第 2 章

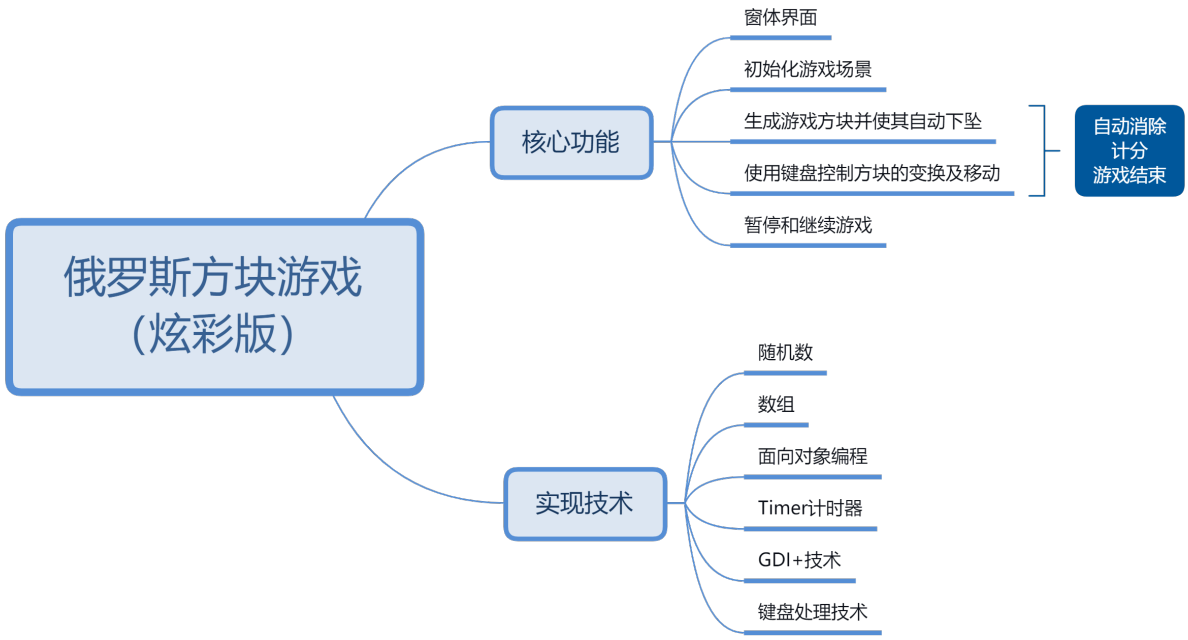
俄罗斯方块游戏（炫彩版）

——随机数 + 数组 + 面向对象编程 + Timer 计时器 + GDI+技术 + 键盘处理

俄罗斯方块游戏是一款广受大众喜欢的经典益智类游戏，其游戏规则非常简单。在屏幕上堆积各种形状的方块，满行即消除该行，并得到相应的分数，而当方块堆积到屏幕最上方时，游戏结束。俄罗斯方块游戏不仅可以考验玩家的即时反应能力，还可以锻炼玩家的观察力和专注力。本章将讲解如何使用 C#开发一个炫彩版的俄罗斯方块游戏。本项目对经典的俄罗斯方块游戏做了一定的优化，增强了游戏的视觉吸引力，有助于玩家更快速地识别和匹配方块，提升了游戏的可玩性。

项目微视频

本项目的核心功能及实现技术如下：



2.1 开 发 背 景

俄罗斯方块是一款经典的游戏，自从其诞生之日起，就以简单易懂、受众极广的特性迅速风靡全球。基于 C#强大的面向对象编程能力、丰富的图形用户界面库以及 GDI+绘图技术的支持，我们能够快速、高效地进行俄罗斯方块游戏的开发。本章就来开发一款炫彩版的俄罗斯方块游戏。

本项目的实现目标如下：

☑ 用户界面简单，使玩家能够轻松愉快地享受游戏。

☑ 实现俄罗斯方块游戏的主要逻辑，包括方块的形状、旋转、下落和消除等。
☑ 实现一个随机生成不同形状方块的系统，为游戏增加更多的不确定性和挑战性。
☑ 建立一个计分系统，根据玩家消除的方块数量计算分数，并在游戏界面上实时显示。

2.2 系 统 设 计

2.2.1 开发环境

本项目的开发及运行环境如下：
☑ 操作系统：推荐 Windows 10、11 及以上。
☑ 开发工具：Visual Studio 2022。
☑ 开发语言：C#。

2.2.2 业务流程

俄罗斯方块游戏（炫彩版）启动时，首先进入游戏主窗体中，在该窗体中单击“开始”按钮后，可以自动生成方块并下坠。在方块下坠的过程中，用户可以使用键盘来控制方块的变换及移动。同时，程序会自动判断是否有可消除的行，以及方块是否已经在屏幕的顶格处且无法移动，并根据判断结果执行相应的业务处理。另外，在游戏过程中，用户可以手动控制游戏的暂停与继续。

本项目的业务流程如图 2.1 所示。

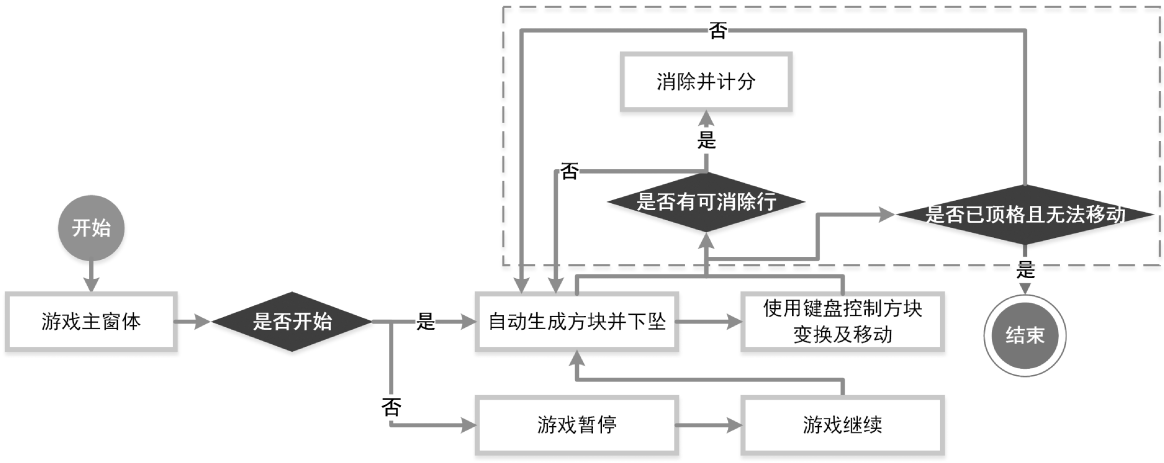


图 2.1 俄罗斯方块游戏（炫彩版）业务流程

说明

图 2.1 中使用虚线标注的逻辑部分会在移动方格时自动进行处理。

2.2.3 功能结构

本项目的功能结构已经在章首页中给出。作为一个经典的俄罗斯方块游戏项目，本项目实现的具体功能如下：

- ☑ 游戏界面设计：使用 C#中的 WinForm 技术创建游戏的窗体界面，界面简洁明了，易于操作。
- ☑ 游戏功能设计：
  - ➢ 游戏初始化：初始化游戏的场景，并绘制初始的俄罗斯方格。
  - ➢ 方块的生成：随机生成不同形状的方块，并将待操作方块显示在游戏界面上。
  - ➢ 游戏逻辑实现：包括方块的自动下落、通过键盘控制方块旋转、满行自动消除、判断方块是否已经顶格且不能移动等功能。
  - ➢ 计分系统：根据玩家消除的方块数量计算分数，并在游戏界面上实时显示。
  - ➢ 控制游戏暂停与继续：在游戏过程中，玩家可以手动控制游戏的暂停与继续。

