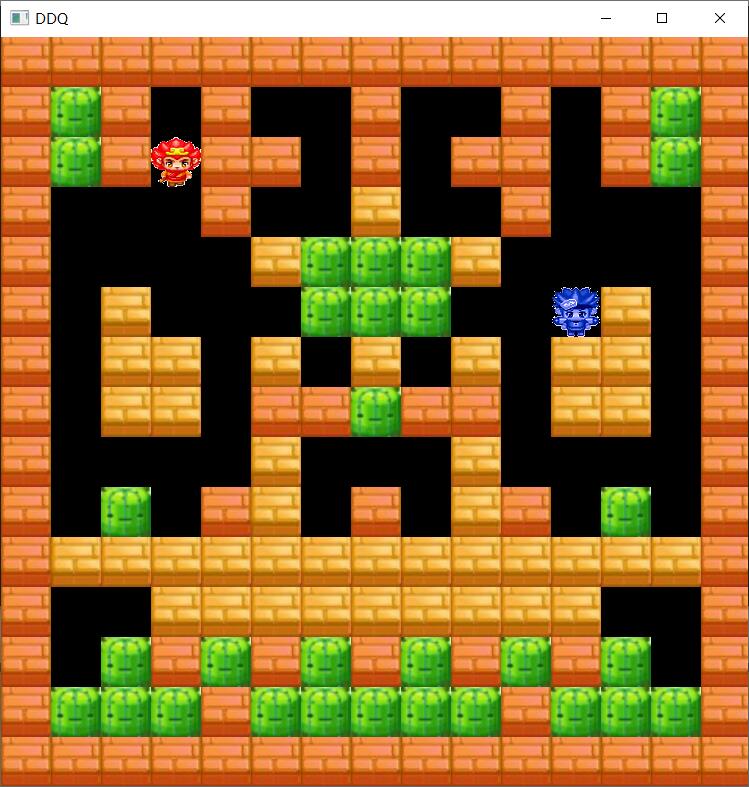
## 2.2 图形界面

界面设计总共分成三个部分，一是游戏开始界面，二是游戏运行界面，三是游戏结束界面。

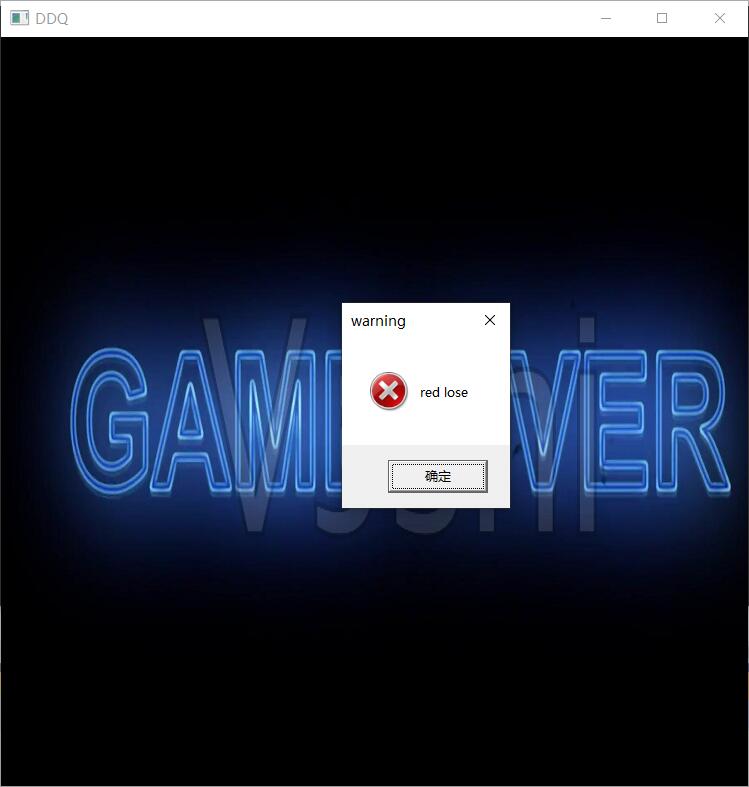
展示效果：



图片1 游戏开始界面



图片2 游戏执行界面



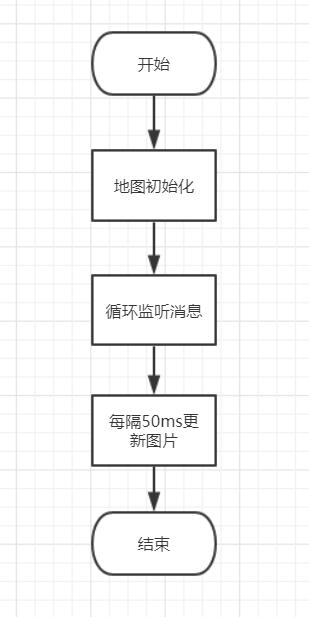
图片3 游戏结束界面

1. 整体思路：

初始地图g\_map为一个15\*15的全局数组，每一个数组所对应的图片为50\*50，根据数组元素中值的不同，从而对地图的不同位置展现不同的图片。具体值对应的状态图片如下：（炸弹部分不清楚）

|  |  |
| --- | --- |
| 数组的值 | 图片种类 |
| 0 | 空白 |
| 1 | 黄色砖头 |
| 2 | 橙色砖头 |
| 3 | 仙人掌 |
| 4 | 爆炸区域 |
| 8 | 人物1 |
| 9 | 人物2 |
| 10-15 | 5秒爆炸 |

（1）具体流程图如下：



整个游戏过程中，循环监听消息，并执行相应的回调函数，通过回调函数对g\_map数组的修改，每隔50ms对地图图片进行更新。

（2）代码展示：

