汇编语言与接口技术——2048 实现

1120180329 陈稳 1120180336 吴雪龙 1120180484 付宇

一、2048 简介

1.1 背景

2048 是一款 2014 年推出的新颖益智手机游戏，游戏的规则十分简单，仅仅通过滑动手指将其中的数字进行合并，而就是这么一个简单的游戏规则，却时常让玩家感到十分虐心。那么本次汇编组队实验，我们就通过汇编语言对该游戏进行实现。

1.2 游戏规则

2048 这款游戏的规则十分简单。我们会给出一个空的 4×4 的大方块，在游戏刚开始时随机生成数字 2，并将其放入 16 小方块中的任意空方块中；随后，玩家选定某一移动方向，对其进行平移操作，移动后在同一方向上，相同数字的方块在靠拢、相撞时会相加。不断的叠加最终拼凑出 2048 这个数字就算成功。而如果在中途无法通过移动进行合成、没有空方块用以随机生成数字，即为失败，如图所示。

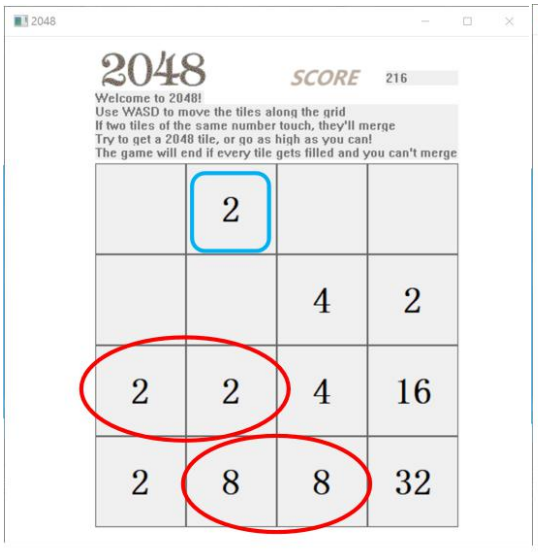


图 1-1

图 1-2

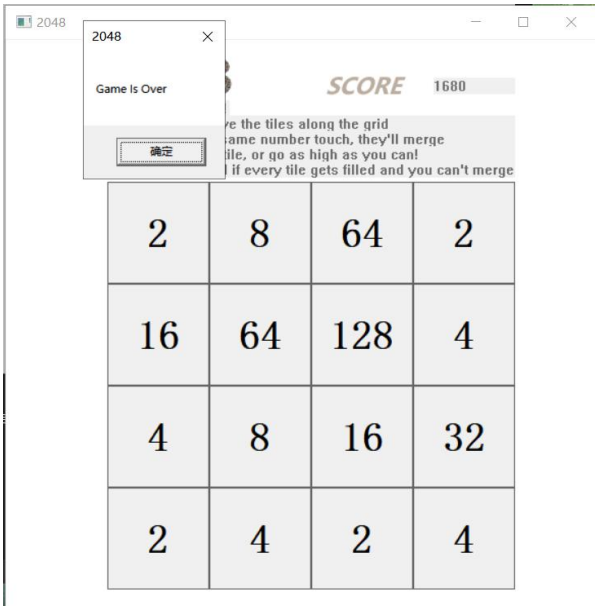


图 1-3 游戏结束

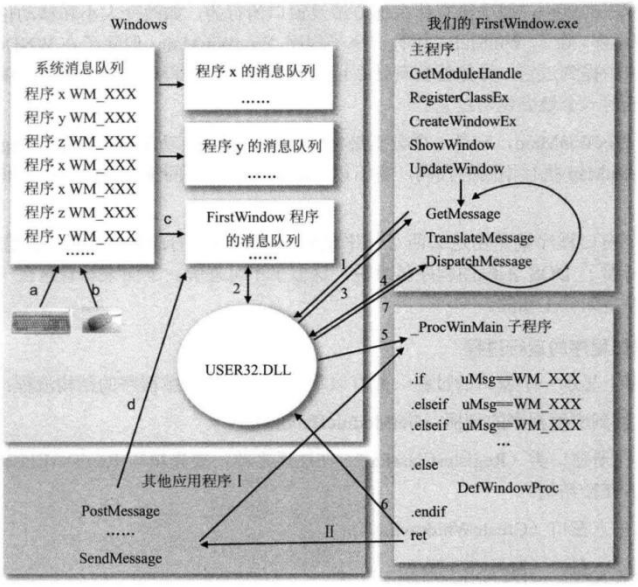
1.3 需求分析

通过对 2048 游戏的分析，我们将该游戏的实现分为了三个主要部分并进行分工：绘制游戏界面和动画、随机生成新方块，得分计算，判断游戏胜利和失败、实现方块的移动，分别由陈稳、吴雪龙、付宇分模块完成。

