实验楼《用Python做2048游戏》

黑板客

实验1-认识wxpython

一、实验说明

1. 环境登录

无需密码自动登录，系统用户名shiyanlou，密码shiyanlou

2. 环境介绍

本实验环境采用带桌面的Ubuntu Linux环境，实验中会用到桌面上的程序：

1. LX终端（LXTerminal）：Linux命令行终端，打开后会进入Bash环境，可以使用Linux命令
2. Firefox：浏览器
3. sublime/GVim：好用的编辑器
4. git，用于获得参考代码

3. 环境使用

使用GVim编辑器输入实验所需的代码及文件，使用LX终端（LXTerminal）运行所需命令进行操作。

二、课程介绍

课程目的：

```
熟悉实验环境
了解wxpython
能运行出一个窗口
理解ClientDC和PaintDC的区别
```

课程环境已装好wxpython，配套代码仓库

```
http://git.shiyanlou.com/heibanke/shiyanlou_cs427
```

获得参考代码敲命令 git clone 上面的代码仓库

1. wxpython

wxpython是基于Python的GUI库，优点如下：

1. 跨平台，32-bit Microsoft Windows，大多数Unix/Linux，Mac
2. 开源免费
3. 简单易用

官方文档链接

不错的英文教程适合系统的学习各种控件的使用。

<>还不错的中英文版书籍，网上很容易搜到电子版

2. 运行一个窗口

直接实例化

```
import wx

# 每个wxPython的程序必须有一个wx.App对象.
app = wx.App()

# 实例化一个frame
"""
None: 当前窗口的父窗口parent，如果当前窗口是最顶层的话，则parent=None,如果不是顶层窗口，则它的值为所属frame的名字
-1: id值, -1的话程序会自动产生一个id
pos: 位置
size: 宽，高大小
还有风格参数style，不填默认是这样几个的组合
wx.MAXIMIZE_BOX| wx.MINIMIZE_BOX| wx.RESIZE_BORDER|wx.SYSTEM_MENU| wx.CAPTION| wx.CLOSE_BOX
你可以去掉几个看看效果，比如
style = wx.SYSTEM_MENU| wx.CAPTION| wx.CLOSE_BOX
"""
frame = wx.Frame(None, -1, title='wx_00_base.py', pos=(300,400), size=(200,150))


# 居中处理
#frame.Centre()

# 显示frame
frame.Show()

# 进入循环，等待窗口响应
app.MainLoop()
```

3. 定义Frame子类的方式

```
#coding=utf-8

import wx
class Example(wx.Frame):
def __init__(self, title):
    super(Example, self).__init__(None, title=title,
        size=(600, 400))
    self.Centre()
    self.Show()

if __name__=="__main__":
    app = wx.App()
    Example('Shapes')
    app.MainLoop()
```

上面两种方式是一致的。

4. 接下来画一条线

```
# -*- coding: utf-8 -*-
import wx

class Example(wx.Frame):
    def __init__(self, title):
        super(Example, self).__init__(None,title=title,
            size=(250, 150))

        self.Centre()
        self.Show()

        dc = wx.ClientDC(self)
        # 画一条线，参数为起始点的x,y，终点的x,y
        dc.DrawLine(50, 60, 190, 60)

if __name__ == '__main__':
    app = wx.App()
    Example('Line')
    app.MainLoop()
```

试着把窗口resize到很小，然后再放大，你会发现什么。

5. 使用PaintDC

```
# -*- coding: utf-8 -*-
import wx

class Example(wx.Frame):
    def __init__(self, title):
        super(Example, self).__init__(None, title=title,
            size=(250, 150))

        # 绑定渲染窗口的动作到OnPaint
        # 这样当resize窗口，会重新调用该函数
        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show()

    # 画一条线，参数为起始点的x,y，终点的x,y
    def OnPaint(self, e):
        dc = wx.PaintDC(self)
        dc.DrawLine(50, 60, 190, 60)

if __name__ == '__main__':
    app = wx.App()
    Example('Line')
    app.MainLoop()
```

运行后试着resize窗口，线就在那里。

最后我们把画一根线替换成画多根线，就把DrawLine换成DrawLines，例如：

```
dc.DrawLines(((20, 60), (100, 60), (100, 10),(20, 10), (20,60)))
```

