俄罗斯方块项目报告

分工

学号	姓名	分工	占比
11712531	张婷婷	coder/tester	34%
11712325	张家毓	coder/tester	33%
11610615	郭孟维	coder/tester	33%

模块设计

游戏模型设计

- 模型一

- 模型二

- 模型三

- 模型四

- 模型五

- 模型六

- 模型七

按键模块设计

- WK_UP按键：实现功能为，按下WK_UP按键模型左移一格，对应到LCD上为左移20个像素点距离。
- KEY1按键：实现功能为，按下KEY1按键模型顺时针旋转90度。
- KEY0按键：实现功能为，按下KEY0按键模型右移一格，对应到LCD上为右移20个像素点距离。

> 需要注意的点为：WK_UP的输入模式应设计为Pull-Down模型。

状态栏模块设计

我们将LCD屏幕最右边的部分用作状态的显示。在状态栏里，我们将显示将要下落的两个模型的等比例缩小的图像和得分和游戏等级。游戏的得分和等级的提升遵循我们的游戏规则。

具体实现

全局变量

我们设计了一系列的全局变量来方便我们对功能的实现。

- xoffset：模型的水平偏移量。
- if_bottom：判断模型是否触底。
- arrays[16][10]：记录LCD屏幕里对应方格的颜色。
- temp_arr[16][10]：记录消除满行后的方格颜色。
- rotation：标记模型旋转的形状。
- drop：模型的竖直位移量。
- speed: 模块的初始速度。
- count_row： 计算消除的满行数量。

二维数组

我们使用二维数组array[16][10]来记录整个屏幕有方格的位置对应的颜色。

- 当一个模型停止下落时，我们将记录下方格的位置,将该模型对应的颜色写入二维数组中。
- 当模块需要移动时，我们将检测模块即将移动的位置是否已经在二维数组中被记录。
- 当游戏发生满行消除时，根据游戏方块的整体位置更新二维数组。
- 当方块堆积到屏幕顶部时，使用二维数组判断游戏是否结束

主要方法

基于整体模块架构，我们设计编写了以**draw_model_x**，**judge**，**fill_record**，**showStatus**， **check_array** 等方法为核心的源码程序.

接下来，我们将分别对这五个核心方法进行具体分析。

1.draw_model_x(以draw_model_1为例)

I.方法结构

void draw_model_1(int lor);

II.方法功能

此方法将根据输入的参数lor，对于当前下落方块的行为进行判断（左移，右移，变形，下降）。并在屏幕上绘制出执行对应行为后的方块图形。

- **lor = -1:** 左移操作
- **lor = 0:** 下移操作
- **lor = 1:** 右移操作
- **lor = 2:** 变形操作

III.方法实现

正如前文所述，我们的设计思路是使用两个全局变量来表示当前下落方块在整个屏幕中的位置。因此，draw_model这一系列方法首先需要根据不同形状方块的特征来把方块的中心坐标转化为对应该方法的图形。即将xoffset与drop两个变量转化(x1, y1), (x2, y2），如方块由多个部件组成，则分别一一转化。

