**BOMB!  
——基于汇编语言的益智游戏**

***Version 1.0***

**编写者**

**Prepared by**

**团队：王陈陈**

**Group Name: wcc**

|  |  |  |
| --- | --- | --- |
| **王肖佑** | **2011013273** | **wxyyxw09090@sina.com** |
| **陈璐** | **2011013249** | **chenlu11@mails.tsinghua.edu.cn** |
| **陈华榕** | **2011013236** | **chenhuarongzp@gmail.com** |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| **Instructor:** | **王朝坤** |
| **Course:** | **计算机网络体系结构（2）** |
| **Teaching Assistant:** | **陈俊** |
| **Date:** | **2014/04/12** |

目录

[文档修订 iii](#_Toc385086440)

[1 开发概况 1](#_Toc385086441)

[1.1 开发团队 1](#_Toc385086442)

[1.2 开发时间轴 1](#_Toc385086443)

[1.3 开发成果 1](#_Toc385086444)

[2 游戏规则 2](#_Toc385086445)

[2.1 游戏背景 2](#_Toc385086446)

[2.2 操作方式 2](#_Toc385086447)

[2.3 操作规则 3](#_Toc385086448)

[2.4 胜利条件 3](#_Toc385086449)

[2.5 失败条件 3](#_Toc385086450)

[3 程序模块 4](#_Toc385086451)

[3.1 数据与逻辑 4](#_Toc385086452)

[3.2 Win32入口 5](#_Toc385086453)

[3.3 文件存取 10](#_Toc385086454)

[4 总结 10](#_Toc385086455)

# 文档修订

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号  Version | 主要作者  Primary Author(s) | 简述  Description of Version | 完成时间  Date Completed |
| 1.0 | 王肖佑、陈璐、陈华榕 | 本文档撰写完成。 | 2014/04/12 |