参数是一个一个点，注意这里参数格式是点x，y的元组列表。

6. 保存代码

如果不熟悉git可以baidu看看教程。

```
1. git clone 自己的仓库地址
2. 实验代码拷过去以后用
3. git add -A .
4. git commit -am "your comment"
5. git push origin master
```

以后再用就clone自己的代码仓库就ok

三. 总结

本节课我们掌握要点如下：

1. 了解wxpython
2. 熟悉环境特别是git
3. 能运行出wxpython的窗口
4. 了解ClientDC和PaintDC的差别
5. 画线

如果通过具体的点来画多边形，很不方便，而且如果再来计算多边形的面积，比较各个多边形的大小等，那就更不好操作了。下一节我们通过自定义类来画各种形状，从而理解类的使用。

帮助

如果对实验操作有疑问可以在实验课提出问答。

实验楼《用Python做2048游戏》

黑板客

实验2-先画个方圆

一、实验说明

1. 环境登录

无需密码自动登录，系统用户名shiyanlou，密码shiyanlou

2. 环境介绍

本实验环境采用带桌面的Ubuntu Linux环境，实验中会用到桌面上的程序：

1. LX终端（LXTerminal）: Linux命令行终端，打开后会进入Bash环境，可以使用Linux命令
2. Firefox：浏览器
3. sublime/GVim：好用的编辑器
4. git，用于获得参考代码

3. 环境使用

使用GVim编辑器输入实验所需的代码及文件，使用LX终端（LXTerminal）运行所需命令进行操作。

二、课程介绍

课程目的：

```
考虑如何设计类来画各种形状
画方形
画三角形
画圆形
```

GUI只是工具，关键是你用GUI来做什么。我们先用它来理解类的使用。这就是为什么我把wxpython放在《用Python做些事》的第5章的原因。

课程代码仓库

```
http://git.shiyanlou.com/heibanke/shiyanlou_cs427
```

1. 基类点

上节课我们知道画线要有点，形状也是由点组成的。所以我们要有基础类——点。它的属性就是它的位置x,y。

点的位置有绝对位置和相对位置，B点相对A点的位置，就是B.x-A.x，B.y-A.y。因此我们定义点的加减法来计算相对位置。

另外我们还定义了静态函数dist来计算两个点a，b的距离。

最后，我们在调用DrawLines函数时需要点位置的元组形式，因此我们定义了属性xy。

```
import math
class Point(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def __sub__(self, other):
        return Point(self.x-other.x,self.y-other.y)
    def __add__(self, other):
        return Point(self.x+other.x,self.y+other.y)

    @property
    def xy(self):
        return (self.x,self.y)

    def __str__(self):
        return "x={0},y={1}".format(self.x,self.y)
    def __repr__(self):
        return str(self.xy)

    @staticmethod
    def dist(a,b):
        return math.sqrt((a.x-b.x)**2+(a.y-b.y)**2)
```

2. 基类多边形

形状由点组成。我们用points列表来表示这些点。由于我们要画它，而且DrawLines的参数是元组，因此我们用drawPoints来返回所需要的参数格式。area用来代表形状的面积，不同形状有不同算法，因此用抽象函数实现。（这里的形状默认是凸闭合的形状）两个多边形的比较用面积来比较。不同形状可以用不同的颜色线来画，因此加了属性color。

```
from abc import ABCMeta,abstractmethod
class Polygon(object):
    __metaclass__ = ABCMeta
    def __init__(self,points_list,**kwargs):
        for point in points_list:
            assert isinstance(point,Point),"input must be Point type"
        self.points=points_list[:]
        self.points.append(points_list[0])
        self.color=kwargs.get('color','#000000')

    def drawPoints(self):
        points_xy=[]
        for point in self.points:
            points_xy.append(point.xy)
        print points_xy
        return tuple(points_xy)

    @abstractmethod
    def area(self):
        raise("not implement")

    def __lt__(self,other):
        assert isinstance(other,Polygon)
        return self.area<other.area
```

