毕业设计（论文）

课题名称: 火拼俄罗斯

专 业 系: 信 息 工 程 系

班级名称: 软件 043 班

学生姓名: 李 勇

指导老师: 袁开鸿

完成日期: 2006 年 12 月 16 日


参数设置
快速帮助
呢称 111
等级 学徒 2
呢称 222
等级 学徒 3
MAP
NEXT
呢称: ttt
积分: 5
等级: 学徒
火拼俄罗斯
QQGame仿真制作
制作人:
丁鹏,李勇,阳洲,王任,蒋杰
蔡谭林,张涛
指导老师:袁开鸿
555 开始
444 开始
333 开始
222 开始
111 开始
网络发生错误
发言
呢称 333
等级 学徒 4
呢称 444
等级 学徒 5
呢称 555
等级 学徒 6
选择编组
俄罗斯道具栏
俄罗斯道具说明
从底部起减三层方块
开始
练习
退出

第一部分　引言

俄罗斯方块是一款风靡全球的电视游戏机和掌上游戏机游戏，它曾经造成的轰动与造成的经济价值可以说是游戏史上的一件大事。这款游戏最初是由苏联的游戏制作人 Alex Pajitnov 制作的，它看似简单但却变化无穷，令人上瘾。相信大多数用户都还记得为迷得茶不思饭不想的那个俄罗斯方块时代。 究其历史，俄罗斯方块最早还是出现在 PC 机上，而我国的用户都是通过红白机了解、喜欢上它的。现在联众又将重新掀起这股让人沉迷的俄罗斯方块风潮。对一般用户来说，它的规则简单，容易上手，且游戏过程变化无穷，而在"联众俄罗斯方块"中，更有一些联众网络游戏所独有的魅力——有单机作战与两人在线对战两种模式，用户可任选一种进行游戏。网络模式还增加了积分制，使用户既能感受到游戏中的乐趣，也给用户提供了一个展现自己高超技艺的场所。

关键技术：C#双缓冲绘图，C#线称的同步与互斥以及线程池，C#类设计模式—工厂设计模式，C#网络编程，C#接口，C#类的继承，方法重载，Xml 的读写，C#自定义控件，《数据结构》的矩阵、图、 队列、查找，回朔算法，Windows API 函数,DirectX 游戏编程。

第二部分　可行性分析

2.1 技术可行性

C# (C Sharp) C#（发音为 C sharp）是一种简单、现代化、面向对象、类型安全的开发语言，使程序员快速容易的为 Windows Server System 集成软件产品构建解决方案。

优点:

代码复用。用 C# 设计的组件可以很容易的转换成 Web 服务，可以以任意操作系统的任意语言从 Internet 上调用。

增强的性能。垃圾收集 (GC) 技术。不需要手工的内存管理。

增强的可靠性。 变量都已自动初始化为类型安全的。

改进的开发效率。版本控制与伸缩性支持。

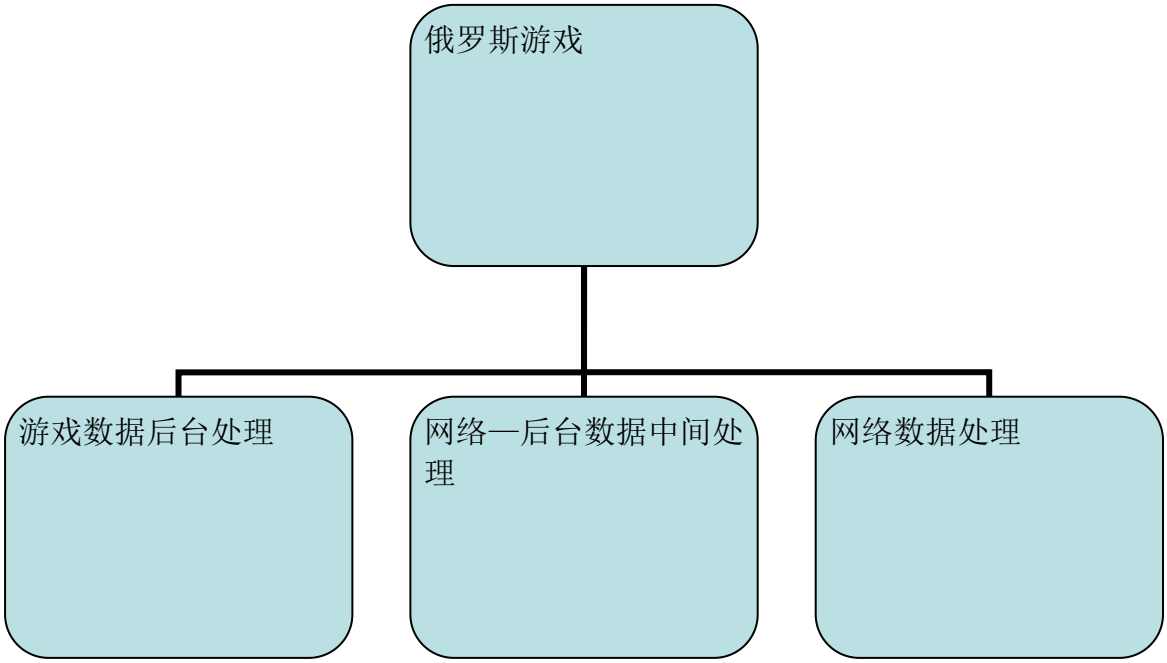
更快的市场反应。在业务流程与部署应用程序之间实现更好的映射。开发人员可以定义特定域的属性，并将它们应用于任何语言元素如类，接口等。定义之后，每个元素的属性都可以被编程访问。

广泛的交互操作性。对 COM 和 Windows-API 的天生支持。

C# 是非常成功的基于 Web 的 Consensus 3.0a 程序的关键部分，旨在允许 Microsoft 用户或小组不费力的创建，分布及管理电子调查，以获得有价值的反馈。开发此应用程序的团队使用 .NET 框架和 C# 语言创建了多线程的邮件服务。使用 C# 可以在不丢失面向对象特性的基础上降低开发与测试时间。对这个项目来说，C# 语言看

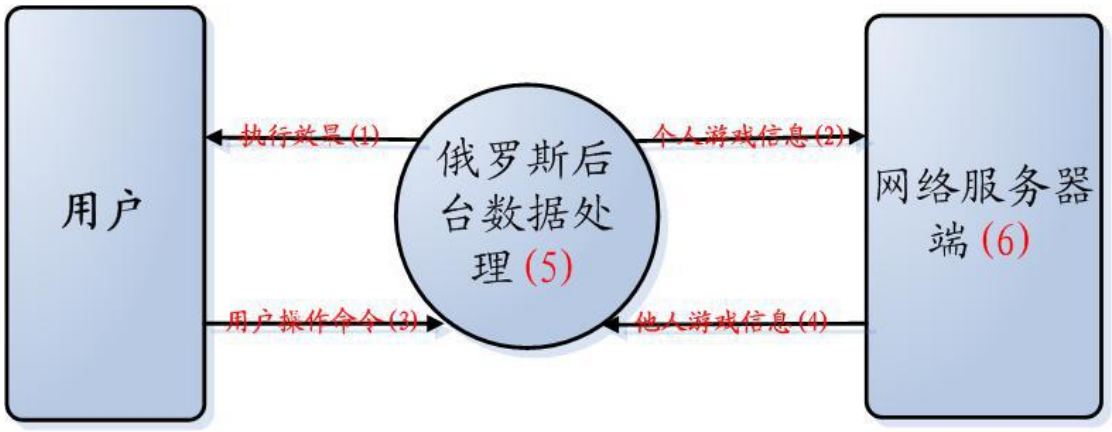
起来是最合适做应用程序设计的了；实际上，因为有了来自 .NET 框架对核心编程任务的支持，这个团队发现使用 C# 比使用 Visual Basic 更容易。

第三部分　软件需求说明



如上图所示，整个系统由 游戏数据后台处理、网络—后台数据中间处理、 网络数据处理 三个部分组成。

3.1 顶端数据流图



数据字典：

(1)数据流：执行效果

说明：表示经后台处理显示在游戏区域的数据；

(2)数据流：个人游戏信息

数据流来源：俄罗斯后台处理

(3)数据流：用户操作命令

说明：指用户从键盘输入的命令
组成：上，下，左，右…..
平均流量：10 个/s;
高峰流量：50 个/s;

(4)数据流：他人游戏信息

说明：在线用户的游戏信息
组成：主要由 网络信息数据流(详细信息请察看 数据流：网络信息)组成。
平均流量：15 条/s;
高峰流量：30 条/s;

