# 操作系统实验日志

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 | 201808010718 | 姓名 | 肖鹏 | 专业年级班级 | 智能1802 |
| 实验日期 | 2020.12.8 | 实验项目 | 第11天：制作窗口 | | |

## 一、实验主要内容

### 1、鼠标优化

**内容：**修复之前鼠标不能够实现右下边界上的“隐藏”功能；

**重点总结：**主要就是对两个地方进行修改：

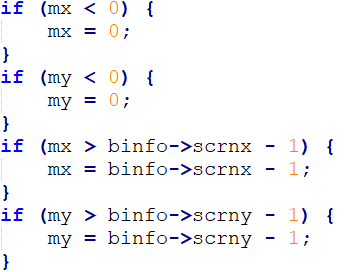
1）鼠标移动范围，让它可以移动到边界处；

2）refresh更新的范围，禁止更新vram范围之外的显示，否则会造成鼠标的“溢出现象”。

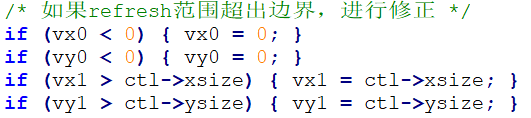


*（右边界鼠标“溢出”至左边界）*

**关键代码及注释：**



*（修改鼠标的可行域）*

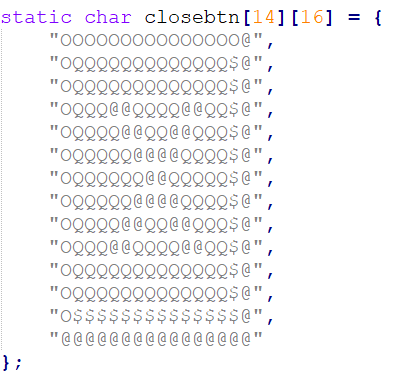


*（修正refresh范围）*

### 2、绘制窗口

**内容：**

a）像绘制鼠标一样，首先我们为窗口的细节处——关闭键设计一个显示模板，因为个“x”按钮并不能够简单的通过矩形绘制来实现。

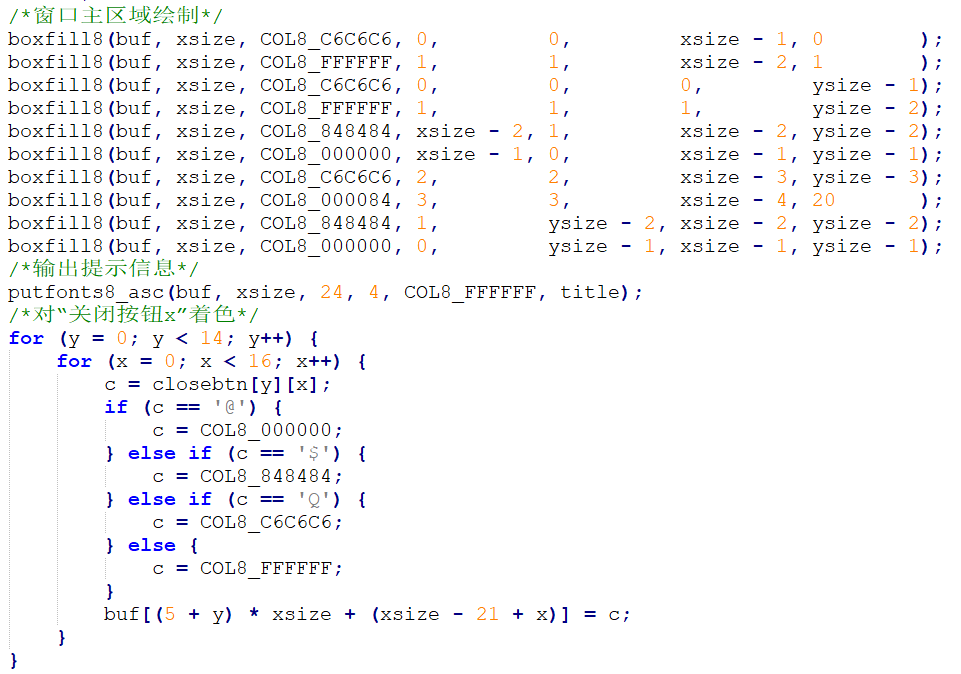


*（窗口界面“关闭按钮”的模板）*

b）对于窗口的主体部分，通过简单的矩形组合即可得到，类似于当初绘制任务栏是所用到的知识，没有涉及新内容。

**重点总结：**无

**关键代码及注释：**



*（绘制窗口）*

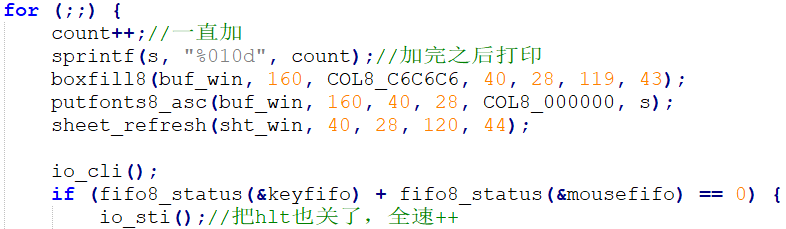


*（bootpack.c中为实现窗口做的工作）*

### 3、高速计数器修复闪烁

**内容：**

a）实现一个高速计数器，看起来很高级，我一开始以为通过定时器来实现的，其实就是在 for循环里一直对count变量++。



*（计数器实现）*

可能这个不是重点，主要是想让高刷新率的图层来测试我们现在窗口显示的bug，这不， 果然有bug！



