图形用户界面和游戏开发

基于tkinter模块的GUI

GUI是图形用户界面的缩写，图形化的用户界面对使用过计算机的人来说应该都不陌生，在此也无需进行赘述。Python默认的GUI开发模块是tkinter（在Python 3以前的版本中名为Tkinter），从这个名字就可以看出它是基于Tk的，Tk是一个工具包，最初是为Tcl设计的，后来被移植到很多其他的脚本语言中，它提供了跨平台的GUI控件。当然Tk并不是最新和最好的选择，也没有功能特别强大的GUI控件，事实上，开发GUI应用并不是Python最擅长的工作，如果真的需要使用Python开发GUI应用，wxPython、PyQt、PyGTK等模块都是不错的选择。

基本上使用tkinter来开发GUI应用需要以下5个步骤：

1. 导入tkinter模块中我们需要的东西。
2. 创建一个顶层窗口对象并用它来承载整个GUI应用。
3. 在顶层窗口对象上添加GUI组件。
4. 通过代码将这些GUI组件的功能组织起来。
5. 进入主事件循环(main loop)。

下面的代码演示了如何使用tkinter做一个简单的GUI应用。

```
import tkinter
import tkinter.messagebox


def main():
    flag = True

    # 修改标签上的文字
    def change_label_text():
        nonlocal flag
        flag = not flag
        color, msg = ('red', 'Hello, world!')\
            if flag else ('blue', 'Goodbye, world!')
        label.config(text=msg, fg=color)

    # 确认退出
    def confirm_to_quit():
        if tkinter.messagebox.askokcancel('温馨提示', '确定要退出吗?'):
            top.quit()

    # 创建顶层窗口
    top = tkinter.Tk()
    # 设置窗口大小
    top.geometry('240x160')
    # 设置窗口标题
    top.title('小游戏')
    # 创建标签对象并添加到顶层窗口
    label = tkinter.Label(top, text='Hello, world!', font='Arial -32', fg='red')
    label.pack(expand=1)
    # 创建一个装按钮的容器
```

```
    panel = tkinter.Frame(top)
    # 创建按钮对象 指定添加到哪个容器中 通过command参数绑定事件回调函数
    button1 = tkinter.Button(panel, text='修改', command=change_label_text)
    button1.pack(side='left')
    button2 = tkinter.Button(panel, text='退出', command=confirm_to_quit)
    button2.pack(side='right')
    panel.pack(side='bottom')
    # 开启主事件循环
    tkinter.mainloop()


if __name__ == '__main__':
    main()
```

需要说明的是，GUI应用通常是事件驱动式的，之所以要进入主事件循环就是要监听鼠标、键盘等各种事件的发生并执行对应的代码对事件进行处理，因为事件会持续的发生，所以需要这样的一个循环一直运行着等待下一个事件的发生。另一方面，Tk为控件的摆放提供了三种布局管理器，通过布局管理器可以对控件进行定位，这三种布局管理器分别是：Placer（开发者提供控件的大小和摆放位置）、Packer（自动将控件填充到合适的位置）和Grid（基于网格坐标来摆放控件），此处不进行赘述。

使用Pygame进行游戏开发

Pygame是一个开源的Python模块，专门用于多媒体应用（如电子游戏）的开发，其中包含对图像、声音、视频、事件、碰撞等的支持。Pygame建立在SDL的基础上，SDL是一套跨平台的多媒体开发库，用C语言实现，被广泛的应用于游戏、模拟器、播放器等的开发。而Pygame让游戏开发者不再被底层语言束缚，可以更多的关注游戏的功能和逻辑。

下面我们来完成一个简单的小游戏，游戏的名字叫“大球吃小球”，当然完成这个游戏并不是重点，学会使用Pygame也不是重点，最重要的我们要在这个过程中体会如何使用前面讲解的面向对象程序设计，学会用这种编程思想去解决现实中的问题。

制作游戏窗口

```
import pygame


def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
```

```
if __name__ == '__main__':
    main()
```

在窗口中绘图

可以通过pygame中draw模块的函数在窗口上绘图，可以绘制的图形包括：线条、矩形、多边形、圆、椭圆、圆弧等。需要说明的是，屏幕坐标系是将屏幕左上角设置为坐标原点`(0, 0)`，向右是x轴的正向，向下是y轴的正向，在表示位置或者设置尺寸的时候，我们默认的单位都是像素。所谓像素就是屏幕上的一个点，你可以用浏览图片的软件试着将一张图片放大若干倍，就可以看到这些点。pygame中表示颜色用的是色光三原色表示法，即通过一个元组或列表来指定颜色的RGB值，每个值都在0~255之间，因为是每种原色都用一个8位（bit）的值来表示，三种颜色相当于一共由24位构成，这也就是常说的“24位颜色表示法”。

```
import pygame


def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 设置窗口的背景色(颜色是由红绿蓝三原色构成的元组)
    screen.fill((242, 242, 242))
    # 绘制一个圆(参数分别是: 屏幕, 颜色, 圆心位置, 半径, 0表示填充圆)
    pygame.draw.circle(screen, (255, 0, 0,), (100, 100), 30, 0)
    # 刷新当前窗口(渲染窗口将绘制的图像呈现出来)
    pygame.display.flip()
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False


if __name__ == '__main__':
    main()
```

####加载图像

如果需要直接加载图像到窗口上，可以使用pygame中image模块的函数来加载图像，再通过之前获得的窗口对象的`blit`方法渲染图像，代码如下所示。

```
import pygame


def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
```

```
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 设置窗口的背景色(颜色是由红绿蓝三原色构成的元组)
    screen.fill((255, 255, 255))
    # 通过指定的文件名加载图像
    ball_image = pygame.image.load('./res/ball.png')
    # 在窗口上渲染图像
    screen.blit(ball_image, (50, 50))
    # 刷新当前窗口(渲染窗口将绘制的图像呈现出来)
    pygame.display.flip()
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False


if __name__ == '__main__':
    main()
```