# 开发概况

## 开发团队

WCC团队是由来自THSS机构的三个风格迥异的geek组成。

王肖佑童鞋，勤劳踏实好欺负，爱生活，不push。

陈璐小盆友，江湖人称nunu。哪里有nunu，哪里就有笑声，哪里有nunu，哪里就有奇思妙想。和nunu一起开发项目，绝对可以年轻好几岁。

陈华榕大帝，团队中的大BOSS。虽然有的时候有点儿DB(doubi)，但是有DD在的地方，就没有解决不了的问题，一切难题遇到他瞬间灰飞烟灭。

如果说一个好的团队是项目成功的一半，那么WCC团队做的项目绝对是百分之百的成功。

## 开发时间轴

2014-3-17 确立游戏规则，并建立初始化工程

2014-3-20 完成相关素材抽取制作，实现基本绘图功能

2014-3-23 数据处理逻辑部分完成（附有可运行控制台程序）

2014-3-24 实现不带动画版游戏，实现基本交互功能

2014-3-27 新增保存、读取功能

2014-3-29 添加新方块出现、移动方块、合成方块时的动画（双缓存实现）和爆炸音效

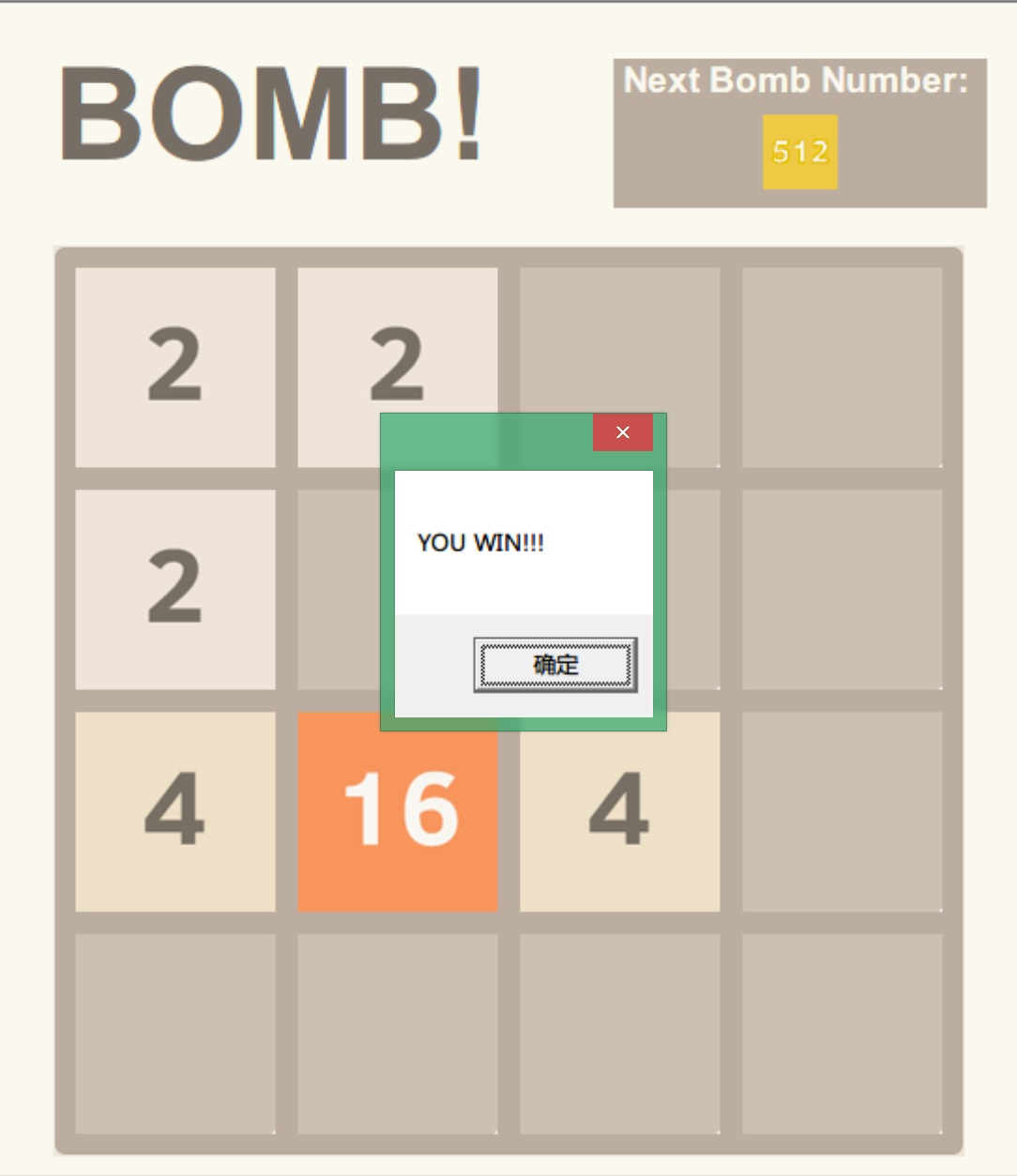
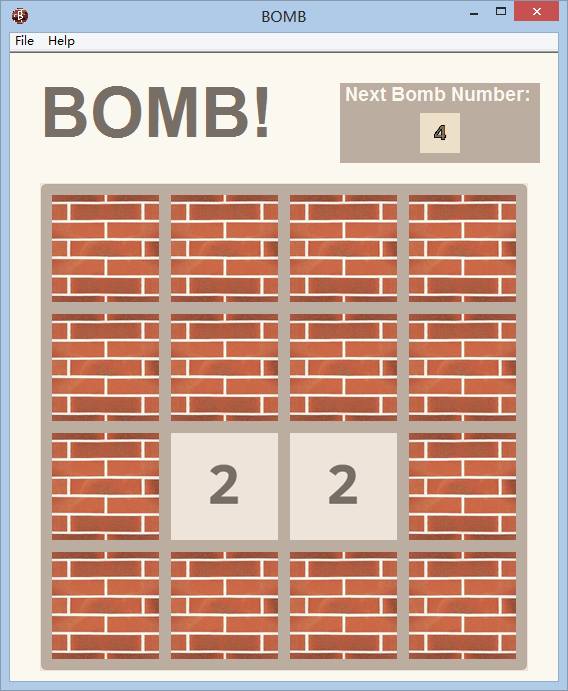
2014-4-09 新增按键队列，扩展游戏规模，完善游戏交互

我们的团队启动早肯投入、攻克大量技术难点、时间规划合理，高效且高质量地完成了BOMB!游戏从设计到开发的全过程。

## 开发成果

我们经过这段时间的开发，设计并用汇编语言在Windows平台实现了一款名为BOMB!的游戏。

该游戏基于Win32汇编，操作简单，规则有趣，具有较高的可玩性，我们开发人员在测试过程中发现，这个游戏一玩就“根本停不下来”。



游戏刚开始与游戏胜利的情形

# 游戏规则

## 游戏背景

火药由中国人发明，距今已有超过一千年了。最近几年由于科技进步，人类掌握了通过信息技术合成炸弹的方法，若能推广使用，将大大减少炸弹制作过程中的资源消耗以及引爆后的环境污染。