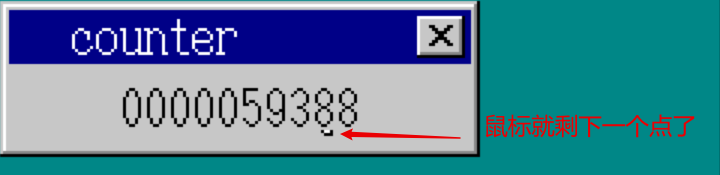
*（bug1）*

这是因为，我们使用的refresh函数，是从最底层（也就是桌面）开始重绘，这样一层一层 地来达到刷新的效果。这样的话，由于总要花费时间，所以这样一个在绘制完“桌面”和 绘制“窗口”之前的一个间隙，就是bug1中的效果了。

b）解决方法1：只对变化的图层、以及该图层之上的图层进行refresh；

如何实现呢？需要向现有的refreshsub函数中额外传入一个图层高度参数h0，表示我们 只更区域[h0,∞)的图层。具体见重点总结a）。

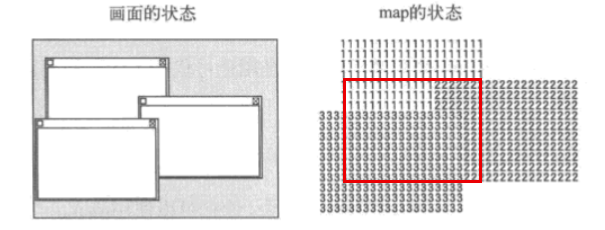
但是这种方法并没有很好的解决，因为这里还是有延迟问题，由于窗口在实时变化，所以 每次的refresh都是先刷新窗口、然后再刷新鼠标，这样就导致一个时间差，下面这个bug2 图能够很好的说明这个问题：



*（bug2）*

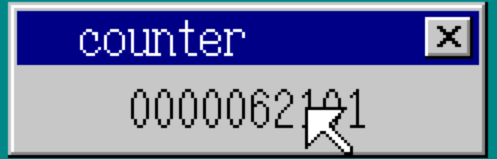
c）解决方法2：

之前的方法是因为我们的刷新方式是以图层为单位，这样如论如何优化， 在涉及到多 图层的更新时，总是会出现这样闪烁的情况，这是无可避免的，因为更新一个 图层肯 定需要时间。现在我们换一个思路，我们不再以图层为单位整体更新，而是通过一 个与 vram等大小的map数组，来记录显示器上每一个像素点对应的图层号，这样在更新 的时 候以map为依据，对于重叠的像素点，只更新位于最上方的，而并不是在每一个低图 层中都事先更新一次。



*（map原理示意）*

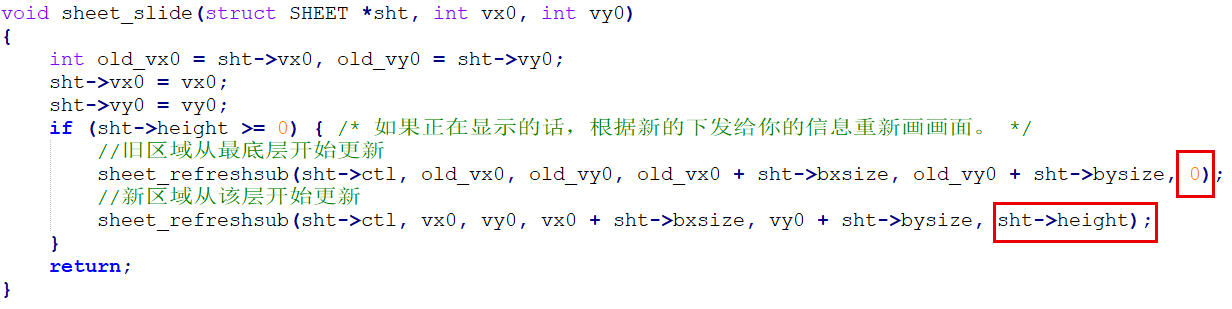
比如对于图中的这一重叠部分，按照之前方法1，我们会把三个图层的所有像素点都更 新；但是现在我们更具map，那么在更新图层1的时候，就只会更新map中标记为1的 像素点，对于重叠区域不做修改，等到这些更新这些区域对应的图层时再更新，这样就 不会出现底层的数字先更新、导致上层鼠标被覆盖、再更新上层鼠标这样一个“闪烁” 的现象。



