飞行游戏说明

1、背景

随着人们生活质量的不断提高以及个人电脑和网络的普及，人们的业余生活的要求也在不断提高，选择一款好玩，耐玩，容易上手，品质优良的游戏已经成为一种流行的休闲方式。可以说在人们的日常生活中，除了工作、学习，玩游戏的时间正在逐渐增加。飞行射击类游戏正是老少咸宜的小游戏，在等待途中，无聊时，亦或是其它空闲时分，是非常好的打发时间的游戏，也是目前游戏平台玩家众多的一类游戏。所以，开发出大家都比较喜欢的，高品质的休闲互动游戏，将会受到人们的普遍欢迎。让人们在工作学习之余，享受游戏的快乐，也是一款游戏真正成功的意义。

基于.net 的飞行游戏设计与实现，是为了能开发出一款好玩，耐玩，容易上手的小型飞行射击类游戏。通过这款游戏使人们可以打发无聊的时间。同时，也不会太难而以至于可玩性不强。

2、设计的内容

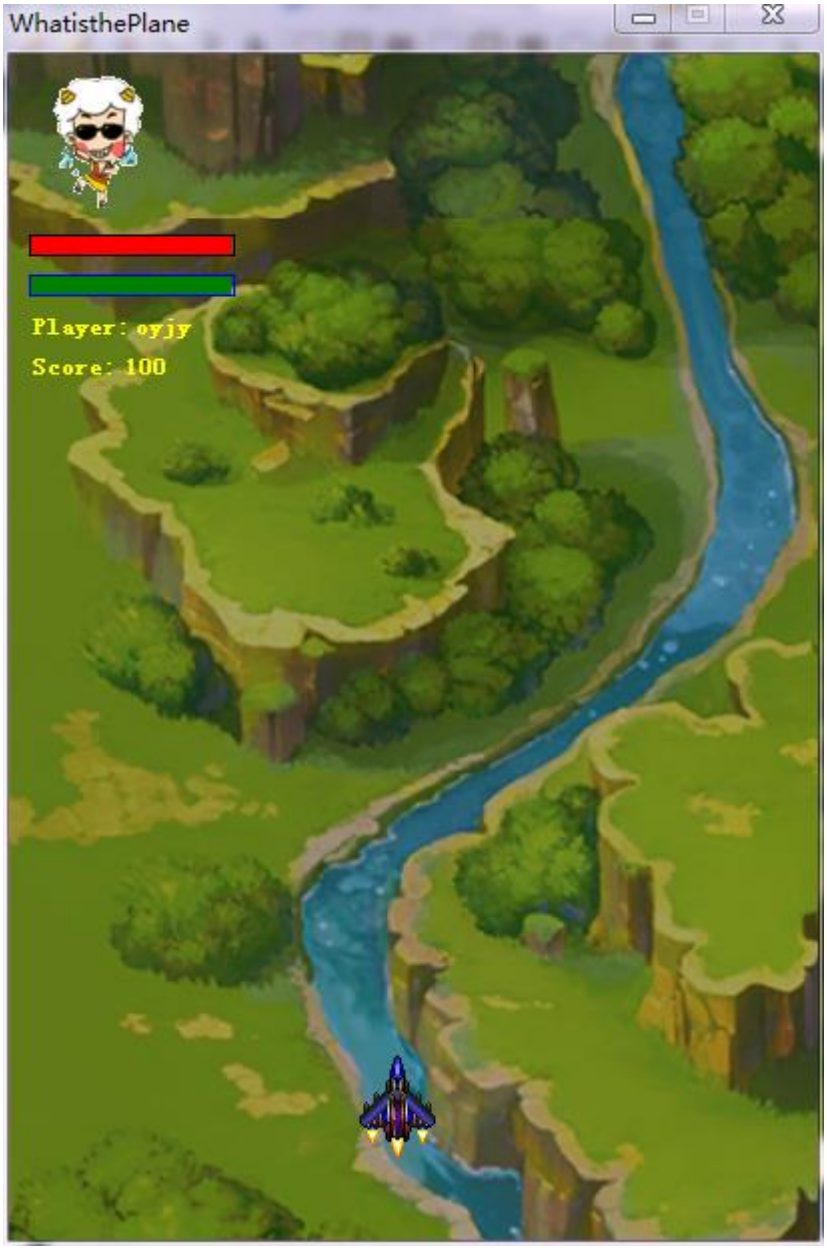
本系统为基于.net 的飞行游戏，主要要求如下：

1) 游戏背景滚动
2) 游戏音乐循环播放
3) 能用键盘控制我方飞机上下左右移动，并能发射子弹
4) 能随机生成各种类型敌机，并发射子弹攻击我方
5) 能进行碰撞检测，子弹击中飞机，生命值减少直到消失
6) 能生成加血包，补充我方生命值

3、程序运行说明

1) 程序初始化，玩家点击开始游戏并进入游戏。
2) 玩家通过控制移动按键实现对飞机的方向控制，按相应按钮发射子弹。若子弹打到敌机上，则敌机爆炸则玩家增加积分。若敌机的子弹打到玩家的飞机身上，玩家减少一定血条。
3) 游戏中，随机掉落不同用途的道具，例如：增加蓝条、血条，花式子弹等等。
4) 当玩家积分达到 1000 时，出现 BOSS，打败 BOSS 则晋级下一关。
5) 当玩家的子弹射击到敌机时，敌机发生爆炸。当玩家血量为零时，玩家发生爆炸。

4、预期效果



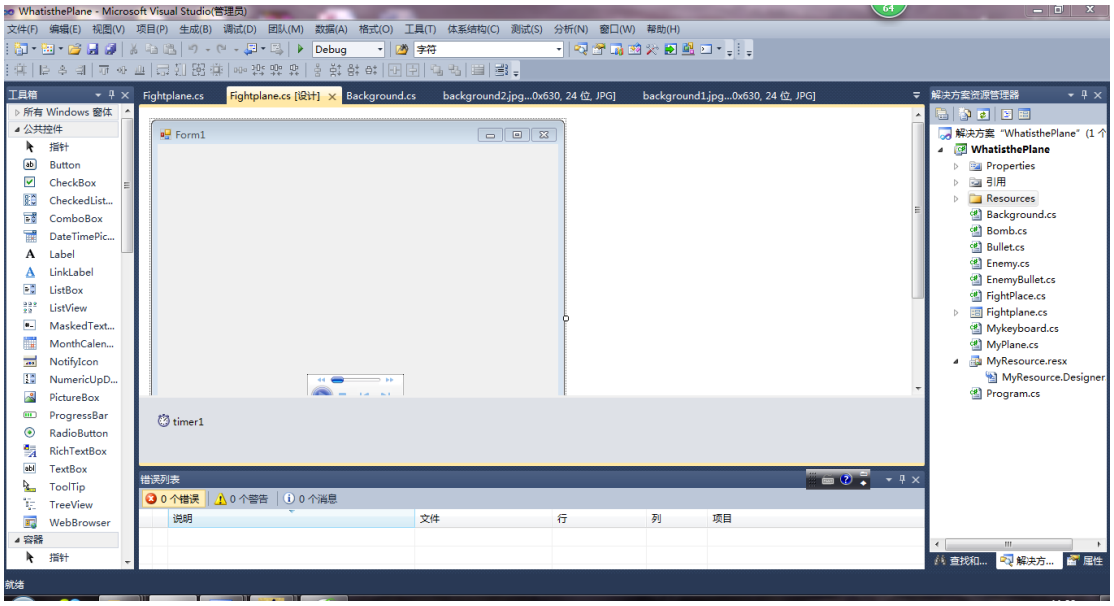

WhatisthePlane
Player: oyjy
Game Over

飞行游戏开发指导

1、安装 vs2010 开发平台

Microsoft Visual Studio（简称 VS）是美国微软公司的开发工具包系列产品。VS 是一个基本完整的开发工具集，它包括了整个软件生命周期中所需要的大部分工具，如 UML 工具、代码管控工具、集成开发环境(IDE)等等。所写的目标代码适用于微软支持的所有平台，包括 Microsoft Windows、Windows Mobile、Windows CE 、.NET Framework 、.NET Compact Framework 和 Microsoft Silverlight 及 Windows Phone。

Visual Studio 2010 作为一个集成解决方案，适用于无论是个人或者各种规模的开发团队。Visual Studio2010 实现了同事间的无缝协作，提高了生产效率与专注度，最终好的点子变成了优秀的现实应用。



2、使用 C#作为开发语言

C#是一种安全的、稳定的、简单的、优雅的，由 C 和 C++衍生出来的面向对象的编程语言。它在继承 C 和 C++强大功能的同时去掉了一些它们的复杂特性（例如没有宏以及不允许多重继承）。C#综合了 VB 简单的可视化操作和 C++的高运行效率，以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程的支持成为.NET 开发的首选语言。它的优点如下：

（1）强大的.Net Framework 托管代码集合类：封装了大多数 windows 上使用的技术组件类，文件系统，UI 界面，数据源访问，网络访问，COM 互操作(图形图像多媒体,WPF 图形系统)，没有的可以通过.net 的平台调用 win API 函数来得到。

（2）较简单的语言特性：自动内存管理，单继承，支持事件、委托、属性、Linq 等一系列让业务开发更简单的功能。

简单的 C#程序

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Proj2_1                    //定义命名空间
{
   class Program                     //定义类
     {
     static void Main(string[] args)
       /*程序入口，static表示静态方法。args是形参*/
      {
            int a, b, c;                    //定义变量
          Console.WriteLine("a:");          //输出屏幕提示信息
          a=int.Parse(Console.ReadLine());        //从键盘获取字符串并转换成整数
           Console.WriteLine("b:");
           b = int.Parse(Console.ReadLine());
            c = a + b;                              //加法运算
            Console.WriteLine("a+b={0}", c);      //输出结果
        }
     }
}
```

例如，以下声明了一个Person类：

```
public class Person
{  public int pno;                    //编号
        public string pname;    //姓名
        public void setdata(int no,string name)
        {
                pno=no; pname=name;
        }
        public void dispdata()
        {
                Console.WriteLine("{0} {1}", pno, pname);
        }
}
```

```
using System;
namespace Proj6_1
{   public class TPoint       //声明类TPoint
      {       public int x,y;                 //类的私有字段
            public void setpoint(int x1,int y1)
                 {    x=x1;y=y1;    }
            public int getpointx()
             {return x;    }
            public int getpointy()
             {return y;    }
}
      class Program
      {     int x,y;
            static void Main(string[] args)
            {     TPoint p1 = new TPoint();       //  定义对象p1
                  p1.setpoint(2,6);
                 x = p1.x;
                 y = p1.y;
          }
      }
}
```

3、事件驱动编程

窗体（Form）是一个窗口或对话框，是存放各种控件（包括标签、文本框、命令按钮等）的容器，可用来向用户显示信息。

在 C#中，窗体分为如下两种类型：

（1）普通窗体，也称为单文档窗体（SDI），前面所有创建的窗体均为普通窗体。普通窗体又分为如下两种：

● 模式窗体。这类窗体在屏幕上显示后用户必须响应，只有在它关闭后才能操作其他窗体或程序。

● 无模式窗体。这类窗体在屏幕上显示后用户可以不必响应，可以随意切换到其他窗体或程序进行操作。通常情况下，当建立新的窗体时，都默认设置为无模式窗体。

（2）MDI 父窗体，即多文档窗体，其中可以放置普通子窗体。

【例8.4】 设计一个窗体，说明单选按钮的使用方法。

Form4窗体：

（1）设计界面

按钮点击事件

（2）事件过程：

