扫雷开发文档

1. 引言：

1.1 背景：

“扫雷”是 Windows 操作系统自带的一款小游戏，以其简约的界面而不失逻辑推理的玩法深得广大玩家的喜爱。该游戏通过左键打开安全的格子，格子上的数字表明了在此格周围的八个格子中地雷的总数；用右键标记地雷、待定地雷；双击已开的格子自动开出周围八个未开格子。将所有非雷格子全部打开则游戏成功，开到有雷格子则游戏失败。

1.2 开发环境与工具：

经过 125 代表队成员的一致讨论，最终决定在 Visual C++6.0 开发环境下，用 C 语言以及从 easyX 图形库下载的扩充库<graphics.h>、标准输入输出库<stdio.h>、屏幕处理库<conio.h>、日期和时间头文件<time.h>、多媒体相关应用程序接口"winmm.lib"、Windows7 扫雷音效文件等完成此次开发。

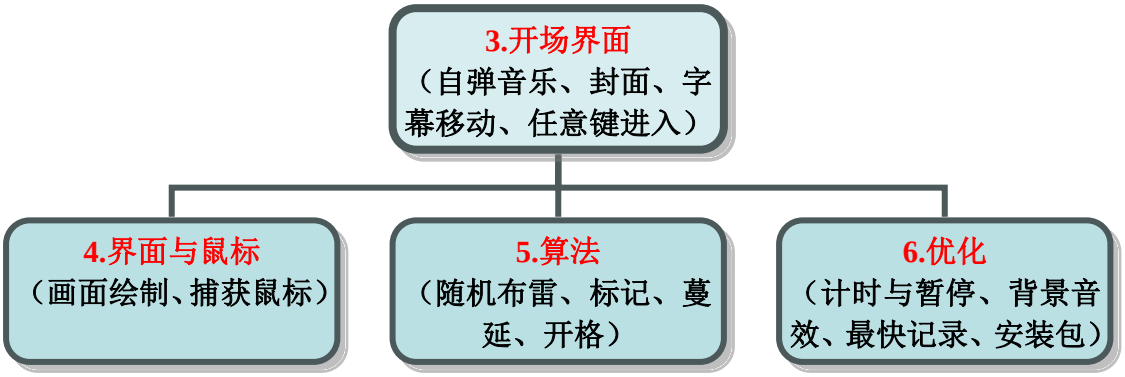
1.3 意义：

虽然并没有涉及到优秀的游戏编程语言 C++或 JAVA,也没有利用可观的界面框架 MFC，没有载入任何漂亮的图片。用的只是最基础的 C 语言，去操控着整个游戏画面的形成，都是用最简单的函数一点点地勾勒出来的界面，而算法也是整个团队在经过多个日夜的讨论与交流，实践与测试最终定下来的。其间，没有参考任何相关算法,全凭团队的团结协作、对扫雷这款游戏的热爱与敏锐的洞察力所成。尽管是大一新生，但我们至少学了 C 语言，这门基础而又不基础的语言，我们学以致用，能够从理论到达实际，这便是我们最大的收获。

2. 总体构架与分析：

2.1 设计流程：

游戏整体设计思路即从三大方面展开：界面与鼠标、算法、优化。下图简单表明了大概的设计流程：



2.2 ※基本思路与算法：

首先需要建立这个 9X9 表格，利用二维数组 map[9][9]存储每个格子的情况（1 就是有雷，0 就是无雷，当然这里用宏定义可以提高可读性，作为初学者我们务必要遵循这些良好的习惯,因此用 MINE 和 NOMINE 表示有无雷），另外还建立了一个_map[9][9]存储格子状态，由右键控制其变化（2 就是未翻，3 就是已翻，4 就是标记，5 就是待定）。当玩家左击下去，利用鼠标捕获函数获得用户点击的坐标(x,y)然后将它转换成数组下标（这样就实现了界面到代码化的过度）会出现两种情况，有雷与无雷，有雷则直接失败，无雷又分 9 种情况（编一个判断函数 judge()（后面会详细介绍）:以点击的这个格子为中心搜索周围 8 个的情况累计总数）。周围有 0—8 个雷，当是 1-8 个雷的时候则只会在你点下去的格子上显示数字并不会延展（这是最简单的情况也是递归的出口！），但当是 0 个的时候，则要进入递归去搜索以周围 8 个格子“各自”为中心的雷数情况。而这 8 个格子每个格子又都套用刚才“点击那个格子”的方法去计算他们各自周围的雷数。依次类推，而递归必须要出口，这个出口就是直到递归到表格的边界或递归到一个有数字的格子（可以从 Windows 上的扫雷的边缘观察，看是不是这样的）。另外还有开格函数 open()（后面会详细介绍），它也嵌套调用了 judge()函数,只不过添加了一些更为复杂的判断。至此，扫雷的基本思路就已经理清了。

2.3 头文件、宏定义、所有函数声明、全局变量：

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<time.h>
#pragma comment(lib,"winmm.lib")
/**********************************
/*宏定义*/
/**********************************
#define NOMINE 0/*无雷*/
#define MINE 1/*有雷*/
#define UNHIT 2/*未翻*/
#define HIT 3/*已翻*/
#define MARK 4/*红旗*/
#define UNDETERMINE 5/*问号*/
#define STOP 6/*暂停*/
#define KEEPON 7/*继续*/
/**********************************
/*函数声明*/
/**********************************
int  first();/*游戏开场画面*/
void menu();/*菜单栏*/
void move();/*难度方块移动效果*/
void stop();/*暂停图样*/
void keepon();/*继续图样*/
void picture(int x);/*难度方块*/
void cover(int x);/*覆盖难度方块*/
void extend();/*伸展开始*/
void amount9();/*9X9表格*/
void cover9();/*9X9覆盖表格*/
```

```
void amount16();/*16X16表格*/
void cover16();/*16X16覆盖表格*/
void clock(time_t t1);/*时间*/
void minenumber();/*剩雷数字*/
void mine(int x,int y);/*雷图样(坐标)*/
void blank(int a,int b);/*光标覆盖(下标)*/
void remove(int a,int b);/*光标还原(下标)*/
void colornumber(int m,int x,int y);/*彩色数字(下标)*/
void sign(int a,int b);/*红旗图样(下标)*/
void remain(int a,int b);/*问号图样(下标)*/
void number9(int n,int a,int b);/*9X9数字方块(下标)*/
void number16(int n,int a,int b);/*16X16数字方块(下标)*/
void judge9(int i,int j);/*9X9搜索(下标)*/
void open9(int i,int j);/*9X9开格(下标)*/
void judge16(int i,int j);/*16X16搜索(下标)*/
void open16(int i,int j);/*16X16开格(下标)*/
void sound9(int i,int j);/*9X9音效*/
void sound16(int i,int j);/*16X16音效*/
/*********************************************************************
/*主函数*/
/*********************************************************************
/*定义全局二维数组*/            /*存储内容*/
int map1[9][9],map2[16][16];/*MINE/NOMINE*/
int _map1[9][9],_map2[16][16];/*HIT/UNHIT/MARK/UNDETERMINE*/
int sum/*定义全局变量统计已翻方块个数*/,minenum/*计算剩余雷数*/;
time_t t1,t2,t3,t4;
//t1记录最先开始时间,t2刷新当前时间
//t3记录暂停按下的时间,t4记录继续按下的时间
```

2.4　主函数框架：main()

