汇编语言与接口技术实验报告

第零章　成员信息

班号	学号	姓名
07111807	**1120182328**	**明睿博**
07111806	**1120181348**	**桂梦婷**
07111806	**1120181412**	**郝瑞宁**
07111808	**1120182431**	**谢梦莹**
07111808	**1120182394**	**蒋泽林**

第一章　概要

经讨论决定，我们组决定分别实现三个控制台风格小游戏：贪吃蛇、飞机游戏、俄罗斯方块。

在实现每个任务时，我们先使用纯 C 语言开发，捋清思路。再使用汇编语言与 C 语言混合编程开发（诸如捕获键盘动作等操作，我们直接调用了 C 库中对应的函数）。

具体任务分工如下：

明睿博（组长）：统筹安排、C 语言贪吃蛇、汇编语言贪吃蛇、书写报告文档第一章及第二章。

桂梦婷（组员）：开发 C 语言俄罗斯方块，汇编语言俄罗斯方块 start, freshPaint, randomType 部分，书写报告文档第三章。

郝瑞宁（组员）：开发俄罗斯方块，汇编语言剩余部分(initPaint, moveDirect, rotate, fullRelease)完成及对应报告部分书写。

谢梦莹（组员）：开发飞机游戏，汇编语言飞机游戏（play_line，sp_line）函数，汇编主题编写以及对应的报告。

蒋泽林（组员）：开发飞机游戏，C 语言飞机游戏，汇编语言飞机游戏，书写报告文档第四章。

汇编语言与接口技术实验报告

第二章 贪吃蛇

1.1 实验目的

使用汇编语言编写一个控制台风格的贪吃蛇小游戏，操作键位是 WASD 键位。
目前包括的功能有：

1. 随机出现食物。
2. 撞墙/撞身体结束游戏。

期待加入的功能有：

1. 计分并显示。
2. 加入 perk（混淆方向/速度加快/速度减慢）。

1.2 实验环境

Visual Studio 2015
masm32

1.3 实验内容

1.3.1 使用 C 语言编写贪吃蛇小游戏

```
#include <cstdio>#include <cstdlib>#include <conio.h>#include <Windows.h>
/* todo:              score(with border)                buff(lives'L', speeddown'D')
          debuff(confuse'C', speedup'U')                美化
*/const int height = 20;const int width = 30;int mp[height][width] = { 0 };                    //
越慢

                                                                                                //
          HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
          COORD pos;
          pos.X = x;
          pos.Y = y;
          SetConsoleCursorPosition(handle, pos);
          return;}
```

```
/*  ·移动小蛇  ·第一步扫描数组 mp 的所有元素，找到正数元素都加 1  ·找到最大元素（即蛇尾巴），把其变为 0  ·找到等于 2 的元素（即蛇头），根据输出的上下左右方向把对应的另一个像素值设为 1（新蛇头）  */void moveSnake(){
        int i, j;
        for (int i = 1; i < height - 1; i++)
                for (int j = 1; j < width - 1; j++)
                        if (mp[i][j] > 0)
                                mp[i][j]++; //here

        int oldTail_i, oldTail_j, oldHead_i, oldHead_j;
        int max = 0;
        for (i = 1; i < height - 1; i++)
                for (j = 1; j < width - 1; j++)
                        if (mp[i][j] > 0)
                        {
                                if (max < mp[i][j])
                                {
                                        max = mp[i][j];
                                        oldTail_i = i;
                                        oldTail_j = j;
                                }

                                if (mp[i][j] == 2)
                                {
                                        oldHead_i = i;
                                        oldHead_j = j;
                                }
                        }

        int newHead_i, newHead_j;
        if (direction == 1)                     //  向上移动
        {
                newHead_i = oldHead_i - 1;
                newHead_j = oldHead_j;
        }
        if (direction == 2)                     //  向下移动
        {
                newHead_i = oldHead_i + 1;
                newHead_j = oldHead_j;
        }
        if (direction == 3)                     //  向左移动
        {
                newHead_i = oldHead_i;
                newHead_j = oldHead_j - 1;
```

```
        }
        if (direction == 4)                         //  向右移动
        {
                newHead_i = oldHead_i;
                newHead_j = oldHead_j + 1;
        }

        //  如果新蛇头吃到食物
        if (mp[newHead_i][newHead_j] == -2)
        {
                mp[food_x][food_y] = 0;
                //  产生一个新的食物
                food_x = rand() % (height - 5) + 2;
                food_y = rand() % (width - 5) + 2;
                mp[food_x][food_y] = -2;

                //  原来的旧蛇尾留着，长度自动加 1
        }
        else                                                                    //  否
则，原来的旧蛇尾减掉，保持长度不变
                mp[oldTail_i][oldTail_j] = 0;

        //  小蛇是否和自身碰撞或者和边框撞，游戏失败
        if (mp[newHead_i][newHead_j] > 0 || mp[newHead_i][newHead_j] == -1)
        {
                printf("游戏失败！\n");
                exit(EXIT_FAILURE);
        }
        else
                mp[newHead_i][newHead_j] = 1;}
//  设置初始值 void startUp(){
        //  边框
        for (int i = 0; i < height; i++)
        {
                mp[i][0] = -1;
                mp[i][width - 1] = -1;
        }
        for (int j = 0; j < width; j++)
        {
                mp[0][j] = -1;
                mp[height - 1][j] = -1;
        }
```

```
        //  蛇头
        mp[height  /  2][width  /  2]  =  1;
        //  蛇身
        for  (int  i  =  1;  i  <=  4;  i++)
        {
                mp[height  /  2][width  /  2  -  i]  =  i  +  1;
        }

        //  移动方向
        direction  =  4;

        //  移动速度
        t_cool  =  100;

        //  食物
        food_x  =  rand()  %  (height  -  5)  +  2;
        food_y  =  rand()  %  (width  -  5)  +  2;
        mp[food_x][food_y]  =  -2;
        return;}
//  绘制屏幕 void  show(){
        moveCursor(0,  0);      //  光标每秒归零

        for  (int  i  =  0;  i  <  height;  i++)
        {
                for  (int  j  =  0;  j  <  width;  j++)
                {
                        if  (mp[i][j]  ==  -2)
                                putchar('$');                                   //  食
物
                        else  if  (mp[i][j]  ==  -1)
                                putchar('#');                                   //  边
框
                        else  if  (mp[i][j]  ==  0)
                                putchar('  ');                                  //  空
格
                        else  if  (mp[i][j]  ==  1)
                                putchar('@');                                   //  蛇
头
                        else  if  (mp[i][j]  >  1)
                                putchar('*');                                   //  蛇
身
                }
                putchar('\n');
```

```
        }
        Sleep(t_cool);}
//  无论是否有触发都需要做的事 void  updateWithoutInput(){
        moveSnake();}
//  检测到触发需要做的事 void  updateWithInput(){
        if  (_kbhit())
        {
                char  c  =  _getch();
                switch  (c)  {
                case  'w':
                        direction  =  1;
                        break;
                case  's':
                        direction  =  2;
                        break;
                case  'a':
                        direction  =  3;
                        break;
                case  'd':
                        direction  =  4;
                        break;
                }
        }
        return;}
int  main(void){
        startUp();  //  初始化
        while  (1)
        {
                show();                                         //  绘制屏幕
                updateWithInput();              //  激发事件
                updateWithoutInput();   //  必然事件
        }
        return  0;}
```

1.3.2 使用汇编语言编写贪吃蛇小游戏

1.3.2.1 数据区

我们使用一维数组 mp 实现整张地图，高（20px）和宽（40px）均可调节。direction 记录用户的实时方向。

mp=

-2：食物

-1：边框

0：空白

1：蛇头

2-INF：蛇身（依次递增）

direction=

1：向上

2：向下

3：向左

4：向右

```
.data
;高 20，宽 30
h                       equ         20
w                       equ         40
mp          dword       h           dup(w       dup(0))     ;地图

direction   dword       ?           ;1234：上下左右
food_x      dword       ?
food_y      dword       ?
old_tail    dword       0
old_head    dword       0
new_head    dword       0
max_num                 dword       ?
game_over   dword       0           ;若为 1 则游戏结束

sz_int      byte        "%d ",  0
sz_char     byte        "%c",   0
```

1.3.2.2 确定总体框架

经过 C 语言的实现，我逐渐探索出一套合适的实现思路。主函数如下：

1. 在游戏开始时进行初始化，比如设置移动方向，生成蛇头、蛇身等。

2. 我将整个周期划分为三个子函数，每个周期的间隔时间设置为 50ms（周期越长，速度越慢）。

(1) 首先，对用户的输入进行响应，当用户输入的是 W/A/S/D 时，改变蛇头的方向；如果用户输入的是垃圾字符，则不做改变。

(2) 其次，对一些不需要用户输入部分进行更新，比如我们每个周期都需要移动蛇。

(3) 最后，我们进行画面的展示，这部分包括刷新屏幕以及绘制屏幕。

(4) 另外，还需要进行游戏是否结束的判断。我们设置一个全局变量即可实现。一旦用户撞墙/撞身体，就会立即结束游戏。

```
main        proc        C
            invoke      startUp
            .while  TRUE
                        invoke      updateWithInput
                        invoke      updateWithoutInput
                        invoke      show
                        .if         game_over
                                    .break
                        .endif
                        invoke      sleep
            .endw
            ret
main        endp
```

1.3.2.3 startUp 函数

在这一部分，我们需要对全局变量进行初始化，诸如地图 mp 的边框设置、蛇头的初始位置和蛇身的初始长度、以及初始移动方向、食物位置的设置。

```
startUp     proc        C
            ;绘制边框
            ;invoke     crt_printf, offset      sz_int,     mp[30  *  20  *  4  -  4]
            mov         ecx,        h
            .while      ecx
                        dec         ecx
                        mov         eax,        w*4
                        mul         ecx
                        mov         mp[eax  +  0*4],                    -1
                        mov         mp[eax  +  (w-1)*4],    -1
            .endw
            mov         ecx,        w
            .while      ecx
                        dec         ecx
                        mov         mp[0*4  +  ecx*4],                                      -1
                        mov         mp[(h-1)*w*4  +  ecx*4], -1
            .endw
            ;invoke     crt_printf, offset      sz_int,     mp[0  *  4]                     ;左上角
            ;invoke     crt_printf, offset      sz_int,     mp[(w-1)  *  4]                 ;右上角
            ;invoke     crt_printf, offset      sz_int,     mp[(h-1)  *  w  *  4]   ;左下角
```

```
        ;invoke     crt_printf, offset      sz_int,     mp[h  *  w  *  4  -  4] ;右下角
        ;invoke     crt_printf, offset      sz_int,     mp[h  *  w  *  4]                ;数组界外
        ;invoke     crt_printf, offset      sz_int,     mp[w  *  4]                      ;第二行左一
        ;invoke     crt_printf, offset      sz_int,     mp[w  *  4  +  4]                ;第二行左二

        ;蛇头
        mov         mp[h/2*w*4  +  w/2*4],  1

        ;蛇身
        mov         ecx,        2
        .while      ecx  <  6
                    mov         eax,        ecx
                    dec         eax
                    mov         mp[h/2*w*4  +  w/2*4  +  eax*4],    ecx
                    inc         ecx
        .endw

        ;移动方向
        mov         direction,  4

        ;食物
        invoke      get_rand
        mov         mp[eax*4],  -2
        ret
startUp endp
```