转换坐标完毕后，根据lor的值，在judge函数通过的情况下更新xoffset与drop两个位置值。如果命令是旋转，则基于新的转换规则获取新的图形绘制区域。

最后，基于更新的xoffset与drop值，更新获得新的图形绘制区域，通过**LCD_Fill**方法绘制方块。

IV.方法源码

```
void draw_model_1(int lor)
{

    int x1 = 0;
    int y1 = 0;
    int x2 = 0;
    int y2 = 0;

    // last state
    if (rotation % 2 == 0)
    {
        x1 = 60 + xoffset;
        y1 = 0 + drop;
        x2 = 140 + xoffset;
        y2 = 20 + drop;
    }
    else
    {
        x1 = 100 + xoffset;
        y1 = 0 + drop;
        x2 = 120 + xoffset;
        y2 = 80 + drop;
    }

    // erase the previous state



    // judge the next state

    // judge left
    if (lor == -1) {
```

```
        if(judge(x1-20, y1, x2-20, y2) == 1) {
            xoffset -= 20;
        }else{

        }
    }
    // judge right
     if (lor == 1) {
        if(judge(x1+20, y1, x2+20, y2) == 1) {
            xoffset += 20;
        }else{

        }
    }
    // judge drop
    if (lor == 0) {
        // if next state is legal
        if (judge(x1, y1+speed, x2, y2+speed) == 1) {
            drop += speed;
        // if next state is illegal
        }else{

            if_bottom = 1;
            fill_record(x1, y1, x2, y2,1);
        //  LCD_Fill(x1,y1,x2,y2,YELLOW);
        //  return ;
        }
    }
   // judge rotate
    if (lor == 2) {
        //LCD_Fill(x1, y1-speed, x2, y2-speed, WHITE);

        int last = rotation - 1;

        if (last % 2 == 0)
        {
            LCD_Fill(60 + xoffset, 0 + drop, 140 + xoffset, 20 + drop, WHITE);
        }

        else
        {
            LCD_Fill(100 + xoffset, 0 + drop, 120 + xoffset, 80 + drop, WHITE);
        }

        if (judge(x1,y1,x2,y2) == 0) {
            rotation -= 1;
        }
    }

   // update the next state
    if (rotation % 2 == 0)
      {
          x1 = 60 + xoffset;
          y1 = 0 + drop;
          x2 = 140 + xoffset;
          y2 = 20 + drop;
       }
     else
```

```
     {
          x1 = 100 + xoffset;
          y1 = 0 + drop;
          x2 = 120 + xoffset;
          y2 = 80 + drop;
     }


    // draw the block
    if (if_bottom == 0) {
        if ( lor == -1 && judge(x1, y1, x2, y2) == 1) {
            LCD_Fill(x1+20, y1, x2+20, y2, WHITE);
        } else
        if (lor == 1 && judge(x1, y1, x2, y2) == 1) {
            LCD_Fill(x1-20, y1, x2-20, y2, WHITE);
        }else if (lor == 0) {
            LCD_Fill(x1, y1-speed, x2, y2-speed, WHITE);
        }

        LCD_Fill(x1, y1, x2, y2, YELLOW);
    }
}
```

2. judge

I.方法结构

int judge(int x_1, int y_1, int x_2, int y_2);

II.方法功能

此方法将根据输出四个参数x_1,y_1,x_2,y_2，判断由（x_1. y_1）,(x_2, y_2) 组成的矩形区域在屏幕上是否能够绘制图形。如果能够绘制图形，则返回1；如果超出屏幕边界或者与已绘制图形区域重合，则返回0.

III.方法实现

首先，因为屏幕的游戏区域固定，我们可以很容易地判断指定的绘制范围是否碰壁。关键点在于，之前未消除方块很可能与判断区域存在重合。为了解决这个问题，我们设计了指代整个屏幕颜色情况的二维数组arrays。通过该数组，我们可以知道屏幕上某个固定区域的颜色情况（每个区域大小为20*20）。在judge函数中，我们会将输入的绘制范围转化为二维数组中的数据范围，随后检察数组中此范围内是否存在颜色不为白色的值（即数值不为0）。如果存在，返回0；如果不存在，返回1。

IV.方法源码

```
int judge(int x_1, int y_1, int x_2, int y_2)
{
```

```
    if (x_2 > 200 || x_1 < 0 || y_1 < 0 || y_2 > 320)
    {
        return 0;
    }
    int x1 = x_1 / 20;
    int y1 = y_1 / 20;
    int x2 = x_2 / 20;
    int y2 = y_2 / 20;

    //如果刚好在在方格里
    if (y_2 % 20 == 0)
    {
        int i = 0;
        int j = 0;
        int f1 = 1;
        int f2 = 1;
        for (j = x1; j < x2; j++)
        {
            for (i = y1; i < y2; i++)
            {
                if (arrays[i][j] > 0)
                {
                    f1 = 0;
                }
            }
        }
        for (int i = x1; i < x2; i++)
        {
            if (arrays[i][y2] > 0)
            {
                f2 = 0;
            }
        }
        if (f1 == 1 && f2 == 1)
        {
            return 1;
        }
    }

    //如果不在在方格里
    else
    {
        int i = 0;
        int j = 0;

        y2 = y2 + 1;
        for (j = x1; j < x2; j++)
        {
            for (i = y1; i < y2; i++)
            {
                if (arrays[i][j] > 0)
                {
                    return 0;
                }
            }
        }
    }
```

```
    return 1;
}
```