3. 子类矩形

基于基类Polygon，但初始化的时候更简单，只需要指定长，宽，和起始点即可。另外要记得实现area方法。

```
class RectAngle(Polygon):
    def __init__(self,startPoint,w,h,**kwargs):
        self._w = w
        self._h = h
        Polygon.__init__(self,
[startPoint,startPoint+Point(w,0),startPoint+Point(w,h),startPoint+Point(0,h)],*
*kwargs)

    def area(self):
        return self._w*self._h
```

4. 子类三角形

基于基类Polygon，初始化的时候指定三个点。记得判断三个点不在一条直线上。

```
class TriAngle(Polygon):
    pass
```

三角形自己实现吧，提示：

```
1．三点在一条直线上，报异常
2．计算面积可以用海伦公式
```

5. 子类圆

圆可以看作多边形，当边足够多时，就成了圆形。初始化参数可以是中心点，半径和实现的边数。

```
class Circle(Polygon):
    pass
```

提示： 1. 面积pi*r^2 2. 点的位置可以由半径的sin，cos函数获得。

6. 各实例化一个后画出来

这里只写矩形的例子。其他类似。

首先，先定义Frame类，和上节课的画图类似。初始化时多了一个参数shapes，把要画的形状作为列表传进去。如何画，在OnPaint里。调用shape.color和shape.drawPoints()即可。

```
import wx

class Example(wx.Frame):
    def __init__(self, title,shapes):
        super(Example, self).__init__(None, title=title,
            size=(600, 400))
        self.shapes = shapes

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show()

    def OnPaint(self, e):
        dc = wx.PaintDC(self)

        for shape in self.shapes:
            dc.SetPen(wx.Pen(shape.color))
            dc.DrawLines(shape.drawPoints())
```

然后就剩下画了。实例化就写到测试代码里吧。

```
if __name__ == '__main__':

    prepare_draws=[]

    start_p = Point(50,60)
    a=RectAngle(start_p,100,80,color="#ff0000")
    prepare_draws.append(a)

    # TriAngle

    # Circle

    for shape in prepare_draws:
        print shape.area()

    app = wx.App()
    Example('Shapes',prepare_draws)
    app.MainLoop()
```

好了，这样的框架，再想画任何多边形，只需要定义子类就ok了。这就是面向对象的设计思想。

7. 保存代码

如果不熟悉git可以baidu看看教程。

1. git clone 自己的仓库地址
2. 实验代码拷过去以后用
3. git add -A .
4. git commit -am "your comment"
5. git push origin master

以后再用就clone自己的代码仓库就ok

三. 总结

本节课我们掌握要点如下：

1. 了解面向对象设计
2. 如何在实际例子里设计类的接口和功能
3. 如何设计可扩展性强的程序

你是不是还想画点别的，自己试试吧。更漂亮的图形会在你的手上实现的。别忘了截个图给大家分享。

下一节我们来实现个更实用些的，一个计算器。

帮助

如果对实验操作有疑问可以在实验课提出问答。

实验楼《用Python做2048游戏》

黑板客

实验3-做个计算器

一、实验说明

1. 环境登录

无需密码自动登录，系统用户名shiyanlou，密码shiyanlou

2. 环境介绍

本实验环境采用带桌面的Ubuntu Linux环境，实验中会用到桌面上的程序：

1. LX终端（LXTerminal）: Linux命令行终端，打开后会进入Bash环境，可以使用Linux命令
2. Firefox：浏览器
3. sublime/GVim：好用的编辑器
4. git，用于获得参考代码

3. 环境使用

使用GVim编辑器输入实验所需的代码及文件，使用LX终端（LXTerminal）运行所需命令进行操作。

二、课程介绍

课程目的：

```
了解一些控件的使用以及如何布局
wx.BoxSizer
wx.GridSizer
wx.TextCtrl
wx.Button的回调函数
```

GUI只是工具，关键是你用GUI来做什么。这一节我们用它来做个计算器。

课程代码仓库

```
http://git.shiyanlou.com/heibanke/shiyanlou_cs427
```

1. Frame类

```
# -*- coding: utf-8 -*-
import wx
from math import*
class CalcFrame(wx.Frame):

    def __init__(self, title):
        super(CalcFrame, self).__init__(None, title=title,
            size=(300, 250))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):
        pass

if __name__ == '__main__':

    app = wx.App()
    CalcFrame(title='Calculator')
    app.MainLoop()
```

