第5章

Tkinter图形绘制——图形版发牌程序

第4章以Tkinter模块为例学习建立一些简单的GUI(图形用户界面),使编写的程序像大家平常熟悉的那些程序一样,有窗体、按钮之类的图形界面,本书后面章节的游戏界面也都使用Tkinter开发。在游戏开发中不仅有按钮、文本框等,实际上需要绘制大量图形图像,本章介绍使用Canvas技术实现游戏中画面的绘制任务。

5.1 扑克牌发牌窗体程序功能介绍

在扑克牌游戏中有4位牌手,计算机随机将52张牌(不含大小王)发给4位牌手,并在屏幕上显示每位牌手的牌。程序的运行效果如图5-1所示。以Tkinter模块中图形Canvas绘制为例学习建立一些简单的GUI(图形用户界面)游戏界面。

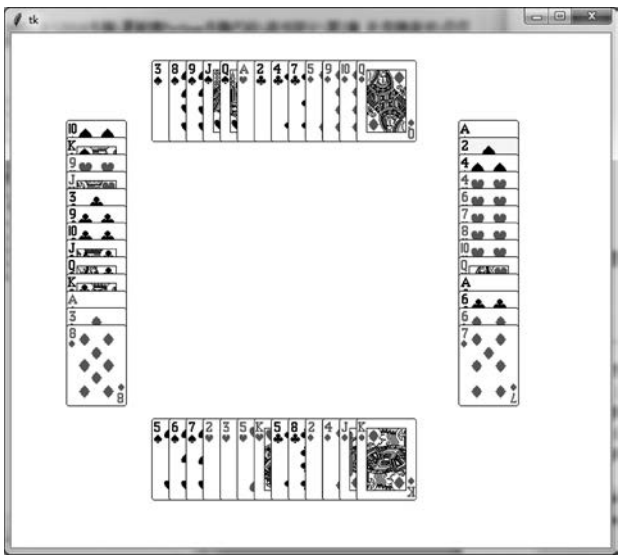

图5-1 扑克牌发牌运行效果

下面将介绍开发扑克牌发牌窗体程序的思路和Canvas关键技术。

5.2 程序设计的思路

将游戏中的52张牌，按梅花0～12，方块13～25，红桃26～38，黑桃39～51的顺序编号并存储在pocker列表（未洗牌之前），列表元素存储的则是某张牌（实际上是牌的编号）。同时按此编号顺序存储在扑克牌图片imgs列表中。也就是说imgs[0]存储梅花A的图片，imgs[1]存储梅花2的图片，imgs[14]存储方块2的图片。

发牌后，根据每位牌手（p1，p2，p3，p4）各自牌的编号列表，从imgs获取对应牌的图片并使用create_image((x坐标，y坐标)，image=图像文件）显示在指定位置。


视频讲解

5.3 Canvas图形绘制技术

Canvas为Tkinter提供了绘图功能，其绘制图形函数包括线形、圆形、椭圆、多边形、图片等几何图案绘制。使用Canvas进行绘图时，所有的操作都是通过Canvas。

5.3.1 Canvas画布组件

Canvas（画布）是一个长方形的区域，用于图形绘制或复杂的图形界面布局。可以在画布上绘制图形、文字，放置各种组件和框架。

1. 创建Canvas对象

可以使用下面的方法创建一个Canvas对象。

```
Canvas对象 = Canvas(窗口对象, 选项, …)
```

Canvas画布中的常用选项如表5-1所示。

表5-1 Canvas画布的常用选项

属　性	说　明
bd	指定画布的边框宽度，单位是像素
bg	指定画布的背景颜色
confine	指定画布在滚动区域外是否可以滚动。默认为True，表示不能滚动
cursor	指定画布中的鼠标指针，例如arrow、circle、dot
height	指定画布的高度
highlightcolor	选中画布时的背景色
relief	指定画布的边框样式，可选值包括sunken、raised、groove、ridge
scrollregion	指定画布的滚动区域的元组（w，n，e，s）

2. 显示Canvas对象

在模块中显示Canvas对象的方法如下：

```
Canvas 对象.pack()
```

例如创建一个白色背景、宽度为 300、高度为 120 的 Canvas 画布，代码如下。

```
from tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white', width = 300, height = 120)
cv. create_line(10,10,100,80,width = 2, dash = 7)      #绘制直线
cv.pack()                                              #显示画布
root.mainloop()
```