*（perfect效果）*

**重点总结及代码注释：**

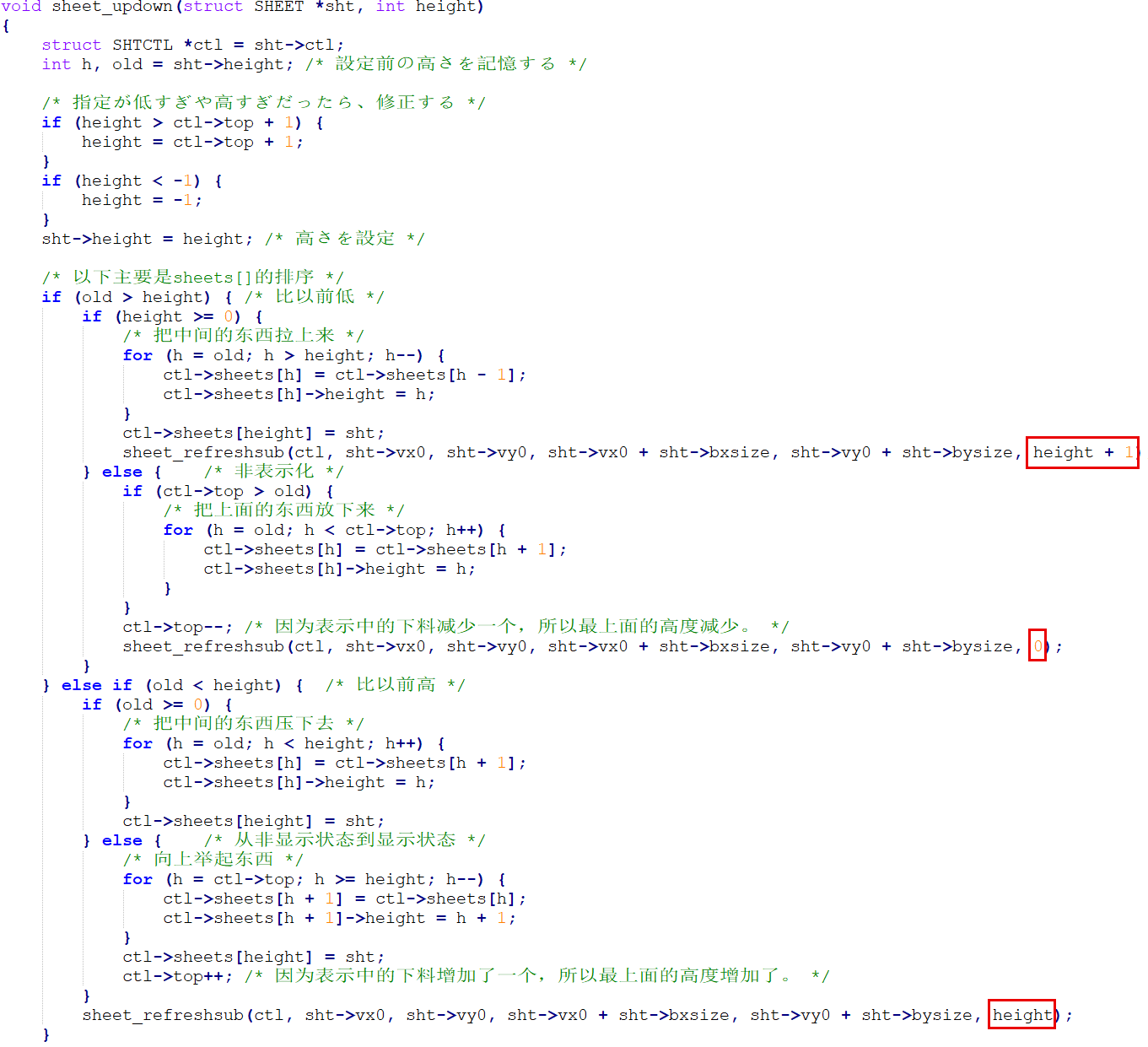
a）对于解决方法1中的实现思路有以下两个要点：

1）对于slide函数，即窗口在图层内上下左右移动时，会导致该图层以下的图层的信 息会重新显露出来，还可能导致某些中间图层的信息被覆盖。这就需要我们分别对于 两个区域：**①移动前的old区域**，从最底层0开始refresh，这样让显露出来的信息得 以显示；**②移动后的old区域**，由于比当前高度低的图层将被覆盖，所以我们对于比 当前高度低的图层也不需要更新（就算更新了也会被后面更新的高图层覆盖，效果一 样），所以我们只要更新[h0,∞)的图层即可。 

*（方法1中slide函数的修改）*

2）对于updown函数，由于我们可能上移/下移，所以有以下两种情况：**①若是下移**， 则说明之前处于低处的图层可能获得显示的机会，所以我们需要从该图层下降到的高 度+1开始，更新上层的所有图层[h0+1,∞)。至于为什么不更新h0本身这一层呢？思 考一下，我们这里默认是在图层下降到h0之后，其上方是还有图层的，所以默认h0 图层会被覆盖，没有必要更新。但是万一这个图层本身就是唯一的图层呢，也就是h0 仍然是全局最高的，没有图层位于h0+1，那么循环也不会成立，最终也不会有画面覆 盖该图层，所以这个写法是正确且高效的。特殊的，如果下移到“隐藏”，我们便从0 开始更新；**②若是上移，**则说明上移到h0后，高度小于h0的都会被覆盖，我们对于 被覆盖的内容均不更新，只对[h0,∞)范围的图层进行更新。

