Python程序设计

第八章 图形界面程序开发

皇甫伟
北京科技大学计算机与通信工程学院

综合练习3：自绘苹果画板

为什么需要自绘控件？

- 已有的控件不能满足需求，就需要自己定义外观和事件响应
- 当需要控件绘制自己时，会收到(re-)paint消息，本质上自绘控件就是接管该消息的响应函数，自行重绘。

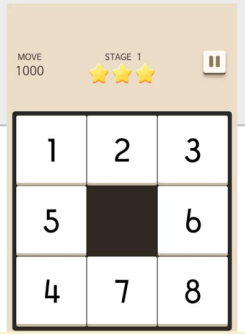
自绘方法

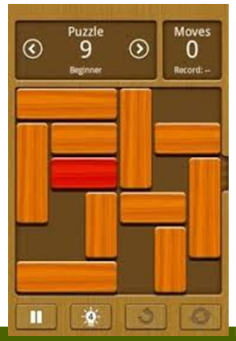
- 重写父类的paintEvent方法
- 在该方法中创建QPainter的实例
- 利用QPainter进行绘制
  - 绘制前后需要调用begin/end函数。

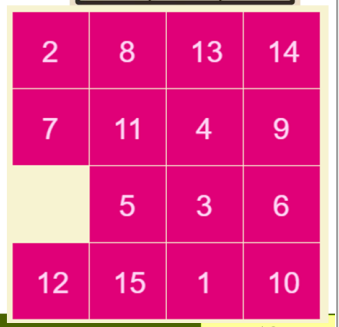
综合练习4：滑块游戏

滑块游戏

- 华容道

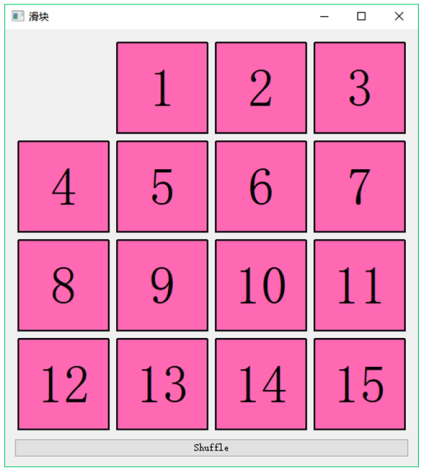






滑块游戏的设计

- 绘制15滑块和1个空白位置
- 允许打乱
- 用户可以点击移动滑块



代码示意1

```
import sys, random
from PyQt4.QtCore import *
from PyQt4.QtGui import *

GAP = 5
SIZE = 120

class BlockWindow(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.data = list(range(16))
        self.setMinimumSize(SIZE*4, SIZE*4)

    def paintEvent(self, event):
        qp = QPainter()
        qp.begin(self)
        self.drawMe(qp)
        qp.end()
```

每个滑块的大小为SIZE；在四周留出GAP的间隙。

此处用self.data存储一个所有滑块的列表，用0表示空白位置。

代码示意2

```
def drawMe(self, qp):
    pen = QPen(Qt.black, 2, Qt.SolidLine)
    brush = QBrush(QColor("hotpink"))
    font = QFont()
    font.setPixelSize(64)
    qp.setPen(pen)
    qp.setBrush(brush)
    qp.setFont(font)
    for i in range(4):
        for j in range(4):
            ix = i * 4 + j
            if self.data[ix]!=0:
                x = SIZE * j + GAP
                y = SIZE * i + GAP
                qp.drawRect(x, y, SIZE-2*GAP, SIZE-2*GAP)
                qp.drawText(x, y, SIZE-2*GAP, SIZE-2*GAP,
                            Qt.AlignCenter, f"{self.data[ix]:d}")
```

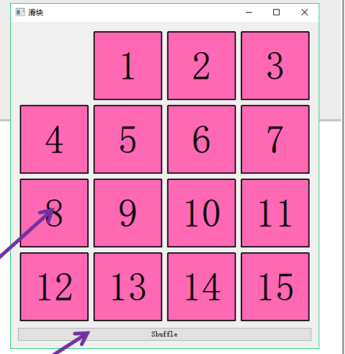
浅粉色

字体的设置，大小设为64。

*注意第i行第j列的滑块在data中的位置是i*4+j。*

绘制文字，且在矩形框中居中。

代码示意3




```
class TopWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
        self.initSignalSlot()
        self.setWindowTitle("滑块")

    def initUI(self):
        vLayout = QVBoxLayout()
        self.canvas = BlockWindow(self)
        self.shuffleButton = QPushButton('Shuffle', self)

        vLayout.addWidget(self.canvas, stretch=1)
        vLayout.addWidget(self.shuffleButton)

        self.setLayout(vLayout)
```