```
private void button1_Click(object sender, EventArgs e)
{   if (radiobutton3.Checked)
      MessageBox.Show("您选对了,这是微软公司开发的操作系统",
          "信息提示", MessageBoxButtons.OK);
    else if (radiobutton1.Checked || radiobutton4.Checked)
      MessageBox.Show("您选错了,这是程序设计语言",
          "信息提示", MessageBoxButtons.OK);
    else
      MessageBox.Show("您选错了,这是数据库管理系统",
          "信息提示", MessageBoxButtons.OK);
}
```

4、GDI+绘图

什么是GDI+?

想一下，如果同学们要进行绘画，你要准备什么工作？

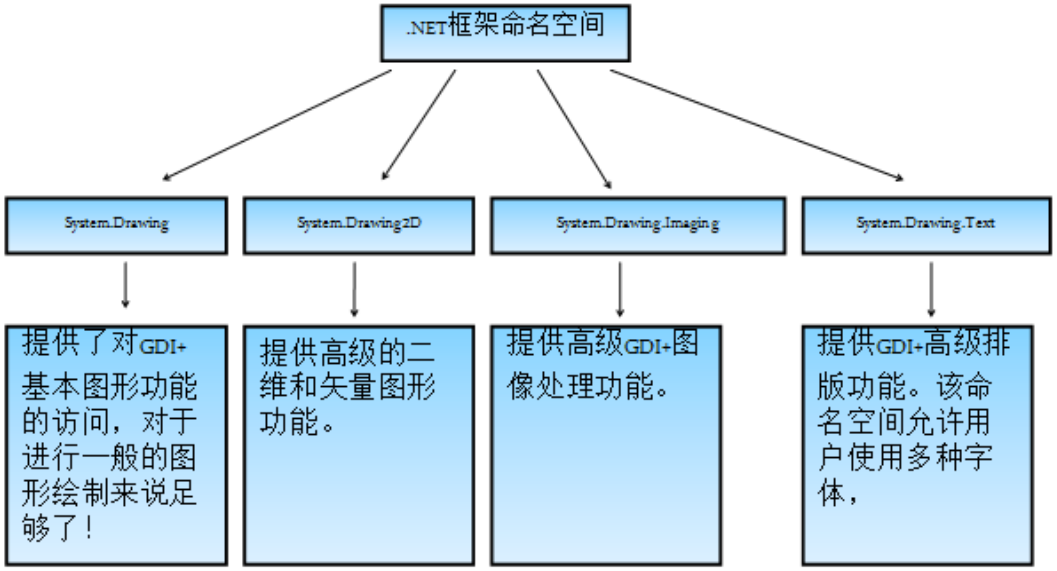
怎么样开始绘画呢？

GDI+的基本概念

GDI：即Graphics Device Interface，图形设备接口

GDI+是Microsoft的新一代的二维图形系统，它完全面向对象，可在Windows窗体应用程序中以编程方式绘制或操作图形图像。

GDI+的命名空间



GDI+结构？

Graphics对象

- Graphics类是核心，创建的Graphics对象相当于一张画布。

一般，图形设计过程分为两步：
创建Graphics对象、
使用Graphics对象的方法进行绘图。

创建Graphics对象的方法

A 利用窗体或控件的Paint事件的参数PaintEventArgs

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e){
        Graphics g=e.Graphics;
}
```

B(一般使用) 使用窗体或控件的CreateGraphics方法，用于对象已经存在的情况下：

```
Graphics g;
g=this.CreateGraphics();
```

C 使用Image类的派生类创建Graphics对象，用于在C#中对图像进行处理的场合：

```
Bitmap b=new Bitmap("ddd.bmp");
Graphics g=Graphics.FromImage(b);
```

Pen对象

Pen 对象又称为画笔对象。
用途：绘制线条、多边形、曲线等几何图形。
Pen 对象的主要属性：宽度、样式、颜色

```
Pen p1=new Pen(Color);//创建某一颜色的Pen 对象
Pen p2=new Pen(Brush);//创建某一刷子样式的Pen 对象
Pen p3=new Pen(Brush,float);//创建某一刷子样式并具有相应宽度的的Pen 对象
Pen p4=new Pen(Color,float);//创建某一颜色和相应宽度的Pen 对象
```

Brush对象（画刷）

作用：一般用来填充图形。
Brush类是一个抽象类，不能实例化，只能使用它的派生类

SolidBrush（单色画刷）
（包含在命名空间System.Drawing中）、
HatchBrush（阴影画刷）、
LinearGradientBrush（颜色渐变画刷）、
PathGradientBrush（使用路径及复杂的混色渐变画刷）、
TextureBrush（纹理画刷）

Font对象

Font对象建立不同的字体。
Font对象的常用属性如下：Bold、Italic、Regular、Strikeout、Underline等等。

```
Graphics g=this.CreateGraphics();
Font fi=new Font("Tahoma",20,FontStyle.Bold|FontStyle.Italic);
g.DrawString("GDI+编程世界",fi,new SolidBrush(Color.Blue),14,10);
```

方正启体简体 方正正粗黑简体
方正正大黑简体 方正正黑简体
方正正纤黑简体 方正正中黑简体
方正正准黑简体 移动专用字体

常用图形的绘制

1、画直线
使用Graphics类的DrawLine方法，格式为：
DrawLine(画笔,x1,y1,x2,y2)
功能：在点（x1,y1），（x2,y2）之间画一条直线

```
Graphics g=this.CreateGraphics();//生成图形对象
Pen Mypen=new Pen(Color.Blue ,5);//生成画笔,蓝色，5个像素
g.DrawLine(Mypen,1,1,30,30);//画线
Point pt1=new Point(1,30);//生成起点
Point pt2=new Point(30,1);//生成终点
g.DrawLine(Mypen,pt1,pt2);//画线
```

2、画椭圆

使用Graphics类的DrawEllipse方法，格式为：

A、DrawEllipse（画笔,矩形结构数据）

功能：绘制一个边界由矩形结构数据定义的椭圆。

B、DrawEllipse（画笔,x，y，width，height）

功能：绘制一个由边框定义的椭圆。

```
Graphics g=this.CreateGraphics();//生成图形对象
Pen Mypen=new Pen(Color.Blue ,5);//生成画笔,蓝色，5个像素
g.DrawEllipse(Mypen,1,1,80,40);//画椭圆
Rectangle rect=new Rectangle(85,1,165,40);//生成矩形
g.DrawEllipse (Mypen,rect);//画椭圆
```

3、画圆弧

使用Graphics类的DrawArc方法，格式为：

A、DrawArc（画笔,矩形结构数据，实数，实数）

功能：绘制由指定矩形的内接椭圆的一段圆弧。

B、DrawArc（画笔,x，y，width，height，整数，整数）

功能：绘制一段弧线，该弧线由一对坐标、宽度、高度指定椭圆的一段圆弧。

```
Graphics g=this.CreateGraphics();//生成图形对象
Pen Mypen=new Pen(Color.Blue ,5);//生成画笔,蓝色，5个像素
g.DrawArc(Mypen,1,1,80,40,90,270);//画弧线
Rectangle rect=new Rectangle(85,1,165,40);//生成起点 生成矩形结构
g.DrawArc (Mypen,rect,0,90);//画弧线
```

5、画矩形

使用Graphics类的DrawRectangle方法，格式为：

A、DrawRectangle（画笔,矩形结构数据）

功能：绘制一个边界由矩形结构数据定义的矩形。

B、DrawRectangle（画笔,x，y，width，height）

功能：绘制一个由左上角坐标、宽度、高度定义的矩形

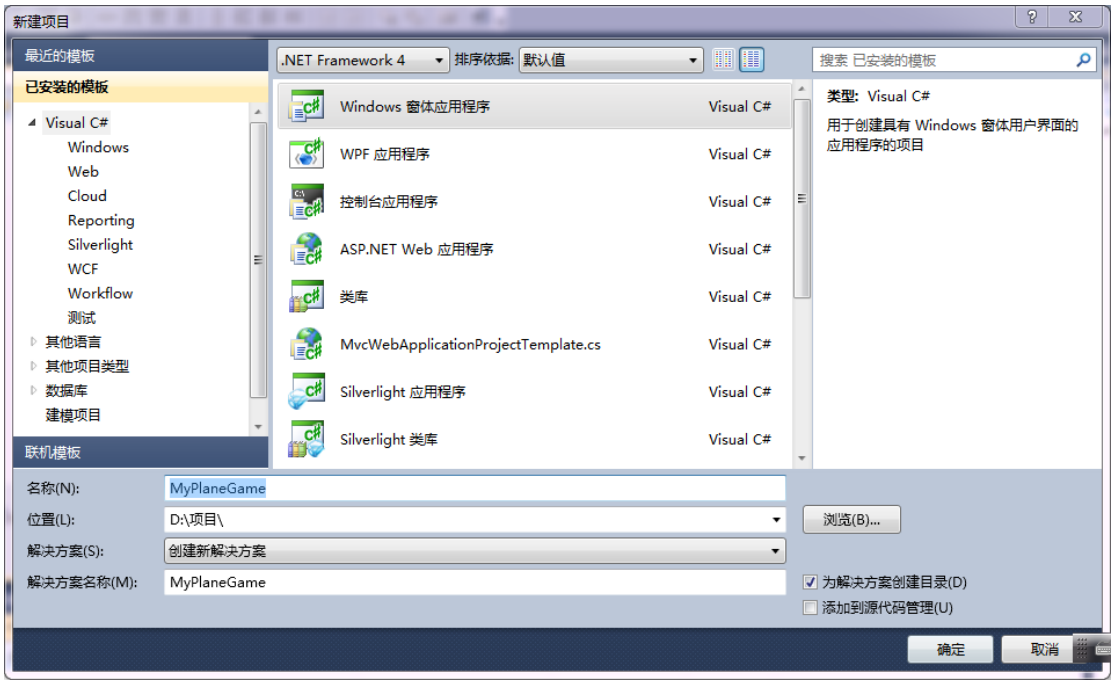
```
Graphics g=this.CreateGraphics();//生成图形对象
Pen Mypen=new Pen(Color.Blue ,2);//生成画笔,蓝色，2个像素
g.DrawRectangle (Mypen,5,5,80,40);//画矩形
Rectangle rect=new Rectangle(85,15,140,50);//生成矩形
g.DrawRectangle (Mypen,rect);//画矩形
```

