# 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 | 2019.11.5 | 实验项目 | 第7天：FIFO与鼠标控制 | | |

## 一、实验主要内容

**1、中断返回、加速以及数据获取**

**内容：**

a）从中断中返回，一次中断触发之后，便要在中断处理函数中告诉CPU已经处理完了这次中断，需要及时返回并且继续等待下一次中断的到来；这样才能够持续接受外设信息；

b）中断处理需要在短时间内完成，否则将造成卡顿的现象；对于中断处理中的打印字符的操作，实际上需要花费很多时间；一个字符的规模是8\*16的像素矩阵，打印一个字符需要执行128次if语句，还有其他的地址计算，这都是很耗费时间的，而在这一段时间中，有可能已经错过了很多次设备的中断，造成了数据的丢失；所以优化办法是在中断处理函数中只保存接收到的数据字节，然后返回到用户代码中执行相关的打印处理；

c）对于键盘设备，每次的数据是从端口0x0060获得的，一次获得8-bits；对于鼠标也是同一个端口，因为鼠标控制电路包含在键盘控制电路中，但是鼠标会先后发送3个字节的数据；

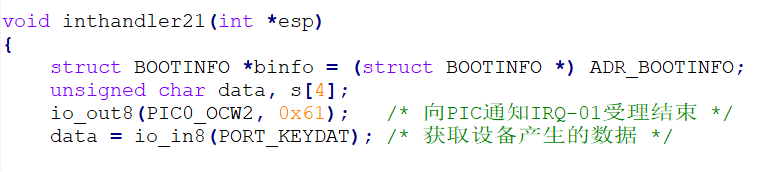
**重点总结：**

a）在对键盘的中断返回说明中，只需要告诉主PIC键盘受理已完成；但是对于鼠标，需要有两句说明，先是从PIC告诉主PIC，IRQ-12的受理已经完成，然后再通知主PIC IRQ2受理也完成，这样才能让PIC持续关注鼠标的中断；

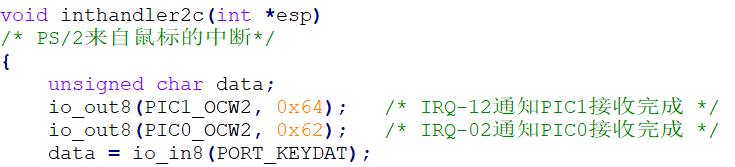
b）想要把中断函数做的事情放到用户代码中，就先要把接收到的数据用结构体存起来；

结构体能够缓存的数据是有限的，通过判断是否还有空间来选择是否接收目前信息；

**关键代码及注释：**



*（键盘中断程序）*

 *（鼠标中断程序）*

**2、制作FIFO缓冲区**

**内容：**

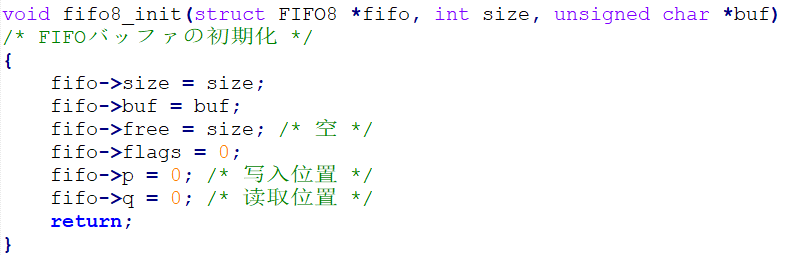
a）设计缓冲区数据结构，最开始是用一个字符来作为缓冲区的，这样的缓冲能力也非常小，当数据来的很频繁的时候容易造成数据丢失；所以需要用有一定容量的数据结构来存储，这里对键盘设定的是32-bytes来缓存数据、而对于鼠标，由于一次中断会产生3-bytes的数据，所以设定一个128-bytes的缓冲区；

b）对于缓存区中的数据，我们总是逐个储存/取出，在储存的时候，只要依次往后面的地址存放，但在取出的时候，每取一个都会造成一个空洞，这样就需要把所有剩余的数据往前挪动；这种挪动的操作很费时间，所以为了避免这种繁琐的操作，我们通过维护两个读、写指针来代替这种操作；除此之外，再增加一些能够表示缓冲区状态信息的变量在结构体中，使得判断时更便捷；  
c）在使用缓冲区时，需要进行边界检测，如果缓冲区已满，则即使再有数据需要存入，那么也只能被丢弃；当缓冲区为空时，想要读取数据也是不行的；