1.3.2.4 UpdateWithInput 函数

在这一部分，我们进行用户输入的读取以及判断。并据此做出相应的动作响应，如移动方向的改变。

注意：调用的 turn 函数是 C 函数。

```
updateWithInput         proc        C
        invoke      turn
        .if         eax
                    mov         direction,  eax
        .endif
        ret
updateWithInput         endp
```

```
int  turn()  {
```

```
        if (_kbhit()) {
                char c = _getch();
                switch (c) {
                case 'w':
                        return 1;
                case 's':
                        return 2;
                case 'a':
                        return 3;
                case 'd':
                        return 4;
                default:
                        return 0;
                }
        }
        return 0;}
```

1.3.2.5 UpdateWithoutInput 函数

在这一部分，我们对每个周期内不需要用户输入的部分进行更新，也就是蛇的移动。

函数的主要流程是：

1. 为每个部分的蛇身递增（蛇身的表示方法见 1.3.2.1 节）。
2. 搜索移动前的蛇头和蛇尾的位置。
3. 根据目前的方向（direction 变量）计算出移动后蛇头的位置。
4. 判断新蛇头是否吃到食物。
   (1) 若没吃到：将原来的蛇尾变为空白，并随即在随机位置产生新的食物。
   (2) 若吃到：原来的蛇尾保留，即蛇身长度+1。
5. 判断是否撞墙/撞身体。
   (1) 若没发生碰撞：将新蛇头的位置设为 1。
   (2) 若发生碰撞：则游戏结束。

注意：调用的 get_rand 函数是 C 函数。

```
updateWithoutInput      proc        C
        invoke      moveSnake
        ret
updateWithoutInput      endp
```

```
moveSnake   proc        C

        ;蛇++
```

```
        mov         ecx,        h
        .while      ecx
                    dec         ecx         ;from  h-1  to  0
                    mov         edx,        w
                    .while      edx
                                dec         edx         ;from  w-1  to  0
                                push        ecx
                                push        edx

                                mov         eax,        w*4
                                            push        edx
                                mul         ecx
                                            pop                     edx
                                mov         ebx,        mp[eax  +  edx*4]

        ;  now,  the  register  "ebx"  is  the  value  of  the  corresponding  point  in  the  mp
                                cmp         ebx,        0
                                jg          Positive1
                                jmp         Negative1
                                Positive1:
                                            inc         ebx
                                            mov         mp[eax  +  edx*4],      ebx
                                Negative1:
                                pop                     edx
                                pop                     ecx
                    .endw
                    push        ecx
                    pop         ecx
        .endw

        ;寻找old_head、old_tail
        mov         max_num,    0
        mov         ecx,        h
        .while      ecx
                    dec         ecx         ;from  h-1  to  0
                    mov         edx,        w
                    .while      edx
                                dec         edx         ;from  w-1  to  0
                                push        ecx
                                push        edx

                                mov         eax,        w*4
                                            push        edx
```

```
                        mul         ecx
                                    pop                     edx
                        mov         ebx,        mp[eax  +  edx*4]

    ;  now,  the  register  "ebx"  is  the  value  of  the  corresponding  point  in  the  mp
                        cmp         ebx,        0
                        jg          Positive2
                        jmp         Negative2
                        Positive2:
                                    push        ebx
                                    .if  ebx  ==  2
                                    mov         ebx,        eax
                                    mov         eax,        4
                                    mul         edx
                                    add         ebx,        eax
                                    mov         old_head,   ebx
                                    .endif
                                    pop                     ebx
                                    .if  max_num  <  ebx
                                    mov         max_num,    ebx
                                    mov         ebx,        eax
                                    mov         eax,        4
                                    mul         edx
                                    add         ebx,        eax
                                    mov         old_tail,   ebx
                                    .endif
                        Negative2:
                        pop                     edx
                        pop                     ecx
            .endw
            push        ecx
            pop         ecx
    .endw

    ;移动蛇，获得 new_head
    mov         eax,        old_head
    .if         direction  ==  1
            add         eax,        w*4
    .elseif     direction  ==  2
            sub         eax,        w*4
    .elseif     direction  ==  3
            add         eax,        4
    .elseif     direction  ==  4
```

```
                sub         eax,        4
        .endif
        mov         new_head,   eax

        ;是否吃到食物
        mov         ebx,        new_head
        mov         eax,        mp[ebx]
        .if         eax  ==  -2
                mov         mp[ebx],    0
                invoke      get_rand
                mov         mp[eax*4],  -2
        .else
                mov         ebx,        old_tail
                mov         mp[ebx],    0
        .endif

        ;游戏是否结束
        mov         ebx,        new_head
        mov         eax,        mp[ebx]
        .if         eax  >  0
                mov         game_over,  1
        .else
                mov         mp[ebx],    1
        .endif

        ;控制蛇的速度
        invoke      sleep

        ret
moveSnake   endp
```

```
int  get_rand()  {
        int  num;
        srand((unsigned)time(NULL));
        num  =  (rand()  %  (20  -  5)  +  2)  *  40  +  (rand()  %  (40  -  5)  +  2);
        return  num;}
```

1.3.2.6 show 函数

首先清除上一轮的显示，然后根据新循环的情况，进行新一轮的屏幕绘制。
在开发过程中发现，如果调用系统清屏函数 system(“cls”)会导致光标发生严重的

闪烁。所以我重写了清屏函数，将清屏动作变为将光标移动到屏幕的左上角，有效解决了光标的闪烁问题。

注意：调用的 moveCursor 是 C 函数。

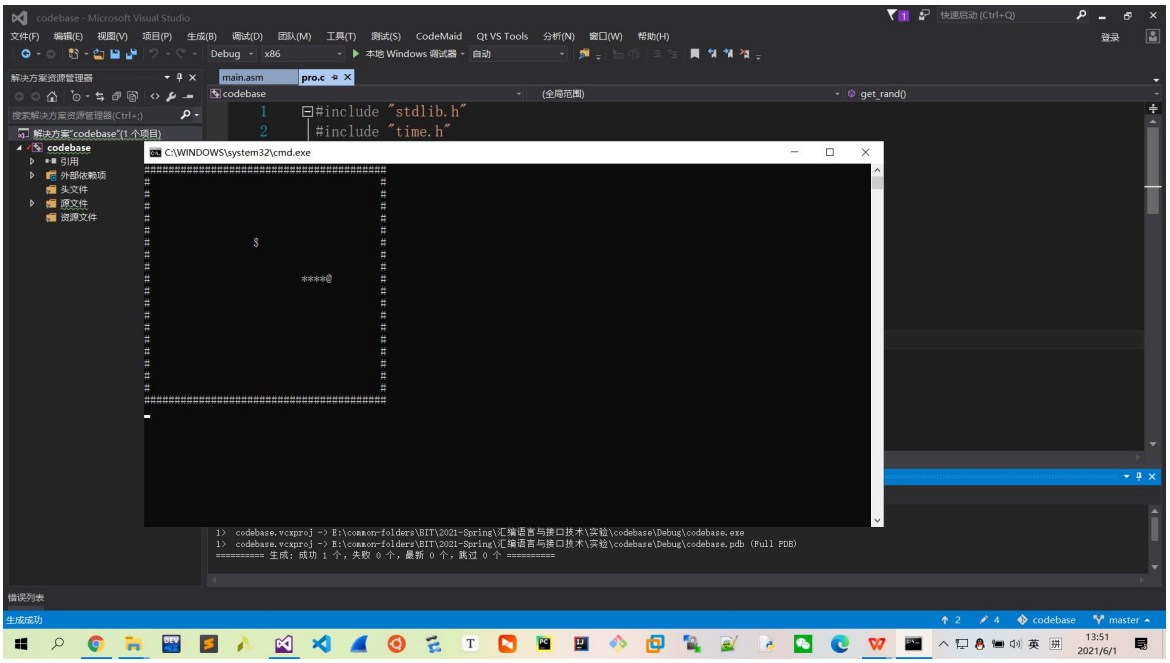
```
show        proc        C
            invoke      moveCursor, offset  0,  offset  0

            mov         ecx,        h
            .while      ecx
                        dec         ecx         ;from  h-1  to  0
                        mov         edx,        w
                        .while      edx
                                    dec         edx         ;from  w-1  to  0
                                    push        ecx
                                    push        edx

                                    mov         eax,        w*4
                                                push        edx
                                    mul         ecx
                                                pop                     edx
                                    mov         eax,        mp[eax  +  edx*4]

            ;  now,  the  register  "eax"  is  the  value  of  the  corresponding  point  in  the  mp
                                    .if  eax  ==  0
                                                invoke      crt_printf, offset      sz_char,    "  "
                                    .elseif     eax  ==  -2
                                                invoke      crt_printf, offset      sz_char,    "$"
                                    .elseif  eax  ==  -1
                                                invoke      crt_printf, offset      sz_char,    "#"
                                    .elseif     eax  ==  1
                                                invoke      crt_printf, offset      sz_char,    "@"
                                    .elseif  eax  >  1
                                                invoke      crt_printf, offset      sz_char,    "*"
                                    .endif

                                    pop                     edx
                                    pop                     ecx
                        .endw
                        push        ecx
                        invoke      crt_printf, offset      sz_char,    0ah
                        pop         ecx
            .endw
```

```
            ret
show        endp
```

