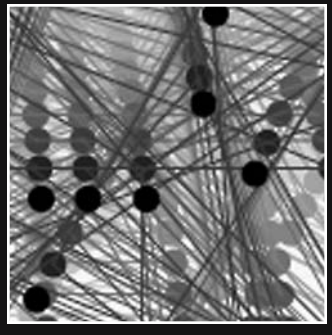
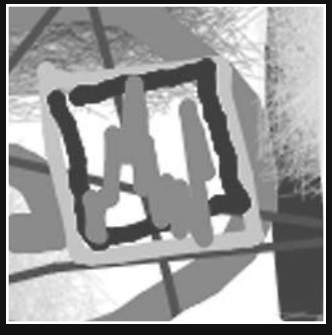
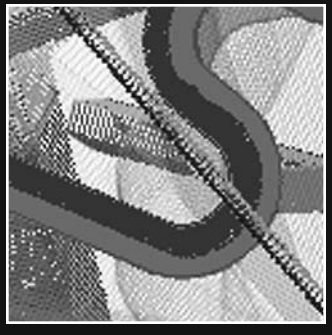
12 INTERACTIVITY

One of the most compelling aspects of digital/code art is interactivity. Interactive art is not in itself a completely new concept—for example, we've all become quite accustomed to "please touch" type children's museums. However, what is unique in code art is the fact that users can not only interact with a work of art, but in some cases they can actually redefine it, or even use the piece of art to create their own original works of art. Mark Napier's piece *net.flag* (`www.guggenheim.org/internetart/welcome.html`) is an excellent example of this. Viewers use Napier's web-based piece to design a flag, which (conceptually) becomes the flag of the entire Internet. Their creation remains the current flag of the Net until another viewer changes it. Each of the flags created also gets put into a permanent viewable database. Napier's piece—and others like it—represents a radical break with the established view of a work of art as contemplative object/space. Instead, in the case of *net.flag*, the work of art almost disappears, becoming a dynamic tool that others can use to express themselves.

In this chapter, you'll learn how to both detect and handle events. By **events**, I mean inputs to the system, such as mouse clicks or key presses. Of course, there are events that occur at a lower-level than the graphical user interface (GUI), as well—such as an internal timer or other system process that communicates with a running application. For example, my mail program periodically checks for incoming mail without my having to request it. It's also possible to detect other less standard high-level events, such as input from a video camera, microphone, or motion sensor. Processing has the capacity to interface with all these types of events. While in this chapter I'll only be covering mouse and key events, the principles discussed apply to most input devices.

Interactivity simplified

As you've probably come to expect by now, Processing simplifies the process of interactivity by encapsulating a bunch of the code you'd be required to include using pure Java. For an example of this, take a look at the following three sketches, each of which functions identically. The first two are written in Java and the third in Processing, but all three can be run in Processing. When run, the sketches print to the text area in the Processing application when the mouse is pressed within the Processing display window.

```
//  2 different Java approaches
// Java approach 1 implements the Java MouseListener interface:
void setup(){
  new MouseEventDemo();
}
class MouseEventDemo implements MouseListener {
  //constructor
  MouseEventDemo(){
    addMouseListener(this);
  }
  // we're required to implement all MouseListener methods
  public void mousePressed(MouseEvent e) {
    println("Java MouseListener interface example"+ ➥
            "mouse press detected");
```