这些之前的课程都已经熟悉了。所以关键在于InitUI方法。

2. InitUI

```
def InitUI(self):
    vbox = wx.BoxSizer(wx.VERTICAL)
    self.textprint = wx.TextCtrl(self, style=wx.TE_RIGHT)
    vbox.Add(self.textprint, flag=wx.EXPAND|wx.TOP|wx.BOTTOM, border=4)
```

首先定义BoxSizer，这个东西可以允许我们以行或列放置控件。我们先放个TextCtrl文本框，再放个GridSizer用来放置按钮。

```
gridBox = wx.GridSizer(5, 4, 5, 5)

labels=['AC','DEL','pi','CLOSE','7','8','9','/','4','5','6',
        '*','1','2','3','-','0','.','=','+']
for label in labels:
    gridBox.Add(wx.Button(self, label=label), 1, wx.EXPAND)

vbox.Add(gridBox, proportion=1, flag=wx.EXPAND)
self.SetSizer(vbox)
```

gridsizer允许我们以二维布局控件。四个参数分别是

```
rows, 行数
cols, 列数
vgap, 格子之间垂直间隔
hgap, 格子之间水平间隔
```

因为定义了5行4列，因此依次放置20个按钮。

加完这些控件就可以显示一下布局是否正确。运行后，你可以看到一个计算器的界面。

3. TextCtrl

现在TextCtrl可以按键盘输入数字，我们只希望计算器上的数字被输入，不希望键盘输入。可以把TextCtrl初始化参数改为：

```
self.textprint = wx.TextCtrl(self,-1,'',style=wx.TE_RIGHT|wx.TE_READONLY)
```

然后我们定义self.equation=""，用来存储textprint的内容。通过self.textprint.SetValue(self.equation)就可以把equation的内容显示在textprint上。

4. Button

计算器的重点在于Button的回调函数。点击不同按钮我们希望根据按钮的label选择不同的回调函数进行绑定。因此我们可以这样实现放置按钮到gridbox：

```
for label in labels:
    buttonIterm = wx.Button(self,label=label)
    self.createHandler(buttonIterm,label)
    gridBox.Add(buttonItem, 1, wx.EXPAND)
```

然后实现createHandler方法

```
#创建按钮处理方法
def createHandler(self,button,labels):
    item = "DEL AC = CLOSE"
    if labels not in item:
        self.Bind(wx.EVT_BUTTON,self.OnAppend,button)
    elif labels == 'DEL':
        self.Bind(wx.EVT_BUTTON,self.OnDel,button)
    elif labels == 'AC':
        self.Bind(wx.EVT_BUTTON,self.OnAc,button)
    elif labels == '=':
        self.Bind(wx.EVT_BUTTON,self.OnTarget,button)
    elif labels == 'CLOSE':
        self.Bind(wx.EVT_BUTTON,self.OnExit,button)
```

根据label的不同，我们把按钮分别绑定到5个不同的回调函数上。

接下来最后一步就是实现回调函数。

5. 回调函数

```
#添加运算符与数字
def OnAppend(self,event):
    eventbutton = event.GetEventObject()
    label = eventbutton.GetLabel()
    self.equation += label
    self.textprint.SetValue(self.equation)
def OnDel(self,event):
    self.equation = self.equation[:-1]
    self.textprint.SetValue(self.equation)
def OnAc(self,event):
    self.textprint.Clear()
    self.equation=""
def OnTarget(self,event):
    string = self.equation
    try:
        target = eval(string)
        self.equation = str(target)
        self.textprint.SetValue(self.equation)
    except SyntaxError:
        dlg = wx.MessageDialog(self,u'格式错误，请输入正确的等式!',
                               u'请注意', wx.OK|wx.ICON_INFORMATION)
        dlg.ShowModal()
        dlg.Destroy()
def OnExit(self,event):
    self.Close()
```

运行看看，计算器可以正常工作不。

三. 总结

本节课我们掌握要点如下:

1. 了解一些控件的使用以及如何布局
2. wx.BoxSizer
3. wx.GridSizer
4. wx.TextCtrl
5. wx.Button的回调函数

你是不是觉得这个计算器功能不够强大？那么你自己实现sqrt，log，平方，cos，sin，取模等等。你会发现这种结构增加功能十分方便，而且增加新的功能基本不需要更改原有的代码。

