Java大作业连连看课程报告

组员：王致渊，付兴

分工：王致渊——连连看基本功能，实现了本地存储和存盘，支持加载后继续执行功能。

付兴———连连看基本功能，实现了游戏回放的设计，实现了游戏的网络存储。

源代码见最后，或见https://github.com/adtureven/LianLianKan

游戏简介：本次连连看游戏使用`Java`进行制作，使用GUI其中的`swing`和`awt`库进行创作，实现了连连看的基本功能——选中相同方块进行消除，实现了游戏回放功能，支持本地和网络存储，关闭后仍能继续执行其功能。

模块设计：
连连看游戏可以分为以下几个模块：

1. `GamePanel`类：该类实现了游戏界面的设计和主要功能的实现。它包含了游戏面板的绘制、图片的加载和显示、点击事件的处理、连线的判断和消除等功能。`GamePanel`类是游戏的核心模块，负责游戏的整体逻辑和交互。
2. `GameState`类：该类用于记录游戏的状态，并实现了存储功能。它可以记录当前游戏的进度、得分、剩余时间等信息，并提供了保存和读取游戏状态的方法。`GameState`类可以在游戏暂停或退出时保存当前状态，在下次进入游戏时读取之前的状态，实现游戏的断点续玩功能。
3. `GameClient`类：该类实现了主界面的设计和功能。它包含了游戏的开始、暂停、重新开始等按钮的处理，以及游戏状态的显示和切换等功能。`GameClient`类负责游戏的整体控制和界面展示，通过与`GamePanel`和`GameState`类的交互，实现了游戏的开始、暂停、保存等功能。

4. `Map`和`MapFactory`类：这两个类实现了图片的随机排列功能。Map类用于存储游戏面板上的图片信息，包　括图片的种类、位置等。`MapFactory`类负责生成随机的图片排列，并提供了判断两个图片是否可以消除的　　方法。这两个类为`GamePanel`类提供了游戏面板的初始化和更新功能，保证了游戏的可玩性和随机性。

5. `GameServer`类：该类实现了网络通信的功能，支持用户端和服务端之间的通信，实现了连连看游戏的网络　存储功能。

具体实现——分模块介绍 *具体介绍本人负责的功能（基本功能，本地存储和存盘功能）*

`Map`和`MapFactory`类：

```
//MapFactory
public static int[][] getMap(int n){
    map = new int[n][n];//生成n*n地图
    //初始化地图信息为空
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            map[i][j] = -1;
        }
    }
    return map;
}
```

该函数的功能是生成一个连连看游戏的地图。参数n表示地图的大小，即`n*n`的方格数。函数首先创建一个`n*n`大小的地图数组map。然后通过两个嵌套的for循环，将地图数组的每个元素初始化为-1，表示该位置为空。最后返回生成的地图数组。

```
//Map
public int[][] getMap(){
    ArrayList<Integer> list = new ArrayList<Integer>(); // 创建一个存放图片ID的列表
    for(int i=0;i<n*n/10;i++){ // 循环n*n/10次，表示每种图片ID出现的次数
        for(int j=0;j<count;j++){ // 循环count次，表示每种图片ID的数量
            list.add(j); // 将图片ID添加到列表中
        }
    }
    for(int i=0;i<n;i++){ // 遍历地图的行
        for(int j=0;j<n;j++){ // 遍历地图的列
            int index = (int) (Math.random()*list.size()); // 从列表中随机取一个图片ID的索引
            map[i][j] = list.get(index); // 将图片ID添加到地图数组中
            list.remove(index); // 从列表中删除已经使用的图片ID
        }
    }
    return map; // 返回生成的地图数组
}
```

该函数用于生成连连看游戏的地图数组。首先，根据每种图片ID出现的次数和数量，将图片ID添加到一个列表中。然后，遍历地图的每个位置，随机从列表中取一个图片ID，并将其添加到地图数组中，同时从列表中删除已经使用的图片ID。最后，返回生成的地图数组。

`GameState`**类:**

```
//GameState
public class GameState implements Serializable {
    private String count;       //记录游戏状态中的分数
    private int[][] map;        //记录游戏过程中消除图片的地图数组
    GameState(String count,int[][]map){
        this.count=count;
        this.map=map;
    }
    public String getCount(){               //设置方法使其可以得到状态中的分数
        return count;
    }
    public int[][] G_getMap(){              //设置方法使其可以得到状态中的地图
        return map;
    }
}
```

