俄罗斯方块程序文档

2019012137 工物90 张鸿琳

2021 年 7 月 7 日

目录

1 大作业逻辑结构

本个程序分为两个部分：①俄罗斯方块的实现②音乐的播放。为了实现这些功能，用到了OLED.c，handle.c和sound.c文件，其中OLED.c中主要是控制OLED的函数，handle.c中主要是控制俄罗斯方块运动的函数，而sound.c中主要是控制音乐播放的函数。具体实现思路如下。

1.1 俄罗斯方块的程序实现

1.1.1 单点控制

为了实现俄罗斯方块，首先要实现对OLED屏幕细化到单个像素的控制，因为块下落的时候是以单个像素点为单位下落的，所以依据老师提供的以8个像素点为单位的控制函数OLED_W_Dot，结合存储数组state，得到了单点控制函数dot，如下：

```
void dot(unsigned char lx,unsigned char ly,char val){
        char temp=lx%8;
        int t=lx/8;
        char vall;
        if(val==1){
        switch(temp)
        {
        case 0:vall=0b00000001;break;
        case 1:vall=0b00000010;break;
        case 2:vall=0b00000100;break;
        case 3:vall=0b00001000;break;
        case 4:vall=0b00010000;break;
        case 5:vall=0b00100000;break;
        case 6:vall=0b01000000;break;
        case 7:vall=0b10000000;break;
        }
        state[ly][t]|=vall;
        OLED_W_Dot(lx,ly,state[ly][t]);
        }
        else if(val==0){
        switch(temp)
                {
                case 0:vall=0b11111110;break;
                case 1:vall=0b11111101;break;
                case 2:vall=0b11111011;break;
                case 3:vall=0b11110111;break;
                case 4:vall=0b11101111;break;
                case 5:vall=0b11011111;break;
```

```
                case 6:vall=0b10111111;break;
                case 7:vall=0b01111111;break;
                }
        state[ly][t]&=vall;
        OLED_W_Dot(lx,ly,state[ly][t]);
        }
}
```

1.1.2 对块的操作

实现了单点操作函数后，本程序中以2*2为单元格，而每个俄罗斯方块都由四个单元格构成，这样先实现在特定位置的单元格点亮函数ll和单元格熄灭函数dd，方便下一步实现块的生成和对块的各种操作。

块的类型共有7种，由lightened函数利用单点点亮函数实现，在固定的初始位置生成下落的块。

块的运动有：①下移②左移③右移④逆时针旋转⑤顺时针旋转。下移是自动进行的，在程序开始后，块会自动下落，而左移、右移和旋转是人为控制的，在进行某个操作前，需要对操作的合理性进行判定，所以需要实现相应的判定函数，即testmove，testright，testleft，testlrotate，testrrotate，分别用于判定是否可以下落、右移、左移、逆时针旋转、顺时针旋转，其原理是先对块的操作进行模拟，得到块在操作后的单元格坐标，再判定模拟出的坐标是否越界或者在该位置处已经存在点亮的块。

在判定某个操作合理后，该操作会被执行，对应的操作执行函数分别为move，right，left，lrotate，rrotate，其都是利用ll和dd函数，根据现在块所处的单元格位置进行变换。其中lrotate和rrotate函数比较冗杂，因为不同的块的不同姿态对应的旋转操作的实现都有所区别，只能分情况实现。如果testmove函数判定可以移动，块会不断下移。

