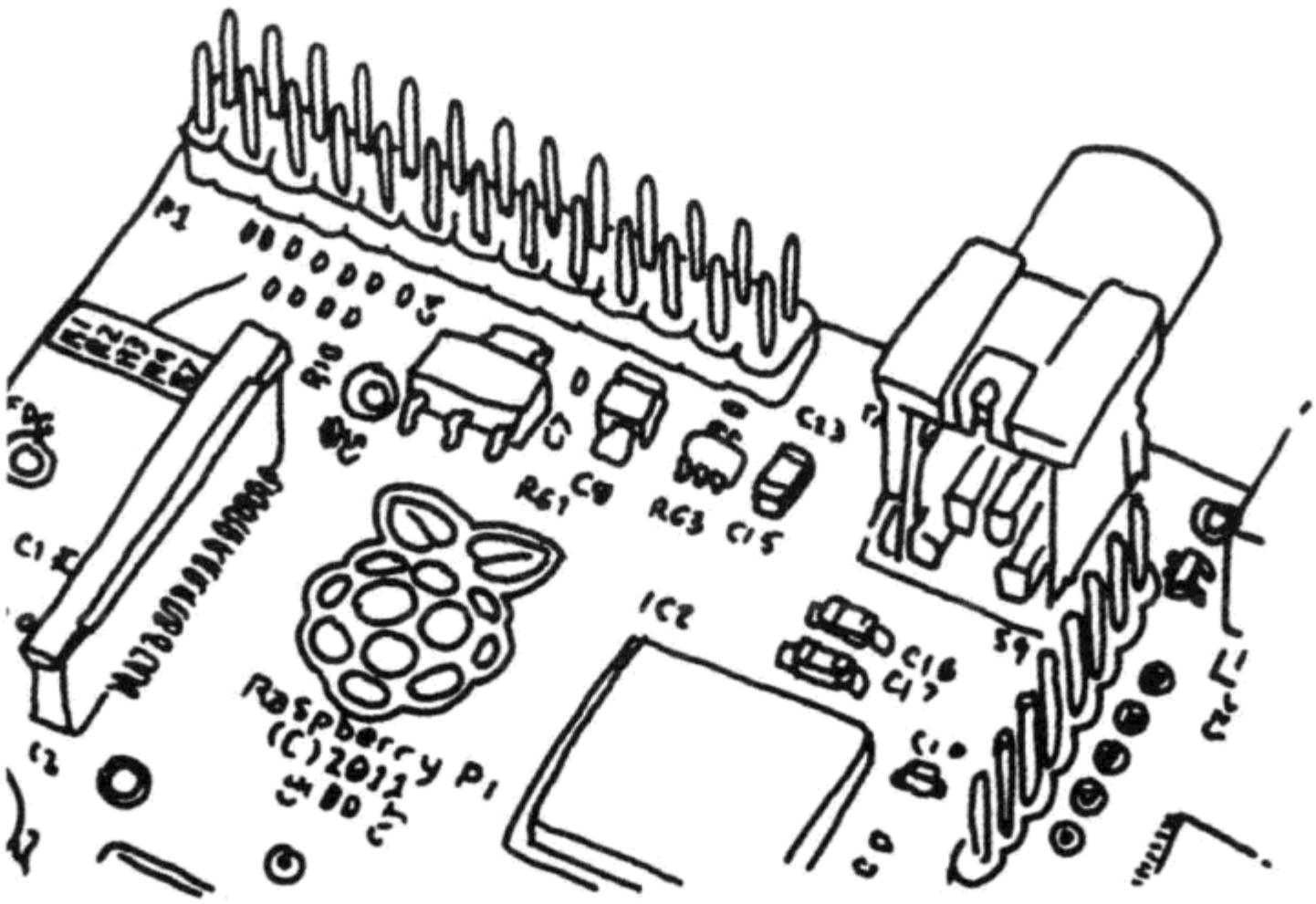
第 4 章

用 Python 实现动画与多媒体

Animation and Multimedia in Python



Pygame 是一个用于通过 Python 创建小游戏的轻量级框架。你也可以把它看作是一个用于多媒体程序设计的工具，通过它可以很方便地在屏幕上绘图、播放声音或处理键盘和鼠标事件。

