面向对象程序设计方法课程

综合训练实验报告

软件工程 42

陈景琦 2141601025

陈　铮 2141601026

郭家梁 2141601027

欧阳鹏程 2141601030

实验指导老师：梁力

实验地点：软件学院教学实验中心机房

实验结束日期：2015-7-1

实验报告提交日期：2015-7-6

联系电话：陈 铮 18292886986

目录

正文

实验一、建立队列类模板

队列是一种经典的数据结构。其具有明显的现实意义，如排队队伍。具有"先进先出"的特点（FIFO），组织一个队列的方法可以是线性表或者链表甚至散列表均可。队列的成员应该包括：

1、数据成员 Val[]

2、队首位置 Front

3、队列容量 Max_Vol

4、队列大小 Now_Size

一个队列至少应该支持以下操作（方法）：

1、入队操作 push

2、出队操作 pop

3、取队首元素 front

由于需要可视化演示，我们需要在不改变队列的前提下，或者说保持数据成员私有的前提下，读取队列中的所有元素，因此需要有透视功能的方法。

4、读取元素 GetVal

5、读取大小 GetSize

6、读取容量 GetVol

存储结构

我们采用的存储结构是结合线性表与散列表思想的**循环队列**结构。先线性地申请一块内存空间 `Val[]`，元素个数为队列容量 `Max_Vol`。队列中的第 `n` 个元素用 `Val[(Front+n)%Max_Val]`来访问(这里的 `n` 从 `0` 开始计数),利用取模运算完成了循环处理，避免了在入队出队操作中重复申请内存，提高了效率。

队列容量更改与拷贝构造函数

在更改队列容量(`Resize`)的时候，若容量高于原容量，则先申请一块大内存，再将原数据全部拷贝过来;若容量低于原容量,则申请一块小内存,再将原队列中数据依次出队，再将其推送到新队列中。这里会使用自定义的**拷贝构造函数**的方法。

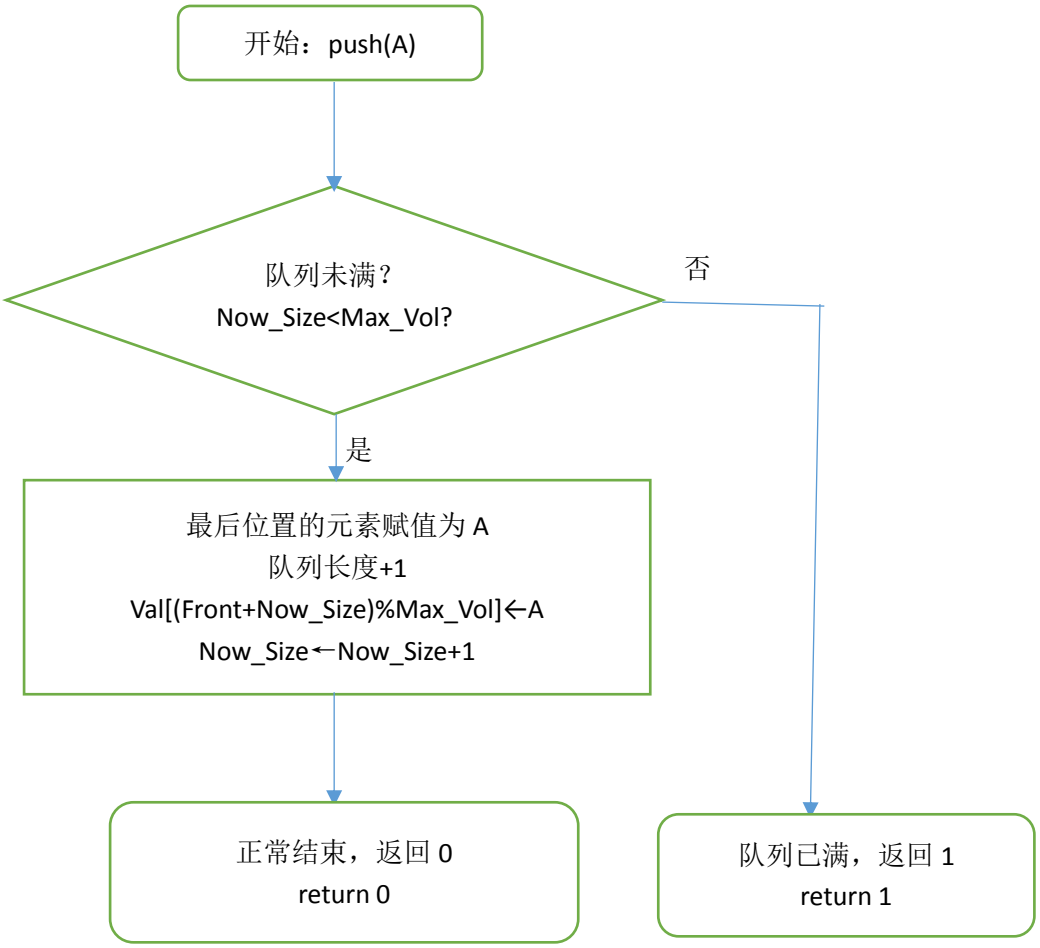
关于模板类

在队列类(`Queue`)定义的前面加上一句 `template <class T>`即可使单一数据类型的 `Queue` 变为 `Queue<T>`。当然之后的成员函数的实现的前面都要加一句 `template <class T>`

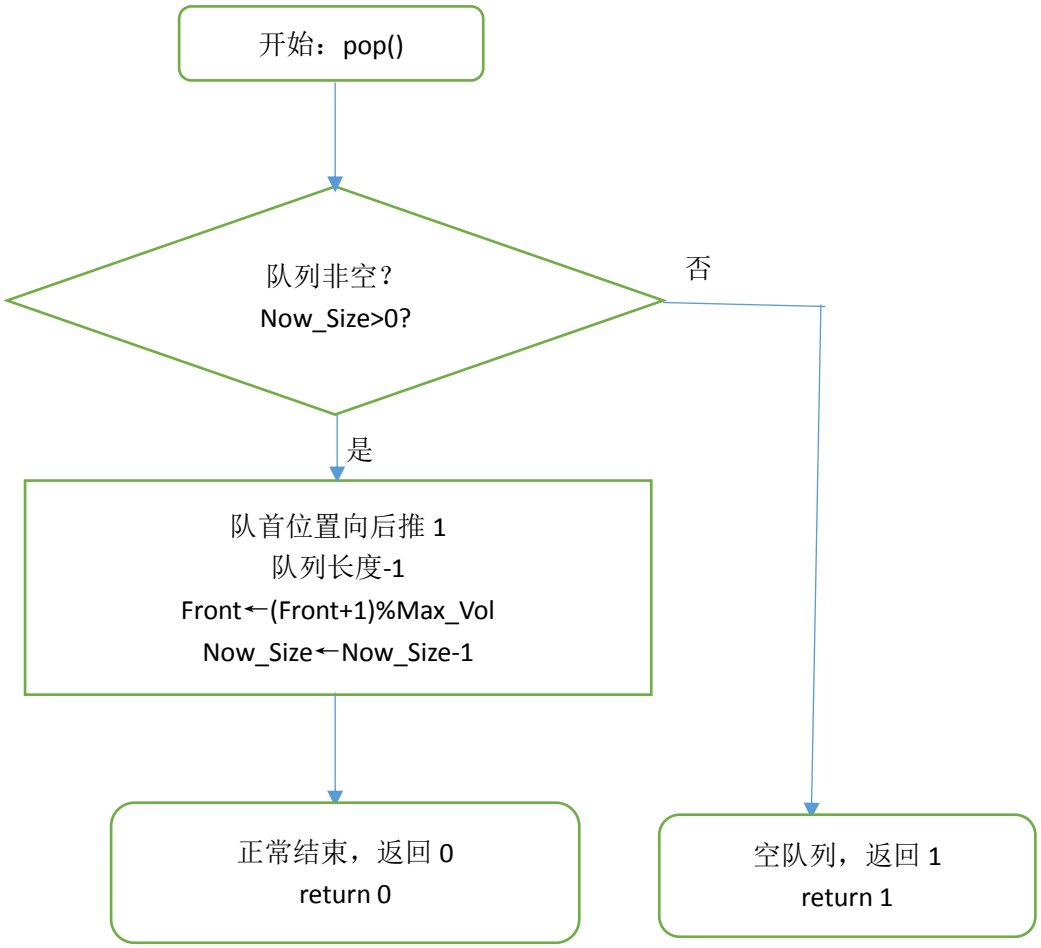
*拓展：我们可以利用序列化数据类型的方法，利用 `MFC` 的 `ListBox` 控件简单快捷地实现队列模板。

算法描述

入队 push


开始：push(A)
队列未满？
Now_Size<Max_Vol?
否
是
最后位置的元素赋值为 A
队列长度+1
Val[(Front+Now_Size)%Max_Vol]←A
Now_Size←Now_Size+1
正常结束，返回 0
return 0
队列已满，返回 1
return 1

出队 pop



*拓展： ListBox 控件上的队列实现

MFC 中 listbox 控件是为了显示一系列的文本，每个文本占一行。

基于对话框的 MFC 自带 ListBox 控件，具有以下方法：

1、在 ListBox 的尾部添加一项 String

```
int AddString(LPCTSTR Str);
```

2、获取 ListBox 行的总数

```
int GetCount();
```

3、删除 ListBox 中第 nIndex 行的文本，注意 nIndex 从 0 开始

```
int DeleteString(uint nIndex);
```

通过以上方法，以及适当地建模，我们就能实现队列模板类的可视化。

int QUEUE_VOL 是之前设定的队列容量，CListBox List1 为队列的 ListBox 载体对象。

CString QUEUE_DataString 为对数据个体建模后的 String 序列（序列化）。

注意：使用 ListBox 之前要对其属性做出一些调整，在属性中将 Sort 改为 False，若 Sort 为 True，在某种规范的序列化后，就能实现【**优先队列**】数据结构。

另外也要注意这些重要的属性：

Vertical Scrollbar：是否具有垂直滚动条

Horizontal Scrollbar：是否具有水平滚动条

MultiColumn：是否多列显示

入队代码：

```
if (List1.GetCount() < QUEUE_VOL)
   List1.AddString(QUEUE_DataString);
else
   MessageBox(CString("队列已满"));
```

出队代码：

```
if (List1.GetCount() > 0)
   List1.DeleteString(0);
else
   MessageBox(CString("队列已空"));
```

调整容量代码：

```
for (int i = QUEUE_VOL; i < List1.GetCount(); i++)
   List1.DeleteItem(i);
```

可以看到，这种实现过程非常简单与简洁，而且多样性与可重用性强。

因为一个数据结构与 CString 总是可以互相映射（转换）的。只要在类模板中实现由 CString 为参数的构造函数与将所有成员变量映射到 CString 上的函数（序列化），即可