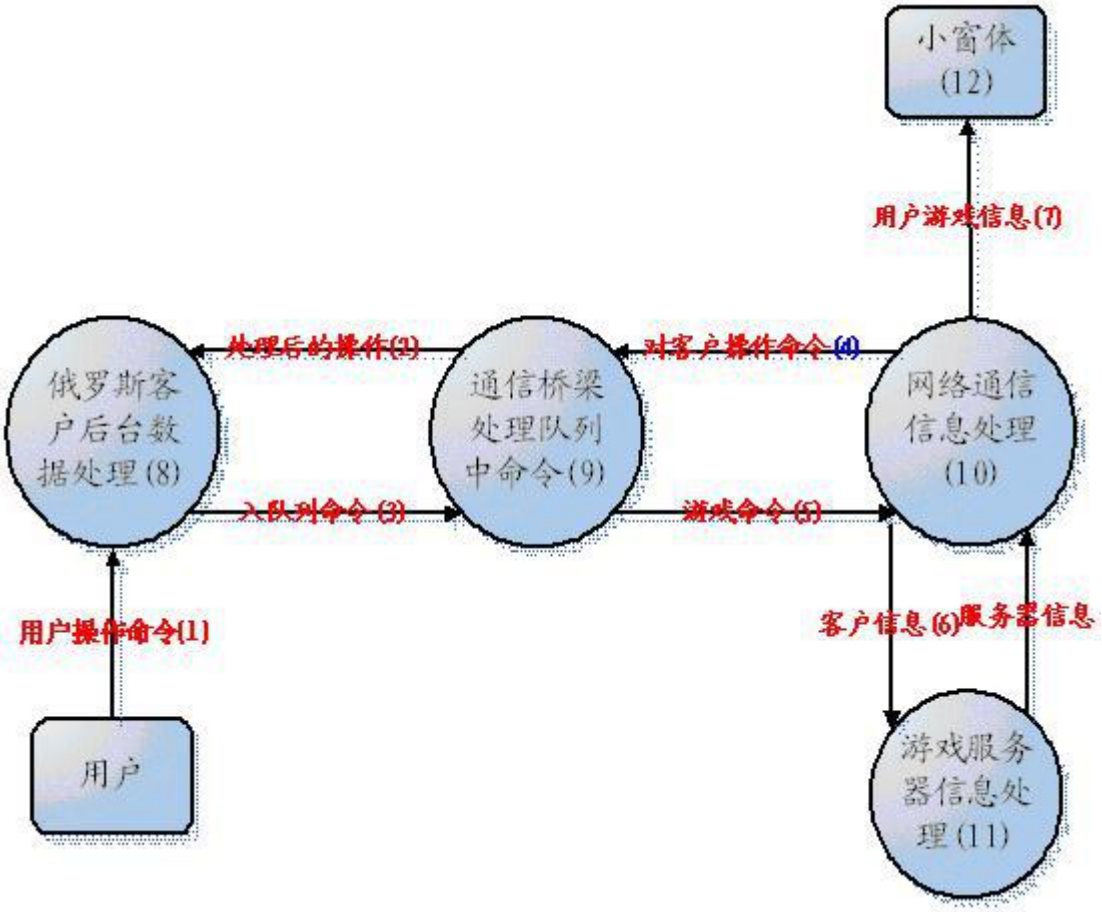
(5) 处理过程：俄罗斯后台数据处理；

说明：是指俄罗斯后台数据服务程序；
功能：主要处理游戏数据以及网络数据;

(6)网络服务器端

功能：负责网络数据的发送;

3.2 第二层数据流图



数据字典

(4)数据流：对客户的操作命令。

说明：主要是游戏中道具的发送与使用。

组成：由 数据流：操作命令详细说明 的 30—43(详细情况请查看数据流：操作命令详细说明)组成。

(7)数据流：用户游戏信息

说明：游戏过程中用户的游戏情况(矩阵)和有关道具

(9) 处理过程：通信桥梁处理队列中命令。

说明：是俄罗斯后台与网络之间通信的一个缓冲处理，起到桥梁作用，内部有一个缓冲存储了游戏命令和入队列命令。确保通信的安全。

(10)数据处理：网络通信信息处理；

说明:他负责与游戏服务器进行通信.

功能: 发送客户端信息和接收服务器信息。

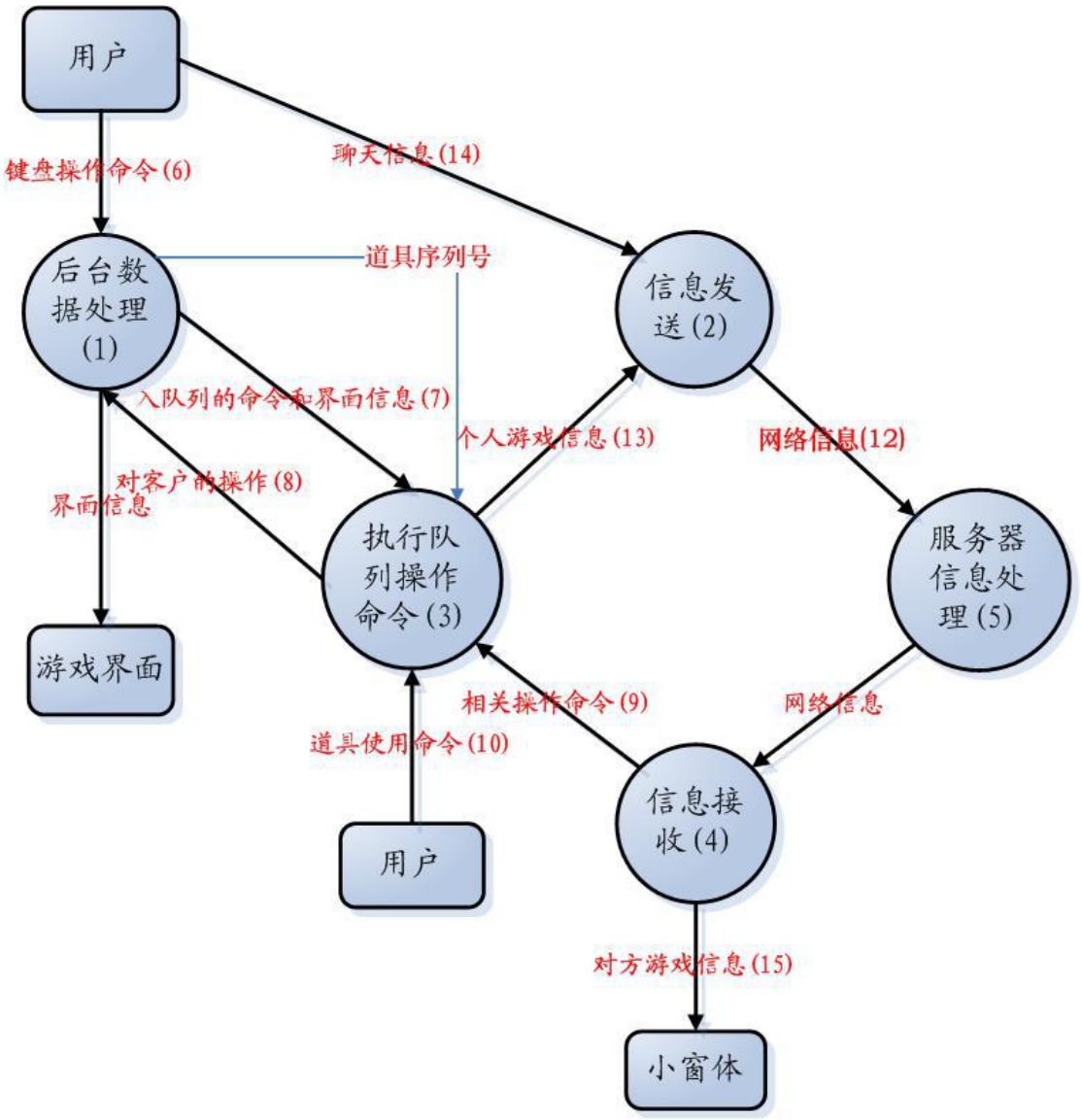
(11) 处理过程：游戏服务器信息处理

说明:服务器端数据处理

功能：接收客户端数据并向客户端发送数据；

(12)数据结构：小窗体:输出在线用户游戏信息 的窗口

3.3 第三层数据流图



数据字典

A 数据流：操作命令详细说明

组成：整数

1 对客户自己使用道具
2 转序（切换道具 S 键）
3 客户地图矩阵发生变化 ，向网络发送 Map|head|content|
12 对 2 号客户使用道具
13 对 3 号客户使用道具
14 对 4 号客户使用道具
15 对 5 号客户使用道具
16 对 6 号客户使用道具
22 每消 3 行对其他人加 2 行
23 每消 4 行对其他人加 3 行
30 网络清空游戏池
31 对手给客户消一行
32 对手给客户消 2 行
33 对手给客户消 3 行
35 对手给客户加速
36 对手给客户减速
41 对手给客户增 1 行
42 对手给客户增 2 行
43 对手给客户增 3 行

B 数据流：道具序列

组成：整数 9—17

9  攻击道具增 1 行 +1
10  攻击道具增 2 行 +2
11  攻击道具增 3 行 +3
12  攻击道具加速 ↑
13  防御道具减 1 行 -1
14  防御道具减 2 行 -2
15  防御道具减 3 行 -3
16  防御道具减速 ↓
17  清空游戏池

C 数据流：网络信息

说明：游戏服务器与客户端之间流通的网络数据

组成：如下

CONN|sender|

此时接收到的命令格式为：命令标志符（CONN）|发送者的用户名|， 服务器 将向向所有当前在线的用户除发送者的用户发送该用户信息，信息格式为 JOIN|Name|

JOIN|Name|

向客户端发送 JOIN 命令，以此来提示有新的客户进入聊天室。客户端接收后在小窗体上显示相应的客户信息。

CHAT|Content|

此时接收到的命令的格式为：命令标志符（CHAT）|发送者的用户名：发送内容| ，向所有当前在线的用户转发此信息

EXIT|Sender|

此时接收到的命令的格式为：命令标志符（EXIT）|发送者的用户名|,

向所有当前在线的用户发送该用户已离开的信息 —OUT|Sender| ，同时服务器断开与发送者的用户的网络连接，向发送者的用户发 QUIT|

OUT|NAME|

名为 name 的 人离开，客户端接收后在小窗体上清除客户信息

MAP|NAME|HEAD|CONTENT|

此时接收到的命令的格式为:命令标志符(MAP)|发送者的用户名|地图起始位置| 压缩信息|，向所有当前在线的用户除发送者的用户发送该用户游戏的信息

GAME|Sender|Receive|order|

此时接收到的命令的格式为：命令标志符(GAME)|发送者的用户名|接收者用户名 |操作命令 |，向所有当前在线的用户除发送者的用户发送该用户游戏的信息。操作命令(order)对应 30—43(详情查看操作命令详细说明)

