俄罗斯方块

——c语言实现与汇编语言实现之比较

学号：16340274 姓名：杨元昊

开发环境说明

汇编程序是在dosbox虚拟机系统下，使用masm5.0编译。该程序为16位程序，在一般的32位系统环境中无法正常运行。如需执行文件4.exe，需要在dos下打开。、

c语言程序是利用visual studio 2017开发的，点击.sln文件，然后按运行按钮即可。

项目演示情况

汇编项目演示视频可以在此观看。

c语言项目演示视频可以在此观看。

一. 项目说明

本学期计算机组成与原理课程中讲述了x86汇编程序的编写。编程这个游戏具体的功能是：

1. 使用左右下空格键控制方格：

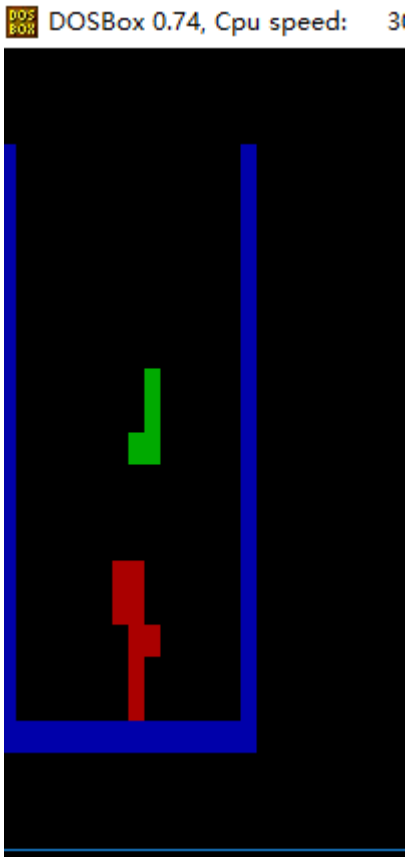
空格键会旋转方格

2. 方格降落时就会改变颜色：

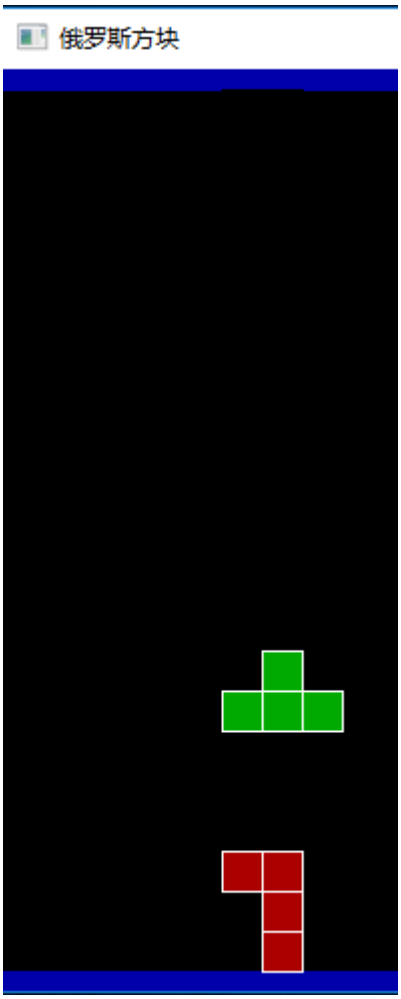
方格堆满一行时会自动消失

3. 方格触碰到界面最顶端时游戏失败退出

以下为汇编语言的程序运行结果



以下为C语言程序的运行结果



二. 选题原因

选择该题目，有以下几个原因：

1. 能够使用汇编语言直接操控硬件

我们早已学过C语言，了解到C语言作为一门较为底层的语言，能够直接操控内存，与硬件做交互。可是c语言与硬件间，仍然隔了一层汇编语言，没有突破汇编语言这一关，我们不能更好的理解程序是如何直接操控硬件的。在学习汇编语言的过程中，我了解到汇编语言能够通过直接改写显存来控制显示屏的输出，还能够直接与键盘交互，读取用户的键盘输入。相比大一学习c语言编程时调用windows.h这样的API，觉得直接使用汇编语言来操控电脑会更有意义。因此，想更深入地去理解汇编语言操控显存，键盘输入方面的知识。

