10 Algorithms

"Lather. Rinse. Repeat."
— Unknown

10.1 Where have we been? Where are we going?

Our friend Zoog had a nice run. Zoog taught us the basics of the shape drawing libraries in *Processing*. From there, Zoog advanced to interacting with the mouse, to moving autonomously via variables, changing direction with conditionals, expanding its body with a loop, organizing its code with functions, encapsulating its data and functionality into an object, and finally duplicating itself with an array. It is a good story, and one that treated us well. Nonetheless, it is highly unlikely that all of the programming projects you intend to do after reading this book will involve a collection of alien creatures jiggling around the screen (if they do, you are one lucky programmer!). What we need to do is pause for a moment and consider what we have learned and how it can apply to what *you want to do.* What is your idea and how can variables, conditionals, loops, functions, objects, and arrays help you?

In earlier chapters we focused on straightforward "one feature" programming examples. Zoog would jiggle and only jiggle. Zoog didn't suddenly start hopping. And Zoog was usually all alone, never interacting with other alien creatures along the way. Certainly, we could have taken these early examples further, but it was important at the time to stick with basic functionality so that we could really learn the fundamentals.

In the real world, software projects usually involve many moving parts. This chapter aims to demonstrate how a larger project is created out of many smaller "one feature" programs like the ones we are starting to feel comfortable making. You, the programmer, will start with an overall vision, but must learn how to break it down into individual parts to successfully execute that vision.

We will start with an idea. Ideally, we would pick a sample "idea" that could set the basis for any project you want to create after reading this book. Sadly, there is no such thing. Programming your own software is terrifically exciting because of the immeasurable array of possibilities for creation. Ultimately, you will have to make your own way. However, just as we picked a simple creature for learning the fundamentals, knowing we will not really be programming creatures all of our lives, we can attempt to make a generic choice, one that will hopefully serve for learning about the process of developing larger projects.

Our choice will be a simple game with interactivity, multiple objects, and a goal. The focus will not be on good game design, but rather on good *software design.* How do you go from thought to code? How do you implement your own algorithm to realize your ideas? We will see how a larger project divides into four mini-projects and attack them one by one, ultimately bringing all parts together to execute the original idea.

We will continue to emphasize object-oriented programming, and each one of these parts will be developed using a *class.* The payoff will be seeing how easy it then is to create the final program by bringing the self-contained, fully functional classes together. Before we get to the idea and its parts, let's review the concept of an *algorithm* which we'll need for steps 2a and 2b.

Our Process

1. **Idea**—Start with an idea.
2. **Parts**—Break the idea down into smaller parts.
   a. **Algorithm Pseudocode**—For each part, work out the algorithm for that part in pseudocode.
   b. **Algorithm Code**—Implement that algorithm with code.
   c. **Objects**—Take the data and functionality associated with that algorithm and build it into a class.
3. **Integration**—Take all the classes from Step 2 and integrate them into one larger algorithm.

10.2 Algorithm: Dance to the beat of your own drum.

An algorithm is a procedure or formula for solving a problem. In computer programming, an algorithm is the sequence of steps required to perform a task. Every single example we have created so far in this book involved an algorithm.

An algorithm is not too far off from a recipe.

1. Preheat oven to 400°F.
2. Place four boneless chicken breasts in a baking dish.
3. Spread mustard evenly over chicken.
4. Bake at 400°F for 30 min.

The above is a nice algorithm for cooking mustard chicken. Clearly we are not going to write a *Processing* program to cook chicken. Nevertheless, if we did, the above pseudocode might turn into the following code.

```
preheatOven(400);
placeChicken(4,"baking dish"167);
spreadMustard();
bake(400,30);
```

An example that uses an algorithm to solve a math problem is more relevant to our pursuits. Let's describe an algorithm to evaluate the sum of a sequence of numbers 1 through *N*.

$$SUM(N) = 1+2+3+ \ \dots \ +N$$

where *N* is any given whole number greater than zero.

```
1. Set SUM = 0 and a counter $I = 1$
2. Repeat the following steps while $I$ is less than or equal to $N$.
   a. Calculate SUM + $I$ and save the result in SUM.
   b. Increase the value of $I$ by 1.
3. The solution is now the number saved in SUM.
```

Translating the preceding algorithm into code, we have:

```
int sum = 0;
int n = 10;
int i = 0;

while (i <= n) {
  sum = sum + i;
  i++;
}

print ln(sum);
```

Step 1. Set sum equal to 0 and counter $i = 0$.

Step 2. Repeat while $i <= n$.
Step 2a. Increment sum.
Step 2b. Increment *i.*

Step 3. The solution is in sum. Print sum!

Traditionally, programming is thought of as the process of (1) developing an idea, (2) working out an algorithm to implement that idea, and (3) writing out the code to implement that algorithm. This is what we have accomplished in both the chicken and summation examples. Some ideas, however, are too large to be finished in one fell swoop. And so we are going to revise these three steps and say programming is the process of (1) developing an idea, (2) breaking that idea into smaller manageable parts, (3) working out the algorithm for each part, (4) writing the code for each part, (5) working out the algorithm for all the parts together, and (6) integrating the code for all of the parts together.

This does not mean to say one shouldn't experiment along the way, even altering the original idea completely. And certainly, once the code is finished, there will almost always remain work to do in terms of cleaning up code, bug fixes, and additional features. It is this thinking process, however, that should guide you from idea to code. If you practice developing your projects with this strategy, creating code that implements your ideas will hopefully feel less daunting.

10.3 From Idea to Parts

To practice our development strategy, we will begin with the idea, a very simple game. Before we can get anywhere, we should describe the game in paragraph form.

Rain Game

The object of this game is to catch raindrops before they hit the ground. Every so often (depending on the level of difficulty), a new drop falls from the top of the screen at a random horizontal location with a random vertical speed. The player must catch the raindrops with the mouse with the goal of not letting any raindrops reach the bottom of the screen.

Exercise 10.1: Write out an idea for a project you want to create. Keep it simple, just not too simple. A few elements, a few behaviors will do.