

课程代码：42D1232

学号：2120162016

南京财经大学

“基于.NET 的开发技术” 期末作业

中文题目：扫雷游戏设计与实现

英文题目：The Development of Minesweeper Game

所在院系：信息工程学院

专业班级：软件工程 1601 班

学生姓名：方泽强

指导老师：苏杭丽

完成时间：2019 年 01 月 04 日

目 录

目 录.....	2
扫雷游戏设计与实现.....	3
一、 课题内容、程序运行环境	1
二、 程序功能模块设计	1
三、 功能实现.....	6
四、 主要代码.....	8
五、 运行情况.....	22
总结.....	24
参考文献.....	24

扫雷游戏设计与实现

摘要: 扫雷游戏程序分为可视化窗体及其后台源代码。本次设计的扫雷游戏后台源代码实现了传统扫雷的基础功能，布雷，插旗，标记，显示地雷数，计时，判断游戏胜负，设置游戏难度，绘制方形雷区等；窗体主要实现了扫雷界面的可视化功能，游戏菜单，显示剩余地雷数，显示累计事件，表情反馈按钮，自定义地雷设置窗口等。本游戏采用 C# 开发语言，集成开发环境为 Microsoft Visual Studio 2017。程序界面简洁，游戏结果直观，用户体验良好。

关键词: C#;Microsoft Visual Studio;扫雷游戏;可视化窗体

The Development of Minesweeper Game

Abstract: The Minesweeper Game program is divided into a visual form and its background source code. The code of this game realizes the basic functions that traditional minesweeping holds, such as mine, flag, mark, displaying the number of mines, timing, judging the outcome of the game, setting the level of difficulty, drawing the minefield, & etc.; The form mainly realizes the visualization function of the minesweeping interface, the game menu, displays the remaining mines, displays the accumulated events, the expression feedback button, and the custom mine setting window, etc. This game was developed by C#, and the integrated development environment was Microsoft Visual Studio 2017. The program interface is simple and the game results are intuitive, which makes the user 's experience great.

Keywords: C#;Microsoft Visual Studio; Minesweeper Game; Visual Forms

一、 课题内容、程序运行环境

要求运用课程所学习的 C# .NET 知识，在 Visual Studio 2017 .NET 平台下开发扫雷游戏。要求实现扫雷的基本功能，界面美观，程序运行稳定，最后发布生成 setup 安装程序。

二、 程序功能模块设计

1) 界面设计:

1. 程序界面顶部设有菜单栏，有游戏 (Game) ，帮助(Help)两个父菜单，如下图所示：

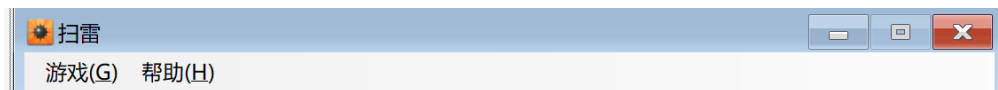


图 1 扫雷程序菜单栏

2. 点开父菜单“游戏 (Game) ”后有新建游戏，三种难度模式选择，自定义游戏设置，退出功能，点开“帮助”则有通知项目，如下图：

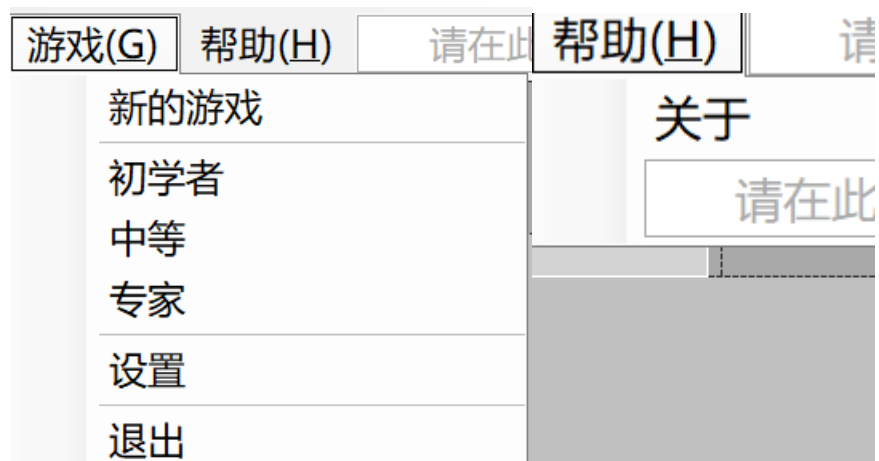


图 2 游戏菜单

3. 游戏菜单下面设有剩余地雷数量文本提示，经典扫雷的表情反馈按钮（点击可重玩游戏）和计时器可视化界面，如下图：



图 3 菜单下部可视化区域

4. 在程序运行时，我们希望程序能在主要游戏页面下绘出事先设定好的雷区图，最好有如下效果：

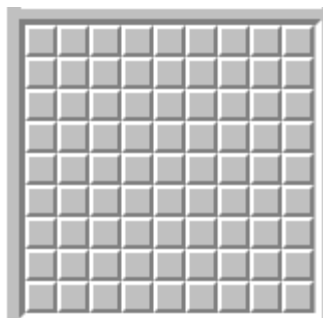


图 4 主要游戏区域

5. 点击界面顶部菜单游戏下的设置子项目，会弹出自定义雷区的界面，要求可以调节雷区长度，宽度，地雷数量，有确认和取消按键，且可以通过自己输入数字进行设置，设计效果如下：

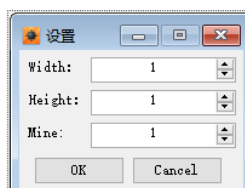


图 5 设置界面

6. 期望游戏主体界面，效果如下：

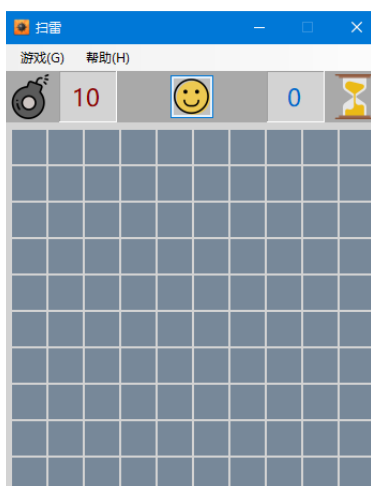


图 6 主界面

2) 游戏规则设计:

交互规则:

7. 鼠标在雷区上任意移动，鼠标所指位置的小方块颜色会高亮进行反馈并且鼠标在小方块上按下不松，顶部的笑脸会变成惊讶脸

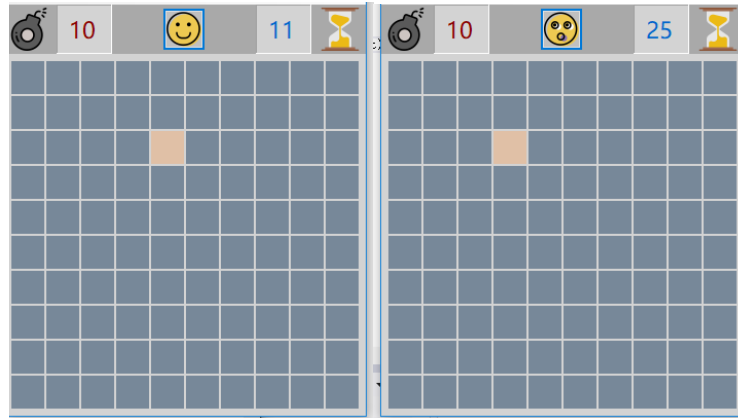


图 7 反馈示例 1

8. 右键鼠标会为方块插上小旗子，这里由于分辨率原因，选用坐标点代替，并且对一个未被点击方块右键两下，会有问号标记，插上小旗子，右上角剩余地雷数会相应少一个

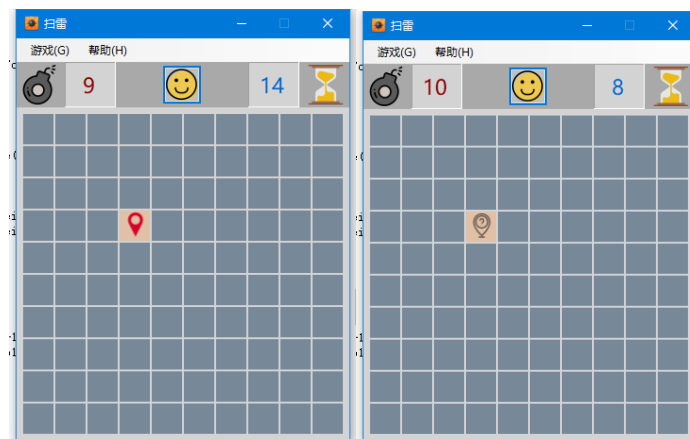


图 8 反馈示例 2

9. 点击一个方块时若不是地雷，但周围八个方块里有地雷，自身就显示周围方块的地雷数，如果周围没有雷，就展开周围方块，直到过程中检测到有雷的区域，停止展开，效果如下:

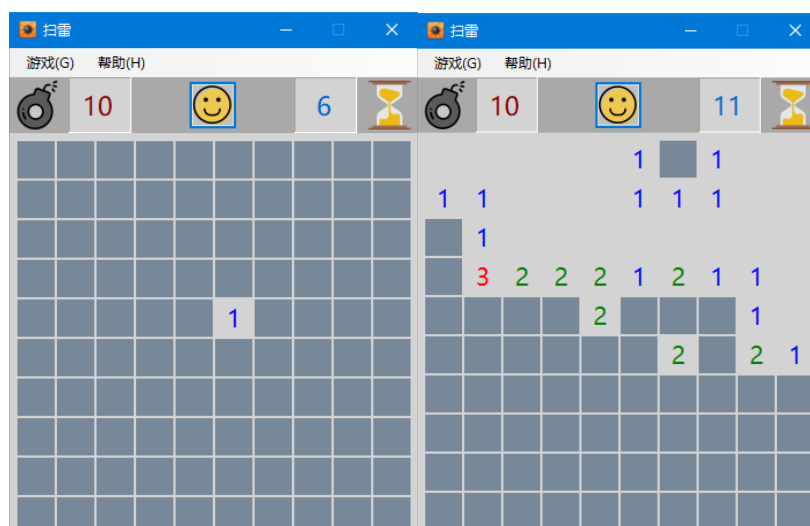


图 8 反馈示例 3

10. 点击一个方块是地雷时，展开所有含地雷的方块，笑脸变成悲伤脸，时间停止计数，效果如下：



图 8 反馈示例 4




输赢规则：

11. 游戏失败的判定条件为，鼠标点击一个未曾访问的方块时触发地雷，且剩余地雷数大于等于 0，具体设计思路为：
 - a) 当鼠标左键下落的事件发生时，系统检测对应方块的是否有雷以及是否被访问；
 - b) 当鼠标左键的抬起事件发生时，系统判断剩余雷数是否等于 0；
 - c) 若该方块有雷，未被访问，且目前剩余雷数大于 0，则弹出创建窗口表示游戏失败；
12. 游戏胜利的判定条件为，雷区剩余地雷数为 1，未有地雷被触发，且有且仅剩一个未被触发的方块或被问号标记的方块，具体设计思路为：

- a) 系统检测当前雷区剩余地雷数是否等于 1, 且仅剩一个未被访问的无插旗方块;
- b) 系统检测目前地雷方块的是否被插旗;
- c) 若目前界面所有插旗的方块都为雷区, 剩余地雷数等于 1, 且仅剩一个未被访问的无插旗方块, 或者仅剩一个被问号标记的方块, 则弹出创建窗口表示游戏胜利, 并显示游戏总时长;

3) 功能设计:

13. 雷区

- 每当一面旗子  在方块上放置后, 左上角的剩余雷数减 1
- 每当鼠标右键点击旗子 , 旗子会变成问号 , 此时左上角剩余雷数加 1
- 默认雷区长度, 宽度, 地雷数量的初始值为 10, 10, 10
- 点击一个不是地雷的方块时, 若其周围有地雷就会显示周围八个方块的地雷数, 每个数字都有不同的颜色
- 鼠标左键点击一个方块若不是地雷, 且周围八个方块没有雷, 就持续展开周围方块, 直到过程中检测到所展开的方块周围有雷, 停止展开, 周围有雷的方块显示周围地雷数, 周围没有地雷的方块不显示数字