这看似不可能的研究成果正是在清华大学软件学院某实验室诞生的，但由于该技术严重影响火药生产商的利益，在推广过程中受到了很大阻碍（也因为此，现在甚至连软院的同学都几乎不知道这个能震惊世界的成果）。该实验室希望通过事实证明这项技术是切实可行、可靠的，然后将该成果公之于众。你被该实验室招募为一名终身测试人员，将参与未来对炸弹合成技术的测试。

WCC团队在该实验室的委托下，开发了BOMB!游戏，以此模拟炸弹合成、引爆等过程，并加入游戏化元素。这样，测试人员只需要通过玩BOMB!游戏就能在悄然间完成一次次的测试工作。

您可以将BOMB!游戏分享给亲朋好友，那将大大加快测试进度，但请切记不要将涉及的炸弹合成技术测试相关背景透露给任何人。为了保证您的人身安全，也请不要透露您是该技术的终身测试人员这一身份。

由于涉及的逻辑分支非常之多而且必须通过人力驱动，因此我们初步推算，即使全球能有10亿人不间断地参与，该实验室要完成完整的测试也大约要到2222年。

## 操作方式

本游戏借鉴了风靡一时的2048小游戏的操作方式，采用上下左右这四个方向键来控制所有方块的移动，并采用快捷键和菜单栏实现了新建、读取等控制功能。

## 操作规则

### 游戏元素

|  |  |  |
| --- | --- | --- |
| 墙 | C:\Users\chenlu11\Documents\GitHub\BOMB\asm\BOMB\images\brick107.bmp | 还没有被炸开的方块 |
| 炸弹 | C:\Users\chenlu11\Documents\GitHub\BOMB\asm\BOMB\images\bomb107.bmp | 达到爆炸要求的方块 |
| 空白区域 | C:\Users\chenlu11\Documents\GitHub\BOMB\asm\BOMB\images\pile107-0.bmp | 被炸开的空白区域 |
| 数字 |  | 带有数字的方块 |

### 变化规则

本游戏对方块移动的控制和2048类似，当某一方向键按下时，所有方块都朝该方向移动，移动期间数字相同的两个方块会合并成一个方块且数字增倍，直到碰到边界或墙或不可合并的其他方块为止。（具体的合并规则可参见3.2.3方向逻辑）

移动完成后，会在游戏的空白区域随机出现一个新的数字为2或4的方块。

在游戏的右上角有一个显示当方块上的数字达到多少时才会发生爆炸的指示数字，该数字初始是4，每次爆炸后，该数字都会倍增。当爆炸发生时，发生爆炸的方块上下左右四个方向上的墙被炸开（如果不是墙，则不会被炸开），该方块自己也会被炸开。用户需要尽可能地合并数字更大的方块，并让方块在合适的位置爆炸，以炸开更多的墙来取得游戏胜利。

## 胜利条件

在所有地图块都不可移动或合成前，所有的墙都被炸开。

## 失败条件

在所有的墙都被炸开前，所有的地图块都无法移动或合成。

# 程序模块

## 数据与逻辑

这部分主要涉及bomb\_core.inc和bomb\_core.asm两个源代码文件。

### 地图与基本数据

|  |  |  |
| --- | --- | --- |
| 标识名 | 类型 | 说明 |
| map | 行列数相等的SDWORD类型的二维数组。 | 地图，可能取值负数表示墙，0表示空地，还可取值2^n（n>=1）表示用于合成的数字。 |
| mapSize | SDWORD类型的变量。 | 设置的地图大小，取值必须满足0<mapSize<=20。 |
| bombTarget | DWORD类型的变量。 | 当合成出>=bombTarget的数字时，将得到炸弹，同时bombTarget<<=1。地图初始化时将设定为4。 |
| resultQueue | 一维DWORD类型的数组。 | 用于给交互界面返回结果的队列，详见下一节“结果队列”。 |
| emptyBlocks | SDWORD类型的变量。 | 表示当前空地的格子的数量。 |
| numberBlocks | SDWORD类型的变量。 | 表示当前数字的格子的数量。 |

### 结果队列

为方便播放动画，当用户请求执行某方向动作后，数据逻辑部分通过结果队列将动画详情返回给交互界面调用者。结果队列是一维DWORD类型的数组，其特定二进制位有特别含义，每个DWORD表示一个结果。