代码示意4：辅助函数

```
def getNum(self, i, j):
    "取得第i行j列的数字，如果出界则返回-1"
    if 0<=i<4 and 0<=j<4:
        ix = i * 4 + j
        return self.data[ix]
    else:
        return -1

def swap(self, i, j, m, n):
    "交换第i行j列和第m行n列的数字"
    ix1 = i * 4 + j
    ix2 = m * 4 + n
    tmp = self.data[ix1]
    self.data[ix1] = self.data[ix2]
    self.data[ix2] = tmp
    self.update()
```

update是继承的函数，通知系统重绘自己，也就是让系统发出Paint消息，进而在消息队列中调用PaintEvent()。

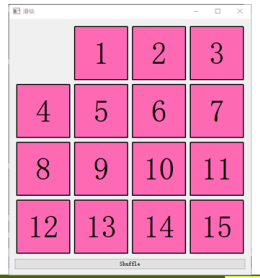
代码示意5：动起来

```
def mousePressEvent(self, evt):
    x, y = evt.x(), evt.y()
    j = int(x / SIZE)
    i = int(y / SIZE)
    if 0<=i<4 and 0<=j<4:
        if self.getNum(i-1, j)==0:
            self.swap(i, j, i-1, j)
        elif self.getNum(i+1, j)==0:
            self.swap(i, j, i+1, j)
        elif self.getNum(i, j-1)==0:
            self.swap(i, j, i, j-1)
        elif self.getNum(i, j+1)==0:
            self.swap(i, j, i, j+1)
```

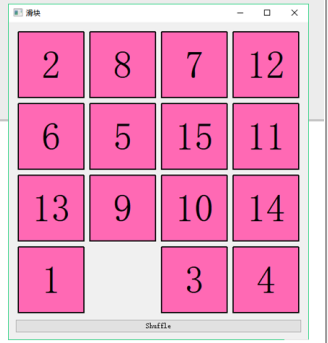
当鼠标点击（按下）时会调用此函数，类似于paintEvent()。

参数evt里面包含了很多信息，包括点击的位置，点击的按键等。

如果点击在游戏区，且旁边有空白位置，则交换。



代码示意5：打乱功能




```
class BlockWindow:
    def shuffle(self):
        random.shuffle(self.data)
        self.update()

class TopWindow:
    def initSignalSlot(self):
        self.shuffleButton.clicked.connect(self.onShuffle)

    def onShuffle(self, evt):
        self.canvas.shuffle()
```


自定义Signal

- 数据存储在窗口BlockWindow的类中
- 移动发生在窗口BlockWindow中
- 因此，主窗口TopWindow并不知道是否游戏已经成功完成？
- 需要自定义signal，表明本窗口发生了什么事件，以通知其他窗口。

代码示意6：

```
class BlockWindow(QWidget):

    finished = pyqtSignal()  在类的层面定义变量，作为信号。

    def isFinished(self):
        return self.data[0]==6      判定是否成功？此处简单的判定第一个位置是不是6。

    def swap(self, i, j, m, n):
        ix1 = i * 4 + j
        ix2 = m * 4 + n
        tmp = self.data[ix1]
        self.data[ix1] = self.data[ix2]
        self.data[ix2] = tmp
        if self.isFinished():
            self.finished.emit()      在合适的时机，激发信号。由相应的槽进行处理。
        self.update()
```


代码示意7：

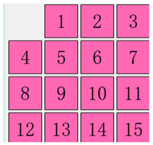
```
class TopWindow(QWidget):

    def initSignalSlot(self):
        self.shuffleButton.clicked.connect(self.onShuffle)
        self.canvas.finished.connect(self.onFinished)

    def onFinished(self):
        ok = QMessageBox.question(self,
             '消息框',
             "祝贺你，太棒了！",
             QMessageBox.Ok)
```

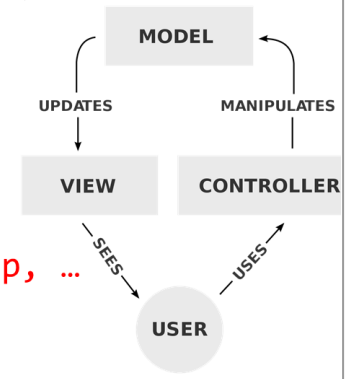
绑定自定义信号到槽。

在槽里面，弹出对话框进行鼓励。

MVC的概念

- 此例中出现了明显的MVC的使用
- M：model，数据
  - self.data
- V：view, 视图
- C：controller
  - mousepress, getnum, swap, …