实现完别忘了保存代码，并且截个图给大家分享哦。

下一节我们来实现个更有趣的，2048游戏。

帮助

如果对实验操作有疑问可以在实验课提出问答。

实验楼《用Python做2048游戏》

黑板客

实验4-做2048游戏

一、实验说明

1. 环境登录

无需密码自动登录，系统用户名shiyanlou，密码shiyanlou

2. 环境介绍

本实验环境采用带桌面的Ubuntu Linux环境，实验中会用到桌面上的程序：

1. LX终端（LXTerminal）：Linux命令行终端，打开后会进入Bash环境，可以使用Linux命令
2. Firefox：浏览器
3. sublime/GVim：好用的编辑器
4. git，用于获得参考代码

3. 环境使用

使用GVim编辑器输入实验所需的代码及文件，使用LX终端（LXTerminal）运行所需命令进行操作。

二、课程介绍

课程目的：

> 进一步了解一些控件的使用以及如何布局
> 界面和逻辑如何分开
> 核心逻辑在小游戏中占多大比重

GUI只是工具，关键是你用GUI来做什么。前面我们学会如何画图形，如何绑定动作。这一节我们用它们来做个有趣的2048游戏。

2048原版游戏地址

wxpython版游戏代码可以clone自我的代码仓库。

http://git.shiyanlou.com/heibanke/shiyanlou_cs427

1. 先定义Frame类

```
class Frame(wx.Frame):

    def __init__(self,title):
        super(Frame,self).__init__(None,-1,title,
                style=wx.DEFAULT_FRAME_STYLE^wx.MAXIMIZE_BOX^wx.RESIZE_BORDER)

        self.colors = {0:(204,192,179),2:(238, 228, 218),4:(237, 224, 200),
                8:(242, 177, 121),16:(245, 149, 99),32:(246, 124, 95),
                64:(246, 94, 59),128:(237, 207, 114),256:(237, 207, 114),
                512:(237, 207, 114),1024:(237, 207, 114),2048:(237, 207, 114),
                4096:(237, 207, 114),8192:(237, 207, 114),16384:(237, 207, 114),
                32768:(237, 207, 114),65536:(237, 207, 114),131072:(237, 207,
114),
                262144:(237, 207, 114),524288:(237, 207, 114),1048576:(237, 207,
114),
                2097152:(237, 207, 114),4194304:(237, 207, 114),
                8388608:(237, 207, 114),16777216:(237, 207, 114),
                33554432:(237, 207, 114),67108864:(237, 207, 114),
                134217728:(237, 207, 114),268435456:(237, 207, 114),
                536870912:(237, 207, 114),1073741824:(237, 207, 114),
                2147483648:(237, 207, 114),4294967296:(237, 207, 114),
                8589934592:(237, 207, 114),17179869184:(237, 207, 114),
                34359738368:(237, 207, 114),68719476736:(237, 207, 114),
                137438953472:(237, 207, 114),274877906944:(237, 207, 114),
                549755813888:(237, 207, 114),1099511627776:(237, 207, 114),
                2199023255552:(237, 207, 114),4398046511104:(237, 207, 114),
                8796093022208:(237, 207, 114),17592186044416:(237, 207, 114),
                35184372088832:(237, 207, 114),70368744177664:(237, 207, 114),
                140737488355328:(237, 207, 114),281474976710656:(237, 207, 114),
                562949953421312:(237, 207, 114),1125899906842624:(237, 207,
114),
                2251799813685248:(237, 207, 114),4503599627370496:(237, 207,
114),
                9007199254740992:(237, 207, 114),18014398509481984:(237, 207,
114),
                36028797018963968:(237, 207, 114),72057594037927936:(237, 207,
```

```
114)}

        self.setIcon()
        self.initGame()
        self.initBuffer()
        panel = wx.Panel(self)
        panel.Bind(wx.EVT_KEY_DOWN,self.onKeyDown)
        panel.SetFocus()

        self.Bind(wx.EVT_SIZE,self.onSize) # use wx.BufferedPaintDC
        self.Bind(wx.EVT_PAINT,self.onPaint)
        self.Bind(wx.EVT_CLOSE,self.onClose)
        self.SetClientSize((505,720))
        self.Center()
        self.Show()

    def onClose(self,event):
        self.saveScore()
        self.Destroy()

    def setIcon(self):
        icon = wx.Icon("icon.ico",wx.BITMAP_TYPE_ICO)
        self.SetIcon(icon)

    def loadScore(self):
        if os.path.exists("bestscore.ini"):
            ff = open("bestscore.ini")
            self.bstScore = ff.read()
            ff.close()

    def saveScore(self):
        ff = open("bestscore.ini","w")
        ff.write(str(self.bstScore))
        ff.close()

if __name__ == "__main__":
    app = wx.App()
    Frame(u"2048 v1.0.1 by heibanke")
    app.MainLoop()
```