2.3 技 术 准 备

2.3.1 技术概览

- ☑ 随机数：C#中生成随机数使用 Random 类实现，该类是一个生成随机数的类，其中的 Next()方法用来随机生成指定范围内的数字。例如，本项目中根据随机生成的数字来确定方块的样式，关键代码如下：

```
Random rand = new Random();                                    //实例化 Random
CakeNO = rand.Next(1, 8);                                      //获取随机数
MyRussia.CakeMode(CakeNO);                                     //设置方块的样式
```

- ☑ 数组：数组是最为常见的一种数据结构，存储的是一组相同类型的数据序列或对象序列。根据维数的不同，可将数组分为一维数组、二维数组等。本项目中使用 Point 类型的一维数组来存储方块的位置，关键代码如下：

```
Point[] ArryPoi = new Point[4];                                //方块的数组
ArryPoi[1] = new Point(firstPoi.X, firstPoi.Y - Cake);         //设置第二块方块的位置
ArryPoi[2] = new Point(firstPoi.X, firstPoi.Y + Cake);         //设置第三块方块的位置
ArryPoi[3] = new Point(firstPoi.X + Cake, firstPoi.Y + Cake);  //设置第四块方块的位置
```

- ☑ 面向对象编程：面向对象编程的核心思想是要以对象来思考问题，需要将现实世界的实体抽象为对象，然后考虑这个对象具备的属性和行为。在 C#中，对象的属性是以成员变量的形式定义的，而对象的行为是以方法的形式定义的。例如，本项目中定义了一个 Russia 类，该类中定义了俄罗斯方块游戏中的公共成员变量及方法，示例代码如下：

```
class Russia
{
    public Point firstPoi = new Point(140, 20);                //定义方块的起始位置
    public static Color[,] PlaceColor;                         //记录方块的颜色
    public static bool[,] Place;                               //记录方块的位置
    public void CakeMode(int n)
    {                                                          //省略实现代码
    }
    //省略部分代码
}
```

- ☑ Timer 计时器：Timer 计时器在 WinForm 应用中以 Timer 控件来体现，它可以定期引发事件，其时间间隔由 Interval 属性定义，通过它的 Start()方法和 Stop()方法可以启动和停止计时器。例如，本项目中实现游戏的暂停及继续功能时，通过 Timer 计时器进行控制，关键代码如下：

```
if (timer1.Enabled == true)
{
```

```
        timer1.Stop();                                          //暂停
        button2.Text = "继续";
        ispause = false;
        textBox1.Focus();                                       //获取焦点
    }
    else
    {
        timer1.Start();                                         //继续
        button2.Text = "暂停";
        ispause = true;
    }
```

☑ GDI+技术：GDI+是微软在 Windows 平台上提供的一种对图形图像进行操作的应用程序编程接口（API），它是.NET 框架的一部分，主要用来进行二维图形的绘制、图像处理等。Graphics 类是 GDI+的核心，是进行 GDI+操作的基础类。Graphics 对象表示 GDI+绘图表面，它提供了将对象绘制到显示设备的方法，包括绘制直线、曲线、矩形、圆形、多边形、图像和文本等的方法。本项目中使用 GDI+技术来完成方块的绘制，关键代码如下：

```
g.FillRectangle(SolidB, rect);                                  //绘制一个矩形块
```

有关 C#中的随机数、数组、面向对象编程、Timer 计时器、GDI+技术等知识在《C#从入门到精通（第 7 版）》中有详细的讲解，对这些知识不太熟悉的读者可以参考该书对应的内容。下面将对俄罗斯方块游戏中的方块组变换以及键盘控制技术进行必要介绍，以确保读者可以顺利完成本项目。

2.3.2 方块组变换分析

要开发俄罗斯方块游戏（炫彩版），首先要明确该游戏的开发思路，具体如下：

☑ 明确俄罗斯方块游戏的规则。例如，方块在移动时，不能超出边界；方块与方块要罗列在一起；当某行方块填满时，要去除该行，并使该行以上的行下移；当屏幕中的方块已顶格并且不能消除时，游戏结束。

☑ 计算各方块组内每个小方块的显示位置，如“L”“T”“田”等方块组。

☑ 计算各方块组一共有几种变换样式。

☑ 由于俄罗斯方块是用一个个方块组合而成的，所以要根据背景的行数和列数定义多维数组。程序主要根据方块的行数和列数记录其是否存在，以及当前方块的颜色。

☑ 用 Timer 组件实时控制方块组的下移。

☑ 当方块组下移或变换样式时，判断其是否超出边界，是否与已经排列好的方块重叠。如果超出边界，或与方块重叠，停止下移或变换样式。

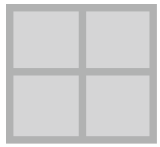
☑ 当方块下移完成后，根据方块所在的最大行和最小行，判断其是否有填满的行。如果有，则去除该行，并将该行以上的各行下移。

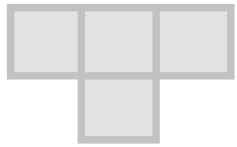
☑ 在去除指定的行后，重新生成一个新的随机方块组。

从上面的开发思路可以看出，俄罗斯方块游戏的核心是方块的组合、变换及移动，接下来对常见的几种方块组合样式进行介绍。俄罗斯方块游戏（炫彩版）中常见的几种方块组合样式如图 2.2 所示。

接下来分析如何制作方块组，以及方块组如何进行变换。

在俄罗斯方块游戏（炫彩版）中，所有的方块组都是用 4 个子方块组成的。在计算各方块组时，首先要明确方块组中哪一个子方块是起始方块，然后通过起始方块的位置计算其他子方块。下面以“L”方块组的组合及变换过程进行说明。图 2.3 表示“L”方块组的起始样式，图 2.4～图 2.6 为“L”方块组以起始方块为中心的变换过程。