####实现动画效果

说到动画这个词大家都不会陌生，事实上要实现动画效果，本身的原理也非常简单，就是将不连续的图片连续的播放，只要每秒钟达到了一定的帧数，那么就可以做出比较流畅的动画效果。如果要让上面代码中的小球动起来，可以将小球的位置用变量来表示，并在循环中修改小球的位置再刷新整个窗口即可。

```
import pygame


def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 定义变量来表示小球在屏幕上的位置
    x, y = 50, 50
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
        screen.fill((255, 255, 255))
        pygame.draw.circle(screen, (255, 0, 0,), (x, y), 30, 0)
        pygame.display.flip()
        # 每隔50毫秒就改变小球的位置再刷新窗口
        pygame.time.delay(50)
```

```
        x, y = x + 5, y + 5


if __name__ == '__main__':
    main()
```

碰撞检测

通常一个游戏中会有很多对象出现，而这些对象之间的“碰撞”在所难免，比如炮弹击中了飞机、箱子撞到了地面等。碰撞检测在绝大多数的游戏中都是一个必须得处理的至关重要的问题，pygame的sprite（动画精灵）模块就提供了对碰撞检测的支持，这里我们暂时不介绍sprite模块提供的功能，因为要检测两个小球有没有碰撞其实非常简单，只需要检查球心的距离有没有小于两个球的半径之和。为了制造出更多的小球，我们可以通过对鼠标事件的处理，在点击鼠标的位置创建颜色、大小和移动速度都随机的小球，当然要做到这一点，我们可以把之前学习到的面向对象的知识应用起来。

```
from enum import Enum, unique
from math import sqrt
from random import randint

import pygame


@unique
class Color(Enum):
    """颜色"""

    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)
    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    GRAY = (242, 242, 242)

    @staticmethod
    def random_color():
        """获得随机颜色"""
        r = randint(0, 255)
        g = randint(0, 255)
        b = randint(0, 255)
        return (r, g, b)


class Ball(object):
    """球"""

    def __init__(self, x, y, radius, sx, sy, color=Color.RED):
        """初始化方法"""
        self.x = x
        self.y = y
        self.radius = radius
        self.sx = sx
        self.sy = sy
```

```
        self.color = color
        self.alive = True

    def move(self, screen):
        """移动"""
        self.x += self.sx
        self.y += self.sy
        if self.x - self.radius <= 0 or \
                self.x + self.radius >= screen.get_width():
            self.sx = -self.sx
        if self.y - self.radius <= 0 or \
                self.y + self.radius >= screen.get_height():
            self.sy = -self.sy

    def eat(self, other):
        """吃其他球"""
        if self.alive and other.alive and self != other:
            dx, dy = self.x - other.x, self.y - other.y
            distance = sqrt(dx ** 2 + dy ** 2)
            if distance < self.radius + other.radius \
                    and self.radius > other.radius:
                other.alive = False
                self.radius = self.radius + int(other.radius * 0.146)

    def draw(self, screen):
        """在窗口上绘制球"""
        pygame.draw.circle(screen, self.color,
                           (self.x, self.y), self.radius, 0)
```

事件处理

可以在事件循环中对鼠标事件进行处理，通过事件对象的`type`属性可以判定事件类型，再通过`pos`属性就可以获得鼠标点击的位置。如果要处理键盘事件也是在这个地方，做法与处理鼠标事件类似。

```
def main():
    # 定义用来装所有球的容器
    balls = []
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            # 处理鼠标事件的代码
            if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
                # 获得点击鼠标的位置
```

```
                x, y = event.pos
                radius = randint(10, 100)
                sx, sy = randint(-10, 10), randint(-10, 10)
                color = Color.random_color()
                # 在点击鼠标的位置创建一个球(大小、速度和颜色随机)
                ball = Ball(x, y, radius, sx, sy, color)
                # 将球添加到列表容器中
                balls.append(ball)
        screen.fill((255, 255, 255))
        # 取出容器中的球 如果没被吃掉就绘制 被吃掉了就移除
        for ball in balls:
            if ball.alive:
                ball.draw(screen)
            else:
                balls.remove(ball)
        pygame.display.flip()
        # 每隔50毫秒就改变球的位置再刷新窗口
        pygame.time.delay(50)
        for ball in balls:
            ball.move(screen)
            # 检查球有没有吃到其他的球
            for other in balls:
                ball.eat(other)


if __name__ == '__main__':
    main()
```

上面的两段代码合在一起，我们就完成了“大球吃小球”的游戏（如下图所示），准确的说它算不上一个游戏，但是做一个小游戏的基本知识我们已经通过这个例子告诉大家了，有了这些知识已经可以开始你的小游戏开发之旅了。其实上面的代码中还有很多值得改进的地方，比如刷新窗口以及让球移动起来的代码并不应该放在事件循环中，等学习了多线程的知识后，用一个后台线程来处理这些事可能是更好的选择。如果希望获得更好的用户体验，我们还可以在游戏中加入背景音乐以及在球与球发生碰撞时播放音效，利用pygame的mixer和music模块，我们可以很容易的做到这一点，大家可以自行了解这方面的知识。事实上，想了解更多的关于pygame的知识，最好的教程是pygame的官方网站，如果英语没毛病就可以赶紧去看看啦。 如果想开发3D游戏，pygame就显得力不从心了，对3D游戏开发如果有兴趣的读者不妨看看Panda3D。