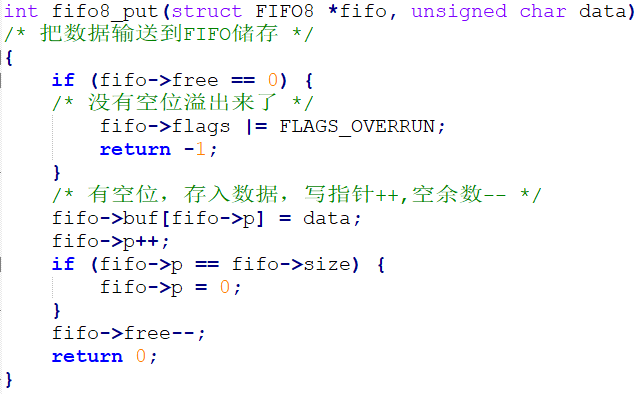
**重点总结：**

a）FIFO缓冲区借鉴了队列的思想，先进入的数据先被处理，通过维护两个读写指针的方式避免了大量重复的数据平移操作；但是由于缓冲区大小有限，所以需要利用循环数组的思想，当指针达到最大值时重新置为0；

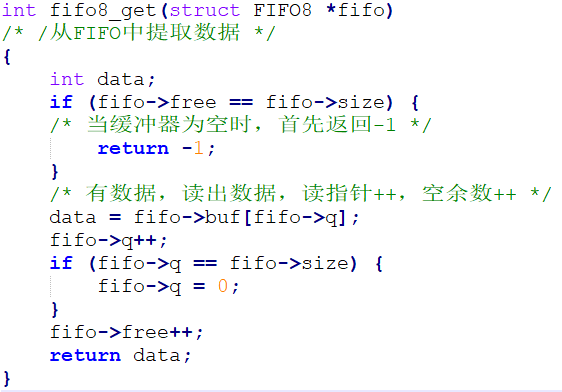
**关键代码及注释：**



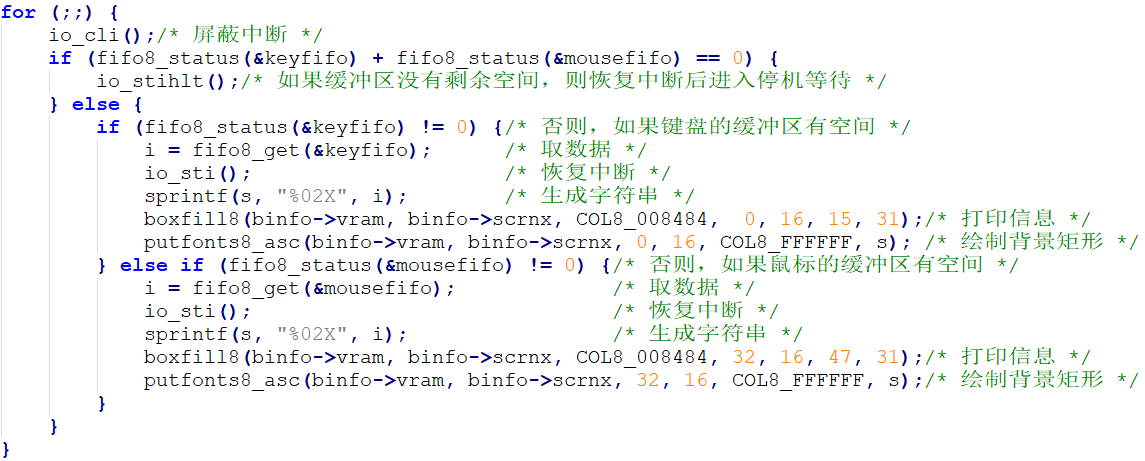
*（缓冲区初始化）*



*（向缓冲区写数据）*



*（从缓冲区读数据）*



*（Hairmain中的中断处理部分）*

**3、鼠标中断初期**

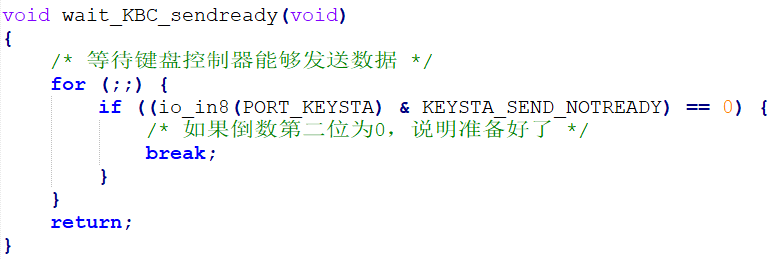
**内容&重点总结：**

a）先让键盘控制电路做好准备工作，等待指令的到了；因为外设的工作速度很慢，如果CPU不顾设备的接受能力一昧的发送指令，会使得指令不被执行；

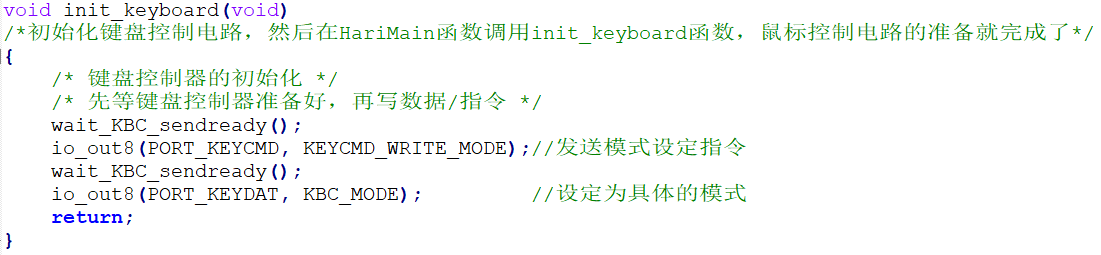