Pygame 是对另外一个名为 Simple DirectMedia Library（SDL）的库的封装，通过 SDL 库可以对 Pi 底层键盘、鼠标、音频驱动和视频驱动进行各种操作。Pygame 可以让 SDL 用起来更简便。

本章的重点是展示 Pygame 在多媒体编程方面的一些能力，而并非是游戏编程的一个教程。本章末尾列出了一些有关游戏编程的参考资料。

初识 Pygame

Raspberry Pi 上预装了 Pygame 库。虽然 Pygame 有支持 Python 3.0 的版本，但 Raspbian 中所提供的版本只能在 Python 2.7 中使用。因此，你可以使用非 3.0 版本的 IDLE（双击 IDLE 图标，不要双击 IDLE 3 图标），也可以直接在命令行中用 2.7 版本的 Python 解释器来运行你的脚本。

下面的例子演示了一个最简单的 Pygame 程序，它在程序窗口中画了一个圆：

```
import pygame ❶

width = 640 ❷
```

```
height = 480
radius = 100
fill = 1

pygame.init() ❸

window = pygame.display.set_mode((width, height)) ❹
window.fill(pygame.Color(255, 255, 255)) ❺
while True:❻
    pygame.draw.circle(window,❼
                       pygame.Color(255, 0, 0),
                       (width/2, height/2),
                       radius, fill)
    pygame.display.update()❽
```

❶ 导入 Pygame 模块，使 Pygame 中的对象和函数在这个脚本中可用。

❷ 初始化一些全局变量。

❸ 先调用一个 `init()` 函数，执行 Pygame 的一些初始化动作，使你可以正常使用 Pygame 模块。另外，在这个函数中也调用了 Pygame 中所有子模块的 `init()` 函数。

❹ 创建一个窗口，准备用于在上面画图。这个窗口是一个 Pygame 的 *Surface* 类的对象。

❺ 把窗口填充为白色。

❻ 不断循环，你可以把每次循环所执行的代码看成是绘制了一个动画中的一帧。

❼ 在窗口的中央画一个红色的圈。

❽ 循环中的所有绘画命令其实都是在一个不可见的画布上执行的，当你画完了一帧后，调用 `display.update()` 把画好的图像显示在屏幕上。

在这些例子中，你需要按 Control-C 来中断程序的运行。如果你希望能够通过 Pygame 窗口上的关闭按钮来结束程序运行的话，可以在 while 循环的尾部添加下面的代码：

```
while True:
    if pygame.QUIT in [e.type for e in
    pygame.event.get()]:
        break
```

这样，按下关闭按钮的产生事件就可以被正确捕捉并处理。有关事件的详细介绍，请参考“处理事件与输入”。

Pygame 中包含了很多子模块和对象，本章的后续章节中将陆续介绍与之相关的基础知识。

Pygame 的 Surface

Pygame 中的 Surface 可以被想象成一个矩形的图像，Surface 可以由多个图像帧组成，用于实现游戏或动画中一个场景。Surface 上的每一个像素由一组 8 位的 RGB 数字来表示，如 (0,255,0) 表示绿色。如果加上第 4 个数字，则可以用于表示透明度，如 (0,255,0,127) 表示具有 50% 透明度的绿色。

图像显示窗口是一个作为背景的 Surface，可以在上面绘制其他的 Surface。通过 pygame.display 模块可以控制显示窗口或获取显示窗口的各种参数。set_mode() 函数用于创建一个新的图像显示窗口，update() 函数用于在绘制每一帧图像后刷新图像显示窗口。

使用 pygame.image 模块中的 load() 函数可以把一幅图像加载到 Surface 中并用于显示，使用 blit() 函数把加载后的图像与

创建好的 Surface 进行合并：

```
import pygame

pygame.init()
screen = pygame.display.set_mode((450, 450))
background = pygame.image.load("background.png") ❶
background.convert_alpha() ❷
screen.blit(background, (0, 0)) ❸
while True:
    pygame.display.update()
```