5.3.2 Canvas 上的图形对象

1. 绘制图形对象

Canvas 画布上可以绘制各种图形对象，通过调用相应绘制函数即可实现，函数及功能如下。

create_arc()：绘制圆弧。

create_line()：绘制直线。

create_bitmap()：绘制 Python 内置的位图。

create_image()：绘制位图图像。

create_oval()：绘制椭圆。

create_polygon()：绘制多边形。

create_window()：绘制子窗口。

create_text()：创建一个文字对象。

Canvas 上每个绘制对象都有一个标识 id（整数），使用绘制函数创建绘制对象时，返回绘制对象的 id。例如：

```
id1 = cv. create_line(10,10,100,80,width = 2, dash = 7)      #绘制直线
```

id1 可以得到绘制对象直线 id。

在创建图形对象时可以使用属性 tags 设置图形对象的标记(tag)，例如：

```
rt = cv.create_rectangle(10,10,110,110, tags = 'r1')
```

上面的语句指定矩形对象 rt 具有一个标记 r1。

也可以同时设置多个标记(tag)，例如：

```
rt = cv.create_rectangle(10,10,110,110, tags = ('r1','r2','r3'))
```

上面的语句指定矩形对象 rt 有三个标记：r1、r2 和 r3。

指定标记后，使用 find_withtag()方法可以获取到指定 tag 的图形对象，然后设置图形对象的属性。find_withtag()方法的语法如下：

```
Canvas 对象.find_withtag(tag 名)
```

find_withtag()方法返回一个图形对象数组，其中包含所有具有 tag 名的图形对象。

使用 itemconfig()方法可以设置图形对象的属性，语法如下：

```
Canvas 对象. itemconfig(图形对象, 属性 1 = 值 1, 属性 2 = 值 2, …)
```

【例 5-1】 使用属性 tags 设置图形对象标记的例子。

```
from tkinter import *
root = Tk()
#创建一个 Canvas,设置其背景色为白色
cv = Canvas(root, bg = 'white', width = 200, height = 200)
#使用 tags 指定给第一个矩形指定三个 tag
rt = cv.create_rectangle(10,10,110,110, tags = ('r1','r2','r3'))
cv.pack()
cv.create_rectangle(20,20,80,80, tags = 'r3')      #使用 tags 指定给第 2 个矩形指定一个 tag
#将所有与 tag('r3')绑定的 item 边框颜色设置为蓝色
for item in cv.find_withtag('r3'):
    cv.itemconfig(item,outline = 'blue')
root.mainloop()
```

下面学习使用绘制函数绘制各种图形对象。

2. 绘制圆弧

使用 create_arc()方法可以创建一个圆弧对象，可以是一个和弦、饼图扇区或者一个简单的弧，其具体语法如下：

```
Canvas 对象. create_arc(弧外框矩形左上角的 x 坐标, 弧外框矩形左上角的 y 坐标, 弧外框矩形右下角的 x 坐标, 弧外框矩形右下角的 y 坐标, 选项, …)
```

创建圆弧常用选项：outline 指定圆弧边框颜色，fill 指定填充颜色，width 指定圆弧边框的宽度，start 指定起始角度，extent 指定角度偏移量而不是终止角度。

【例 5-2】 使用 create_ arc()方法创建圆弧，其运行效果如图 5-2 所示。

```
from tkinter import *
root = Tk()
#创建一个 Canvas,设置其背景色为白色
cv = Canvas(root,bg = 'white')
cv.create_arc((10,10,110,110),)          #使用默认参数创建一个圆弧,结果为 90°的扇形
d = {1:PIESLICE,2:CHORD,3:ARC}
for i in d:
    #使用三种样式,分别创建了扇形、弓形和弧形
    cv.create_arc((10,10 + 60 * i,110,110 + 60 * i),style = d[i])
    print(i,d[i])
```

```
#使用 start/extent 分别指定圆弧起始角度与偏移角度
cv.create_arc(
        (150,150,250,250),
        start = 10,                    #指定起始角度
        extent = 120                   #指定角度偏移量(逆时针)
        )
cv.pack()
root.mainloop()
```

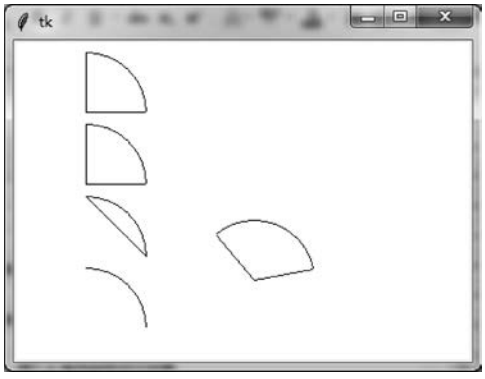

图 5-2　创建圆弧对象运行效果

3. 绘制线条

使用 create_line()方法可以创建一个线条对象,具体语法如下:

```
line = canvas.create_line(x0, y0, x1, y1, …, xn, yn,选项)
```