左移、右移、旋转是人为操作实现的，所以需要利用按键中断，中断函数如下：

```
void PORTA_IRQHandler(){
        if((GPIOA_PDIR & 0b1000000000000)==0)
        {
                if(testlrotate()==1)
                {
                        lrotate();
                }

        }
        else if((GPIOA_PDIR & 0b10000000000000)==0)
        {
                if(testrrotate()==1)
                {
                        rrotate();
                }
        }
```

```
17          else if((GPIOA_PDIR & 0b100000000000000)==0)
18          {
19                  if(testleft()==1)
20                  {
21                          left();
22                  }
23          }
24          else if((GPIOA_PDIR & 0b10000000000000000)==0)
25          {
26                  if(testright()==1)
27                  {
28                          right();
29                  }
30          }
31          else if((GPIOA_PDIR & 0b1000000000000000)==0)
32                  {
33                  if(testmove()==1)
34                  {
35                          move();
36                  }
37                  if(testmove()==1)
38                  {
39                          move();
40                  }
41                  if(testmove()==1)
42                  {
43                          move();
44                  }
45                  if(testmove()==1)
46                  {
47                          move();
48                  }
49                  }
50          delay();
51          PORTA_PCR12|=0x01000000;
52          PORTA_PCR13|=0x01000000;
53          PORTA_PCR14|=0x01000000;
54          PORTA_PCR15|=0x01000000;
55          PORTA_PCR16|=0x01000000;
```

```
56 }
```

1.1.3 分数统计以及刷新

当一个块落到底部后，其信息会被更新到lightened数组中，表明该区域已经被之前的块占据，用于之后的操作判定中。

同时程序会执行score函数，判定是否有一层满了，由于游戏区域被限定在64*80的区域内，所以只需要对lightened数组中的某一列求和，如果和等于极限值，那么该层就满了，为了减少判定时间，所以引入了high和low，确定最后落下的块的行范围，这样求和检测操作只需要执行1-4次即可，节约了时间。

如果存在某一层满了，就会执行above函数，也就是将该层消去，同时将上面的层平移下来，利用lightened数组和ll、dd函数，这样的操作是很容易实现的，同时也是为了节约时间，引入了min变量记录堆积的最高层数。

之后会更新相应的分数，通过addscore显示出更新的分数。

之后会执行lose函数，判定是否越界，如果越界，则会显示“Game Over”，并进入新的游戏。

1.2 音乐的播放

音乐播放的代码较为简单，利用老师所给的代码思路即可完成。主要思路就是利用时钟中断和PWM波，通过时钟中断切换不同的音高并控制音长，而利用PWM的频率设置不同的音高。由于乐谱需要，在老师所给代码的基础上又引入了几个音高，实现了butterfly的播放。

在游戏上加上BGM是考虑到原来的主程序可以使用时钟中断和delay函数两种方式实现块的周期下落，如果使用delay函数和按键中断搭配实现游戏控制，就可以再利用时钟中断实现BGM播放，因为按键中断和时钟中断两个中断基本不会发生冲突。

2 一些待改进处

本次作业还有很多可以改进之处：

- 随机函数不够随机。由于使用了rand函数随机生成块，而rand是和系统时间相关的，这导致前面几个生成的块总是一样的，不过该影响随着游戏进行而消失
- 由于中断发生的时间随机，所以假如在程序的move函数中发生中断，可能会导致出现一两个像素块出问题，不过在调试后基本没有发生过，理论概率也很低，对程序整体运行也没有影响
- 在游戏设计方面，块的大小比较小，而整个游戏空间较大，以至于游戏缺少挑战性
- 游戏的互动性还可以增强，理论上，可以使用加速度传感器控制块的移动，同时BGM可以替换为游戏音效，在某些操作时触发
- 游戏的灵活性也还可以增强，比如可以在开始时增加选择界面，利用按键确定块的下落速度，以及游戏空间大小，还可以增加暂停键
- 没有设置中断嵌套，所以时钟中断和按键中断嵌套的影响尚且未知

- 代码方面，还不够精简，在编写最后发现一些函数还可以进一步合并和归纳整理

3 借鉴部分

- 负责OLED屏幕控制的OLED.c和OLED.h文件的大部分内容（除了单点控制函数）是曾老师提供的
- 负责音乐播放控制的sound.c和sound.h文件中的初始化函数和整个代码逻辑是曾老师提供的