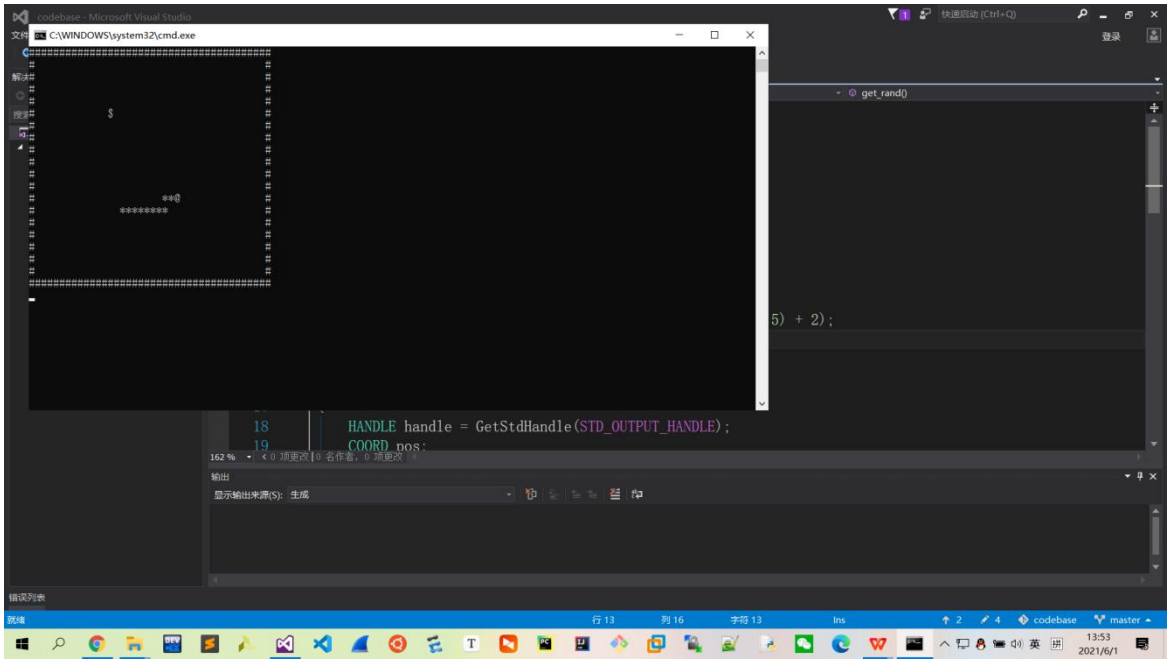
```
void  moveCursor(int  x,  int  y){
            HANDLE  handle  =  GetStdHandle(STD_OUTPUT_HANDLE);
            COORD  pos;
            pos.X  =  x;
            pos.Y  =  y;
            SetConsoleCursorPosition(handle,  pos);
            return;}
```

1.3.3 程序运行截图

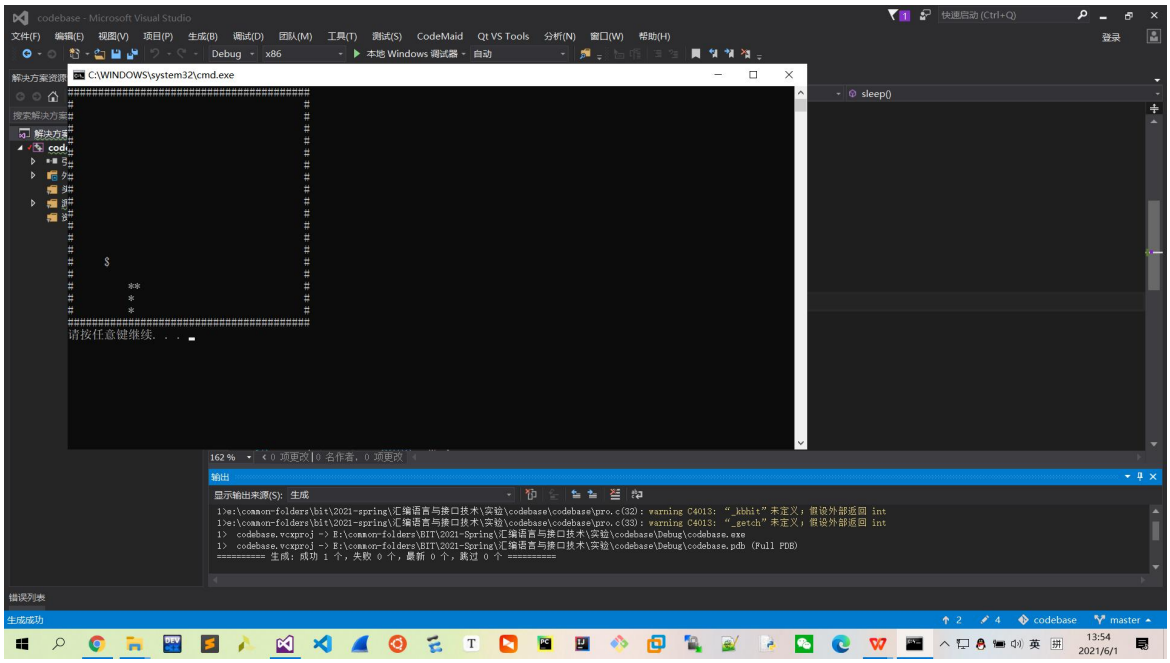
1.3.3.1 初始界面



1.3.3.2 吃到食物，身体变长



1.3.3.3 游戏结束



1.4 实验心得体会

经过这一次的游戏开发，我对多语言混合编程、内存与栈的存储机制都有了更深

刻的认识。

下面是一些开发经验：

1. 注意会改变寄存器的值的指令（如 mul、div 等），在使用前后要将寄存器的值通过寄存器栈保存下来。

2. 注意连续内存的存储方式。

3. 注意不要让食物出现在边界处。

4. 有符号数比较大小时，不能用.if 伪指令，因为寄存器会按照无符号数读取。必须使用 cmp+jg（带符号数比较并跳转指令）的组合。

5. getchar()需要输入回车确认，_getch()不需要。

第三章 开发俄罗斯方块

3.1 实验目的

使用汇编语言编写一个控制台风格的俄罗斯方块小游戏，操作键位是 WASD 键位。

目前包括的功能有：

3. 随机出现方块并下落，左右移动
4. 改变形状
5. 计分和游戏结束

期待加入的功能有：

3. 速度加快和减慢

3.2 实验环境

Visual Studio 2019
masm32

3.3 实验内容

3.3.1 使用 C 语言编写俄罗斯方块小游戏

```
#include <stdio.h>#include <stdlib.h>#include <string.h>#include <windows.h>#include <conio.h>#include <time.h
>


#define HEIGHT 20 #define WEIGHT 18 int Graph[HEIGHT + 2][WEIGHT + 2] = { 0 };int bottom = 1;int
right = 0, left = 0;int score = 0;int endGame = 0;typedef struct {
            int type;
            int top;
            int bot;
            int left;
            int right;}block_state;block_state nowBlock;
 //定义方块，和方块旋转算法 ,方块必须要靠上 int block[19] = {
            0x2222,0xF000,
            0x6600,
            0x2700,0x2320,0x0720,0x2620,  //T 型
            0x3600,0x8C40,
            0xC600,0x4C80,
```

```
        0x4460,0x7400,0x6220,0x2E00,
        0x2260,0x8E00,0x6440,0x7100};int  blockpos[19][4]  =  {//top  bot  left  right
        {  0,3,2,2  },{  0,0,0,3  },
        {  0,1,1,2  },
        {  0,1,1,3  },{  0,2,2,3  },{  1,2,1,3  },{  0,2,1,2  },
        {  0,1,1,3  },{  0,2,0,1  },
        {  0,1,0,2  },{  0,2,0,1  },
        {  0,2,1,2  },{  0,1,1,3  },{  0,2,1,2  },{  0,1,0,2  },
        {  0,2,1,2  },{  0,1,0,2  },{  0,2,1,2  },{  0,1,1,3  }};
//初始化界面 void  initPaint()  {
        int  i,  j;
        for  (i  =  0;  i<WEIGHT  +  2;  i++)  {
                Graph[0][i]  =  1;//第一行
                Graph[HEIGHT  +  1][i]  =  1;
        }
        for  (i  =  0;  i<HEIGHT  +  2;  i++)  {
                Graph[i][0]  =  1;
                Graph[i][WEIGHT  +  1]  =  1;
        }
        /*          for(i  =  0;i<WEIGHT+2;i++){        for(j=0;j<HEIGHT+1;j++){

        }           }           */
}
//绘制界面函数+刷新  void  freshPaint()  {
        system("cls");//清屏问题需要再搞一下
        int  i,  j,  shiftNum  =  0;
        for  (i  =  0;  i<HEIGHT  +  2;  i++)  {
                for  (j  =  0;  j<WEIGHT  +  2;  j++)  {
                        if  (Graph[i][j]  ==  1)  {
                                printf("■");
                        }
                        else  if  (Graph[i][j]  ==  2)  {
                                printf("□");
                        }

        else  if  (i  >=  nowBlock.top&&i  <=  nowBlock.bot&&j  >=  nowBlock.left&&j  <=  nowBlock.right)  {
                                //俄罗斯方块

        shiftNum  =  (4  -  i  +  nowBlock.top  -  1)  *  4  +  (4  -  j  +  nowBlock.left  -  1);
                                if  ((block[nowBlock.type]  >>  shiftNum)  &  1  ==  1)
                                        printf("□");
                                else
                                        printf("  ");
```

```
                                }
                                else {
                                        printf("  ");
                                }
                        }
                        printf("\n");
                }
                printf("得分：%d\n", score);
                if (endGame == 1) {
                        printf("游戏结束！\n");
                }}
//判断是否有满的行并且整体下移 void fullRelease() {
                int i, j;
                int movei, movej;
                int flag = 0;
                for (i = HEIGHT; i>1; i--) {
                        flag = 0;
                        for (j = 1; j<WEIGHT + 1; j++) {
                                if (Graph[i][j] != 2) {
                                        flag = 1;
                                }
                        }
                        if (flag == 0) {
                                //这行消掉，上面的往下掉
                                score++;
                                for (movej = 1; movej <= WEIGHT + 1; movej++) {
                                        Graph[i][j] = 0;
                                }//最上边一行是 0
                                for (movei = i - 1; movei>1; movei--) {
                                        for (movej = 1; movej <= WEIGHT + 1; movej++) {
                                                Graph[movei + 1][movej] = Graph[movei][movej];
                                        }
                                }
                        }
                }
                //在判断是否到达顶端
                for (i = 1; i<WEIGHT; i++) {
                        if (Graph[1][i] == 2) {
                                flag = 2;
                                break;
                        }
                }
                if (flag == 2) {
```

```
                    endGame = 1;
            }}
//旋转方块 void rotate() {
            int reLeft, reTop, reType;
            int flag = 0, shiftNum = 0;
            int i, j;
            switch (nowBlock.type) {
                    //判断能不能转
            case 0:case 1: {
                    //左边不能是墙或者别的 block
                    reType = 1 - nowBlock.type;
                    break;
            }
            case 2: {
                    reType = nowBlock.type;
                    break;
            }
            case 3:case 4:case 5:case 6: {
                    reType = (nowBlock.type - 3 + 1) % 4 + 3;
                    break;
            }
            case 7:case 8: {
                    reType = 15 - nowBlock.type;
                    break;
            }
            case 9:case 10: {
                    reType = 19 - nowBlock.type;
                    break;
            }
            case 11:case 12:case 13:case 14: {
                    reType = (nowBlock.type - 11 + 1) % 4 + 11;
                    break;
            }
            case 15:case 16:case 17:case 18: {
                    reType = (nowBlock.type - 15 + 1) % 4 + 15;
                    break;
            }
            }
            reLeft = nowBlock.left + blockpos[reType][2];
            reTop = nowBlock.top + blockpos[reType][0];
            //搜索上下左右是否有其他障碍物
            for (i = nowBlock.top; i <= nowBlock.top + blockpos[reType][1] && !flag; i++) {
                    for (j = nowBlock.left; j <= nowBlock.left + blockpos[reType][3]; j++) {
```

```
            shiftNum = (4 - i + nowBlock.top - 1) * 4 + (4 - j + nowBlock.left - 1);

            if ((block[reType] >> shiftNum) & 1 == 1 && Graph[i][j - 1] == 2) {
                                        flag = 1;
                                        break;
                                }
                        }
            }
            if (flag == 0) {
                        nowBlock.type = reType;
            }}
//判断左右边界和上下边界 void move(char ch) {
            int shiftNum = 0;
            switch (ch) {
            case 'a':case 'A': {
                        //是左方，要考虑任意块左边是不是有方块
                        //到最左方了
                        if (nowBlock.left + (blockpos[nowBlock.type][2]) <= 1) {

                        }
                        else {
                                    int i, j, flag = 0;
                                    for (j = nowBlock.left; j <= nowBlock.right; j++) {

            for (i = nowBlock.top + blockpos[nowBlock.type][0]; i <= nowBlock.top + blockpos[nowBlock.type]
[1] && !flag; i++) {

            shiftNum = (4 - i + nowBlock.top - 1) * 4 + (4 - j + nowBlock.left - 1);

            if ((block[nowBlock.type] >> shiftNum) & 1 == 1 && Graph[i][j - 1] == 2) {
                                                            flag = 1;
                                                            break;
                                                }
                                        }
                                    }
                                    if (flag == 0) {
                                                nowBlock.right--;
                                                nowBlock.left--;
                                    }
                                    break;
                        }
                        break;
```

```
            }
            case 'd':case 'D': {
                        if (nowBlock.left + blockpos[nowBlock.type][3] >= WEIGHT) {

                        }
                        else {
                                    int i, j, flag = 0;
                                    for (j = nowBlock.left; j <= nowBlock.right; j++) {

            for (i = nowBlock.top + blockpos[nowBlock.type][0]; i <= nowBlock.top + blockpos[nowBlock.type]
[1] && !flag; i++) {

            shiftNum = (4 - i + nowBlock.top - 1) * 4 + (4 - j + nowBlock.left - 1);

            if ((block[nowBlock.type] >> shiftNum) & 1 == 1 && Graph[i][j + 1] == 2) {
                                                                    flag = 1;
                                                                    break;
                                                        }
                                            }
                                    }
                                    if (flag == 0) {
                                                nowBlock.right++;
                                                nowBlock.left++;
                                    }
                                    break;
                        }
                        break;
            }
            case 'S':case 's': {
                        //TODO:这里得考虑已经落下的方块
                        //第一次判断，从下到上判断
                        if ((nowBlock.bot - (4 - blockpos[nowBlock.type][1]) + 1 >= HEIGHT)) { //触底
了

                                    int i, j;
                                    for (i = nowBlock.top; i<HEIGHT + 2 && i <= nowBlock.bot; i++) {

            for (j = nowBlock.left; j<WEIGHT + 2 && j <= nowBlock.right; j++) {

            shiftNum = (4 - i + nowBlock.top - 1) * 4 + (4 - j + nowBlock.left - 1);
                                                        if ((block[nowBlock.type] >> shiftNum) & 1 == 1)
                                                                    Graph[i][j] = 2;
                                            }
```

```
                                }
                                bottom = 1;
                        }
                        else {
                                int i, j, flag = 0;

        for (i = nowBlock.top + blockpos[nowBlock.type][0]; i <= nowBlock.top + blockpos[nowBlock.type]
[1] && !flag; i++) {
                                        for (j = nowBlock.left; j <= nowBlock.right; j++) {

        shiftNum = (4 - i + nowBlock.top - 1) * 4 + (4 - j + nowBlock.left - 1);

        if ((block[nowBlock.type] >> shiftNum) & 1 == 1 && Graph[i + 1][j] == 2) {
                                                        flag = 1;
                                                        break;
                                                }
                                        }
                                }
                                if (flag == 1) {
                                        for (i = nowBlock.top; i <= nowBlock.bot; i++) {

        for (j = nowBlock.left; j <= nowBlock.right; j++) {

        shiftNum = (4 - i + nowBlock.top - 1) * 4 + (4 - j + nowBlock.left - 1);

        if ((block[nowBlock.type] >> shiftNum) & 1 == 1)
                                                                Graph[i][j] = 2;
                                                }
                                        }
                                        bottom = 1;
                                }
                                else {
                                        nowBlock.top++;
                                        nowBlock.bot++;
                                }

                        }
                        break;
                }
                case ' ': {
                        rotate();
                        break;
                }
```

```
        }
        fullRelease();}
//随机出现方块，随机有点问题  void  randomType()  {
        int  ran  =  0,  number  =  0;
        srand(time(0));
        number  =  rand()  %  20;
        nowBlock.type  =  number;//这是个4乘4的block
        nowBlock.top  =  1;
        nowBlock.bot  =  4;
        do  {
                srand(time(0));
                ran  =  rand()  %  WEIGHT;
        }  while  (ran<1  ||  ran  +  3  >=  WEIGHT  +  1);
        //直接按type给出真正的左边和右边
        nowBlock.left  =  ran;
        nowBlock.right  =  ran  +  3;

        //nowBlock.left  =  1;
        //nowBlock.right  =  blockpos[number][3];}
int  main()  {
        char  ch;
        scanf("%c",  ch);
        initPaint();

        while  (!endGame)  {
                if  (bottom  ==  1)  {
                        randomType();
                        bottom  =  0;
                }
                if  (kbhit())  {
                        ch  =  getch();
                        move(ch);
                }
                else  {
                        Sleep(200);
                        move('S');
                        //move('S');
                }
                freshPaint();
                //Sleep(1000);
        }
        getch();
        return  0;}
```