b）在对键盘控制器（鼠标控制电路在键盘控制电路内）发送指令的时候，每发送一条指令之前都需要先执行等待，以确保此时的键盘控制电路是准备好接收指令的状态；

c）激活鼠标电路时，先向键盘控制电路发送一个0xd4指令，这表示这条指令之后的数据都是发送给鼠标控制电路的，鼠标在收到激活指令之后会发送一个0xfa的数据表示已经准备好工作状态，将开始不停的发送鼠标信息；

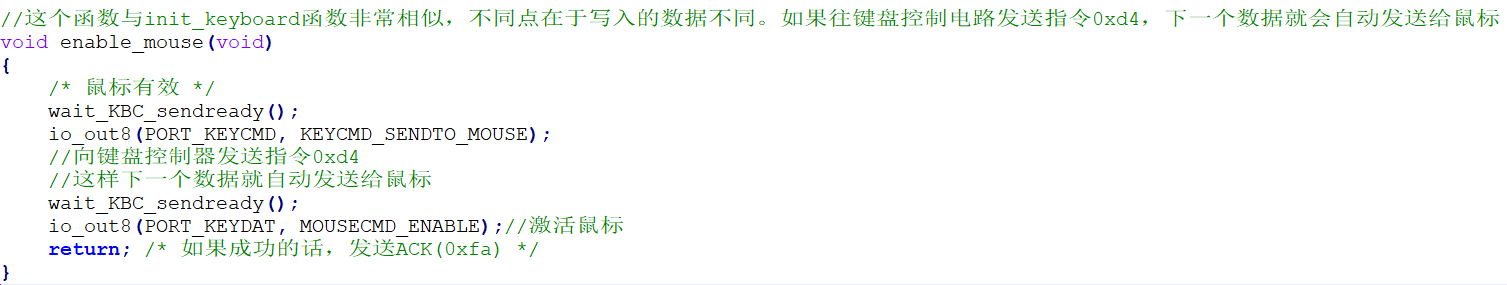
**关键代码及注释：**



*（wait\_KBC\_sendready函数）*



*（init\_keyboard函数）*



*（enable\_mouse函数）*

## 二、遇到的问题及解决方法

1、在实现创新点的时候遇到的编程问题

**问题描述：**

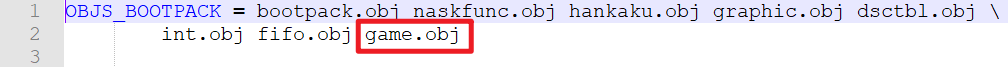
由于自己写了一个.c文件，需要把它和所有其他的.c文件一起编译生成.obj对象然后链接在一起，这里需要对Makefile进行内容增加；由于make run指令不能自动的把文件夹中所有的.obj文件都链接到一起去，所以需要在.obj文件列表中增加game.obj；