```
void main()/*主函数*/
{
    initgraph(640, 480);
    if(first())
    {
        cleardevice();
        menu();/*画菜单*/
        stop();/*初始化"暂停图样"*/
        MOUSEMSG m; /*定义鼠标变量*/
        while(true)/*"一重永真循环"用以捕获对"菜单栏"的鼠标信息
        {
            m=GetMouseMsg();
            if(m.uMsg==WM_LBUTTONDOWN)/*按下左键*/
                if((m.x>20&&m.x<140)&&(m.y>30&&m.y<80))/*识别玩
                {
                    printf("\a");
                    move();/*调用"难度方块移动效果"函数*/
                    extend();/*调用"伸展开始"函数*/
                }
                else if((m.x>20&&m.x<140)&&(m.y>230&&m.y<280))
                {
                    printf("\a");
                    closegraph();
                    exit(0);/*关闭窗口并退出操作系统*/
                }
        }
    }
}
```

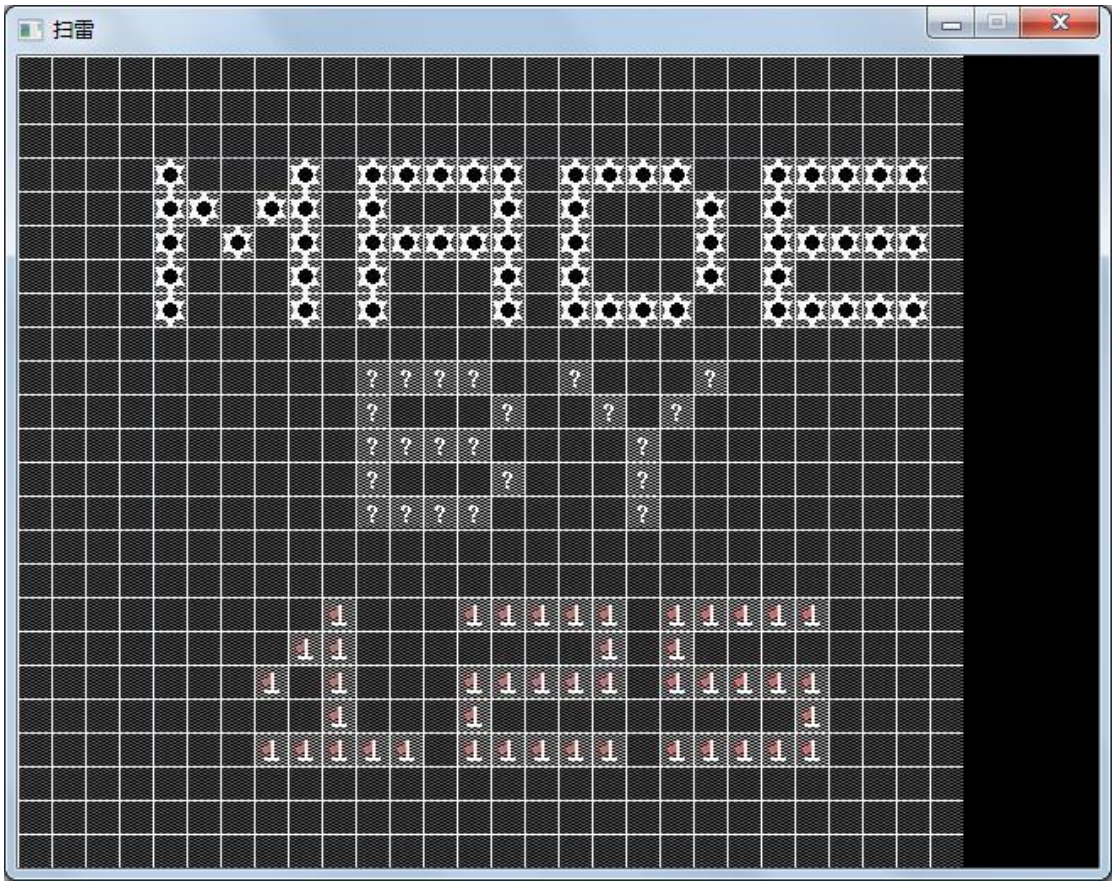
主函数中用 initgraph(640,480)创建一个 640*480 大小的图形窗口，包含对界面的初始化函数 menu() stop()，以及定义类 MOUSEMSG（其中用到了对象的坐标、以及鼠标信息（左键、右键、双击)），再在 whlie(true)永真循环内对鼠标的时时捕获函数 GetMouseMsg()，随时锁定了用户鼠标的所在位置。如果鼠标落在事先设定好的“难度方块”范围内则用 move()函数伸展出 9X9 16X16 的选择方块，在 extend()函数中进一步对用户的鼠标进行捕获。而如果鼠标落在事先设定好的“退出方块”范围内则 closegraph()退出操作系统。如此以来，会形成多重永真循环，在未到达出口时（即没有扫到雷或者用户没有主动点击退出)，那么是在游戏的进行阶段，当到达出口时（即用户扫到雷或者扫雷成功或主动点击退出)，那么将调用 MessageBox()函数询问用户是否要重来，是的话则嵌套调用 extend（）函数重新开始，否则退出操作系统。

2.5 ※游戏正式开始函数框架：extend()）