“田”方块组


“T”方块组


“一”方块组

“Z”方块组

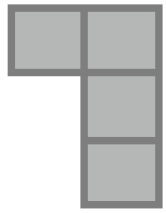

“L”方块组

图 2.2　俄罗斯方块游戏（炫彩版）的方块样式

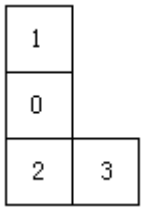


图 2.3　“L”方块组的起始样式

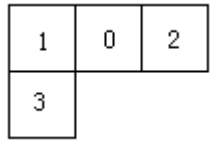


图 2.4　“L”方块组的变换 1

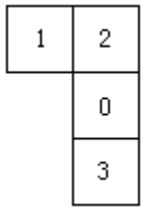


图 2.5　“L”方块组的变换 2

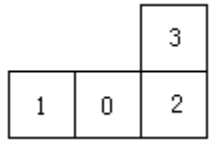


图 2.6　“L”方块组的变换 3

2.3.3　键盘处理技术

在俄罗斯方块游戏（炫彩版）中，可以通过键盘控制方块的形状变换及移动，这主要用到 C#窗体应用中的键盘处理事件 KeyDown 和 KeyUp。其中，KeyDown 事件在键盘按下时触发，而 KeyUp 事件在键盘抬起时触发，它们的语法格式如下：

```
private void Form1_KeyDown(object sender, KeyEventArgs e){}
private void Form1_KeyUp(object sender, KeyEventArgs e){}
```

参数说明如下：

- ☑ sender：触发事件的对象的引用，它通常是窗体或控件的引用。
- ☑ e：KeyEventArgs 类型，是包含事件数据的参数，它包含有关按键事件的信息，如哪个键被按下或释放。KeyEventArgs 对象提供了一个 KeyCode 属性，可以获取 KeyDown 或 KeyUp 事件的键盘代码，它的值是一个 Keys 枚举值，如表 2.1 所示。

表 2.1　Keys 枚举值及说明

枚 举 值	说　明	枚 举 值	说　明	枚 举 值	说　明
A	A 键	P	P 键	D0	0 键
B	B 键	Q	Q 键	D1	1 键
C	C 键	R	R 键	D2	2 键
D	D 键	S	S 键	D3	3 键
E	E 键	T	T 键	D4	4 键
F	F 键	U	U 键	D5	5 键
G	G 键	V	V 键	D6	6 键
H	H 键	W	W 键	D7	7 键
I	I 键	X	X 键	D8	8 键
J	J 键	Y	Y 键	D9	9 键
K	K 键	Z	Z 键	Up	向上箭头键
L	L 键	Alt	Alt 修改键	Down	向下箭头键
M	M 键	Shift	Shift 修改键	Left	向左箭头键
N	N 键	Tab	Tab 键	Right	向右箭头键
O	O 键	Enter	Enter 键	……	

例如，判断是否按下了“向下箭头”键，并执行方块下移操作，关键代码如下：

```
private void Frm_Main_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Down)                          //如果当前按下的是↓键
    {
        timer1.Interval = MyRussia.UpCareer - 50;        //增加下移的速度
        MyRussia.ConvertorMove(g,0);                     //方块下移
    }
    //省略部分代码
}
```

2.4　公共类设计

开发 C#项目时，通过合理设计公共类可以减少重复代码的编写，有利于代码的重用及维护。俄罗斯方块游戏（炫彩版）项目中创建了一个公共类，名为 Russia，该类中定义的主要方法及作用如表 2.2 所示。

表 2.2　Russia 类中定义的主要方法及作用

方　法	说　明
CakeMode(int n)	设置方块的样式
ConvertorClear()	清空游戏背景
ConvertorDelete()	清空当前方块的区域
MyConvertorMode()	变换当前方块的样式
ConvertorMode(int n)	设置方块的变换样式
Protract(Control control)	绘制方块组合
MyPaint(Graphics g, SolidBrush SolidB, Rectangle rect)	对方块的单个块进行绘制
ConvertorMove(int n)	移动方块

续表

方　法	说　明
RefurbishRow(int Max,int Min)	去除已填满的行
PlaceInitialization()	对信息进行初始化
MoveStop(int n)	判断方块移动时是否出边界

在 Russia 类中定义全局变量，代码如下：

```
public Point firstPoi = new Point(140, 20);                    //定义方块的起始位置
public static Color[,] PlaceColor;                             //记录方块的位置
public static bool[,] Place;                                   //记录方块的位置
public static int conWidth = 0;                                //记录列数
public static int conHeight = 0;                               //记录行数
public static int maxY = 0;                                    //方块在行中的最小高度
public static int conMax = 0;                                  //方块落下后的最大位置
public static int conMin = 0;                                  //方块落下后的最小位置
bool[] tem_Array = { false, false, false, false };             //记录方块组中哪一块所在行中已满
Color ConColor = Color.Coral;
Point[] ArryPoi = new Point[4];                                //方块的数组
Point[] Arryfront = new Point[4];                              //前一个方块的数组
int Cake = 20;                                                 //定义方块的大小
int Convertor = 0;                                             //变换器
Control Mycontrol = new Control();                             //创建 Control
public Label Label_Linage = new Label();                       //创建 Label，用于显示去除的行数
public Label Label_Fraction = new Label();                     //创建 Label，用于显示分数
public static int[] ArrayCent = new int[] { 2, 5, 9, 15 };     //记录加分情况
```

自定义方法 ConvertorClear()用来清空游戏背景，代码如下：

```
///<summary>
///清空游戏背景
///</summary>
public void ConvertorClear()
{
    if (Mycontrol != null)                                                    //如果已载入背景控件
    {
        Graphics g = Mycontrol.CreateGraphics();                              //创建背景控件的 Graphics 类
        Rectangle rect = new Rectangle(0, 0, Mycontrol.Width, Mycontrol.Height);  //获取背景区域
        MyPaint(g, new SolidBrush(Color.Black), rect);                        //用背景色填充背景
    }
}
```

自定义方法 PlaceInitialization()用于初始化记录各方块位置和颜色的多维数组，代码如下：

```
///<summary>
///对信息进行初始化
///</summary>
public void PlaceInitialization()
{
    conWidth=Mycontrol.Width / 20;                                            //获取背景的总行数
    conHeight = Mycontrol.Height / 20;                                        //获取背景的总列数
    Place = new bool[conWidth, conHeight];                                    //定义记录各方块位置的数组
    PlaceColor = new Color[conWidth, conHeight];                              //定义记录各方块颜色的数组
    //对各方块的信息进行初始化
    for (int i = 0; i < conWidth; i++)
    {
        for (int j = 0; j < conHeight; j++)
        {
            Place[i, j] = false;                                              //方块为空
            PlaceColor[i, j] = Color.Black;                                   //与背景色相同
        }
    }
    maxY = conHeight * Cake;                                                  //记录方块的最大值
}
```