**出现原因：**

对于Makefile中的指令了解不足，误以为会自动的把所有的.c文件编译成.obj文件然后链接在一起，其实是需要自己进行修改的。

**解决方法：**

在文件头部的OBJS\_BOOTPACK增加一项



## 三、程序设计创新点

**1、小游戏——键盘控制方块移动、变化大小**

**创新点：**

由于目前我们已经可以自如地使用键盘中断，而且对于每一次触发中断的按键我们都能够获得按键码，所以这就让我们可以很方便的使用这些按键码制作一个简单的“游戏”；

首先我们通过测试得到8个方向按键被按下时的按键码：

W/↑：0x48/0x11

S/↓：0x50/0x1f

A/←：0x4b/0x1e

D/→：0x4d/0x20

然后我们并不直接修改原有的中断处理函数，而是通过对按键”G”的按下与否，来**开启/关闭这个游戏模式**，所以需要知道”G”的按键码为0x22；

最后还设置了通过”回车---0x1c”、”退格---0x0e”分别**控制方块的增大、缩小**；

通过按键”C---0x2e”来控制**方格颜色的变换**；

在操作部分，对方格的移动做了边界检测，即只能在320\*200的像素区间移动；

为了实现这样一个可操作的功能，需要对源文件做一下增改：

a）增加game.c文件，里面实现需要用到的函数；

b）在bootpack.h中增加game.c文件中涉及到的数据结构、函数的声明；

c）在Makefile中，把game.c文件增加到编译列表中；

d）在bootpack.c中修改中断触发后的函数，改成游戏相关函数调用；

通过以上步骤，便可以完整的**插入一个由键盘中断驱动的游戏功能**，而且对原有功能基本无影响；

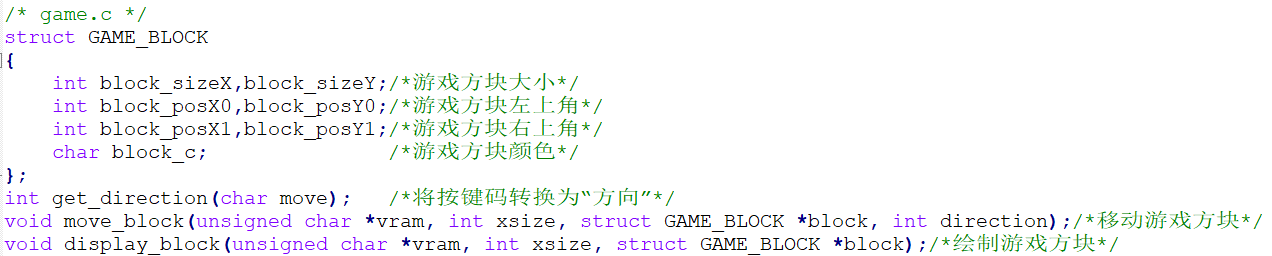
对于代码部分，其实就是一些if-else分支语句，如果写成switch语句的话可能会提高代码可读性，但是时间仓促，就写成这样繁琐的if-else了；但是也不用花时间去看具体是怎么实现的，因为就是对之前函数的调用，基本上没有新函数，下面给出一些函数功能说明：

①get\_direction()——通过按键码，返回一个对应功能的数字，一开始仅仅是四个方向的移动而已，所以取名叫direction，后来增加了几个功能，没有改名字；