START|Number|

此时接收到的命令的格式为：命令标志符(GAME)|发送者的用户名|接收者用户名 |道具序列号|，向所有当前在线的用户除发送者的用户发送该用户游戏的信息。

FAIL|Name|

发送者的用户游戏失败信息，服务器向所有当前在线的用户除发送者的用户发送该用户游戏失败的信息。

WIN|Name|

发送用户游戏胜利信息。

OVER|

服务器关闭。

FULL|

服务器人数已满。

MISS|

游戏已开始。

QUIT|

服务断开与客户的 连接。

数据流：对方游戏信息；

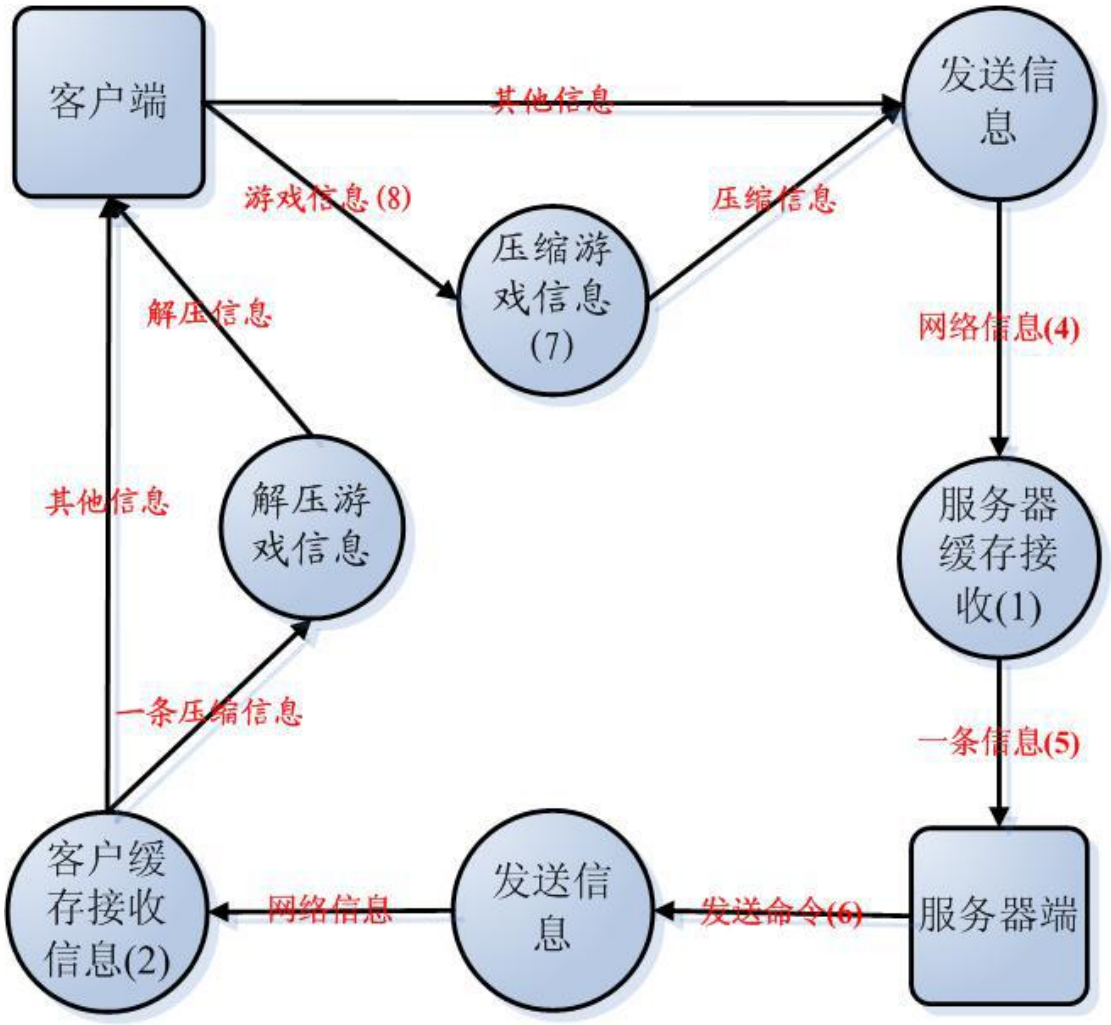
说明：是网络信息　MAP|NAME|HEAD|CONTENT|　的|head|content|解压缩；

数据流：个人游戏信息；

说明：客户当前状态和道具使用；

组成：GAME|Sender|Receive|order|+ MAP|NAME|HEAD|CONTENT|

3.4 客户端——服务器缓冲



3.4 客户端——服务器缓冲数据流图

1 处理过程：服务器缓存接收

处理：监听客户端，将接收到的信息存入缓冲队列，避免数据的丢失。

输入：网络信息；

输出：网络信息

(5)数据流:一条信息;

说明：从服务器接收缓存队列中取出存入的一条信息;

(8)数据流：游戏信息

说明： 是指矩阵的信息
组成: 客户当前游戏区域的矩阵

(7) 处理过程: 压缩游戏信息

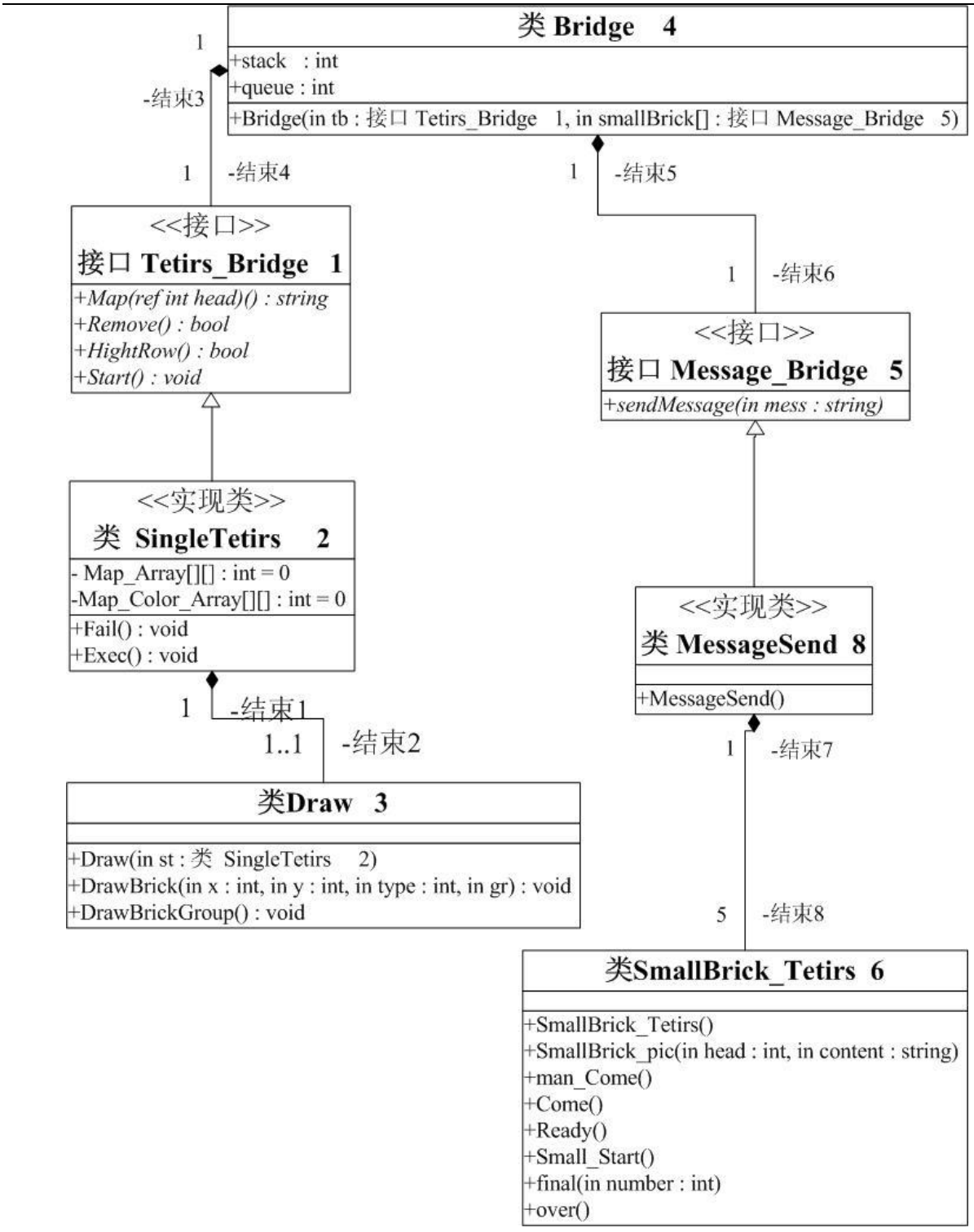
说明：对游戏信息进行压缩，减少网络传输的压力,使游戏更加快捷。
功能：以某种算法对游戏信息进行处理，减少网络数据的传输量.