❶ 加载当前目录下的背景图片文件。

❷ convert_alpha() 函数用于把 Surface 的格式转换为与当前显示模式相匹配的格式。这不是一个必须进行的操作，但建议通过这样做来提高图像绘制的速度。

❸ 默认情况下 Surface 是纯黑色的，使用 blit() 函数把背景图片与黑色的 Surface 合并。

下面是一个把两张图片合并显示的例子（图 4.1）：

```
import pygame

pygame.init()
screen = pygame.display.set_mode((450, 450))
background = pygame.image.load("background.png").convert_
    alpha()
theremin = pygame.image.load("theremin.png").convert_alpha()
screen.blit(background, (0, 0))
screen.blit(theremin, (135, 50))
while True:
    pygame.display.update()
```

pygame.transform 模块提供了用于缩放和旋转 Surface 的函数。如果需要获取 Surface 上单个像素的颜色值，可以使用

pygame.surfarray 模块中的相关函数。

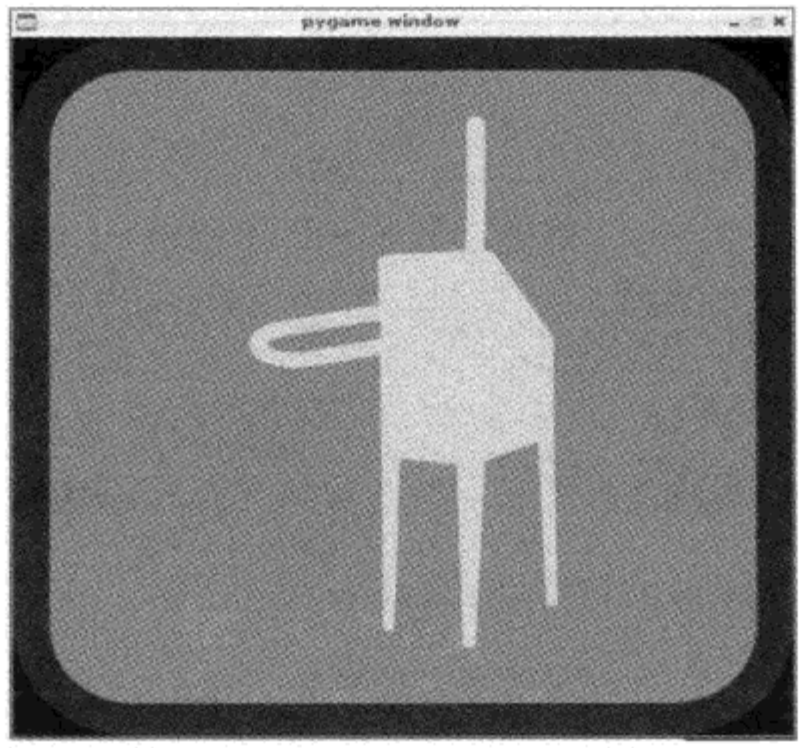


图4.1　把两张图片合并显示

新创建的 Surface 总是一个矩形的图片——尽管在上一个例子中通过使用图片的透明区域使 Surface 看上去是非矩形的，但 Surface 对象本身仍然是矩形的。如果想让 Surface 具有一个非矩形的边框（如用于进行像素级的碰撞检测），你可以使用 pygame.mask 模块通过创建另一个 Surface 对象设置一个蒙版。本章后面的有关游戏编程的参考资料中介绍了蒙版的具体使用方法。

在 Surface 上绘图

前面我们演示了用 Pygame 来绘制一个圆形，除此之外，pygame.draw 模块中还提供了绘制矩形、直线、弧线和椭圆形的功能。pygame.gfxdraw 模块则提供了另外一种绘制图形的方法，它提供了更多的绘制选项，但这个模块仍处于试验期，很多 API 以后还会发生变化。