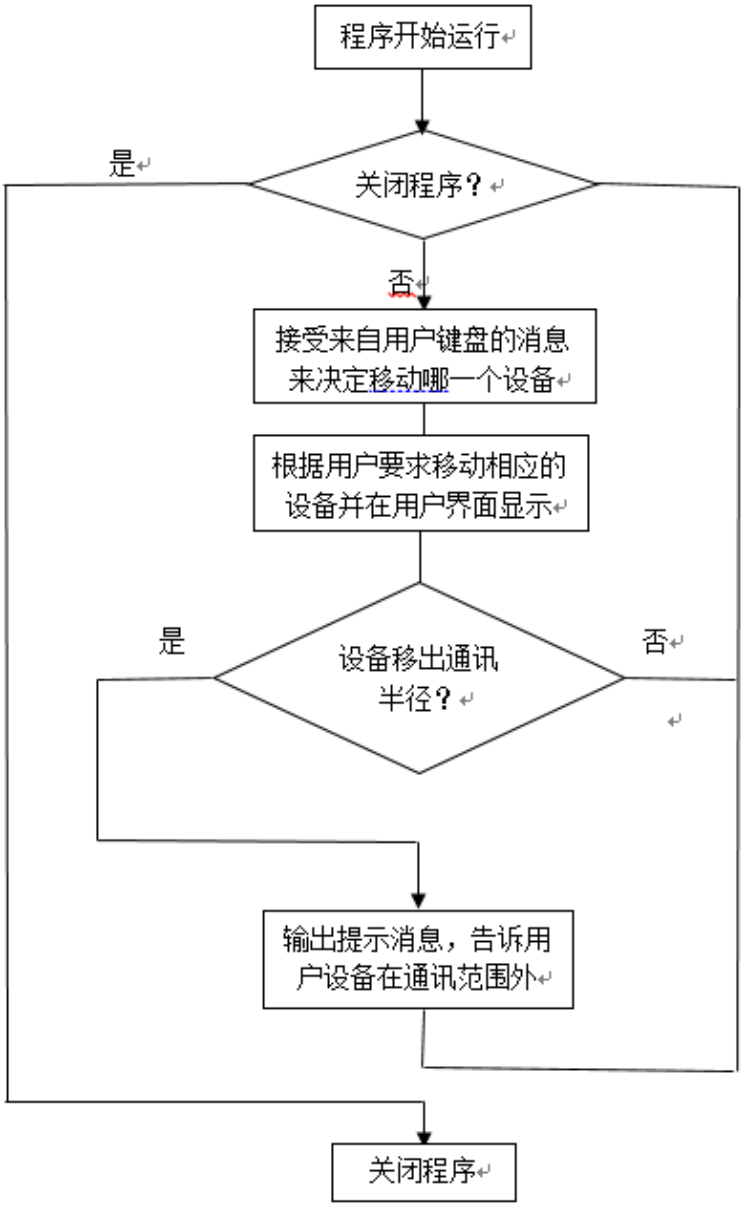
完成类模板与 `ListBox` 的对接。

实验二、简单蜂窝移动通信系统演示

一、实验任务及要求

在本实验中，我们的目标是要实现一个拥有基站和移动台的系统，通过在用户的可视化界面上对移动台的移动来不断告诉用户的设备是否还在基站的通讯范围内

二、程序的算法描述



本实验的关键在于实时在用户界面实现移动台的坐标输出以及图形的移动

界面初始化实现代码：

```
void C移动基站View::OnDraw(CDC* /*pDC*/)
{
    C移动基站Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: 在此处为本机数据添加绘制代码
    CImage imag, imag1, imag2, imag3, imag4, imag5, imag6;
    CClientDC dc(this);
    imag.Load(_T("res//base.png"));
    imag.Draw(dc, 400, 100, 500, 500);

    imag1.Load(_T("res//bijiben.png"));
    imag1.Draw(dc, lap.X, lap.Y, 80, 62);

    imag2.Load(_T("res//shouji.png"));
    imag2.Draw(dc, pho.X, pho.Y, 28, 49);

    imag3.Load(_T("res//shoubiao.png"));
    imag3.Draw(dc, wat.X, wat.Y, 28, 30);

    imag4.Load(_T("res//pingban.png"));
    imag4.Draw(dc, surf.X, surf.Y, 51, 35);

    imag5.Load(_T("res//PDA.png"));
    imag5.Draw(dc, pda.X, pda.Y, 41, 46);

    imag6.Load(_T("res//lanyaerji.png"));
    imag6.Draw(dc, ear.X, ear.Y, 40, 22);
    CClientDC cdc(this);
    CString str;
    str.Format(_T("笔记本电脑坐标为：(%d,%d)"), lap.X - 650, 350 -
lap.Y);
    cdc.TextOutW(100, 100, str);
    str.Format(_T("手机坐标为：(%d,%d)"), pho.X - 650, 350 - pho.Y);
```

```
    cdc.TextOutW(100, 120, str);
    str.Format(_T("手表坐标为：(%d,%d)"), wat.X - 650, 350 - wat.Y);
    cdc.TextOutW(100, 140, str);
    str.Format(_T("平板电脑坐标为：(%d,%d)"), surf.X - 650, 350 -
surf.Y);
    cdc.TextOutW(100, 160, str);
    str.Format(_T("PDA 坐标为：(%d,%d)"), pda.X - 650, 350 - pda.Y);
    cdc.TextOutW(100, 180, str);
   str.Format(_T("蓝牙耳机坐标为：(%d,%d)"), ear.X - 650, 350 - ear.Y);
    cdc.TextOutW(100, 200, str);
    cdc.TextOutW(100, 450, _T("使用 qwer 控制笔记本电脑的上下左右"));
    cdc.TextOutW(100, 470, _T("使用 uiop 控制平板电脑的上下左右"));
    cdc.TextOutW(100, 490, _T("使用 asdf 控制手机的上下左右"));
    cdc.TextOutW(100, 510, _T("使用 hjkl 控制 PDA 的上下左右"));
    cdc.TextOutW(100, 530, _T("使用 zxcv 控制手表的上下左右"));
    cdc.TextOutW(100, 550, _T("使用 nm,.控制蓝牙耳机的上下左右"));
}
```

因为有 6 个移动台，所以首先定义 6 个 CImag 对象，分别代表 6 个移动台，分别载入并画图，以基站的圆心为原点，所以需要进行简单的坐标变换。由于 TextOutW 函数要求的第三个参数是字符串型，则无法实时进行坐标输出，所以先定义 CString 对象并实时用 Format 函数将坐标转换为字符串，再用 TextOutW 输出。

根据用户键盘输入移动相应设备实现代码：

```
void C移动基站View::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO:  在此添加消息处理程序代码和/或调用默认值
```

```
    CView::OnChar(nChar, nRepCnt, nFlags);
    if (nChar == 'q')
    {
        --lap.Y;
        Draw();
        if (((lap.X - 650)*(lap.X - 650) + (lap.Y - 350)*(lap.Y -
350)) >= lap.Lradius*lap.Lradius)
            MessageBox(_T("你的笔记本电脑已经超出范围！"));

    }
    else if (nChar == 'w')
    {
        ++lap.Y;
        Draw();
        if (((lap.X - 650)*(lap.X - 650) + (lap.Y - 350)*(lap.Y -
350)) >= lap.Lradius*lap.Lradius)
            MessageBox(_T("你的笔记本电脑已经超出范围！"));

    }
    else if (nChar == 'e')
    {
        --lap.X;
        Draw();
        if (((lap.X - 650)*(lap.X - 650) + (lap.Y - 350)*(lap.Y -
350)) >= lap.Lradius*lap.Lradius)
            MessageBox(_T("你的笔记本电脑已经超出范围！"));

    }
    else if (nChar == 'r')
    {
        ++lap.X;
        Draw();
        if (((lap.X - 650)*(lap.X - 650) + (lap.Y - 350)*(lap.Y -
350)) >= lap.Lradius*lap.Lradius)
            MessageBox(_T("你的笔记本电脑已经超出范围！"));

    }
    else if (nChar == 'a')
    {
        --pho.Y;
```

```
        Draw();
        if (((pho.X - 650)*(pho.X - 650) + (pho.Y - 350)*(pho.Y -
350)) >= pho.Pradius*pho.Pradius)
            MessageBox(_T("你的手机已经超出范围！"));

    }
    else if (nChar == 's')
    {
        ++pho.Y;
        Draw();
        if (((pho.X - 650)*(pho.X - 650) + (pho.Y - 350)*(pho.Y -
350)) >= pho.Pradius*pho.Pradius)
            MessageBox(_T("你的手机已经超出范围！"));

    }
    else if (nChar == 'd')
    {
        --pho.X;
        Draw();
        if (((pho.X - 650)*(pho.X - 650) + (pho.Y - 350)*(pho.Y -
350)) >= pho.Pradius*pho.Pradius)
            MessageBox(_T("你的手机已经超出范围！"));

    }
    else if (nChar == 'f')
    {
        ++pho.X;
        Draw();
        if (((pho.X - 650)*(pho.X - 650) + (pho.Y - 350)*(pho.Y -
350)) >= pho.Pradius*pho.Pradius)
            MessageBox(_T("你的手机已经超出范围！"));

    }
    else if (nChar == 'z')
    {
        --wat.Y;
        Draw();
        if (((wat.X - 650)*(wat.X - 650) + (wat.Y - 350)*(wat.Y -
350)) >= wat.Wradius*wat.Wradius)
            MessageBox(_T("你的手表已经超出范围！"));
```

```
    }
    else if (nChar == 'x')
    {
        ++wat.Y;
        Draw();
        if (((wat.X - 650)*(wat.X - 650) + (wat.Y - 350)*(wat.Y -
350)) >= wat.Wradius*wat.Wradius)
            MessageBox(_T("你的手表已经超出范围！"));

    }
    else if (nChar == 'c')
    {
        --wat.X;
        Draw();
        if (((wat.X - 650)*(wat.X - 650) + (wat.Y - 350)*(wat.Y -
350)) >= wat.Wradius*wat.Wradius)
            MessageBox(_T("你的手表已经超出范围！"));

    }
    else if (nChar == 'v')
    {
        ++wat.X;
        Draw();
        if (((wat.X - 650)*(wat.X - 650) + (wat.Y - 350)*(wat.Y -
350)) >= wat.Wradius*wat.Wradius)
            MessageBox(_T("你的手表已经超出范围！"));

    }
    else if (nChar == 'u')
    {
        --surf.Y;
        Draw();
        if (((surf.X - 650)*(surf.X - 650) + (surf.Y - 350)*(surf.Y
- 350)) >= surf.Sradius*surf.Sradius)
            MessageBox(_T("你的平板电脑已经超出范围！"));

    }
    else if (nChar == 'i')
    {
```

```
        ++surf.Y;
        Draw();
        if (((surf.X - 650)*(surf.X - 650) + (surf.Y - 350)*(surf.Y
- 350)) >= surf.Sradius*surf.Sradius)
            MessageBox(_T("你的平板电脑已经超出范围！"));

    }
    else if (nChar == 'o')
    {
        --surf.X;
        Draw();
        if (((surf.X - 650)*(surf.X - 650) + (surf.Y - 350)*(surf.Y
- 350)) >= surf.Sradius*surf.Sradius)
            MessageBox(_T("你的平板电脑已经超出范围！"));

    }
    else if (nChar == 'p')
    {
        ++surf.X;
        Draw();
        if (((surf.X - 650)*(surf.X - 650) + (surf.Y - 350)*(surf.Y
- 350)) >= surf.Sradius*surf.Sradius)
            MessageBox(_T("你的平板电脑已经超出范围！"));

    }
    else if (nChar == 'h')
    {
        --pda.Y;
        Draw();
        if (((pda.X - 650)*(pda.X - 650) + (pda.Y - 350)*(pda.Y -
350)) >= pda.PDAradius*pda.PDAradius)
            MessageBox(_T("你的 PDA 已经超出范围！"));

    }
    else if (nChar == 'j')
    {
        ++pda.Y;
        Draw();
        if (((pda.X - 650)*(pda.X - 650) + (pda.Y - 350)*(pda.Y -
350)) >= pda.PDAradius*pda.PDAradius)
```

```
            MessageBox(_T("你的 PDA 已经超出范围！"));

    }
    else if (nChar == 'k')
    {
        --pda.X;
        Draw();
        if (((pda.X - 650)*(pda.X - 650) + (pda.Y - 350)*(pda.Y -
350)) >= pda.PDAradius*pda.PDAradius)
            MessageBox(_T("你的 PDA 已经超出范围！"));

    }
    else if (nChar == 'l')
    {
        ++pda.X;
        Draw();
        if (((pda.X - 650)*(pda.X - 650) + (pda.Y - 350)*(pda.Y -
350)) >= pda.PDAradius*pda.PDAradius)
            MessageBox(_T("你的 PDA 已经超出范围！"));

    }
    else if (nChar == 'n')
    {
        --ear.Y;
        Draw();
        if (((ear.X - 650)*(ear.X - 650) + (ear.Y - 350)*(ear.Y -
350)) >= ear.Eradius*ear.Eradius)
            MessageBox(_T("你的蓝牙耳机已经超出范围！"));

    }
    else if (nChar == 'm')
    {
        ++ear.Y;
        Draw();
        if (((ear.X - 650)*(ear.X - 650) + (ear.Y - 350)*(ear.Y -
350)) >= ear.Eradius*ear.Eradius)
            MessageBox(_T("你的蓝牙耳机已经超出范围！"));

    }
```

```
    else if (nChar == ',')
    {
        --ear.X;
        Draw();
        if (((ear.X - 650)*(ear.X - 650) + (ear.Y - 350)*(ear.Y -
350)) >= ear.Eradius*ear.Eradius)
            MessageBox(_T("你的蓝牙耳机已经超出范围！"));

    }
    else if (nChar == '.')
    {
        ++ear.X;
        Draw();
        if (((ear.X - 650)*(ear.X - 650) + (ear.Y - 350)*(ear.Y -
350)) >= ear.Eradius*ear.Eradius)
            MessageBox(_T("你的蓝牙耳机已经超出范围！"));

    }
}
```