自定义方法 CakeMode()主要通过参数值 n 获取指定样式的方块组，代码如下：

```
///<summary>
///设置方块的样式
///</summary>
///<param n="int">标识，方块的样式</param>
public void CakeMode(int n)
{
    ArryPoi[0] = firstPoi;                                                    //记录方块的起始位置
    switch (n)                                                                //根据标识设置方块的样式
    {
        case 1:                                                               //组合“L”方块
            {
                ArryPoi[1] = new Point(firstPoi.X, firstPoi.Y - Cake);        //设置第二块方块的位置
                ArryPoi[2] = new Point(firstPoi.X, firstPoi.Y + Cake);        //设置第三块方块的位置
                ArryPoi[3] = new Point(firstPoi.X + Cake, firstPoi.Y + Cake); //设置第四块方块的位置
                ConColor = Color.Fuchsia;                                     //设置当前方块的颜色
                Convertor = 2;                                                //记录方块的变换样式
                break;
            }
        case 2:                                                               //组合“Z”方块
            {
                ArryPoi[1] = new Point(firstPoi.X, firstPoi.Y - Cake);
                ArryPoi[2] = new Point(firstPoi.X - Cake, firstPoi.Y - Cake);
                ArryPoi[3] = new Point(firstPoi.X + Cake, firstPoi.Y);
                ConColor = Color.Yellow;
                Convertor = 6;
                break;
            }
        case 3:                                                               //组合倒“L”方块
            {
                ArryPoi[1] = new Point(firstPoi.X, firstPoi.Y - Cake);
                ArryPoi[2] = new Point(firstPoi.X, firstPoi.Y + Cake);
                ArryPoi[3] = new Point(firstPoi.X - Cake, firstPoi.Y + Cake);
                ConColor = Color.CornflowerBlue;
                Convertor = 8;
                break;
            }
        case 4:                                                               //组合倒“Z”方块
            {
                ArryPoi[1] = new Point(firstPoi.X, firstPoi.Y - Cake);
                ArryPoi[2] = new Point(firstPoi.X + Cake, firstPoi.Y - Cake);
                ArryPoi[3] = new Point(firstPoi.X - Cake, firstPoi.Y);
                ConColor = Color.Blue;
                Convertor = 12;
                break;
            }
        case 5:                                                               //组合“T”方块
            {
                ArryPoi[1] = new Point(firstPoi.X, firstPoi.Y - Cake);
                ArryPoi[2] = new Point(firstPoi.X + Cake, firstPoi.Y - Cake);
                ArryPoi[3] = new Point(firstPoi.X - Cake, firstPoi.Y - Cake);
                ConColor = Color.Silver;
                Convertor = 14;
                break;
            }
        case 6:                                                               //组合“一”方块
            {
                ArryPoi[1] = new Point(firstPoi.X + Cake, firstPoi.Y);
                ArryPoi[2] = new Point(firstPoi.X - Cake, firstPoi.Y);
                ArryPoi[3] = new Point(firstPoi.X - Cake*2, firstPoi.Y);
                ConColor = Color.Red;
                Convertor = 18;
                break;
            }
        case 7:                                                               //组合“田”方块
            {
                ArryPoi[1] = new Point(firstPoi.X - Cake, firstPoi.Y);
```

```
                ArryPoi[2] = new Point(firstPoi.X - Cake, firstPoi.Y - Cake);
                ArryPoi[3] = new Point(firstPoi.X, firstPoi.Y - Cake);
                ConColor = Color.LightGreen;
                Convertor = 19;
                break;
            }
        }
    }
```

自定义方法 ConvertorMode()主要通过参数 n 获取变换后的方块组样式，如果变换后的方块组超出边界，或与已经排列好的方块重叠，则不对当前方块组进行变换。代码如下：

```
///<summary>
///设置方块的变换样式
///</summary>
///<param n="int">标识，判断变换的样式</param>
public void ConvertorMode(int n)
{
    Point[] tem_ArrayPoi = new Point[4];                                    //定义一个临时数组
    Point tem_Poi = firstPoi;                                               //获取方块的起始位置
    int tem_n = n;                                                          //记录方块的下一个变换样式
    //将当前方块的位置存入临时数组中
    for (int i = 0; i < tem_ArrayPoi.Length; i++)
        tem_ArrayPoi[i] = ArryPoi[i];
    switch (n)                                                              //根据标识变换方块的样式
    {
        case 1:                                                             //设置“L”方块的起始样式
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X + Cake, tem_Poi.Y + Cake);
                tem_n = 2;                                                  //记录变换样式的标志
                break;
            }
        case 2:                                                             //“L”方块组顺时针旋转 90° 的样式
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_ArrayPoi[3] = new Point(tem_Poi.X + Cake, tem_Poi.Y - Cake);
                tem_n = 3;
                break;
            }
        case 3:                                                             //“L”方块组顺时针旋转 180° 的样式
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X - Cake, tem_Poi.Y - Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_n = 4;
                break;
            }
        case 4:                                                             //“L”方块组顺时针旋转 270° 的样式
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[3] = new Point(tem_Poi.X - Cake, tem_Poi.Y + Cake);
                tem_n = 1;                                                  //返回方块的起始样式
                break;
            }
        case 5:                                                             //“Z”型方块
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X - Cake, tem_Poi.Y - Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_n = 6;
                break;
            }
        case 6:
            {
```

```
                tem_ArrayPoi[1] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X + Cake, tem_Poi.Y - Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_n = 5;
                break;
            }
        case 7:                                                          //“倒 L”型方块
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X - Cake, tem_Poi.Y + Cake);
                tem_n = 8;
                break;
            }
        case 8:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_ArrayPoi[3] = new Point(tem_Poi.X + Cake, tem_Poi.Y + Cake);
                tem_n = 9;
                break;
            }
        case 9:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X + Cake, tem_Poi.Y - Cake);
                tem_n = 10;
                break;
            }
        case 10:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_ArrayPoi[3] = new Point(tem_Poi.X - Cake, tem_Poi.Y - Cake);
                tem_n = 7;
                break;
            }
        case 11:                                                         //“倒 Z”型方块
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X + Cake, tem_Poi.Y - Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_n = 12;
                break;
            }
        case 12:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X - Cake, tem_Poi.Y - Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_n = 11;
                break;
            }
        case 13:                                                         //“T”型方块
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X + Cake, tem_Poi.Y - Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X - Cake, tem_Poi.Y - Cake);
                tem_n = 14;
                break;
            }
        case 14:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_n = 15;
                break;
```

```
            }
        case 15:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_ArrayPoi[3] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_n = 16;
                break;
            }
        case 16:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[3] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_n = 13;
                break;
            }
        case 17:                                                    //“一”型方块
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X + Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[3] = new Point(tem_Poi.X - Cake * 2, tem_Poi.Y);
                tem_n = 18;
                break;
            }
        case 18:
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_ArrayPoi[2] = new Point(tem_Poi.X, tem_Poi.Y + Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X, tem_Poi.Y + Cake * 2);
                tem_n = 17;
                break;
            }
        case 19:                                                    //“田”型方块
            {
                tem_ArrayPoi[1] = new Point(tem_Poi.X - Cake, tem_Poi.Y);
                tem_ArrayPoi[2] = new Point(tem_Poi.X - Cake, tem_Poi.Y - Cake);
                tem_ArrayPoi[3] = new Point(tem_Poi.X, tem_Poi.Y - Cake);
                tem_n = 19;
                break;
            }
    }
    bool tem_bool = true;                                           //判断方块是否可变
    //遍历方块的各个子方块
    for (int i = 0; i < tem_ArrayPoi.Length; i++)
    {
        if (tem_ArrayPoi[i].X / 20 < 0)                             //变换后是否超出左边界
        {
            tem_bool = false;                                       //不变换
            break;
        }
        if (tem_ArrayPoi[i].X / 20 >= conWidth)                     //变换后是否超出右边界
        {
            tem_bool = false;
            break;
        }
        if (tem_ArrayPoi[i].Y / 20 >= conHeight)                    //变换后是否超出下边界
        {
            tem_bool = false;
            break;
        }
        if (Place[tem_ArrayPoi[i].X / 20, tem_ArrayPoi[i].Y / 20])  //变换后是否与其他方块重叠
        {
            tem_bool = false;
            break;
        }
    }
    if (tem_bool)                                                   //如果当前方块可以变换
    {
```

```
            //改变当前方块的样式
            for (int i = 0; i < tem_ArrayPoi.Length; i++)
                ArryPoi[i] = tem_ArrayPoi[i];
            firstPoi = tem_Poi;                                    //获取当前方块的起始位置
            Convertor = tem_n;                                     //获取方块下一次的变换样式
        }
    }
```