```
void extend()/*伸展开始*/
{
    /*"重来"后清空上局数据*/
    while(true)/*不停捕获在按了难度方块之后鼠标的消息*/
    {
        m=GetMouseMsg();/*获取鼠标*/
        /*如果"选难度后""按下左键"*/
        {
            /*1.如果"选难度后"按下"9    X    9"*/
            {
                /*画表*/
                /*布雷*/
                /*游戏开始*/
                /*获取初始时间*/
                while(true)/*不停捕获在"按了9X9后"鼠标的消息*/
                {
                    /*暂停初始化处理*/
                    m=GetMouseMsg();/*获取鼠标*/
                    /*1.如果"游戏中"鼠标在"9X9表格内"*/
                    {
                        /*鼠标光标移动效果*/
                        /*1.1如果"左键点击"*/
                        {
                            /*调用judge()搜索函数*/
                        }
                        /*1.2如果"右键点击"*/
                        {
                            /*标记、问号、还原*/
                        }
                        }
                        /*1.3如果"左键双击"*/
                        {
                            /*调用open()开格函数*/
                        }
                    }
                    /*2.如果"游戏中"鼠标点击"暂停键"*/
                    {
                        while(true)/*暂停处理*/
                        {
                            /*2.1如果"暂停中"鼠标点击"继续键"*/
                            {
                                /*时间处理*/
                                break;
                            }
                            /*2.2如果"暂停中"鼠标点击"重来"*/
                            /*2.3如果"暂停中"鼠标点击"退出"*/
                        }
                    }
                    /*3.如果"游戏中"鼠标点击重来键*/
                    /*4.如果"游戏中"鼠标点击退出键*/
            /*2.如果"选难度后"按下"16    X    16"*/
            {
                /*同9X9*/
            }
            /*3.如果"选难度后"按下"退出"*/
```

3. 开场界面：此界面被封装在 int first()函数中,当用户按下任意键时返回1,主函数响应其值，进入游戏主界面。

3.1　自弹音乐：随着用 Beep()函数编写的的背景音乐“星之所在”的响起，屏幕逐渐打印出如下图示。(此部分代码单独编写在“音符.cpp”文件中)



(自弹音乐下的开场界面 1 图示)

※算法详解：利用<windows.h>头文件中的 Beep(频率值，时间间隔)可以蜂鸣出不同的音效。通过参考网站上提供的 do,ri,mi,fa,so,la,xi 不同音阶对应的频率值，可用 switch()筛选出传递来的音符和节拍，并将单个的音符声响编写成一个 Note(int note,int time)函数。而对于乐曲“星之所在”，又通过参考网站上的乐谱将所有音符“频率化”，将所有节拍“时间化”(定义宏 E 代表八分音符，F 代表四分音符)，就构成了所需参数。而频率和时间分别储存于两个含有 142 个元素的数组 note[]和 time[]中。再利用指针推移可提取对应音符应该声响的秒数。并将调用 Note()函数的这个主调函数取名为 Music()，再在 first()函数中调用 Music()。这样就模拟了弹奏。而对于画面的逐渐显示，则通过建立 IMAGE 类型对象 img,调用 loadimage()(加载指定路径位图到 img 中缓存)和 putimage()(在指定坐标处显示在位图对象里以指定坐标为起始点的指定宽度、高度的位图)。最终构成函数 Picture()，只要在 Note()函数之前调用次函数，那么每次声响都会伴随着画面的一点点推移。下面是“音符.cpp”文件中关于弹奏的代码：

```
void Picture(int i)/*封面*/
{
    putimage(i,0,5,480,&img,i,0);/*在以
}
/*************************************
void Note(int note,int time)/*音符*/
{
    switch(note)/*筛选音符*/
    {
        /*升调:#5*/
        case 25:Beep(831,time);break;

        /*中音:0,1,2,3,4,5,6,7*/
        case 0:Beep(0  ,time);break;
        case 1:Beep(523,time);break;
        case 2:Beep(578,time);break;
        case 3:Beep(659,time);break;
        case 4:Beep(698,time);break;
        case 5:Beep(784,time);break;
        case 6:Beep(880,time);break;
        case 7:Beep(988,time);break;

        /*高音:1,2,3,4,5,6,7*/
        case 11:Beep(1046,time);break;
        case 12:Beep(1175,time);break;

        case 13:Beep(1318,time);break;
        case 14:Beep(1397,time);break;
        case 15:Beep(1568,time);break;
        case 16:Beep(1760,time);break;
        case 17:Beep(1976,time);break;
    }
}
/*************************************
void Music()/*音乐播放*/
{
    loadimage(&img,"\\扫雷\\图片\\封面

    /*存储乐曲的142个音符(频率)*/
    int note[142]={11,7,11,13,7,6,5,6,
                   12,15,14,13,12,11,7
                   13,12,11,7,25,6,11,
                   6};
    /*存储乐曲的142个节拍(时间)*/
    int time[142]={E,E,E,E,E+F,E,E,E,E
                   E,2*F+E,E,E,F,F,E,F
                   E,E,E,E+F,E,E,E,E,E

    int *pnote=note,*ptime=time,i=0;/*
    for(;pnote<=note+141;pnote++,ptime
    {
        Picture(i);/*打印图片*/
        Note(*pnote,*ptime);/*传入音符
    }
    cleardevice();/*清屏进入游戏*/
```

（开场界面 2 图示）

3.2　雷图样的绘制：

（1）　先画出一个正立的正三角形，再画出一个倒立的正三角形。三角形属于多边形所以可以用填充多边形函数 fillpoly(顶点数，存有各点横纵坐标的数组地址)进行显示。

（2）　再在经过计算的坐标处画出填充圆 fillcircle(圆心横坐标，圆心纵坐标，圆半径)（在此之前要用 setfillstyle(颜色)声明接下来的填充颜色，默认的填充颜色是白色，那么画黑色圆则要用 setfillstyle(BLACK)）

3.3　“125 代表队”字样显示：outtextxy(字样横坐标,纵坐标,存字数组)

3.4　“按任意键进入”字样往返移动效果：

※算法详解：设想这一排字在移动，利用 outtextxy()可以在指定位置打印指定的文字，而字样只需在固定的纵坐标处以不同的横坐标打印出来即可，相当于 x 是每次自增 1，而 y 不变，那么用 for 循环就可以很容易实现。但是字幕要有移动的效果，必须兼具两点：第一，字幕要在原地停留小段时间；第二，上一时刻的字幕要被覆盖成原来的样子。至于停留，我们可以用 Sleep(毫秒数)暂停。这样就解决了单向移动问题，至于字幕到达右端又返回的过程可以以此类推。但是我们在此其间可以在任意时刻按任意键，我们先来解决“任意时刻”，显然如果为原 for 语句套上一个 while(true){……}永真循环那么就能让字幕一直往返运动了，但是这样的话就没有了出口，到底何时才是个尽头呢？巧妙的是我们要按任意键啊~于是在 while 语句里面加上一个 if(kbhit())如果有按键就返回主

函数开始游戏。下面是代码截图：

```
IMAGE tmp;
getimage(&tmp,0,0,textwidth("按任意键进入"),
        textheight("按任意键进入"));/*保存当前图像*/
while(true)/*如果不按任意键就一直移动*/
{
    for(int i=0;i<510;i++)/*字样在横坐标0-510范围移动*/
    {
        outtextxy(i,450,"按任意键进入");/*"按任意键进入"字样
        Sleep(20);/*延时*/
        putimage(i,450,&tmp);/*打印当前图像*/
        if(kbhit())/*按键信息*/
        {
            printf("\a");
            return 1;/*按键返回1*/
        }