用四个按键分别代表一个设备的上下左右，使用 nChar 接受用户输入的指令，每移动一次都相应判断一次该设备是否移出了通讯半径，若移出了通讯半径，则弹出一个对话框告诉用户相应设备在通讯范围外

由于 Windows 在画图界面下移动产生的是叠加效应，所以我们每进行一次移动都要重新将屏幕刷新一次，以避免产生图片重叠的情况。

屏幕刷新实现代码：

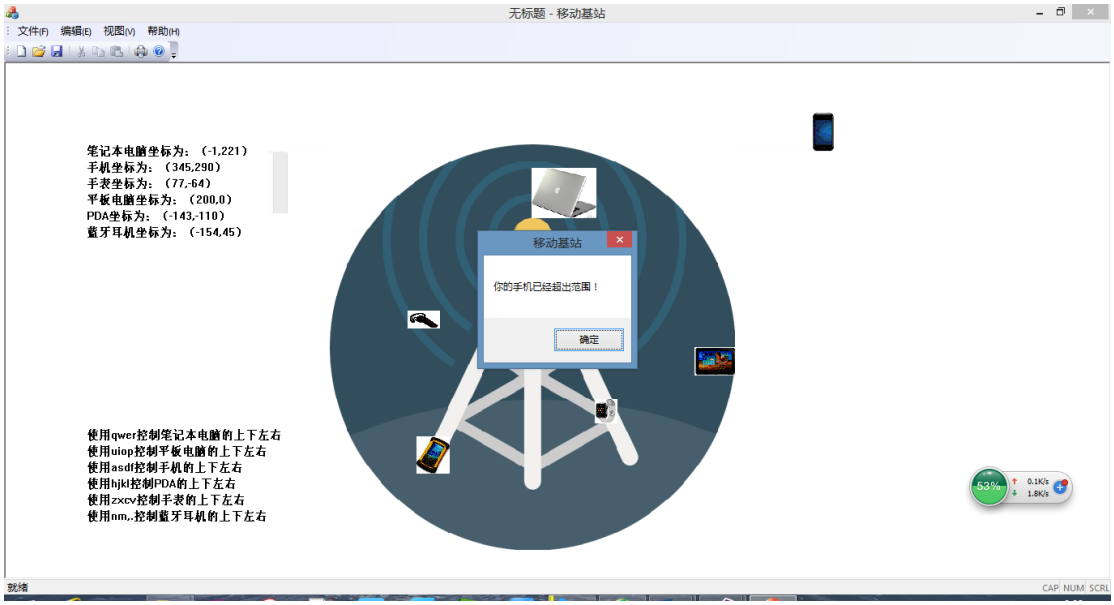
```
void C移动基站View::Draw()
{
    CImage img, img1, img2, img3, img4, img5, img6;
    CClientDC dcd(this);

    img.Load(_T("res//base.png"));
    img.Draw(dcd, 400, 100, 500, 500);

    img1.Load(_T("res//bijiben.png"));
```

```
    img1.Draw(dcd, lap.X, lap.Y, 80, 62);

    img2.Load(_T("res//shouji.png"));
    img2.Draw(dcd, pho.X, pho.Y, 28, 49);

    img3.Load(_T("res//shoubiao.png"));
    img3.Draw(dcd, wat.X, wat.Y, 28, 30);

    img4.Load(_T("res//pingban.png"));
    img4.Draw(dcd, surf.X, surf.Y, 51, 35);

    img5.Load(_T("res//PDA.png"));
    img5.Draw(dcd, pda.X, pda.Y, 41, 46);

    img6.Load(_T("res//lanyaerji.png"));
    img6.Draw(dcd, ear.X, ear.Y, 40, 22);
    CClientDC cdc2(this);

    CString str;

    str.Format(_T("笔记本电脑坐标为：(%d,%d)"), lap.X - 650, 350 -
lap.Y);
    cdc2.TextOutW(100, 100, str);

    str.Format(_T("手机坐标为：(%d,%d)"), pho.X - 650, 350 - pho.Y);

    cdc2.TextOutW(100, 120, str);

    str.Format(_T("手表坐标为：(%d,%d)"), wat.X - 650, 350 - wat.Y);

    cdc2.TextOutW(100, 140, str);

    str.Format(_T("平板电脑坐标为：(%d,%d)"), surf.X - 650, 350 -
surf.Y);
    cdc2.TextOutW(100, 160, str);

    str.Format(_T("PDA 坐标为：(%d,%d)"), pda.X - 650, 350 - pda.Y);

    cdc2.TextOutW(100, 180, str);

   str.Format(_T("蓝牙耳机坐标为：(%d,%d)"), ear.X - 650, 350 - ear.Y);

    cdc2.TextOutW(100, 200, str);

    cdc2.TextOutW(100, 450, _T("使用 qwer 控制笔记本电脑的上下左右"));

    cdc2.TextOutW(100, 470, _T("使用 uiop 控制平板电脑的上下左右"));

    cdc2.TextOutW(100, 490, _T("使用 asdf 控制手机的上下左右"));
```

```
    cdc2.TextOutW(100, 510, _T("使用 hjkl 控制 PDA 的上下左右"));
    cdc2.TextOutW(100, 530, _T("使用 zxcv 控制手表的上下左右"));
    cdc2.TextOutW(100, 550, _T("使用 nm,.控制蓝牙耳机的上下左右"));

}
```

这样，我们就能每进行一次移动之后都实时看到最新的移动之后的图像而没有叠加效应。

最后，附上程序运行的结果：



实验三、实现电影院售票

自定义类

Cinema 类，其实例化产生的对象就是一个具体的电影院。

● 数据成员

对于一个 Cinema 首要的就是说明其规模(N×N 的正方形)。

```
int N;
```

available 数组，表示对应的位置是否可买，如果已经卖出就为 false。

selected 数组，表示对应的位置是否被选中。

这两个数组的大小都为 N*N，动态申请。

```
bool *available;
bool *selected;
```

另外是两个关于特定查询的数据成员：QueryList 与 QuerySize

```
int *QueryList;
int QuerySize;
```

QueryList 表示在一次 Query 中查询到的列表（多解问题）。

QuerySize 则表示对应查询的答案规模（有多少解）。

*拓展：这里可以使用 vector 来存 Query，可以更好地利用空间。

● 构造函数

构造函数需要给出 Cinema 的大小，在这里要用 new 运算符申请足够的空间。同时注意在之后的操作中不要有越界行为。另外也要对数组进行必要的初始化。available 与 selected 必须逐一赋值，然而 Query 相关的只要使 QuerySize=0 就可以了。

```
cinema::cinema(int n)
{
    N=n;
    available=new bool[n*n];
    selected=new bool[n*n];
    QueryList=new int [n*n];
    for(int i=0;i<N*N;i++){
        available[i]=true;
        selected[i]=false;
    }
    QuerySize = 0;
}
```

- **成员函数（方法）**

➢ 读取状态方法

由于成员都是私有的，在类外直接访问被禁止，所以要开放一个方法来查询状态，特别是票能不能买（available）与该票有没有被选中（selected）。GetPos 与 GetQuerySize 是 Query 相关的状态读取方法。

```
bool IsAvailable();
bool IsSelected();
int GetPos(int index);
int  GetQuerySize();
```

➢ 写入状态方法

同上理，有这样的设置：

Select：改变 index 位置的 selected 状态，选中与未选中的切换。

```
void Select(int index);
```

SelectClear:清空所有位置的 selected 状态，就是取消所有选择。

```
void SelectClear();
```

➢ 实际效应方法

以上读写方法的实现都很容易，重点是后面的实际效应方法。

Buy（买票）与 query（查询）方法。

注意这里断言(assert)：index 位置的票是可买的，所以在调用 Buy 之前要确保可买。

```
void cinema::Buy(int index)
{
    assert(available[index]);
    available[index]=false;
}
```

Query（方形连票查询）是实验要求的查询 R×R 的位置的方法，返回值为找到的方案数，即查询中符合条件的解的个数(QuerySize)。同时会将结果存储在 QueryList 中。