3.3.2 使用汇编语言编写贪吃蛇小游戏

3.3.2.1 数据区

我们使用一张二维数组存储当前布局，并且使用了 HEIGHT/WEIGHT 记录了游戏范围大小。

```
HEIGHT equ 20
WEIGHT equ 18
temp equ WEIGHT+2
Graph byte (HEIGHT+2)dup((WEIGHT+2)dup(0))
bottom DD 1
right DD 0
left DD 0
score DD 0
endGame DD 0
```

对于不同方块的形状的存储，我采用了 16 位二进制数表示它的位置，压缩了存储空间。

```
block DD 2222h,0F000h,
    6600h,
    2700h,2320h,0720h,2620h,
    3600h,8C40h,
    0C600h,4C80h,
    4460h,7400h,6220h,2E00h,
    2260h,8E00h,6440h,7100h
```

使用了一个结构体存储了当前方块的位置，类型等信息

```
block_state STRUC  8
    typenum dd ?
    top dd ?
    bot dd ?
    left dd ?
    right dd ?
    empl dd ?
    emp2 dd ?
    emp3 dd ?
block_state ENDS
nowBlock block_state <>
```

并且对打印内容进行了规定：

```
chget byte 0
msgch byte '%c',0
msgscore byte '得分: %d',0ah,0
msgend byte '游戏结束',0ah,0
wall byte '■',0
blockSign byte '□',0 ;如果方格是下落物体，打印这个
emptySign byte '  ',0 ;如果方格没有东西，则打印这个
enterSign byte 0ah,0
msgright byte 'yes',0ah,0
clearScreen byte 'cls',0
```

确定总体框架

采用了模块化的 C 语言设计方法后，我大概将实现思路分为以下几个阶段：

（1）总体逻辑是先初始化，再根据输入情况对方块进行移动（若未输入则默认自动下落），具体的处理放到几个子函数中

（2）子函数分为重新绘制（freshPaint），移动和接触判断（moveDirect），旋转块(rotate)和每行消除（fullRelease）

3.3.2.2 start 部分

这一部分体现了游戏的逻辑，即总体框架第一条提到的初始化

```
start:
            xor eax,eax
            mov score,eax
            mov eax,1
            mov bottom,eax
            mov ebx,offset Graph
            invoke initPaint
            ;invoke printf,offset msgright
            invoke freshPaint
inLoop:
            cmp dword ptr[endGame],1
            jz endLoop
            cmp dword ptr[bottom],1
            jnz findChar
            ;invoke printf,offset msgright
            invoke randomType
            ;invoke printf,offset msgright
            mov eax,0
            mov dword ptr[bottom],eax
findChar:
            invoke _kbhit
            cmp eax,0
```

```
            jz autoDown ;==0
            invoke _getch
            mov byte ptr[chget],al
            invoke moveDirect
            ;
            jmp freshPic
autoDown:
            invoke _sleep,200
            mov al,'S'
            mov byte ptr[chget],al
            invoke moveDirect
freshPic:
            invoke freshPaint

            jmp inLoop
endLoop:
            invoke _getch
            ret
end start
```

3.3.2.3 randomType 函数

randomType 函数负责产生随机方块（即产生 0-18 内的随机数）和随机方块的左右位置

在这个模块中，我们使用了 rand,srand 两个 C 函数产生随机数

```
randomType proc C
LOCAL ran:dword,number:dword
            mov ebx,offset nowBlock
            xor ebx,ebx
            xor eax,eax
            invoke time,0
            invoke srand,eax
            invoke rand
            mov bx,20
            div bx ;al 里存放 rand 结果
            mov eax,edx
            AND eax,000Fh
            mov nowBlock.typenum,eax
            mov nowBlock.top,1
            mov nowBlock.bot,4
findRand:
```

```
            invoke time,0
            invoke srand,eax
            invoke rand
            mov bx,WEIGHT+2
            div bx
            mov eax,edx
            AND eax,000Fh
            mov ran,eax
            cmp ran,1
            jb findRand ;ran<1
            mov eax,WEIGHT+2
            cmp ran,eax
            jae findRand ;ran+3>=weight+1
            mov eax,ran
            mov nowBlock.left,eax
            ADD eax,3
            mov nowBlock.right,eax
            ret
randomType endp
```