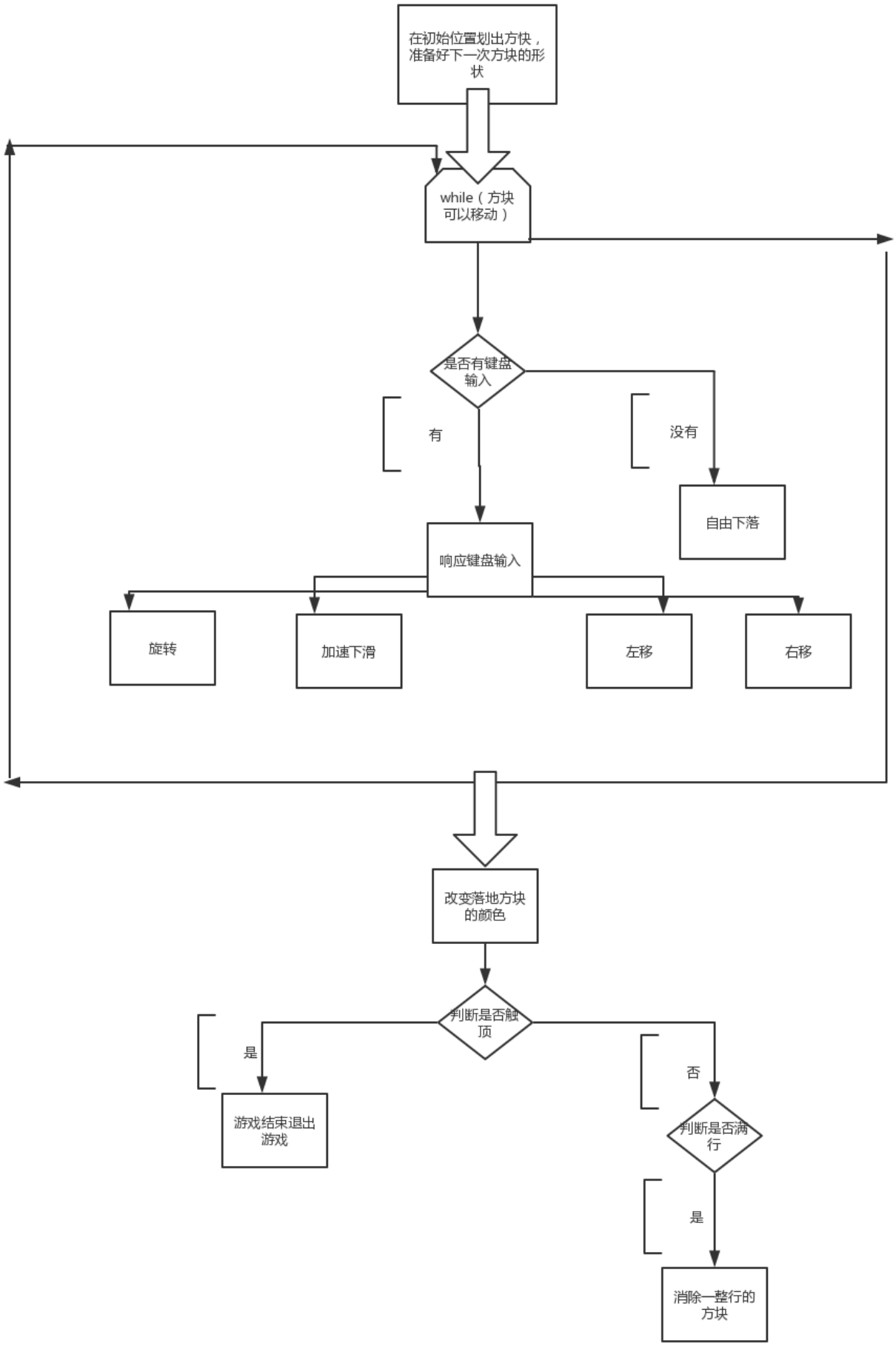
2. 俄罗斯方块更能考验汇编语言的编程的综合水平

会写汇编语言，不仅仅要求顺序，分支，循环三种基本结构能够实现，更重要的是能够从底层的原理了解函数的调用，了解硬件的交互。而这个游戏在各方面都有一些难度。能够解决在实现项目中的各种问题，更能加深自己对汇编语言的了解。