自定义方法 MyPaint()用来使用 GDI+技术绘制单个方块，代码如下：

```
///<summary>
///对方块的单个块进行绘制
///</summary>
///<param g="Graphics">封装一个绘图的类对象</param>
///<param SolidB="SolidBrush">画刷</param>
///<param rect="Rectangle">绘制区域</param>
public void MyPaint(Graphics g, SolidBrush SolidB, Rectangle rect)
{
    g.FillRectangle(SolidB, rect);                                 //填充一个矩形
}
```

自定义方法 Protract()用来根据 Control 区域绘制方块组合，代码如下：

```
///<summary>
///绘制方块组合
///</summary>
///<param control="Control">控件</param>
public void Protract(Control control)
{
    Mycontrol = control;
    Graphics g = control.CreateGraphics();                         //创建背景控件的 Graphics 类
    //绘制方块的各个子方块
    for (int i = 0; i < ArryPoi.Length; i++)
    {
        Rectangle rect = new Rectangle(ArryPoi[i].X + 1, ArryPoi[i].Y + 1, 19, 19);   //获取子方块的区域
        MyPaint(g, new SolidBrush(ConColor), rect);                //绘制子方块
    }
}
```

自定义方法 MyConvertorMode()用来变换当前方块的样式，代码如下：

```
///<summary>
///变换当前方块的样式
///</summary>
public void MyConvertorMode()
{
    ConvertorDelete();                                             //清空当前方块的区域
    ConvertorMode(Convertor);                                      //设置方块的变换样式
    Protract(Mycontrol);                                           //绘制变换后的组合方块
}
```

自定义方法 ConvertorMove()主要用来对方块组进行下移、左移和右移操作。如果在移动时超出边界，或与其他方块重叠，停止移动；如果方块组移到底端，或与其他方块排列在一起，生成新的方块组。代码如下：

```
///<summary>
///方块移动
///</summary>
///<param n="int">标识，对左右下进行判断</param>
public void ConvertorMove(int n)
{
    //记录方块移动前的位置
    for (int i = 0; i < Arryfront.Length; i++)
        Arryfront[i] = ArryPoi[i];
    switch (n)                                                     //方块的移动方向
    {
        case 0:                                                    //下移
```

```
            {
                //遍历方块中的子方块
                for (int i = 0; i < Arryfront.Length; i++)
                    //使各子方块下移一个方块位
                    Arryfront[i] = new Point(Arryfront[i].X, Arryfront[i].Y + Cake);
                break;
            }
        case 1:                                                          //左移
            {
                for (int i = 0; i < Arryfront.Length; i++)
                    Arryfront[i] = new Point(Arryfront[i].X - Cake, Arryfront[i].Y);
                break;
            }
        case 2:                                                          //右移
            {
                for (int i = 0; i < Arryfront.Length; i++)
                    Arryfront[i] = new Point(Arryfront[i].X + Cake, Arryfront[i].Y);

                break;
            }
    }
    bool tem_bool = MoveStop(n);                                         //记录方块移动后是否出边界
    if (tem_bool)                                                        //如果没有出边界
    {
        ConvertorDelete();                                               //清空当前方块的区域
        //获取移动后方块的位置
        for (int i = 0; i < Arryfront.Length; i++)
            ArryPoi[i] = Arryfront[i];
        firstPoi = ArryPoi[0];                                           //记录方块的起始位置
        Protract(Mycontrol);                                             //绘制移动后方块
    }
    else                                                                 //如果方块到达底部
    {
        if (!tem_bool && n == 0)                                         //如果当前方块是下移
        {
            conMax = 0;                                                  //记录方块落下后的顶端位置
            conMin = Mycontrol.Height;                                   //记录方块落下后的底端位置
            //遍历方块的各个子方块
            for (int i = 0; i < ArryPoi.Length; i++)
            {
                if (ArryPoi[i].Y < maxY)                                 //记录方块的顶端位置
                    maxY = ArryPoi[i].Y;
                Place[ArryPoi[i].X / 20, ArryPoi[i].Y / 20] = true;      //记录指定的位置已存在方块
                PlaceColor[ArryPoi[i].X / 20, ArryPoi[i].Y / 20] = ConColor;  //记录方块的颜色
                if (ArryPoi[i].Y > conMax)                               //记录方块的顶端位置
                    conMax = ArryPoi[i].Y;
                if (ArryPoi[i].Y < conMin)                               //记录方块的底端位置
                    conMin = ArryPoi[i].Y;
            }
            Random rand = new Random();                                  //创建 Random
            int CakeNO = rand.Next(1, 8);                                //获取随机数
            firstPoi = new Point(140, 20);                               //设置方块的起始位置
            CakeMode(Form1.CakeNO);                                      //设置方块的样式
            Protract(Mycontrol);                                         //绘制组合方块
            RefurbishRow(conMax,conMin);                                 //去除已填满的行
            Form1.become = true;                                         //标识，判断可以生成下一个方块
        }
    }
}
```