(10)数据存储：客户缓存

说明：存储由服务器发送到客户端的信息
流入数据流：网络信息（详细信息请查看 3.3 第三层数据流图 - C 数据流：网络信息);
流出数据流:单个网络信息命令;
平均流量:平均每分钟 600 条;

第四部分 概要设计

4.1SuperBrickUML 类图



1 接口 Tetirs_Bridge:

说明：Bridge 类通过此接口来执行命令缓冲队列中的命令，主要包括 数据字典中的—A 数据流：操作命令详细说明的 22—43 号命令；

方法说明：

方法：string Map(ref int head) :主要用于将游戏区域地图数据（也称矩阵）转换为网络信息，向网络中发送。参数 head 是矩阵中开始有数据的一维数组下标，返回压缩后的数据。对应 C 数据流：网络信息中的 MAP|NAME|head|content|;

方法：bool Remove(int num):执行道具使用和网络道具使用中的消行功能，参数 num 是删除几行。返回 bool 型，表示是否有行可删。

方法：bool HightRow(int num): 执行道具使用和网络道具使用中的增行功能，参数 num 是增加几行。返回 bool 型，表示是否失败。

2 类 SingleTetirs:

说明：是俄罗斯方块后台数据处理的核心部分。

属性说明：

Int Map_Array[][]:是游戏区域的 矩阵，保存了整个游戏区域的方块信息，以 0 表示没有方块，以 1 表示有方块。对他的操作用到了《数据结构》的矩阵知识，绘图，消行，增行等功能的实现都用到了。

Int Map_Color_Array[][]:同样是矩阵，它保存的 是方块的颜色。

保存的从 1—17 种方块，其中 1—8 为方块，9—17 为道具。

3 类 Draw

说明：主要负责方块的绘制。

4 类 Bridge

说明:是网络与客户游戏端的连接桥梁。通过调用 Tetirs_Bridge 和 Message_Bridge 两个接口完成命令缓冲中的操作命令。操作命令的执行是通过自身的多线程完成的。每执行完一条命令，就删除一条。

属性说明：

ArrayList static stack：存储操作命令，所有的命令在 数据流：A 数据流：操作命令详细说明 中；

ArrayList static queue:存储道具序列号，详细信息请查看 B 数据流：道具序列

5 接口 Message_Bridge

说明：给 Bridge 提供一个接口，用以向网络发送游戏信息。

6 类 SmallBrick_Tetirs

说明：主要是绘制在线用户的游戏信息。他有一个线程绘制方块。

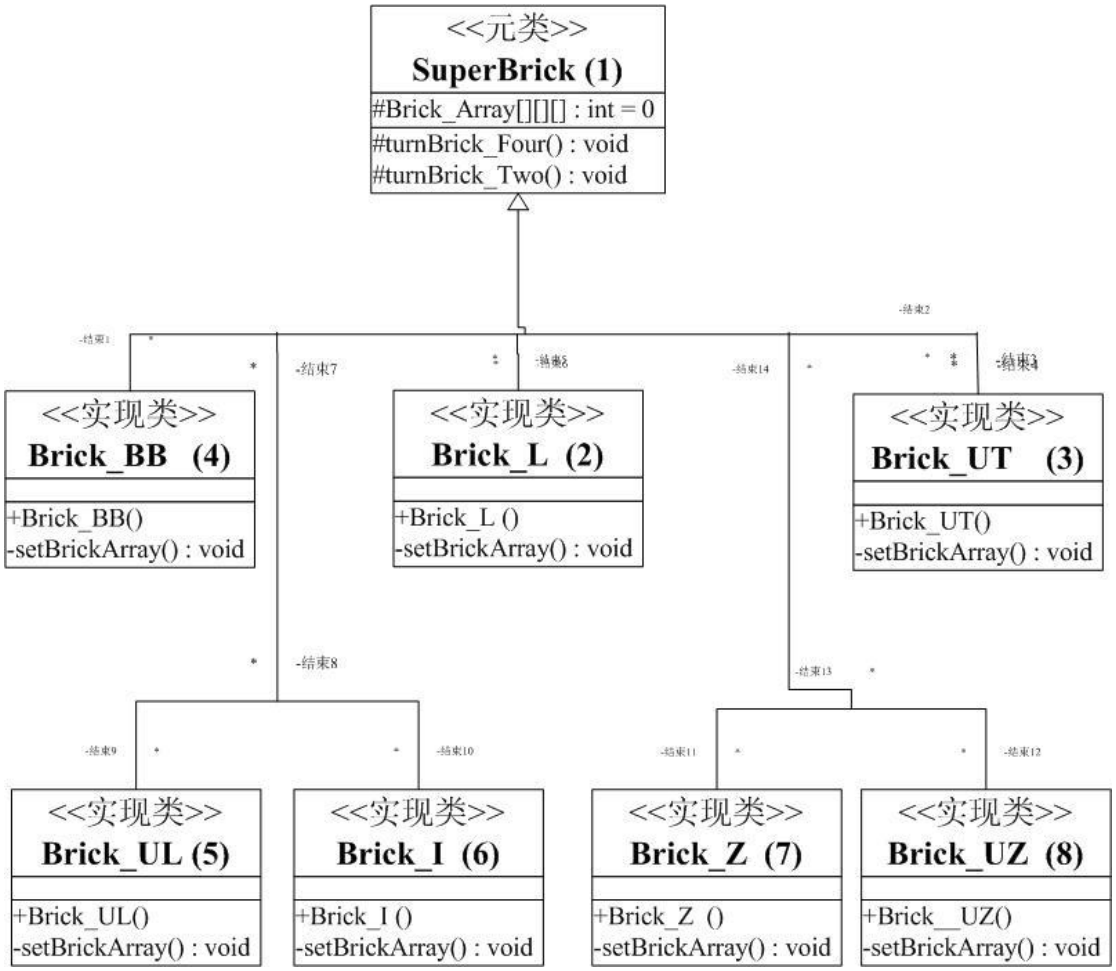
方法说明：

Void SmallBrick_Pic(int head,strig content):获取网络游戏信息，并通知线程绘制游戏信息。

8 类 MessageSend

说明：主要负责信息的 发送与接收。

4.2 方块继承



4.2 方块继承的 UML 类图

1 父类 SuperBrick

说明：他是七个俄罗斯方块的父类。集中了所有方块的旋转。

属性说明：int Brick_Array[][][] 以三维数组保存矩阵在四个转置矩阵（分别为矩阵在四个方向的转置）；

方法说明：

Void turn_Four():将矩阵旋转 4 个方向保存到 Brick_Array[][]中。

Void turn_Two():将矩阵旋转 2 个方向保存到 Brick_Array[][]中。

2 类 Brick_L

说明：方块 L 型类，继承了 Brick_Array[][]属性，构造方法中调用父类的 turnBrick_Four()。

3 类 Brick_UT

说明：倒 T 型方块。

4 类 Brick_BB

说明：田字型方块。

5 类 Brick_UL

说明：倒 T 型方块。

6 类 Brick_I

说明：长条型方块。

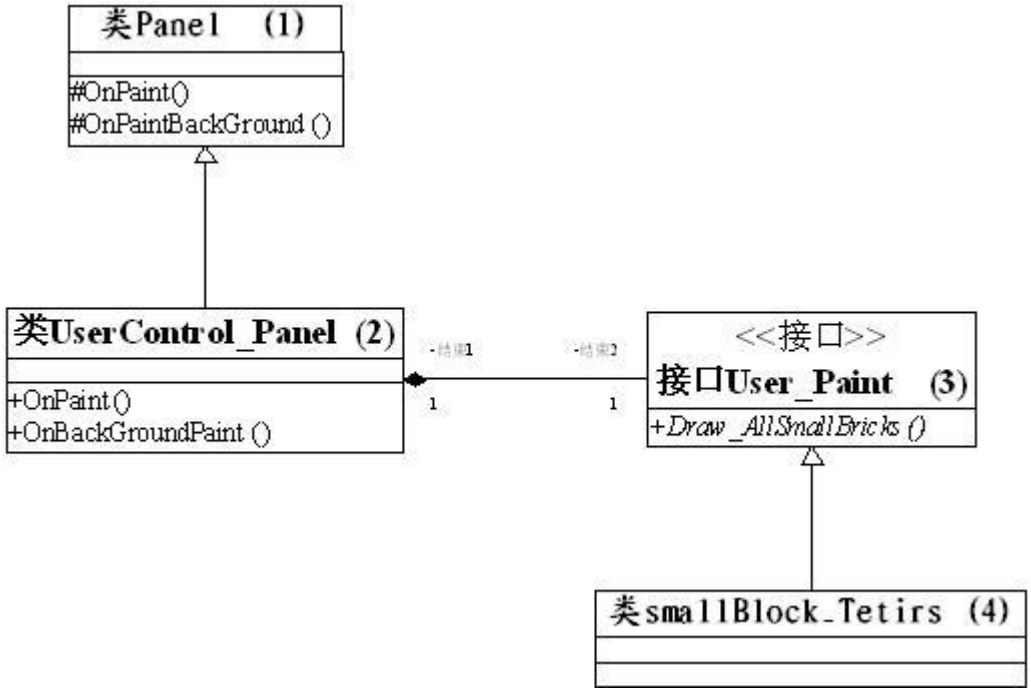
7 类 Brick_Z

说明：Z 型方块。

8 类 Brick_UZ

说明：倒 Z 型方块。

4.3 自定义控件绘图



4.3 自定义控件绘图的类图

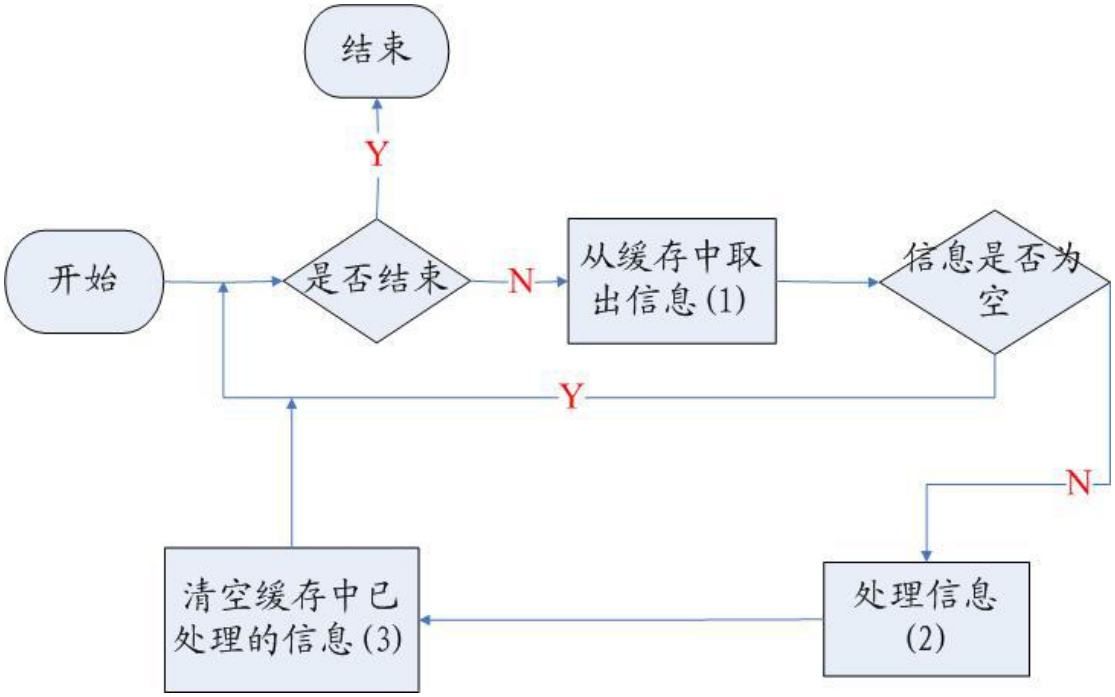
(1)类 Panel;