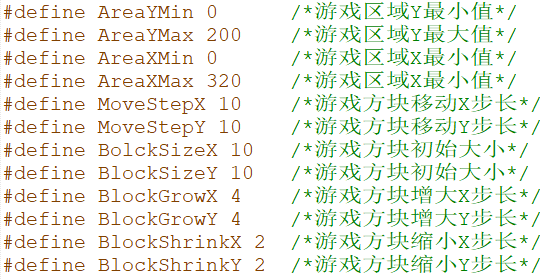
②display\_block()——绘制背景界面、然后再绘制方块；

③move\_block()——更具相应的功能，对方块进行相应的参数修正，然后调用display\_block()函数；

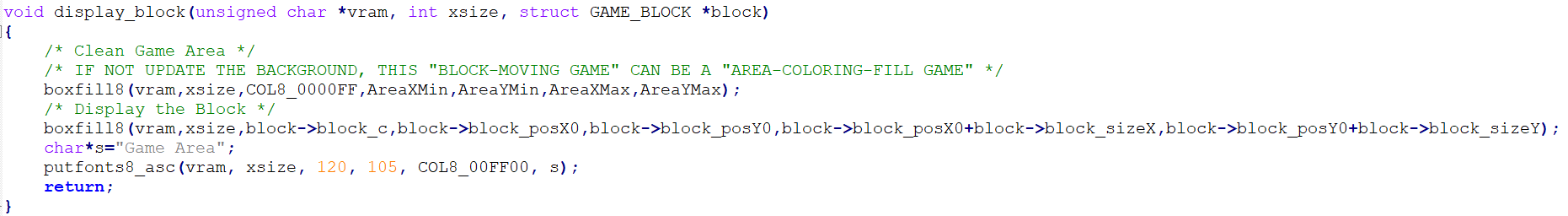
**关键代码：**



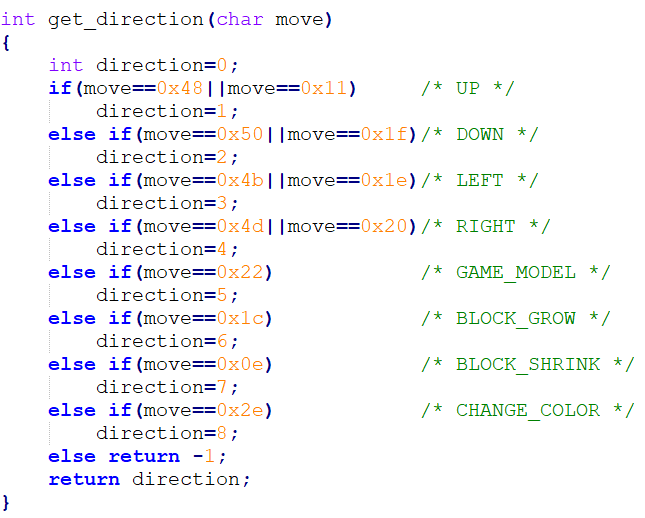
*（bootpack.h文件中，增加的game.c涉及到的函数声明）*



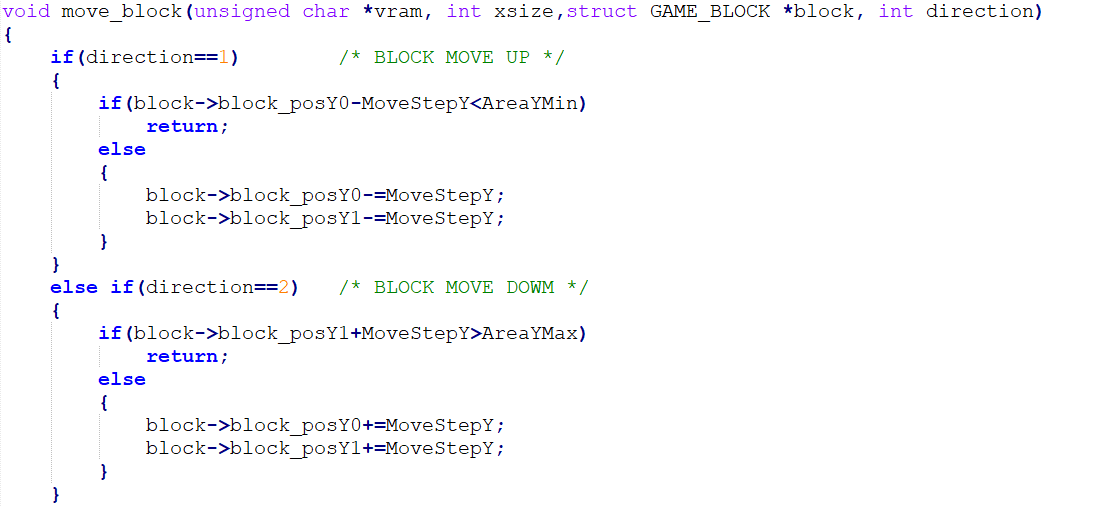