参数 x0、y0、x1、y1、……、xn、yn 是线段的端点。

创建线段常用选项:width 指定线段宽度,arrow 指定是否使用箭头(没有箭头为 none,起点有箭头为 first,终点有箭头为 last,两端有箭头为 both),fill 指定线段颜色,dash 指定线段为虚线(其整数值决定虚线的样式)。

【例 5-3】 使用 create_line()方法创建线条对象,其运行效果如图 5-3 所示。

```
from tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white', width = 200, height = 100)
cv.create_line(10, 10, 100, 10, arrow = 'none')          #绘制没有箭头的线段
cv.create_line(10, 20, 100, 20, arrow = 'first')         #绘制起点有箭头的线段
cv.create_line(10, 30, 100, 30, arrow = 'last')          #绘制终点有箭头的线段
cv.create_line(10, 40, 100, 40, arrow = 'both')          #绘制两端有箭头的线段
cv. create_line(10,50,100,100,width = 3, dash = 7)       #绘制虚线
cv.pack()
root.mainloop()
```

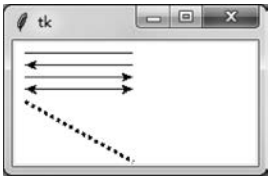

图 5-3　创建线条对象运行效果

4. 绘制矩形

使用 create_ rectangle()方法可以创建矩形对象，具体语法如下：

```
Canvas 对象. create_rectangle(矩形左上角的 x 坐标, 矩形左上角的 y 坐标, 矩形右下角的 x 坐标,
矩形右下角的 y 坐标, 选项, …)
```

创建矩形对象时的常用选项：outline 指定边框颜色，fill 指定填充颜色，width 指定边框的宽度，dash 指定边框为虚线，stipple 使用指定自定义画刷填充矩形。

【例 5-4】 使用 create_rectangle()方法创建矩形对象，其运行效果如图 5-4 所示。

```
from tkinter import *
root = Tk()
#创建一个 Canvas,设置其背景色为白色
cv = Canvas(root, bg = 'white', width = 200, height = 100)
cv.create_rectangle(10,10,110,110, width = 2,fill = 'red')    #指定矩形的填充色为红色,
                                                              #宽度为 2
cv.create_rectangle(120, 20,180, 80, outline = 'green')       #指定矩形的边框颜色为绿色
cv.pack()
root.mainloop()
```

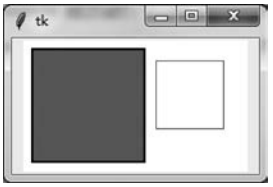

图 5-4　创建矩形对象运行效果

5. 绘制多边形

使用 create_polygon()方法可以创建一个多边形对象，可以是一个三角形、矩形或者任意一个多边形，具体语法如下：

```
Canvas 对象. create_polygon(顶点 1 的 x 坐标, 顶点 1 的 y 坐标, 顶点 2 的 x 坐标, 顶点 2 的 y 坐
标, …, 顶点 n 的 x 坐标, 顶点 n 的 y 坐标, 选项, …)
```

创建多边形对象时的常用选项：outline 指定边框颜色，fill 指定填充颜色，width 指定边框的宽度，smooth 指定多边形的平滑程度（等于 0 表示多边形的边是折线；等于 1 表示

多边形的边是平滑曲线)。

【例 5-5】 分别创建三角形、正方形、对顶三角形对象,其运行效果如图 5-5 所示。

```
from tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white', width = 300, height = 100)
cv.create_polygon(35,10,10,60,60,60, outline = 'blue', fill = 'red', width = 2)#等腰三角形
cv.create_polygon(70,10,120,10,120,60, outline = 'blue', fill = 'white', width = 2)
                                                                    #直角三角形
cv.create_polygon(130,10,180,10,180,60, 130,60, width = 4)          #黑色填充正方形
cv.create_polygon(190,10,240,10,190,60, 240,60, width = 1)          #对顶三角形
cv.pack()
root.mainloop()
```

6. 绘制椭圆

使用 create_oval()方法可以创建一个椭圆对象,具体语法如下:

```
Canvas 对象. create_oval(包裹椭圆的矩形左上角 x 坐标, 包裹椭圆的矩形左上角 y 坐标, 包裹椭圆的矩形右下角 x 坐标, 包裹椭圆的矩形右下角 y 坐标, 选项, …)
```

创建椭圆对象时的常用选项:outline 指定边框颜色,fill 指定填充颜色,width 指定边框宽度。如果包裹椭圆的矩形是正方形,绘制后则是一个圆形。

【例 5-6】 分别创建椭圆和圆形,其运行效果如图 5-6 所示。

```
from tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white', width = 200, height = 100)
cv.create_oval(10,10,100,50, outline = 'blue', fill = 'red', width = 2)     #椭圆
cv.create_oval(100,10,190,100, outline = 'blue', fill = 'red', width = 2)   #圆形
cv.pack()
root.mainloop()
```