说明：是 System.Windows.Forms.Panel，通过重载 OnPaint() 和 OnPaintBackGround()方法，更加快捷的完成重绘制小窗体上 的方块，在最小化窗体和重绘游戏区域时不会造成游戏区域的闪动。

类 SmallBlock_Tetirs 接收网络信息后，调用 panel 的 Invalidate()来发送重绘信息。而 panel 的 OnPaint()方法则调用接口 User_Paint 的 Draw_AllSmallBrick()方法向界面绘制信息 。

第五部分　详细设计

5.1 Bridge 类线程方法体流程图



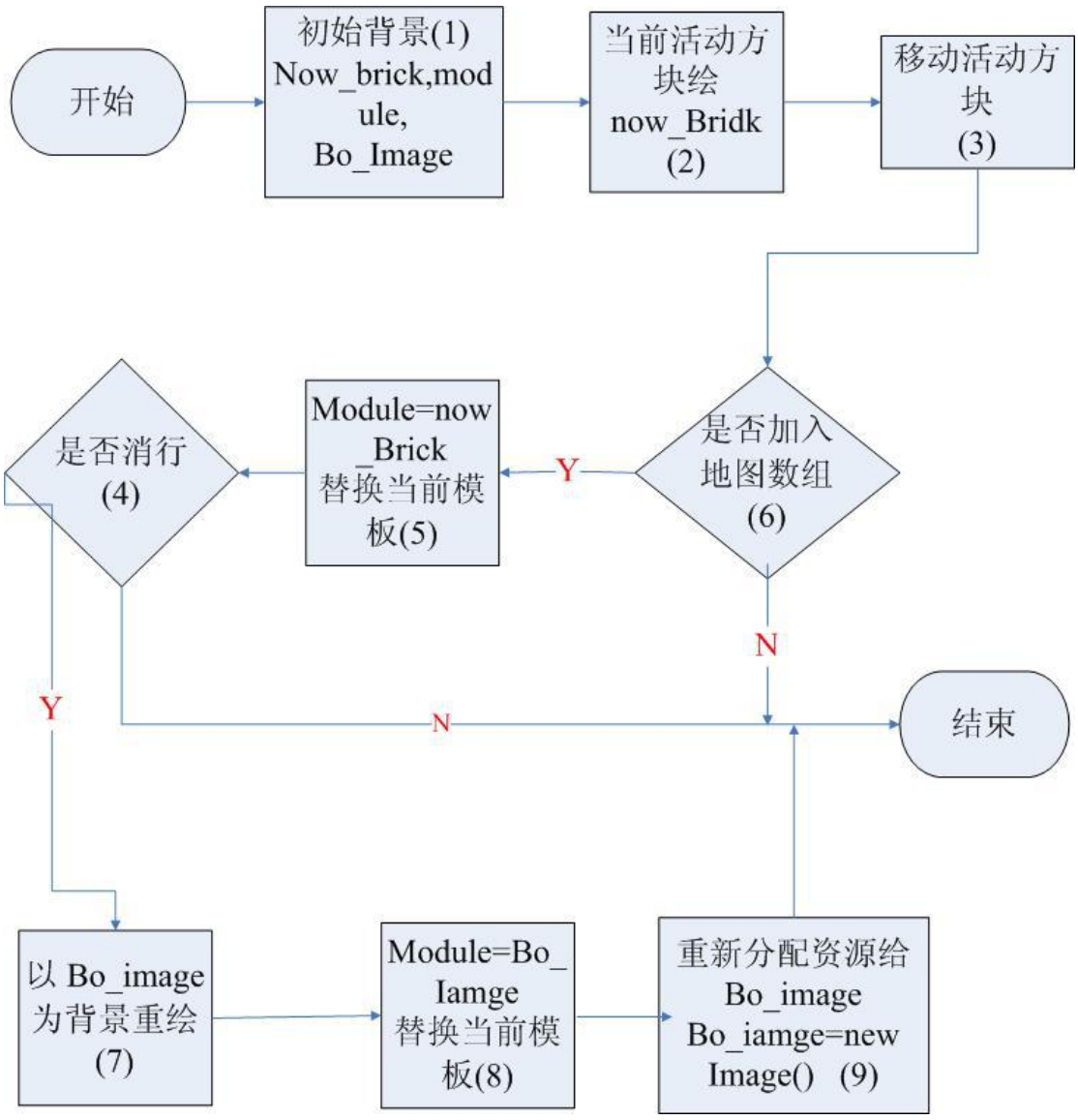
说明： 1(从缓存中取出信息)是指取出 stack 中 stack[0].

2(处理信息)通过调用Tetirs_Bridge接口和Message_Bridge接口完成响应操作。详细情况请查看 数据字典：操作命令详细说明。

3 从 stack 中删除 stack[0];

5.2 SingleTetirs 类

A SingleTetirs 类三缓存绘流程图



(5)(是否消行)是矩阵行的判断，源码如下：

```
bool blFull=true;
int delLine=0;
for(int j=this.Now_Brick_Y;j<this.Now_Brick_Y+5;j++)
{
    if(j>=this.Gridy||j<0)
        break;
    blFull=true;
    for(int i=0;i<this.Gridx;i++)
    {
        if(j>=0 &&j<this.Gridy)
        {
            if(this.Map_Array[i,j]!=1)
```

```
                    {
                        blFull=false;
                        break;
                    }
                }
            }
            if(blFull)
            {
                this.removeFull(j);
                delLine++;
            }
        }
```

(6)是否加入地图数组

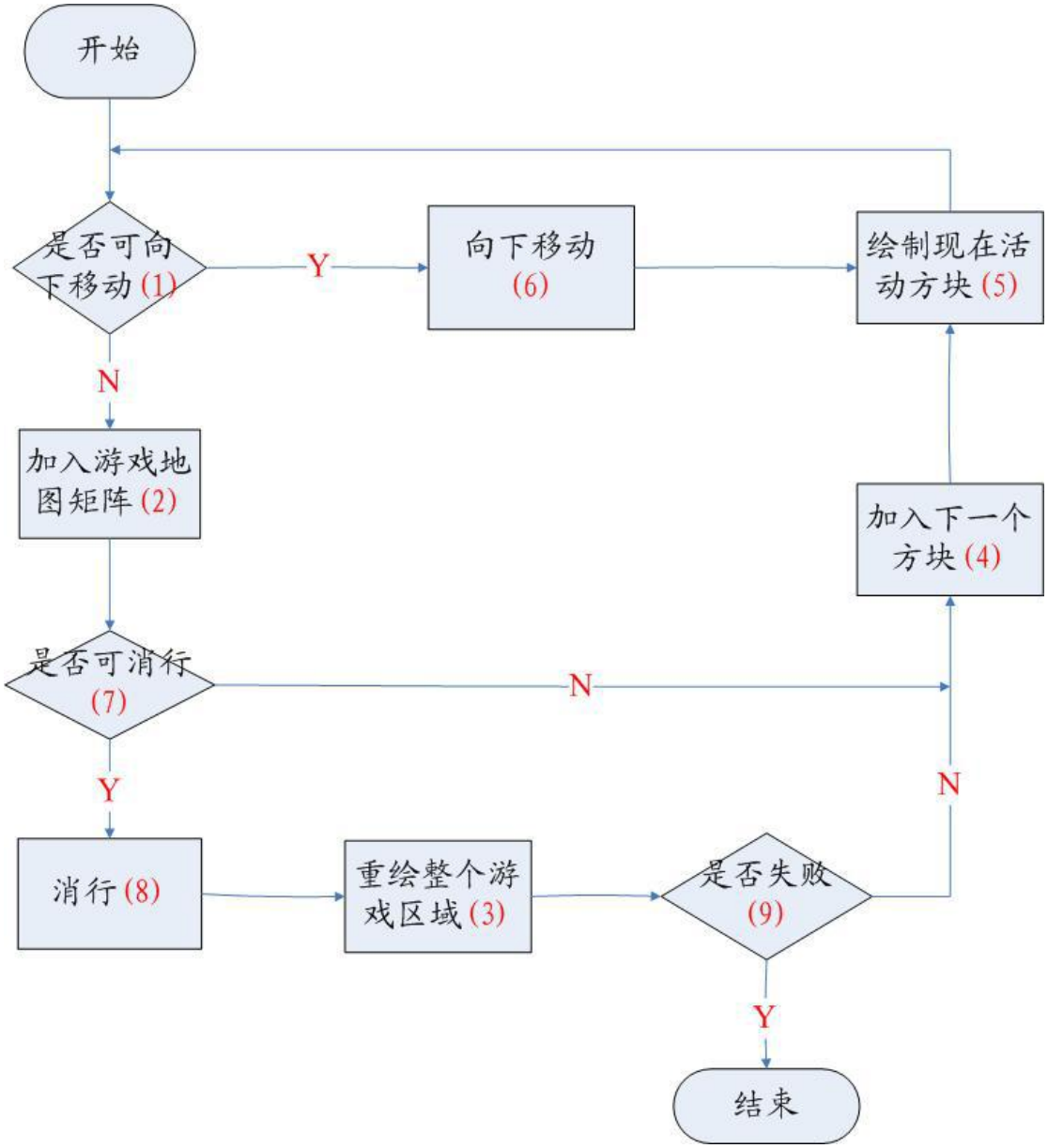
```
        public bool isPut(int x,int y,int type,int mode)//mode=direct  方块是否可
    {
        SuperBrick sb=(SuperBrick)this.BrickArray[type];
        for(int i=0;i<5;i++)
            for(int j=0;j<5;j++)
                if(sb.Brick_Array[mode,i,j]==1)
                {
                    if(          (i+x)<0          ||(i+x)>(this.Gridx-1)          ||(j+y)<0||(j+y)>(this.Gridy-1)||
(this.Map_Array[i+x,j+y]==1))//                                  {
                        return false;
                    }
                }
        return true;
    }
```