14. 菜单

- 点击顶部菜单“游戏”中的“新的游戏”, 或单击顶部中间笑脸, 都可以清空游戏区, 重新布雷, 重新计时, 开始新的游戏
- 用户也可以在菜单中选择从 初学者到专家三个梯度的游戏难度, 其雷区属性分别对应了 (10, 10, 10) (16, 16, 40) (30, 16, 99), 方便玩家快速设定难度
- 设置界面中, 可以调节雷区长度, 宽度, 地雷数量, 调节最小单位为 1, 也可自己输入数字, 三个属性的数值区间为 (1, 30) (1, 16) (1, 99), 按 ok 键可保存, 按 cancel 键可取消设置

4) 细节设计:

- 为保障游戏的可玩性和公平性, 鼠标第一个按下的方块不能是地雷
- 计时器将在第一个方块被点击后开始计时

- 失败或胜利后，游戏界面的方块应该都失效，例如鼠标点击任意未被访问的方块都不能再展开，方块不能响应用户的点击行为
- 失败后，胜利后计时器都停止
- 失败后“笑脸”变成“悲伤脸”，胜利后“笑脸”变成“崇拜脸”
- 自定义雷区界面初始数值应与系统目前雷区各项属性一致
- 运行后界面能根据雷区面积自动调节大小，且不能被改变

三、功能实现

控件

1. 准备主界面需要用到的素材，png 格式的炸弹，旗帜，问号标识，秒表和四张不同表情的脸；ico 格式的扫雷游戏图标，并在开始前将这些图片导入 Resources 中，Properties 中 Resources.resx 的预览图如下：

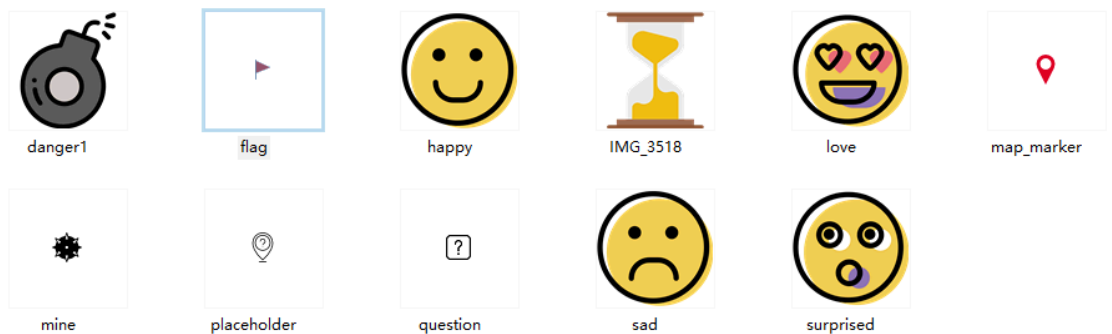


图 9 图片素材

2. 设计菜单，顶部容器，剩余地雷数，计时器图片，显示数字所用到的控件如下：

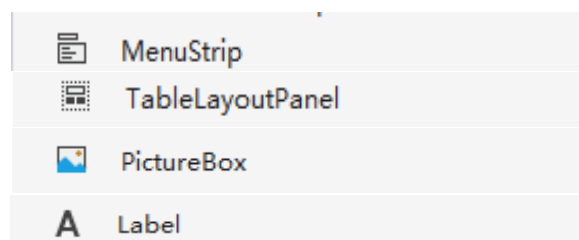


图 10 界面控件 1

设计 TableLayoutPanel 上的排布，使界面整洁，并且其中 PictureBox 的属性设置为，注意 Dock 设置成 Fill。

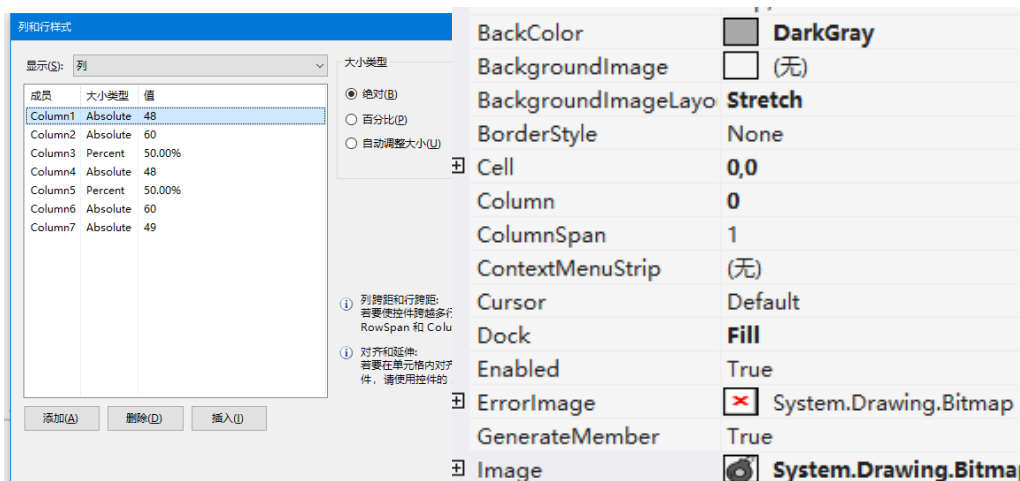


图 11 属性设置 1

3. 中间笑脸按钮，计时器功能则用到以下控件：



图 12 界面控件 2

二者属性分别设置如下，为了美观其高度都为 42，图片都设为正方形，注意 timer 的 interval 设置成 1000，其最小单位为 1 毫秒，我们通过设置让其最小单位变成秒，而 label 的字体设置成微软雅黑，16 号。