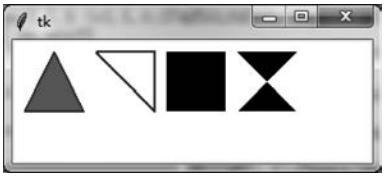

图 5-5 创建三角形对象运行效果

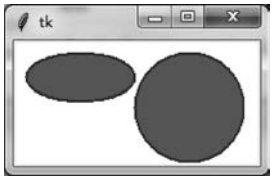

图 5-6 创建椭圆和圆形运行效果

7. 绘制文字

使用 create_text()方法可以创建一个文字对象,具体语法如下:

```
文字对象 = Canvas 对象.create_text((文本左上角的 x 坐标,文本左上角的 y 坐标), 选项, …)
```

创建文字对象时的常用选项：text 指定文字对象的文本内容，fill 指定文字颜色，anchor 控制文字对象的位置（其取值'w'表示左对齐，'e'表示右对齐，'n'表示顶对齐，'s'表示底对齐，'nw'表示左上对齐，'sw'表示左下对齐，'se'表示右下对齐，'ne'表示右上对齐，'center'表示居中对齐，anchor 默认值为'center'），justify 设置文字对象中文本的对齐方式（其取值'left'表示左对齐，'right'表示右对齐，'center'表示居中对齐，justify 默认值为'center'）。

【例 5-7】 创建文本的例子，其运行效果如图 5-7 所示。

```
from tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white', width = 200, height = 100)
cv.create_text((10,10), text = 'Hello Python', fill = 'red', anchor = 'nw')
cv.create_text((200,50), text = '你好,Python', fill = 'blue', anchor = 'se')
cv.pack()
root.mainloop()
```

select_from()方法用于指定选中文本的起始位置，具体用法如下：

```
Canvas 对象. select_from(文字对象, 选中文本的起始位置)
```

select_to()方法用于指定选中文本的结束位置，具体用法如下：

```
Canvas 对象. select_ to(文字对象, 选中文本的结束位置)
```

【例 5-8】 选中文本的例子，其运行效果如图 5-8 所示。

图 5-7 创建文本运行效果

图 5-8 选中文本运行效果

```
from tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white', width = 200, height = 100)
txt = cv.create_text((10,10), text = '中原工学院计算机学院', fill = 'red', anchor = 'nw')
#设置文本的选中起始位置
cv.select_from(txt,5)
#设置文本的选中结束位置
cv.select_to(txt,9)        #选中"计算机学院"
cv.pack()
root.mainloop()
```

8. 绘制位图和图像

1）绘制位图

使用 create_bitmap()方法可以绘制 Python 内置的位图，具体方法如下：

```
Canvas 对象. create_bitmap((x 坐标,y 坐标),bitmap = 位图字符串, 选项, …)
```

其中,(x 坐标,y 坐标)是位图放置的中心坐标;常用选项有 bitmap、activebitmap 和 disabledbitmap,分别用于指定正常、活动、禁用状态显示的位图。

2) 绘制图像

在游戏开发中需要使用大量图像,采用 create_image()方法可以绘制图形图像,具体方法如下:

```
Canvas 对象. create_image((x 坐标,y 坐标), image = 图像文件对象, 选项, …)
```

其中,(x 坐标,y 坐标)是图像放置的中心坐标;常用选项有 image、activeimage 和 disabledimage 用于指定正常、活动、禁用状态显示的图像。

注意:可以如下使用 PhotoImage()函数来获取图像文件对象。

```
img1 = PhotoImage(file = 图像文件)
```

例如,img1 = PhotoImage(file = 'C:\\aa. png')可以获取笑脸图形。Python 支持图像文件格式一般为. png 和. gif。

【例 5-9】 绘制图像示例,运行效果如图 5-9 所示。

```
from tkinter import *
root = Tk()
cv = Canvas(root)
img1 = PhotoImage(file = 'C:\\aa.png')            #笑脸
img2 = PhotoImage(file = 'C:\\2.gif')             #方块 A
img3 = PhotoImage(file = 'C:\\3.gif')             #梅花 A
cv.create_image((100,100),image = img1)           #绘制笑脸
cv.create_image((200,100),image = img2)           #绘制方块 A
cv.create_image((300,100),image = img3)           #绘制梅花 A
d = {1:'error',2:'info',3:'question',4:'hourglass',5:'questhead',
     6:'warning',7:'gray12',8:'gray25',9:'gray50',10:'gray75'}#字典
#cv.create_bitmap((10,220),bitmap = d[1])
#以下遍历字典绘制 Python 内置的位图
for i in d:
    cv.create_bitmap((20 * i,20),bitmap = d[i])
cv.pack()
root.mainloop()
```