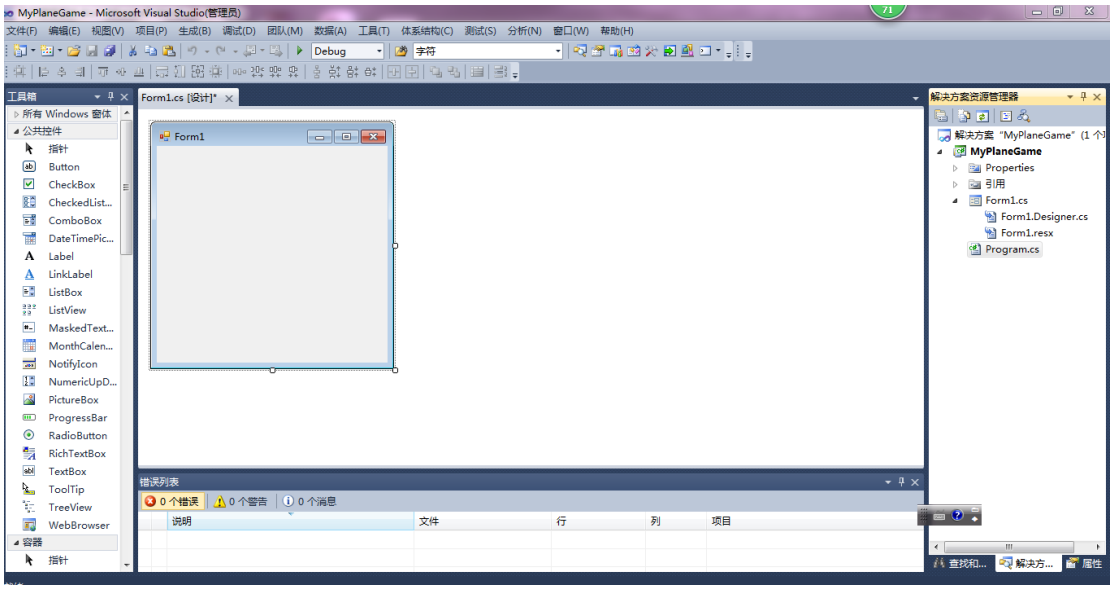
5、绘制游戏背景

（1）从网络或 PS 选择背景图片

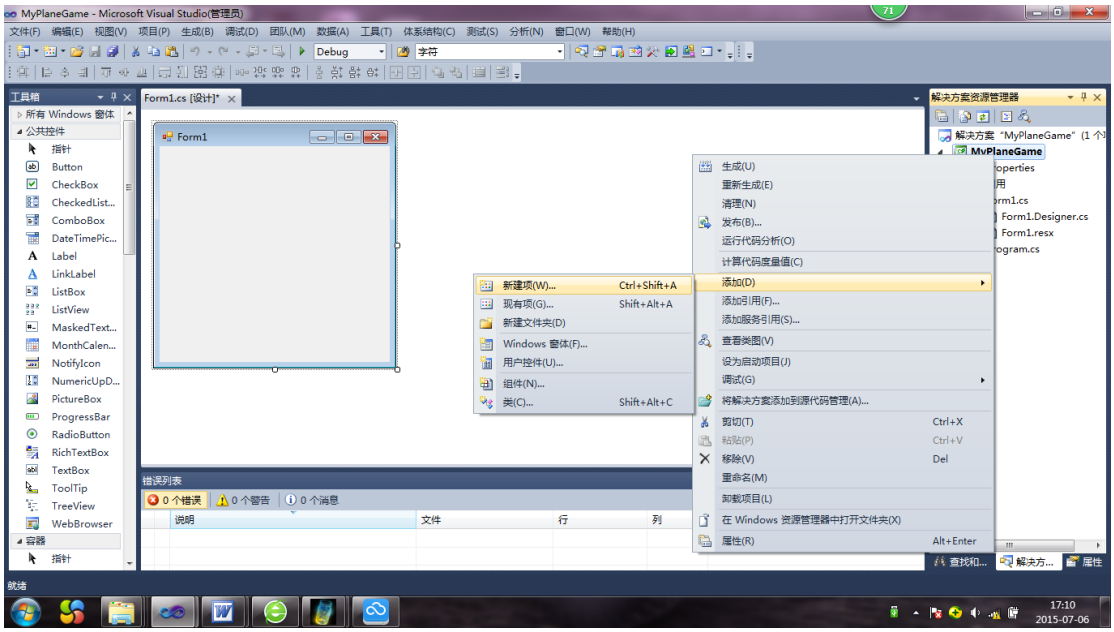
注意背景图片的连续性，因为背景实际上是多张背景图片的切换，要保证切换效果，必须让图片的上下过渡自然，可以 PS 处理一下.

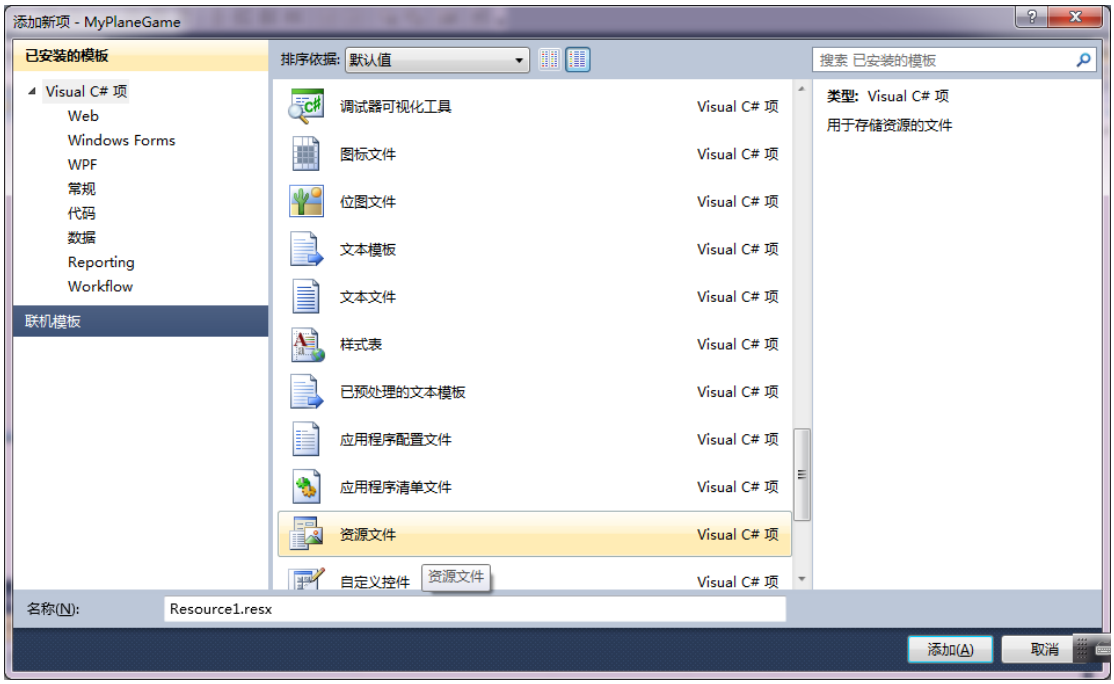
(2)在 VS 中建立游戏项目

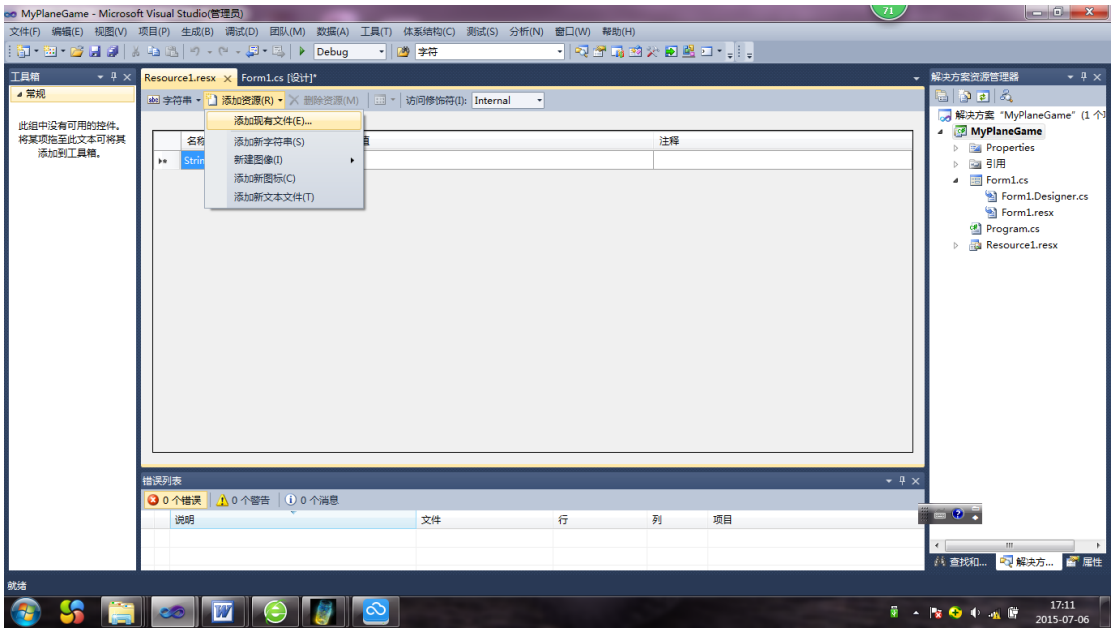

新建项目
最近的模板
已安装的模板
Visual C#
Windows
Web
Cloud
Reporting
Silverlight
WCF
Workflow
测试
其他语言
其他项目类型
数据库
建模项目
联机模板
.NET Framework 4
排序依据: 默认值
搜索 已安装的模板
Windows 窗体应用程序 Visual C#
WPF 应用程序 Visual C#
控制台应用程序 Visual C#
ASP.NET Web 应用程序 Visual C#
类库 Visual C#
MvcWebApplicationProjectTemplate.cs Visual C#
Silverlight 应用程序 Visual C#
Silverlight 类库 Visual C#
类型: Visual C#
用于创建具有 Windows 窗体用户界面的应用程序的项目
名称(N): MyPlaneGame
位置(L): D:\项目\
浏览(B)...
解决方案(S): 创建新解决方案
解决方案名称(M): MyPlaneGame
为解决方案创建目录(D)
添加到源代码管理(U)
确定
取消