    }
    for(;i>0;i--)/*循环移动*/
    {
        outtextxy(i,450,"按任意键进入");
        Sleep(20);
        putimage(i,450,&tmp);
        if(kbhit())
        {
            printf("\a");
            return 1;
        }
    }
}
```

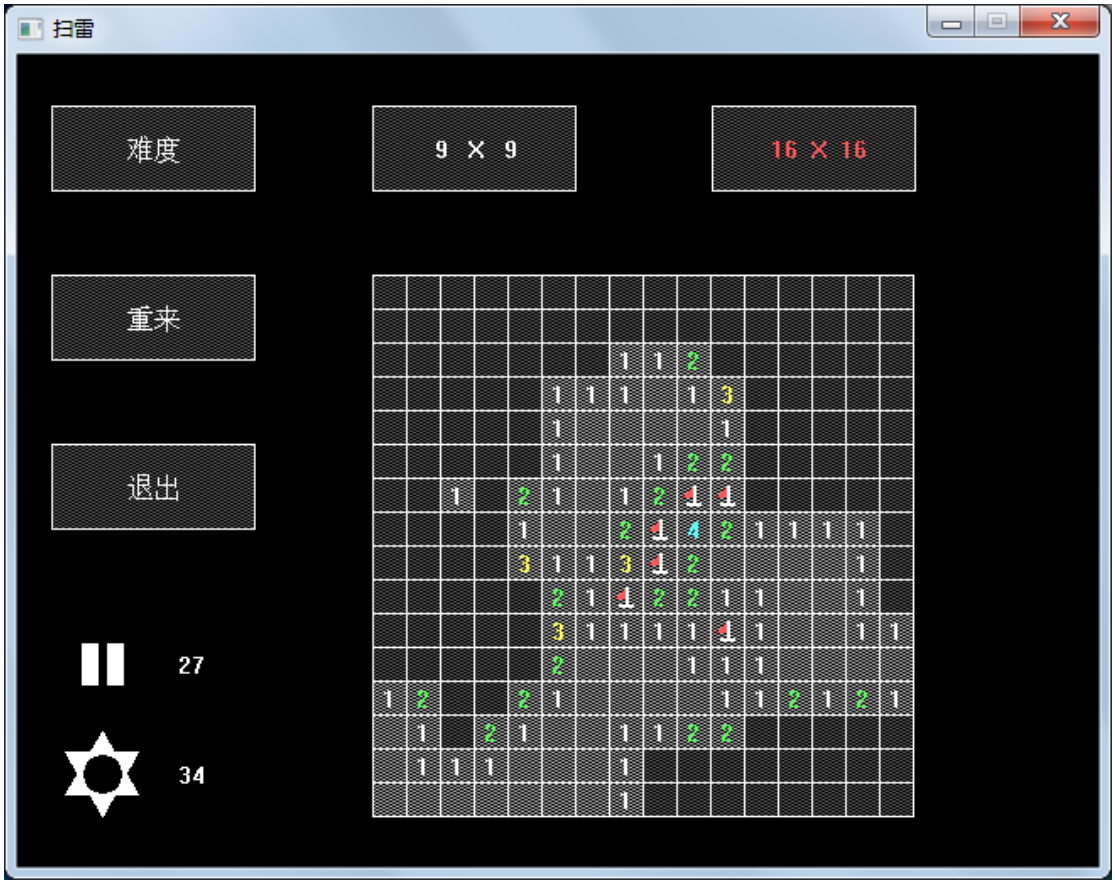
下面是对代码的详细解释：

（1） 外部定义一个 IMAGE 类的对象 tmp,此对象用于存储指定坐标区域的位图。

（2） 在 while(true)循环外利用 getimage()函数截取当前字样连同其背景的矩形存储于 tmp 对象中，getimage(&tmp,i,450,textwidth("按任意键进入"),textheight("按任意键进入"))其中&tmp 表明了要存入的地址，i 是上述所说的在推移的横坐标，450 是不变的纵坐标，textwidth（）\textheight（）的作用是获取按引号内字串的宽度和高度。这样以来，就 tmp 中就存储了一个同“按任意键进入”宽度高度相同的长条黑色矩形。

（3） 在 for 语句里用 outtextxy(i,450,“按任意键进入”)打印字样，又用 Sleep(20)让字样暂留一小段时间后，再用 putimage(i,450,&tmp)放出刚才保存的黑色矩形覆盖字样。

4 界面与鼠标：



（游戏界面图示）

4.1　矩形与网格的绘制：

```
void amount9()/*9X9表格*/
{
    setcolor(WHITE);/*声明现在的边框颜色*/
    setfillstyle(DARKGRAY,24);/*声明现在的填充颜色*/
    bar(210,130,390,310);/*填充矩形范围*/
    for(int k=0;k<10;k++)/*画9x9的格子线*/
    {
        line(210+20*k,130,210+20*k,310);
        line(210,130+20*k,390,130+20*k);
    }
}
```

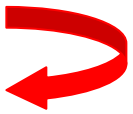
用 bar(左上 x,左上 y,右下 x,右下 y)绘制一个灰色矩形，并在利用循环推移 k 变量的增加，在内用 line(起点 x,起点 y,终点 x,终点 y)为矩形绘制表格。

4.2　鼠标与数组下标，数组下标与中心的关系：

之前定义了 MouseMsg 类型的对象 m,又时时利用 GetMouseMsg()获取了鼠标的信息，其中的 m.uMsg 中存有鼠标的横纵坐标。但是对于用户的点击，我们并不希望就在他点击的位置打印数字，标记红旗等，我们希望的是在表格中的小方块中心打印图样，这样会非常美观。不仅如此，还要对表格内的情况进行记录，如哪些位置有雷，哪些位置无雷，哪些已经翻过，哪些被标记过。下面就是对这两种转换（其中（130，210）是 9x9 表格的左上坐标，20 是每个小方格的边长）：

点击位置坐标->数组下标：i=(m.y-130)/20;j=(m.x-210)/20;

数组下标->表格中心：x=210+20*j;y=130+20*i;



4.3　各种图样绘制：

（1）暂停与继续图样：