freshPaint函数

freshPaint 函数负责获取 Graph 表的内容并打印，主要有三种块，首先要打印外围的墙，还有内部的空格和空方快（即俄罗斯方块的内容）

```
wall byte '■',0
blockSign byte '□',0 ;如果方格是下落物体，打印这个
emptySign byte '  ',0 ;如果方格没有东西，则打印这个
enterSign byte 0ah,0
```

```
freshPaint proc C
            LOCAL shiftNum:DWORD
            LOCAL imax:DWORD
            LOCAL topm:dword,botm:dword
            LOCAL i:dword,nowpos:dword
            invoke system,offset clearScreen
            ;invoke printf,offset msgright
            xor eax,eax
            mov i,eax
            mov nowpos,eax
            mov eax,HEIGHT+2
            mov ebx,WEIGHT+2
            mul bx
            mov imax,eax
```

```
            mov eax,nowBlock.top
            mov edx,WEIGHT+2
            mul dl
            mov topm,eax

            mov eax,nowBlock.bot
            mov edx,WEIGHT+2
            mul dl
            mov botm,eax
            xor eax,eax
cmpOutLoop:
            cmp eax,imax ;i<HEIGHT+2
            jae printEnd
            ;j=0
            xor ebx,ebx
cmpInLoop:
            cmp ebx,WEIGHT+2 ;j<WEIGHT+2
            jae printInEnd
            ;mov cl,Graph[eax][ebx]
            cmp Graph[eax][ebx],1 ;if == 1
            jnz printBlock
            push eax ;i
            push ebx ;j
            ;现在栈里有 eax（外层循环）,ebx 内层循环
            invoke printf,offset wall
            pop ebx
            pop eax
            INC ebx
            jmp cmpInLoop
printBlock: ;else if == 2
;change
            cmp Graph[eax][ebx],2
            jnz otherCond
            push eax
            push ebx
            invoke printf,offset blockSign
            pop ebx
            pop eax
            INC ebx ;j++
            inc nowpos
            jmp cmpInLoop
otherCond: ;else if
            ;i>=nowBlock.top
```

```
        mov edx,nowBlock.top
        cmp i,edx
        jb defaultCond
        ;i<=nowBlock.bot
        mov edx,nowBlock.bot
        cmp i,edx
        ja defaultCond
        ;j>=nowBlock.left
        cmp ebx,nowBlock.left
        jb defaultCond
        ;j<=nowBlock.right
        cmp ebx,nowBlock.right
        ja defaultCond

        push eax

        xor eax,eax ;count shiftnum
        add eax,4-1 ;-1+4
        add eax,nowBlock.top ;+nowBlock.top
        sub eax,i ;-i

        mov ecx,4
        mul cl;*4

        add eax,4-1 ;+4-1
        add eax,nowBlock.left ;+left
        sub eax,ebx ;-j
        mov shiftNum,eax



        mov ecx,shiftNum
        mov eax,nowBlock.typenum
        ;block[nowBlock.type]
        mov edx,block[eax]
        mov eax,edx
        ;>>shiftnum
        shr eax,cl
        AND eax,1

        pop eax
        jz  defaultCond

        push eax
```

```
            push ebx
            invoke printf,offset blockSign
            pop ebx
            pop eax

            inc ebx
            jmp cmpInLoop
defaultCond: ;else
            push eax
            push ebx
            invoke printf,offset emptySign
            pop ebx
            pop eax
            INC ebx
            inc nowpos
            jmp cmpInLoop ;j++
printInEnd:
            push eax
            invoke printf,offset enterSign
            pop eax
            ADD eax,WEIGHT+2
            mov edx,1
            add i,edx
            xor ebx,ebx
            jmp cmpOutLoop
printEnd:
            invoke printf,offset msgscore,score
            cmp endGame,0
            jz proend
            invoke printf,offset msgend
proend:
            ret
            freshPaint endp
```

3.3.2.4 initPaint 函数

initPaint 函数主要通过给最外围的 Graph 数组赋值，结合 freshPaint 打印函数来实现外围边框（墙）的确定

```
initPaint proc C
  LOCAL i:dword
  LOCAL j:dword
L0:
```

```
            mov i,0
            jmp L2
L1:
          inc i
L2:
            cmp i,14h
            jge L4
L3:
            mov ecx,i
            mov Graph[ecx],1
            mov Graph[(HEIGHT+1)*(WEIGHT+2)+ecx],1
            jmp L1
L4:
            mov i,0
            jmp L6
L5:
          inc i
L6:
            cmp i,16h
            jge L8
L7:
            mov eax,WEIGHT+2
            mov ebx,i
            mul ebx
            mov Graph[eax+0],1
            mov Graph[eax+WEIGHT+1],1
            jmp L5
L8:

 ret
initPaint endp
```

3.3.2.5 moveDirect 函数

moveDirect 函数主要根据其所接受的输入参数跳转到四个部分：负责左移的 TypeL，负责右移的 typeR，负责下落的 typeQ 和负责变形的 typeC。其中，左移和右移分别要考虑是否接触到墙的情况，如果左/右移后不会超出边框，才能对 nowBlock 块进行操作；下落则在此基础上要考虑是否接触到底边或已落下的其他方块，如果接触，则将 Graph 表中的对应位置赋值，配合 freshPaint 函数将方块打印；变形调用 rotate 函数进行实现

```
moveDirect  proc  C
LOCAL  shiftNum:DWORD
```

```
LOCAL  i,j,flag:DWORD
            mov  shiftNum,0

            mov  bl,chget
            cmp  bl,'a'
            jz  typeL
            cmp  bl,'A'
            jz  typeL

            cmp  bl,'d'
            jz  typeR
            cmp  bl,'D'
            jz  typeR

            cmp  bl,'s'
            jz  typeQ
            cmp  bl,'S'
            jz  typeQ

            cmp  bl,'w'
            jz  typeC
            cmp  bl,'W'
            jz  typeC



typeL:
                        xor  eax,eax
                        xor  ebx,ebx
                        xor  ecx,ecx
                        mov  eax,4
                        mov  ebx,nowBlock.typenum
                        mul  ebx
                        mov  bl,blockpos[eax+2]
                        mov  eax,nowBlock.left
                        add  eax,ebx

                        cmp  eax,1
                        jg  L1
L0:
                        jmp  endCase
L1:
                        mov  flag,0
```

```
                    mov  eax,nowBlock.left
                    mov  j,eax
                    jmp  L3
L2:
                    inc  j
L3:
                    mov  eax,nowBlock.right
                    cmp  j,eax
                    jg  L11
L4:
                    mov  eax,4
                    mov  ebx,nowBlock.typenum
                    mul  ebx
                    mov  bl,blockpos[eax+0]
                    mov  eax,nowBlock.top
                    add  eax,ebx
                    mov  i,eax
                    jmp  L6
L5:
                    inc  i
L6:
                    mov  eax,4
                    mov  ebx,nowBlock.typenum
                    mul  ebx
                    mov  bl,blockpos[eax+1]
                    mov  eax,nowBlock.top
                    add  eax,ebx

                    cmp  i,eax
                    jg  L10

                    cmp  flag,0
                    jnz  L10
L7:
                    mov  eax,nowBlock.top
                    add  eax,4
                    sub  eax,i
                    sub  eax,1

                    mov  ebx,4
                    mul  ebx
                    mov  ecx,eax
```

```
                    mov  eax,nowBlock.left
                    add  eax,4
                    sub  eax,i
                    sub  eax,1

                    add  ecx,eax
                    mov  shiftNum,ecx

                    mov  eax,nowBlock.typenum
                    mov  ax,block[eax]
                    sar  eax,cl
                    and  eax,1
                    cmp  eax,1
                    jnz  L9

                    mov  eax,WEIGHT+2
                    mov  ebx,i
                    mul  ebx
                    mov  ebx,j
                    sub  ebx,1

                    cmp  Graph[eax+ebx],2
                    jnz  L9
L8:
                    mov  flag,1
                    jmp  L2
L9:
                    jmp  L5
L10:
                    jmp  L2
L11:
                    cmp  flag,0
                    jnz  L13
L12:
                    dec  nowBlock.right
                    dec  nowBlock.left
L13:
                    jmp  endCase



typeR:  ;右侧
                    xor  eax,eax
                    xor  ebx,ebx
```

```
                    xor  ecx,ecx
                    mov  eax,4
                    mov  ebx,nowBlock.typenum
                    mul  ebx
                    mov  bl,blockpos[eax+3]
                    mov  eax,nowBlock.left
                    add  eax,ebx
                    cmp  eax,WEIGHT
                    jl  R1
R0:
                    jmp  endCase
R1:
                    mov  flag,0
                    mov  eax,nowBlock.left
                    mov  j,eax
                    jmp  R3
R2:
                    inc  j
R3:
                    mov  eax,nowBlock.right
                    cmp  j,eax
                    jg  R11
R4:
                    mov  eax,4
                    mov  ebx,nowBlock.typenum
                    mul  ebx
                    mov  bl,blockpos[eax+0]
                    mov  eax,nowBlock.top
                    add  eax,ebx
                    mov  i,eax
                    jmp  R6
R5:
                    inc  i
R6:
                    mov  eax,4
                    mov  ebx,nowBlock.typenum
                    mul  ebx
                    mov  bl,blockpos[eax+1]
                    mov  eax,nowBlock.top
                    add  eax,ebx

                    cmp  i,eax
                    ja  R10
```

```
                    cmp  flag,0
                    jnz  R10
R7:
                    mov  eax,nowBlock.top
                    add  eax,4
                    sub  eax,i
                    sub  eax,1

                    mov  ebx,4
                    mul  ebx
                    mov  ecx,eax

                    mov  eax,nowBlock.left
                    add  eax,4
                    sub  eax,j  ;这错了
                    sub  eax,1

                    add  ecx,eax
                    mov  shiftNum,ecx

                    mov  eax,nowBlock.typenum
                    mov  ax,block[eax]
                    sar  eax,cl
                    and  eax,1
                    cmp  eax,1
                    jnz  R9

                    mov  eax,WEIGHT+2
                    mov  ebx,i
                    mul  ebx
                    mov  ebx,j
                    inc  ebx
                    cmp  Graph[eax+ebx],2
                    jnz  R9
R8:
                    mov  flag,1
                    jmp  R2
R9:
                    jmp  R5
R10:
                    jmp  R2
R11:
```

```
                        cmp  flag,0
                        jnz  R13
R12:
                        inc  nowBlock.right
                        inc  nowBlock.left
R13:
                        jmp  endCase

typeQ:
                        ;+blockpos[nowBlock.type][1]
                        mov  eax,4
                        mov  ebx,nowBlock.typenum
                        mul  bx
                        mov  ecx,eax
                        add  ecx,1
                        mov  al,blockpos[ecx]
                        ;+nowBlock.bot
                        add  eax,nowBlock.bot
                        ;-3
                        sub  eax,3
                        cmp  eax,HEIGHT
                        jb  Q10
;if
                        ;i=nowBlock.top
                        mov  eax,nowBlock.top
                        mov  i,eax
                        jmp  Q1
Q0:
                        inc  i
Q1:
                        cmp  i,HEIGHT+2
                        jge  Q9
                        ;i>nowBlock.bot
                        mov  eax,nowBlock.bot
                        cmp  i,eax
                        jg  Q9
Q2:
                        mov  eax,nowBlock.left
                        mov  j,eax
                        jmp  Q4
Q3:
                        inc  j
Q4:
```

```
                    cmp  j,WEIGHT+2
                    jge  Q8

                    mov  eax,nowBlock.right
                    cmp  j,eax
                    jg  Q8
Q5:
                    mov  eax,nowBlock.top
                    add  eax,3
                    sub  eax,i

                    mov  ebx,4
                    mul  bx

                    add  eax,nowBlock.left
                    add  eax,4
                    sub  eax,j
                    sub  eax,1

                    mov  shiftNum,eax
                    mov  ecx,shiftNum
                    mov  ebx,nowBlock.typenum
                    mov  ax,block[ebx]
                    shr  eax,cl  ;SAR 不对，因为不能当作带符号数处理
                    AND  eax,1
                    cmp  eax,1
                    jnz  Q7
Q6:
                    mov  eax,WEIGHT+2
                    mov  ebx,i
                    mul  bx
                    mov  ebx,j
                    mov  Graph[eax+ebx],2
Q7:
                    jmp  Q3
Q8:
                    jmp  Q0
Q9:
                    mov  bottom,1
                    jmp  endCase

;else
Q10:
```

```
                        mov  flag,0
                        mov  i,0
                        mov  j,0
Q11:  ;i=nowBlock.top+blockpos[nowBlock.type][0]
                        mov  eax,4
                        mov  ebx,nowBlock.typenum
                        mul  bx
                        mov  bl,blockpos[eax+0]
                        mov  eax,nowBlock.top
                        add  eax,ebx
                        mov  i,eax
                        jmp  Q13
Q12:
                        inc  i
Q13:        ;i<=nowBlock.top+blockpos[nowBlock.type][1]
                        mov  eax,4
                        mov  ebx,nowBlock.typenum
                        mul  ebx
                        mov  bl,blockpos[eax+1]
                        mov  eax,nowBlock.top
                        add  eax,ebx

                        cmp  i,eax
                        jg  Q21
                        ;!flag
                        cmp  flag,0
                        jnz  Q21
Q14:
                        mov  eax,nowBlock.left
                        mov  j,eax
Q15:
                        inc  j
Q16:
                        mov  eax,nowBlock.right
                        cmp  j,eax
                        jg  Q20
Q17:
                        mov  eax,nowBlock.top
                        add  eax,3
                        sub  eax,i

                        mov  ebx,4
                        mul  bx
```

```
                    add  eax,nowBlock.left
                    add  eax,4
                    sub  eax,j
                    sub  eax,1
                    ;(block[nowBlock.type]>>shiftNum)&1  ==  1
                    mov  shiftNum,eax
                    mov  ecx,shiftNum
                    mov  ebx,nowBlock.typenum
                    mov  ax,block[ebx]
                    shr  eax,cl
                    and  eax,1
                    cmp  eax,1
                    jnz  Q19
                    ;Graph[i+1][j]==2
                    mov  eax,WEIGHT+2
                    mov  ebx,i
                    inc  ebx
                    mul  bx
                    mov  ebx,j
                    cmp  Graph[eax+ebx],2
                    ;!=2 跳转到 j++
                    jnz  Q19
Q18:;flag=1;break;
                    mov  flag,1
                    jmp  Q21
Q19:
                    jmp  Q15
Q20:
                    jmp  Q12

Q21:
                    cmp  flag,1
                    jnz  Q32
;if                  ;i=nowBlock.top
                    mov  eax,nowBlock.top
                    mov  i,eax
                    jmp  Q23
Q22:
                    inc  i
Q23:
                    mov  eax,nowBlock.bot
                    cmp  i,eax
```

```
                    jg  Q31
Q24:        ;j=nowBlock.left
                    mov  eax,nowBlock.left
                    mov  j,eax
                    jmp  Q26
Q25:
                    inc  j
Q26:
                    mov  eax,nowBlock.right
                    cmp  j,eax
                    jg  Q30
Q27:
                    mov  eax,nowBlock.top
                    add  eax,4
                    sub  eax,i
                    sub  eax,1

                    mov  ebx,4
                    mul  bx

                    add  eax,nowBlock.left
                    add  eax,4
                    sub  eax,j
                    sub  eax,1

                    mov  shiftNum,eax
                    mov  ecx,shiftNum

                    mov  ebx,nowBlock.typenum
                    mov  ax,block[ebx]
                    shr  eax,cl
                    and  eax,1
                    cmp  eax,1
                    jnz  Q29
Q28:
                    mov  eax,WEIGHT+2
                    mov  ebx,i
                    mul  ebx
                    mov  ebx,j
                    mov  Graph[eax+ebx],2
Q29:
                    jmp  Q25
Q30:
```

```
                        jmp  Q22
Q31:;每次 bottom 很奇怪
                        mov  eax,1
                        mov  dword  ptr[bottom],eax
                        jmp  endCase
;else
Q32:
                        inc  nowBlock.top
                        inc  nowBlock.bot
                        ;mov  eax,1
                        ;mov  dword  ptr[init],eax
                        jmp  endCase

typeC:
                        invoke  rotate
                        jmp  endCase

endCase:
                        invoke  fullRelease
                        ret
moveDirect  endp
```