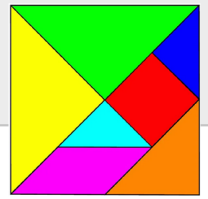




MVC

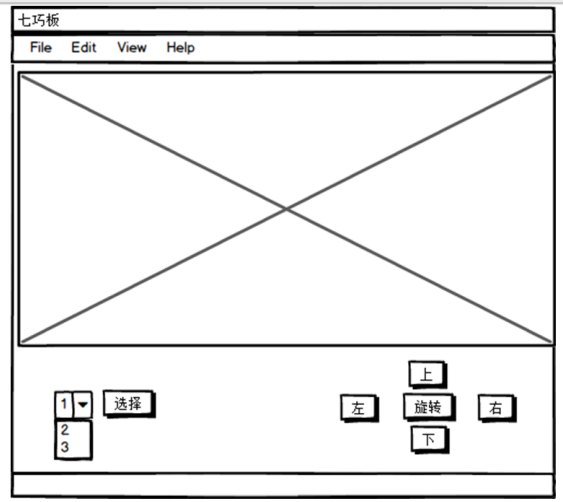
- Model（模型）是应用程序中用于处理应用程序数据逻辑的部分，通常模型对象负责在数据库中存取数据。
- View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。
- Controller（控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。
- MVC 分层有助于管理复杂的应用程序，因为您可以在一个时间内专门关注一个方面。例如，您可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。
- MVC 分层同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

七巧板



- Model：
  - 7个板块的位置、角度、颜色（数据）
- View：
  - 根据板块的位置、角度进行绘制
- Control
  - 用户选定某个板块
    - 如何选：鼠标点击某个位置（判定）？下拉框选定某个？
  - 用户对板块的操作
    - 如何操作：鼠标拖曳表示移动？用按钮（方向键）控制移动和旋转？

七巧板



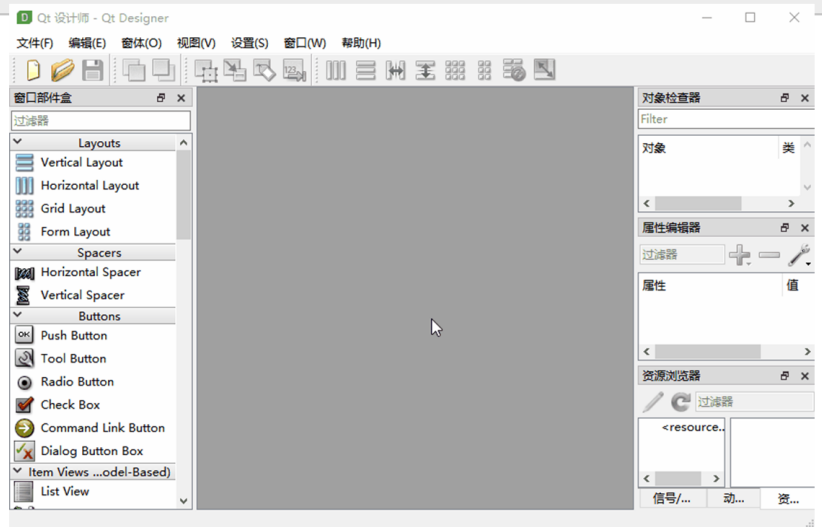
小结

- 鼠标事件的处理
- 自定义信号的方法
- MVC框架的概念

综合练习5：扩展部分

QT Designer的使用

可以实现基于拖曳的所见即所得的界面设计！
文件可以保存为UI文件，
也可以自动生成py文件。

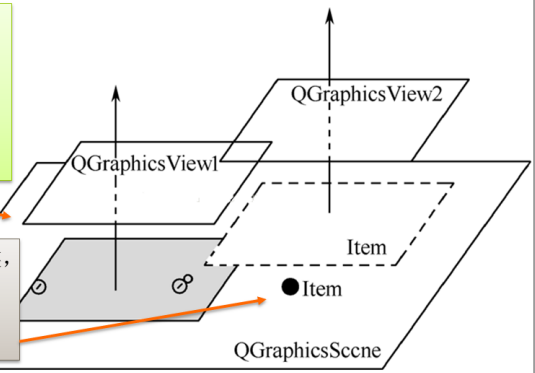


QT Graphics-View

- 实现了模型－视图结构
- 对大量图元进行管理

视图：QGraphicsView提供可视的窗口，用于显示场景中的图元，一个场景中可以有多个视图。

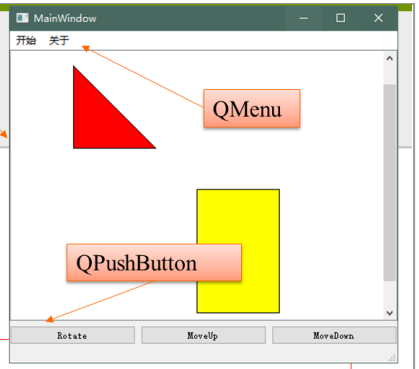
场景：QGraphicsScene是一个存储图元的容器，必须QGraphicsView视图来显示及与外界进行交互，主要提供图元的操作接口、传递事件和管理各个图元状态，提供无变换的绘制功能。