`GamePanel`**类:**

构造方法

```
//GamePanel
public GamePanel(GameState gameState){
    // 设置游戏面板的大小为600x600
    setSize(600, 600);
    // 初始化游戏面板的行列数为10
    n=10;
    // 从游戏状态对象中获取当前剩余可消除的方块数量，并转换为整型
    count=Integer.parseInt(gameState.getCount());
    // 将count的值赋给count1
    count1=count;
    // 从游戏状态对象中获取当前游戏地图的二维数组，并赋值给map
    map=gameState.G_getMap();
    // 将map的值复制给map1，用于备份地图数据
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            map1[i][j]=map[i][j];
        }
    }
    // 设置游戏面板可见
    this.setVisible(true);
    // 添加键盘监听器和鼠标监听器
    this.addKeyListener(this);
    this.addMouseListener(this);
```

```
    // 设置游戏面板获取焦点
    this.setFocusable(true);
    // 加载游戏所需的图片资源
    getPics();
    // 重绘游戏面板
    repaint();
}
```

获取图片资源 `getPics()`

```
//GamePanel
private void getPics() {
    pics = new Image[10];
    for(int i=0;i<=9;i++){
        pics[i] = Toolkit.getDefaultToolkit().getImage("pic"+(i+1)+".png");
    }   //从相对路径中获取游戏的图片素材
}
```

绘图 `paint()`

```
//GamePanel
public void paint(Graphics g){
    // 清除指定区域的图像
    g.clearRect(0, 0, 800, 30);

    // 遍历游戏地图的每个格子
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            // 如果该格子不为空白状态
            if(map[i][j]!=BLANK_STATE){
                // 在指定位置绘制图片
                g.drawImage(pics[map[i][j]],leftX+j*50,leftY+i*50,50,50,this);
            }else{
                // 清除指定位置的图像
                g.clearRect(leftX+j*50,leftY+i*50,50,50);
            }
        }
    }
    // 设置k的值为1
    k=1;
}
```

该函数用于绘制连连看游戏的游戏面板。首先，通过调用`g.clearRect()`方法清除指定区域的图像，以便绘制新的游戏面板。然后，通过遍历游戏地图的每个格子，判断该格子的状态。如果该格子不为空白状态，则在指定位置绘制对应的图片。如果该格子为空白状态，则清除该位置的图像。最后，将k的值设置为1。

