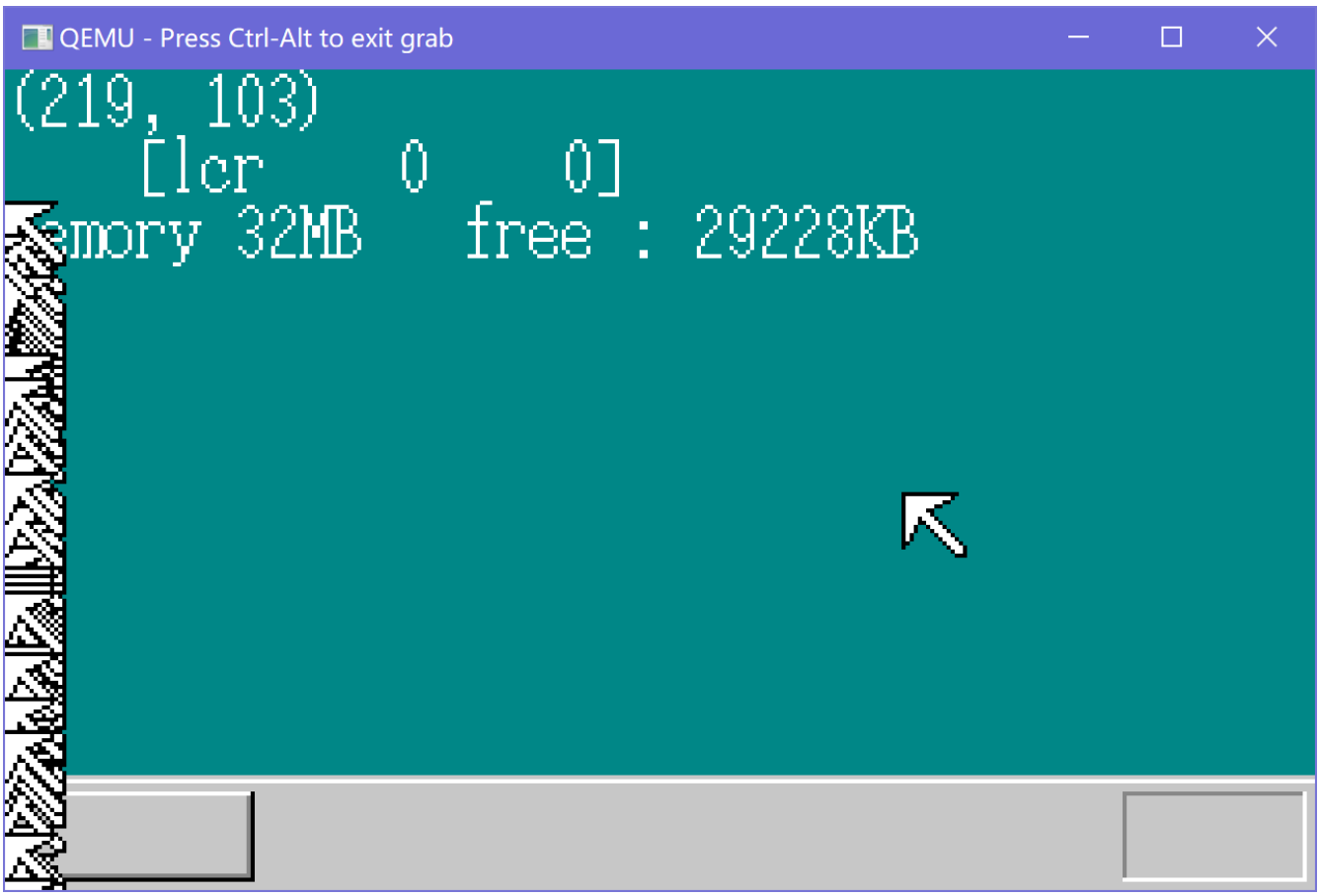
Day 11

Phase 1

SubPhase 1

激动！作者终于要处理鼠标不能移到屏幕外面的这个feature了

现阶段的代码如果硬改的话，是会产生以下我们不希望看到的效果的。



我们来修改以下图层绘制函数吧

```
void sheet_refreshsub(struct SHTCTL *ctl, int vx0, int vy0, int vx1, int vy1)
{
    int h, bx, by, vx, vy, bx0, by0, bx1, by1;
    unsigned char *buf, c, *vram = ctl->vram;
    struct SHEET *sht;
    if (vx0 < 0) { vx0 = 0; }
    if (vy0 < 0) { vy0 = 0; }
    if (vx1 > ctl->xsize) { vx1 = ctl->xsize; }
    if (vy1 > ctl->ysize) { vy1 = ctl->ysize; }
    for (h = 0; h <= ctl->top; h++) {
        sht = ctl->sheets[h];
        buf = sht->buf;
```

```
        bx0 = vx0 - sht->vx0;
        by0 = vy0 - sht->vy0;
        bx1 = vx1 - sht->vx0;