3. 想更深入的对比一下汇编语言和c语言编程出的可执行文件的不同

俄罗斯方块是一个可以在电脑中长时间运行，且要耗费一定内存的游戏，方便我们比较exe执行的不同。

三. 程序框图


在初始位置划出方快，准备好下一次方块的形状
while（方块可以移动）
是否有键盘输入
有
没有
自由下落
响应键盘输入
旋转
加速下滑
左移
右移
改变落地方块的颜色
判断是否触顶
是
游戏结束退出游戏
否
判断是否满行
是
消除一整行的方块

四. 项目实现的主要思路

1.实现界面的绘制

将整个游戏界面看成是n*n的矩阵，通过寄存器直接修改显存，可以改变屏幕显示的颜色。我们用screen存储界面边界墙的位置，如果是0的话就表示为屏幕背景颜色，1的话就表示改变成墙的蓝颜色。我们用ds存储方块的位置，使用loop循环命令，依次改变各个ds中各个位置在屏幕显示的颜色。于是可以实现方块的显示。当判断方块已经无法下落时就再次修改方块的颜色。

2. 实现方块的移动

在判断没有碰撞时，使用loop循环将方块的每个位置移动（矩阵中表示的是1），在新位置画上颜色，然后将旧位置的颜色设置为背景颜色。对于自动移动而言，我们需要设置一个空循环，空循坏内设置没有意义的n++，这样可以实现定时功能，当检测到没有键盘输入时，方块就会自动下落。

3. 方块变形

如一下设置所示，每一组BLOCK_*（*是类型）都是不同类型图案的多种不同形态，用4 * 4的矩阵来将它们统一表示，它们的顺序也是按照逆时针旋转进行排列的。 比如我们可以对BLOCK_T型的四个小矩阵进行编号为1，2，3，4。 那么显然1 - 旋转 - > 2 - 旋转 - > 3 - 旋转 - > 4 - 旋转 - >1 。我们通过设置BLOCK_MAX_STATUS表示当下正在下落的方块的n种形态，每当我们按下空格键的时候就可以实现形态的转变（方块表示矩阵整体在游戏界面的位置不变，但矩阵内的0,1分布发生改变）。颜色的显示根据矩阵的0,1来判断。这样可以极好的节省代码量，减少出错几率。

```
BLOCK_O         db  '1100'
            db  '1100'
            db  '0000'
            db  '0000'

BLOCK_I         db  '0000'
            db  '1111'
            db  '0000'
            db  '0000'

            db  '0100'
            db  '0100'
            db  '0100'
            db  '0100'

BLOCK_S         db  '0000'
            db  '0110'
            db  '1100'
            db  '0000'

            db  '0100'
            db  '0110'
            db  '0010'
            db  '0000'

BLOCK_Z         db  '0000'
            db  '1100'

            db  '0110'
```

```
            db  '0000'

            db  '0010'
            db  '0110'
            db  '0100'
            db  '0000'

BLOCK_T         db  '0100'
            db  '1110'
            db  '0000'
            db  '0000'

            db  '0100'
            db  '0110'
            db  '0100'
            db  '0000'

            db  '0000'
            db  '1110'
            db  '0100'
            db  '0000'

            db  '0100'
            db  '1100'
            db  '0100'
            db  '0000'

BLOCK_J         db  '0010'
            db  '0010'
            db  '0110'
            db  '0000'

            db  '0000'
            db  '1000'
            db  '1110'
            db  '0000'

            db  '0110'
            db  '0100'
            db  '0100'
            db  '0000'

            db  '0000'
            db  '1110'
            db  '0010'
            db  '0000'

BLOCK_L         db  '0100'
            db  '0100'
            db  '0110'
            db  '0000'


            db  '0000'
```

```
            db  '1110'
            db  '1000'
            db  '0000'

            db  '0110'
            db  '0010'
            db  '0010'
            db  '0000'

            db  '0000'
            db  '0010'
            db  '1110'
            db  '0000'

BLOCK_MAX_STATUS    dw  0,1,1,1,3,3,3
```