图元：QGraphicsItem是图元的基础类，派生出矩形(QGraphicsRectItem)、椭（QGraphicsEllipseItem)、文本(QGraphicsTextItem)等子类。

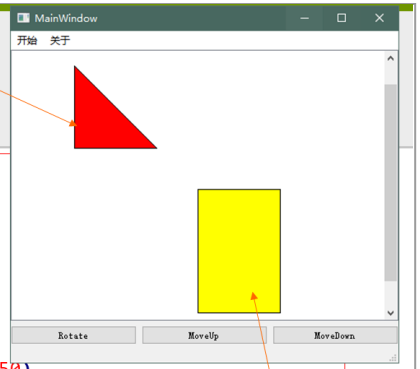
七巧板

- 基于Designer生成界面，含：
  - graphicsView
  - pushButtonRotate



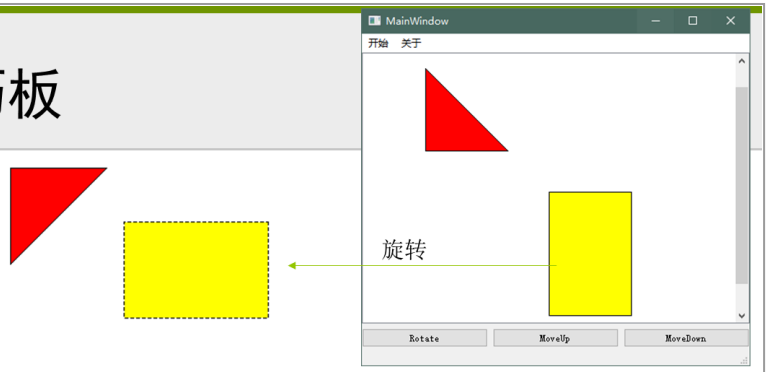

```
from PyQt5 import uic
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi("q7.ui", self)
        self.scene = QGraphicsScene(0, 0, 400, 400)
        self.graphicsView.setScene(self.scene)
        self.init_shapes()
        self.pushButtonRotate.clicked.connect(self.onRotate)
        self.pushButtonMoveUp.clicked.connect(self.onMoveUp)
        self.pushButtonMoveDown.clicked.connect(self.onMoveDown)
```


七巧板



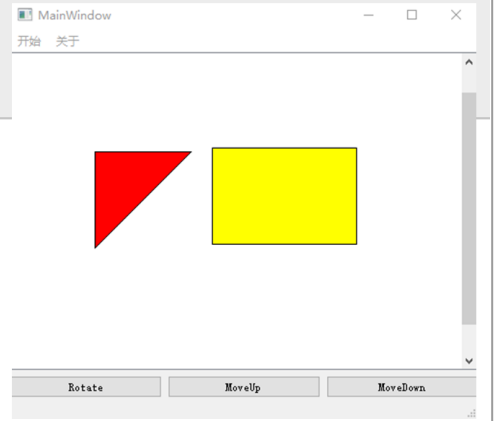

```
def init_shapes(self):
    polygon = QGraphicsPolygonItem(
        QPolygonF([QPointF(0,0),
                   QPointF(0,100),
                   QPointF(100,100)]))
    polygon.setBrush(QBrush(Qt.red))
    polygon.setPos(50, 50)

    rect = QGraphicsRectItem(0, 0, 100, 150)
    rect.setBrush(QBrush(Qt.yellow))
    rect.setPos(200, 200)

    polygon.setFlags(QGraphicsItem.ItemIsSelectable|
        QGraphicsItem.ItemIsMovable)
    # the same for rect

    self.scene.addItem(polygon)
    # the same for rect
```


七巧板




```
def onRotate(self, evt):
    items = self.scene.selectedItems()
    for item in items:
        item.setRotation(90+item.rotation())

def onMoveUp(self, evt):
    items = self.scene.selectedItems()
    for item in items:
        item.moveBy(0,-10)
```


七巧板




```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec()
```

投票 最多可选1项

你更喜欢：

A 界面非常美观，功能相对一般的软件

B 界面相对一般，功能非常强大的软件

本章小结

- GUI的历史和概念
- 以PyQT为例介绍了GUI的开发
  - 事件循环的思想
- 通常GUI程序的规模较大，本章还进一步介绍了实例和软件工程的一些思想
- MVC的运用
- 趋势：基于Web的GUI