如果要绘制一段文字，你需要先创建一个 Font 对象（由

pygame.font 模块提供），然后用它去加载字体文件并渲染文本。通过 pygame.font.get_fonts() 函数可以获取 Raspberry Pi 上可用的字体列表：

```
import pygame
pygame.init()
for fontname in pygame.font.get_fonts():
    print fontname
```

可以看到，Raspberry Pi 上默认预装了少量的字体。下面的代码通过使用 SysFont 对象加载 freeserif 字体渲染了一段文本：

```
import pygame
pygame.init()
screen = pygame.display.set_mode((725, 92))
font = pygame.font.SysFont("freeserif", 72, bold = 1)
textSurface = font.render("1 Theremin Per Child!", 1,
                          pygame.Color(255, 255, 255))
screen.blit(textSurface, (10, 10))
while True:
    pygame.display.update()
```

如果你需要使用更多的字体，可以用下面的命令安装：

```
sudo apt-get install ttf-mscorefonts-installer
sudo apt-get install ttf-liberation
```

处理事件与输入

在 Pygame 中，用户所触发的事件（如按下键盘上的某个键、移动或点击鼠标）会被捕获并生成 Event 对象放入消息队列，等待程序去做相应的处理。pygame.event 模块提供了从消息队列中获取未处理的消息并对之进行处理的功能。你还可以通过创建自己的消

息类型来实现一个消息系统。下面的例子中，我们使用消息队列来扩展前面的画圆的程序：在每一帧中，根据鼠标箭头所在的位置重新绘制一个圆形，鼠标箭头离窗口边框越近则绘制的圆的半径越大（图 4.2）。

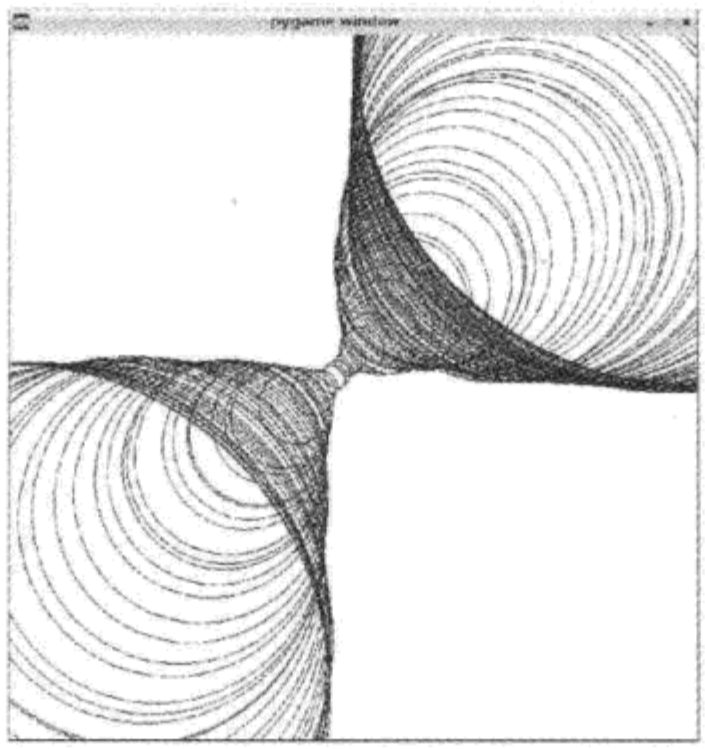


图4.2　Pygame消息处理例子的输出

```
import pygame
from pygame.locals import * ❶

width, height = 640, 640
radius = 0
mouseX, mouseY = 0, 0 ❷

pygame.init()
window = pygame.display.set_mode((width, height))
window.fill(pygame.Color(255, 255, 255))

fps = pygame.time.Clock() ❸

while True: ❹
    for event in pygame.event.get(): ❺
        if event.type == MOUSEMOTION: ❻
            mouseX, mouseY = event.pos
```

```
        if event.type == MOUSEBUTTONDOWN: ❼
            window.fill(pygame.Color(255, 255, 255))
    radius = (abs(width/2 - mouseX)+abs(height/2 -
mouseY))/2 + 1 ❽
    pygame.draw.circle(window, ❾
                       pygame.Color(255, 0, 0),
                       (mouseX, mouseY),
                       radius, 1)
    pygame.display.update()
    fps.tick(30)❿
```