二、2048 实现

2.1 游戏界面绘制

游戏界面绘制窗口程序主要有_WinMain 和_ProcWinMain 两个函数来构成。



_WinMain 是生成窗口的主函数，主要流程为：

（1）得到应用程序句柄
（2）注册窗口类
（3）建立窗口
（4）显示窗口
（5）刷新窗口客户区
（6）维护消息获取和处理的循环
通过 GetMessage 获取消息
TranslateMessage 翻译消息
DispatchMessage 分派消息给回调函数处理
通过 User32 来调用回调函数

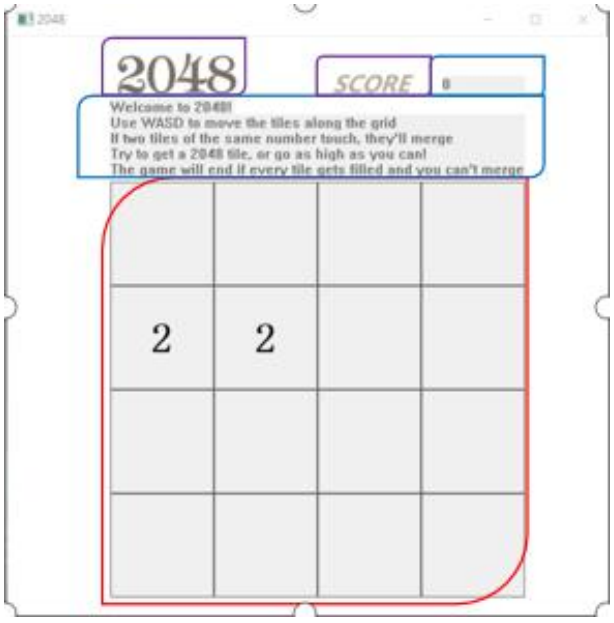
_ProcWinMain 是用来处理消息的，是窗口的回调函数，也叫窗口过程。消息如 WM_PAINT，WM_CHAR 等。

2.1.1 界面构成

界面主要由 4 个部分 3 个模块构成.

4 个部分分别是
（1）游戏主体部分的方块
（2）游戏说明部分
（3）显示得分部分
（4）游戏名称部分

3 个模块分别是
（1）静态控件'Static'绘制游戏模块
（2）文本框'edit'绘制得分和游戏说明
（3）位图装饰界面



2.1.2 绘制静态控件

需要绘制 4X4 的方块，采用一个 2 重循环，通过 i，j 控制绘制的位置和绘制的控件 ID

```
;eax=i*100+140，绘制的x坐标，140为起始坐标
imul eax,i,100
add eax,140
;ecx=j*100+100，绘制的y坐标，100为起始坐标
imul ecx,j,100
add ecx,100
.IF Data[0] =='0'
    ;创建静态控件，居中有边框
    invoke CreateWindowEx,NULL,offset static,offset EmptyText,\
    WS_CHILD or WS_VISIBLE or SS_CENTER or WS_BORDER or SS_CENTERIMAGE,ecx,eax,100,100,\
    hWnd,edx,hInstance,NULL  ;句柄为edx
.else
    invoke CreateWindowEx,NULL,offset static,offset Data,\
    WS_CHILD or WS_VISIBLE or SS_CENTER or WS_BORDER or SS_CENTERIMAGE,ecx,eax,100,100,\
    hWnd,edx,hInstance,NULL  ;句柄为edx
.endif
;edx=i*4+j，表示第[i][j]个方块
imul edx,i,4
add edx,j
;存储窗口句柄，句柄返回值在eax中
mov hGame[edx*4],eax
```