```
void stop()/*暂停图样*/
{
    setcolor(WHITE);
    setfillstyle(WHITE);
    bar(38,348,47,372);
    bar(53,348,62,372);
}
/*****************************
void keepon()/*继续图样*/
{
    int points[6];
    points[0]=38;
    points[1]=348;
    points[2]=38;
    points[3]=372;
    points[4]=62;
    points[5]=360;
    setcolor(WHITE);
    setfillstyle(WHITE);
    fillpoly(3,points);
}
```

两个矩形构成一个暂停按钮，一个三角形构成一个启动按钮。而三角形同上文中的雷图样的画法类似，均是用填充多边形函数 fillpoly()绘制。

（2）选中小方块与底色小方块的绘制：

```
void blank(int a,int b)/*光标覆盖(下标)*/
{
    setcolor(WHITE);
    setfillstyle(LIGHTGRAY,24);
    fillrectangle(210+20*b,130+20*a,210+20*b+20,130+20*a+20);
}
/*****************************************************************
void remove(int a,int b)/*光标还原(下标)*/
{
    setcolor(WHITE);
    setfillstyle(DARKGRAY,24);
    fillrectangle(210+20*b,130+20*a,210+20*b+20,130+20*a+20);
}
```

利用 fillrectangle()函数绘制带有边框的填充矩形，而两者的不同则是选中小方块是浅灰色的，底色小方块是和底色相同的深灰色。另外还需注意，此函数之所以带参，说明它并不是一个在指定位置打印的图片的函数。与用户点下的位置密切相关，其打印的位置是按“点击位置坐标->数组下标（实参）”转换，而这样以来，再在函数中经过“数组下标（形参）->表格中心处”的转换即可达到打印的目的（之所以不直接传入点击处的坐标值，是缘于在编写过程中需要用

数组下标去标记表格内的有雷、无雷、点击过、未点击过等情况，所以重用避免混淆）。**这样可以在之后绘制雷、小红旗、问号（前三项如法炮制，不再敷述）、彩色数字、光标选中效果中嵌套调用**，提供方便。

（3） 彩色数字的绘制：

```
void colornumber(int m,int x,int y)/*彩色数字*/
{
    char str[2];
    blank(x,y);
    switch(m)/*根据不同数字显示不同颜色*/
    {
        case 1:
            {
                setcolor(WHITE);
                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
        case 2:
            {
                setcolor(LIGHTGREEN);
                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
        case 3:
            {
                setcolor(YELLOW);
                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
        case 4:
            {

                setcolor(LIGHTCYAN);
                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
        case 5:
            {
                setcolor(LIGHTRED);
                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
        case 6:
            {
                setcolor(BLACK);
                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
        case 7:
            {
                setcolor(CYAN);
                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
        case 8:
            {
                setcolor(LIGHTMAGENTA);

                setbkmode(TRANSPARENT);
                sprintf(str,"%d",m);
                outtextxy(210+20*y+6,130+20*x+3,str);
            }break;
    }
}
```

此函数 colornumber(m,x,y)中第一个参数是指周围的雷数目，其余就是上文提到的数组下标。具体过程则是当用户按下一个非雷格子的时候出现以浅灰色（表示已经翻过)的格子为背景的 1-8 不同颜色的数字。显然可以先用 blank(x,y)打印一个已翻的格子图样，再用 switch..case..语句筛选这 8 个数字，不同的 case 对应不同的颜色。**而需要注意一点的是 outtextxy()函数的形参是数组地址，那么不能直接把整型变量写入，必须用 sprintf(数组地址，原始数据的格式，原始变量)进行转换。**

4.4 鼠标移动时的选中效果：

```
int  lastx=1,lasty=1;/*存储上一时刻鼠标所在位置*/

if(_map1[(lasty-130)/20][(lastx-210)/20]==UNHIT)
/*第一次是充数,之后每一次都要检验
上一时刻鼠标所在位置是否被扫过*/
    remove((lasty-130)/20,(lastx-210)/20);
/*还原底色:第一次是充数,之后每一次都要
把上一时刻遗留下来的光标覆盖还原成底色*/
if((m.x>210&&m.x<390)&&(m.y>130&&m.y<310))/*鼠标在9X9表格范围内*/
{
    if(_map1[(m.y-130)/20][(m.x-210)/20]==UNHIT)/*鼠标所在位置是"未翻"*/
    {
        blank((m.y-130)/20,(m.x-210)/20);
        lastx=m.x;lasty=m.y;/*存储当前时刻鼠标横纵坐标,以备下一次覆盖使用*/
    }
```

※基本算法：定义整型变量 lastx,lasty 存储上一时刻鼠标所在的位置。“此时”鼠标坐标经转换成数组下标,再用全局数组 _map1[x][y]判断鼠标“此时”所在位置是否“未翻”，如果未翻，则用 blank()函数将鼠标所在位置指向的小方块变成浅灰色，这样就实现了“选中状态”。但是 lastx,lasty 还没有起到作用，我们可以将“此时”的坐标赋给“lastx,lasty”，这样当下一次 while(true)循环时，lastx,lasty 就拥有的是上一时刻鼠标的位置。于是趁循环才开始，就把上一时刻所在的位置用 remove()覆盖还原成底色。

5 算法：

5.2 初始化与随机布雷：

（1）初始化：

```
for(x=0;x<9;x++)/*初始化所有方块"未翻"*/
    for(y=0;y<9;y++)
        _map1[x][y]=UNHIT;/*UNHIT表示"未翻"*/

for(x=0;x<9;x++)/*初始化所有方块"无雷"*/
    for(y=0;y<9;y++)
        map1[x][y]=NOMINE;/*NOMINE表示"无雷"*/
```

利用两个循环将_map1[9][9]与 map1[9][9]进行了初始化。分别将这 81 个格子的状态用 UNHIT 和 NOMINE 来表征。前者表征了所有格子的附加状态（右键），后者表征了格子有无雷这一重要状态,两者分开使用使得代码更加清晰,有条理。

（2）随机布雷：

```
srand((unsigned)time(NULL));/*利用time(NULL)获取时间实现播种*/
while(true)
{
    if(s<10)/*限制雷数10个*/
    {
        x=rand()%9;y=rand()%9;/*数组下标为0-8的随机整数*/
        if(map1[x][y]!=MINE)/*MINE表示"有雷"*/
        {
            map1[x][y]=MINE;
            s++;/*统计已布雷的个数*/
        }
    }
    else break;
}
```

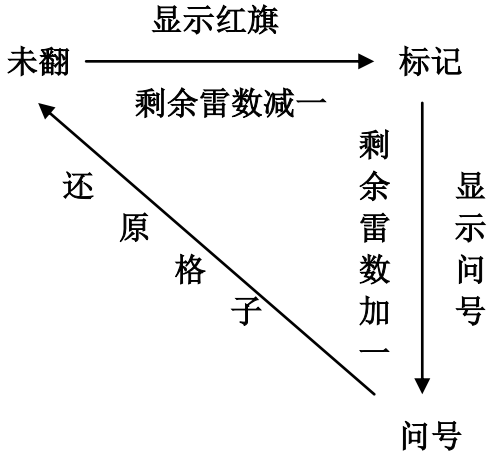
利用随机数发生器的初始化函数 srand()实现随机种子的产生，其参数是 unsigned 型的变量，并且每一个不同的值都对应着一组不同的随机数。要想每次都是不同组的随机数，那么可以调用 time(NULL)函数获取当前系统的时间并将其强制转换成 unsigned 型，就能达到此目的。此后，它同随机数产生函数 rand()联合使用才可以达到产生不同组随机数的目的。下面分析代码：代码中用永真循环实现了随机布雷，其中 rand()%9 表明产生 0-8 的随机数（套用公式 a-b 的随机数 a+rand()%(b-a+1)），同时用 s 记录已经随机布雷的个数，满 10 个后则退出循环。

5.3 标记：