(7)重绘背景，重绘整个游戏区域，源码如下：

```
        public void drawBlocks(Image m)
        {
            lock(this)
            {
                    Graphics paint_g=this.con.CreateGraphics();
                    Graphics    mg=Graphics.FromImage(m);
```

```
                    for(int i=0;i<this.Gridx;i++)
                        for(int j=0;j<this.Gridy;j++)
                            if(this.Map_Array[i,j]==1)
                                d.DrawBrick(i,j,this.Map_Color_Array[i,j],mg);
paint_g.DrawImage(m,0,0,this.con.Width,this.con.Height);
                    mg.Dispose();
                    paint_g.Dispose();
                    this.module=m;
                    this.draw_NowBrick();//画现在的方块
                }
        }
```

B 方块向下移动流程图



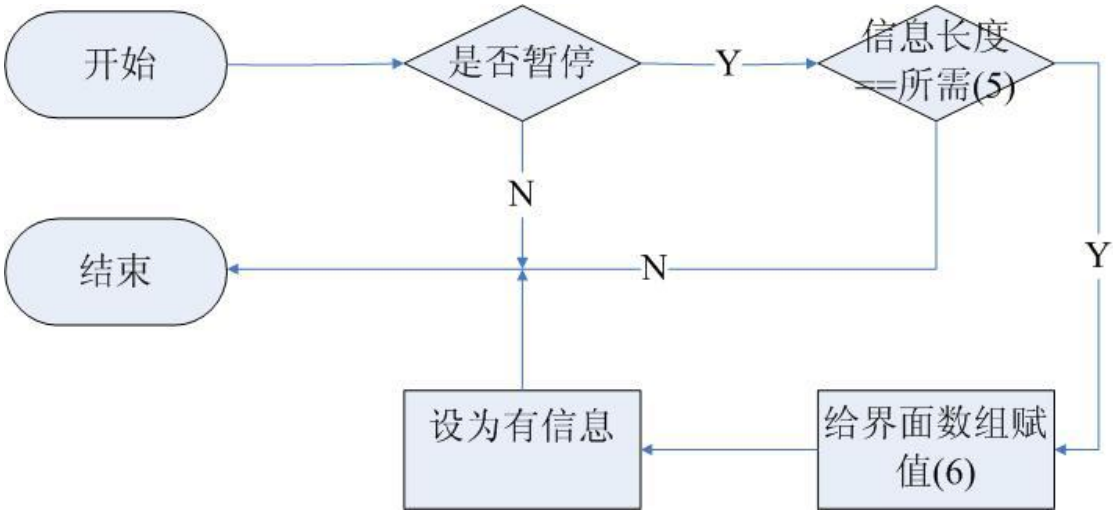
(5)绘制现在活动方块，源码如下：

```
//画活动方块
    private void draw_NowBrick()
    {
        try
        {
            this.now_BrickImage=new Bitmap(this.module,this.con.Width,this.con.Height);
            lock(this)
            {
                Graphics paint_g=this.con.CreateGraphics();
                                Graphics    mg=Graphics.FromImage(this.now_BrickImage);
```

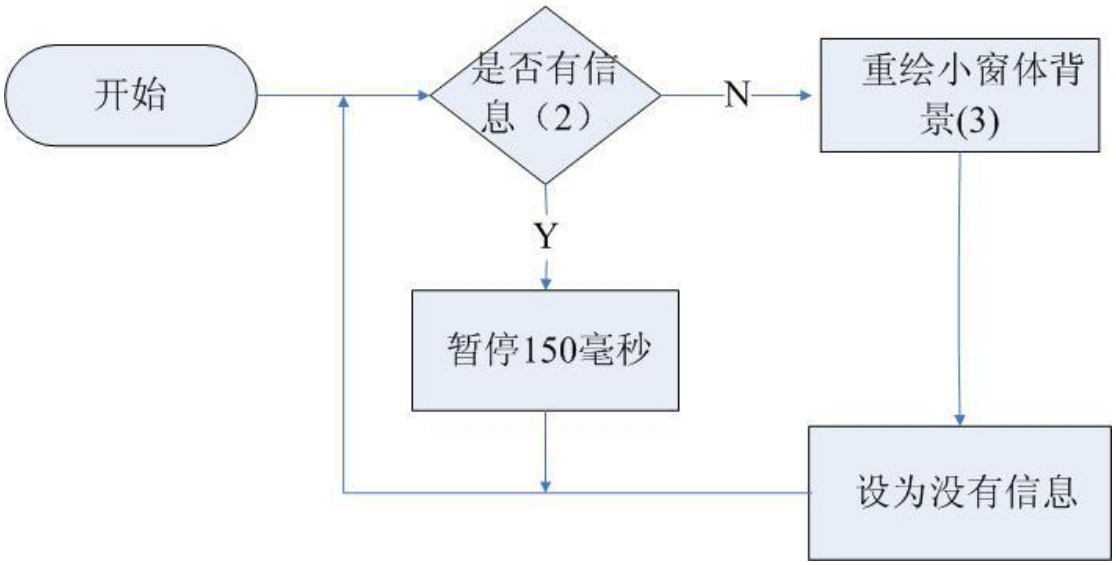
```
                    this.d.DrawBrickGroup(mg);
paint_g.DrawImage(this.now_BrickImage,0,0,this.con.Width,this.con.Height);
                    mg.Dispose();
                    paint_g.Dispose();
                }
            }
            catch{}
        }
```