invoke GameInit, @hWnd

.while TRUE

invoke PeekMessage, addr @msg, NULL, 0, 0, PM\_REMOVE ;等待消息

.if eax == 1

invoke TranslateMessage, addr @msg

invoke DispatchMessage, addr @msg

.endif

invoke GetTickCount

sub eax,g\_tPre

.if eax>=50 ;每隔50ms更新图片

;.if bomb1\_flag==1

; sub bomb1\_time,1

;.endif

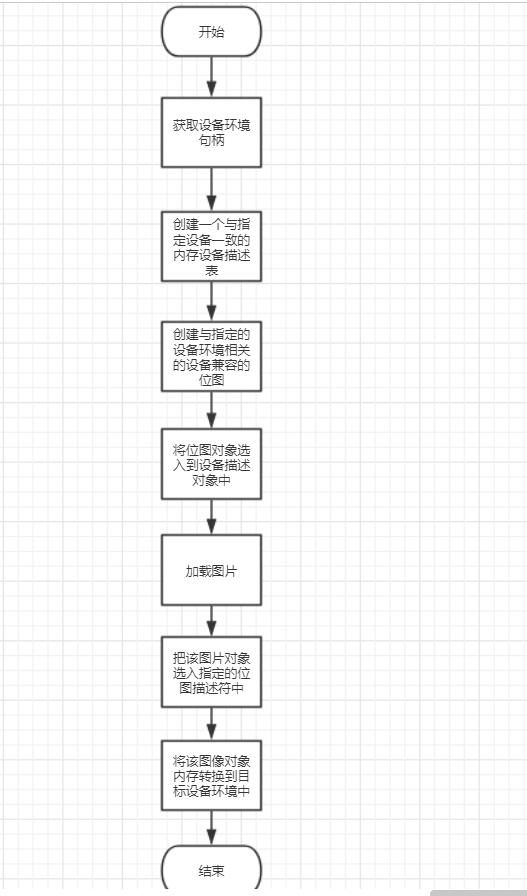
invoke GamePaint,@hWnd

.endif

.endw

2. 显示图片

（1）具体流程：



首先调用GetDC获得设备环境句柄，便于画图；因为绘图对象是位图，需要创建一个与位图句柄相联系的DC，也就是用函数CreateCompatibleDC来创建与目标句柄兼容的临时内存；再创建一个位图句柄CreateCompatibleBitmap，之后采用SelectObject函数选入到临时内存DC中，表示之后使用该临时内存对象时都是位图对象。