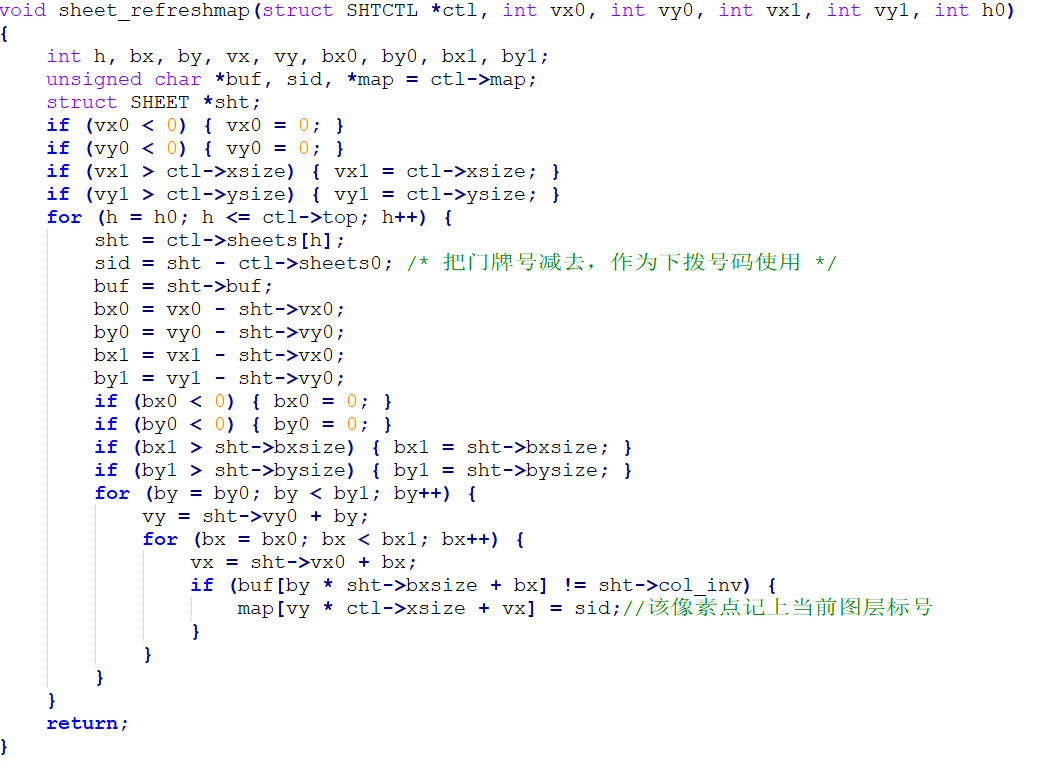
这里为什么又包含了h0呢？由于这里是上移操作，所以实际上是将之前“被隐藏”的 一个图层给他一个可能显示的机会，这个机会存不存在取决于h0上面还有没有更高的 图层，由于我们没有事先判断h0和top的关系，所以我们先更新h0层，万一h0之上 还有图层，那么在调用refresh函数时也会将其覆盖；若是没有，则显示的最顶层便是 我们现在的h0。