一系列消除的判断逻辑

```
//GamePanel

//判断能否直接水平连接
private boolean horizontalLink(int clickX1, int clickY1, int clickX2, int
clickY2) {
    // 保证clickY1小于等于clickY2
    if(clickY1>clickY2){
        int temp1 = clickX1;
        int temp2 = clickY1;
        clickX1 = clickX2;
        clickY1 = clickY2;
        clickX2 = temp1;
        clickY2 = temp2;
    }
    // 如果两个方块在同一列
    if(clickX1==clickX2){
        // 判断两个方块之间是否有其他方块阻挡
        for(int i=clickY1+1;i<clickY2;i++){
            if(map[clickX1][i]!=BLANK_STATE){
                return false;
            }
        }
        // 设置连接方式为水平连接
        linkMethod = LINKBYHORIZONTAL;
        return true;
    }
    return false;
}

/*该函数用于判断两个方块是否可以水平连接。首先，通过比较clickY1和clickY2的大小，保证clickY1
小于等于clickY2。然后，如果两个方块在同一列（即clickX1等于clickX2），则遍历两个方块之间的所有
行，判断是否有其他方块阻挡，如果有，则返回false；如果没有，则设置连接方式为水平连接
（linkMethod = LINKBYHORIZONTAL），并返回true。如果两个方块不在同一列，则直接返回false。
*/


//判断能否直接垂直消除
private boolean verticalLink(int clickX1, int clickY1, int clickX2, int clickY2)
{
    // 保证clickX1小于等于clickX2
    if(clickX1>clickX2){
        int temp1 = clickX1;
        int temp2 = clickY1;
        clickX1 = clickX2;
        clickY1 = clickY2;
        clickX2 = temp1;
        clickY2 = temp2;
    }
    // 判断两个点击点是否在同一行
    if(clickY1==clickY2){
        // 遍历两个点击点之间的所有格子
        for(int i=clickX1+1;i<clickX2;i++){
            // 如果有非空格子，则无法垂直连连看
```

```
            if(map[i][clickY1]!=BLANK_STATE){
                return false;
            }
        }
        // 设置连连看方式为垂直连连看
        linkMethod = LINKBYVERTICAL;
        return true;
    }
    return false;
}


// 判断两个方块是否可以通过一个拐角相连
private boolean oneCornerLink(int clickX1, int clickY1, int clickX2, int
clickY2) {

    // 确保clickY1小于等于clickY2
    if(clickY1>clickY2){
        int temp1 = clickX1;
        int temp2 = clickY1;
        clickX1 = clickX2;
        clickY1 = clickY2;
        clickX2 = temp1;
        clickY2 = temp2;
    }

    // 如果clickX1小于clickX2
    if(clickX1<clickX2){

        // 判断clickX1和clickY2之间是否为空白方块，并且clickX1和clickY1之间以及clickX2和
clickY2之间可以水平和垂直相连
        if(map[clickX1][clickY2]==BLANK_STATE && horizontalLink(clickX1,
clickY1, clickX1, clickY2)&&verticalLink(clickX2,clickY2,clickX1,clickY2)){
            linkMethod = LINKBYONECORNER;
            z1 = new Node(clickX1, clickY2);
            return true;
        }

        // 判断clickX2和clickY1之间是否为空白方块，并且clickX2和clickY2之间以及clickX1和
clickY1之间可以水平和垂直相连
        if(map[clickX2][clickY1]==BLANK_STATE&&horizontalLink(clickX2, clickY2,
clickX2, clickY1)&&verticalLink(clickX1,clickY1,clickX2, clickY1)){
            linkMethod = LINKBYONECORNER;
            z1 = new Node(clickX2, clickY1);
            return true;
        }

    }else{ // 如果clickX1大于等于clickX2

        // 判断clickX2和clickY1之间是否为空白方块，并且clickX2和clickY2之间以及clickX1和
clickY1之间可以水平和垂直相连
        if(map[clickX2][clickY1]==BLANK_STATE&&horizontalLink(clickX2, clickY2,
clickX2, clickY1)&&verticalLink(clickX1,clickY1,clickX2, clickY1)){
            linkMethod = LINKBYONECORNER;
            z1 = new Node(clickX2, clickY1);
```

```
            return true;
        }

        // 判断clickX1和clickY2之间是否为空白方块，并且clickX1和clickY1之间以及clickX2和
clickY2之间可以水平和垂直相连
        if(map[clickX1][clickY2]==BLANK_STATE&&horizontalLink(clickX1, clickY1,
clickX1, clickY2)&&verticalLink(clickX2,clickY2,clickX1, clickY2)){
            linkMethod = LINKBYONECORNER;
            z1 = new Node(clickX1, clickY2);
            return true;
        }

    }

    return false;
}
/*1. 首先，通过比较clickY1和clickY2的大小，确保clickY1小于等于clickY2。
2. 然后，判断clickX1和clickX2的大小关系：
   - 如果clickX1小于clickX2，执行以下操作：
     - 判断clickX1和clickY2之间是否为空白方块，并且clickX1和clickY1之间以及clickX2和
clickY2之间可以水平和垂直相连。
     - 如果满足条件，则将linkMethod设置为LINKBYONECORNER，并将z1设置为一个Node对象，其横
纵坐标为clickX1和clickY2，然后返回true。
     - 判断clickX2和clickY1之间是否为空白方块，并且clickX2和clickY2之间以及clickX1和
clickY1之间可以水平和垂直相连。
     - 如果满足条件，则将linkMethod设置为LINKBYONECORNER，并将z1设置为一个Node对象，其横
纵坐标为clickX2和clickY1，然后返回true。
   - 如果clickX1大于等于clickX2，执行以下操作：
     - 判断clickX2和clickY1之间是否为空白方块，并且clickX2和clickY2之间以及clickX1和
clickY1之间可以水平和垂直相连。
     - 如果满足条件，则将linkMethod设置为LINKBYONECORNER，并将z1设置为一个Node对象，其横
纵坐标为clickX2和clickY1，然后返回true。
     - 判断clickX1和clickY2之间是否为空白方块，并且clickX1和clickY1之间以及clickX2和
clickY2之间可以水平和垂直相连。
     - 如果满足条件，则将linkMethod设置为LINKBYONECORNER，并将z1设置为一个Node对象，其横
纵坐标为clickX1和clickY2，然后返回true。
3. 如果以上条件都不满足，则返回false。*/


//判断两个折角能否消除
private boolean twoCornerLink(int clickX1, int clickY1, int clickX2, int
clickY2);

//判断两个方块在水平方向上是否可以连通
private boolean throughHorizontalLink(int clickX, int clickY, boolean flag);
/*该函数用于判断两个方块在水平方向上是否可以连通。通过传入点击方块的横坐标clickX、纵坐标clickY
以及判断方向的标志flag，判断两个方块是否可以通过水平方向上的空白方块连通。
如果flag为true，表示向上判断，函数会从点击方块的上方开始向上遍历，如果遇到非空方块，则返回
false，表示无法连通；如果遍历到最上方仍然没有遇到非空方块，则返回true，表示可以连通。
如果flag为false，表示向下判断，函数会从点击方块的下方开始向下遍历，如果遇到非空方块，则返回
false，表示无法连通；如果遍历到最下方仍然没有遇到非空方块，则返回true，表示可以连通。
最后，如果没有遇到非空方块，则返回true，表示可以连通。*/

//判断是否可以通过垂直方向连接
private boolean throughVerticalLink(int clickX,int clickY,boolean flag);
```

```
/*该函数的功能是判断在连连看游戏中，从点击的方块出发，是否可以通过垂直方向连接到其他方块。函数通过传入的参数clickX和clickY确定了点击的方块的位置，通过flag参数确定了遍历的方向（向上或向下）。函数首先根据flag的值选择遍历的方向，然后从点击的方块的位置开始遍历，判断遍历过程中是否遇到了非空方块，如果遇到了非空方块，则返回false，表示不能通过垂直方向连接。如果遍历完成后都没有遇到非空方块，则返回true，表示可以通过垂直方向连接。*/
```