colors是定义2，4，8，不同分数的块颜色不同。三个数值分别是RedGreenBlue。saveScore和loadScore是记录你的最高分，记录到文本文件里。

大部分之前的课程都已经熟悉了。关键在于initGame，initBuffer和绑定的几个方法。我们慢慢看。

2. initGame

```
def initGame(self):
    self.bgFont = wx.Font(50,wx.SWISS,wx.NORMAL,wx.BOLD,face=u"Roboto")
    self.scFont = wx.Font(36,wx.SWISS,wx.NORMAL,wx.BOLD,face=u"Roboto")
    self.smFont = wx.Font(12,wx.SWISS,wx.NORMAL,wx.NORMAL,face=u"Roboto")
    self.curScore = 0
    self.bstScore = 0
    self.loadScore()
    self.data = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
    count = 0
    while count<2:
        row = random.randint(0,len(self.data)-1)
        col = random.randint(0,len(self.data[0])-1)
        if self.data[row][col]!=0:
            continue
        self.data[row][col] = 2 if random.randint(0,1) else 4
        count += 1
```

这里初始化各种字体，curScore为当前分数。bstScore为最高分数。data为4×4的棋盘。然后while循环是在4×4棋盘上随机位置产生2个块2或4。

3. initBuffer

```
def initBuffer(self):
    w,h = self.GetClientSize()
    self.buffer = wx.EmptyBitmap(w,h)

def onSize(self,event):
    self.initBuffer()
    self.drawAll()
```

initBuffer定义一块buffer，用来使用BufferedDC来画图。 onSize是在窗口改变时重新initBuffer，并调用drawAll来画界面。

所以界面的关键在drawAll，用户的动作响应在onKeyDown。这两个是该程序的重点。

4. drawAll

```
def drawAll(self):
    dc = wx.BufferedDC(wx.ClientDC(self),self.buffer)
    self.drawBg(dc)
    self.drawLogo(dc)
    self.drawLabel(dc)
    self.drawScore(dc)
    self.drawTiles(dc)

def drawChange(self,score):
    dc = wx.BufferedDC(wx.ClientDC(self),self.buffer)
    if score:
        self.curScore += score
        if self.curScore > self.bstScore:
            self.bstScore = self.curScore
        self.drawScore(dc)
    self.drawTiles(dc)
```

drawAll就是把界面逐次画出来，包括背景，logo，Label，Score和棋盘里的块(Tiles)。
drawChange是当用户有动作后，把改变的地方重新画一遍，如score和Tiles。

这些大家自己看吧。主要是wxpython控件的用法。之前没讲到的，请查阅wxpython资料。

5. onKeyDown

```
def onKeyDown(self,event):
    keyCode = event.GetKeyCode()

    if keyCode==wx.WXK_UP:
        self.doMove(*self.slideUpDown(True))
    elif keyCode==wx.WXK_DOWN:
        self.doMove(*self.slideUpDown(False))
    elif keyCode==wx.WXK_LEFT:
        self.doMove(*self.slideLeftRight(True))
    elif keyCode==wx.WXK_RIGHT:
        self.doMove(*self.slideLeftRight(False))
```