自定义方法 MoveStop()用于判断当前移动的方块组是否超出边界。代码如下：

```
///<summary>
///判断方块移动时是否出边界
///</summary>
public bool MoveStop(int n)
{
    bool tem_bool = true;
    int tem_width = 0;
```

```
    int tem_height = 0;
    switch (n)
    {
        case 0:                                                          //下移
            {
                //遍历方块中的各个子方块
                for (int i = 0; i < Arryfront.Length; i++)
                {
                    tem_width = Arryfront[i].X / 20;                     //获取方块的横向坐标值
                    tem_height = Arryfront[i].Y / 20;                    //获取方块的纵向坐标值
                    //判断是否超出底边界，或是与其他方块重叠
                    if (tem_height == conHeight || Place[tem_width, tem_height])
                        tem_bool = false;                                //超出边界
                }
                break;
            }
        case 1:                                                          //左移
            {
                for (int i = 0; i < Arryfront.Length; i++)
                {
                    tem_width = Arryfront[i].X / 20;
                    tem_height = Arryfront[i].Y / 20;
                    //判断是否超出左边界，或是与其他方块重叠
                    if (tem_width == -1 || Place[tem_width, tem_height])
                        tem_bool = false;
                }
                break;
            }
        case 2:                                                          //右移
            {
                for (int i = 0; i < Arryfront.Length; i++)
                {
                    tem_width = Arryfront[i].X / 20;
                    tem_height = Arryfront[i].Y / 20;
                    //判断是否超出右边界，或是与其他方块重叠
                    if (tem_width == conWidth || Place[tem_width, tem_height])
                        tem_bool = false;
                }
                break;
            }
    }
    return tem_bool;
}
```

自定义方法 ConvertorDelete()用于清空当前位置的方块组。代码如下：

```
///<summary>
///清空当前方块的区域
///</summary>
public void ConvertorDelete()
{
    Graphics g = Mycontrol.CreateGraphics();                             //创建背景控件的 Graphics 类
    for (int i = 0; i < ArryPoi.Length; i++)                             //遍历方块的各个子方块
    {
        Rectangle rect = new Rectangle(ArryPoi[i].X, ArryPoi[i].Y, 20, 20);  //获取各子方块的区域
        MyPaint(g, new SolidBrush(Color.Black), rect);                   //用背景色填充背景
    }
}
```

自定义方法 RefurbishRow()主要判断当前落下的方块组不能再移动时，已经排列好的所有方块中是否有填满一行的情况。如果有，则去除该行，并将该行以上的各行下移。代码如下：

```
///<summary>
///去除已填满的行
///</summary>
public void RefurbishRow(int Max,int Min)
{
    Graphics g = Mycontrol.CreateGraphics();                             //创建背景控件的 Graphics 类
```

```
int tem_max = Max / 20;                                          //获取方块的最大位置在多少行
int tem_min = Min / 20;                                          //获取方块的最小位置在多少行
bool tem_bool = false;
//初始化记录刷新行的数组
for (int i = 0; i < tem_Array.Length; i++)
    tem_Array[i] = false;
int tem_n = maxY;                                                //记录最高行的位置
for (int i = 0; i < 4; i++)                                      //查找要刷新的行
{
    if ((tem_min + i) > 19)                                      //如果超出边界
        break;                                                   //退出本次操作
    tem_bool = false;
    //如果当前行中有空格
    for (int k = 0; k < conWidth; k++)
    {
        if (!Place[k, tem_min + i])                              //如果当前位置为空
        {
            tem_bool = true;
            break;
        }
    }
    if (!tem_bool)                                               //如果当前行为满行
    {
        tem_Array[i] = true;                                     //记录为刷新行
    }
}
int Progression = 0;                                             //记录去除的几行
//如果有刷新行
if (tem_Array[0] == true || tem_Array[1] == true || tem_Array[2] == true || tem_Array[3] == true)
{
    int Trow = 0;                                                //记录最小行数
    for (int i = (tem_Array.Length - 1); i >= 0; i--)            //遍历记录刷新行的数组
    {
        if (tem_Array[i])                                        //如果是刷新行
        {
            Trow = Min / 20 + i;                                 //记录最小行数
            //将刷新行到背景顶端的区域下移
            for (int j = Trow; j >=1 ; j--)
            {
                for (int k = 0; k < conWidth; k++)
                {
                    PlaceColor[k, j] = PlaceColor[k, j - 1];     //记录方块的位置
                    Place[k, j] = Place[k, j - 1];               //记录方块的位置
                }
            }
            Min += 20;                                           //方块的最小位置下移一个方块位
            //将背景的顶端清空
            for (int k = 0; k < conWidth; k++)
            {
                PlaceColor[k, 0] = Color.Black;                  //记录方块的位置
                Place[k, 0] = false;                             //记录方块的位置
            }
            Progression += 1;                                    //记录刷新的行数
        }
    }
    //在背景中绘制刷新后的方块图案
    for (int i = 0; i < conWidth; i++)
    {
        for (int j = 0; j <= Max / 20; j++)
        {
            //获取各方块的区域
            Rectangle rect = new Rectangle(i * Cake + 1, j * Cake + 1, 19, 19);
            MyPaint(g, new SolidBrush(PlaceColor[i, j]), rect);  //绘制已落下的方块
        }
    }
    //显示当前的刷新行数
    Label_Linage.Text = Convert.ToString(Convert.ToInt32(Label_Linage.Text) + Progression);
    //显示当前的得分情况
    Label_Fraction.Text = Convert.ToString(Convert.ToInt32(Label_Fraction.Text) + ArrayCent[Progression - 1]);
```

```
    }
}
```

2.5 功 能 设 计

2.5.1 设计窗体

俄罗斯方块游戏（炫彩版）的窗体设计主要分为两个步骤，分别是设置窗体、向窗体中添加控件或组件，下面分别介绍。

1. 设置窗体

俄罗斯方块游戏（炫彩版）的主窗体为 Frm_Main 窗体，其属性设置如表 2.3 所示。

表 2.3 Frm_Main 窗体的属性值列表

属　　性	值	说　　明
KeyPreview	True	向窗体注册键盘事件
MaximizeBox	false	设置窗体不可以最大化
MinimizeBox	false	设置窗体不可以最小化
Width	431	设置窗体的宽度
Height	450	设置窗体的高度
StartPosition	CenterScreen	设置窗体首次出现时的位置为屏幕中心
Text	俄罗斯方块	设置窗体的标题