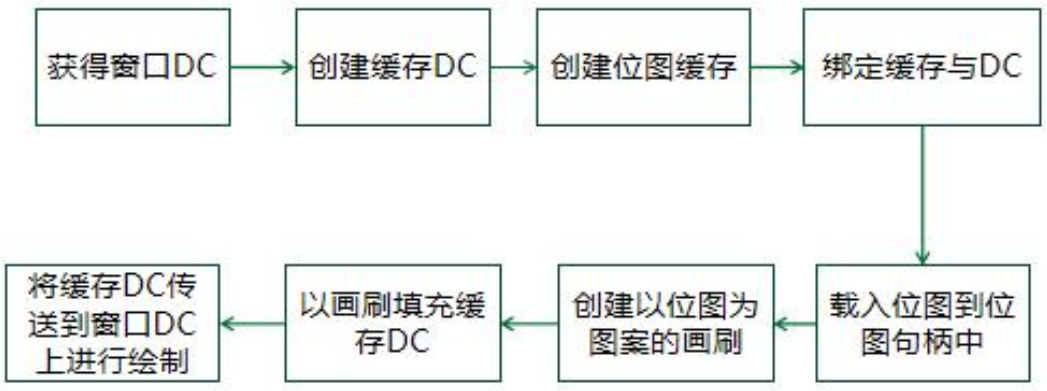
2.1.2 绘制文本框

绘制文本框同绘制方块一样，通过调用 CreateWindowEx 函数，指定对应的值即可，由于文本框可以在界面上进行编辑，所以绘制时指定为禁用状态。

```
;绘制游戏说明部分
;创建文本框，但设为Disabeled防止玩家更改
invoke CreateWindowEx,NULL,offset edit,offset szText1,\
WS_CHILD or WS_VISIBLE OR WS_DISABLED,100,60,120,15,\
hWnd,16,hInstance,NULL
MOV hGame[64],eax
```

2.1.3 绘制位图

加载位图主要通过调用 gdi32 库所支持的 API，其主要流程为：



具体实现代码如下：

```
;加载位图
;首先获取窗口DC
invoke GetDC, hWnd
mov @hDc,eax
;创建兼容窗口DC的缓存dc
invoke CreateCompatibleDC,@hDc
mov hdcIDB_BITMAP1,eax
;创建位图缓存
invoke CreateCompatibleBitmap, @hDc,150,80
mov hbmIDB_BITMAP1,eax
;将hbm与hdc绑定
invoke SelectObject,hdcIDB_BITMAP1,hbmIDB_BITMAP1
;载入位图到位图句柄中
invoke LoadBitmap,hInstance,BITMAP1
mov @hBm,eax
;创建以位图为图案的画刷
invoke CreatePatternBrush,@hBm
push eax
;以画刷填充缓存DC
invoke SelectObject,hdcIDB_BITMAP1,eax
;按照PATCOPY的方式
invoke PatBlt,hdcIDB_BITMAP1,0,0,150,80,PATCOPY
pop eax
;删除画刷
invoke DeleteObject,eax
;在主窗口DC上绘制位图dc
invoke BitBlt,@hDc,90,0,150,80,hdcIDB_BITMAP1,0,0,SRCCOPY
```

2.1.4 显示数字和分数

我们通过使用一个数组 gameMat 来保存游戏的局面，由于存储的是数字，所以需要通过不断除以 10，按位转换为数字字符串来显示，之后调用 SetWindowText 函数显示文字。

```
    ;清空寄存器，32位除法需要
    xor eax,eax
    xor edx,edx
    xor ebx,ebx
    ;被除数放到eax中
    mov eax,number
    mov ecx,10
    ;2048/10=204...8压入8
    ;204/10=20...4压入4
    ;20/10=2...0压入0
    ;2/10=0...2压入2
    ;商为0结束循环
L1:
    ;ebx记录number位数
    inc ebx
    ;eax/ecx，32位除法，商在eax中，余数在edx中
    idiv ecx
    ;余数+'0'='0'-'9'
    add edx,30H
    ;先压栈后续一起处理
    push edx
    ;记得清0，32位除法被除数为edx:eax
    xor edx,edx
    ;eax为商，商=0表示除尽了
    cmp eax,0
    ;大于0继续循环
    jg L1
```

```
    ;esi=0，表示第esi个字符
    mov esi,0
L2:
    ;ebx为之前记录的number位数，每次循环减1，直到为0
    dec ebx
    ;将栈中的字符出栈存到eax中
    pop eax
    ;结果只为'0'-'9'，只在8位寄存器中，无所谓了
    mov byte ptr Data[esi],al
    inc esi
    cmp ebx,0
    jg L2

    ;循环结束，末尾赋0表示结束
    mov Data[esi],0
    ret

num2byte endp
```

```
    ;edx=i*4+j表示[i][j]块方块
    imul edx,i,4
    add edx,j
    ;转换值
    invoke num2byte,dword ptr gameMat[edx*4]
    imul edx,i,4
    add edx,j
    ;设置控件中的值
    .if Data[0] =='0'
        INVOKE SetWindowText,hGame[edx*4],offset EmptyText
    .else
        INVOKE SetWindowText,hGame[edx*4],offset Data
    .endif
```