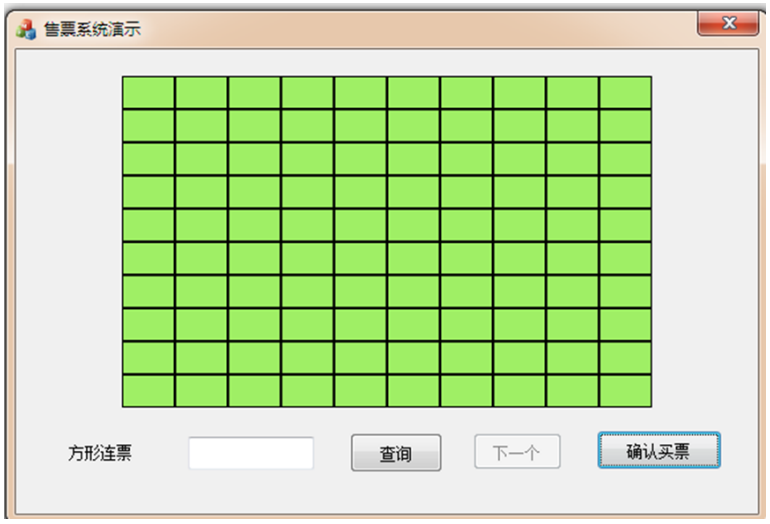
这里的这种 Query 方案，本质是暴力法。

```
int cinema::query(int R)
{
    QuerySize = 0;
    if (R > N)
        return 0;
    for (int t = 0; t <= N - R; t++)
    for (int i = t*N; i <= t*N + N - R; i++)
    if (available[i] == true)
    {
        int s = 0;
        for (int j = 0; j < R; j++)
        for (int k = 0; k < R; k++)
        if (available[i + j*N + k] == true)
            s = s + 1;
        if (s == R*R)//是解
        {
            //存储解
            QueryList[QuerySize++] = i;
        }
    }
```

```
    return QuerySize;
}
```

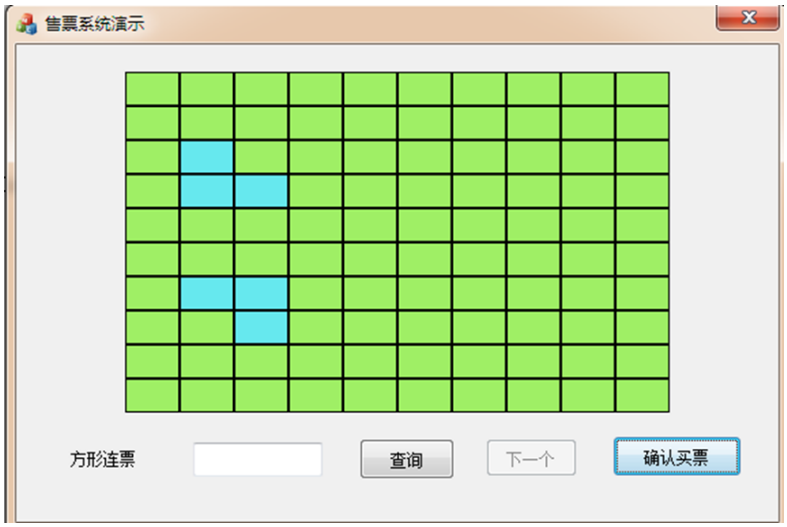
对话框类

● 功能演示



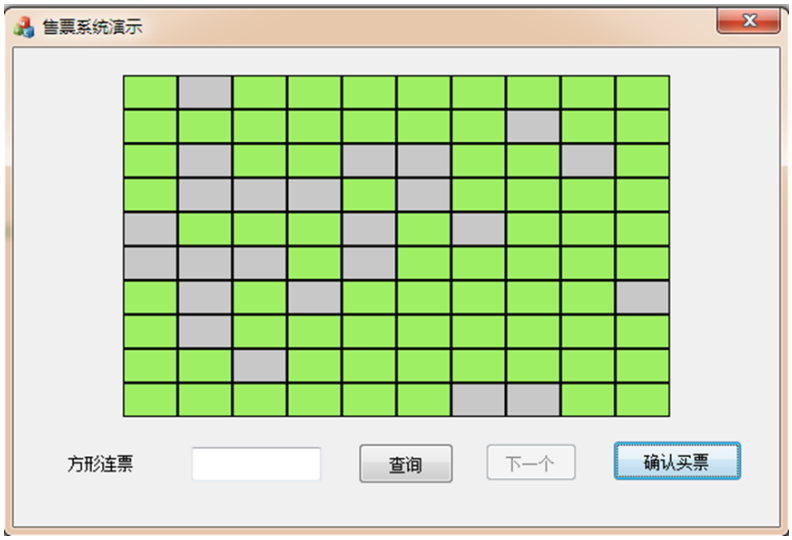
进入售票窗口，包括座位（鼠标点击某一座位位置可以选定，初始绿色表示该座位的票未售出）、输入框（输入1~10的数字查询方形连票）、【查询】按钮、【下一个】按钮（单解时此按钮不可按，多解时激活，点击显示下一组解所在位置，并循环显示）、【确认买票】按钮（选定座位后买票）。

输入12（超出10）进行查询，弹出对话框，显示无符合条件的查询结果。



用鼠标点击想购买的位置，区域颜色转为蓝色，表示已选定。可同时选定多个座位。

点击【确认买票】，弹出窗口（通过messagebox实现），显示买票成功，所选位置颜色变成灰色，表示该座位的票已经售出。



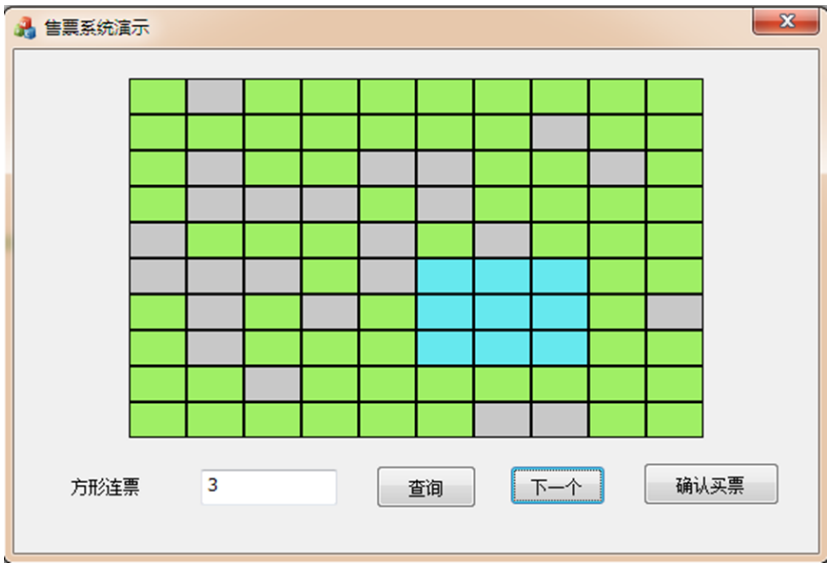
灰色位置（已售出的座位）不可再被点击（选中），售票一段时间后座位情况如图所示。

此时开始查询5x5方阵的票，显示无符合条件的查询结果。

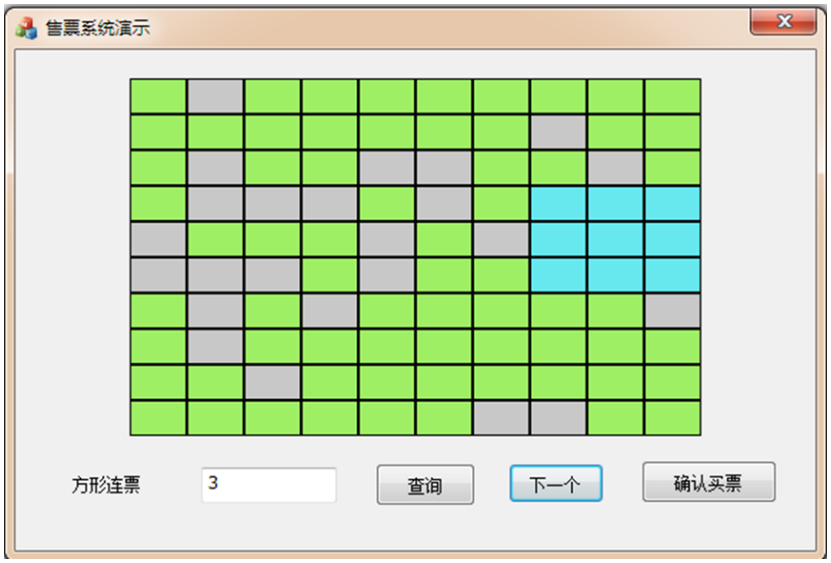
查询4x4方阵的连票，弹出窗口，显示方阵左上角的坐标，同时，在座位区域自动将第一

组解（在这里是唯一解）选定，方阵变为蓝色，从而显示出来。

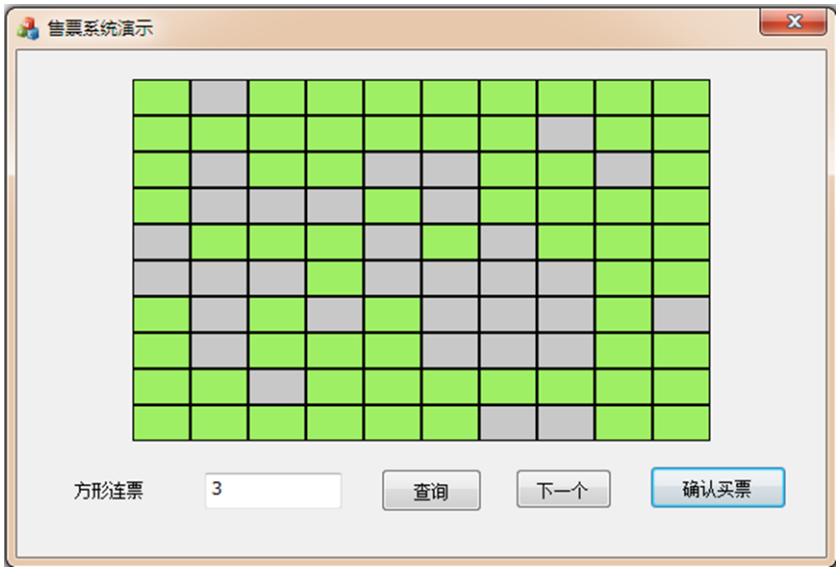
查询3x3方阵的连票，显示多解，并在座位区域将第一组解选定。



点击【下一个】按钮后，选定区域跳至下一个解。



继续点击【下一次】按钮，某一次点击后显示的座位。



通过点击【下一个】按钮，比对剩余可选位置后，找到理想位置的方形连票，成功买票后界面如图所示。

● 全局变量（成员变量）

此 MFC 工程是基于对话框的，所以对话框类就是主体（全局）。

```
const int N=10;
cinema Cinema(N);
int StartX=80,StartY=20;
int Width=40,Height=25;
```

```
int SelectIndex;
```

其中，N 表示电影院规模，StartX,StartY 表示图上绘制表格的左上角坐标，Width,Height 表示格子的宽度与高度。

Query 操作会产生 QuerySize 个解，而当前选中其中的第 SelectIndex 个解。

● 消息&方法

➢ 绘图 OnPaint()

这是关键的可视化方法。初始化的时候会调用一次，另外每当状态发生改变的时候要及时调用 OnPaint()进行重绘，确保即时变化，没有延迟的 BUG。

```
void CMFCDlg::OnPaint()
{
    if (IsIconic()){…}
    else{…}
        CDC* dc=GetDC();

        //创建画笔，定义（填充）颜色

        CBrush NOT_AVAILABLE,AVAILABLE,SELECTED;
        NOT_AVAILABLE.CreateSolidBrush(RGB(200,200,200));
        AVAILABLE.CreateSolidBrush(RGB(159,239,101));
        SELECTED.CreateSolidBrush(RGB(102,232,238));

        for(int i=0;i<N;i++)
            for(int j=0;j<N;j++)
            {

                //选择画笔与颜色

                dc->SelectObject(NOT_AVAILABLE);
                if(Cinema.IsAvailable(i*N+j))
                    dc->SelectObject(AVAILABLE);
                if(Cinema.IsSelected(i*N+j))
                    dc->SelectObject(SELECTED);


        //画矩形（格子）

    dc->Rectangle(StartX+Width*i,StartY+Height*j,StartX+Width*(i+1
),StartY+Height*(j+1));
```

```
        }
}
```