以下所称第x位代表二进制的右起第x位，以a\_n表示数a的第n位，以a\_m\_n表示数a的第m到n位。如对于b=10010b，有b\_1=0，b\_2=1。

结果队列有四种可能：地图块移动、数字合成、出现炸弹。

#### 地图块移动

以a表示要加入结果队列的数，记要移动第x行第y列的地图块，移动距离为z。这里不需要记录移动方向，因为结果队列一定是在执行某方向移动后得到的，方向已经隐含在其中了。

则a\_2\_1=0，a\_7\_3=x，a\_12\_8=y，a\_32\_13=z。

#### 数字合成

数字合成的动画在地图块移动的动画完成后进行，只需要指明哪个地图块进行了合成即可，假设为第x行第y列的地图块，并以a表示要加入结果队列的数。

则a\_2\_1=1，a\_7\_3=x，a\_12\_8=y，a\_32\_13=0。

#### 出现炸弹

炸弹出现的动画涉及炸弹的位置，需要指明的是地图块的行x和列y。以a表示要加入结果队列的数，则a\_2\_1=2，a\_7\_3=x，a\_12\_8=y，a\_32\_13=0。

爆炸后应将周围的墙夷为平地，这是隐含的，不需要在结果队列中体现。

### 方向逻辑

#### DoMove

通过DoMove过程实现方向的操作。该过程只有一个参数，即方向，可取值DIR\_UP、DIR\_DOWN、DIR\_LEFT、DIR\_RIGHT，通过eax传入。

在其中针对不同方向进行方块移动，依照游戏规则，移动的逻辑顺序是：根据移动方向的反方向枚举各个地图块（比如向下移动就列数从大到小枚举，向上移动则列数从小到大枚举），每次找到不为墙的地图块A，就向移动方向找到第一个不为0的地图块B，若B不为墙，就接着判断A，如果A为0，就直接把B移至A，而如果A=B，就将两个地图块合成，其他情况不操作。

当地图上的所有地图块都经过上述移动操作后，再进行合成炸弹判断，枚举一遍地图，将所有满足合成条件（数字>=bombTarget）的地图块替换为炸弹，然后将该地图块四周的墙炸为空地。

上述所有有效操作均会在结果队列产生新内容。

该过程返回结果队列的长度，通过eax返回。

#### AddNum

该过程调用后将给地图增加一个数字块，即用一个数字替换一块空地。

首先根据当前空地数生成一个随机数X，然后根据该随机数按特定顺序找到地图上第X块空地。再通过随机数决定要生成的数字，90%概率生成2，同时10%概率生成4。

该过程通过eax返回生成的情况。以a\_n表示返回值的第n位，a\_m\_n表示返回值的第m到n位，则返回值表示在第a\_11\_7行、第a\_6\_2列生成了一个数(2<<a\_1)，这里的行列数可取0。

### CheckMap

该过程调用后检查地图状态，返回值有三种可能：MAP\_CONTINUE表示可继续游戏；MAP\_WIN表示已将所有墙夷为平地，游戏已取得胜利；MAP\_FAIL表示已没有可移动数字块，游戏已经失败。