2. 向窗体中添加控件或组件

Frm_Main 窗体是俄罗斯方块游戏（炫彩版）的主窗体，该窗体布局稍显复杂。因此，首先看一下它的整体布局效果，如图 2.7 所示。

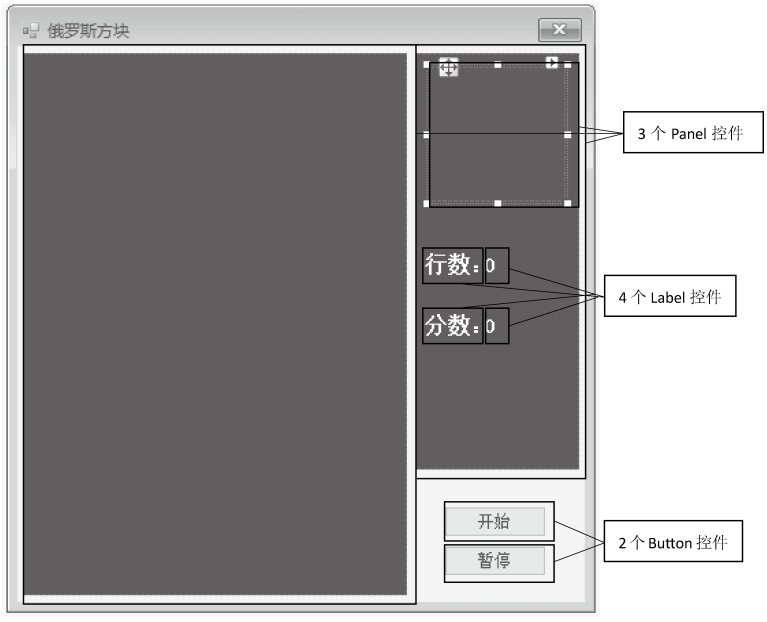


图 2.7 Frm_Main 窗体整体布局效果

Frm_Main 窗体中用到的控件、组件及其属性如表 2.4 所示。

表 2.4 Frm_Main 窗体中用到的控件、组件及对应属性设置

控件/组件类型	属　性	值	说　明
Panel	BackColor	WindowText	设置游戏区的背景色
	X	4	设置游戏区的起始点 X 坐标
	Y	5	设置游戏区的起始点 Y 坐标
	Width	281	设置游戏区的宽度
	Height	401	设置游戏区的高度
Panel	BackColor	Black	设置游戏提示区的背景色
	X	291	设置游戏提示区的起始点 X 坐标
	Y	5	设置游戏提示区的起始点 Y 坐标
	Width	120	设置游戏提示区的宽度
	Height	308	设置游戏提示区的高度
Panel	BackColor	Black	设置方格提示区的背景色
	X	10	设置方格提示区的起始点 X 坐标
	Y	10	设置方格提示区的起始点 Y 坐标
	Width	100	设置方格提示区的宽度
	Height	100	设置方格提示区的高度
Label	Font	宋体，12pt，style=Bold	设置行数标识控件的字体及字体大小
	ForeColor	Window	设置行数标识控件的字体颜色
	X	4	设置行数标识控件的 X 坐标
	Y	148	设置行数标识控件的 Y 坐标
	Text	行数：	设置行数标识控件的文本
Label	Font	宋体，12pt，style=Bold	设置分数标识控件的字体及字体大小
	ForeColor	White	设置分数标识控件的字体颜色
	X	4	设置分数标识控件的 X 坐标
	Y	193	设置分数标识控件的 Y 坐标
	Text	分数：	设置分数标识控件的文本
Label	Font	宋体，10.5pt，style=Bold	设置显示行数标签的字体及字体大小
	ForeColor	White	设置显示行数标签的字体颜色
	X	48	设置显示行数标签的 X 坐标
	Y	150	设置显示行数标签的 Y 坐标
	Text	0	设置显示行数标签的文本
Label	Font	宋体，10.5pt，style=Bold	设置显示分数标签的字体及字体大小
	ForeColor	White	设置显示分数标签的字体颜色
	X	48	设置显示分数标签的 X 坐标
	Y	195	设置显示分数标签的 Y 坐标
	Text	0	设置显示分数标签的文本

续表

控件/组件类型	属　性	值	说　明
Button	X	312	设置“开始”按钮的 X 坐标
	Y	340	设置“开始”按钮的 Y 坐标
	Text	开始	设置“开始”按钮显示的文本
Button	X	312	设置“暂停”按钮的 X 坐标
	Y	369	设置“暂停”按钮的 Y 坐标
	Text	暂停	设置“暂停”按钮显示的文本
TextBox	X	291	设置辅助的文本框控件的 X 坐标
	Y	480	设置辅助的文本框控件的 Y 坐标
Timer	Interval	300	设置计时器组件的事件触发频率

说明

向 Frm_Main 窗体中添加表 2.4 所列的控件时，第 3 个 Panel 控件和所有的 Label 控件都添加在第 2 个 Panel 控件中。具体方法为：将鼠标焦点定位到第 2 个 Panel 控件中，然后在“工具箱”窗口中双击要添加的 Panel 控件或者 Label 控件。

2.5.2　初始化游戏场景

在 Frm_Main 窗体的设计界面，单击右键，选择“查看代码”菜单项，切换到 Frm_Main 窗体的代码页，首先在 Frm_Main 窗体类的内部声明公共的变量及对象，代码如下：

```
Russia MyRussia = new Russia();                    //实例化 Russia 类，用于操作游戏
Russia TemRussia = new Russia();                   //实例化 Russia 类，用于生成下一个方块样式
public static int CakeNO = 0;                      //记录下一个方块样式的标识
public static bool become = false;                 //判断是否生成下一个方块的样式
public static bool isbegin = false;                //判断当前游戏是否开始
public bool ispause = true;                        //判断是否暂停游戏
```

在 Frm_Main 窗体类的内部，定义一个无返回值类型的 beforehand()方法，用来生成下一个方块的样式。beforehand()方法代码如下：

```
///<summary>
///生成下一个方块的样式
///</summary>
public void beforehand()
{
    Graphics P3 = panel3.CreateGraphics();
    P3.FillRectangle(new SolidBrush(Color.Black), 0, 0, panel3.Width, panel3.Height);
    Random rand = new Random();                    //实例化 Random
    CakeNO = rand.Next(1, 8);                      //获取随机数
    TemRussia.firstPoi = new Point(50, 30);        //设置方块的起始位置
    TemRussia.CakeMode(CakeNO);                    //设置方块的样式
    TemRussia.Protract(panel3);                    //绘制组合方块
}
```

切换到 Frm_Main 窗体的设计界面，双击“开始”按钮控件，自动触发其 Click 事件。该事件中，主要使用 Russia 公共类中的相关方法初始化游戏的场景，并绘制初始的俄罗斯方块。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    MyRussia.ConvertorClear();                     //清空整个控件
    MyRussia.firstPoi = new Point(140, 20);        //设置方块的起始位置
```