➢ 鼠标点击 OnLButtonDown()

当鼠标点击格子时，选中该格子。

这里利用简单的数学知识计算了鼠标位置与点击的格子的映射关系。

另外内层的 if 语句确保了"已经卖出的票不能被选中"。

```
void CMFCDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值

    //合法性：点击区域在规定范围内
    if(point.x>StartX&&point.y>StartY&&point.x<StartX+Width*N&&poi
nt.y<StartY+Height*N)
    {
        int i=(point.x-StartX)/Width;
        int j=(point.y-StartY)/Height;
        if(Cinema.IsAvailable(i*N+j))
            Cinema.Select(i*N+j);

        //重绘
        OnPaint();
    }
    CDialogEx::OnLButtonDown(nFlags, point);
}
```

➢ 确认买票 OnBnClickedButton1()

这里 Button1 是【确认购票】的按钮控件。

```
void CMFCDlg::OnBnClickedButton1()
{
    // TODO: 在此添加控件通知处理程序代码

    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
```

```
            //可买&&已选
            if(Cinema.IsAvailable(i*N+j)&&Cinema.IsSelected(i*N+j))
            {
                //买
                Cinema.Buy(i*N+j);
                //取消选择
                Cinema.Select(i*N+j);
            }
    //重绘
    OnPaint();
    MessageBox(CString("买票成功！"),CString(" "));
}
```

➢ 方形连票查询 OnBnClickedButton2()

这里 Button2 是【查询】按钮。流程在代码注释里讲得很清楚了。

```
void CMFCDlg::OnBnClickedButton2()
{
    // TODO: 在此添加控件通知处理程序代码
    //获取文本框中的数字=>R
    CString S;
    Edit1.GetWindowTextW(S);
    int R=_ttoi(S);
    //调用 query 函数
    int cnt=Cinema.query(R);
    //非法
    if(R<=0) return;
    //无解
    if(cnt==0)
    {
        MessageBox(CString("对不起，没有符合条件的查询结果"),CString("
"));
```

```
        //清除所有已选
        Cinema.SelectClear();
        //禁用【下一个】按钮
        Button3.EnableWindow(false);
    }
    //有解
    else
    {
        //取消已选
        Cinema.SelectClear();
        //从第一个开始
        SelectIndex=0;
        //获取第一个解的位置
        int pos=Cinema.GetPos(SelectIndex);
        //选中对应的格子
        for(int i=0;i<R;i++)
            for(int j=0;j<R;j++)
                Cinema.Select(pos+i*N+j);
        //显示提示
        CString Result,S;
        Result.Format(_T("共有%d种选择，方阵左上角坐标为:\n"),Cinema.GetQuerySize());
        for(int i=0;i<Cinema.GetQuerySize();i++)
        {
            pos=Cinema.GetPos(i);
            S.Format(_T("第%d行，第%d列\n"),pos%N+1,pos/N+1);
            Result+=S;
        }
        MessageBox(Result,CString("查询结果"));
        //启用【下一个】按钮
        Button3.EnableWindow();
```

```
    }
    //重绘
    OnPaint();
}
```

下一个查询结果 OnBnClickedButton3()

这里 Button3 是【下一个】按钮。初始状态是 Disabled（禁用）状态的，原因很简单，仅当有解的情况下才有【下一个】解的概念。其功能就是跳转到下一个查询解，与 Button2（查询）按钮配套使用。

```
void CMFCDlg::OnBnClickedButton3()
{
    // TODO: 在此添加控件通知处理程序代码
    //获取文本框数字
    CString S;
    Edit1.GetWindowTextW(S);
    int R=_ttoi(S);
    //获取当前选中解位置,并取消选中
    int pos=Cinema.GetPos(SelectIndex);
    for(int i=0;i<R;i++)
        for(int j=0;j<R;j++)
            Cinema.Select(pos+i*N+j);
    //下一个解（循环）
    SelectIndex=(SelectIndex+1)%Cinema.GetQuerySize();
    //获取解的位置，选中
    pos=Cinema.GetPos(SelectIndex);
    for(int i=0;i<R;i++)
        for(int j=0;j<R;j++)
            Cinema.Select(pos+i*N+j);
    //重绘
    OnPaint();
}
```

*DEBUG：错误的变更与【下一个】按下时间

分析之前的代码可以发现，【下一次】按钮事件，会即时查询文本框中的数字，如果之前获得了 4×4 的解，而点击【下一次】之前将文本框中数字改为 5，解空间还是 4×4 的解，而程序会当成 5×5 的解处理，这时就会出现 Bug。

这种情况归根结底就是变更查询的时候没能及时重置造成的，只要重载文本框的消息函数 OnEnChangeEdit1()即可（这里 Edit1 是文本框控件变量）。

在变更数字的时候禁用【下一个】按钮即可完成 Debug。

```
void CMFCDlg::OnEnChangeEdit1()
{
    Button3.EnableWindow(false);
}
```

实验四、记事本

实验要求：

实现记事本小程序，能够设定事件通知信息，并按照设定的时间进行事件通知，具体要求如下：

1．界面友好，美观大方；

2．支持事件的录入，包括时间、事件、备注信息等；

3．支持在设定的时间进行事件提醒；

4．支持对已录入事件大的查看、修改和删除；

5. 其他更多功能，可根据情况自行添加。

实验结果：

1.程序实现了事件的录入，包括时间、事件、备注信息。

2.支持在特定的时间进行提醒。

3.程序常驻内存，并且可以最小化到 Windows 下的任务栏或者 OS X 下的 Docker，使用鼠标右键唤出。

4.程序完成了 UI 的大小自适应。

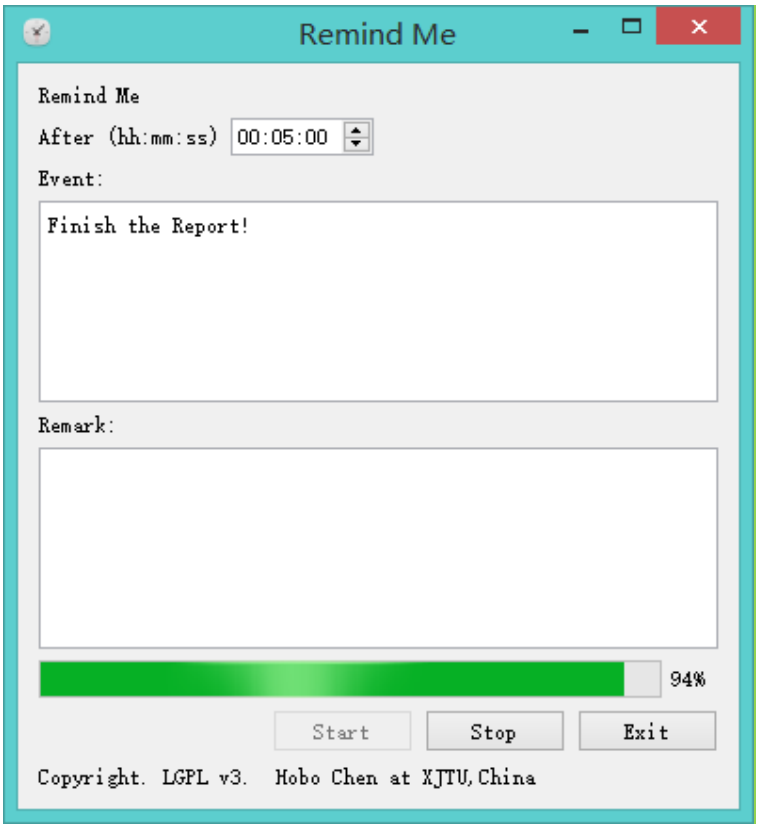
5.程序自一开始即使用 Git 进行版本控制，并且抛弃了一切 API 重构（比如 MFC）以保证完整的跨平台特性。

6.程序完成了并行化与多实例化，每个提醒的实例均独立占用一个线程。

7.程序通过调用模拟 API 实现了在时间到的时候在屏幕的右下角弹出提醒，并默认放置

于所有程序的最前方。

最终应用程序在 Windows 下使用 Qr5.5.0+MSVC2013_x64 编译，在 Windows 8.1 下的运行界面如图。



源码文件

Reminder.pro 文件

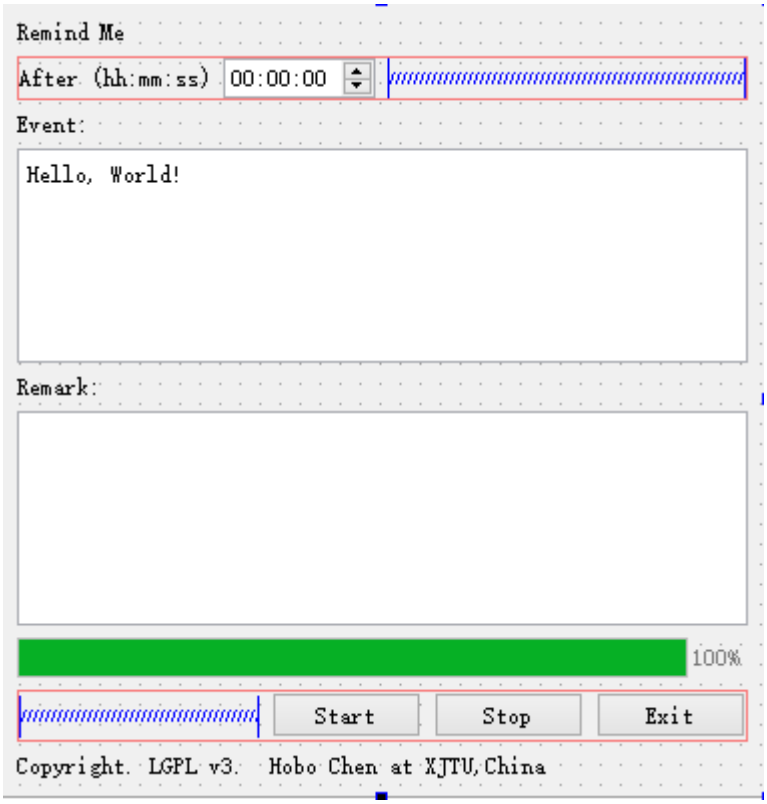
```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET =    Reminder
TEMPLATE =  app
SOURCES +=  main.cpp\
          window.cpp \
          popup.cpp
```

```
HEADERS  += window.h \
          popup.h
FORMS    += window.ui \
          popup.ui

RESOURCES += \
    icons.qrc
RC_FILE += reminder.rc
```