3.fill_record

I.方法结构

void fill_record(int x_1, int y_1, int x_2, int y_2, int color);

II.方法功能

此方法将根据参数中指定的屏幕范围，将指定的颜色写入代表整个屏幕颜色情况的二维数组中。

III.方法实现

方法实现清晰明了，按照20：1的比例尺转化输入参数为二维数组坐标，再将颜色值写入坐标范围中即可。需要注意的是，由于数组建立的方向与屏幕坐标轴方向相反，更新数组时x与y要反过来。

IV.方法源码

```
void fill_record(int x_1, int y_1, int x_2, int y_2, int color)
{
    int x1 = x_1 / 20;
    int x2 = x_2 / 20;
    int y1 = y_1 / 20;
    int y2 = y_2 / 20;

    if (x2 == x1 + 1)
    {
        int i = 0;
        for (i = y1; i < y2; i++)
        {
            arrays[i][x1] = color;
        }
    }
    else if (y2 == y1 + 1)
    {
        int i = 0;
        for (i = x1; i < x2; i++)
        {
            arrays[y1][i] = color;
        }
    }
    else if (x2 >= x1 + 2 && y2 >= y1 + 2)
    {
        int i = 0;
        int j = 0;

        for (i = x1; i < x2; i++)
```

```
        {
            for (j = y1; j < y2; j++)
            {
                arrays[j][i] = color;
            }
        }
    }
}
```

4.showStatus

I.方法结构

static void showStatus(int levelNum, int scoreNum, int next_shape1, int next_shape2);

II.方法功能

此方法用来绘制标识有等级，分数与方块预告的游戏状态栏。

III.方法实现

此方法实现十分简单，即根据 **LCD_ShowString**与**LCD_Fill**方法将关卡等级与分数值实时显示在状态中。唯一麻烦的地方是要根据七种不同的方块预先写好方块预告的绘制位置。

IV.方法源码

```
static void showStatus(int levelNum, int scoreNum, int next_shape1, int next_shape2)
{

    char levelMsg[5];
    char scoreMsg[5];
    sprintf(levelMsg, "%d", levelNum);
    sprintf(scoreMsg, "%d", scoreNum);
    LCD_ShowString(205, 210, 15, 15, 12, levelMsg);
    LCD_ShowString(205, 280, 15, 15, 12, scoreMsg);
    LCD_Fill(204,70,240,160,WHITE);
    switch (next_shape1)
    {
     ////////////////////////绘制预告图形部分代码过于冗余，故不引用////////////////////////
    }

}
```

5.check_array

I.方法结构

*void check_array(void);

II.方法功能

此方法在每块方块停止下落时被触发，用来判断是否有一些行已经满了，如果满了则消除掉，并记录消除后原来未被消除位置的颜色并下落到相应位置上。

III.方法实现

此方法实现方式较为直接，即自上而下遍历数组，若检测到一行已满，则先消除检测到的这行，并让这行上面的元素下落一行存到temp_array中，然后record_array再更新为temp_array里的二维数组，然后再重新调用此函数，递归操作，直至检测到最后一行也没有满行为止。