3.3.2.6 rotate 函数

rotate 函数由 moveDirect 函数调用，负责实现 nowBlock 的变形功能。函数首先根据当前方块的类型进行判断，在当前块周围尚未已被其他方块占用时，将新方块赋值给原方块

```
rotate  proc  C
LOCAL  reLeft,reTop,reType:dword
                        LOCAL  flag,shiftNum:dword
                        LOCAL  i,j:dword

                        mov  flag,0
                        mov  shiftNum,0

                        mov  ebx,nowBlock.typenum

                        cmp  ebx,1
                        jle  type0

                        cmp  ebx,2
                        jle  type1
```

```
                    cmp  ebx,6
                    jle  type2

                    cmp  ebx,8
                    jle  type3

                    cmp  ebx,10
                    jle  type4

                    cmp  ebx,14
                    jle  type5

                    cmp  ebx,18
                    jle  type6

type0:
                    mov  eax,1
                    sub  eax,nowBlock.typenum
                    mov  reType,eax
                    jmp  L0
type1:
                    mov  eax,nowBlock.typenum
                    mov  reType,eax
                    jmp  L0
type2:
                    mov  eax,nowBlock.typenum
                    sub  eax,2
                    xor  edx,edx
                    mov  ebx,4
                    div  ebx
                    add  edx,3
                    mov  reType,edx
                    jmp  L0
type3:
                    mov  eax,15
                    sub  eax,nowBlock.typenum
                    mov  reType,eax
                    jmp  L0
type4:
                    mov  eax,19
                    sub  eax,nowBlock.typenum
                    mov  reType,eax
```

```
                    jmp  L0
type5:
                    mov  eax,nowBlock.typenum
                    sub  eax,10
                    xor  edx,edx
                    mov  ebx,4
                    div  ebx
                    add  edx,11
                    mov  reType,edx
                    jmp  L0
type6:
                    mov  eax,nowBlock.typenum
                    sub  eax,14
                    xor  edx,edx
                    mov  ebx,4
                    div  ebx
                    add  edx,15
                    mov  reType,edx
                    jmp  L0
L0:
                    mov  eax,4
                    mov  ebx,reType
                    mul  ebx
                    mov  bl,blockpos[eax+2]
                    mov  eax,nowBlock.left
                    add  eax,ebx
                    mov  reLeft,eax

                    mov  eax,4
                    mov  ebx,reType
                    mul  ebx
                    mov  bl,blockpos[eax+0]
                    mov  eax,nowBlock.top
                    add  eax,ebx
                    mov  reTop,eax

                    mov  eax,nowBlock.top
                    mov  i,eax
                    jmp  L2
L1:
                    inc  i
L2:
                    cmp  flag,0
```

```
                    jnz  L10

                    mov  eax,4
                    mov  ebx,reType
                    mul  ebx
                    mov  bl,blockpos[eax+1]
                    mov  eax,nowBlock.top
                    add  eax,ebx
                    cmp  j,eax
                    jg  L10
L3:
                    mov  eax,nowBlock.left
                    mov  j,eax
                    jmp  L5
L4:
                    inc  j
L5:
                    mov  eax,4
                    mov  ebx,reType
                    mul  ebx
                    mov  bl,blockpos[eax+3]
                    mov  eax,nowBlock.left
                    add  eax,ebx
                    cmp  j,eax
                    jg  L9
L6:
                    mov  eax,nowBlock.top-1
                    add  eax,4
                    sub  eax,i
                    mov  ebx,4
                    mul  ebx
                    mov  ecx,eax

                    mov  eax,nowBlock.left-1
                    add  eax,4
                    sub  eax,i

                    add  ecx,eax
                    mov  shiftNum,ecx

                    mov  eax,reType
                    mov  ax,block[eax]
                    shr  eax,cl
```

```
                    and  eax,1
                    cmp  eax,1
                    jnz  L8

                    mov  eax,WEIGHT+2
                    mov  ebx,i
                    mul  ebx
                    mov  ebx,j
                    sub  ebx,1
                    cmp  Graph[eax+ebx],2
                    jnz  L8
L7:
                    mov  flag,1
                    jmp  L1
L8:
                    jmp  L4
L9:
                    jmp  L1
L10:
                    cmp  flag,0
                    jnz  L12
L11:
                    mov  eax,reType
                    mov  nowBlock.typenum,eax
L12:
                    ret
rotate  endp
```

3.3.2.7 fullRelease 函数

FullRelease 函数实现的是俄罗斯方块的核心玩法：满行消去并下坠。从下向上进行判断，若 Graph 数组的某一行已全部被填满，则将第一行赋值为零，第二行开始到该行之间的其他行号+1，实现下坠。此外，如果 Graph 表第一行某值已被赋值为 2，则说明方块触底，游戏结束。

```
fullRelease  proc  C
           LOCAL  i,j:dword
           LOCAL  movei,movej:dword
           LOCAL  flag:dword

           mov  flag,0
L0:
```

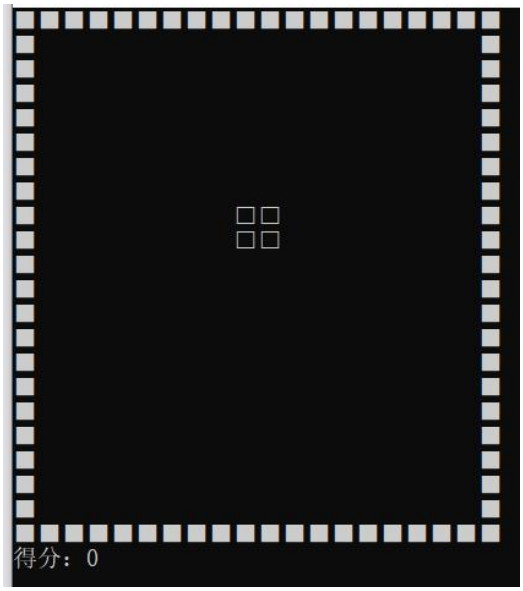
```
            mov  i,HEIGHT
            jmp  L2
L1:
            dec  i
L2:
            cmp  i,1
            jle  L23
L3:
            mov  flag,0
L4:
            mov  j,1
            jmp  L6
L5:
            inc  j
L6:
            cmp  j,WEIGHT+1
            jge  L9
L7:
            mov  eax,WEIGHT+2
            mov  ebx,i
            mul  ebx
            mov  ebx,j

            cmp  Graph[eax+ebx],2
            jz  L8

            mov  flag,1
L8:
            jmp  L5
L9:
            cmp  flag,0
            jnz  L22

            inc  score
L10:
            mov  movej,1
            jmp  L11
L11:
            inc  movej
L12:
            cmp  movej,WEIGHT+1
            jg  L14
L13:
```

```
            mov  eax,WEIGHT+2
            mov  ebx,i
            mul  ebx
            mov  ebx,j
            mov  Graph[eax+ebx],0
            jmp  L11
L14:
            mov  eax,i-1
            mov  movei,eax
            jmp  L16
L15:
            dec  movei
L16:
            cmp  movei,1
            jle  L22
L17:
            mov  movej,1
            jmp  L19
L18:
            inc  movej
L19:
            cmp  movej,WEIGHT+1
            jg  L21
L20:
            mov  eax,WEIGHT+2
            mov  ebx,movei
            mul  ebx
            mov  ebx,movej
            mov  cl,Graph[eax+ebx]

            mov  eax,WEIGHT+2
            mov  ebx,movei+1
            mul  ebx
            mov  ebx,movej
            mov  Graph[eax+ebx],cl

            jmp  L18
L21:
            jmp  L15
L22:
            jmp  L1
L23:
            mov  i,1
```