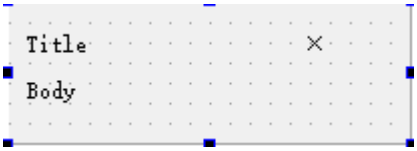
Main.cpp 文件：

```
#include <QApplication>
#include "window.h"

int main(int argc, char *argv[])
{
    // create an application with commandline arguments
    QApplication a(argc, argv);
    // turn off quit the application when all windows are closed
    a.setQuitOnLastWindowClosed(false);
    Window w;
    // show the window
    w.show();

    return a.exec();
}
```

Windows.ui 文件：



Popup.ui 文件



Windows.h 文件

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTimer>
#include <QDesktopWidget>
#include <QMessageBox>
#include <QSystemTrayIcon>

#include "popup.h"
```

```
namespace Ui {
class Window;
}

class Window : public QWidget
{
    Q_OBJECT

public:
    explicit Window(QWidget *parent = 0);
    ~Window();

public slots:
    void resetReminder();
    void restartReminder();

private slots:
    void startTimer();
    void stopTimer();
    void showReminder();
    void showProgress();
    void
systemTrayIconActivated(QSystemTrayIcon::ActivationReason);

private:
    Ui::Window *ui;
    QTimer timer_progress;
    QTimer timer_reminder;
    uint reminder_time;
    QSystemTrayIcon *systemTrayIcon;
    Popup *popup;
};

#endif // WIDGET_H
```

Windows.cpp 文件

```
#include "window.h"
#include "ui_window.h"

// inherit QWidget to create a window
Window::Window(QWidget *parent) : QWidget(parent), ui(new
Ui::Window)
```

```
{
    // init reminder_time to zero
    reminder_time = 0;
    // create a Popup window variable
    popup = new Popup;

    // create a system tray
    systemTrayIcon = new QSystemTrayIcon(this);
    // set it's icon
    systemTrayIcon->setIcon(QIcon(":/reminder-icon"));
    // show the tray icon
    systemTrayIcon->show();

    // create reminder window form
    ui->setupUi(this);
    // disable Stop button on startup
    ui->bStop->setDisabled(true);

    // call startTimer() when Start button is clicked
    connect(ui->bStart, SIGNAL(clicked()), this,
SLOT(startTimer()));
    // call sopTimer() when Stop button is clicked
    connect(ui->bStop, SIGNAL(clicked()), this,
SLOT(stopTimer()));
    // quit the application when Exit button is clicked
    connect(ui->bExit, SIGNAL(clicked()), qApp, SLOT(quit()));

    // show progress on 1s timer
    connect(&timer_progress, SIGNAL(timeout()), this,
SLOT(showProgress()));
    // actual timer used by the application
    connect(&timer_reminder, SIGNAL(timeout()), this,
SLOT(showReminder()));

    // when the cross button of the popup is clicked, call
resetReminder()
    connect(popup, SIGNAL(closed()), this, SLOT(resetReminder()));
    // when the popup is clicked anywhere, call restartReminder()
    connect(popup, SIGNAL(clicked()), this,
SLOT(restartReminder()));

    // show or hide the window when system tray icon is clicked
    connect(systemTrayIcon,
SIGNAL(activated(QSystemTrayIcon::ActivationReason)),
```

```
            this,
SLOT(systemTrayIconActivated(QSystemTrayIcon::ActivationReason)))
;

    // center the window inside the desktop
    setGeometry(QStyle::alignedRect(
                Qt::LeftToRight,
                Qt::AlignCenter,
                this->size(),
                qApp->desktop()->availableGeometry()));

    setWindowTitle("Remind Me");
}

Window::~Window()
{
    delete ui;
}

void Window::resetReminder()
{
    // clear all data
    ui->about->clear();
    ui->timeAfter->setTime(QTime(0,0,0));
    ui->timeAfter->setFocus();
}

void Window::restartReminder()
{
    // show the window and let user set custom time
    ui->timeAfter->setFocus();
    showNormal();
}

void Window::startTimer()
{
    // get time value from user
    QTime after = ui->timeAfter->time();
    // convert it into microseconds
    reminder_time = (after.hour()*3600 + after.minute()*60 +
after.second()) * 1000;

    // if the time is less than 1 second or 1 microsecond
    if(reminder_time < 1000)
```

```
    {
        // show error
        QMessageBox::critical(this,
                              "Error",
                              "Cannot set timer for less than 1s");
        return;
    }

    // start the timer with user specified time
    timer_reminder.start(reminder_time);
    // set progressbar maximum value
    ui->progressBar->setMaximum(reminder_time);
    ui->progressBar->setValue(reminder_time);
    // start updating progress
    showProgress();
    timer_progress.start(1000);

    // enable the progressbar, disable Start button, enable Stop
button
    ui->progressBar->setEnabled(true);
    ui->bStart->setDisabled(true);
    ui->bStop->setEnabled(true);

    // hide itself
    close();
}

void Window::stopTimer()
{
    // stop everything
    timer_progress.stop();
    timer_reminder.stop();
    ui->progressBar->reset();
    // disable the progressbar
    ui->progressBar->setEnabled(false);
    // enable Start button
    ui->bStart->setDisabled(false);
    // disable Stop button
    ui->bStop->setEnabled(false);
}

void Window::showReminder()
{
    // stop the timer
```

```
    stopTimer();
    // show desktop popup
    popup->showPopup(
              "Reminder",
              ui->about->toPlainText());
}

void Window::showProgress()
{
    // update values to the progressbar
    ui->progressBar->setValue(timer_reminder.remainingTime());
}

void
Window::systemTrayIconActivated(QSystemTrayIcon::ActivationReason
)
{
    // if the window is hidden, show it
    if(!isVisible())
        showNormal();
    // else hide it
    else
        close();
}
```

Popup.h 文件

```
#ifndef POPUP_H
#define POPUP_H

#include <QDialog>
#include <QDesktopWidget>
#include <QMouseEvent>

namespace Ui {
class Popup;
}

class Popup : public QDialog
{
    Q_OBJECT

public:
    explicit Popup(QWidget *parent = 0);
```

```
    ~Popup();

    void showPopup(QString title, QString description);
    void mousePressEvent(QMouseEvent*);

private slots:
    void closePopup();

signals:
    void closed();
    void clicked();

private:
    Ui::Popup *ui;
};

#endif // POPUP_H
```

Popup.cpp 文件

```
#include "popup.h"
#include "ui_popup.h"

Popup::Popup(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Popup)
{
    ui->setupUi(this);

    setAttribute(Qt::WA_ShowWithoutActivating);

    setWindowFlags(
              Qt::FramelessWindowHint | // No window border
              Qt::WindowDoesNotAcceptFocus | // No focus
              Qt::WindowStaysOnTopHint // Always on top
              );

    connect(ui->button, SIGNAL(clicked()), this,
SLOT(closePopup()));

    setWindowTitle("Reminder Notification");
}

Popup::~Popup()
```

```
{
    delete ui;
}

void Popup::showPopup(QString title, QString description)
{
    // show the title in bold
    ui->title->setText(QString("<b>%1</b>").arg(title));
    // and the body in normal text
    ui->description->setText(description);

    // position the notification at desktop right bottom
    setGeometry(QStyle::alignedRect(
                Qt::RightToLeft,
                Qt::AlignBottom,
                size(),
                qApp->desktop()->availableGeometry()));
    show();
}

void Popup::closePopup()
{
    emit closed();
    close();
}

void Popup::mousePressEvent(QMouseEvent*)
{
    emit clicked();
    close();
}
```

GUI 设计：

在 GUI 设计的过程中，我力争简洁、美观、易用，同时还确保了程序在跨平台下的兼容性。

首先，程序实现了在界面的自适应，即在全屏或者手动调整下实现了所有的控件能够根据原有的设计自行更改大小。极大地改善了程序的美观程度与易用程度。

其次，我更改了默认的程序图标和常驻任务栏的图标，使其更像一个成品的可以商用的程序而不是一个大作业。

最后，我添加了版权信息，并使用 LGPLv3 协议进行授权，并在 GitHub 上开放了所有的源代码，以让更多有需要有能力的人能在我的基础上进行二次开发。

实验五、七巧板

七巧板是我国传统的智力游戏与消遣工具，仅仅 7 块小板，竟能千变万化，因此深受人们喜爱。现在要将这个经典的游戏利用 Visual C++再现到计算机上。

自定义类

几何形状类 Shape

构造函数

构造函数有 3 份重载

```
// 默认构造函数
Shape();
// 预设块的构造函数
Shape(int ID);
// 从预设 Shape 中拷贝构造前 cnt 个图形
Shape(Shape * S, int cnt);
```

预设块

一般来说，预设块是指七巧板中有 7 块经典的预设拼块。

为了增加游戏性，在后面的版本中还可能加入新的拼块，甚至用户自定义的拼块。

数据成员

某些全局常量可以利用静态成员变量来定义，避免使用宏来规避一些意外的错误。

```
static int SIZE;
static CPoint START;
```

SIZE 表示整个拼块的大小（一开始七巧板是拼成一个大正方形的，这个 SIZE 就是指其边长），START 表示初始化时拼块们的起点，这些拼块在初始化构造的时候将使用 START 变量进行定位。

```
int MAX_POINT;
CPoint *Vertex;
```

MAX_POINT 表示这个拼块的顶点数，而 Vertex 数组则用于存放顶点。

```
COLORREF COLOR;
```

COLOR 表示拼块的颜色，这通常使用 Visual Studio 自带的 RGB 宏来进行赋值。

```
bool selected
```

selected 表示拼块是否被选中，选中的拼块是当前要操作的拼块，我们应该确保不能有多个拼块被同时选中。

成员函数（方法）

```
void Draw(CDC * pDC,int mode=0);
void Select();
void Select(bool flag);
bool IsInside(CPoint point);
CPoint GetCenter();
void Move(CPoint start, CPoint end);
void Rotate(CPoint Center, int deg);
```

绘图函数

对一个图形的绘制，首先要确定载体，即画在哪里的问题，这个可以用 CDC* 来定位，另外就是绘制模式 mode。

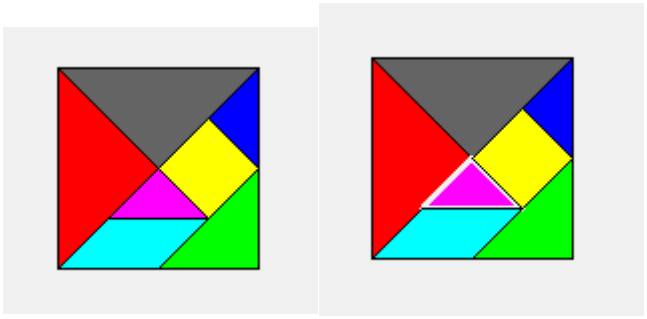
要知道绘制的东西是不能擦除的，只能覆盖。比如说一个物体移动了，我重绘了新的位置，然而仅仅是这样的话，老地方会残留着物体的残影。

解决方案之一是在每次重绘之前将整个可见区域刷白。

解决方案之二就是只将必要区域刷白。

显然方案二的效率要高许多，具体的实现就是考虑不同的绘制模式 `mode`，`mode` 默认为 `0`，当 `mode` 为 `1` 的时候，将画笔与刷子的颜色改为背景色，这样就能将最小的必要区域刷白。

另外一个小细节是，选中（`Selected`）与未选中（`!Selected`）的边框是不一样的，这样就可以造成“**选中的突出效果**”。