*（方法1中updown函数的修改）*

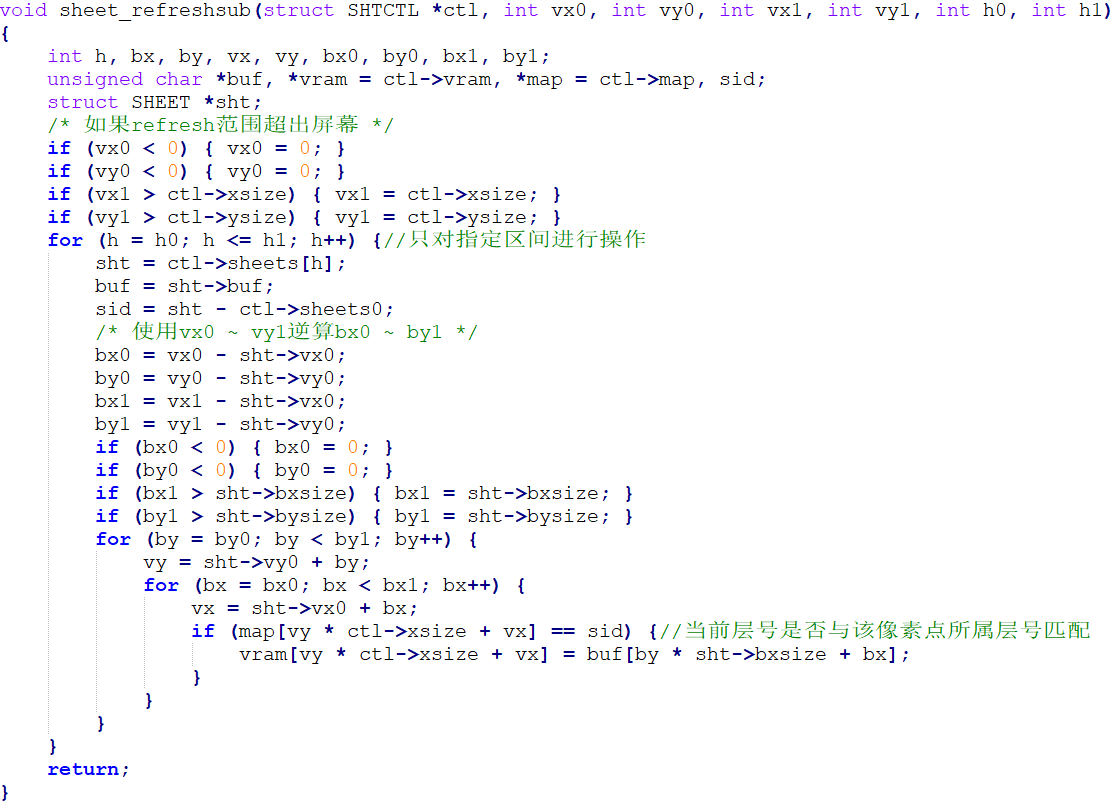
b）对于解决方法2中涉及到的新函数：

1）refreshmap：跟新map，原理就是从传参h0的图层开始，修改map中对应像素 点的图层标号，通过图层从低往高的遍历、覆盖，重叠区域最终将以该区域内最高图 层的标号作为标记，这符合我们图层显示的原理——只显示最高层、低层被覆盖，这 样便实现了更新map。



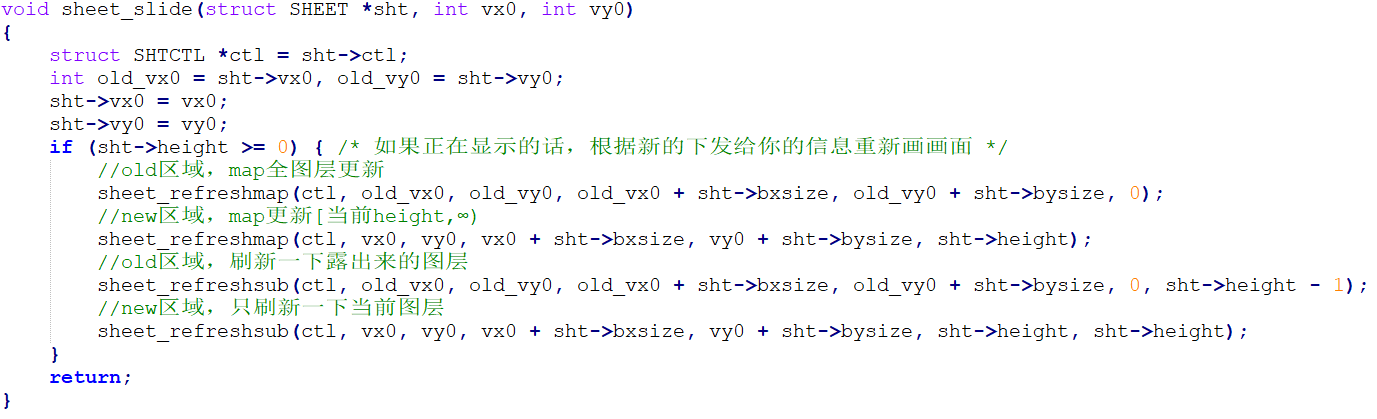
*（方法2中refreshmap函数实现）*

2）以map为更新依据的refreshsub：现在更新将按照map来实现，所以传参也有一 点变动，现在可以指定一个区间[h0,h1]进行更新（之前是一个起始点），这是因为我们 现在可以参照map，比如现在这个例子，有了这个方法之后，其实一直在变化的（鼠 标不动的时候）就是图层1的计数窗口，我们可以只更新图层1就能够实现画面的更 新（即refresh函数），而不需要额外更新鼠标层。所以，现在单个图层发生变化后， 只需要更新这一个图层就可以了。可以理解成把这些多余的更新放到了对map的维护 中。