*（*bootpack.h文件中，增加的*game.c涉及到的宏定义）*



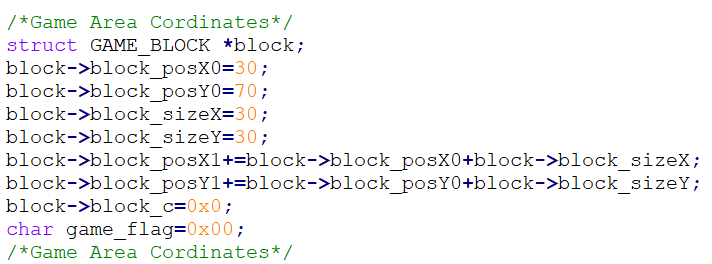
*（game.c的dispay\_block函数实现）*

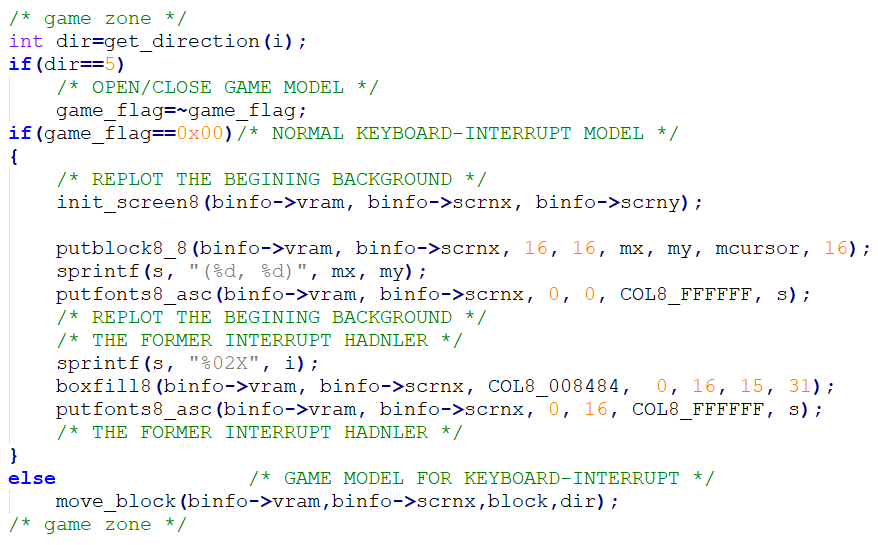


*（game.c的get\_direction函数实现）*



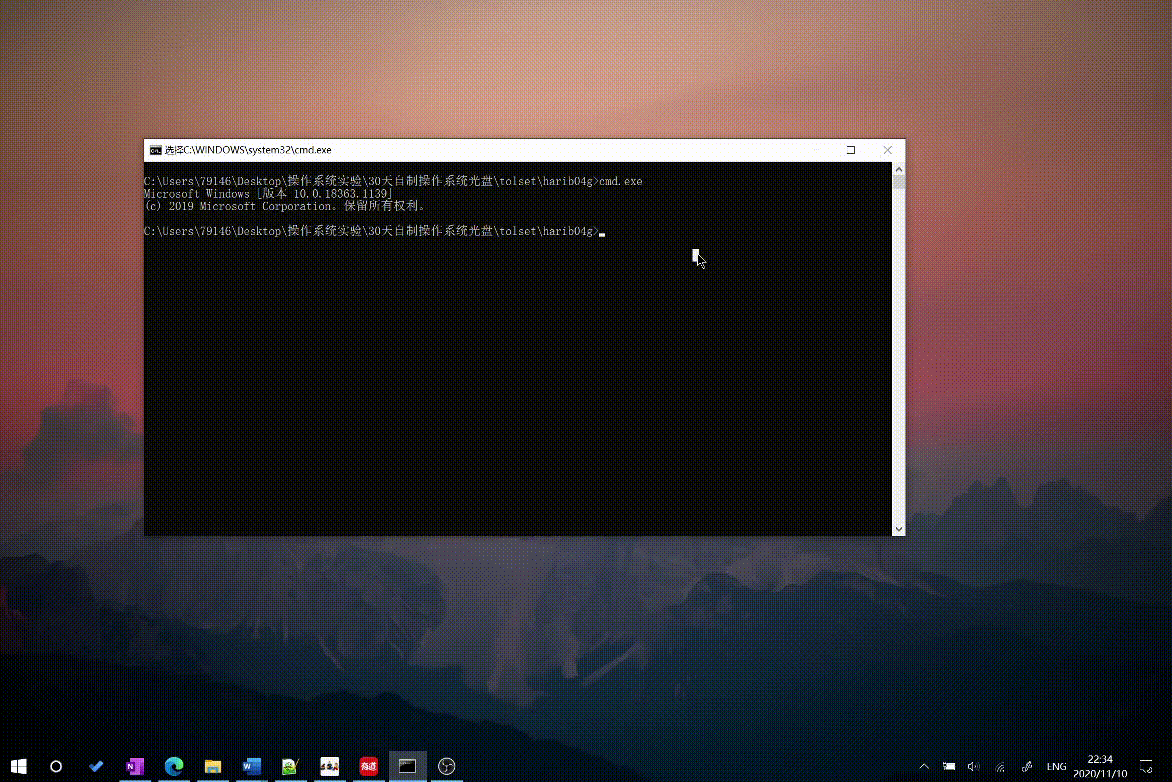
*（game.c的move\_block函数实现节选）*



*（bootpack.c文件中，Harimain函数中增加的方块结构体初始化）*

*(bootpack.c文件中，Hariman函数中增加的键盘中断分支)*