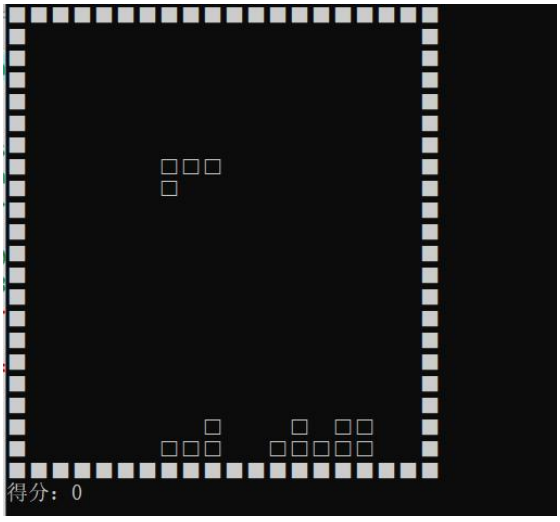
```
            jmp  L25
L24:
            inc  i
L25:
            cmp  i,WEIGHT
            jge  L28
L26:
            mov  eax,i
            cmp  Graph[(WEIGHT+2)+eax],2
            jnz  L27

            mov  flag,2
            jmp  L28
L27:
            jmp  L24
L28:
            cmp  flag,2
            jnz  L29
            cmp  init,0
            jz  L29
            mov  endGame,1
L29:
            ret
fullRelease  endp
```

3.4.2 程序运行截图

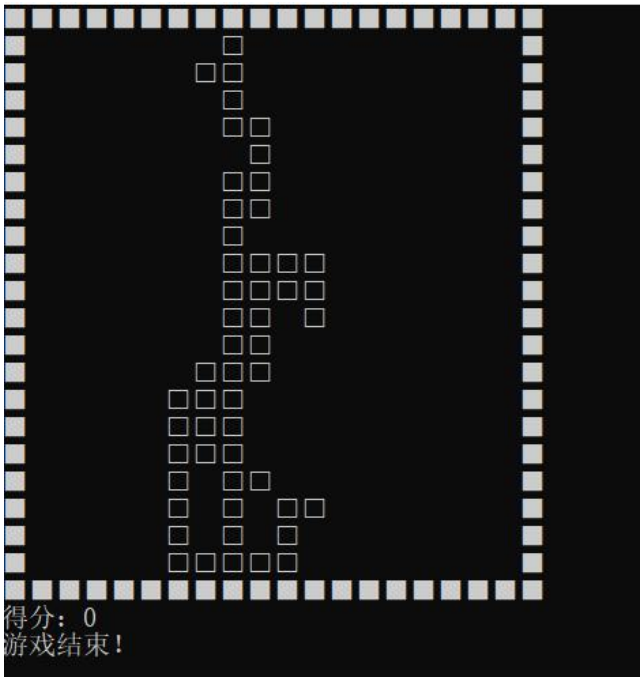
3.4.2.1 初始界面



3.4.2.2 落到地面上堆积



3.4.2.3 游戏结束



3.5 实验心得体会

经过这一次的游戏开发，我们对汇编模块化编程有了一些了解。

下面是一些开发经验：

1.DW/DD/DB 不是简单的和 int/char 等对应的，因为使用了 DD，我们发现将对应的块的代码完成后，取到寄存器中时，并不是占据了低 16 位，而是在各个地方散布着。应该根据数据的长度分配它的空间。

2.内外层逻辑需要条理清晰，比如跳转。

3.对于 Graph[i][j]，我们以为其中的 i 和 C 语言二维数组的 i 表示方法一致，但实际上在汇编中 Graph[i][j]=Graph[i+j]，而 C 语言中的 Graph[i][j]=Graph[i*每行长度+j]，所以需要自己对 i 进行处理，以便满足汇编的需求。

4.汇编语言中，一些常见指令的用法如 div 等与 c 语言有较大差别，需要掌握好指令的格式及功能，以及指令执行对响应寄存器的影响，才能较好地完成转汇编工作。

第四章 开发飞机游戏

4.1 实验目的

使用汇编语言编写一个控制台风格的飞机小游戏，操作键位是 WASD 键位。目前包括的功能有：

1、 随机出现障碍物并下落，阻挡飞机飞行
2、 左右移动操控飞机，并向怪物开火

期待加入的功能有：

3、 怪物种类增加
4、 每一关卡有大 BOSS
5、 飞机开火数量可调节

4.2 实验环境

Visual Studio 2019
masm32

4.3 实验内容

4.3.1 使用 C 语言编写飞机小游戏

```c
#include<stdio.h>
#include<windows.h>
#include<conio.h>
//定义全局变量
int high,width;                    //定义边界
int position_x,position_y;         //飞机位置
int bullet_x,bullet_y;             //子弹位置
int enemy_x,enemy_y;
int score;
int flag;                          //飞机状态
void gotoxy(int x,int y)           //光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
```

```
    SetConsoleCursorPosition(handle,pos);
}
void HideCursor() // 用于隐藏光标
{
    CONSOLE_CURSOR_INFO cursor_info = {1, 0};  // 第二个值为 0 表示隐藏光标
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursor_info);
}


void startup()                          //数据初始化
{
    high=18;
    width=26;

    position_x=high-3;
    position_y=width/2;

    bullet_x=0;
    bullet_y=position_y;

    enemy_x=0;
    enemy_y=position_y;

    score=0;

    flag=0;                             //飞机完好

    HideCursor();
}
void show()                             //显示界面
{
    int i,j;
    for(i=0;i<high;i++)
    {
        for(j=0;j<width;j++)
        {
            if(flag)
                break;

            else if((i==position_x)&&(j==position_y))        //飞机坐标
                printf("*");
            else if((i==enemy_x)&&(j==enemy_y))              //敌机坐标
```

```
                printf("*");
            else if((i==bullet_x)&&(j==bullet_y))           //子弹坐标
                    printf("|");
            else if ((j==width-1)||(i==high-1)||(j==0)||(i==0))
 //打印边界
                printf("#");
            else
                printf(" ");
        }
        printf("\n");
    }
    printf("\n");
    if((position_x==enemy_x)&&(position_y==enemy_y))
    {
        flag=1;                             //飞机撞毁 游戏结束
        printf("得分: %d\n",score);
        printf("游戏结束");
    }
    else
        printf("得分: %d\n",score);
}
void withoutInpute()                            //与用户输入无关
{
    if(bullet_x>0)                              //子弹上升效果
        bullet_x--;
    if((bullet_x==enemy_x)&&(bullet_y==enemy_y))        //子弹命中敌机
    {
        score++;
        bullet_x=-1;
        enemy_x=1;
        enemy_y=2+rand()%width-2;
    }


    static int speed;
    if(speed<30)                    //减慢敌机速度，不影响飞机和子弹速度
        speed++;
    if(speed==30)
    {
        if(enemy_x<high)
            enemy_x++;
        else
```

```
        {
            enemy_x=0;
            enemy_y=2+rand()%width-2;
        }
        speed=0;
    }




}
void withInpute()                       //与用户输入有关
{
    char input;
    if(kbhit())                                     //控制飞机方向
    {
        input=getch();
        if((input=='w')&&position_x>1)
            position_x--;
        if((input=='s')&&position_x<high-2)
            position_x++;
        if((input=='a')&&position_y>1)
            position_y--;
        if((input=='d')&&position_y<width-2)
            position_y++;
        if(input==' ')
        {
            bullet_x=position_x-1;
            bullet_y=position_y;
        }
    }
}
int main()
{
    system("color 2f");
    startup();                  // 数据初始化
    while(1)                    //  游戏循环执行
    {
        gotoxy(0,0);
        show();                 // 显示画面
        withoutInpute();        // 与用户输入无关的更新
        withInpute();           // 与用户输入有关的更新
```

```
    }
}
```

4.3.2 使用汇编语言编写俄罗斯方块小游戏

4.3.2.1 数据区

我们使用一张二维数组存储当前布局，并且定义了 height 和 width 记录了游戏的范围大小。

```
.data
height byte 00H        ;设置显示方式为 320*200 彩色图形方式
width byte 04H
```

4.3.2.2 start 部分

这一部分体现了游戏的总体实现逻辑，各个函数之间的调用与执行顺序，需要首先对画布进行初始化。然后画出飞机，使飞机左右移动，然后再执行开火功能。但是这里并没有一个总的函数进行封装，都放在主函数里。

```
           mov al,34h   ; 设控制字值
        out 43h,al   ; 写控制字到控制字寄存器
        mov ax,0ffffh ; 中断时间设置
        out 40h,al   ; 写计数器 0 的低字节
        mov al,ah    ; AL=AH
        out 40h,al   ; 写计数器 0 的高字节


    xor ax,ax           ; AX = 0
    mov ds,ax           ; DS = 0
    mov word ptr ds:[20h],offset Timer  ; 设置时钟中断向量的偏移地址
    mov ax,cs
    mov word ptr ds:[22h],ax        ; 设置时钟中断向量的段地址=CS

lop3:
        call play_plane1    ;擦除飞机轨迹
        call play_plane     ;画飞机
        mov cx,bx
        mov dx,bp
again:
    mov ah,01      ;检测是否有按键，没有的话循环检测
    int 16h
    jz again        ;没有按键，显示移动，再次检测
```

```
        ;从键盘读入字符
        mov ah,0H
        int 16H

        ;判断字符
        cmp ah,72
        je up
        cmp ah,80
        je down
        cmp ah,75
        je left
        cmp ah,77
        je right
    cmp ah,57   ;空格
    je shoot
        cmp ah,16   ;Q 退出
        je quite
        jmp lop3

up:     sub bp,3
        jmp lop3
down:   add bp,3
        jmp lop3
left:   sub bx,3
        jmp lop3
right:  add bx,3
        jmp lop3

shoot:
    call shoot_plane
    jmp lop3

;退出程序
quite:
    mov ah,4ch
    int 21h



Timer:
    push ax
    mov al,byte ptr ds:[timecontrol]   ;timecontrol 为设定的敌人移动速度
```

```
    cmp byte ptr ds:[delay_timer],al   ;delay_timer等于timecontrol时才移动敌
人,否则本次中断不做任何事
    pop ax
    jnz goout
    mov byte ptr ds:[delay_timer],0
    call move_smile
    call play_smile     ;画笑脸
goout:
    inc byte ptr [delay_timer]
    push ax
    mov al,20h          ; AL = EOI
    out 20h,al          ; 发送 EOI 到主 8529A
    out 0A0h,al         ; 发送 EOI 到从 8529A
    pop ax
    iret            ; 从中断返回
```