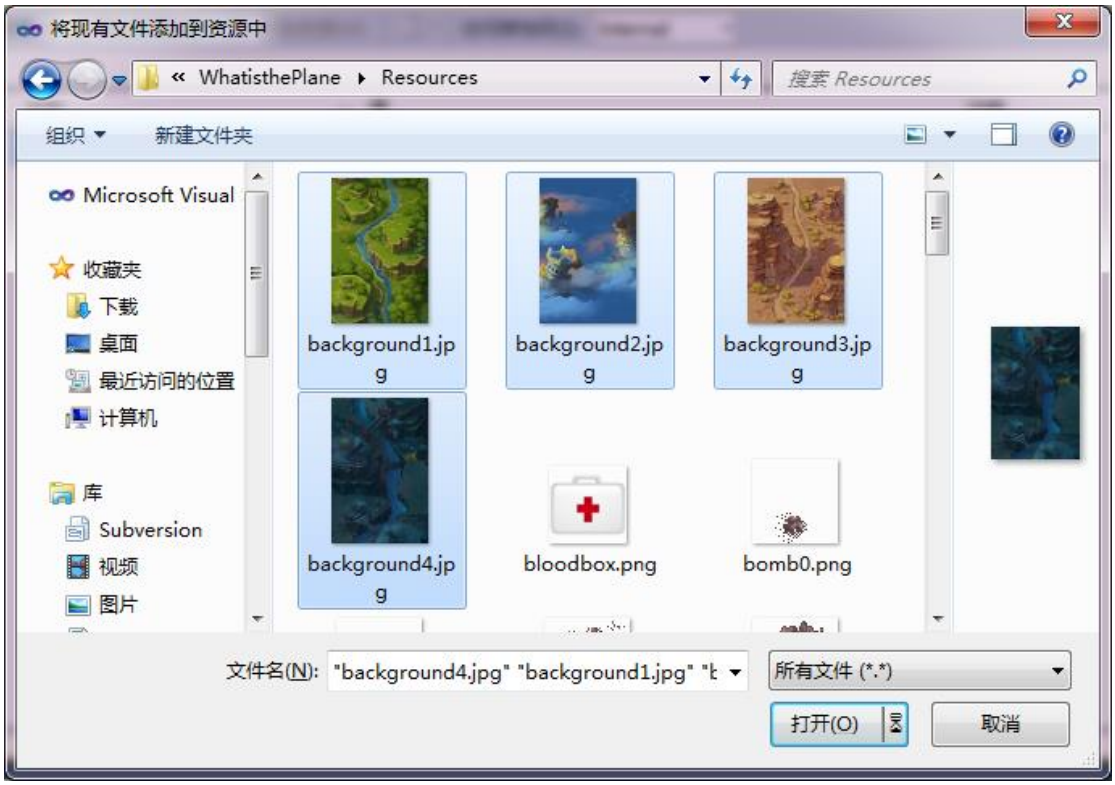

MyPlaneGame - Microsoft Visual Studio(管理员)
Form1.cs [设计]*
Form1
解决方案资源管理器
MyPlaneGame
Properties
引用
Form1.cs
Form1.Designer.cs
Form1.resx
Program.cs
错误列表

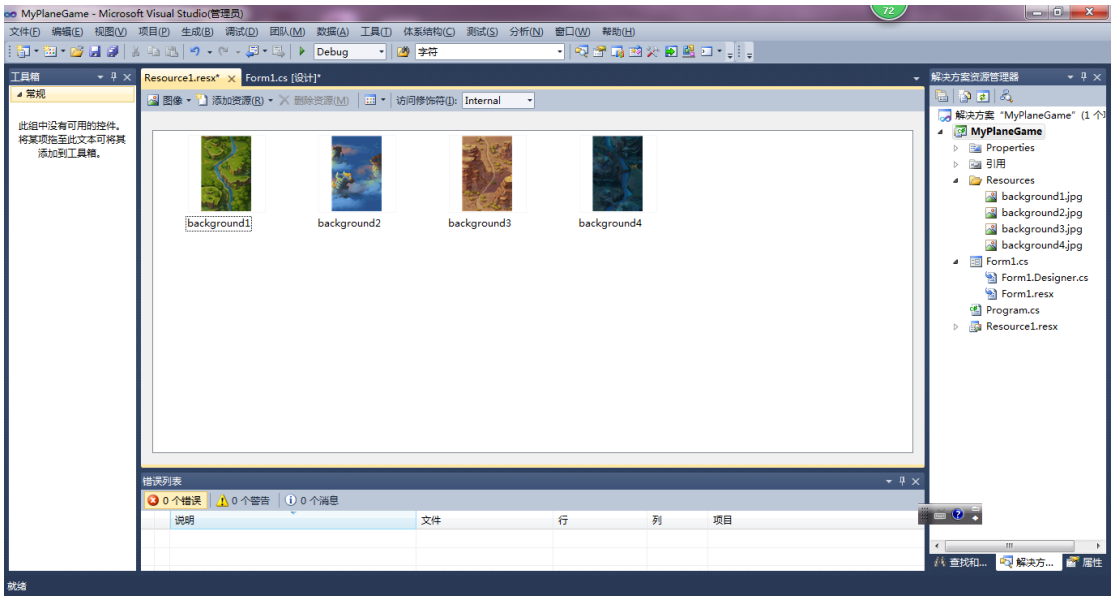
(3)添加背景图片资源


MyPlaneGame - Microsoft Visual Studio(管理员)
Form1.cs [设计]*
Form1
新建项(W)... Ctrl+Shift+A
现有项(G)... Shift+Alt+A
新建文件夹(D)
添加(D)
添加引用(F)...
添加服务引用(S)...
查看类图(V)
设为启动项目(J)
调试(G)
将解决方案添加到源代码管理(A)...
剪切(T) Ctrl+X
移除(V) Del
重命名(M)
卸载项目(L)
在 Windows 资源管理器中打开文件夹(X)
属性(R) Alt+Enter
错误列表
0 个错误
0 个警告
0 个消息
说明
文件
行
列
就绪
17:10
2015-07-06


添加新项 - MyPlaneGame
已安装的模板
Visual C# 项
Web
Windows Forms
WPF
常规
代码
数据
Reporting
Workflow
联机模板
排序依据: 默认值
搜索 已安装的模板
调试器可视化工具 Visual C# 项
图标文件 Visual C# 项
位图文件 Visual C# 项
文本模板 Visual C# 项
文本文件 Visual C# 项
样式表 Visual C# 项
已预处理的文本模板 Visual C# 项
应用程序配置文件 Visual C# 项
应用程序清单文件 Visual C# 项
资源文件 Visual C# 项
自定义控件 Visual C# 项
资源文件
类型: Visual C# 项
用于存储资源的文件
名称(N): Resource1.resx
添加(A)
取消


MyPlaneGame - Microsoft Visual Studio(管理员)
添加现有文件(E)...
添加新字符串(S)
新建图像(I)
添加新图标(C)
添加新文本文件(T)
MyPlaneGame
Properties
引用
Form1.cs
Form1.Designer.cs
Form1.resx
Program.cs
Resource1.resx


将现有文件添加到资源中
WhatisthePlane ▸ Resources
搜索 Resources
组织
新建文件夹
Microsoft Visual
收藏夹
下载
桌面
最近访问的位置
计算机
库
Subversion
视频
图片
background1.jpg
background2.jpg
background3.jpg
background4.jpg
bloodbox.png
bomb0.png
文件名(N): "background4.jpg" "background1.jpg" "b
所有文件 (*.*)
打开(O)
取消



(4)定义变量

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MyPlaneGame
{
    public partial class Form1 : Form
    {
        const int planeOffset = 2;//设置每次定时器触发时图片发生偏移的速度
        private int pix_x = 0;//背景图片移动起始坐标
        private int pix_y = 0;

        private Image[] bgrounds;//设置多张图片，每次运行程序随机产生背景图片
        int Index=0;//背景图片索引
```

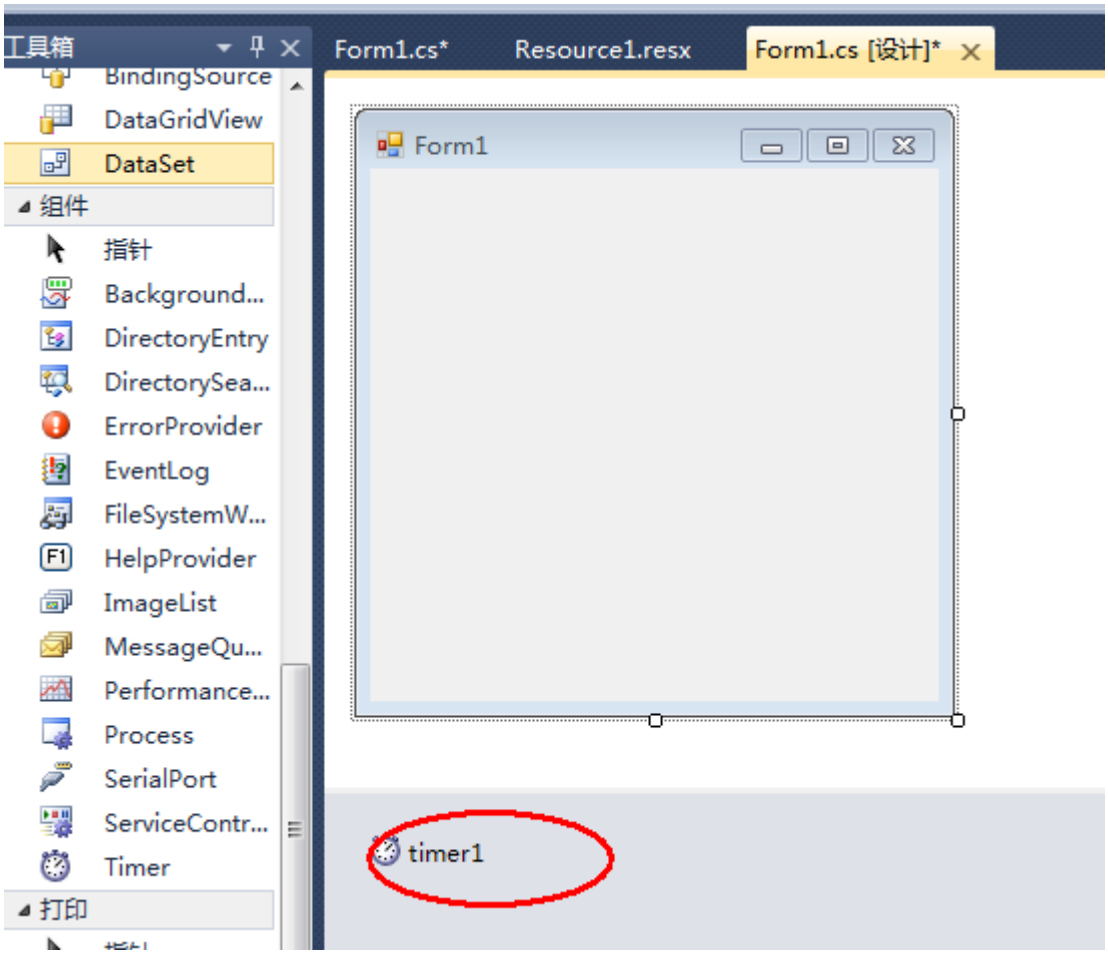
(5)初始化背景