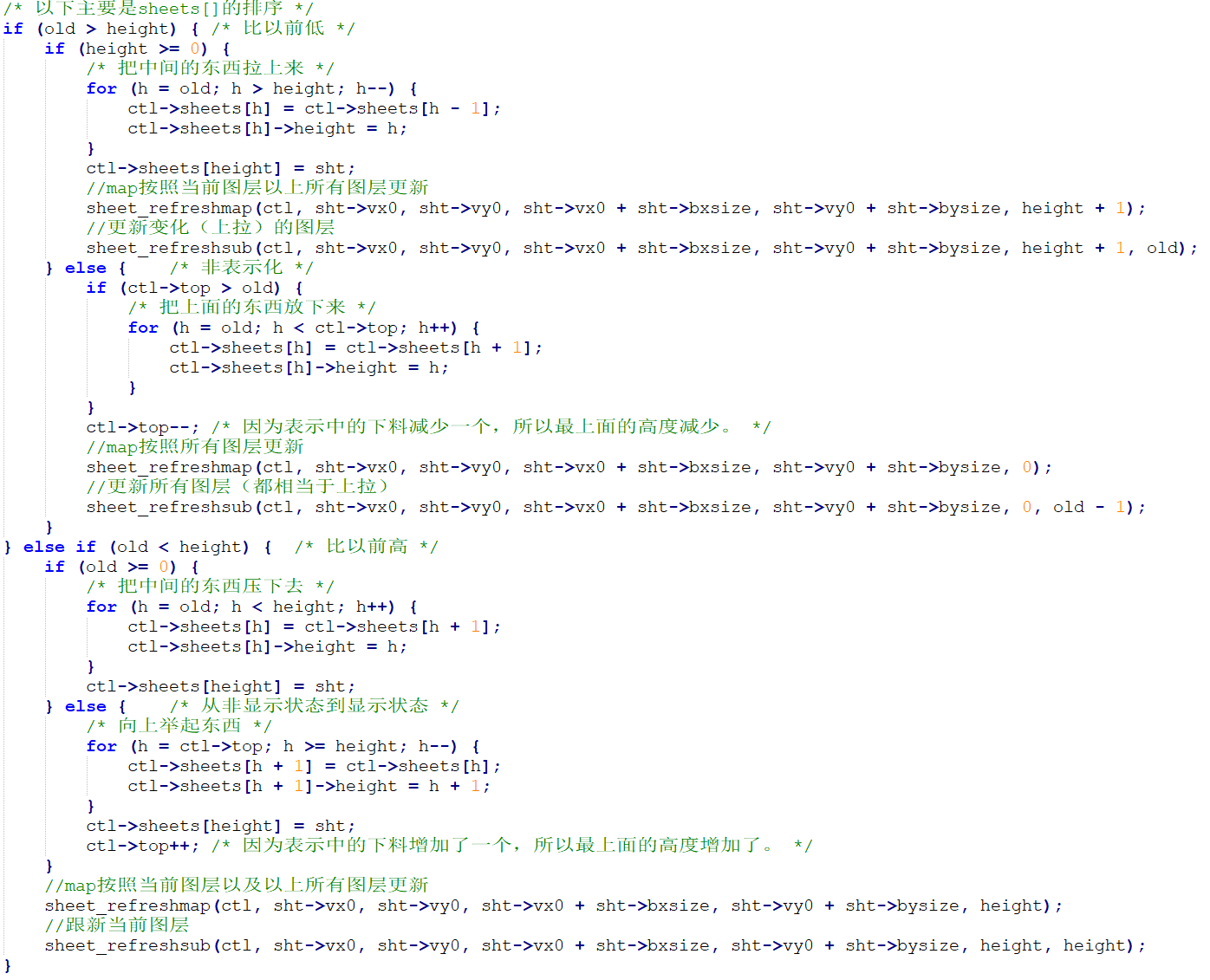
*（方法2中根据map修改的refreshsub函数）*

3）slide：在方法1中用到的策略仍然适用，只不过并没有之间跟新到屏幕上，而是先 更新map，再对old区域、new区域分别更新绘制。对old区域由于所有低层都有可能 出现，所以调用refreshsub的时候需要[0,height-1]，这里由于新的refreshsub可以指定 区间，所以我们限定最多更新到当前height-1的地方；对于new区域，只需要更新变 化了的height这一层即可。



*（方法2中根据map修改的slide函数）*

4）updown：map的更新类似于方法1中的策略；而对于refreshsub更新的区间设定， 分为：**①若是下移**，则需要把暴露出来的区间[height+1,old]都更新。特殊的如果下移到 非显示区，则需要把所有图层都更新；**②若是上移**，则只需要更新该层。



*（方法2中依据map修改的updown函数）*

5）总结一下方法2的一个主要思想：

“刷新=更新map+重绘”

①调用refreshmap的时候，使用方法1中的策略选择起始层；

②调用refreshsub时候，只对涉及到的区间进行重绘，不做多余的重绘。这个“涉 及到的区间”如果只是单个图层内容变化，或者是单个图层上移，那么这个区间只 包含单个图层；如果是图层的下移，那么就需要把暴露出来的图层都重绘（相当于 这些图层都做了一个上移操作，所以每个图层都重绘一下）。