4， 方块消行

定义blockheight变量，存储了在界面下方堆积的方块群体的最高高度。每当有一个方块落下到底时，用bx存储这个方块所在的高度，使用loop语句判断这一高度的一整行是否都有方块。因为整个游戏界面可以看做是存在显存的一个矩阵，0代表没有方块，1代表有正在降落的方块，2代表有已经着陆的方块。于是我们可以从blockheight开始直到bx，用loop循环，把矩阵中上一行的值复制到下一行，实现消除方块占满的这一行。

5，碰撞检测

对正在下落的方块，（内存中是一个4*4的矩阵），预先判断他的下方是否已有方块，有则代表方块需要停在当前位置，改变该方块颜色，接着又开始了生成新的小方块的循环。

五. 心得体会

通过这一次项目，我对以下内容有了更加深刻的了解。

1. 8086CPU中断机制

在阅读王爽著的《汇编语言》的时候，该本书对8086CPU中断机制做了很详细的描述，还给出了如何自己实现中断例程并安装的相关说明。为了对8086中断机制有更加深入地了解，希望能够在自己的项目中使用上8086的中断机制，并且使用自己的中断例程。如在本程序中，用户控制方块的移动与旋转就是依靠中断机制实现的。

2. 对8086CPU端口的了解

8086CPU可以读取三个地方的数据：寄存器，内存，以及与外部设备相连接的端口。每一个硬件所占用的端口，都会连接到该硬件的寄存器组中，通过读写这个寄存器组，就能够实现与外部设备的I/O操作。在程序中，通过读写40h端口与43h端口，能够对可编程计数器/定时器进行操作，读取数据，并进行模运算得到一定范围内的随机数。

3. 非阻塞I/O的实现

在俄罗斯方块这个项目中，一个较难实现的点是：汇编程序发出IO请求等待键盘输入之后，并不等待IO操作完成，而是继续执行下面的指令（非阻塞），这里参考了【CSDN博客：8086汇编初学之贪吃蛇】中的实现（链接详见参考资料）。

4. x8086汇编修改显存的方法

在内存地址结构中，B8000H~BFFFFH共32KB的空间，为80x25彩色字符模式的显示缓冲区。向这个地址空间写入数据，写入的内容将立即出现在显示器上。由此可以实现方块的显示。

5. C语言与汇编语言之比较

在编写代码时，可以很明显的感觉到语言上更繁琐，栈，寄存器，内存都完全归自己操控，很多时候会忘记自己之前在寄存器存了什么，但却更能感受到自己对电脑的掌控，尤其是在汇编语言一书中提及的那样，直接修改显存，显示不同图案，这令我非常兴奋。而c原因的GUI编程库，调用起来方便了很多，确少了一种对电脑完全掌控的快感。

C语言和汇编语言编写了几乎一样的程序，那么他们的运行效率之间会有多少差别呢？

在编译出了.obj文件后，经过对比发现c语言编写得到的目标代码有22kB大小，而汇编语言的目标代码只有3KB。

经过链接得到exe文件后，经过对比发现C原因得到的可执行文件有120KB之大（也可能跟IDE有关）,而汇编语言得到的可执行文件只有3KB大小和目标代码大小保持一致。

可执行文件运行时，可以在任务管理器中发现汇编语言编写的俄罗斯方块几乎不占用内存（任务管理器上执行了俄罗斯方块和不执行俄罗斯方块的dosbox相比占用内存肉眼上看不到差异），而c语言得到的exe文件运行起来会占用大概0.01%-0.02%的CPU。

总而言之，汇编语言进行编程较困难，但却可以极大地节省程序运行的内存消耗，CPU占用，和编译出来的文件大小。

七. 参考资料

在完成这一个项目的时候，以下参考资料给了我很大的帮助

1. 《汇编语言》 王爽著
2. 【csdn博客：汇编语言 俄罗斯方块 】 https://blog.csdn.net/zjbh89757/article/details/53816106
3. 【CSDN博客：8086汇编初学之贪吃蛇】http://blog.csdn.net/to_be_better/article/details/53512913
4. 【CSDN博客：windows下实现win32俄罗斯方块练手，编程的几点心得 】https://blog.csdn.net/wangyaninglm/article/details/50810939
5. 维基百科 词条 【int 10h】： https://zh.wikipedia.org/zh-hans/INT_10H