❶ pygame.locals 模块中定义了包括 MOUSEMOTION 在内的一些常量。以这样的形式导入这个模块后，后续我们需要使用里面定义的常量时就不需要再添加 pygame. 前缀。

❷ 存放鼠标坐标的变量。

❸ 这个函数初始化并返回我们用于作为帧计数器的对象。通过使用 fps 变量（每秒的帧数，Frames per Second）你可以在每一帧图像显示完成后暂停一段时间，以便获取一个稳定的帧速。

❹ 一直循环，每循环一次就生成新的一帧。

❺ 循环处理消息队列，每循环一次，event 变量中会获得消息队列中的下一个消息。

❻ 如果获得的消息是一个鼠标移动消息，更新变量值，记录下鼠标的位置。

❼ 如果获得的消息是一个鼠标点击消息，清除窗口中的显示。

❽ 根据鼠标距离窗口中心的位置，计算出要绘制的圆形的半径。

❾ 画圆。

❿ 暂停一段时间，保证帧速为 30fps。

以下模块在处理消息和用户输入时也很有用。

pygame.time

监测时间的模块。

pygame.mouse

获取鼠标信息的模块。

pygame.key

获取键盘信息的模块，里面提供了很多常量用于表示每一个键。

pygame.joystick

操作游戏操纵杆的模块。

全屏模式

如果想要让你的程序以全屏模式运行，可以在设置显示模式时加上 pygame.FULLSCREEN 选项。在全屏模式下，不能使用 Control-C 键中断脚本的运行，所以你需要保证在程序中提供了某种退出程序的方式。

```
import pygame
import random
from time import sleep

running = True
Pygame.init()
screen=pygame.display.set_mode((0,0), pygame.FULLSCREEN)
while running:
    pygame.draw.circle(
       screen,
       pygame.Color(int(random.random()*255),
                    int(random.random()*255),
                    int(random.random()*255)),
       (int(random.random()*1500),
        int(random.random()*1500)),
       int(random.random()*500), 0)
    pygame.display.update()
    sleep(.1)
```

```
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            running = False
pygame.quit()
```

在这个脚本运行时，按键盘上的任意键就可以退出程序。

Sprite

在一个游戏中，可以把各种可移动、可操控的图形元素都当作 Sprite 对象来处理。`pygame.sprite` 模块提供了在屏幕上绘制 Sprite 和在多个 Sprite 之间进行碰撞检测的功能。多个 Sprite 可以被组合在一起，同时控制或更新。通过使用 Sprite 编写一个完整的游戏的方法已经超出本书的内容范围，可以参考“进一步学习”中的参考资料了解更多的细节。

当需要创建多个屏幕元素并且这些元素共享很多相似的代码时，你就可以考虑使用 Sprite。下面的代码演示了如何创建和更新多个 Sprite，程序的运行结果是在屏幕上绘制两个小球，这两个小球在运动到屏幕边缘时会反弹回来。如果要增加小球的数量，可以创建更多的 Sprite 对象并对其起始坐标、方向和速度进行不同的初始化。

```
import pygame

class Ball(pygame.sprite.Sprite):❶
    def __init__(self, x, y, xdir, ydir, speed):❷
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface([20, 20])
        self.image.fill(pygame.Color(255, 255, 255))
        pygame.draw.circle(self.image,
                           pygame.Color(255,0,0),
                           (10,10), 10, 0)
        self.rect = self.image.get_rect()
```

```
        self.x, self.y = x, y ❸
        self.xdir, self.ydir = xdir, ydir
        self.speed = speed

    def update(self): ❹
        self.x = self.x + (self.xdir * self.speed)
        self.y = self.y + (self.ydir * self.speed)
        if (self.x < 10) | (self.x > 490):
            self.xdir = self.xdir * -1
        if (self.y < 10) | (self.y > 490):
            self.ydir = self.ydir * -1
        self.rect.center = (self.x, self.y)

pygame.init()
fps = pygame.time.Clock()
window = pygame.display.set_mode((500, 500))
ball = Ball(100, 250, 1, 1, 5) ❺
ball2 = Ball(400, 10, -1, -1, 8)

while True:
    ball.update() ❻
    ball2.update()
    window.fill(pygame.Color(255,255, 255))
    window.blit(ball.image, ball.rect) ❼
    window.blit(ball2.image, ball2.rect)
    pygame.display.update()
    fps.tick(30)
```