2.1.5 显示游戏胜利和失败消息

通过维护 gameIsEnd，gameIsWin 和 gameContinue 三个变量。

游戏失败时，gameIsEnd 置 1，弹出游戏失败提示消息，点击 OK 重新开始游戏。

游戏胜利时，gameIsWin 置 1，弹出游戏胜利提示消息，玩家可选择继续游戏或结束游戏。

若游戏继续，gameContinue 置 1，则之后不再对游戏是否胜利进行判断。

2.2 方块移动及合并

方块的移动分为四个方向，所以首先设想为该模块编写四个函数，分别实现上下左右的移动的实现。通过沟通，我们将输入字符 WASD 分别代表向上、向左、向下、向右移动。

```
moveW proc far C uses eax ebx ecx edx...

moveD proc far C uses eax ebx ecx edx...

moveA proc far C uses eax ebx ecx edx...

moveS proc far C uses eax ebx ecx edx...
```

图 2-2-1 实现函数展示

2.2.1 移动方向以及起点

上图的四个函数的实现过程，直观上讲有如下效果：

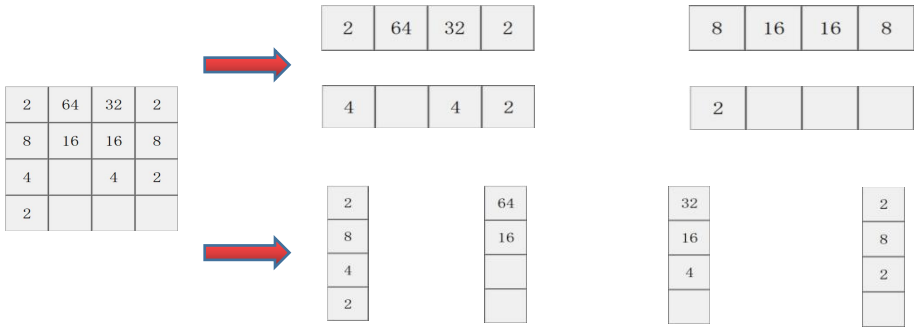


图 2-2-2 对方块进行分组遍历

通过对 16 个方块按行或按列进行划分，然后我们对每个方向的行或列进行遍历，之后通过遍历过程中某方块和他周围方块的关系进行比较，如此实现整个遍历过程。例如我们对向右的移动进行分析，我们就会把所有方块按行分组，对每一行的每一个方块进行遍历。

完成分治后，我们还要对遍历起点进行分析。在实现过程中，我采用了 Loop 的方式进行循环，所以我们的循环起点始终为 4，之后不断递减形成循环。此时我们对不同方向的移动起点进行分析，可知我们的起点如图所示：