```
  //随机生成背景图片
 public void InitBackground()
 {
     bgrounds = new Image[4];
     Index = new Random().Next(0, 4);//产生0-3的随机数，表示不同的背景
     bgrounds[0] = Resource1.background1;//从资源获取图片
     bgrounds[1] = Resource1.background2;
     bgrounds[2] = Resource1.background3;
     bgrounds[3] = Resource1.background4;
 }
```

（6）建立背景移动函数

```
//按照图片的大小设定图片，并通过定时位置让图片发生偏移。防止有空白，两张图片同
   public void BackMove(Graphics e)
{
    pix_y += planeOffset;
    if (pix_y > 630)
    {
        pix_y = 0;
    }
    e.DrawImage(bgrounds[Index], pix_x, pix_y, 420, 630);
    e.DrawImage(bgrounds[Index], pix_x, pix_y - 630, 420, 630);
}
```

(7)增加定时器



(8)设置定时器事件

```
private void timer1_Tick(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();
    BackMove(g);
}
```

(9)其他事件

```
public Form1()
{
    InitializeComponent();
    this.Size = new Size(420, 630);//让窗体与图片一样大
    this.DoubleBuffered = true;
}

private void Form1_Load(object sender, EventArgs e)
{
    InitBackground();//初始化背景
}
```

提升代码质量:

以上代码仅仅实现了背景移动，但代码质量不够优化。就相当于没学过建筑的人也能造房子，但是只能建造简单的房子，如果构建高楼大厦还是必须学习建筑学的。

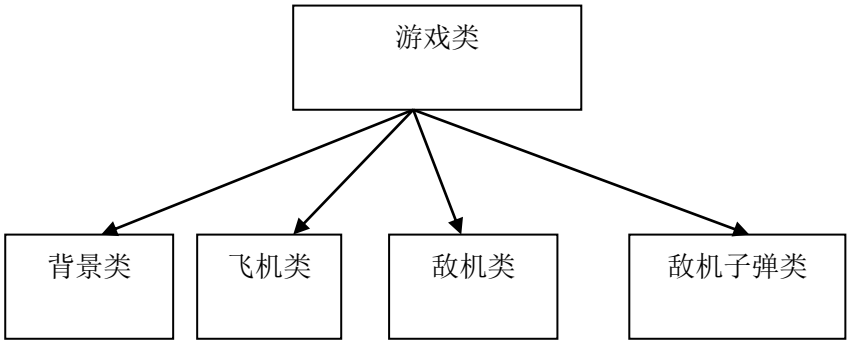
使用面向对象的思想来改造代码是软件质量提高的必备技能。

分析：

在此游戏场景中有如下对象：

背景，飞机、敌机、敌机子弹等

将其作为一个个类来设计，可以让程序结构更清晰，像搭积木一样实现一个个对象，最终完成对游戏的设计



背景类

```
class Background
{
    private Image[] bgrounds;
    private int pix_x = 0;
    private int pix_y = 0;
    private int index;

    public int Pix_x
    {
        get { return pix_x; }
        set { pix_x = value; }
    }
public int Pix_y
{
    get { return pix_y; }
    set { pix_y = value; }
}
public int Index
{
    get { return index; }
    set { index = value; }
}
const int imageHeight = 835;
const int planeOffset = 2;
```

```
public Background()
{
    bgrounds = new Image[4];
    Index = new Random().Next(0, 4);
    bgrounds[0] = MyResource.background1;
    bgrounds[1] = MyResource.background2;
    bgrounds[2] = MyResource.background3;
    bgrounds[3] = MyResource.background4;
}

public void Draw(Graphics e)
{
    e.DrawImage(bgrounds[Index], new Point(Pix_x, Pix_y - imageHeight));
    e.DrawImage(bgrounds[Index], new Point(Pix_x, Pix_y));
}
public void Move()
{
    Pix_y += planeOffset;
    if (Pix_y > imageHeight)
    {
        Pix_y = 0;
    }
}
```

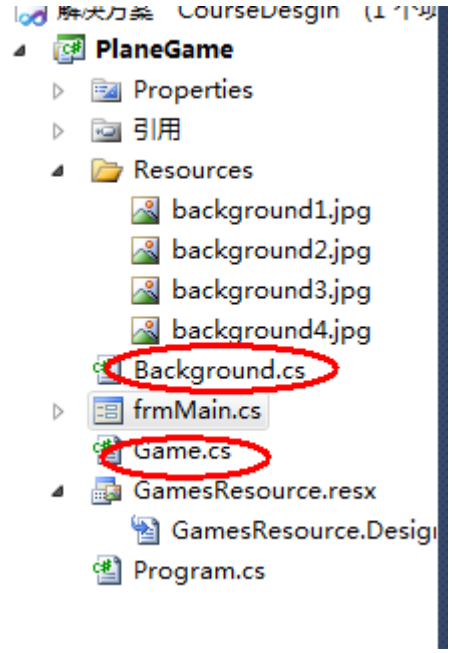
游戏类：

```
class Game
{
    Background bg = new Background();
    public void Move()
    {
        bg.Move();
    }
    public void Draw(Graphics e)
    {
        bg.Draw(e);
    }
}
```

FORM 窗体类：

```
namespace PlaneGame
{
    public partial class FormMain : Form
    {   Game game = new Game();
        public FormMain()
        {
            InitializeComponent();
            this.Size = new Size(420, 630);
            this.DoubleBuffered = true;
        }
        private void gameTimer_Tick(object sender, EventArgs e)
        {
            game.Move();//改变相关数据值
            this.Invalidate();//刷新重绘图片，导致调用FormMain_Paint，使得重绘数据改
        }
        private void FormMain_Paint(object sender, PaintEventArgs e)
        {
            game.Draw(e.Graphics);
        }
    }
}
```

解决方案框架



逻辑是否比原来代码更清晰呢？
后续功能也可如此

6 绘制飞机及生命值

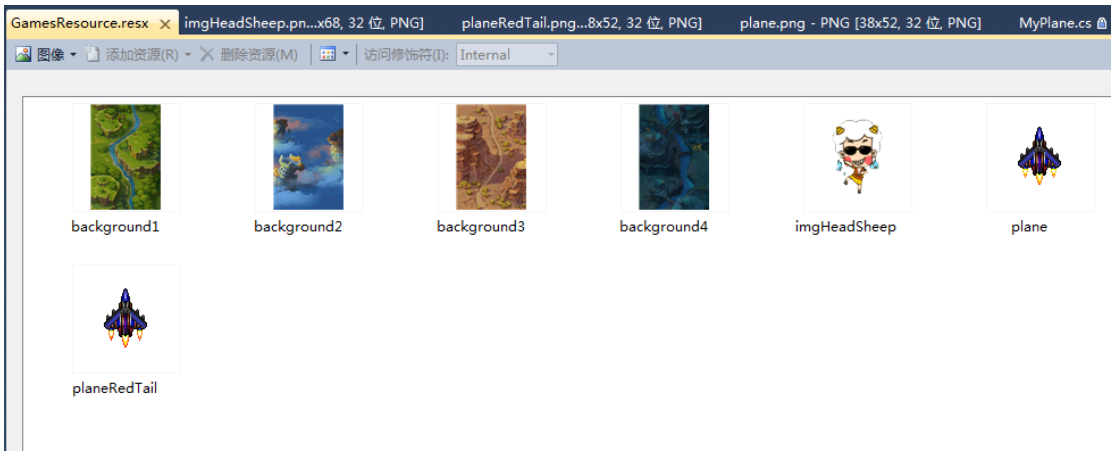
效果如下：

关键技术：Graphic 中的图片绘制函数

```
DrawImage(图片, new Point(x,y));
```

步骤：

（1）添加飞机和头像图片到资源



（2）定义图片对象，从资源获取对象

```
Image myplane;//飞机图片?

Image headimage;//用户头像?

 myplane = GamesResource.plane;

 headimage = GamesResource.imgHeadSheep;
```

（3）绘制飞机

```
g.DrawImage(myplane, new Point(200, 300));
g.DrawImage(headimage, new Point(10, 10));//画头像

g.DrawRectangle(new Pen(Color.Black, 1), 10, 90, 102, 10);//画血量条
g.FillRectangle(Brushes.Red, 11, 91, Health, 9);//填充血量条

g.DrawRectangle(new Pen(Color.Blue, 1), 10, 110, 102, 10);//画完整血量条
g.FillRectangle(Brushes.Green, 11, 111, 100, 9);//填充完整血量条

//显示字符串
g.DrawString("Player: 肖斌", new Font("宋体", 9, FontStyle.Bold), Brushes.Yellow, new Point(10, 130));
g.DrawString("Score: 100", new Font("宋体", 9, FontStyle.Bold), Brushes.Yellow, new Point(10, 150));
```

提示：简单的方法是将这些代码放到form窗体内，好的设计是建立一个飞机类，将这些属性和方法封装在类中。

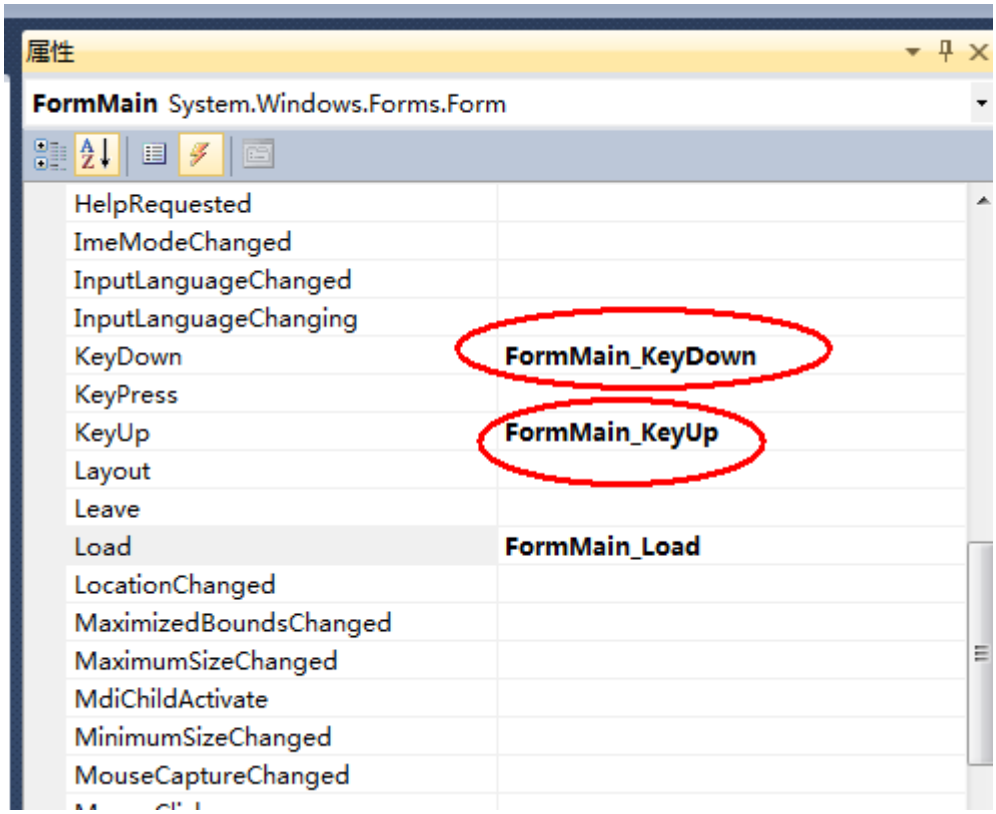
7、移动飞机

按asdx键盘可以上下左右移动飞机，其实质是捕获键盘事件，判断键盘按那些键，如果按了ASDX，就改变飞机坐标位置，重新绘制飞机

DrawImage(图片, new Point(x,y));

将X,Y坐标变化一下

(1)　设置form窗体的按钮事件



(2)在form窗体中可以看到