## Win32入口

这部分主要涉及bomb.inc和bomb.asm两个源代码文件。

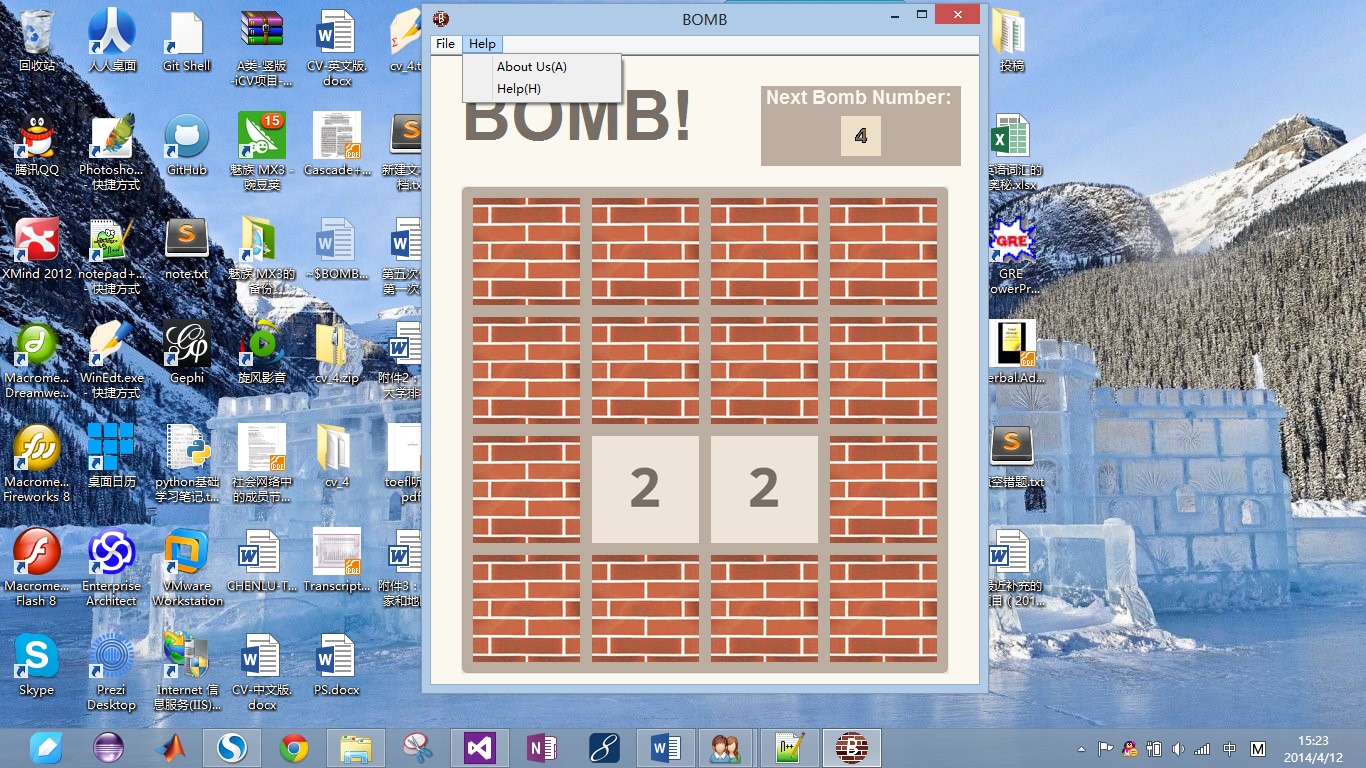
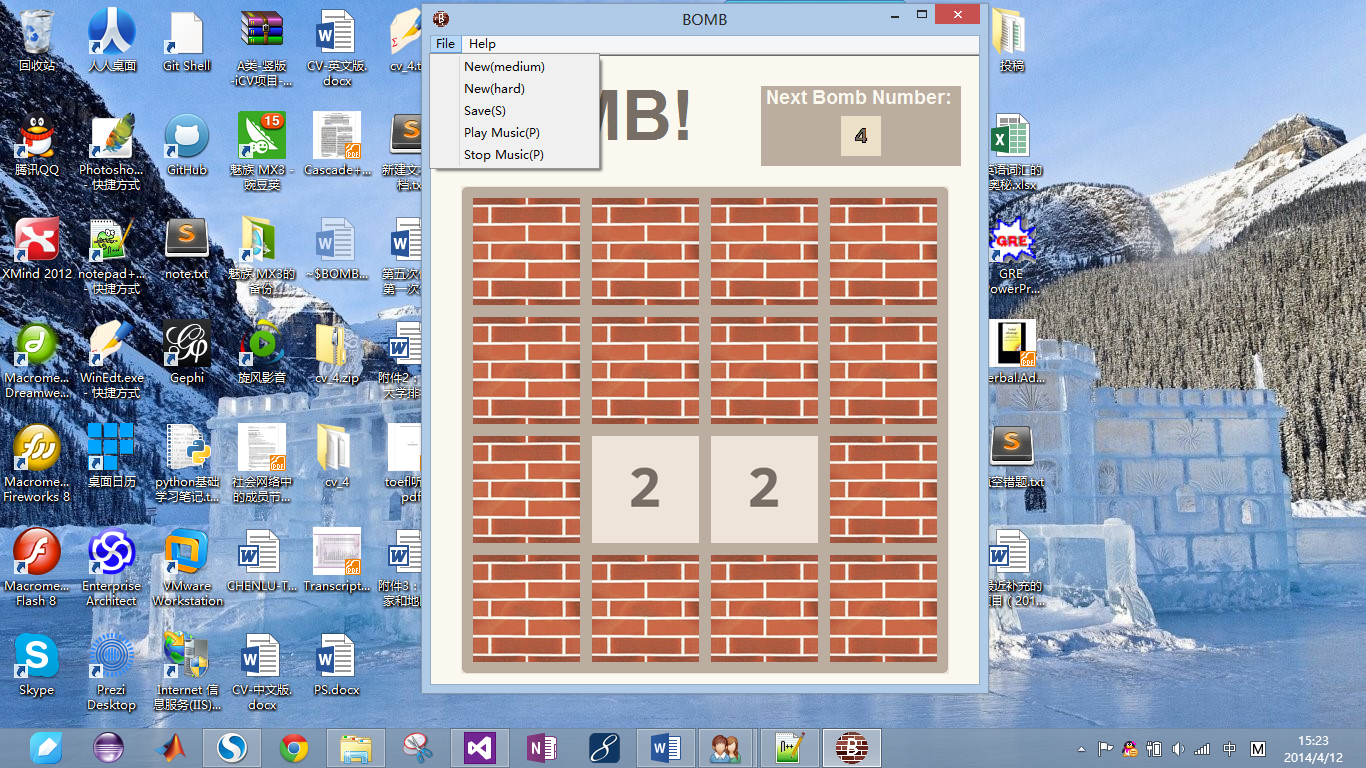
### 程序入口