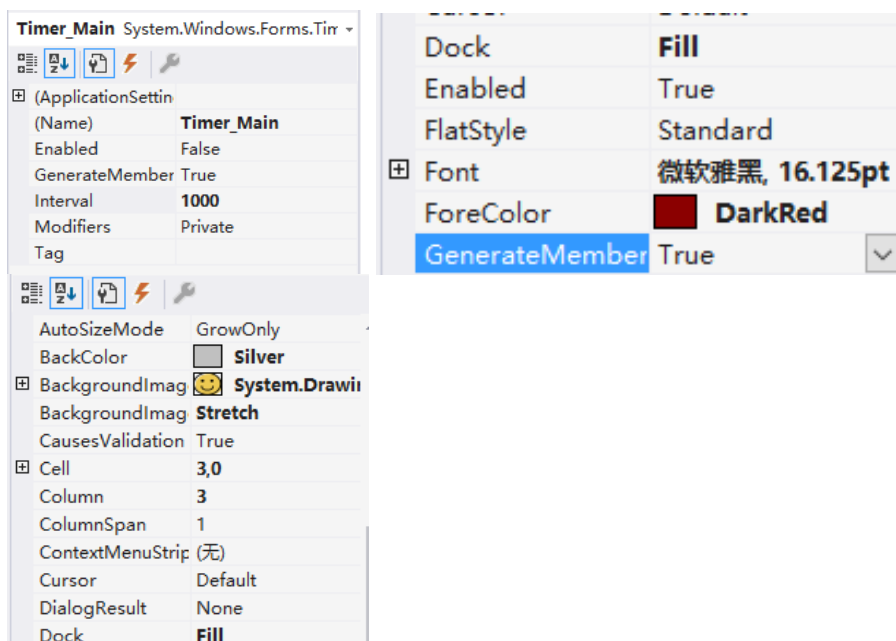


图 13 属性设置 2

4. 设置窗体中调节数值 和 ok/cancel 按钮 用你到了以下控件:

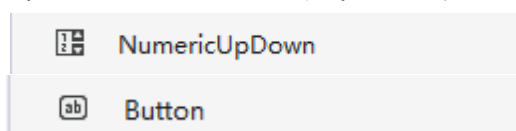


图 14 界面控件 3

NumericUpDown 注意设置最大值和最小值。

Maximum	30
MaximumSize	0, 0
Minimum	1
Maximum	16
MaximumSize	0, 0
Minimum	1
Maximum	99
MaximumSize	0, 0
Minimum	1

图 15 属性设置 3

四、 主要代码

1) 主窗体

1. 主要窗体 Form_Main 运行总体代码设计如下，具体代码将会在下面详细展开。

```
public Form_Main()
{
    InitializeComponent();

    nWidth = Properties.Settings.Default.Width;
    nHeight = Properties.Settings.Default.Height;
    nMineCnt = Properties.Settings.Default.MineCnt;

    bMark = Properties.Settings.Default.Mark;

    UpdataSize();
    SelectLevel();
}
```

2) 游戏难度设置

2. 从实现设置游戏难度（设置雷区宽，高，地雷数三个属性）开始，代码如下：

```
private void SetGame(int Width, int Height, int MineCnt)
{
    nWidth = Width;
    nHeight = Height;
    nMineCnt = MineCnt;
    UpdateSize();
}
```

3. 分别实现从初学者，中等，专家三个难度对雷区的设置功能，代码如下：

```
private void SetGameBeginner()
{
    SetGame(10, 10, 10);
}
private void SetGameMedium()
{
    SetGame(16, 16, 40);
}
private void SetGameExpert()
{
    SetGame(30, 16, 99);
}
```

4. 回到整个类中的开头，事先定义好刚刚写出函数方法里所包含的属性。

```
public int nWidth;
public int nHeight;
public int nMineCnt;
bool bMark;
```

3) 雷区绘图

5. 紧接着我们开始着手绘制雷区，绘制雷区需要用到主窗体的 Paint 事件，鼠标单击进入事件代码区，为了更好的书写代码，在此先写下绘制雷区方法名。

```
private void Form_Main_Paint(object sender, PaintEventArgs e)
{
    PaintGame(e.Graphics);
}
```

```
}

```

6. 在其下部写出核心函数 `PaintGame`，先写出其基本属性，清零用亮灰色，以及 `x`，`y` 轴的偏移量，这是为了让绘制的雷区在主界面稍下方显示不被 `TableLayoutPanel` 遮挡。

```
private void PaintGame(Graphics g)
{
    g.Clear(Color.LightGray);

    int nOffsetX = 6;
    int nOffsetY = 54 + menuStrip1.Height;

```

7. 具体雷区绘制代码结构使用两个 `for` 循环来遍历每行每列，其中穿插嵌套 `if` 判断语句，记得在类的开头定义点属性 `PointF`。

```
Point MouseFocus;
```

8. 先总的判断是否被点开（即判断每个点的 `pState` 是否为 1）如果是，再判断鼠标指向的方块，如果是，则生成颜色较亮的方块，实现方块响应鼠标的高亮效果；如果该点状态是被标注了旗子（即 `pState=2`）那么调用 `flag` 图像，在普通生成的方块上再生成一面小旗子；如果该点状态是问号（即 `pState=3`）那么调用 `doubt` 图像，在普通生成的方块上再生成一个问号。

```
for(int i=1;i<=nWidth;i++)
{
    for (int j = 1; j <= nHeight; j++)
    {
        if (pState[i,j] != 1)
        {
            if (i == MouseFocus.X && j == MouseFocus.Y)
            {
                g.FillRectangle(new SolidBrush(
                    Color.FromArgb(100, Color.SandyBrown)
                ),
                    new Rectangle(nOffsetX + 34 * (i - 1) + 1,
                        nOffsetY + 34 * (j - 1) + 1,
                        32,

```

```

        32));
    }
    else
    {
        g.FillRectangle(Brushes.LightSlateGray,
            new Rectangle(nOffsetX + 34 * (i - 1) + 1,
                nOffsetY + 34 * (j - 1) + 1,
                32,
                32));
    }
    if(pState[i,j]==2)
    {
        g.DrawImage(Properties.Resources.map_marker,
            nOffsetX + 34 * (i - 1) + 1 + 4,
            nOffsetY + 34 * (j - 1) + 1 + 2);
    }
    if(pState[i,j]==3)
    {
        g.DrawImage(Properties.Resources.placeholder,
            nOffsetX + 34 * (i - 1) + 1 + 4,
            nOffsetY + 34 * (j - 1) + 1 + 2);
    }
}
}