```
//class three
private void FormMain_KeyDown(object sender, KeyEventArgs e)
{
  e.KeyCode
}
//class three
private void FormMain_KeyUp(object sender, KeyEventArgs e)
{
   e.KeyCode;
}
```

其中e.keyCode可以判断按那个键

```
if( e.KeyCode ==Keys.A)
```

(2)　编写改变飞机坐标的代码

可以将飞机坐标x,y设置为窗体级变量，然后在键盘事件中修改X,Y坐标，比如

```
private void FormMain_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.A) X = X - 2;
}
```

也可将x,y坐标封装在飞机类中，改变该类的x,y属性

8 改进的键盘响应

如果在FORM Key_Down事件中

```
private void FormMain_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.A) X = X - 2;
}
```

编写类似代码似乎没什么问题，但是当用户既要移动飞机，又要发射子弹，涉及一个时刻多个响应时，把所有键的控制放在此处，效果不理想，可能有迟钝效果,因此需改进键盘处理，构造一个专门的键盘处理类.

(1)　构造键盘处理类

```
class Mykeyboard
{   //定义按键数组
    private static List<Keys> listkey = new List<Keys>();
    public static void KeyUp(Keys key)
    {
        listkey.Remove(key);//有键松开，将键从数组中移出
    }
    public static void KeyDown(Keys key)
    {   //关键代码，思考为什么?
        if (!listkey.Contains(key))//如果有键按下，数组没包含该键,则加入数组
        {
            listkey.Add(key);
        }
    }
    public static bool IsKeyDown(Keys key)
    {
        return listkey.Contains(key);
    }
}
```

(2)　此时飞机类代码如下：

```
class MyPlane//飞机类
{
    public Image myplane;//飞机图片
    Image redplane;//飞机图片
    Image notredplane;//飞机图片
    Image headimage;//用户头像
    public int Health = 100;//飞机血量
    public int planex = 175;//飞机坐标
    public int planey = 500;//飞机坐标

    //初始化图片
    public MyPlane()
    {
        redplane = GamesResource.planeRedTail;
        notredplane = GamesResource.plane;
        myplane = redplane;
        headimage = GamesResource.imgHeadSheep;
    }
public void Draw(Graphics g)
{
    if (Health > 0)
    {
        g.DrawImage(myplane, new Point(planex, planey));
    }
    else if (Health < 0)
    {
        g.DrawImage(myplane, new Point(0, -500));
    }
    g.DrawImage(headimage, new Point(10, 10));//画头像
    g.DrawRectangle(new Pen(Color.Black, 1), 10, 90, 102, 10);//画血量条
    g.FillRectangle(Brushes.Red, 11, 91, Health, 9);//填充血量条
    g.DrawRectangle(new Pen(Color.Blue, 1), 10, 110, 102, 10);//画完整血量条
    g.FillRectangle(Brushes.Green, 11, 111, 100, 9);//填充完整血量条
    //显示字符串
    g.DrawString("Player: 肖斌", new Font("宋体", 9, FontStyle.Bold), Brushes.Yellow, ne
    g.DrawString("Score: 100", new Font("宋体", 9, FontStyle.Bold), Brushes.Yellow, new
}
```

```
//改变我的飞机的坐标
public void Move()
{
    myplane = myplane == redplane ? notredplane : redplane;
    if (Mykeyboard.IsKeyDown(Keys.A))
    {
        myplane = GamesResource.planeLeft;

        if (planex < 0)
        {
            planex = 0;
        }

        planex -= 13;

    }

 if (Mykeyboard.IsKeyDown(Keys.D))
 {
     myplane = GamesResource.planeRight;

     if (planex > 360)
     {
         planex = 360;
     }

     planex += 13;

 }
```

```
        if (Mykeyboard.IsKeyDown(Keys.W))
        {  if (planey < 0)
            {
                planey = 0;
            }

            planey -= 13;

        }
        if (Mykeyboard.IsKeyDown(Keys.S))
        {  if (planey > 535)
            {
                planey = 535;
            }

            planey += 13;

        }
    }
```

(3) 游戏类的代码

```
class Game
{
    Background background = new Background();
    MyPlane myPlane = new MyPlane();//class two//飞机

    public void Move()
    {
        background.Move();//背景移动
        myPlane.Move();//飞机移动

    }
    public void Draw(Graphics e)
    {     background.Draw(e);

        myPlane.Draw(e);//class two

    }
}
```

(4) 游戏窗体form

```
public partial class FormMain : Form
{   Game game = new Game();
    public FormMain()
    {   InitializeComponent();
        this.Size = new Size(420, 630);
        this.DoubleBuffered = true;
    }
    private void gameTimer_Tick(object sender, EventArgs e)
    {  game.Move();//改变相关数据值
        this.Invalidate();//刷新重绘图片，导致调用FormMain_Paint，使得重绘数据改变的图
    }
    private void FormMain_Paint(object sender, PaintEventArgs e)
    { game.Draw(e.Graphics);     }
    private void FormMain_KeyDown(object sender, KeyEventArgs e)
    {Mykeyboard.KeyDown(e.KeyCode);}
    private void FormMain_KeyUp(object sender, KeyEventArgs e)
    {
        Mykeyboard.KeyUp(e.KeyCode);
    }
}
```

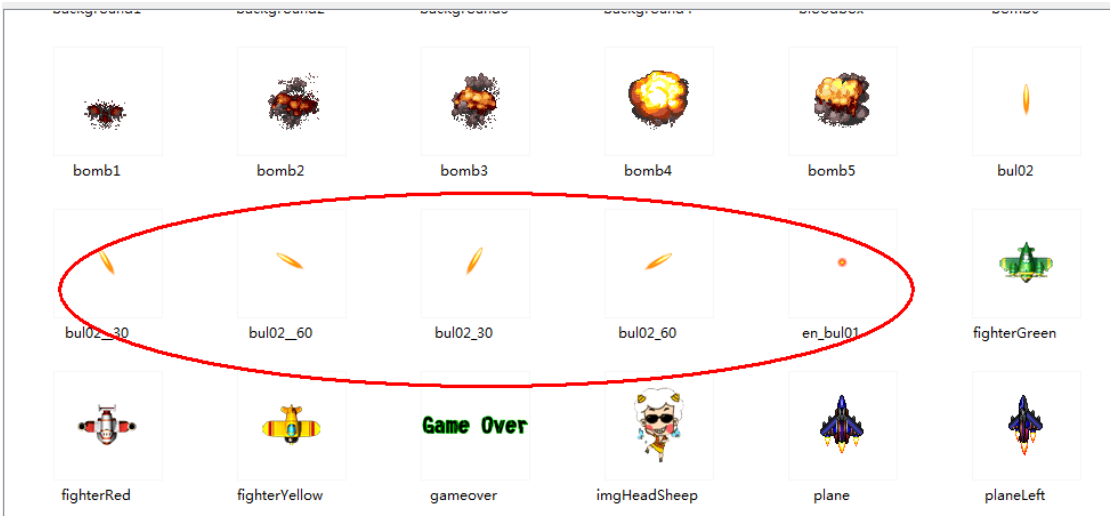
9 发射子弹

效果如下：

可以看到子弹有单发子弹，和多方向多弹两种

实现步骤:

(1)　在资源中加入子弹图片



(2)　建立一个子弹类，设计它的属性与方法

属性：x坐标，y坐标,子弹角度,移动距离

方法：绘制draw,和移动子弹MOVE

```
class Bullet//子弹类
{
    static double pi = Math.PI;
    public int Bx;//子弹X坐标
    public int By;//子弹Y坐标
    public int Angle;//子弹角度
    public int Distance;//子弹一次移动距离
    public Image bulletImage;
```

```
//构造函数
public Bullet(int bx,int by,int angle,int distance)
{
    Bx = bx;
    By = by;
    Angle = angle;
    Distance = distance;
    //控制Angle实现变弹！！！！！！！！！！！！
    switch (Angle)
    {
        case 90:
            bulletImage = MyResource.bul02;//直角子弹
            By -= 17;
            break;
        case 60:
            bulletImage = MyResource.bul02_30;//角度为30的子弹
            Bx += 1;
            By -= 17;
            break;
        case -60:
            bulletImage = MyResource.bul02__30;//角度为60的子弹
            Bx -= 20;

               By -= 17;
               break;
           case 30:
               bulletImage = MyResource.bul02_60;//角度为60的子弹
               Bx += 12;
               By -= 12;
               break;
           case -30:
               bulletImage = MyResource.bul02__60;//角度为60的子弹
               Bx -= 35;
               By -= 12;
               break;
           default:
               break;
       }
   }
public void Draw(Graphics e)
{

    e.DrawImage(bulletImage, new Point(Bx,By));

}
```

```
    public void Move()
    {
        if(角度==90)子弹直线移动
        else if(角度0-90)
         {计算移动距离}
        else (角度<0)
              {计算移动距离}

    }

}
```

(3)　修改Game类，定义子弹对象

```
class Game
{
    Background background = new Background();
    MyPlane myPlane = new MyPlane();//class two//飞机
    List<Bullet> lstBullet = new List<Bullet>();//子弹数组，因为子弹有多个需数组
    bool IsGetsGun = false;//是否得到多方位枪
```

（2）在GAME类中编写产生子弹方法ProduceBullet()