## 二、遇到的问题及解决方法

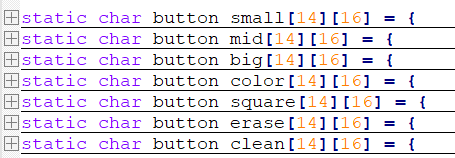
无

## 三、程序设计创新点

### 1、节点考核优化

**①实现功能切换的可视化，绘制了切换功能按键**

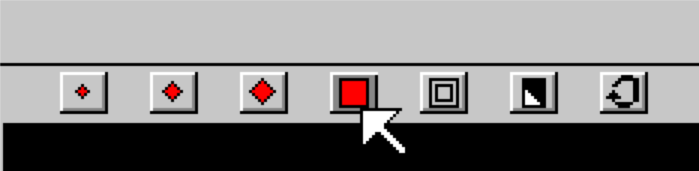
**代码实现：**

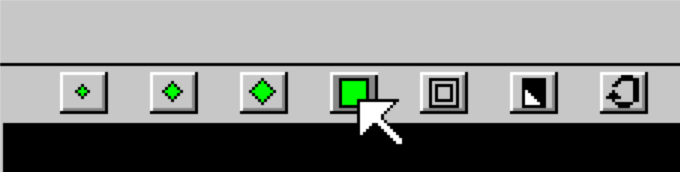


*（各个按钮的像素图）*

其中，对于前三个按钮和颜色按钮，在颜色发生变化时都能够实时更新图标信息；而且在鼠标放在按钮上时，对于所有的按钮都进行背景色加深处理。

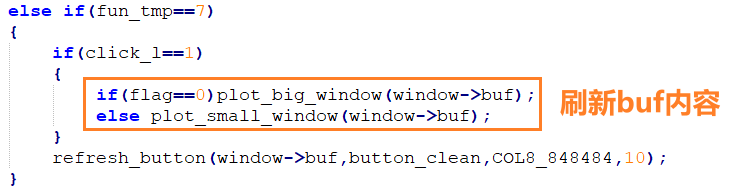
**效果图：**





**②增加“一键清屏”功能**

**代码实现：**



**效果图：**



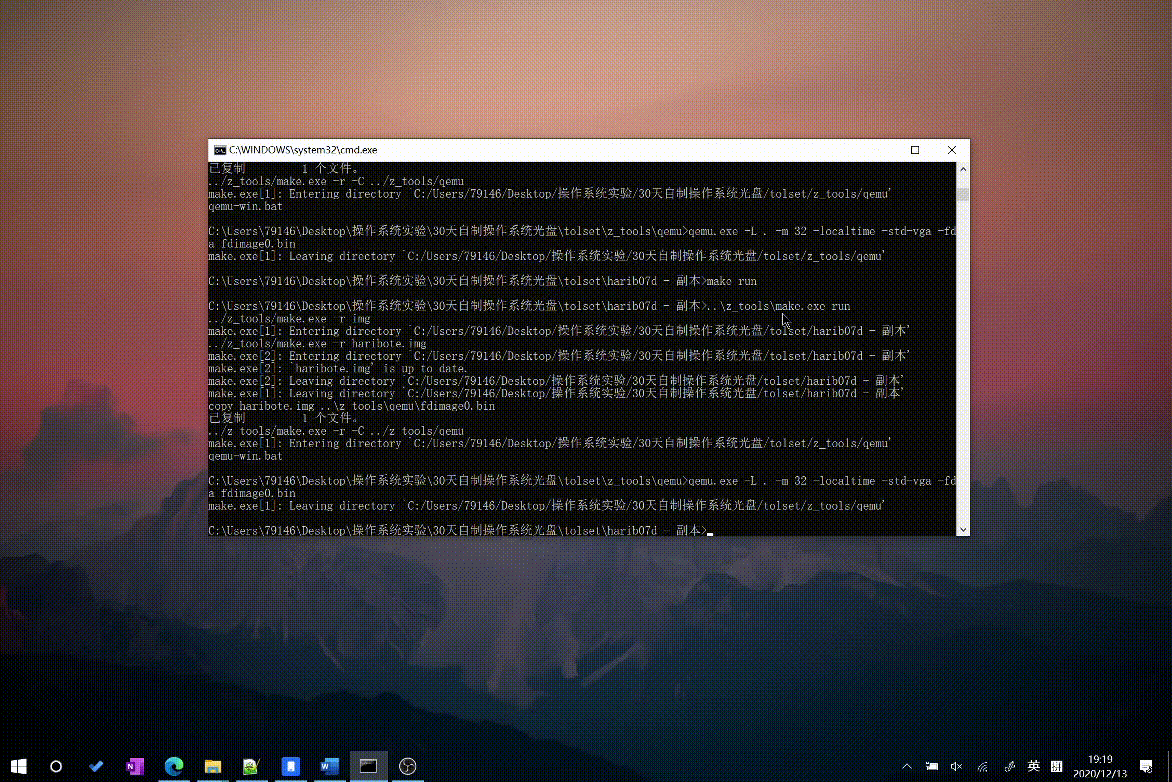


**③实现在四个象限中矩形的动态拖拽以及矩形的回拉**

**代码实现：**



**效果图：**



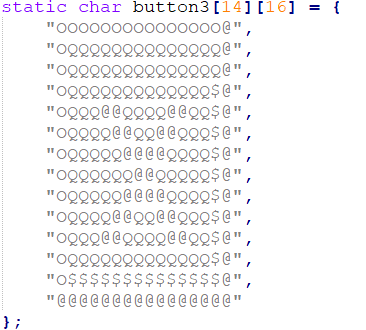
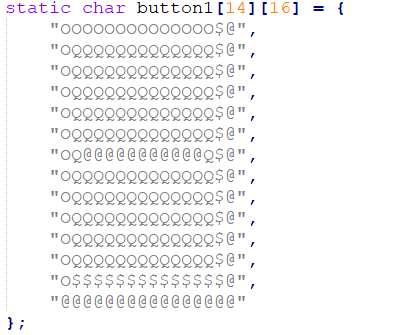
### 2、完善Day 10设计的窗口——增加按钮图标以及动态变化

**创新点：**

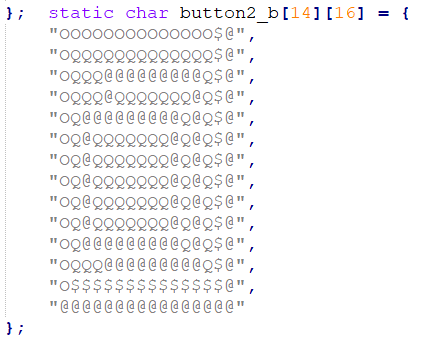
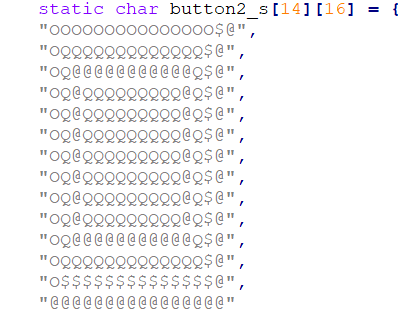
由于之前实现的窗口不是很美观，现在结合今天所学，对之前的窗口按钮进行优化，主要是对三个按钮的图案进行绘制；并且判断当鼠标停留在按钮上时，该按钮的背景色变深色。