```

9. 我们还需判断该方块被点开的情况，若已被点开，每个不是地雷的方块都需显示自身周围的地雷数，每中地雷数的显示颜色都不一样，地雷数为0则不显示，有地雷则调用生成地雷的图片。

```

else if(pState[i,j]==1)
{
    if(MouseFocus.X==i&&MouseFocus.Y==j)
    {
        g.FillRectangle(new SolidBrush(
            Color.FromArgb(100, Color.LightGray)),
            new Rectangle(nOffsetX + 34 * (i - 1) + 1,
                nOffsetY + 34 * (j - 1) + 1,
                32,
                32));
    }
    else
    {
        g.FillRectangle(Brushes.LightGray,
            new Rectangle(nOffsetX + 34 * (i - 1) + 1,

```

```

        nOffsetY + 34 * (j - 1) + 1,
        32,
        32));
    }
    if(pMine[i,j]>0)
    {
        Brush DrawBrush = new SolidBrush(Color.Blue);

        if (pMine[i, j] == 2) { DrawBrush = new SolidBrush(Color.Green); }
        if (pMine[i, j] == 3) { DrawBrush = new SolidBrush(Color.Red); }
        if (pMine[i, j] == 4)
        {
            DrawBrush = new SolidBrush(Color.DarkBlue);
        }
        if (pMine[i, j] == 5) { DrawBrush = new
SolidBrush(Color.DarkRed); }
        if (pMine[i, j] == 6)
        {
            DrawBrush = new SolidBrush(Color.DarkSeaGreen);
        }
        if (pMine[i, j] == 7) { DrawBrush = new SolidBrush(Color.Black); }
        if (pMine[i, j] == 8)
        {
            DrawBrush = new SolidBrush(Color.DarkGray);
        }

        SizeF Size = g.MeasureString(pMine[i, j].ToString(),
                                     new Font("微软雅黑", 16));
        g.DrawString(pMine[i, j].ToString(),
                     new Font("微软雅黑", 16),
                     DrawBrush,
                     nOffsetX + 34 * (i - 1) + 1 + (32 - Size.Width) / 2,
                     nOffsetY + 34 * (j - 1) + 1 + (32 - Size.Height) / 2);
    }
    if(pMine[i,j]==-1)
    {
        g.DrawImage(Properties.Resources.mine,
                    nOffsetX + 34 * (i - 1) + 1 + 4,
                    nOffsetY + 34 * (j - 1) + 1 + 2);
    }
}

```

```

    }
}
}

```

10. 在运行后发现界面大小和雷区大小不匹配，考虑到上述的细节设计提及要求窗体能自动调整大小，我们构造一个自动调节大小方法 `UpdataSize`，代码如下：

```

private void UpdataSize()
{
    int nOffsetX = this.Width - this.ClientSize.Width;
    int nOffsetY = this.Height - this.ClientSize.Height;
    int nAdditionY = menuStrip1.Height + tableLayoutPanel1.Height;
    this.Width = 12 + 34 * nWidth + nOffsetX;
    this.Height = 12 + 34 * nHeight + nAdditionY + nOffsetY;
    newGameToolStripMenuItem_Click(new object(), new EventArgs());
}

```

4) 菜单事件

11. 紧接着我们着手书写游戏菜单里的各种事件，从易到难，先从三个难度设置开始，而且在选定一个难度的同时，该难度左边应有打勾符号标记，代码如下：

```

private void toolStripMenuItem_medium_Click(object sender, EventArgs e)
{
    nWidth = 16;
    nHeight = 16;
    nMineCnt = 40;

    SelectLevel();
    UpdataSize();
}

```

```

private void toolStripMenuItem_expert_Click(object sender, EventArgs e)
{
    nWidth = 30;
    nHeight = 16;
    nMineCnt = 99;

    SelectLevel();
    UpdataSize();
}

```

```

private void beginnerToolStripMenuItem_Click(object sender, EventArgs e)

```



```
{
    nWidth = 10;
    nHeight = 10;
    nMineCnt = 10;

    SelectLevel();
    UpdateSize();
}
private void SelectLevel()
{
    if (nWidth == 10 && nHeight == 10 && nMineCnt == 10)
    {
        beginnerToolStripMenuItem.Checked = true;
        toolStripMenuItem_medium.Checked = false;
        toolStripMenuItem_expert.Checked = false;
        settingToolStripMenuItem.Checked = false;
    }
    else if (nWidth == 16 && nHeight == 16 && nMineCnt == 40)
    {
        beginnerToolStripMenuItem.Checked = false;
        toolStripMenuItem_medium.Checked = true;
        toolStripMenuItem_expert.Checked = false;
        settingToolStripMenuItem.Checked = false;
    }
    else if (nWidth == 30 && nHeight == 16 && nMineCnt == 99)
    {
        beginnerToolStripMenuItem.Checked = false;
        toolStripMenuItem_medium.Checked = false;
        toolStripMenuItem_expert.Checked = true;
        settingToolStripMenuItem.Checked = false;
    }
    else
    {
        beginnerToolStripMenuItem.Checked = false;
        toolStripMenuItem_medium.Checked = false;
        toolStripMenuItem_expert.Checked = false;
        settingToolStripMenuItem.Checked = true;
    }
}
```

12. 完成后我们编辑设置区，也就是自定义雷区的代码，右键菜单中的设置选项进入事件代码区，使该点击事件与 Form_Settings 窗体关联，使鼠标点击设置能跳出相应窗体：

```
private void settingToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form_Setting Setting = new Form_Setting(this);
    Setting.ShowDialog();
    UpdateSize();
}