```
//产生子弹
  public void ProduceBullet()
{
    if (Mykeyboard.IsKeyDown(Keys.J))//如果用户按键是J
    {
        if (!IsGetsGun)//如果没有得到枪，产生普通子弹
        {
            lstBullet.Add(new Bullet(myPlane.planex + 12, myPlane.planey - 10, 90, -20));
        }
        else//得到枪产生多方向子弹,怎么编写
        {
          ..........
        }
    }
}
```

(3)在GAME类中编写子弹移动方法MoveBullet(Graphics e)

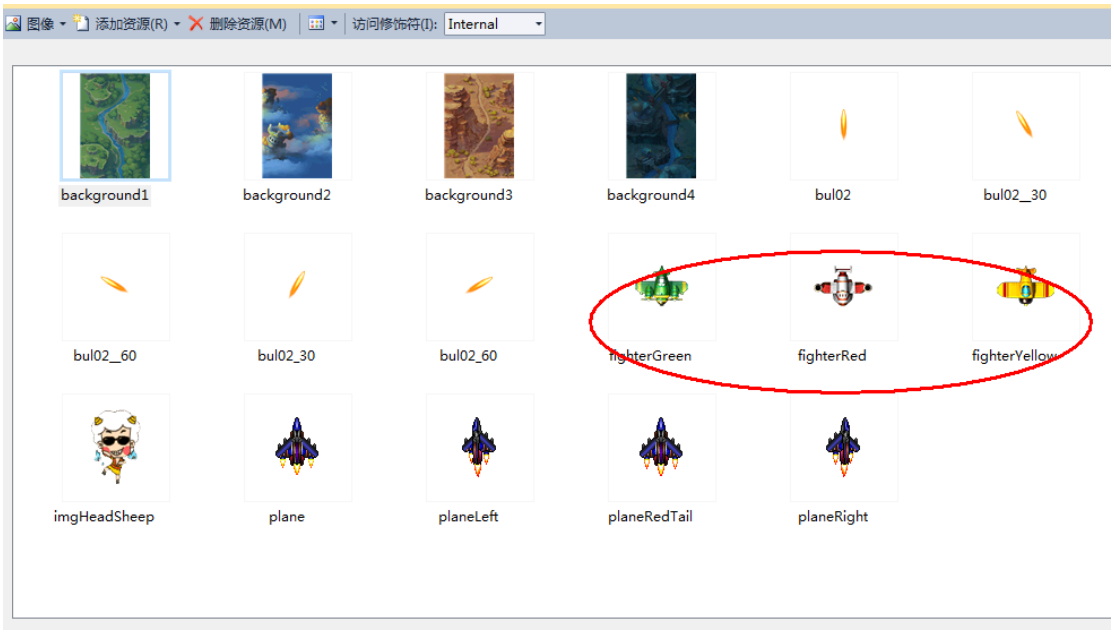
```
//移动子弹
public void MoveBullet(Graphics e)
{
    //遍历子弹
    for (int i = 0; i < lstBullet.Count; i++)
    {
        lstBullet[i].Draw(e);
        lstBullet[i].Move();
        if (lstBullet[i].By < 0)//如果子弹移出到屏幕外则清除子弹
        {
            lstBullet.Remove(lstBullet[i]);
        }
    }
}
```

(4)修改GAME类的方法

```
public void Move()
{
    background.Move();//背景移动
    myPlane.Move();//飞机移动
    ProduceBullet();//产生子弹
}

public void Draw(Graphics e)
{   background.Draw(e);
    myPlane.Draw(e);//class two
    MoveBullet(e);//移动子弹
}
```

10 显示移动敌机

(1) 添加敌机资源



(2) 模仿我方飞机，构建敌机类

```
class Enemy
    {
        private int ex;// 敌机X坐标
        private int ey = 0;
         ...


        public void Drawenemy(Graphics g)
        {

        }

        public void EnemyMove()
        {

        }

    }
```

(3) 在 GAME 类中定义敌机数组，因为敌机也有多个

```
class Game
{

    Background background = new Background();
    MyPlane myPlane = new MyPlane();//class two//飞机
    List<Bullet> lstBullet = new List<Bullet>();//子弹数组，因为子弹有多个需数组
    bool IsGetsGun = false;//是否得到多方位枪
    List<Enemy> lstEnemy = new List<Enemy>();//定义敌机数组
```

(4)在game类中增加显示敌机的方法ProduceEnemy()

```
public void ProduceEnemy()
        {
            Random ran = new Random();
            if (ran.Next(15) == 0)//当随机数为0的时候产生敌机,保证一段时间产生敌机
            {
                 随机产生一种颜色的敌机
                 将该敌机加入敌机数组
            }
        }
```

(5)在GAME 类中增加敌机移动方法MoveEnemy(Graphics e)

类似子弹移动

```
public void MoveEnemy(Graphics e)
        {
            遍历敌机数组
             {
                  绘制敌机
                  敌机移动
                  如果敌机坐标超过屏幕下标，移除敌机
             }
        }
```

(5) 最终 GAME 类效果

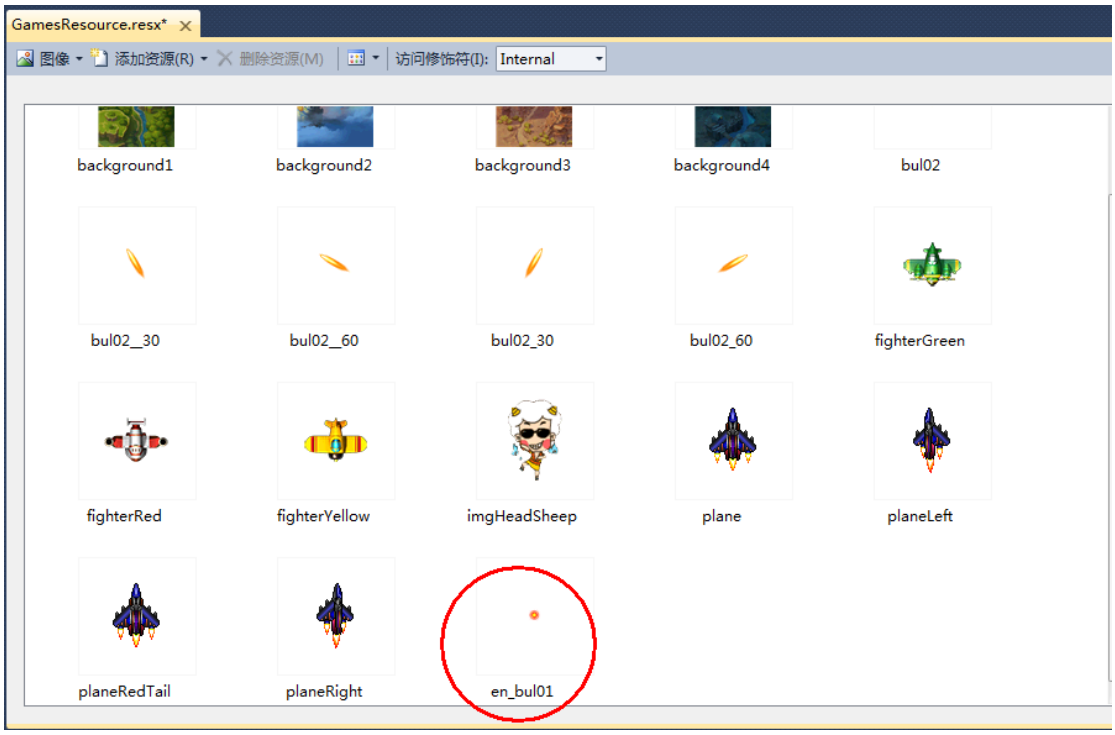
```
class Game
{   Background background = new Background();
    MyPlane myPlane = new MyPlane();//class two//飞机
    List<Bullet> lstBullet = new List<Bullet>();//子弹数组，因为子弹有多个需数组
    bool IsGetsGun = false;//是否得到多方位枪
    List<Enemy> lstEnemy = new List<Enemy>();//定义敌机数组
    public void Move()
    {
        background.Move();//背景移动
        myPlane.Move();//飞机移动
        ProduceBullet();//产生子弹
        ProduceEnemy();//产生敌机
    }
    public void Draw(Graphics e)
    {   background.Draw(e);
        myPlane.Draw(e);//class two
        MoveBullet(e);//移动子弹
        MoveEnemy(e);//移动敌机
    }

    public void ProduceBullet()...
    //移动子弹
    public void MoveBullet(Graphics e)...
    //产生敌机
    public void ProduceEnemy()...
    //移动敌机
    public void MoveEnemy(Graphics e)...

}
```

运行效果

11、敌方发射子弹

和我方发射子弹类似

(1) 添加敌方子弹资源



(2)构建敌方子弹类EnemyBullet

子弹除了有坐标属性，还应该有玩家飞机的坐标，子弹应该在发射时瞄准玩家飞机飞行

```
class EnemyBullet
  {
      public int ebx;
      public int eby;
      ...

      public Image ebulletImage;
      /// <summary>
      /// 构造函数
      /// </summary>
      /// <param name="bx">子弹X坐标</param>
      /// <param name="by">子弹Y坐标</param>
      /// <param name="distance">子弹一次移动距离</param>
      /// <param name="playerx">玩家X坐标</param>
      /// <param name="playery">玩家y坐标</param>
      public EnemyBullet(int bx,int by,int distance,int playerx,int playery)
      {
          ebx = bx;
          eby = by;
          ....
          ebulletImage = MyResource.en_bul01;
      }
      //绘制子弹
      public void DrawEnemyBullet(Graphics e)
      {
         ....
      }
      //改变子弹坐标
      public void EBulletMove()
      {  ....
      }

  }
```

(2)　在game类添加敌机子弹数组，因为子弹有多个

```
class Game
{   Background background = new Background();
    MyPlane myPlane = new MyPlane();//class two//飞机
    List<Bullet> lstBullet = new List<Bullet>();//子弹数组，因为子弹有多个需数组
    bool IsGetsGun = false;//是否得到多方位枪
    List<Enemy> lstEnemy = new List<Enemy>();//定义敌机数组
    List<EnemyBullet> lstEnemyBullet = new List<EnemyBullet>();//定义敌机子弹数组
```

(3)在GAME类中添加产生子弹方法

```
public void ProduceEnemyBullet()
        {
            Random ran = new Random();
            遍历敌机数组
             {
                 if (ran.Next(0, 8) == 0)//使用随机数设定某个时刻产生
                 {
                     实例化一个敌机子弹对象
                     将子弹添加到敌机子弹数组
                 }
             }
        }
```

（4）　在 game 类中添加移动敌机子弹的方法

```
public void MoveEnemyBullet(Graphics e)
        {
            遍历敌机子弹数组
            {
                绘制每个子弹
                移动每个子弹

                if (子弹的y坐标移动到屏幕外面)
                {
                    从敌机子弹数组移除该子弹
                }
            }
        }
```

（5） 在 Game 类的 move 和 draw 分别添加上述两方法