这里doMove的参数是slideUpDown或者slideLeftRight方法返回的元组。 *代表元组，**代表字典。

当用户按上或下方向时，slideUpDown对棋盘的列进行处理，当用户按左或右方向时，slideLeftRight对棋盘的行进行处理。两者类似，这里仅对slideUpDown进行说明。

```
def slideUpDown(self,up):
    score = 0
    numCols = len(self.data[0])
    numRows = len(self.data)
    oldData = copy.deepcopy(self.data)

    for col in range(numCols):
        cvl = [self.data[row][col] for row in range(numRows) if self.data[row]
[col]!=0]

        if len(cvl)>=2:
            score += self.update(cvl,up)
        for i in range(numRows-len(cvl)):
            if up: cvl.append(0)
            else: cvl.insert(0,0)
        for row in range(numRows): self.data[row][col] = cvl[row]
    return oldData!=self.data,score

def update(self,vlist,direct):
    score = 0
    if direct: #up or left
        i = 1
        while i<len(vlist):
            if vlist[i-1]==vlist[i]:
                del vlist[i]
                vlist[i-1] *= 2
                score += vlist[i-1]
                i += 1
            i += 1
    else:
        i = len(vlist)-1
        while i>0:
            if vlist[i-1]==vlist[i]:
                del vlist[i]
                vlist[i-1] *= 2
                score += vlist[i-1]
                i -= 1
            i -= 1
    return score
```

得到行数列数，备份数据到oldData后，对每列数据进行处理。 棋盘的列从上往下序号是

```
0
1 --> 2
2
3 --> 2
```

如果在1，3有两个Tile值为2的话，点击上，则

```
cvl = [self.data[row][col] for row in range(numRows) if self.data[row][col]!=0]
```

将两个2提取出来到cvl，update函数后，将两个2合并为1个4，并return这一步得到的分数。

```
for i in range(numRows-len(cvl)):
    if up: cvl.append(0)
    else: cvl.insert(0,0)
for row in range(numRows):
    self.data[row][col] = cvl[row]
```

然后根据按的是up还是down决定是在cvl后面加0，还是在前面加0，补齐为4个。然后逐一替换到data里。

```
return oldData!=self.data,score
```

最后return一个元组，(数据是否变化的标志move，和这一步得到的分数score)。这个元组就是方法doMove的参数。

这是最简单的情况，再考虑三个Tile相同，三个Tile有2个相同等各种情况是否适用，就差不多了。然后再扩展到下方向和左右方向上。

6. doMove

```
def doMove(self,move,score):
    if move:
        self.putTile()
        self.drawChange(score)
        if self.isGameOver():
            if wx.MessageBox(u"游戏结束，是否重新开始？",u"哈哈",
                    wx.YES_NO|wx.ICON_INFORMATION)==wx.YES:
                bstScore = self.bstScore
                self.initGame()
                self.bstScore = bstScore
                self.drawAll()
```

如果移动了，则putTile用来在棋盘上还是0的位置上随机挑一个，生成一个Tile值为2或4。然后把这些变化画出来就调用drawChange(score)。

最后判断是否GameOver，也就是4个方向分别试一下能不能move，有一个方向能move则没有GameOver。

```
def isGameOver(self):
    copyData = copy.deepcopy(self.data)

    flag = False
    if not self.slideUpDown(True)[0] and not self.slideUpDown(False)[0] and \
            not self.slideLeftRight(True)[0] and not self.slideLeftRight(False)
[0]:
        flag = True
    if not flag: self.data = copyData
    return flag
```

OK了，是不是很简单。其实最容易错的地方就是update和slide。多想想多试试，加些print也可以，总之搞懂它们你就能开发一个属于自己的2048.

三. 总结

本节课我们开发了一个2048游戏，你如果觉得还不过瘾。可以尝试给它增加一些功能。比如：

1. 悔棋。有时候一步错步步错，能否有个按键，点一下能反转时间。呵呵。
2. 扩大。棋盘能不能6×6？
3. 智能。自己打不到8192？没关系，开发个人工智能，提示你往哪里走，突破极限别忘了把代码给我发一份。

这些就当作思考题吧。别忘了保存代码，觉得实验不错的，别忘了写篇感想刺激一下后来人。

到此我们的实验课就结束了。如果你觉得这种方式不错，欢迎到云课堂<<用Python做些事>>里看

看。那里web后端也开发了一门实验课程，**用django开发云记账项目**。希望课程能够对你有帮助，谢谢支持！

帮助

如果对实验操作有疑问可以在实验课提出问答。