**结果截图（GIF模式）：**

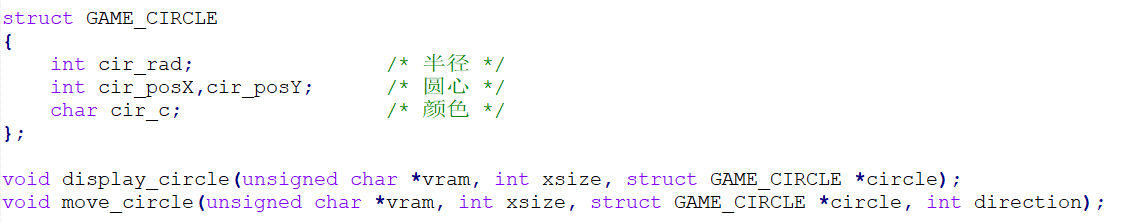


**2、节点考核的内容：**

（呃…..没有料到本次节点考核和这个创新点撞车了……

其实大致内容差不多，下面把不一样的数据结构展示一下、然后纠正一下验收时出现的问题，再附上一张效果图；）

**代码实现：**



*（区别的数据结构）*

**验收时的问题：**

问题描述：圆形的右、下两个突出的点在边界处会被界限“吞噬”；

问题解决：一开始以为是“=”判定出错了，但是发现四个边界的判定条件都一样，但只有 这右、下两个边界出错，故推断不是实现的问题，而是虚拟机本身仿真就会丢失这两个边 界；通过以下代码来验证猜想；