❶ class关键字用于基于Sprite对象[①]创建一个新的Ball对象，你可以为这个新的对象定义绘制这个对象的方法以及更新这个对象时所需要的操作。

❷ 当用 Ball() 创建一个新的对象时，这个函数会被调用。

❸ 这些变量会作为 Ball 实例的一部分，每一个 Ball 对象会保存一份属于它自己的变量值。

① 严格意义上来说，这里的“对象”应该是“类”。在面向对象程序设计中，“类”表示对一类相似概念的抽象定义，“对象”是基于“类”生成的实体。——译者注

❹ update() 函数会在绘制每一帧画面时被调用，对象所在的位置会根据对象中记录的方向和速度做相应的变化。同时，还会检测是否遇到了窗口的边缘：如果遇到边缘，则把对应坐标轴上的方向进行颠倒。

❺ 以不同的起始位置、运动方向和运动速度创建两个 Ball 对象。

❻ 根据当前的位置、运动方向和速度，更新小球的位置。

❼ 在当前位置绘制小球。

播放声音

在 Pygame 中，你可以使用 pygame.mixer 模块加载声音文件并进行播放，也可以用 pygame.midi 模块向 Pi 上运行的 MIDI 程序或 USB 接口上连接的 MIDI 设备发送 MIDI 消息。下面的例子演示了如何播放一个 WAV 格式的音频文件，这个文件你可以从网上下载（*http://archive.org/details/WilhelmScreamSample*）。

```
import pygame.mixer
from time import sleep

pygame.mixer.init(48000, -16, 1, 1024)

sound = pygame.mixer.Sound("WilhelmScream.wav")
channelA = pygame.mixer.Channel(1)
channelA.play(sound)
sleep(2.0)
```

这个程序加载了音频文件（WAV 格式）并与一个通道关联，然后通过混音器在独立的进程[①]中播放每一个通道中的音频，因此，在同一时间可以同时播放多个声音。程序最后的 sleep() 函数调用是为了保证主程序在声音播放完毕后才退出。

① 确切地说应该是线程。——译者注

在第 8 章的发音板程序中，我们还会进一步介绍通过混音器播放音频文件的知识。Raspberry Pi 的 MIDI 功能同样让人着迷，你可以很容易地创建一个 MIDI 控制器程序。截至本书写作时，Raspberry Pi 上的软音源还比较粗糙，模拟音频输出效果也远不如 HDMI 输出，不过你还是可以通过使用一个 USB 接口的 MIDI 设备来获得较好的效果。下面的代码可以列出 Raspberry Pi 上连接的所有 MIDI 设备：

```
import pygame
import pygame.midi ❶

pygame.init()
pygame.midi.init() ❷
for id in range(pygame.midi.get_count()): ❸
    print pygame.midi.get_device_info(id) ❹
```

❶ midi 模块在导入 pygame 模块时不会自动导入。

❷ 同样地，midi 模块还需要单独进行初始化。

❸ get_count() 可以用于获取 Raspberry Pi 上所有 MIDI 设备的数目，包括 USB 设备、软音源或其他虚拟 MIDI 设备。

❹ 显示该设备相关的信息。

如果你在 Raspberry Pi 上连接了一个 MIDI 键盘，你会从这个程序中得到以下输出：

```
('ALSA', 'Midi Through Port-0', 0, 1, 0)
('ALSA', 'Midi Through Port-0', 1, 0, 0)
('ALSA', 'USB Uno MIDI Interface MIDI 1', 0, 1, 0)
('ALSA', 'USB Uno MIDI Interface MIDI 1', 1, 0, 0)
```