**代码实现：**

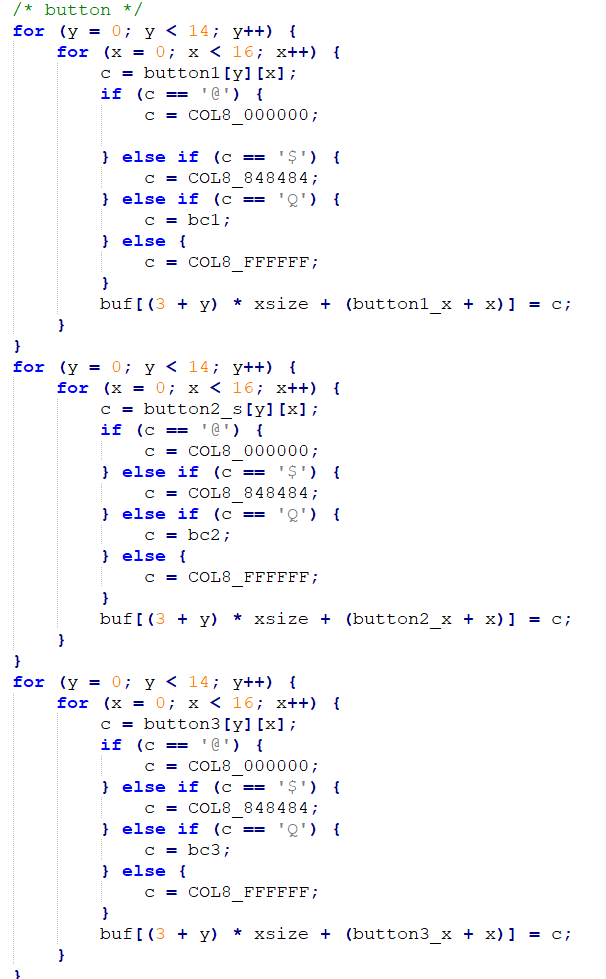
按钮图案设计：



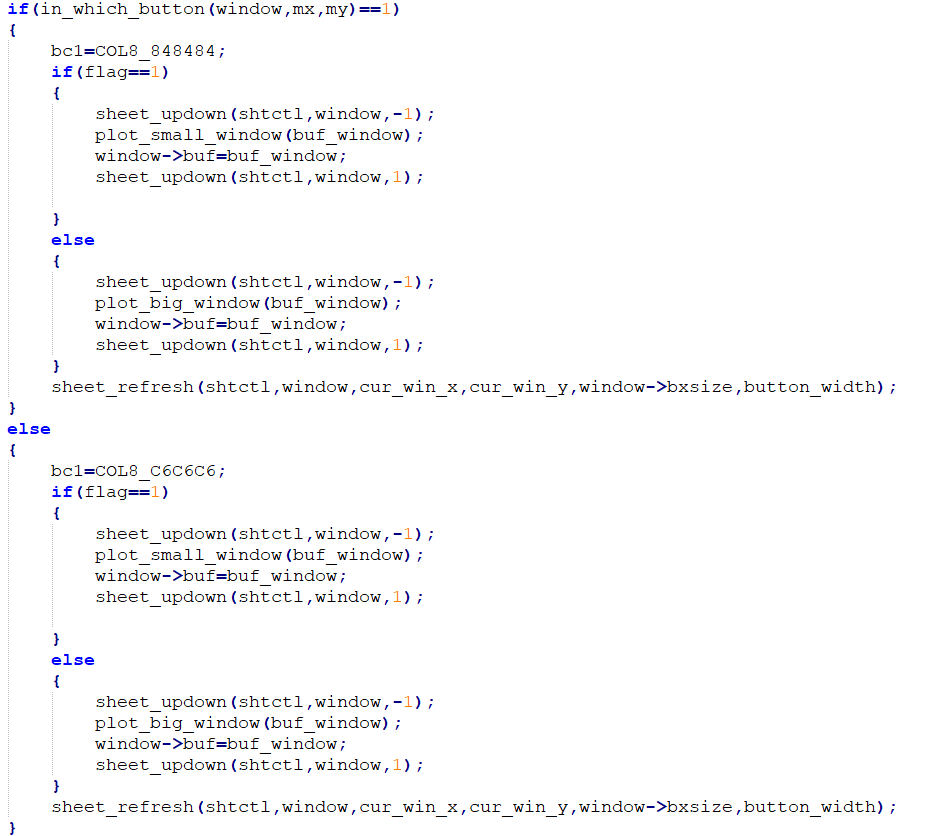
*（拖动按钮） （关闭按钮）*



*（小屏幕时放大按钮） （大屏幕时缩小按钮）*



*（绘制鼠标图案）*



*（判断鼠标停留、更改背景色）*

**效果截图：**



*（小窗口）*



*（全屏）*

## 四、实验心得体会

这次实验对之前的窗口实现有了更进一步的改进，使用新的方法——map，对于图层的更新能够有更好的效果，消除了闪烁的现象。完善了窗口的实现，在之前的缩小、放大、拖动、关闭之上再实现了外观上的一些细节，总的来说，我们的窗口应该是越来越完善了！