```
if(m.uMsg==WM_RBUTTONDOWN)/*右键点击*/
{
    x=(m.y-130)/20;y=(m.x-210)/20;
    switch(_map1[x][y])
    {
        case UNHIT:/*鼠标所在位置是"未翻"*/
        {
            sign(x,y);/*红旗图样*/
            _map1[x][y]=MARK;
            minenum--;
            minenumber();
        }break;
        case MARK:/*鼠标所在位置是"红旗"*/
        {
            remain(x,y);/*问号图样*/
            _map1[x][y]=UNDETERMINE;
            minenum++;
            minenumber();
        }break;
        case UNDETERMINE:/*鼠标所在位置是"问号"*/
        {
            blank(x,y);/*空方块*/
            _map1[x][y]=UNHIT;
        }break;
    }
}
```

当点击右键的时候，转换其鼠标坐标为数组下标，利用 switch..case..语句

对_map1[x][y]进行筛选。其中 minenum 表示显示在左下位置记录剩余雷数个数的变量，minenumber()函数负责打印左下角剩余雷数的个数。下面的流程很清晰的表明了这段代码的用意：



5.3 ※蔓延：此程序段是本游戏的核心算法。

```
void judge9(int i,int j)/*9X9搜索(下标)*/
{
    int num=0;/*统计九宫格内雷数*/
    if(_map1[i][j]==UNHIT||_map1[i][j]==UNDETERMINE)/*该位置是未翻方块或是问号方块*/
    {
        switch(map1[i][j])
        {
            case MINE:/*该位置有雷*/
            {
                for(int e=0;e<9;e++)/*摊牌布下的雷图样*/
                    for(int f=0;f<9;f++)
                        if(map1[e][f]==1)
                            mine(210+20*f+10,130+20*e);
                HWND wnd=GetHWnd();/*游戏结束对话框*/
                if (MessageBox(wnd,"扫雷失败!\n是否重来?","询问",MB_YESNO|MB_ICONQUE
                    {outtextxy(211+36,48,"9   X   9");cover9();extend();}
                else
                    {closegraph();exit(0);}
            }
            case NOMINE:/*该位置无雷并搜索九宫格内雷数目*/
            {
                if(map1[i-1][j-1]==MINE&&((i-1)>=0&&(i-1)<=8)&&((j-1)>=0&&(j-1)<=8))
                    num++;/*若有雷则num增1*/
                if(map1[i-1][j]==MINE&&((i-1)>=0&&(i-1)<=8)&&(j>=0&&j<=8))
                    num++;
                if(map1[i-1][j+1]==MINE&&((i-1)>=0&&(i-1)<=8)&&((j+1)>=0&&(j+1)<=8))
                    num++;
                if(map1[i][j-1]==MINE&&(i>=0&&i<=8)&&((j-1)>=0&&(j-1)<=8))
                    num++;
```

```
            if(map1[i][j+1]==MINE&&(i>=0&&i<=8)&&((j+1)>=0&&(j+1)<=8))
                num++;
            if(map1[i+1][j-1]==MINE&&((i+1)>=0&&(i+1)<=8)&&((j-1)>=0&&(j-1)<=8))
                num++;
            if(map1[i+1][j]==MINE&&((i+1)>=0&&(i+1)<=8)&&(j>=0&&j<=8))
                num++;
            if(map1[i+1][j+1]==MINE&&((i+1)>=0&&(i+1)<=8)&&((j+1)>=0&&(j+1)<=8))
                num++;
        /*若该位置九宫格内无雷,则利用递归继续以九宫格为中心查找他们8个方块各自九
            if(num==NOMINE)
            {
                number9(0,i,j);/*打印空白方块图样*/
                if((i-1)>=0&&(i-1)<=8&&((j-1)>=0&&(j-1)<=8))
                    judge9(i-1,j-1);/*递归查找*/
                if((i-1)>=0&&(i-1)<=8&&(j>=0&&j<=8))
                    judge9(i-1,j);
                if((i-1)>=0&&(i-1)<=8&&((j+1)>=0&&(j+1)<=8))
                    judge9(i-1,j+1);
                if((i>=0&&i<=8)&&((j-1)>=0&&(j-1)<=8))
                    judge9(i,j-1);
                if((i>=0&&i<=8)&&((j+1)>=0&&(j+1)<=8))
                    judge9(i,j+1);
                if((i+1)>=0&&(i+1)<=8&&((j-1)>=0&&(j-1)<=8))
                    judge9(i+1,j-1);
                if((i+1)>=0&&(i+1)<=8&&(j>=0&&j<=8))
                    judge9(i+1,j);
                if((i+1)>=0&&(i+1)<=8&&((j+1)>=0&&(j+1)<=8))
                    judge9(i+1,j+1);
            }