第 1 列的信息告诉你当前系统使用的是 ALSA（Advanced Linux Sound Architecture，高级 Linux 音频架构）音频系统，第 2 列

是对每一个 MIDI 设备的描述。最后 3 列数字表示这个端口是输入设备还是输出设备，以及这个设备当前是打开还是关闭的。在这个例子中，你可以认为是有 4 个不同的 MIDI 设备，端口号分别为 0~3。

MIDI Through Port-0 (0, 1, 0)

0 号端口，输出端口，用于向 Pi 上所运行的某一个软音源设备输出 MIDI 消息。

MIDI Through Port-1 (1, 0, 0)

1 号端口，输入端口，用于获取 Pi 上所运行的某一个 MIDI 控制程序发出的控制信息。

USB Uno MIDI Interface MIDI 1 (0, 1, 0)

2 号端口，输出端口，连接到 USB 口上的一个 MIDI 设备，如 MIDI 键盘。

USB Uno MIDI Interface MIDI 1 (1, 0 , 0)

3 号端口，输入端口，连接到 USB 口上的一个 MIDI 设备，如 MIDI 键盘。

有了这些基础知识，你就可以在 USB 接口连接一个 MIDI 键盘并使用 Pygame 来控制它了，示例代码如下：

```
import pygame
import pygame.midi
from time import sleep

instrument = 0❶
note = 74
volume = 127

pygame.init()
pygame.midi.init()
```

```
port = 2❷
midiOutput = pygame.midi.Output(port, 0)
midiOutput.set_instrument(instrument)
for note in range(0, 127):
    midiOutput.note_on(note, volume)❸
    sleep(.25)
    midiOutput.note_off(note, volume)
del midiOutput
pygame.midi.quit()
```

❶ 这些是 MIDI 消息中使用的编号，通常在 0~127 取值。

❷ 打开 2 号端口，这个端口即 USB 接口上所接的 MIDI 键盘的输出端口。

❸ 发送一个音符开始消息，暂停，发送一个音符结束消息。

由此可见，Raspberry Pi 具有很大的潜力可以成为一个音乐创作平台。

播放视频

使用 pygame.movie 模块可以用于播放视频，它所播放的视频文件必须使用 MPEG1 格式编码。如果你的视频文件是以其他格式编码的，可以使用 ffmpeg 工具对它进行格式转换（先用 sudo apt-get install ffmpeg 安装这个工具）。要回放视频，只需创建一个新的 Moive 对象并调用 play() 函数：

```
import pygame
from time import sleep
pygame.init()

screen = pygame.display.set_mode((320,240))
movie = pygame.movie.Movie("foo.mpg")
movie.play()
while True:
```

```
    if not(movie.get_busy()):
        print("rewind")
        movie.rewind()
    movie.play()
    if pygame.QUIT in [e.type for e in pygame.event.get()]:
        break
```

如果所播放的视频包含声音，则需要在播放之前关闭 Pygame 的混音器。方法是在开始播放视频前调用：

```
pygame.mixer.quit()
```

更多示例

pygame.examples 模块中提供了很多完整的示例程序。可以在 */usr/share/pyshared/pygame/examples* 目录中找到这些程序的源代码。

进一步学习

Pygame 官方文档

（*http://www.pygame.org/docs/*）

Pygame 的官方文档比较散乱，不过希望你在学习完本章内容后，能从该文档中找到你所需的信息。

Al Sweigart 所著的 *Making Games with Python & Pygame* 和 *Invent Your Own Computer Gmaes with Python*

（*http://inventwithpython.com/*）

两本都是以知识共享许可协议（Creative Commons）授权的书，这两本书中开发的游戏都已经预装在了 Raspberry Pi 中。