IV.方法源码

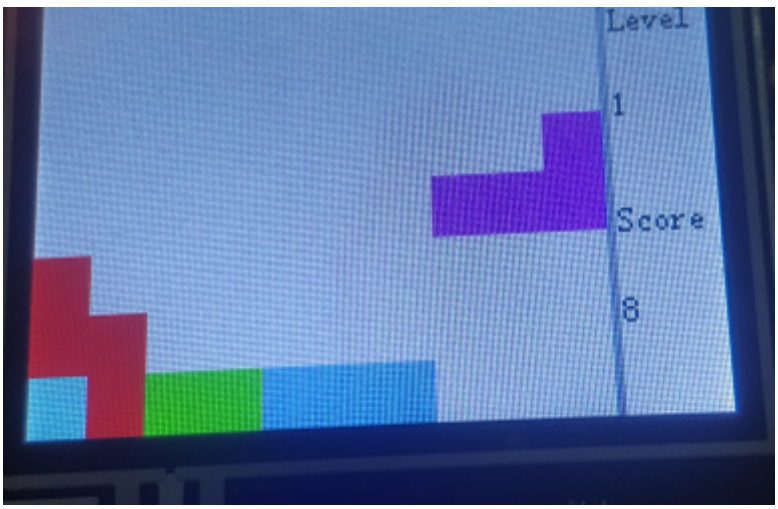
```
void check_array(void)
{
    int i = 0;
    int j = 0;
    int temp_row = -1;

    for (i = 0; i < 16; i++)
    {
        int if_full = 1;
        for (j = 0; j < 10; j++)
        {

            if (arrays[i][j] == 0)
            {
                //  printf("%d\t",17);
                if_full = 0;
                break;
            }
        }
        if (if_full == 0)
        {
            //printf("%d\t",18);
            continue;
        }
        else
        {
            count_row += 1;
            temp_row = i;
            printf("temp_row: %d\t", temp_row);

            int k = 0;
            int o = 0;
            for (k = 0; k < 16; k++)
            {
                for (o = 0; o < 10; o++)
                {
                    if (k <= temp_row)
                    {
```

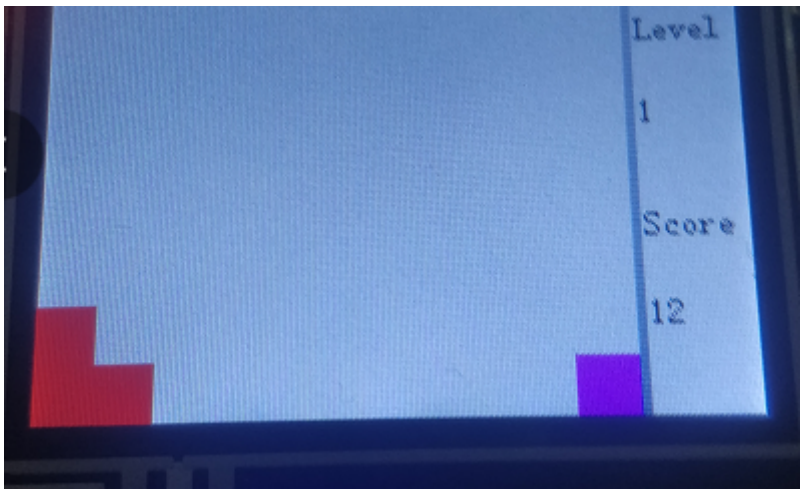
```
                        if (k >= 1)
                        {
                            temp_arr[k][o] = arrays[k - 1][o];
                        }
                    }
                    else
                    {
                        temp_arr[k][o] = arrays[k][o];
                    }
                }
            }
            printf("test: %d\t", temp_arr[0][0]);
            k = 0;
            o = 0;
            for (k = 0; k < 16; k++)
            {
                for (o = 0; o < 10; o++)
                {
                    arrays[k][o] = temp_arr[k][o];
                }
            }
            //   printArrays();
            // printf("\n");
            check_array();
        }
    }
    return 0;
}
```

+:

结果展示

- 右1边状态栏会显示等级和分数，且满行后会消失



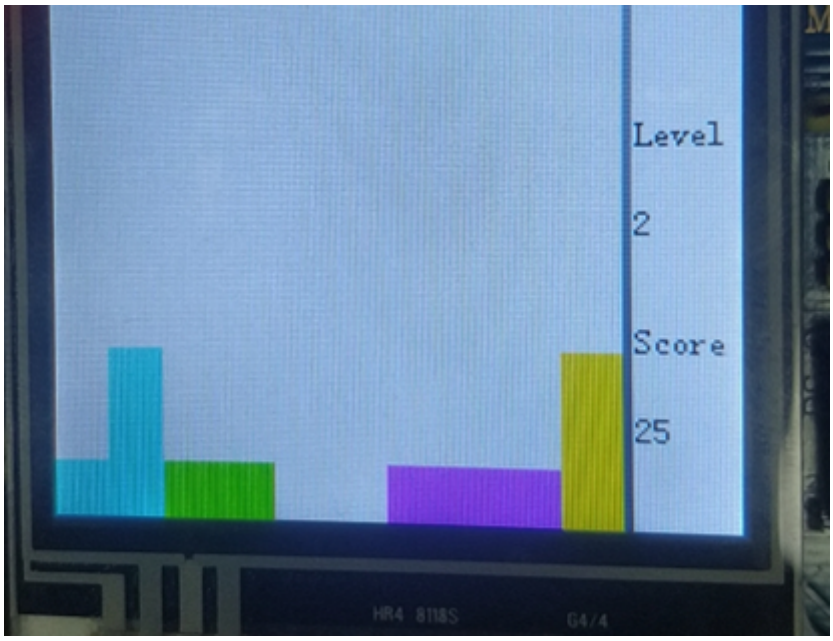


- 状态栏会显示接下来的两个形状

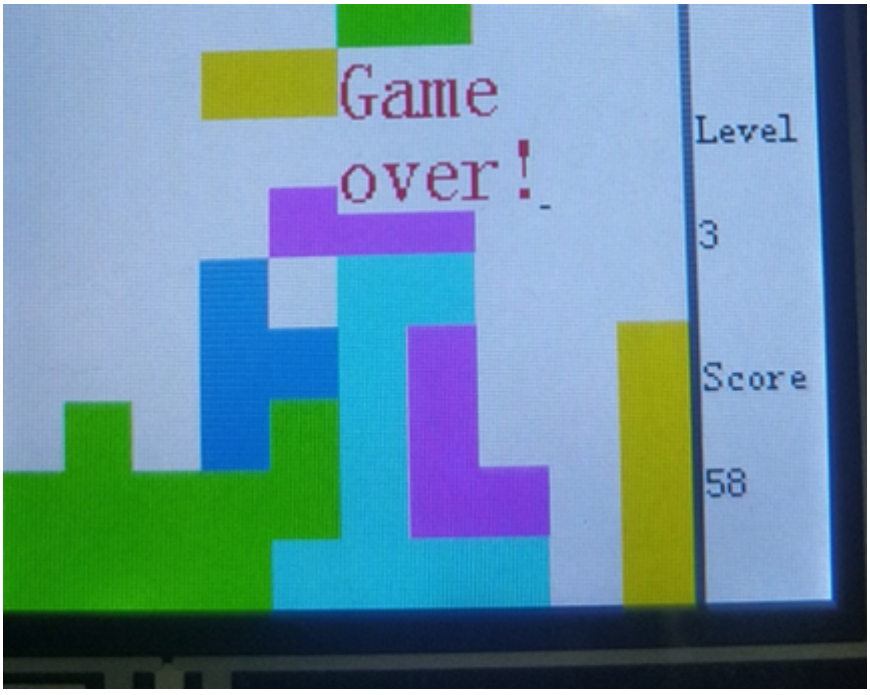
计分规则： 每下落一个记1分，消除1行记3分

- level 1（20分以下，速度为2）

- level 2（超过20分自动升级level2，速度为5）



- level3（超过50自动升级为level3，速度为10）



- 初始形状

- 向左移动

- 向右移动

- 旋转后的形状

未来工作

- 加入更多方块类型
- 提高等级上限，优化关卡限制与分数要求
- 充分利用开发板上LED的闪烁，增强游戏性
- 加入声效