```

13. 最后是最重要的单击“新的游戏”事件，单击该事件后，雷区应该清零，系统重新随机生成不同位置的的地雷，这里用变换每个方块的 pMine 属性实现，并且 Timer 也并清零等下次鼠标单击后重用，代码如下：

```
public void newGameToolStripMenuItem_Click(object sender, EventArgs e)
{
    Array.Clear(pMine, 0, pMine.Length);
    Array.Clear(pState, 0, pState.Length);

    MouseFocus.X = 0;
    MouseFocus.Y = 0;

    Random = new Random();
    for (int i = 1; i <= nMineCnt;)
    {
        int x = random.Next(nWidth) + 1;
        int y = random.Next(nHeight) + 1;

        if (pMine[x, y] != -1)
        {
            pMine[x, y] = -1; //生成地雷
            i++;
        }
    }
    for (int i = 1; i <= nWidth; i++)
    {
        for (int j = 1; j <= nHeight; j++)
        {
            if (pMine[i, j] != -1)
            {
                for (int k = 0; k < 8; k++)
                {
                    if (pMine[i + dx[k], j + dy[k]] == -1)
                    {
                        pMine[i, j]++;
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}
this.Refresh();
Label_Mine.Text = nMineCnt.ToString();
Label_Timer.Text = "0";
bGame = false;
}

```

5) 鼠标交互事件

14. 鼠标在雷区的移动，按下和抬起，系统都会有相应的反馈，代码如下：

```

private void Form_Main_MouseMove(object sender, MouseEventArgs e)//鼠标移动
{
    if (e.X < 6 ||
        e.X > 6 + nWidth * 34 ||
        e.Y < 54 + menuStrip1.Height ||
        e.Y > 54 + menuStrip1.Height + nHeight * 34)
    {
        MouseFocus.X = 0;
        MouseFocus.Y = 0;
    }
    else
    {
        int x = (e.X - 6) / 34 + 1;
        int y = (e.Y - menuStrip1.Height - 54) / 34 + 1;
        MouseFocus.X = x;
        MouseFocus.Y = y;
    }
    this.Refresh();
}
private void Form_Main_MouseDown(object sender, MouseEventArgs e)//鼠标按下
{
    if (pState[MouseFocus.X, MouseFocus.Y] == 0 || bLost == false)
    {
        Button_face.BackgroundImage = Properties.Resources.surprised;
        Timer_Main.Enabled = true;
    }
    if (e.Button == MouseButtons.Left)

```

```

    {
        bMouseLeft = true;
    }
    if(e.Button==MouseButtons.Right)
    {
        bMouseRight = true;
    }
}

```

15. 值得一提的是，鼠标抬起事件较复杂，需判断是右键抬起，左键抬起还是左右键一起抬起，所对应的反馈也不一样，先写出左右键一起抬起的事件，代码如下：

```

private void Form_Main_MouseUp(object sender, MouseEventArgs e)
{
    if(pState[MouseFocus.X,MouseFocus.Y]==0 || bLost==false)
        Button_face.BackgroundImage = Properties.Resources.happy;
    if ((MouseFocus.X == 0 && MouseFocus.Y == 0) || bGame)
    {
        return;
    }
    if (bMouseLeft && bMouseRight)
    {
        if (pState[MouseFocus.X, MouseFocus.Y] == 1 && pMine[MouseFocus.X,
MouseFocus.Y] > 0)
        {
            int nFlagCnt = 0,
                nDoubtCnt = 0,
                nSysCnt = pMine[MouseFocus.X, MouseFocus.Y];
            for (int i = 0; i < 8; i++)
            {
                int x = MouseFocus.X + dx[i];
                int y = MouseFocus.Y + dy[i];
                if (pState[x, y] == 2)
                {
                    nFlagCnt++;
                }
                if (pState[x, y] == 3)
                {
                    nDoubtCnt++;
                }
            }
            if (nFlagCnt == nSysCnt || nFlagCnt + nDoubtCnt == nSysCnt)
            {

```

```

        bool bFlag = OpenMine(MouseFocus.X, MouseFocus.Y);
        if (!bFlag)
        {
            GameOver();
        }
    }
}
}
}
}

```

左键抬起的相关程序:

```

else if (bMouseLeft)
{
    if (pMine[MouseFocus.X, MouseFocus.Y] != -1)
    {
        if (pState[MouseFocus.X, MouseFocus.Y] == 0)
        {
            dfs(MouseFocus.X, MouseFocus.Y);
        }
    }
    else
    {
        GameOver();
    }
}
}

```

右键抬起的相关程序:

```

else if (bMouseRight)
{
    if (bMark)
    {
        if (pState[MouseFocus.X, MouseFocus.Y] == 0)
        {
            if (Convert.ToInt32(Label_Mine.Text) > 0)
            {
                pState[MouseFocus.X, MouseFocus.Y] = 2;
                Label_Mine.Text = Convert.ToString(Convert.ToInt32(Label_Mine.Text)
- 1);
            }
        }
    }
    else if (pState[MouseFocus.X, MouseFocus.Y] == 2)
    {
        pState[MouseFocus.X, MouseFocus.Y] = 3;
        Label_Mine.Text = Convert.ToString(Convert.ToInt32(Label_Mine.Text) +

```

```

1);
        }
        else if (pState[MouseFocus.X, MouseFocus.Y] == 3)
        {
            pState[MouseFocus.X, MouseFocus.Y] = 0;
        }
    }
}
this.Refresh();
GameWin();
bMouseLeft = bMouseRight = false;
}

```

6) 方块展开算法

16. 游戏里有一个机制，如上述功能设计所说，鼠标左键点击一个方块若不是地雷，且周围八个方块没有雷，就持续展开周围方块，直到过程中检测到所展开的方块周围有雷，停止展开，周围有雷的方块显示周围地雷数，周围没有地雷的方块不显示数字，为实现这个功能，构造两个方法，一个是采用了深度优先搜索的访问周围方块 dfs 方法，一个是调用该访问方法的 ExpandMine 方法

```

private void dfs(int sx,int sy)
{
    pState[sx, sy] = 1;
    for(int i=0;i<8;i++)
    {
        int x = sx + px[i];
        int y = sy + py[i];
        if (x >= 1 && x <= nWidth && y >= 1 && y <= nHeight &&
            pMine[x, y] != -1 && pMine[sx, sy] == 0 &&
            (pState[x, y] == 0 || pState[x, y] == 3))
        {
            dfs(x, y);
        }
    }
}
private bool ExpandMine(int sx,int sy)
{
    bool bFlag = true;