之后LoadImage加载位图资源，并且把该资源对象选入到临时内存中，用BitBlt将临时内存DC中的位图拷贝到目标DC上来。

这里由于GDI不支持透明通道，于是采用了图片的叠加，先用一个蒙版让需要显示的图片变成黑色，再用原图像覆盖上去即可。

（2）代码示例：（这里以显示小人图像为例）

invoke GetDC, hwnd *;提取句柄 便于画图*

mov g\_hdc, eax

invoke CreateCompatibleDC, g\_hdc ;创建设备上下文环境的句柄

mov g\_mdc, eax

invoke CreateCompatibleDC, g\_hdc

mov g\_bufdc, eax

invoke CreateCompatibleBitmap, g\_hdc, WINDOW\_WIDTH, WINDOW\_HEIGHT ;创建与指定的设备环境相关的设备兼容的位图

mov @bmp, eax

invoke SelectObject, g\_mdc, @bmp *;把一个对象(位图、画笔、画刷等)选入指定的设备描述表*

;加载图片

invoke LoadImage, NULL, offset g\_player0\_up\_1, IMAGE\_BITMAP, 200, 100, LR\_LOADFROMFILE

mov g\_hSpriteUp\_1, eax

invoke SelectObject, g\_bufdc, g\_hSpriteUp\_1

*;SRCAND: 使用与运算组合源设备和目标设备区域的颜色*

invoke BitBlt, g\_mdc, @man\_x\_rl, @man\_y\_rl, 50, 50, g\_bufdc, @framecoord, 50, SRCAND

;SRCPAINT:使用或运算组合源设备区域颜色和目标设备区域颜色

invoke BitBlt, g\_mdc, @man\_x\_rl, @man\_y\_rl, 50, 50, g\_bufdc, @framecoord, 0, SRCPAINT

invoke BitBlt, g\_hdc, 0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, g\_mdc, 0, 0, SRCCOPY

3. 背景音效

调用mciSendCommand函数，根据其中的MCI\_OPEN，MCI\_PLAY，MCI\_STOP打开音频文件、播放音频文件和停止播放。同时为了不阻塞窗口和其他逻辑函数的执行，创建一个新的线程播放音频。

具体代码如下：

Invoke CreateThread,NULL,0,offset PlayMusicThread,@hInstance,NULL,addr @dwThreadID

PlayMidiFile proc hWin:DWORD,NameOfFile:DWORD

local mciOpenParams:MCI\_OPEN\_PARMS,mciPlayParams:MCI\_PLAY\_PARMS

pushad

mov eax,hWin ;提取句柄

mov mciPlayParams.dwCallback,eax

mov eax,offset szMIDISeqr

mov mciOpenParams.lpstrDeviceType,eax

mov eax,NameOfFile

mov mciOpenParams.lpstrElementName,eax

invoke mciSendCommand,0,MCI\_OPEN,MCI\_OPEN\_TYPE or MCI\_OPEN\_ELEMENT,ADDR mciOpenParams

mov eax,mciOpenParams.wDeviceID

mov MidDeviceID,eax

invoke mciSendCommand,MidDeviceID,MCI\_PLAY,MCI\_NOTIFY,ADDR mciPlayParams ;播放

popad

ret

PlayMidiFile endp

PlayMusicThread proc hwin:DWORD

invoke PlayMidiFile,hwin,ADDR StarMidiName

ret

PlayMusicThread endp