```
    label3.Text = "0";                          //显示去除的行数
    label4.Text = "0";                          //显示分数
    MyRussia.Label_Linage = label3;             //将 label3 控件加载到 Russia 类中
    MyRussia.Label_Fraction = label4;           //将 label4 控件加载到 Russia 类中
    timer1.Interval = 500;                      //设置下移的速度
    MyRussia.Add_degree = 1;
    MyRussia.UpCareer = timer1.Interval;
    timer1.Enabled = false;                     //停止计时
    timer1.Enabled = true;                      //开始计时
    Random rand = new Random();                 //实例化 Random
    CakeNO = rand.Next(1, 8);                   //获取随机数
    MyRussia.CakeMode(CakeNO);                  //设置方块的样式
    MyRussia.Protract(panel1);                  //绘制组合方块
    beforehand();                               //生成下一个方块的样式
    MyRussia.PlaceInitialization();             //初始化 Random 类中的信息
    isbegin = true;                             //判断是否开始
    ispause = true;
    MyRussia.timer = timer1;
    button2.Text = "暂停";
    ispause = true;
    textBox1.Focus();                           //获取焦点
}
```

2.5.3 生成游戏方块并使其自动下落

切换到 Frm_Main 窗体的设计界面，双击 timer1 组件，会自动触发其 Tick 事件。该事件中，首先使用自定义的 beforehand()方法生成下一个游戏方块，然后控制用户未按下键盘上的“向下箭头”键时，方块自动向下移动。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (MyRussia.Add_n != 5)
    {
        MyRussia.ConvertorMove(0);              //方块下移
        if (become)                             //如果显示新的方块
        {
            beforehand();                       //生成下一个方块
            become = false;
        }
        textBox1.Focus();                       //获取焦点
    }
    else
    {
        MyRussia.panel = panel1;
        MyRussia.ConvertorClear();              //清空整个控件
        MyRussia.firstPoi = new Point(140, 20); //设置方块的起始位置
        timer1.Interval = 500;                  //下移的速度
        MyRussia.UpCareer = timer1.Interval;
        timer1.Enabled = false;                 //停止计时
        timer1.Enabled = true;                  //开始计时
        Random rand = new Random();             //实例化 Random
        CakeNO = rand.Next(1, 8);               //获取随机数
        MyRussia.CakeMode(CakeNO);              //设置方块的样式
        MyRussia.Protract(panel1);              //绘制组合方块
        beforehand();                           //生成下一个方块的样式
        MyRussia.PlaceInitialization();         //初始化 Random 类中的信息
        textBox1.Focus();                       //获取焦点
        isbegin = true;                         //判断是否开始
        MyRussia.timer = timer1;
    }
}
```

2.5.4 使用键盘控制方块的变换及移动

切换到 Frm_Main 窗体的设计界面，选中窗体，在其“属性”对话框中单击图标。在列表中找到 KeyDown，然后双击，触发其 KeyDown 事件。该事件中，使用键盘控制方块的向下、向左和向右移动，以及方格样式的变换。代码如下：

```
private void Frm_Main_KeyDown(object sender, KeyEventArgs e)
{
    if (!isbegin)                                   //如果没有开始游戏
        return;
    if (!ispause)                                   //如果游戏暂停
        return;
    if (e.KeyCode == Keys.Up)                       //如果当前按下的是↑键
        MyRussia.MyConvertorMode();                 //变换当前方块的样式
    if (e.KeyCode == Keys.Down)                     //如果当前按下的是↓键
    {
        timer1.Interval = MyRussia.UpCareer - 50;   //增加下移的速度
        MyRussia.ConvertorMove(0);                  //方块下移
    }
    if (e.KeyCode == Keys.Left)                     //如果当前按下的是←键
        MyRussia.ConvertorMove(1);                  //方块左移
    if (e.KeyCode == Keys.Right)                    //如果当前按下的是→键
        MyRussia.ConvertorMove(2);                  //方块右移
}
```

切换到 Frm_Main 窗体的设计界面，选中窗体，在其“属性”对话框中单击图标。在列表中找到 KeyUp，然后双击，触发其 KeyUp 事件。该事件中，首先判断游戏的当前状态，如果是未开始或者暂停状态，则返回；如果当前松开的是“向下箭头”键，则恢复方块的下移速度，并切换鼠标焦点。代码如下：

```
private void Frm_Main_KeyUp(object sender, KeyEventArgs e)
{
    if (!isbegin)                                   //如果游戏没有开始
        return;
    if (!ispause)                                   //如果暂停游戏
        return;
    if (e.KeyCode == Keys.Down)                     //如果当前松开的是↓键
    {
        timer1.Interval = MyRussia.UpCareer;        //恢复下移的速度
    }
    textBox1.Focus();                               //获取焦点
}
```

2.5.5 暂停和继续游戏

切换到 Frm_Main 窗体的设计界面，双击“暂停”按钮控件，自动触发其 Click 事件。该事件中，主要通过计时器的可用状态控制游戏的暂停和继续。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (timer1.Enabled == true)
    {
        timer1.Stop();                              //暂停
        button2.Text = "继续";
        ispause = false;
        textBox1.Focus();                           //获取焦点
    }
    else
    {
        timer1.Start();                             //继续
        button2.Text = "暂停";
        ispause = true;
    }
}
```

2.6 项 目 运 行

通过前述步骤，设计并完成了“俄罗斯方块游戏（炫彩版）”项目的开发。下面运行该项目，检验一下我们的开发成果。使用 Visual Studio 打开俄罗斯方块游戏（炫彩版）项目，单击工具栏中的“启动”按钮或者按 F5 快捷键，即可成功运行该项目。游戏启动后的界面效果如图 2.8 所示。

本章主要讲解了如何使用 C#开发一个炫彩版的俄罗斯方块游戏。其中，在实现游戏的基本逻辑功能时，主要用到了随机数生成、数组、面向对象编程、Timer 计时器等技术，而游戏界面的绘制主要使用了 GDI+技术，游戏中方块的变换、移动等主要通过键盘处理技术进行控制。通过本项目的开发，读者不仅可以巩固 C#编程基础知识，而且能够加深对游戏开发流程的理解，为未来参与更复杂的游戏开发奠定坚实基础。

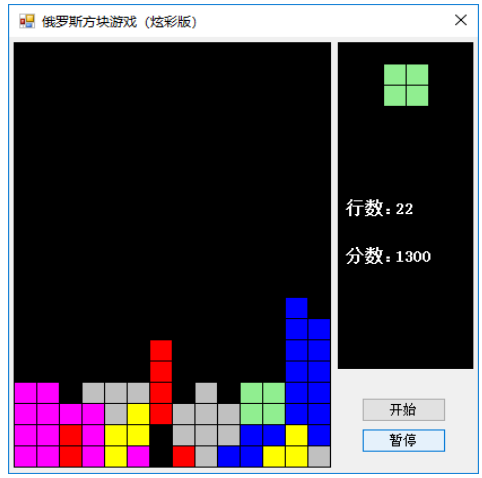


图 2.8 俄罗斯方块游戏（炫彩版）

2.7 源 码 下 载

本章详细地讲解了如何编码实现“俄罗斯方块游戏（炫彩版）”项目的各个功能。为了方便读者学习，本书提供了完整的项目源码，扫描右侧二维码即可下载。

源码下载