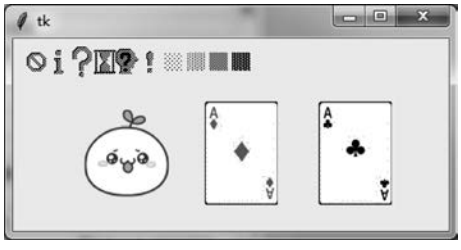

图 5-9 绘制图像示例

学会使用绘制图像，就可以开发图形版的扑克牌游戏了。

9. 修改图形对象的坐标

使用 coords()方法可以修改图形对象的坐标，具体方法如下：

```
Canvas 对象.coords(图形对象,(图形左上角的 x 坐标, 图形左上角的 y 坐标, 图形右下角的 x 坐标, 图形右下角的 y 坐标))
```

因为可以同时修改图形对象的左上角的坐标和右下角的坐标，所以可以缩放图形对象。

注意：如果图形对象是图像文件，则只能指定图像中心点坐标，而不能指定图像左上角的坐标和右下角的坐标，故不能缩放图像。

【例 5-10】 修改图形对象的坐标示例，运行效果如图 5-10 所示。

```
from tkinter import *
root = Tk()
cv = Canvas(root)
img1 = PhotoImage(file = 'C:\\aa.png')                    #笑脸
img2 = PhotoImage(file = 'C:\\2.gif')                     #方块 A
img3 = PhotoImage(file = 'C:\\3.gif')                     #梅花 A
rt1 = cv.create_image((100,100),image = img1)             #绘制笑脸
rt2 = cv.create_image((200,100),image = img2)             #绘制方块 A
rt3 = cv.create_image((300,100),image = img3)             #绘制梅花 A
#重新设置方块 A(rt2 对象)的坐标
cv.coords(rt2,(200,50))                                   #调整 rt2 对象方块 A 位置
rt4 = cv.create_rectangle(20,140,110,220,outline = 'red', fill = 'green')      #正方形对象
cv.coords(rt4,(100,150,300,200))                          #调整 rt4 对象位置
cv.pack()
root.mainloop()
```

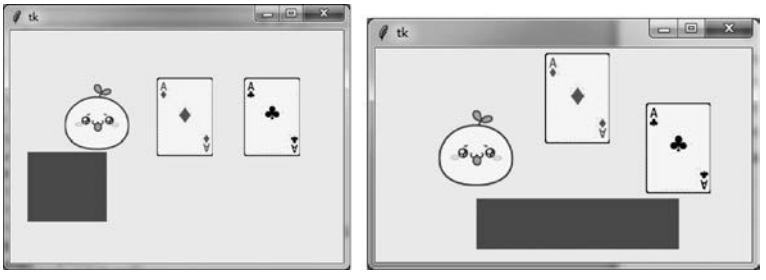

图 5-10　调整图形对象位置之前和之后的效果

10. 移动指定图形对象

使用 move()方法可以修改图形对象的坐标，具体方法如下：

```
Canvas 对象. move(图形对象, x 坐标偏移量, y 坐标偏移量)
```

【例 5-11】 移动指定图形对象示例，运行效果如图 5-11 所示。

```
from tkinter import *
root = Tk()
#创建一个 Canvas,设置其背景色为白色
cv = Canvas(root, bg = 'white', width = 200, height = 120)
rt1 = cv.create_rectangle(20,20,110,110,outline = 'red',stipple = 'gray12',fill = 'green')
cv.pack()
rt2 = cv.create_rectangle(20,20,110,110,outline = 'blue')
cv.move(rt1,20, -10)          #移动 rt1
cv.pack()
root.mainloop()
```

为了对比移动图形对象的效果,程序在同一位置绘制了2个矩形,其中矩形 rt1 有背景花纹,rt2 无背景填充。然后用 move()方法移动 rt1,将被填充的矩形 rt1 向右移动 20 像素,向上移动 10 像素,则出现图 5-11 所示的效果。

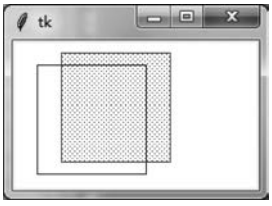

图 5-11　移动指定图形对象运行效果

11. 删除图形对象

使用 delete()方法可以删除图形对象,具体方法如下:

```
Canvas 对象. delete (图形对象)
```

例如:

```
cv.delete(rt1)          #删除 rt1 图形对象
```

12. 缩放图形对象

使用 scale()方法可以缩放图形对象,具体方法如下:

```
Canvas 对象.scale(图形对象, x 轴偏移量,y 轴偏移量,x 轴缩放比例,y 轴缩放比例)
```