程序的入口是WinMain过程。WinMain主要做了以下几部分工作：

* 获取当前程序的句柄
* 初始化窗口类，设置窗口的大小、图标、背景颜色、偏移位置等参数，并且指定用来接收窗口时间消息的过程WndProc
* 注册窗口类并创建主窗口
* 载入游戏中用到的图片资源
* 创建并初始化画刷、字体、RECT等元素
* 初始化游戏地图
* 设置时钟
* 显示并更新主窗口
* 开始一个消息循环来接收和分派处理消息，循环会一直持续到用户关闭了应用程序窗口

### 界面交互

实现了对快捷键和菜单栏的支持，键盘事件的实现参见函数KeyDownProc，菜单事件的实现参见函数WndProc。

菜单栏实现功能：新建游戏(4\*4)，新建游戏(5\*5)、保存游戏进度、播放背景音乐、停止播放背景音乐、关于我们、帮助。

快捷键实现功能：新建游戏(N)、保存游戏进度(S)、播放背景音乐(P)、停止播放背景音乐(P again)。

除此之外，在关闭游戏窗口时会弹出对话框询问是否保存进度。在打开游戏时，若有保存的进度，会弹出对话框询问是否载入进度。

### 附加交互

主要利用mciSendCommand指令，实现了对mp3格式音乐的支持。

播放背景音乐的实现参见PlayMp3BombFile函数，播放爆炸声的实现参见PlayMp3File函数。

### 窗口绘制

#### 图片资源

BOMB! 用到的图片资源均为bmp类型，所有图片在WinMain过程中载入，保存在位图句柄中。图片包括背景图、砖块图、空白方格图、炸弹图、数字图（2~65536）。其中对于砖块图、空白图和数字图，可以通过宏SetCurrentMap来设置当前的图片保存在寄存器eax中，参数为负数则设置砖块图，参数为0则设置空白图，参数为大于0的其它数字则设置对应的数字图。

#### PaintProc

当消息处理过程接收到消息WM\_PAINT时，就会调用绘图过程PaintProc重新绘制窗口，绘制过程是：

1. 绘制游戏背景，包括背景图片和左上角的游戏标题
2. 绘制游戏区域中砖块和空格：遍历地图，若值为负数则画砖块，否则画空格
3. 绘制游戏区域中的数字或者炸弹：遍历地图，若值为正数则画对应数字，否则不画
4. 调用DrawNextBombText在右上角绘制下一个爆炸数字的提示

#### DrawSquare

DrawSquare是封装好的绘制方块的过程。该过程的参数有:

* xIndex，yIndex：所画方块的坐标，xIndex和yIndex的取值范围为0~3或0~4
* bmpObj：方块对应的位图句柄
* movedis：所画方块相对于xIndex, yIndex处的方块的位置偏移量，偏移方向存在了MoveDir中
* scale：所画方块的大小缩放比例，scale为50则表示缩放比例为50%。缩放时方块中心固定。

#### 双缓冲绘图

使用一般的方法进行绘图，窗体在重绘时会由于过频的刷新而引起闪烁现象。为了防止绘制界面时屏幕出现闪烁，故采用双缓冲技术进行绘图。

双缓冲技术的原理是每次窗体在响应WM\_PAINT消息进行重绘时，先在内存中创建一张和窗体一样大的位图，然后在内存中的位图上进行绘制。所有的绘图工作完成后，一次性把内存中的这张位图复制到窗体上，这样就消除了闪烁。