```

```

for (int i = 0; i < 8; i++)
{
    int x = MouseFocus.X + dx[i];
    int y = MouseFocus.Y + dy[i];
    if(pState[x,y]==0)
    {
        pState[x, y] = 1;
        if (pMine[x, y] != -1)
        {
            dfs(x, y);
        }
        else
        {
            bFlag = false;
            break;
        }
    }
}
return bFlag;
}

```

7) 游戏胜负方法

17. 当前游戏被判断为失败时，应有一个方法 `GameOver` 对游戏后续进行处理，如时间停止，所有方块失效，笑脸变成哭脸，该方块变成已访问状态，代码如下：

```

private void GameOver()
{
    Button_face.BackgroundImage = Properties.Resources.sad;
    for(int i=1;i<=nWidth;i++)
    {
        for(int j=1;j<=nHeight;j++)
        {
            if(pMine[i,j]==-1&&(pState[i,j]==0||pState[i,j]==3))
            {
                pState[i, j] = 1;
            }
        }
    }
    bGame = true;
    bLost = true; }

```

18. 当前游戏被判断为胜利时 GameWin，应有一个方法对游戏后续进行处理，如时间停止，所有方块失效，笑脸变成崇拜脸，并弹出窗口提示游戏胜利与游戏时间，代码如下：

```
private void GameWin()
{
    int nCnt = 0;
    for(int i=1;i<=nWidth;i++)
    {
        for(int j=1;j<=nHeight;j++)
        {
            if(pState[i,j]==0||pState[i,j]==2||pState[i,j]==3)
            {
                nCnt++;
            }
        }
    }
    if (nCnt == nMineCnt)
    {
        Button_face.BackgroundImage = Properties.Resources.love;
        Timer_Main.Enabled = false;
        MessageBox.Show(string.Format("游戏胜利, 耗时: {0} 秒 ", Label_Timer.Text),
            "提示", MessageBoxButtons.OK);
        bGame = true;
    }
}
```

8) 完善功能

19. 添加笑脸按钮😊的点击事件，使鼠标单击此按钮可以开始新的游戏，代码如下：

```
private void Button_face_Click(object sender, EventArgs e)
{
    newGameToolStripMenuItem_Click(new object(), new EventArgs());
}
```

20. 计时器将在第一个方块被点击后开始计时

```
if (pState[MouseFocus.X, MouseFocus.Y] == 0 || bLost == false)
{
    Button_face.BackgroundImage = Properties.Resources.surprised;
    Timer_Main.Enabled = true;
}
```


21. 为保障游戏的可玩性和公平性，鼠标第一个按下的方块不能是地雷，具体实现如下，首先在窗体鼠标抬起事件的判断左键抬起的代码区中添加一处判断，即判断按下的方块是否为地雷，且是否为第一次，是的话重新开始新的游戏，不是的话 bFirst 变为 false。

```
else if (bMouseLeft)//左键被按下去
{
    if (pMine[MouseFocus.X, MouseFocus.Y] == -1 && bFirst)
    {
        newGameToolStripMenuItem_Click(new object(), new EventArgs());
    }
    bFirst = false;
```

需要注意，每次新游戏开始时 bFirst 都得为 true，执行鼠标抬起事件时才能被判定为第一次按。

```
public void newGameToolStripMenuItem_Click(object sender, EventArgs e)
{
    bFirst = true;
```