【例 5-12】 缩放图形对象示例,对相同图形对象进行放大或缩小,运行效果如图 5-12 所示。

```
from tkinter import *
root = Tk()
#创建一个 Canvas,设置其背景色为白色
cv = Canvas(root, bg = 'white', width = 200, height = 300)
rt1 = cv.create_rectangle(10,10,110,110,outline = 'red',stipple = 'gray12', fill = 'green')
rt2 = cv.create_rectangle(10,10,110,110,outline = 'green',stipple = 'gray12', fill = 'red')
cv.scale(rt1,0,0,1,2)          #y 方向放大一倍
cv.scale(rt2,0,0,0.5,0.5)      #缩小一半大小
cv.pack()
root.mainloop()
```

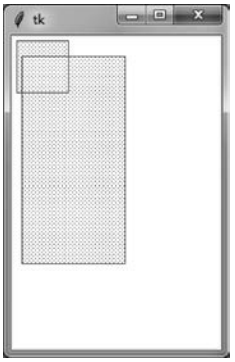

图 5-12　缩放图形对象运行效果


视频讲解

5.4　图形版发牌程序设计的步骤

图形版发牌程序导入相关模块：

```
from tkinter import *
import random
```

假设有 52 张牌，不包括大小王。

```
n = 52
```

gen_pocker(n) 函数实现对 n 张牌的洗牌。方法是随机产生两个下标，将此下标的列表元素进行交换达到洗牌目的。列表元素存储的是某张牌（实际上是牌的编号）。

```
def gen_pocker(n):
    x = 100
    while(x > 0):
        x = x - 1
        p1 = random.randint(0,n - 1)
        p2 = random.randint(0,n - 1)
        t = pocker[p1]
        pocker[p1] = pocker[p2]
        pocker[p2] = t
    return pocker
```

以下是主程序的实现步骤。

将要发的 52 张牌，按梅花 0～12，方块 13～25，红桃 26～38，黑桃 39～51 的顺序编号并存储在 pocker 列表（未洗牌之前）。

```
pocker = [i for i in range(n)]
```

调用 gen_pocker(n)函数实现对 n 张牌的洗牌。

```
pocker = gen_pocker(n)                                    #实现对 n 张牌的洗牌
print(pocker)

(player1,player2,player3,player4) = ([],[],[],[])   #4 位牌手各自牌的图片列表
(p1,p2,p3,p4) = ([],[],[],[])                             #4 位牌手各自牌的编号列表
root = Tk()
#创建一个 Canvas,设置其背景色为白色
cv = Canvas(root, bg = 'white', width = 700, height = 600)
```

将要发的 52 张牌图片,按梅花 0～12,方块 13～25,红桃 26～38,黑桃 39～51 的顺序编号存储到扑克牌图片 imgs 列表中。也就是说 imgs[0]存储梅花 A 的图片“1-1.gif”,imgs[1]存储梅花 2 的图片“1-2.gif”,imgs[14]存储方块 2 的图片“2-2.gif”。目的是根据牌的编号找到对应的图片。

```
imgs = []
for i in range(1,5):
    for j in range(1,14):
        imgs.insert((i - 1) * 13 + (j - 1),PhotoImage(file = str(i) + '-' + str(j) + '.gif'))
```

实现每人发 13 张牌,每轮发 4 张,一位牌手发一张,总计有 13 轮发牌。

```
for x in range(13):          #13 轮发牌
    m = x * 4
    p1.append(pocker[m])
    p2.append(pocker[m + 1])
    p3.append(pocker[m + 2])
    p4.append(pocker[m + 3])
```

对牌手的牌排序,就是相当于理牌,同花色在一起。

```
p1.sort()            #牌手的牌排序
p2.sort()
p3.sort()
p4.sort()
```

根据每位牌手手中牌的编号绘制显示对应的图片。

```
for x in range(0,13):
    img = imgs[p1[x]]
    player1.append(cv.create_image((200 + 20 * x,80),image = img))
    img = imgs[p2[x]]
    player2.append(cv.create_image((100,150 + 20 * x),image = img))
    img = imgs[p3[x]]
    player3.append(cv.create_image((200 + 20 * x,500),image = img))
    img = imgs[p4[x]]
    player4.append(cv.create_image((560,150 + 20 * x),image = img))
print("player1:",player1)
print("player2:",player2)
print("player3:",player3)
```

```
print("player4:",player4)
cv.pack()
root.mainloop()
```

至此完成图形版发牌程序的设计。

5.5 拓展练习——弹球小游戏

上面图形版发牌程序的画面是静止的。但在游戏开发中，游戏界面中物体会不断移动，例如小球下落、坦克移动的动画效果，这些效果是在游戏开发中通过画面不断更新实现的。下面以弹球小游戏为例进行说明。