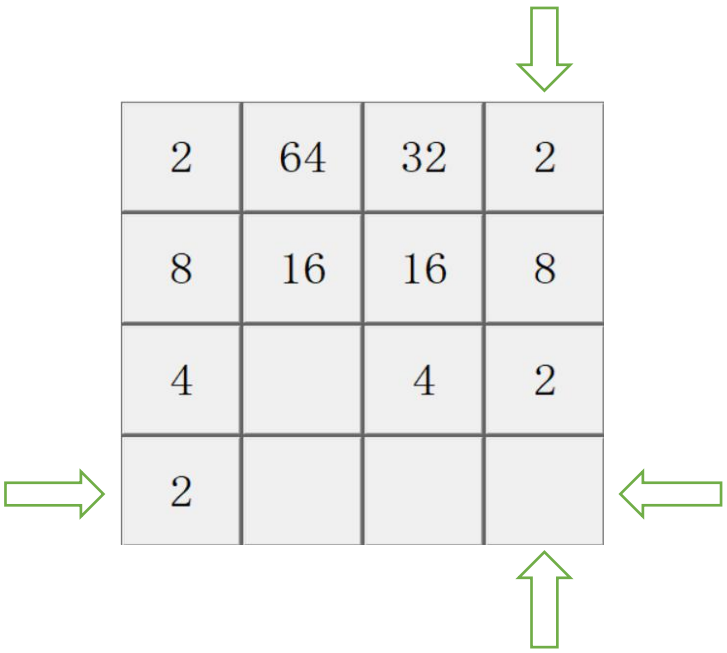


图 2-2-3 各方向循环起点

2.2.2 合并实现

确定了循环起点后，我们还要对具体的方块移动方向进行确定。此处我们仍对 D 方向（向右移动）进行分析。在游戏过程中，我们发现，每个方块都会和其左边的方块进行比较，若某方块的左边出现相同数字或者该相同数字和数字 0 一同出现，则会进行合并。如此，我们就确定了每个循环中行或列的遍历方向是与移动方向相反的，如此才能更快更好的实现该功能。在确定了循环内遍历方向后，我们自然会向确定移动方向一样，对每个行或列的所有方块抽象为一个状态，并对每个状态进行遍历判断。事实上，在开发过程中我发现整个过程可以抽象为一个递归的判断过程，这样实现就更加简单。其具体实现思想如下：

我们不再对每个方向上的四个方块分别抽象，而是对其状态进行抽象：

1）是否为 0

如果我们遍历到该数字为 0 的话，我们直接跳过，因为我们不需要对 0 进行任何操作；

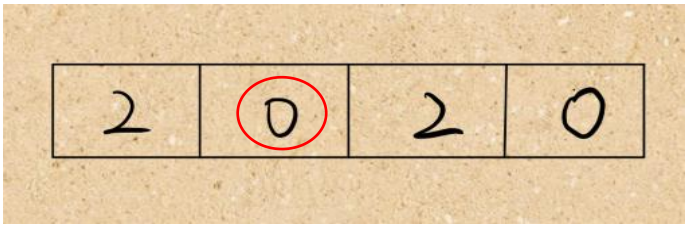


图 2-2-4　0（空方块）忽略

2）左右是否为 0

如果是在移动过程，在其移动方向上若存在 0（存在 0 就说明有空方块，有位置支持移动），我们就对该方块进行移动，即 0 和非零数字的交换，然后继续进行判断，若到达边界或没有空方块则无法进行移动，我们对其进行合并功能的判断。

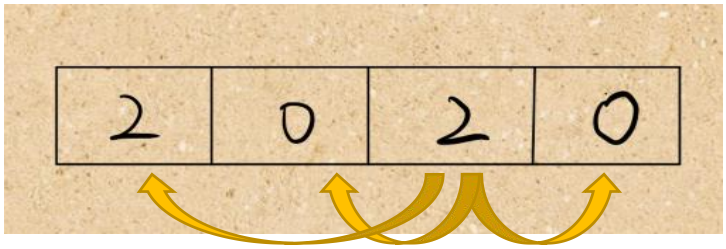


图 2-2-5 邻居方块判断

3）忽略 0 后是否有相同数字

我们知道在合并过程中，我们会对移动过程的反方向进行判断，当然判断过程中，我们仍需要对是否有 0（即是否有空方方块）进行判断，若有则直接跳过，继续遍历；若遇到相同数字，直接合并，并将遍历到的数字置为 0，若没有或者遍历越界，则对循环中的下一个方块进行如上方式的判断，如此就能完成每一行或列的移动和合并。

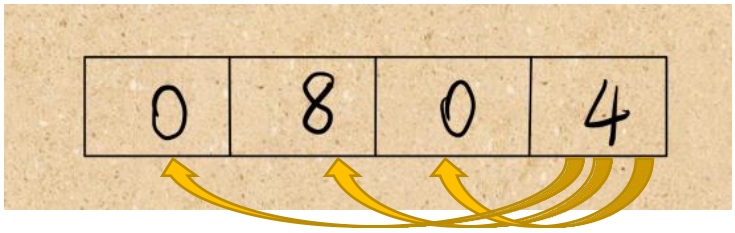


图 2-2-6 合并判断

三、实验总结

通过本次实验，我们学习了汇编语言的使用，并通过汇编语言实现一个具体游戏加深对汇编语言与编程的理解。通过对相对较底层的语言，了解了高级语言的工作方式，认识了汇编的编程思路与思想。通过组队完成一个程序任务，认识到了在合作中模块化的重要性与优势，加强了团队协作水平与沟通能力。

通过对任务底层的调用了解了随机数的生成方式，认识到了 GUI 的运行逻辑以及使用方式，了解到了在汇编中不同寄存器的功能，以及如何使用简单的指令去实现一个较为复杂的功能。感谢本次课程给予的宝贵的实验机会。