```
public void Move()
{
    background.Move();//背景移动
    myPlane.Move();//飞机移动
    ProduceBullet();//产生子弹
    ProduceEnemy();//产生敌机
    ProduceEnemyBullet();//产生敌机子弹
}
public void Draw(Graphics e)
{   background.Draw(e);
    myPlane.Draw(e);//class two
    MoveBullet(e);//移动子弹
    MoveEnemy(e);//移动敌机
    MoveEnemyBullet(e);//移动敌机子弹
}
```

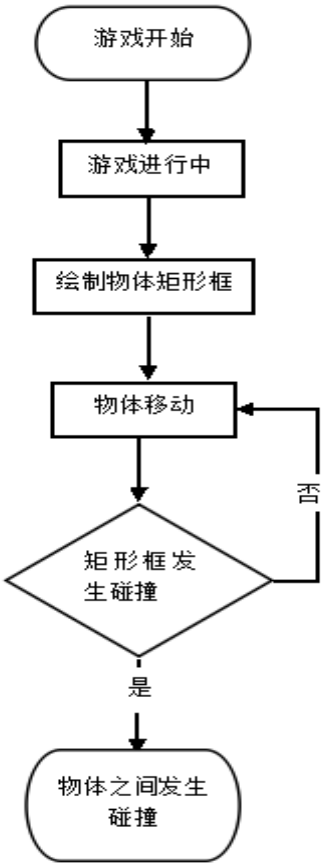
(6) 整个 game 类框架如下:

```
class Game
{   Background background = new Background();
    MyPlane myPlane = new MyPlane();//class two//飞机
    List<Bullet> lstBullet = new List<Bullet>();//子弹数组，因为子弹有多个需数组
    bool IsGetsGun = false;//是否得到多方位枪
    List<Enemy> lstEnemy = new List<Enemy>();//定义敌机数组
    List<EnemyBullet> lstEnemyBullet = new List<EnemyBullet>();//定义敌机子弹数组
    public void Move()...
    public void Draw(Graphics e)...
    public void ProduceBullet()... //产生子弹
    public void MoveBullet(Graphics e)... //移动子弹
    public void ProduceEnemy()...//产生敌机
    public void MoveEnemy(Graphics e)... //移动敌机
    public void ProduceEnemyBullet()... //产生敌机子弹
    public void MoveEnemyBullet(Graphics e)...//移动敌机子弹

}
```

运行效果

12、敌机消亡

在我方发射的子弹击中敌机时，敌机应该消亡。这里涉及到碰撞检测
在游戏中，碰撞检测是检测子弹与飞机的碰撞、飞机与道具的碰撞、是否有效的重要方法。
其流程图如下：



图碰撞检测流程图

在 GDI 中有一个函数可以帮我们实现碰撞检测

Rectangle.IntersectsWith 方法

.NET Framework 4.6 and 4.5 | 其他版本

确定此矩形是否与 *rect* 相交。

rect

类型：System.Drawing.Rectangle
要测试的矩形。

返回值

类型：System.Boolean

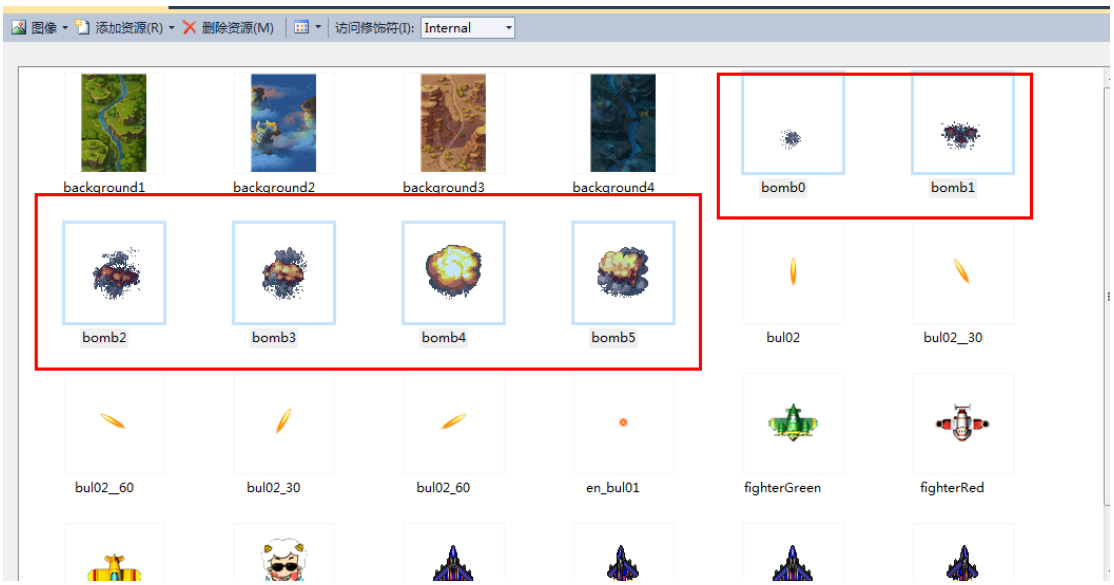
如果有重叠，此方法将返回 **true**；否则将返回 **false**。

（1） 思想

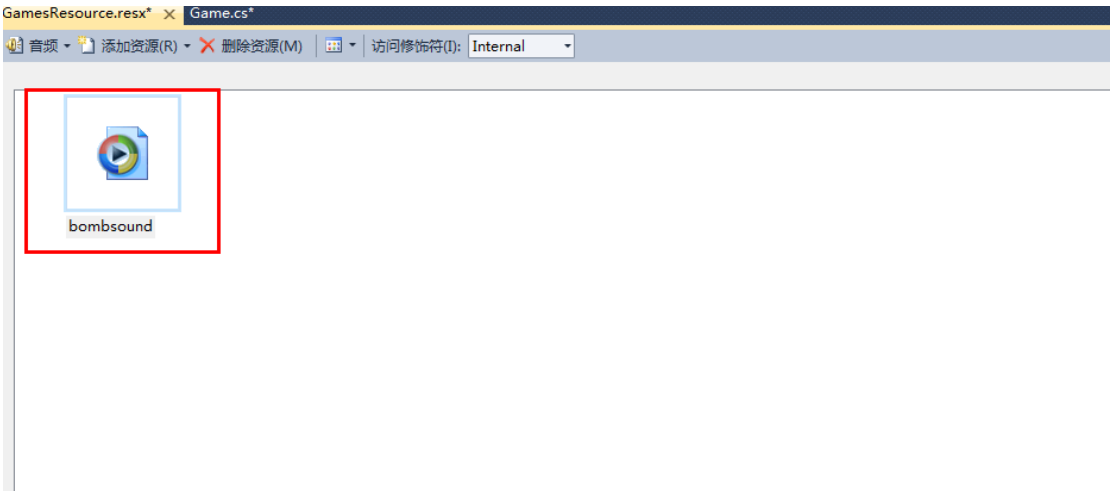
获取我方子弹的位置，将该位置转换为矩形。获取敌方飞机的位置，将该位置转换为矩形，遍历每个子弹是否击中敌机，调用IntersectsWith函数检测两个矩形是否相交，相交则碰撞，显示炸弹效果。

(3) 构造爆炸效果类Bomb

添加爆炸资源图片



爆炸声音



（4）构造爆炸效果类Bomb

```
class Bomb
{
    SoundPlayer sp = new SoundPlayer();
    private int x;
    public int X
    {
        get { return x; }
        set { x = value; }
    }
    private int y;
    public int Y
    {
        get { return y; }
        set { y = value; }
    }
```

```
Point loc;

public Point Loc
{
    get { return loc; }
    set { loc = value; }
}

public int index = 0;

public Image[] bombimage;
```

```
public Bomb()
{    bombimage = new Image[6];
     loc = new Point(x, y);
     bombimage[0] = GamesResource.bomb0;
     bombimage[1] = GamesResource.bomb1;
     bombimage[2] = GamesResource.bomb2;
     bombimage[3] = GamesResource.bomb3;
     bombimage[4] = GamesResource.bomb4;
     bombimage[5] = GamesResource.bomb5;
     sp.Stream = GamesResource.bombsound;
}
public void draw(Graphics g)
{       g.DrawImage(bombimage[0], loc);
        g.DrawImage(bombimage[1], loc);
        g.DrawImage(bombimage[2], loc);
        g.DrawImage(bombimage[3], loc);
        g.DrawImage(bombimage[4], loc);
        g.DrawImage(bombimage[5], loc);
}
public void bombplay()
{ sp.Play(); }
```

(5)在GAME类中添加碰撞检测函数EnemyDisappearAndBomb()

```
public void EnemyDisappearAndBomb(Graphics g)
{   //遍历子弹数组
    for (int i = 0; i <lstBullet.Count; i++)
    {  //获取每个子弹的位置将其转换为矩形
        Rectangle bues = new Rectangle(lstBullet[i].Bx, lstBullet[i].By, lstBullet[i].bulletImage.Width, lstBullet[i].bulletImage.Height);
        //遍历每个敌机
        for (int j = 0; j <lstEnemy.Count; j++)
        {
            //获取每个敌机的位置转换为矩形
            Rectangle emes = new Rectangle(lstEnemy[j].Loc.X, lstEnemy[j].Loc.Y, lstEnemy[j].enemyimage.Width, lstEnemy[j].enemyimage.Height);
            if (emes.IntersectsWith(bues))//碰撞检测
            {
                Bomb bomb = new Bomb();
                bomb.Loc = new Point(lstEnemy[j].Loc.X, lstEnemy[j].Loc.Y);

                lstBullet.Remove(lstBullet[i]);
                lstEnemy.Remove(lstEnemy[j]);
                bomb.draw(g);
                //爆炸声
                bomb.bombplay();
            }
        }
    }
```

（6） 在draw方法调用该碰撞函数

```
public void Draw(Graphics e)
{   background.Draw(e);
    myPlane.Draw(e);//class two
    MoveBullet(e);//移动子弹
    MoveEnemy(e);//移动敌机
    MoveEnemyBullet(e);//移动敌机子弹
    EnemyDisappearAndBomb(e);//碰撞检测，检测敌机是否击中爆炸
}
```

(7) game结构

```
class Game
{   Background background = new Background();
    MyPlane myPlane = new MyPlane();//class two//飞机
    List<Bullet> lstBullet = new List<Bullet>();//子弹数组，因为子弹有多个需数组
    bool IsGetsGun = false;//是否得到多方位枪
    List<Enemy> lstEnemy = new List<Enemy>();//定义敌机数组
    List<EnemyBullet> lstEnemyBullet = new List<EnemyBullet>();//定义敌机子弹数组
    public void Move()...
    public void Draw(Graphics e)...
    public void ProduceBullet()... //产生子弹
    public void MoveBullet(Graphics e)... //移动子弹
    public void ProduceEnemy()...//产生敌机
    public void MoveEnemy(Graphics e)... //移动敌机
    public void ProduceEnemyBullet()... //产生敌机子弹
    public void MoveEnemyBullet(Graphics e)...//移动敌机子弹
    //碰撞检测，如果碰撞，产生爆炸效果
    public void EnemyDisappearAndBomb(Graphics g)...

}
```