        by1 = vy1 - sht->vy0;
        if (bx0 < 0) { bx0 = 0; }
        if (by0 < 0) { by0 = 0; }
        if (bx1 > sht->bxsize) { bx1 = sht->bxsize; }
        if (by1 > sht->bysize) { by1 = sht->bysize; }
        for (by = by0; by < by1; by++) {
            vy = sht->vy0 + by;
            for (bx = bx0; bx < bx1; bx++) {
                vx = sht->vx0 + bx;
                c = buf[by * sht->bxsize + bx];
                if (c != sht->col_inv) {
                    vram[vy * ctl->xsize + vx] = c;
                }
            }
        }
    }
    return;
}
```

SubPhase 2

优化代码。

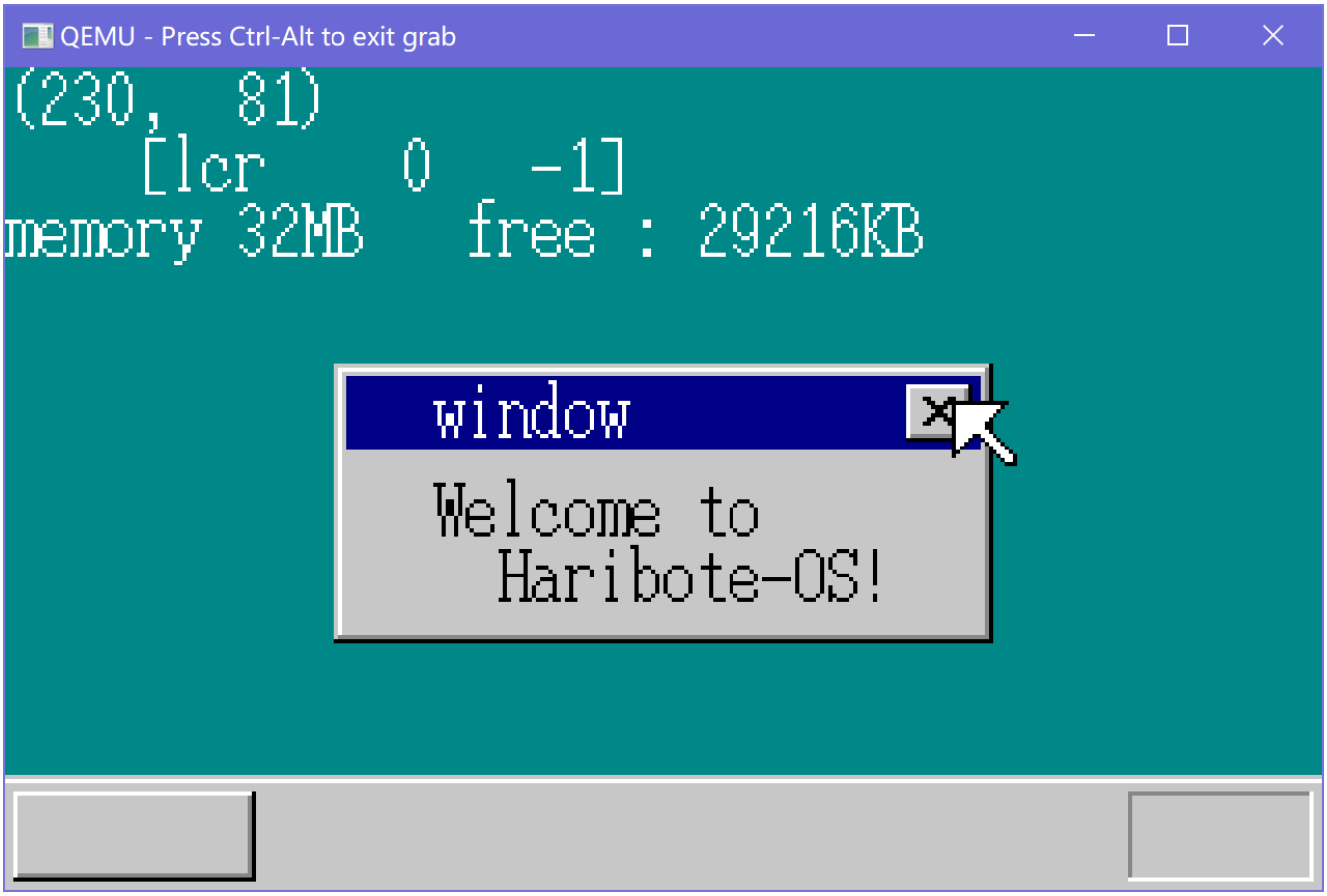
之前*SHTCTL都是作为参数传进去的，优势后我们明明有了SHEET却还要准备*SHTCTL，很不方便，我们来优化一下代码结构，将*SHTCTL纳入到SHEET当中，这样就可以少写一些代码了。