```
void Shape::Draw(CDC * pDC,int mode)
{
    CBrush B;
    CPen NOT_SELECTED,SELECTED;
    CBrush ERASE;

    ERASE.CreateSolidBrush(RGB(240, 240, 240));
    NOT_SELECTED.CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
    SELECTED.CreatePen(PS_SOLID, 3, RGB(240, 240, 240));
    B.CreateSolidBrush(COLOR);
    if (selected)
        pDC->SelectObject(SELECTED);
    else
        pDC->SelectObject(NOT_SELECTED);
    if (mode)
        pDC->SelectObject(ERASE);
    else
        pDC->SelectObject(B);
    pDC->Polygon(Vertex, MAX_POINT);
}
```

方法：选择

这里 Select 有 2 份重载，默认是“改变选择状态”，选中的变为未选中的，未选中的变为选中的。另外一份重载则是直接为 selected 赋值，这在我们不知道对象是选中的还是未选中的时候有用。

```
// 选中/取消
void Shape::Select()
{
    selected = !selected;
}
// SetSelection
void Shape::Select(bool flag)
{
    selected = flag;
}
```

方法：几何关系

IsInside 函数给出了“某点是否在多边形内部”的答案，这在判断“鼠标是否点到了拼块”是有用的。GetCenter 则是给出“图形的默认旋转中心”的答案，由于有特殊存储方法，Vertex 是按照顺序存储的，只要确保等腰直角三角形中 Vertex[0]与 Vertex[2]都是斜边上的顶点，就能保证算法的正确性。

```
bool Shape::IsInside(CPoint point)
{
    CRgn rgn;
    rgn.CreatePolygonRgn(Vertex, MAX_POINT, ALTERNATE);
    return rgn.PtInRegion(point);
}
CPoint Shape::GetCenter()
{
    return CPoint((Vertex[0].x + Vertex[2].x) / 2, (Vertex[0].y +
Vertex[2].y) / 2);
}
```

方法：变更

Rotate（旋转）与 Move（平移）是两种基本的方法。

其中，Rotate 的一般实现要考虑旋转的坐标变换：

逆时针旋转的矩阵运算公式：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

另外，由于算法几何上的精度问题，必须尽量减小浮点运算带来的误差。对于浮点运算要尽量利用更多的有效位数来保证精度。

```
void Shape::Rotate(CPoint Center,int deg)
{
    double rad = deg *Pirad;
    LONG rate = 1000000;
    for (int i = 0; i < MAX_POINT; i++)
    {

        double dx = (Vertex[i].x - Center.x)*rate;
        double dy = (Vertex[i].y - Center.y)*rate;
        Vertex[i].x  =  (LONG)((cos(rad)*dx  -  sin(rad)*dy)/rate  +
Center.x);
        Vertex[i].y  =  (LONG)((sin(rad)*dx  +  cos(rad)*dy)/rate  +
Center.y);
    }
}
```

Move（平移）的实现就相对简单得多，只要知道一个向量即可，或者说，从何方来，又往何处去……

```
void Shape::Move(CPoint start,CPoint end)
{
    for (int i = 0; i < MAX_POINT; i++)
    {
        Vertex[i].x += end.x - start.x;
        Vertex[i].y += end.y - start.y;
    }
```

```
}
```

事件&消息

鼠标响应

鼠标按下的时候应该做什么呢？

当左键按下，如果判断到鼠标在某拼块上，那么意味着选中了这个拼块。用一个 Dlg 类中的成员变量（实际上意味着全局变量）SelectID 来表示选中了谁，当 SelectID 为-1 的时候表示没有选中任何拼块，也可以视为有个虚构的拼块-1。对数据做了一些改变以后要记得重绘图形 Refresh。

*异步处理方法 Refresh 控制重绘的区域，不将所有的绘制工作都放到 OnPaint 里，以提高重绘效率，这在高频率重绘与绘制量大混合的情况下尤为重要，而且实现比较容易。

```
void CMFC 七巧板 Dlg::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO:  在此添加消息处理程序代码和/或调用默认值
    for (int i = 0; i < Shape_Count; i++)
    {
        if (Chip[i].IsInside(point))
        {
            Chip[i].Select();
            SelectID = i;
            SelectPoint = point;
            //防止选中两个
            break;
        }

    }
    Refresh(0);
    CDialogEx::OnLButtonDown(nFlags, point);
}
```

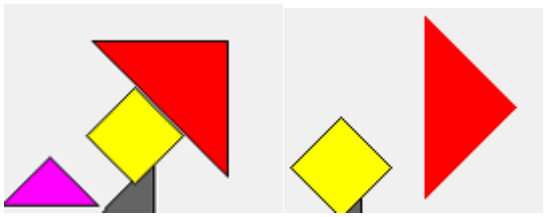
左键放开时要取消 Select，并且重绘。

```
void CMFC七巧板Dlg::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO:  在此添加消息处理程序代码和/或调用默认值
    if (SelectID>=0)
        Chip[SelectID].Select();
    SelectID = -1;
    Refresh(0);
    CDialogEx::OnLButtonUp(nFlags, point);
}
```

在左键按住的同时 ,移动鼠标 ,拼块将跟随鼠标移动。这里要特别注意先擦除再绘制新图形。

```
void CMFC七巧板Dlg::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO:  在此添加消息处理程序代码和/或调用默认值

    if (SelectID >= 0)
    {
        //先擦除
        Chip[SelectID].Draw(GetDC(), 1);
        //移动
        Chip[SelectID].Move(SelectPoint, point);
        //更新位置
        SelectPoint = point;
    }
    //异步重绘
    Refresh(0);
    CDialogEx::OnMouseMove(nFlags, point);
}
```

右键按下时调用旋转方法，这里要注意的是，调用的同时将鼠标位置传入旋转函数，所以拼块以鼠标位置为旋转中心进行旋转。这意味着，你点击拼块的不同位置时，旋转的效果是不同的。



```
void CMFC七巧板Dlg::OnRButtonDown(UINT nFlags, CPoint point)
{
    // TODO:  在此添加消息处理程序代码和/或调用默认值
    if (SelectID >= 0)
    {
        Chip[SelectID].Draw(GetDC(),1);
        Chip[SelectID].Rotate(point,45);
    }
    Refresh(0);
    CDialogEx::OnRButtonDown(nFlags, point);
}
```

键盘响应

键盘相应的部分就很简单了，大体上与鼠标响应没有大区别。区别在于键盘的输入没有给旋转传入旋转中心，这时我们就直接使用几何中心来作为旋转中心了。

说明：Q,E 为不同方向的旋转，W，S，A，D 为四个方向的平移，Z，C 表示选定不同的拼块。

```
void CMFC七巧板Dlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO:  在此添加消息处理程序代码和/或调用默认值
    if (SelectID >= 0)
    {
        CPoint Center = Chip[SelectID].GetCenter();
        switch (nChar)
        {
        case 'E':
            Chip[SelectID].Draw(GetDC(), 1);
            Chip[SelectID].Rotate(Center, 45);
            break;
```

```
        case 'Q':
            Chip[SelectID].Draw(GetDC(), 1);
            Chip[SelectID].Rotate(Center, -45);
            break;
        case 'W':
            Chip[SelectID].Draw(GetDC(), 1);
            Chip[SelectID].Move(Center, Center + CPoint(0, -10));
            break;
        case 'S':
            Chip[SelectID].Draw(GetDC(), 1);
            Chip[SelectID].Move(Center, Center + CPoint(0, 10));
            break;
        case 'A':
            Chip[SelectID].Draw(GetDC(), 1);
            Chip[SelectID].Move(Center, Center + CPoint(-10, 0));
            break;
        case 'D':
            Chip[SelectID].Draw(GetDC(), 1);
            Chip[SelectID].Move(Center, Center + CPoint(10, 0));
            break;
        }
    }
    if (nChar == 'Z')
    {
        if (SelectID >= 0)
            Chip[SelectID].Select();
        SelectID = (SelectID + 1) % Shape_Count - 1;
        Chip[SelectID].Select();
    }
    if (nChar == 'C')
    {
        if (SelectID >= 0)
            Chip[SelectID].Select();
        SelectID = (SelectID - 1 + Shape_Count) % Shape_Count - 1;
        Chip[SelectID].Select();
    }
    Refresh(0);
    CDialogEx::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

*DEBUG：键盘消息&拦截

在对话框上有某些控件时，键盘消息往往会被错误地拦截，交给这些控件，然而这些控

件是没有接收键盘消息的功能的，这是 CWnd 基类的 PreTranslateMessage 这个虚函数需要重写（重载）。PreTanslateMessage 是 MFC 中消息传递机制中的一个分流器，负责将不同的消息分配到不同的控件（容器）。所以如果消息不能正确地被接收，多半是这个函数不正确。

这个函数完全可以取代 KeyDown，也可以将 Msg 传给 KeyDown，这两种解决方案都是可行的。

BOOL CMFC 七巧板 Dlg::PreTranslateMessage(MSG* pMsg)

```
{
    CPoint Center;
    if (SelectID >= 0)
        Center = Chip[SelectID].GetCenter();

    if (pMsg->message == WM_KEYDOWN)
    {
        //判断具体键
        switch (pMsg->wParam)
        {
        case 'E':
            if (SelectID >= 0)
            {
                Chip[SelectID].Draw(GetDC(), 1);
                Chip[SelectID].Rotate(Center, 45);
            }
            break;
        case 'Q':
            if (SelectID >= 0)
            {
                Chip[SelectID].Draw(GetDC(), 1);
                Chip[SelectID].Rotate(Center, -45);
            }
            break;
        case 'W':
            if (SelectID >= 0)
            {
                Chip[SelectID].Draw(GetDC(), 1);
                Chip[SelectID].Move(Center, Center + CPoint(0, -10));
```