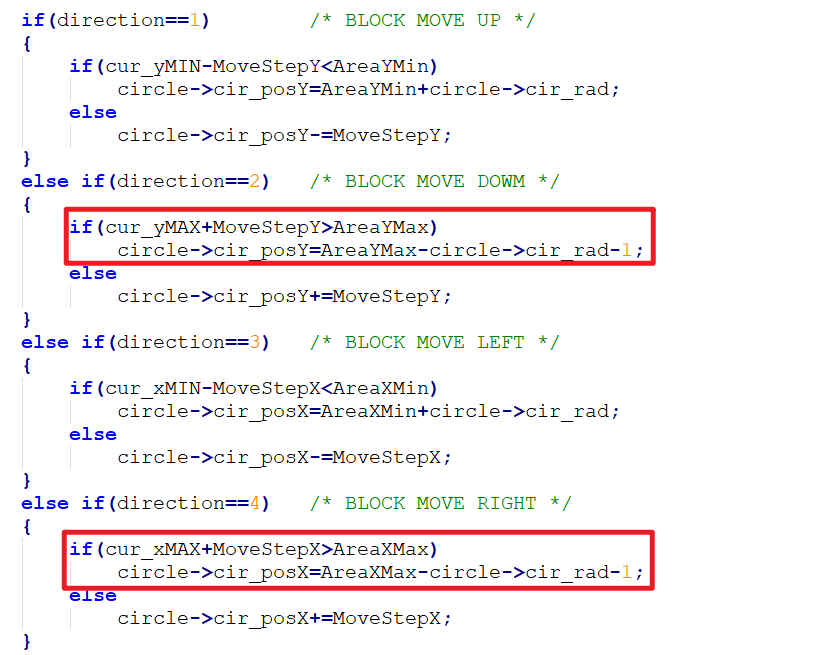
猜想验证：在代码中添加两条语句，作用是在右、下边界画一条红色的边界线，为使对比 度明显，背景色依旧采用蓝色；



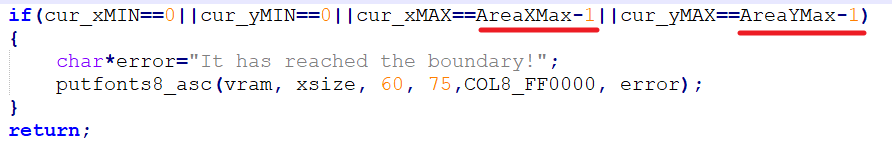
这按道理应该在最下面、最右边画出两条线，但实际上却在左边界画出了红线，下边界 不见了；



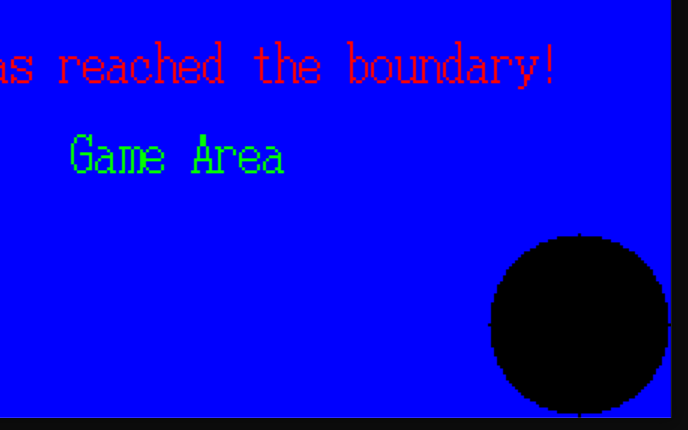
解决：如果要实现左右边界的点都在的话，也不是不可以，只需要把实际的边界缩小 一格，但是这样“实际的边界”和“看见的边界”就不是一个东西了……，实现的时候需 要注意“实际的边界”要小1个像素单位；下面给出实现后的代码和效果：



（坐标修改）



（边界判断）



## 四、实验心得体会

这次实验主要的一个成就感就是，完成了Day 6设想的一个idea，就是制作这样一个由键盘控制的小玩意儿（把它叫做游戏都感觉有点简陋）；虽然比较简单，但是从实际动手来讲的话，通过仿照作者规划好的源码架构，往其中添加自己想要的功能，这样能够让我们更好的理解作者代码的结构，清楚如果我们要自己编写代码，应该从哪些地方开始修改，怎样与已有的代码保持统一，保持可读性。

当然，上面的还是停留在键盘中断的层面，Day 7已经推进到鼠标中断了；

鼠标中断目前实现了数据读取，每时每刻，鼠标都在向缓冲区发送数据，每次中断产生3-bytes的数据，现在我们可以成功的将其读出，但是这些数据表示的具体意义是什么我们暂时还不清楚，以及如何利用这些数据实现一个可以“实时移动”的鼠标图标呢？这些都是还未解决的，但是想要实现一个鼠标，这些肯定是要涉及的，具体的内容就等到Day 8吧！这次的实验算是完成了。