Phase 2

SubPhase 1

窗口

做法是新建一个sheet，然后在上面打印一个看起来像窗口的图像就可以了。



SubPhase 2

高速计数器

如书上所言，窗体更新的部分底色不停的变，在闪烁。这是因为许多时间被浪费在了绘制一定会被覆盖的像素上。我们可以对sheet_refresh稍加修改，只让他绘制更新的图层和更新的图层以上的图层。

但我们发现，仅仅使用这种解决方案是不够的。当鼠标移到上面的时候还是会闪动。这是因为我们多次直接在vram上修改，我们的修改并不一定是我们最后想要显示的结果，所以如果碰见了显示器的某次刷新，那么显示设备也会忠实的把这些中间过程显示出来。

我们可以稍微绕一下，先用一个临时的空间进行绘制，然后再把绘制的结果送到真正的显存当中去，这样就可以避免绘制过程的中间值对显示效果所造成的影响了。

以上是我的解决方法。

作者的解决方法是，对于要重绘的区域，新建一个map数组，记录每一个像素最终对应的是哪一个图层，最后再一起更新，到各个图层中的对应位置找这个像素到底应该是什么颜色。

作者的方法比我的更高效，因为如果各个图层间的覆盖关系没有改变的话，重绘甚至不需要便利整个图层，只需要找对应图层的对应像素点就可以了。

```
void sheet_refreshsub(struct SHTCTL *ctl, int vx0, int vy0, int vx1, int vy1, int h0, int
h1) {
    int h, bx, by, vx, vy, bx0, by0, bx1, by1;
    unsigned char *buf, *vram = ctl->vram, *map = ctl->map, sid;
    struct SHEET *sht;
    if (vx0 < 0)
        vx0 = 0;
    if (vy0 < 0)
        vy0 = 0;
    if (vx1 > ctl->xsize)
        vx1 = ctl->xsize;
    if (vy1 > ctl->ysize)
        vy1 = ctl->ysize;
    for (h = h0; h <= h1; h++) {
        sht = ctl->sheets[h];
        buf = sht->buf;
        sid = sht - ctl->sheets0;
        bx0 = vx0 - sht->vx0;
        by0 = vy0 - sht->vy0;
        bx1 = vx1 - sht->vx0;
        by1 = vy1 - sht->vy0;
        if (bx0 < 0)
            bx0 = 0;
        if (by0 < 0)
            by0 = 0;
        if (bx1 > sht->bxsize)
            bx1 = sht->bxsize;
        if (by1 > sht->bysize)
            by1 = sht->bysize;
        for (by = by0; by < by1; by++) {
            vy = sht->vy0 + by;
            for (bx = bx0; bx < bx1; bx++) {
                vx = sht->vx0 + bx;
                if (map[vy * ctl->xsize + vx] == sid) {
                    vram[vy * ctl->xsize + vx] = buf[by * sht->bxsize + bx];
                }
            }
        }
    }
    return;
}
```