            else
            {
                number9(num,i,j);/*打印num数字方块图样*/
                return;/*退出子函数(递归出口)*/
            }
        }
    }
}
```

（浅灰色蔓延图示）

※**算法详解：**首先定义一个整型变量 num 用于统计以形参 i,j 为数组下标格子周

围 8 个格子的总雷数。并且这种搜索的前提必须是此格子“未翻”或“待定”。满足了此前提后用 switch..case..语句进行一重筛选，筛选出此格子是“有雷”还是“无雷”。

如果是“有雷”，那么表明游戏失败，于是用 for 语句摊牌所有布下的雷，将他们全部都打印出来。并且建立一个 HWND 类的对象 wnd，利用 GetHWnd()获取窗口的句柄作为 MessageBox()的第一个参数。如果玩家选择的是 Yes 按钮则：1.将 9X9 选项的红色字体用 outtextxy()还原成白色。2.用 cover()将上局的表格覆盖成黑色。3.嵌套调用 extend()函数，重新开始。如果玩家选择的是 No 则退出。

如果是“无雷”，那么就要对以此格子为中心周围 8 个格子进行搜索（除了检查是否有雷，还需注意数组下标必须大于等于 0，这样就排除了边界情况）。map1[i-1][j-1],map1[i-1][j],map1[i-1][j+1],map1[i][j-1],map1[i][j+1],map1[i+1][j-1],map1[i+1][j],map1[i+1][j+1]。这八个数组变量分别表示这 8 个格子的有无雷情况。如果有雷，则 num++。最后如果 0<num<=8，那么就用 number()这个函数进行数字的打印、已翻的格子的统计、胜负的判定（如果胜利，其中包含文件读取，读取历史最快记录，同玩家此次游戏所用时间进行比较，进而弹出不同的对话框）；如果 num=0，则说明无雷，则先用 number()函数在此格上打印一个空白浅灰色格子。再在不越界的情况下以周围 8 个格子为中心搜索他们各自周围的总雷数情况（递归调用 judge()函数）。这样，当递归到“边界”或“有雷的格子”时，则此次 judge()就已经找到出口。

5.4 ※开格：这也是本游戏非常重要的算法。

```
void open9(int i,int j)/*9X9开格(下标)*/
{
    int num=0/*记录有雷数目*/,all=0/*记录红旗和有雷匹配数目*/;
    /*检测有雷数*/
    if(map1[i-1][j-1]==MINE&&((i-1)>=0&&(i-1)<=8)&&((j-1)>=0&&(j-1)<=8))
        num++;
    if(map1[i-1][j]==MINE&&((i-1)>=0&&(i-1)<=8)&&(j>=0&&j<=8))
        num++;
    if(map1[i-1][j+1]==MINE&&((i-1)>=0&&(i-1)<=8)&&((j+1)>=0&&(j+1)<=8))
        num++;
    if(map1[i][j-1]==MINE&&(i>=0&&i<=8)&&((j-1)>=0&&(j-1)<=8))
        num++;
    if(map1[i][j+1]==MINE&&(i>=0&&i<=8)&&((j+1)>=0&&(j+1)<=8))
        num++;
    if(map1[i+1][j-1]==MINE&&((i+1)>=0&&(i+1)<=8)&&((j-1)>=0&&(j-1)<=8))
        num++;
    if(map1[i+1][j]==MINE&&((i+1)>=0&&(i+1)<=8)&&(j>=0&&j<=8))
        num++;
    if(map1[i+1][j+1]==MINE&&((i+1)>=0&&(i+1)<=8)&&((j+1)>=0&&(j+1)<=8))
        num++;
    /*检测红旗和有雷位置是否匹配*/
    if(map1[i-1][j-1]==MINE&&_map1[i-1][j-1]==4&&((i-1)>=0&&(i-1)<=8)&&((j-1)>=0&&(j
        all++;
    if(map1[i-1][j]==MINE&&_map1[i-1][j]==4&&((i-1)>=0&&(i-1)<=8)&&(j>=0&&j<=8))
        all++;
    if(map1[i-1][j+1]==MINE&&_map1[i-1][j+1]==4&&((i-1)>=0&&(i-1)<=8)&&((j+1)>=0&&(j
        all++;
    if(map1[i][j-1]==MINE&&_map1[i][j-1]==4&&(i>=0&&i<=8)&&((j-1)>=0&&(j-1)<=8))
        all++;
```

```
    if(map1[i][j+1]==MINE&&_map1[i][j+1]==4&&(i>=0&&i<=8)&&((j+1)>=0&&(j+1)<=8))
        all++;
    if(map1[i+1][j-1]==MINE&&_map1[i+1][j-1]==4&&((i+1)>=0&&(i+1)<=8)&&((j-1)>=0&&(j
        all++;
    if(map1[i+1][j]==MINE&&_map1[i+1][j]==4&&((i+1)>=0&&(i+1)<=8)&&(j>=0&&j<=8))
        all++;
    if(map1[i+1][j+1]==MINE&&_map1[i+1][j+1]==4&&((i+1)>=0&&(i+1)<=8)&&((j+1)>=0&&(j
        all++;
    /*匹配*/
    if(all==num)
    {
        printf("\a");/*点击音效*/
        if(map1[i-1][j-1]!=MINE&&((i-1)>=0&&(i-1)<=8)&&((j-1)>=0&&(j-1)<=8))/*排除以
            judge9(i-1,j-1);/*递归查找*/
        if(map1[i-1][j]!=MINE&&((i-1)>=0&&(i-1)<=8)&&(j>=0&&j<=8))
            judge9(i-1,j);
        if(map1[i-1][j+1]!=MINE&&((i-1)>=0&&(i-1)<=8)&&((j+1)>=0&&(j+1)<=8))
            judge9(i-1,j+1);
        if(map1[i][j-1]!=MINE&&(i>=0&&i<=8)&&((j-1)>=0&&(j-1)<=8))
            judge9(i,j-1);
        if(map1[i][j+1]!=MINE&&(i>=0&&i<=8)&&((j+1)>=0&&(j+1)<=8))
            judge9(i,j+1);
        if(map1[i+1][j-1]!=MINE&&((i+1)>=0&&(i+1)<=8)&&((j-1)>=0&&(j-1)<=8))
            judge9(i+1,j-1);
        if(map1[i+1][j]!=MINE&&((i+1)>=0&&(i+1)<=8)&&(j>=0&&j<=8))
            judge9(i+1,j);
        if(map1[i+1][j+1]!=MINE&&((i+1)>=0&&(i+1)<=8)&&((j+1)>=0&&(j+1)<=8))
            judge9(i+1,j+1);
    }
    /*不匹配*/
    else
    {
        PlaySound("\\扫雷\\音效\\Lose.wav",NULL,SND_FILENAME|SND_ASYNC);/*爆炸音效*/
        for(int e=0;e<9;e++)/*摊牌布下的雷*/
            for(int f=0;f<9;f++)
                if(map1[e][f]==1)
                    mine(210+20*f+10,130+20*e);
        HWND wnd=GetHWnd();/*游戏结束对话框*/
        if (MessageBox(wnd,"扫雷失败!\n是否重来?","询问",MB_YESNO|MB_ICONQUESTION)==
            {outtextxy(211+36,48,"9   X   9");cover9();extend();}
        else {closegraph();exit(0);}
    }
}
```

※算法详解：所谓开格，即指当玩家在以某个格子为中心，在其周围标记了已经确定的雷之后，对着中心格子左键双击，电脑自动帮你把其余的格子打开的过程。但是如果标记错误，那么就直接失败，避免了玩家在游戏中投机取巧的 bug。那么这个过程是具体应该怎么实现呢？下面依次详细解释它的含义：

（1）同 judge()函数中一样搜索周围总雷数 num。

（2）变量 all 用于检测**“雷的位置是否和红旗位置匹配”**，如果匹配则 all++。

（3）如果 all==num，证明匹配数和总数相同，说明此次开格是正确的。那么就用 judge()去递归，去搜索，去蔓延，但是注意前提是“不搜索有雷（标记）的格子且不能越界”。

（4）如果 all!=num，证明匹配数和总数不同，说明此次开格是错误的（玩家想要投机取巧试探雷的位置或者纯属标记错误）。那么则进行游戏失败的处理。

6. 优化：

6.1 ※计时与暂停：此算法有效巧妙地避免了“多线程”，是亮点算法。

（1）计时功能：