用 Python 实现的弹球小游戏，可实现通过键盘左右方向键控制底部挡板左右移动或通过鼠标拖动底部挡板左右移动，以及小球碰撞到移动的挡板时反弹的游戏功能。如果小球落地则游戏结束。游戏界面如图 5-13 所示。

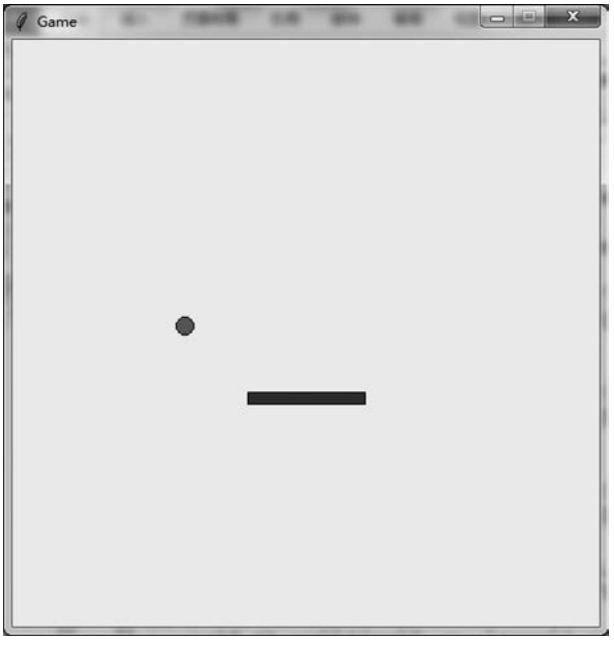

图 5-13　弹球小游戏

为弹球小游戏设计两个类。

1. Ball 弹球类

Ball 弹球类实现移动反弹功能。其中，draw(self)负责移动弹球 Ball，hit_paddle(self, pos)实现和挡板碰撞检测。

```
class Ball:
  def __init__(self, canvas, paddle, color):                    #构造函数
```

```
        self.canvas = canvas
        self.paddle = paddle
        self.id = canvas.create_oval(10, 10, 25, 25, fill = color)
        self.canvas.move(self.id, 245, 100)
        startx = [ - 3, - 2, - 1, 1, 2, 3]
        random.shuffle(startx)                                  #随机产生 x 方向速度
        self.x = startx[0]
        self.y = - 3                                            #y 方向速度(下落速度)
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False                                 #是否触底
    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)                       #获取小球左上角和右下角坐标
                                                                #(top - left bottom - right)
        if (pos[1] < = 0 or self.hit_paddle(pos) == True):      #小球 y 触顶或者小球和挡板碰撞
            self.y = - self.y                                   #y 方向反向
        if (pos[0] < = 0 or pos[2] > = self.canvas_width):      #小球左右方向碰壁
            self.x = - self.x                                   #x 方向反向
        if (pos[3] > = self.canvas_height):                     #超过底部
            self.hit_bottom = True
    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if (pos[2] > = paddle_pos[0] and pos[0] < = paddle_pos[2]):
            if (pos[3] > = paddle_pos[1] and pos[3] < = paddle_pos[3]):      #和挡板碰撞
                return True
        return False
```

2. Paddle 挡板类

Paddle 挡板类实现底部挡板功能。其中，draw(self)负责移动挡板，hit_paddle(self，pos)实现和小球碰撞检测。同时对挡板添加鼠标事件绑定。

```
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill = color)
        self.x = 0
        self.canvas.move(self.id, 200, 300)
        self.canvas_width = self.canvas.winfo_width()
        self.canvas.bind_all("< Key - Left >", self.turn_left)
        self.canvas.bind_all("< Key - Right >", self.turn_right)
        self.canvas.bind("< Button - 1 >", self.turn)           #鼠标单击事件
        self.canvas.bind("< B1 - Motion >", self.turnmove)      #鼠标拖动事件

    def draw(self):
        pos = self.canvas.coords(self.id)
        if (pos[0] + self.x > = 0 and pos[2] + self.x < = self.canvas_width):
```

```
        self.canvas.move(self.id, self.x, 0)
    def turn_left(self, event):
        self.x = -4
    def turn_right(self, event):
        self.x = 4
    def turn(self, event):                                    #鼠标单击事件函数
        print("clicked at", event.x, event.y)
        self.mousex = event.x
        self.mousey = event.y
    def turnmove(self, event):                                #鼠标拖动事件函数
        print("现在位置: ", event.x, event.y)
        self.x = event.x - self.mousex
        self.mousex = event.x
```