5.3 小窗体（smallBlock）流程图

A SmallBrick_Pic 方法体流程图



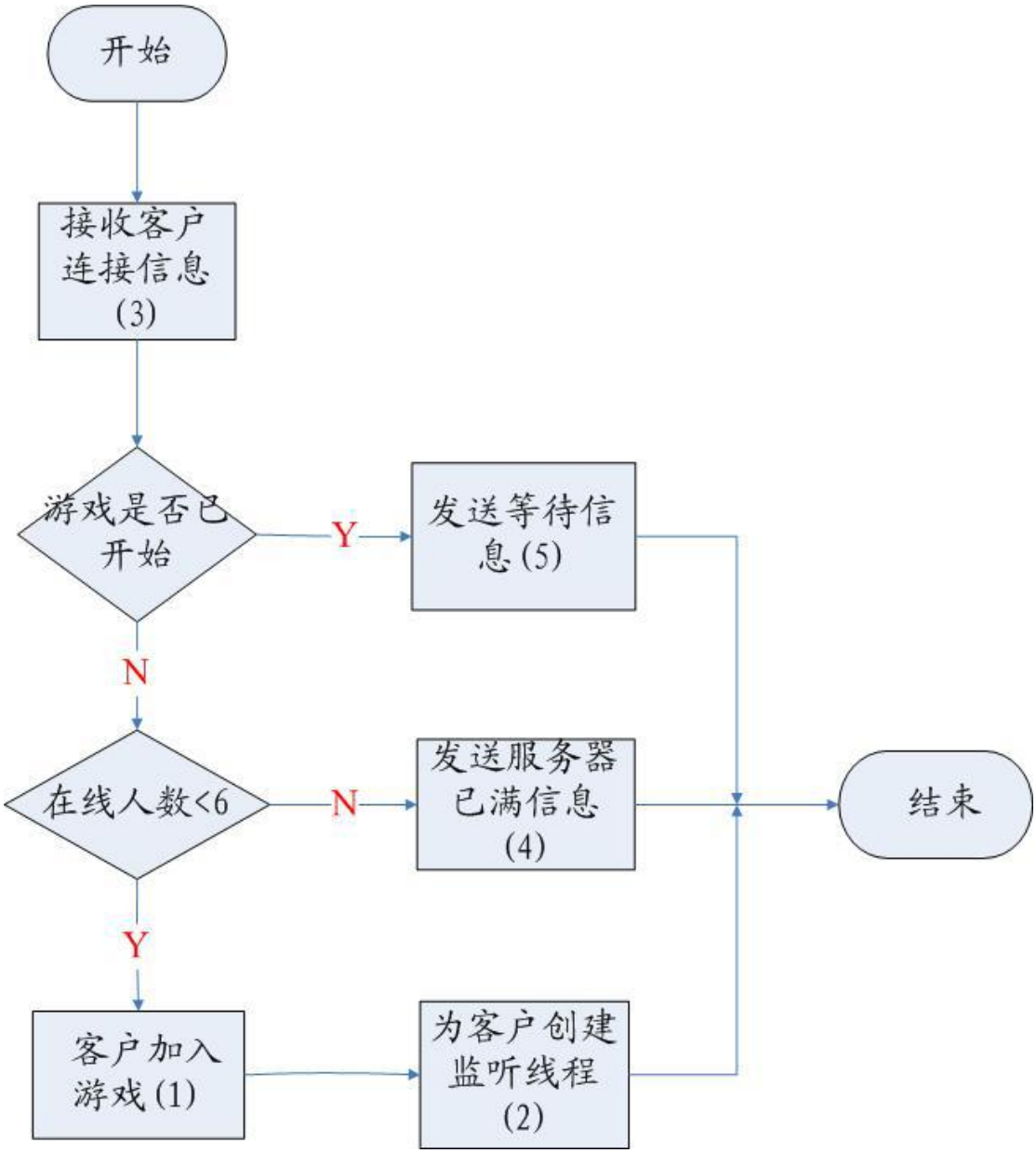
B smallBrick_Tetirs_panelDraw()方法体流程图



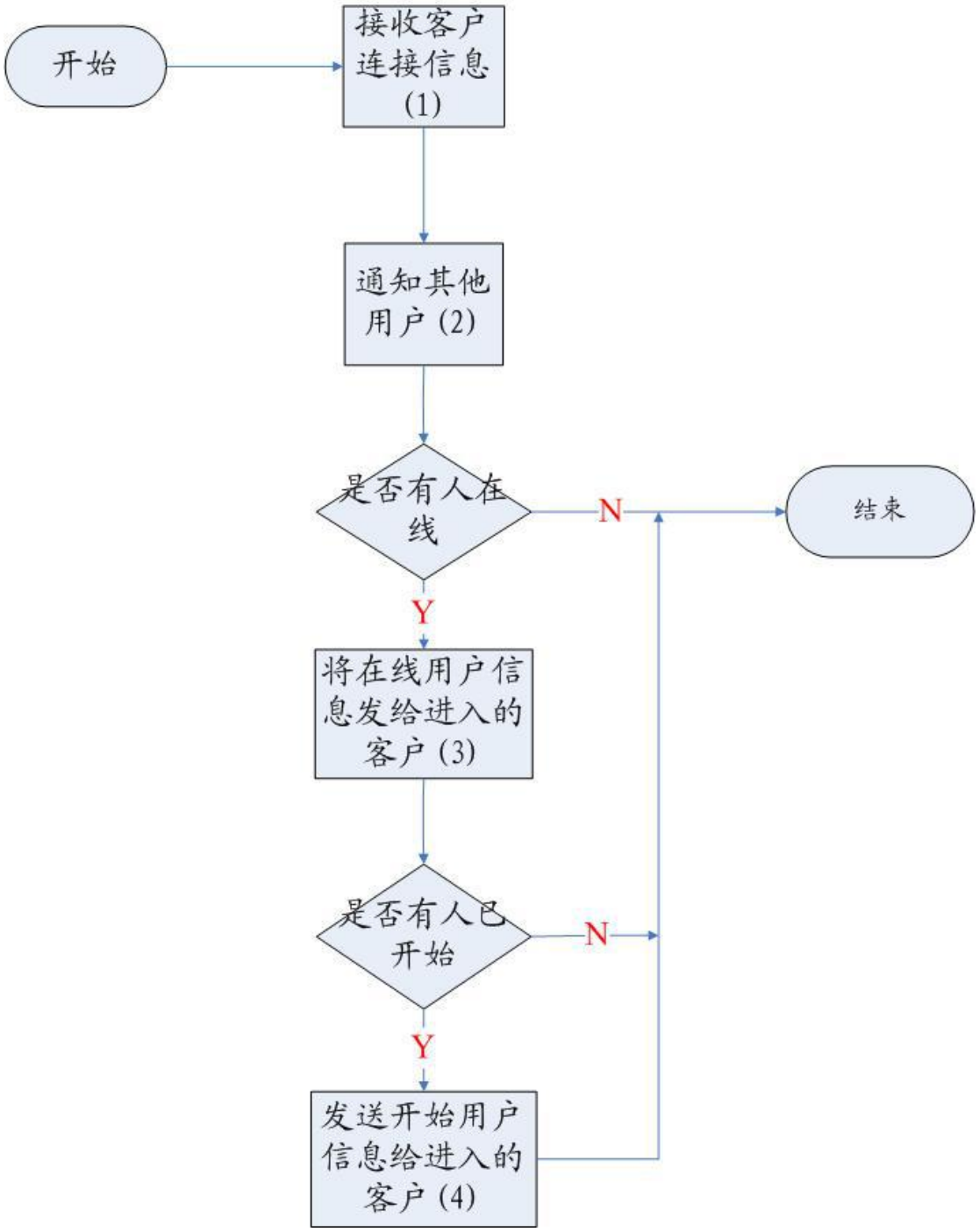
5.4 服务器流图

在新的线程中的操作，它主要用于当接收到一个客户端请求时，确认与客户端的连接，并且立刻启动一个新的线程来处理和该客户端的信息交互（在方法 ServiceClient（）中实现）。