4.3.2.3 sp_line 函数

此函数是一个画线函数，也就是将点连接成线，因为飞机都是像素点的堆叠，因此，只需要将几个连接成几条线便能画出飞机。

```
    ;画水平直线
;入口参数 CX 相当于 X0 DX 相当于 Y0,Y1 si 图像长度 BL 像素

sp_line proc
        pusH ax
        pusH bx
        MOV BL,2    ;飞机的颜色
        MOV AH,0cH
        MOV AL,BL
lop:    INT 10H
        inc CX
        dec si

        jnz lop
        pop bx
        pop ax
        ret
sp_line endp
```

4.3.2.4 play_plane 函数

`　由上面的画线函数，只需要经过线条的堆叠就可以形成一个面，也就是真正画出飞机，所以，此函数的功能就是画出一个飞机在画板上。

```
;//
;//画玩家飞机子程序 传入参数 bx 设置飞机的水平位置 BP 设置飞机的垂直位置   BX,BP 记录飞机的位置
play_plane proc
    push cx
    push dx
    push es
    push si
    push di
    push ax
    jmp sk

play_plane_1: dw 6,1,1,5,2,3,5,3,3,5,4,3,4,5,5,3,6,7,1,7,11,1,8,11,4,9,5,5,10,3,4,11,5,3,12,7,4,13,2,7,13,2 ;X0,Y,长度

sk:
    mov cx,ax
    mov ax,cs
    mov es,ax
    mov di,0

lop2:
    mov cx,word ptr es:[play_plane_1+di]    ;x0
        add cx,bx
        mov dx,word ptr es:[play_plane_1+di+2]   ;y
        add dx,bp
        mov si,word ptr es:[play_plane_1+di+4]    ;长度

        call sp_line
        add di,6
        cmp di,84
        jne lop2

    ;plane_pos 用于记录飞机的位置，此处更新飞机位置
    mov ds:[plane_pos],bx
        mov ds:[plane_pos+2],bp

        pop ax
```

```
        pop di
        pop si
        pop es
        pop dx
        pop cx

        ret
play_plane endp
```

4.3.2.5 play_plane1 函数

此函数的主要功能是实现飞机的移动，但是其中会包含一个擦除函数，因为飞机在从旧位置移动到新位置后，需要在新位置画出飞机，上面已有函数实现，但同时也需要擦除旧位置的飞机。sp_line1 函数就是执行擦除飞机的功能。

```
;画水平直线
;入口参数 CX 相当于 X0 DX 相当于 Y0,Y1 si 图像长度 BL 像素
sp_line1 proc
         pusH ax
         pusH bx
         pusH bp
         pusH di
         MOV bp,CX

         MOV di,11
         MOV BL,0     ;飞机的颜色 用来擦除原来的飞机
         MOV AH,0cH
         MOV AL,BL
lop1:    INT 10H
         inc CX
         dec di

         jnz lop1
         MOV CX,bp

         pop di
         pop bp
         pop bx
         pop ax
         ret
sp_line1 endp
```

```
play_plane1 proc ;擦除飞机轨迹子程序 传入参数 CX,DX

      push si
      push di

      inc cx
      mov si,13

      mov di,0
lop5: inc di
      inc dx
      call sp_line1
      cmp di,14
      jne lop5
      pop di
      pop si

      ret
play_plane1 endp
```

4.3.2.6 shoot_plane 函数

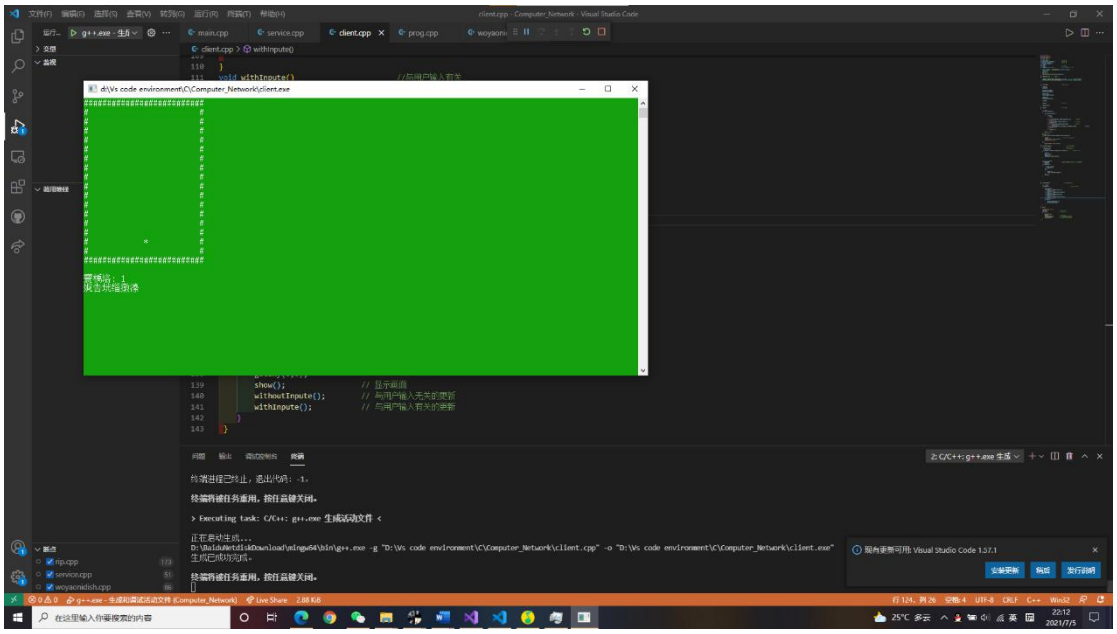
此函数的主要功能是开火，也就是让飞机可以杀死怪物。

```
;//发射子弹子程序
;入口参数 玩家飞机发射口的坐标 bx+5,bp
shoot_plane proc
    push ax
    push bx
    push cx
    push dx
    push si
    push bp
    mov cx,bx
    add cx,5     ;x 坐标 BX+5
    mov dx,bp    ;y 坐标
    dec dx
    ;擦除炮弹轨迹，移动炮弹
a0: MOV BX,2     ;宽度
    INC DX
a1: MOV AH,0CH   ;在绘图模式显示一点
    MOV AL,0     ;颜色(黑色)，用于擦除上一个子弹
```

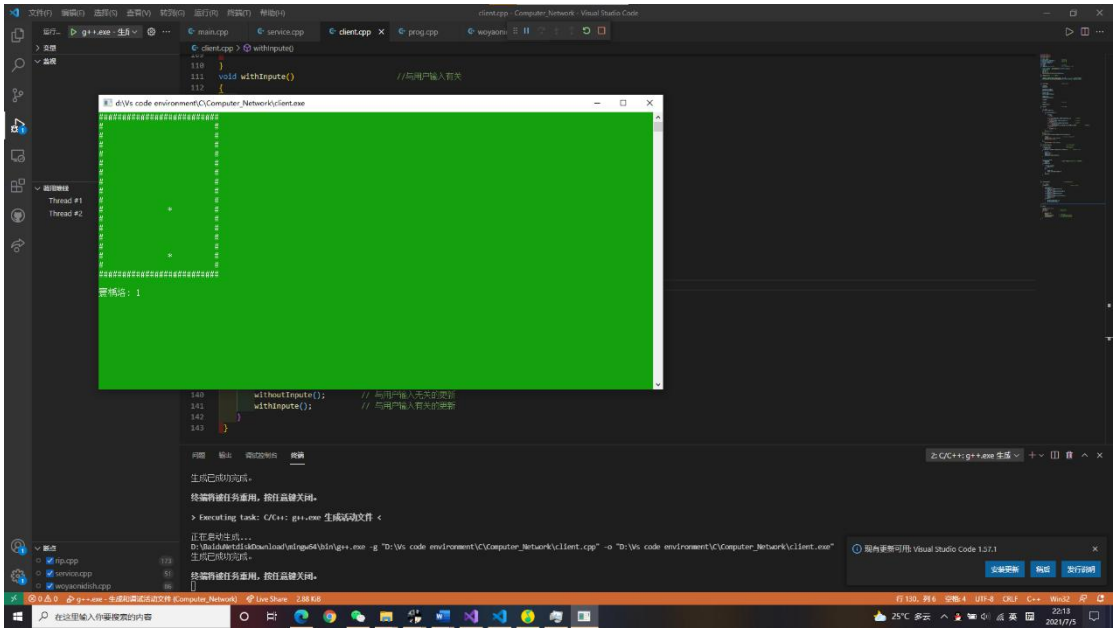
```
    INT 10H
    INC CX
    DEC BX
    JNZ a1      ;擦除炮弹宽度
    SUB CX,2
    MOV BX,2
    DEC DX

a2: MOV AH,0CH  ;在绘图模式显示一点
    MOV AL,11   ;颜色（白色），用于画新子弹
    INT 10H
    INC CX
    DEC BX
    JNZ a2      ;画出炮弹宽度
    SUB CX,2
    CALL delay<span style="white-space:pre">    </span>;时延，可用来调整子弹的
移动速度
    DEC DX
    CMP DX,6    ;循环画炮弹,到顶端才停止
    JA a0
notdes:
    ;最后一次擦除
    mov bp,sp
    mov cx,word ptr ss:[bp+8]
    add cx,5
    mov dx,7
    MOV AH,0CH  ;在绘图模式显示一点
    MOV AL,0    ;颜色
    INT 10H
    inc cx
    MOV AH,0CH  ;在绘图模式显示一点
    MOV AL,0    ;颜色
    INT 10H
    pop bp
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    ret
shoot_plane endp
```

4.4.2 程序运行截图

4.4.2.1 初始界面



4.4.2.2 怪物出现界面



4.5 实验心得体会

经过这一次的游戏开发，我们对汇编模块化编程有了一些了解。理解了汇编作为一个底层语言需要掌握的东西很多，在使用高级语言开发都不需要考虑的事，在汇编程序中都必须考虑清楚与完善。步骤也比较繁琐。

下面是一些开发经验：

1.DW/DD/DB 不是简单的和 int/char 等对应的，因为使用了 DD，我们发现将对应的块的代码完成后，取到寄存器中时，并不是占据了低 16 位，而是在各个地方散布着。应该根据数据的长度分配它的空间。

2.内外层逻辑需要条理清晰，比如跳转。

3.汇编语言中，一些常见指令的用法如 div 等与 c 语言有较大差别，需要掌握好指令的格式及功能，以及指令执行对响应寄存器的影响，才能较好地完成转汇编工作。