3. 主程序

建立无限死循环，实现不断重新绘制 Ball 和 Paddle。如果弹球碰到底部则退出循环，游戏结束。

```
from tkinter import *
import random
import time
tk = Tk()
tk.title("Game")
tk.resizable(0, 0)#not resizable
tk.wm_attributes("-topmost", 1)#at top
canvas = Canvas(tk, width=500, height=500, bd=0, highlightthickness=0)
canvas.pack()
tk.update()
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, 'red')
while True:
    if (ball.hit_bottom == False):          #弹球是否碰到底部
        ball.draw()
        paddle.draw()
        tk.update()
        time.sleep(0.01)                    #游戏画面更新时间间隔 0.01 秒
    else:                                   #游戏循环结束
        break
```

至此弹球小游戏程序设计完成，玩家可以拖动挡板控制小球的反弹。

5.6 图形界面应用案例——关灯游戏

关灯游戏是很有趣的益智游戏，玩家通过单击可以关闭或打开一盏灯。关闭（或打开）一盏灯的同时，也会触动其四周（上、下、左、右）的灯的开关，改变它们的状态，成功关闭所有

的灯即可过关。游戏的运行效果如图 5-14 所示。

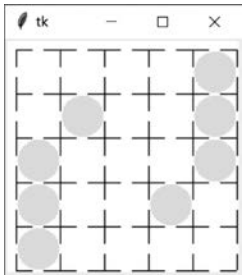

图 5-14　关灯游戏运行效果

分析：游戏中采用二维列表存储灯的状态，'you'表示灯亮(图中含有圆的方格)，'wu'表示灯灭(背景色的方格)。在 Canvas 画布单击事件中，获取鼠标单击位置从而换算成棋盘位置(x1，y1)，并处理四周的灯的状态转换。

代码如下：

```
from tkinter import *
from tkinter import messagebox
root = Tk()
l =[ ['wu', 'wu', 'you', 'you', 'you'],
     ['wu', 'you', 'wu', 'wu', 'wu'],
     ['wu', 'wu', 'wu', 'wu', 'wu'],
     ['wu', 'wu', 'wu', 'you', 'wu'],
     ['you', 'you', 'you', 'wu', 'wu']]
#绘制灯的状态图
def huaqi():
    for i in range(0, 5):
        for u in range(0, 5):
            if l[i][u] == 'you':
                cv.create_oval(i * 40 + 10, u * 40 + 10, (i + 1) * 40 + 10, (u + 1) *
                               40 + 10,outline = 'white', fill = 'yellow', width = 2) #灯亮
            else:
                cv.create_oval(i * 40 + 10, u * 40 + 10, (i + 1) * 40 + 10, (u + 1) *
                               40 + 10,outline = 'white', fill = 'white', width = 2) #灯灭
#反转(x1,y1)处灯的状态
def reserve(x1,y1):
    if l[x1][y1] == 'wu':
            l[x1][y1] = 'you'
    else:
        l[x1][y1] = 'wu'
#单击事件函数
def luozi(event):
    x1 = (event.x - 10) // 40
    y1 = (event.y - 10) // 40
    print(x1, y1)
    reserve(x1,y1) #反转(x1,y1)处灯的状态
    #以下反转(x1,y1)周围的灯的状态
```

```
        #将左侧灯的状态反转
        if x1!= 0:
            reserve(x1 - 1,y1)
        #将右侧灯的状态反转
        if x1!= 4:
            reserve(x1 + 1,y1)
        #将上方灯的状态反转
        if y1!= 0:
            reserve(x1,y1 - 1)
        #将下方灯的状态反转
        if y1!= 4:
            reserve(x1,y1 + 1)
        huaqi()

    #主程序
    cv = Canvas(root, bg = 'white', width = 210, height = 210)
    for i in range(0, 6):                          #绘制网格线
        cv.create_line(10, 10 + i * 40, 210, 10 + i * 40, arrow = 'none')
        cv.create_line(10 + i * 40, 10, 10 + i * 40, 210, arrow = 'none')
    huaqi()                                        #绘制灯的状态图
    p = 0
    for i in range(0, 5):
        for u in l[i]:
            if u == 'wu':
                p = p + 1
    if p == 25:
        messagebox.showinfo('win','你过关了')    #显示游戏过关信息的消息窗口
    cv.bind('<Button - 1>', luozi)
    cv.pack()
    root.mainloop()
```

思考与练习

1. 实现 15×15 棋盘的五子棋游戏界面的绘制。

2. 实现国际象棋界面的绘制。

3. 实现推箱子游戏界面的绘制。

4. 编写程序，实现井字棋游戏。该游戏界面是一个由 3×3 方格构成的棋盘，游戏双方各执一种颜色的棋子，在规定的方格内轮流布棋，如果一方在横、竖、斜三个方向上都形成 3 子相连则该方胜利。