moon's surface. After landing, the lander's door opens while two `AlienOnWheels` objects approach to greet the visitors.



PP 1.7 Create an Alice world depicting a moment from a fight between a troll and a wizard (created from classes in the **Medieval** gallery). As the troll swings his club down to hit the wizard, the wizard points at the club and it goes flying

out of the troll's hand. Then the wizard sinks magically into the ground and disappears, saying "Farewell" as he goes.

PP 1.8 Using various car and truck classes from the **Vehicles** gallery, such as `Car`, `ConvertibleCorvette`, `DumpTruck`, and `Humvee`, create an Alice world in which vehicles are moving right or left across the screen, going in opposite directions on a two-lane road. Use the `Road` class from the **City** gallery to create the road. Stagger the timing of the vehicles, starting them when needed and stopping them after they move out of the camera's view.

PP 1.9 Create an Alice world in which a chicken walks forward a few steps, pecks at the ground twice, and then clucks. Move the chicken's legs, neck, and head appropriately. Open the chicken's mouth while the clucking sound is played. The `Chicken` class is in the **Animals** gallery.

PP 1.10 Create an Alice world in which a helicopter lifts off the ground, flies in a wide circle, and lands. (An object can move in a circle by moving forward and turning one full rotation simultaneously.) Of course, the helicopter's propellers should be rotating whenever the helicopter is in the air. The `Helicopter` class can be found in the **Vehicles** gallery.