```
while(true)
{
    if(onoff==KEEPON)/*之前未暂停*/
        clock(t1);/*获取当前时间*/
    else/*之前暂停过*/
    {
        t1=t1+(t4-t3);/*弥补暂停的时间*/
        clock(t1);
    }
    onoff=KEEPON;/*重置继续标志*/
void clock(time_t t1)/*时间*/
{
    setcolor(WHITE);
    setfillstyle(BLACK);
    t2=time(NULL);
    char str[2];
    bar(95,353,133,373);
    sprintf(str,"%d",t2-t1);
    outtextxy(95,353,str);
}
```

※算法详解：clock()函数是用来获取时间差并在左下角打印时间的函数，其传入的参数是在游戏开始时获取的系统时间（time_t 类型的变量）。在 clock()函数内部也有一个时间变量 t2 用于获取函数被调用时的系统时间（如果未暂停，每 while(true)循环一次就要获取一次时间）这样（t2-t1）得到的就是从游戏开始进行到目前的时间。

（2）暂停功能：

```
else if((m.uMsg==WM_LBUTTONDOWN)&&(m.x>40&&m.x<60)&&(m.y>350&&m.y<370))
    /*识别玩家"游戏中"点击"暂停"信息*/
        {
            printf("\a");
            onoff=STOP;/*暂停标记,以便形成函数分支区别暂停与继续不同处理情况*/
            t3=time(NULL);/*即时获取"暂停时"的系统时间*/
            setcolor(BLACK);setfillstyle(BLACK);bar(38,348,62,372);/*覆盖暂停图样*/
            keepon();/*继续图样*/
            while(true)/*在"暂停中"等待鼠标响应*/
            {
                m=GetMouseMsg();
                if((m.uMsg==WM_LBUTTONDOWN)&&(m.x>40&&m.x<60)&&(m.y>350&&m.y<370))
                    /*识别玩家"暂停中"点击"继续"信息*/
                {
                    printf("\a");
                    t4=time(NULL);/*即时获取"继续时"的系统时间*/
                    setcolor(BLACK);setfillstyle(BLACK);bar(38,348,62,372);/*覆盖继
                    stop();/*暂停图样*/
                    break;
                }
```

※算法详解：内部变量 onoff 用来标记玩家是否按过暂停（按过暂停 STOP，未按暂停 KEEPON）。clock()函数可以打印时间差，那么按下暂停后 clock()函数内的 t2 仍然继续获取时间，唯一能变的就是传入的实参“初始时间 t1”。显然当暂停以后，我们如果能把“暂停到继续”的时间计算出来，然后同游戏开始时获取的系统时间“做加法”，那么就弥补了损失的这段时间即 t1=t1+(t4-t3)。那么，t4 和 t3 应该怎么处理呢？显然我们可以想下，游戏原先是正常进行，当你按下暂停的时候就打破了这个规律，那么着手点显然就是按下暂停的那个 if()语句：首先要将 onoff 标记成 STOP 状态，其次获取此时的系统时间（暂停时）t3,然后进入 while(true)循环，实现暂停状态，而此循环的出口就是玩家点击“继续键”。一旦点击继续键，即时获取继续时的时间 t4，再马上 break，跳出这个暂停状态。（其中 t1,t2,t3,t4 均设置成了全局变量，就是想到他们将在不同函数中使用）

6.2 背景音效：

（1）对于一般按键响铃 printf(“\a”)解决。而游戏开始、游戏成功、游戏失败的音效则利用“restorator2007”将 C:\\Program Files\\Microsoft Games\\Minesweeper\\MineSweeper.dll 的音效文件导出。并用“千千静听”转换成.wav 格式的音频文件。

（2）包含多媒体接口：`#pragma comment(lib,"winmm.lib")`

（3）调用 PlaySound()函数进行.wav 格式文件的播放：

```
PlaySound("\\扫雷\\音效\\Start.wav",NULL,SND_FILENAME|SND_ASYNC);
```

参数一：指定.wav 文件的存放路径

参数二：停止所有与调用任务有关的声音

参数三：SND_FILENAME：指定以 WAV 文件名来调用

SND_ASYNC：用异步方式播放声音，PlaySound 函数在开始播放后立即返回（相当于多线程）

6.3 最快记录：

```
/*最高记录判断*/
FILE *fp1,*fp2;
if((fp1=fopen("record.txt","r"))==NULL)/*没有文件存在*/
{
    fp2=fopen("record.txt","w");
    InputBox(str,10,"恭喜您！！\n创新记录啦！！\n留下大名吧！！","新纪录");
    fprintf(fp2,"%s %d",str,t2-t1);/*将最高记录写入文件*/
    fclose(fp2);
}
else
{
    fscanf(fp1,"%s %d",str,&old);/*读入旧记录*/
    if(t2-t1<old)/*新记录*/
    {
        InputBox(str,10,"恭喜您！！\n创新记录啦！！\n留下大名吧！！","新纪录");
        fp2=fopen("record.txt","w");
        fprintf(fp2,"%s %d",str,t2-t1);/*将最高记录写入文件*/
        fclose(fp2);
    }
    else
    {
        HWND wnd=GetHWnd();
        MessageBox(wnd,"离最高记录就差一点点啦~\n再接再厉喲~~","提示",MB_OK|MB_I
    }
    fclose(fp1);
}
```

这段代码写在获胜的 if 语句下。定义了两个文件指针*fp1（用于读取最高纪录）,*fp2（用于覆盖写入新最高纪录）。首先，以读取方式打开文件。1.如果不存在“record.txt”文件则以写入方式打开并用 InputBox(数组地址，输入长度,“正文”,“标题栏”)弹出一个对话框接受玩家姓名的输入。再用 fprintf()语句写入刚才的姓名和所用时间。2.如果存在“record.txt”文件，则先要用 fscanf()语句读入旧时间记录,并和新的时间进行比较,如果新时间比旧时间快，则重复上述写入操作。反之弹出鼓励玩家的对话框。最后，关闭文件。