在代码中的具体实现是：

|  |
| --- |
| INVOKE CreateCompatibleDC, hDC  mov memDC, eax  INVOKE CreateCompatibleDC, hDC  mov imgDC, eax  INVOKE CreateCompatibleBitmap, hDC, WndWidth, WndHeight  mov hBitmap, eax  INVOKE SelectObject, memDC, hBitmap  mov hOld, eax  [ 在memDC上绘图 ]  INVOKE BitBlt, hDC, 0, 0, WndWidth, WndHeight, memDC, 0, 0, SRCCOPY |

调用CreateCompatibleDC创建两个内存DC，调用CreateCompatibleBitmap创建与hDC环境相关的设备兼容的位图，并将其放入memDC中。而imgDC的作用是在绘图过程中将要画的位图选入，再在memDC上绘图。绘图完成后，调用BitBlt，将memDC中的图一次性绘制到hDC上。

最后，需要注意的是，在绘制完成后要调用DeleteDC和DeleteObject把之前创建的两个内存DC和一个位图删掉，否则会造成内存泄露。

### 游戏动画

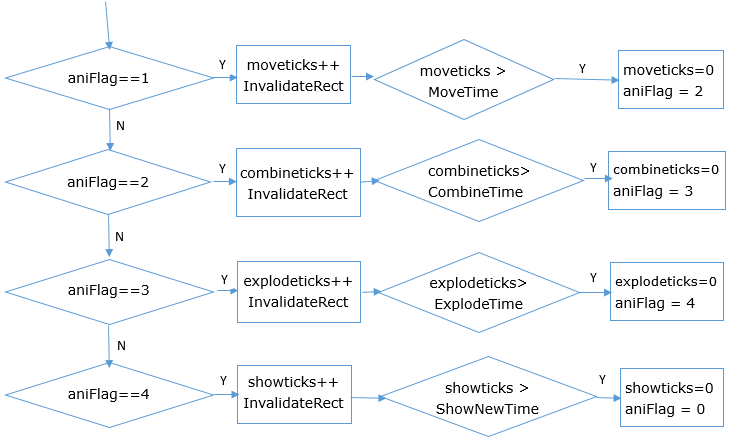
BOMB!的游戏动画包括四部分内容：方块移动，方块合成，爆炸，生成新方块。

#### 动画实现方法

动画的实现主要利用了动画标记变量aniFlag和时钟事件处理函数TimerProc。

aniFlag用于标记当前所在的动画阶段，0表示没有动画，1表示方块移动，2表示方块合成，3表示爆炸，4表示出现新方块。计时器在WinMain中启动，消息处理函数接收到WM\_PAINT时就会调用TimerProc过程。TimerProc过程中会根据aniFlag的值执行不同的动画操作。

对于四种动画分别有MoveTime, CombineTime, ExplodeTime, ShowNewTime来表示动画时间长度。moveticks, combineticks, explodeticks和showticks是四个动画的计时器。下面是TimerProc中动画实现过程的大致流程。



对于不同动画阶段，在调用InvalidateRect刷新界面时，PaintProc的第（3）部分（绘制游戏区域中的数字或炸弹）会有不同的绘制方法。

动画中用到的其它数据结构还有：

|  |  |  |
| --- | --- | --- |
| 标识名 | 类型 | 说明 |
| oldmap | 行列数相等的SDWORD类型的二维数组。 | 老地图，存有在逻辑计算更新map之前的地图数据。 |
| resultQueue | 一维DWORD类型的数组。 | 逻辑部分返回的结果队列，其中存有要执行的动画。 |
| movedis | DWORD | 方块偏移距离，用于画移动中的方块 |
| scale | DWORD | 方块缩放比例，用于画缩放中的方块 |

下面说明四个动画的实现和绘制细节。

#### 方块移动

若处于移动动画阶段，在遍历oldmap过程中，若某个格子的值大于0，则调用GetMoveDis获取移动格数，然后计算movedis = (SquareWidth + Padding) × moveticks / MoveTime。在movedis不断增大的过程中可以画出移动中的方块。

在移动动画结束，将aniFlag = 2，要调用CopyMapWithoutBomb，将新地图爆炸前的数据拷贝到oldmap中。

#### 方块合成

在所有动画开始前，先调用CheckHasCombBomb，扫一遍resultQueue判断是否有合成和爆炸动画。

若没有合成动画，则自动跳到下一个动画。

若处于合成动画阶段，同样在遍历oldmap的过程中，若某格的值大于0，则调用CheckCombine判断这个格子是否有合成动画。若该格有合成动画，则计算scale。

若combineticks < CombineTime / 2, scale = 110 – 60 × 2 × combineticks / CombineTime。

若combineticks >= CombineTime / 2, scale = 50 + 50 × (combineticks – CombineTime / 2) / (CombineTime / 2)