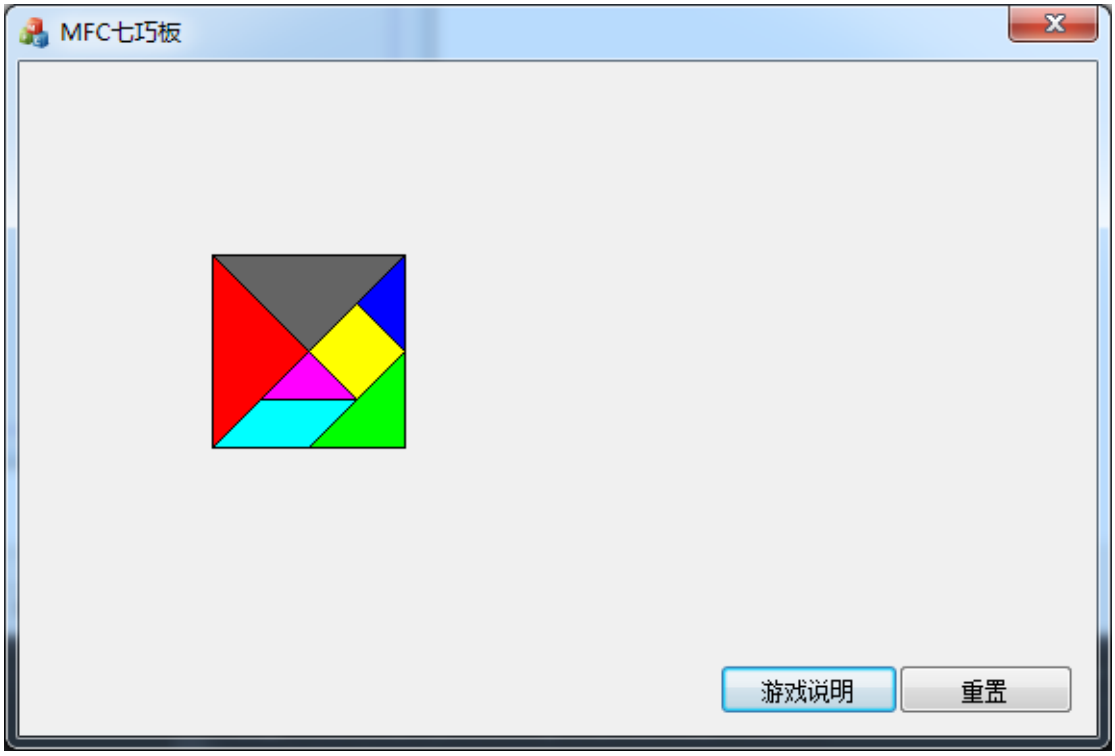
```
            }
            break;
        case 'S':
            if (SelectID >= 0)
            {
                Chip[SelectID].Draw(GetDC(), 1);
                Chip[SelectID].Move(Center, Center + CPoint(0, 10));
            }
            break;
        case 'A':
            if (SelectID >= 0)
            {
                Chip[SelectID].Draw(GetDC(), 1);
                Chip[SelectID].Move(Center, Center + CPoint(-10, 0));
            }
            break;
        case 'D':
            if (SelectID >= 0)
            {
                Chip[SelectID].Draw(GetDC(), 1);
                Chip[SelectID].Move(Center, Center + CPoint(10, 0));
            }
            break;
        case 'Z':
            if (SelectID >= 0)
                Chip[SelectID].Select();
            SelectID = (SelectID + 2) % Shape_Count - 1;
            Chip[SelectID].Select();
            break;
        case 'C':
            if (SelectID >= 0)
                Chip[SelectID].Select();
            SelectID = (SelectID - 1 + Shape_Count) % Shape_Count -
1;
            Chip[SelectID].Select();
            break;
        }
    Refresh(0);
    return CDialog::PreTranslateMessage(pMsg);
}
    return CDialogEx::PreTranslateMessage(pMsg);
}
```

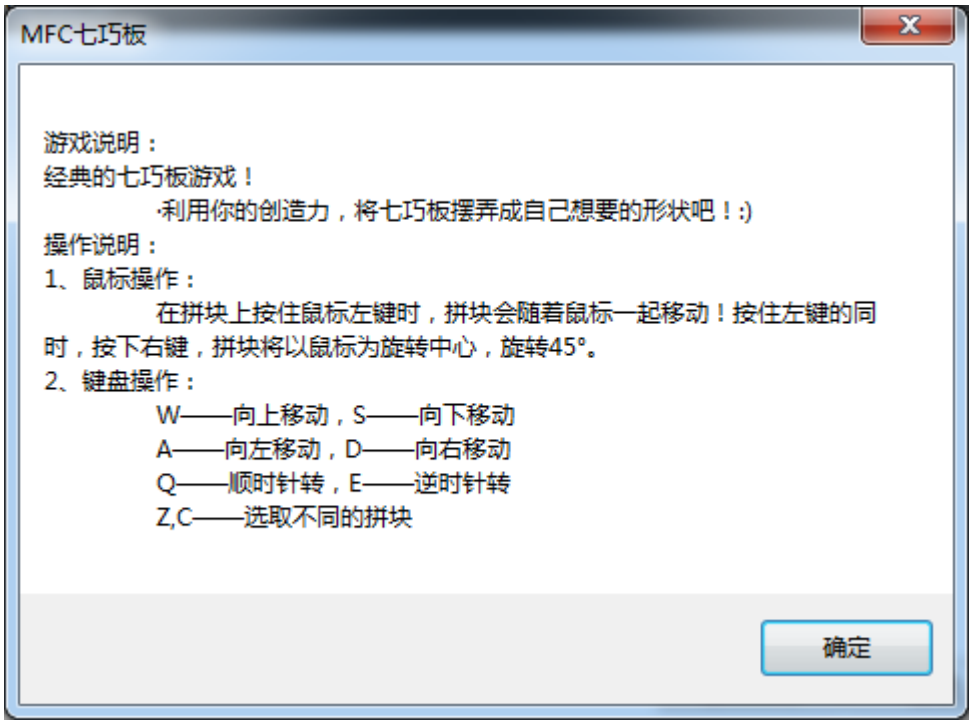
GUI 设计

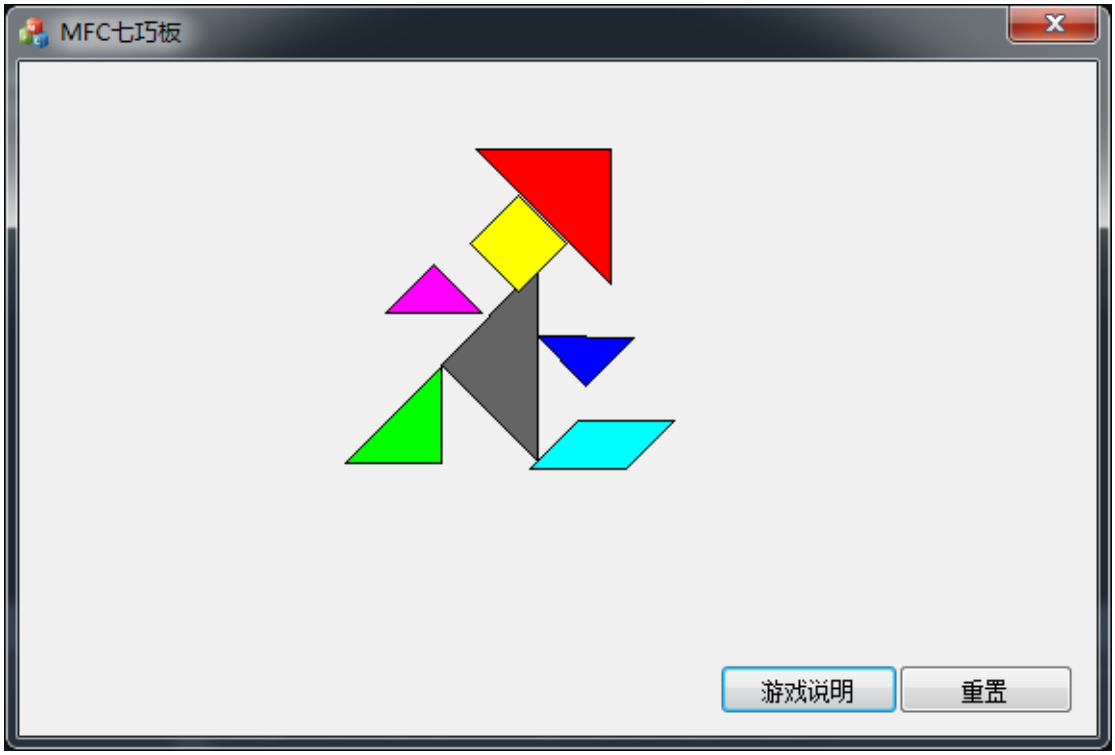
这个与其说是设计，还是说没有设计的好。

但是从简洁的方面考虑，程序界面友好，美观大方，简洁明了，没有一些有的没的多余装饰，

给游戏者最纯净的创造空间。






MFC七巧板
游戏说明
重置

实验总结

陈铮：通过这次实验，我学会了使用 MFC，还加深了对 C++的特性以及各种机制的理解。了解到了隐藏在复杂 C++语言规则之后合理的逻辑性。另外也是很久没有好好写过工程了，完成这些工程之后成就感也是不亚于在 ACM 中通过一道难题的。对实验的题目的想法就是，其实挺好的，只是 MFC 我们平时上课不讲，一切都要靠自己课下学，未免有些迷茫不知所措。希望以后能够开一些课讲授 MFC 的原理，最好是能够穿插在 C++语言学习内讲。另外，我希望实验课中也要引入一些过程式程序设计的精巧案例，一点算法都不要求我觉得也是不好的，这会使我们远远落后于其他高校的学生（如 PKU 计科的一般的大一学生现在已经能熟练使用 DFS 与 BFS 等图论算法了，而我们却一定要拖到大二再讲）。大家的积极性没有被充分调动起来，这是我觉得遗憾之处。

郭家梁：通过这次实验，我不仅加深了对必修课"面向对象的程序设计方法"中一些概念的了解与运用，而且还接触到了 MFC 这个新的平台，并逐渐能够运用它做一些简单的工程。同时，我还掌握了一些设计技巧，让界面更加美观，成果更加用户友好。让理论运用于实际，做出能够真正应用于现实生活的成果，这就是编程人员所追求的吧。

欧阳鹏程：通过这次实验，我们了解了 MFC 的基本知识和操作，很多的东西都是我们自己学习的，走了很多的弯路，有很多能有快捷方法的算法，但是我们却没有学习，也没有太多时间去学习额外的算法，所以我希望类似这样的实验，能建立在我们已经学习了某些算法的基础上再做，以达到巩固所学知识的目的。

陈景琦：

通过这次实验，我从零开始学习了 Qt 框架的用法，和 C++11 中的部分特性—多线程、匿名函数、和 Lamada 表达式。

同时我阅读了设计模式这本经典的书，并看了软件工程这本书的前几章。

这次实验使我的工程能力得到了很大的提高。

致谢词

陈铮：首先感谢小组成员们的辛勤努力，没有"拖后腿"，"吃白饭"的人使我很高兴，大家都在一丝不苟地认真学习，这是大家都愿意看到的。再是感谢网友的热心帮助，Intenet 并不是一个冷漠的圈子，热心网友千千万，我前期学习的很多问题都是问的百度，而这些知识归根结底都是网友的。其次感谢研究生学长学姐的类似的样例源码以及讲义，虽然我并没有抄过任何一份代码，但是关于 MFC 的设计模式还是一定程度上承袭了前辈的。

郭家梁：首先要感谢小组成员在我学习 MFC 过程对我的帮助，我才能圆满成功地完成这次实验的任务，才能提高自己的编程与设计能力。更要感谢梁力老师平常上课对我们 C++基本能力的培养，我们打下了一个良好的基础，才能更加深入地学习和应用新的知识。

欧阳鹏程：通过这次实验，我们学会了如何使用 MFC 编写一些简单的可视化的小程序，在交互界面上完成一些简单的操作。当然，要感谢此次实验的指导老师梁力老师，以及热心帮助我的同学们，有了他们的帮助，我才会更快更好的学会 MFC 的基本知识和操作，才让我这次的实验有了一个完美的结束。

陈景琦：感谢所有为了软件工程这门学科做出贡献的科学家、工程师。

感谢所有为了 GPL Foundation 付出过的程序员、设计师，正是你们的付出让更多的人用

到了更好的自由软件。

感谢所有帮助过我的老师、同学。

参考文献

《C++编程思想》 [M] (美)埃克尔等著；刘宗田等译.—北京：机械工业出版社，2011.7