五、 运行情况

发布生成 setup 后，安装文件后开始游戏；
进入主界面，光标在雷区上移动，方块高亮。

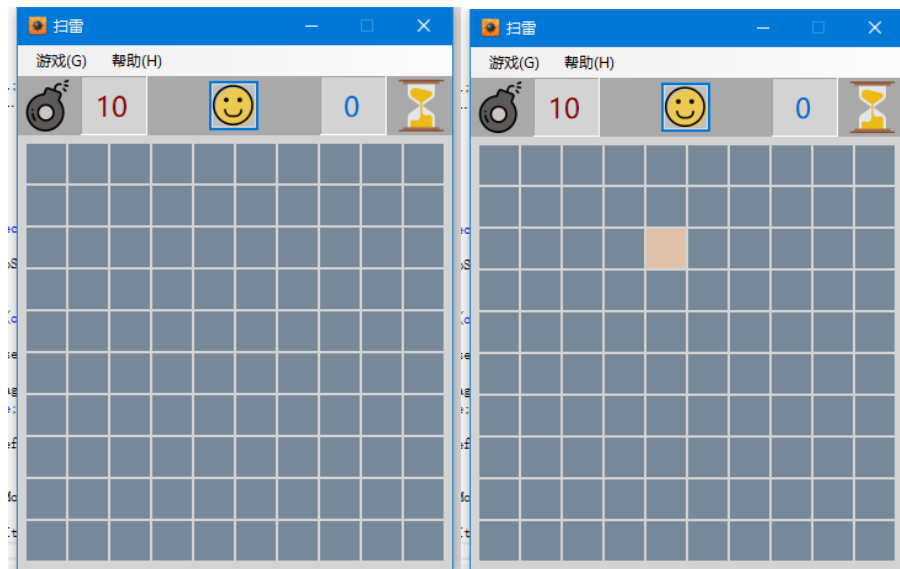


图 16 运行情况 1

单击第一个方块，后根据扫雷规则进行游戏，期间不小心踩到地雷。

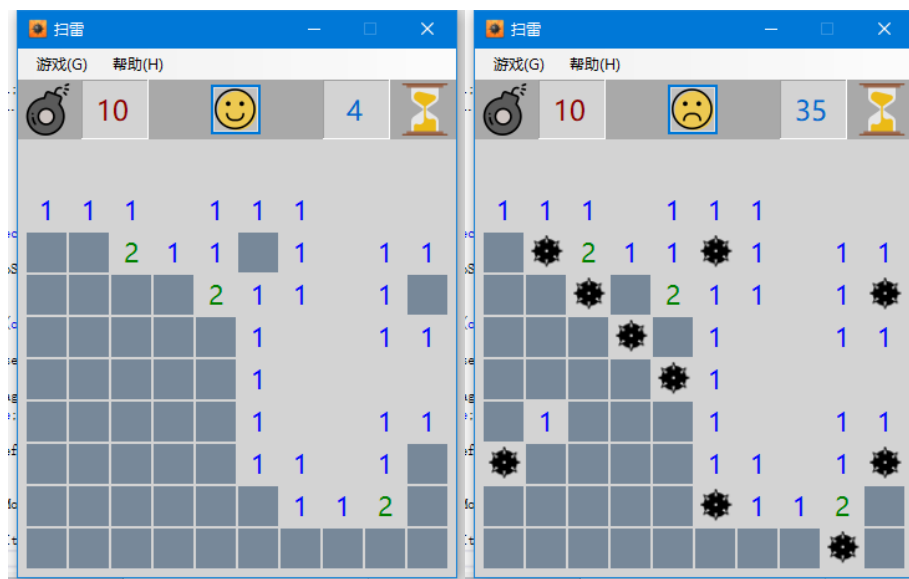


图 17 运行情况 2

点击表情按钮，重新开始游戏，游戏过程中进行插旗，标注。

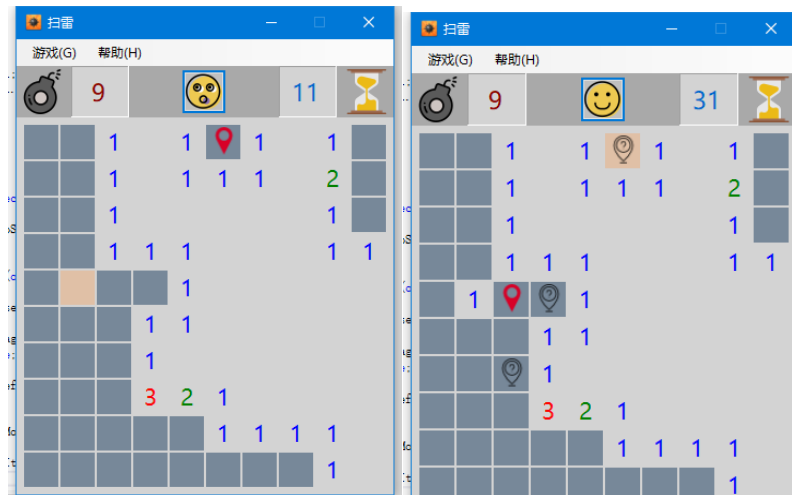


图 18 运行情况 3

最后依照扫雷规则，最终完成胜利。

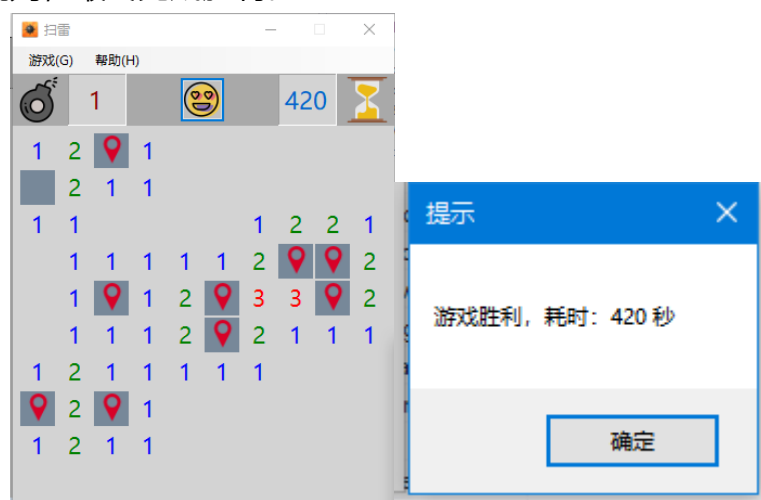


图 19 运行情况 4

在自定义雷区中设置 (10, 10, 99) 来测试第一步不可能会踩到雷是否实现成功, 发现运行成功, 第一个是雷都会重新开始游戏, 直到第一次点击不是地雷为止。

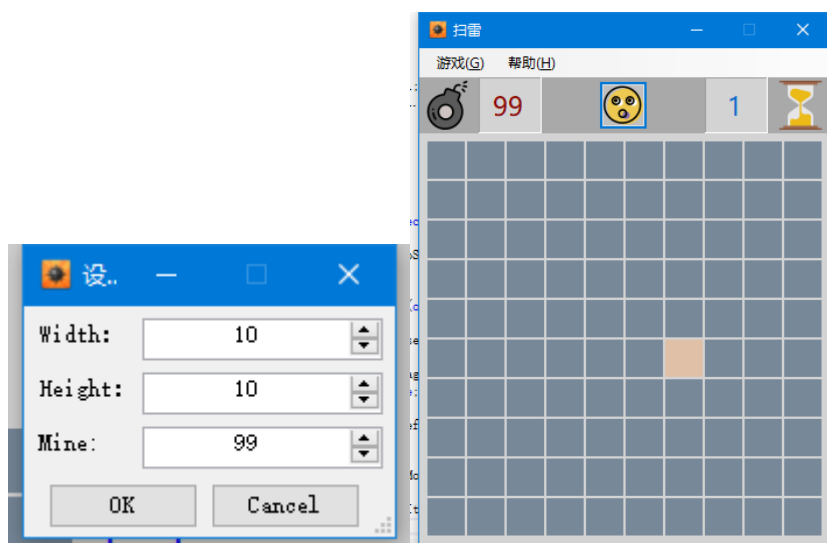


图 20 运行情况 5

总结

整体扫雷程序实现起来较为容易, 扫雷有许多功能, 所以内容略显庞杂。笔者认为游戏设计难点在于雷区的绘制, 和访问周围方块的深度搜索算法, 前者考察对于控件属性名的熟悉程度, 而后者考察的就是个人的计算思维能力。关于程序还有其不完美之处, 如界面主题颜色可以更加美观; 游戏失败显示地雷也可以添加爆炸特效; 对于最高难度有 100 颗地雷, 游戏时间长, 开发存档功能, 可以拓展游戏的可玩性, 便利用户。

参考文献

- [1]夏敏捷,罗菁主编.Visual C# NET 基础与应用教程 [J].北京: 清华大学出版社, 2014.
- [2] lvy-End .扫雷游戏制作过程 (C#描述) .2015.