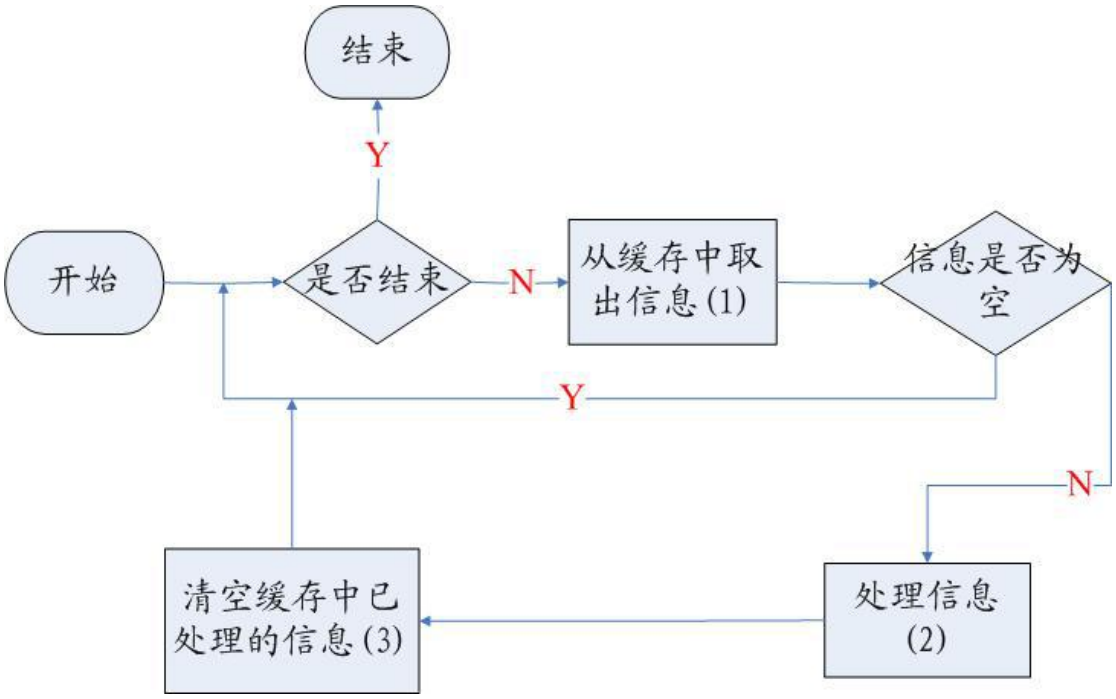
A 客户请求连接时流程图



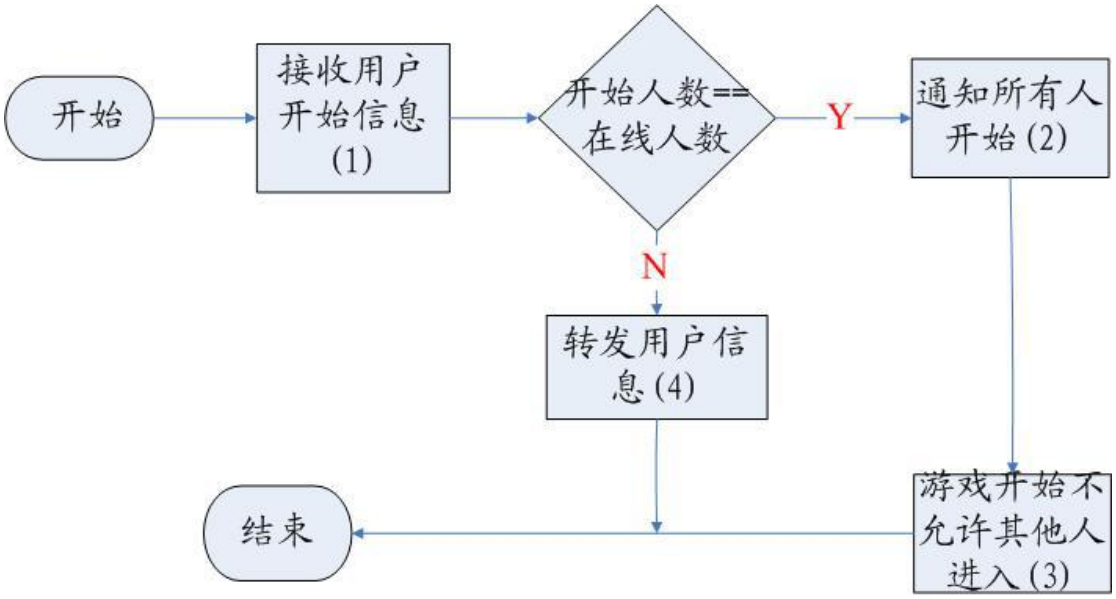
B 客户连接服务器时的流程图



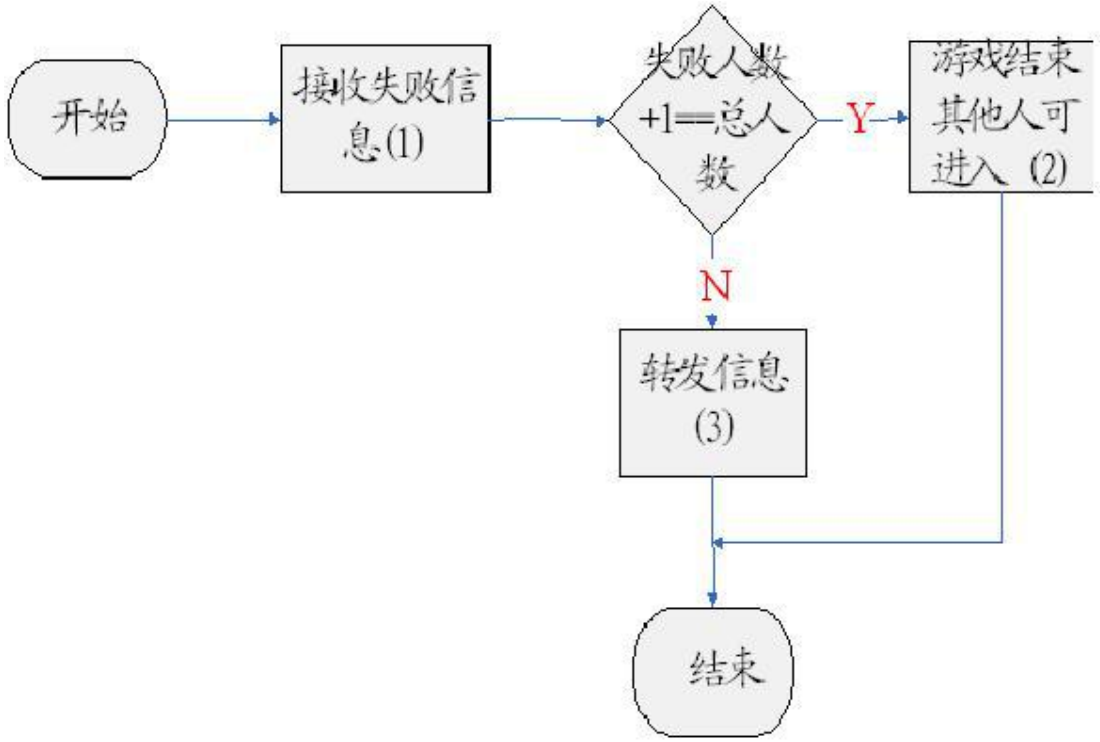
C 服务器线程监听客户网络连接流程图　ServiceClient()



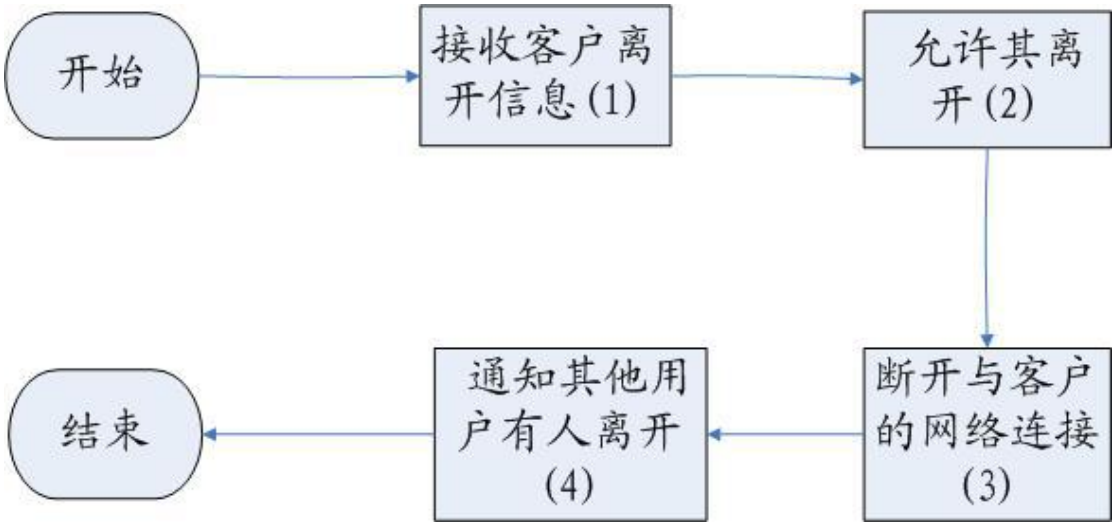
D 服务器接收 Start 信息流图



E 服务器接收 Fail 信息流程图



F 服务器接收 Exit 流程图



第六部分　程序源代码

6.1 SingleTetirs 键盘操作命令执行源码：

```
public void keyPress(Keys k)
        {
            int x=this.Now_Brick_X;
                int y=this.Now_Brick_Y;
                int direct=this.Now_Brick_Direct;
                switch(k)
                {
                    case Keys.Up:
                        this.Now_Brick_Direct++;
                        if(this.Now_Brick_Direct>=4)
                            this.Now_Brick_Direct=0;
    if(this.isPut(this.Now_Brick_X,this.Now_Brick_Y,this.Now_Brick_Num,this.Now_Brick_Direct))
                        {
                            this.draw_NowBrick();
                            Sound.Play(7);
                        }
                        else
                        {
                            this.Now_Brick_Direct--;
                            if(this.Now_Brick_Direct<0)
                                this.Now_Brick_Direct=3;
                        }
                        break;
                    case Keys.Down:

    if( this.isPut(this.Now_Brick_X,this.Now_Brick_Y+1,this.Now_Brick_Num,this.Now_Brick_Direct))
                        {
                            this.Now_Brick_Y++;
                            //this.drawBlocks();
                            this.draw_NowBrick();
                        }
                        else
                        {
                            this.addBrickToMap();
                            this.checkFull();
                            this.addNextBrick();
                            this.drawBlocks();
                            this.checkFail();
```

```
                }
                break;
            case Keys.Left:

if( this.isPut(this.Now_Brick_X-1,this.Now_Brick_Y,this.Now_Brick_Num,this.Now_Brick_Direct))
                {
                    this.Now_Brick_X--;
                    //this.drawBlocks();
                    this.draw_NowBrick();
                }
                break;
            case Keys.Right:
    if( this.isPut(this.Now_Brick_X+1,this.Now_Brick_Y,this.Now_Brick_Num,this.Now_Brick_Direct))
                {
                    this.Now_Brick_X++;
                    //this.drawBlocks();
                    this.draw_NowBrick();
                }
                break;
            case Keys.Space:
while( this.isPut(this.Now_Brick_X,this.Now_Brick_Y+1,this.Now_Brick_Num,this.Now_Brick_Direct))
                {
                    this.Now_Brick_Y++;
                    x=this.Now_Brick_X;
                    y=this.Now_Brick_Y;
                }
                //this.drawBlocks();
                this.draw_NowBrick();
                Sound.Play(3);
                this.addBrickToMap();
                this.checkFull();
                this.addNextBrick();
                this.drawBlocks();
                //this.draw_NowBrick();
                this.formControl.timer1.Enabled=true;
                break;
        }
    }
```

第七部分　系统测试与评价

测试环境：小型局域网。操作系统为 Widows XP。

6 台客户机，1 台服务机。

测试人员 7 人。

测试用例：6 个客户同时进入，6 个客户顺序进入，6 个客户同时退出，6 个客户同时游戏，发言。

测试效果：没有出现故障。

因设备，人员和时间的限制，软件还要做如下测试：模拟网络故障，干扰网络等破坏性测试。

第八部分　系统使用说明

1 游戏窗口说明

1.1 游戏窗口如下图所示

1.2 标题栏

标题栏：显示游戏客户端版本号及房间信息
最小化按钮：将游戏最小化
最大化按钮：切换游戏正常 / 全屏模式
关闭按钮：关闭游戏（当游戏在进行中时弹出逃跑提示）
参数设置按钮：自定义键位设置、开启 / 关闭音效
快速帮助按钮：弹出游戏快速说明页面

1.3 玩家信息栏

next 区域显示即将落下的下一块方块，和下下块方块。即将落下的第二块方块以灰色形式显示。

1.4 道具说明栏

道具说明栏显示玩家获得的第一个道具（道具栏最左端）的简要说明，进攻性道具以红色文字显示，防御性道具以蓝色文字显示

1.5 开始按钮

选择好队伍后（或者默认为自由人队）按开始按钮开始一局游戏。在游戏中开始按钮为灰色，游戏结束后允许开始新的游戏时该按钮亮起。

1.6 聊天框

聊天框只支持文字的输入和发送。按回车或者点击“发送”按钮可以把聊天文字发送出去。当焦点在游戏区域时聊天框无法打字，反之当焦点在聊天框时，游戏无法操作。可以用鼠标点击来切换游戏焦点。

2. 游戏规则

游戏为对战型俄罗斯方块，玩家依靠自己消层和使用道来打击对手。当玩家游戏池中的砖块累积到顶端时游戏失败。

游戏以组队形式提高乐趣，同一队玩家全部失败则该队失败（自由人自己属于一队，不和其他自由人组队），失败队伍扣除相应分数。唯一剩下的一队将成为获胜队，获胜队将平分其他玩家失败所扣除的所有分数。（当玩家失败，但是他所在的队伍还未失败时将暂时不返回该玩家名次，等该队全部结束游戏后结算其分数。若玩家在结算分数前离开游戏，则扣除玩家最高分数 12 分）。

玩家消层后，将对与其不同队的所有玩家造成伤害。 一次消四层则对其他玩家加三层，一次消三层则对其他玩家加两层。

玩家可以使用道具对对手进行伤害或者对队友进行援救，游戏道具分为红色（攻击）和蓝色（防御）两种，目前共 9 种。红色为：加一层，加两层，加三层，加速。蓝色道具有：减一层，减两层，减三层，屏幕上砖块全消，减速。

3. 操作说明

游戏界面上的按钮以鼠标操作

键盘［ 左、右 ］用来控制落下砖块的左右移动

键盘［ 下 ］用来控制砖块加速下落

键盘［ 上 ］用来控制砖块变形

键盘［ 空格 ］用来控制砖块直接落到底

键盘［1-6 （大键盘） ］用来对指定座位的玩家使用道具（玩家座位号在玩家信息栏有显示）

键盘［s］用来切换道具，将所有道具依次往右移动一格，最右边的道具移动到最左边

（以上的键位设置是游戏默认的设置，您也可以按自己的习惯随时更改。）

4. 道具说明：

道具分为红色（攻击）和蓝色（防御）两种 ；

参考文献：

1 Visual C#.NET 编程精 150 例；　　冶金工业出版社
2 C#经典范例 50 讲；　　清华大学出版社
3 数据结构——朱若愚　　电子工业出版社
4 数据结构习题与解析——李春葆　　清华大学出版社