这样就可以做出先缩小后放大的合成动画。

#### 爆炸

同样，若没有爆炸动画，则自动跳到下一个动画。

若处于爆炸动画阶段，同样在遍历oldmap时，遇到某格数值大于0，判断是否爆炸，若爆炸则在该格画一个从小变到大的炸弹图。画法同上面的合成，计算scale，scale = 50 + 50× explodeticks / ExplodeTime。

在爆炸动画结束后，调用CopyMap，将完整的新地图map拷贝到oldmap中。

#### 生成新方块

在第一次进入aniFlag = 4时，要调用AddNum在后台添加一个新方块，并把新方块的位置和数字记录下来。之后在PaintProc中判断，若处于生成新方块动画阶段，则在对应位置画一个从小变到大的方块。

在生成新方块动画结束后，不仅要将aniFlag置为0以及调用CopyMap。还要调用CheckMap判断游戏状态，是否赢或者输。

### 按键队列

为了使游戏玩起来更加流畅，动画的实现加入了按键队列。

#### 按键队列实现

按键队列的数据结构是AQueue，它是一个循环队列。宏AppendAQueue和宏ExtractAQueue分别用于插入数据和取出数据。

窗体接收到WM\_KEYDOWN消息时，会调用KeyDownProc过程处理。若按键为上下左右四个键之一，则调用AppendAQueue将该按键加入到按键队列中。

从按键队列中取出元素并执行的过程要用到变量aniLock和过程TryExtractAction，它们的作用如下：

aniLock：动画锁，没有执行动画时值为ANIUNLOCKED，执行动画时值为ANILOCKED。初始化时将aniLock置为ANUUNLOCKED

TryExtractAction：尝试从队列中取出按键，若队列中没有元素，则将aniLock置为ANIUNLOKCED。若成功取出元素，则执行对应逻辑操作，并开始动画（将aniFlag赋为1）。

若aniLock的值为ANIUNLOCKED，则在按完键后调用TryExtractAction；若aniLock的值为ANILOCKED，则依次在每个按键的所有动画结束后调用TryExtractAction，直至队列为空，将aniLock置为ANIUNLOCKED。

#### 动画变速

当用户快速按键的时候，动画速度需要变快，这样能给用户带来更好的体验。调节动画速度是通过调节动画时间（MoveTime, CombineTime, ShowNewTime, ExplodeTime）来实现的。每次响应WM\_TIMER调用TimerProc时，若处于动画的执行过程中（即aniFlag不为0），则会调用宏AdjustAniTimeLim调整动画时间。

AdjustAniTimeLim中根据按键队列中的元素个数来调整动画时间。若元素个数大于等于2，则动画时间为原来的1/2；若元素个数大于等于4，则动画时间为原来的1/4；若元素个数大于等于6，则动画时间为原来的1/8。最后，若元素个数为0，则恢复原动画时间。

## 文件存取

这部分主要涉及serialization.inc和serialization.asm两个源代码文件。

### 序列化

有必要保存的关键数据是：地图大小mapSize、地图map、合成目标bombTarget。将它们按顺序以二进制编码即可完成序列化。同样能方便地进行解序列化。

这里map将保存整个二维数组（而非只保存mapSize大小的地图），这样实现起来更简单，且给保存文件加入冗余信息，也是数据的一种保护。

### 文件保存

SaveSerialization过程实现文件保存，会按序列化顺序将二进制数据输出至工作目录下的bomb.wcc文件。

### 文件读取

LoadSerialization过程实现文件读取，会按序列化顺序将二进制数据从工作目录下bomb.wcc文件中读入。

# 总结

也不知道是谁规定的，实验文档通常有一个总结。

BOMB!从规则到实现都包含我们的耐心思考与劳动，虽然从操作方式、界面上能看到诸多2048游戏（<http://gabrielecirulli.github.io/2048/>）的影子，也都给用户带来“根本停不下来”的感觉，但我们针对规则的改进让游戏更具趣味性，同时引入的5\*5地图如果按常规方式很难取胜（因为理论上来说需要消耗大量时间），但通过我们特别设定的小trick（比如同时合成多个炸弹只将bombTarget翻倍一次）可以大大缩短取胜的用时。

实际上，BOMB!对游戏者控制特定地图块的能力有了更高的要求，游戏者应具备将特定地图块移动至特地新位置的能力，否则将造成大量无用的合成，进而导致陷入时间的无底洞。

总之，我们对BOMB!的游戏设定、汇编语言的实现效果都非常满意。