以上列出了6种连连看中消除方块的逻辑，可以在游戏进行时进行对玩家选中方块是否能消除进行判断，并返回判断值。

对可消除两个方块之间进行连线

```
//GamePanel
private void drawLink(int x1, int y1, int x2, int y2) {
    // 获取Graphics对象，用于绘制图形
    Graphics g = this.getGraphics();
    // 计算连接线的起始点和终止点的坐标
    Point p1 = new Point(y1*50+leftX+25,x1*50+leftY+25);
    Point p2 = new Point(y2*50+leftX+25,x2*50+leftY+25);
    // 根据连接方式绘制连接线
    if(linkMethod == LINKBYHORIZONTAL || linkMethod == LINKBYVERTICAL){
        // 如果连接方式是水平或垂直连接，则直接绘制一条直线
        g.drawLine(p1.x, p1.y,p2.x, p2.y);
    }else if(linkMethod == LINKBYONECORNER){
        // 如果连接方式是通过一个拐角连接，则需要绘制两条线段
        Point point_z1 = new Point(z1.y*50+leftX+25,z1.x*50+leftY+25);
        g.drawLine(p1.x, p1.y,point_z1.x, point_z1.y);
        g.drawLine(p2.x, p2.y,point_z1.x, point_z1.y);
    }else{
        // 如果连接方式是通过两个拐角连接，则需要绘制三条线段
        Point point_z1 = new Point(z1.y*50+leftX+25,z1.x*50+leftY+25);
        Point point_z2 = new Point(z2.y*50+leftX+25,z2.x*50+leftY+25);
        // 确保起始点和拐角点的坐标关系正确
        if(p1.x!=point_z1.x&&p1.y!=point_z1.y){
            Point temp;
            temp = point_z1;
            point_z1 = point_z2;
            point_z2 = temp;
        }
        g.drawLine(p1.x, p1.y, point_z1.x, point_z1.y);
        g.drawLine(p2.x, p2.y, point_z2.x, point_z2.y);
        g.drawLine(point_z1.x,point_z1.y, point_z2.x, point_z2.y);
    }
    // 更新计数器，显示连接次数
    count+=2;
    GameClient.textField.setText(count+"");
    // 线程休眠500毫秒，以便玩家观察连接效果
    try {
        Thread.currentThread().sleep(500);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    // 重新绘制游戏面板
    repaint();
```

```
    // 将连接的两个方块标记为空白状态
    map[x1][y1] = BLANK_STATE;
    map[x2][y2] = BLANK_STATE;
    // 检查是否胜利
    isWin();
}
```

该方法的功能是在游戏面板上绘制连接线，并更新计数器、重新绘制游戏面板、标记连接的两个方块为空白状态，并检查是否胜利。具体实现根据连接方式的不同，绘制不同数量的线段。连接方式有三种：水平或垂直连接、通过一个拐角连接、通过两个拐角连接。在绘制连接线之前，先计算连接线的起始点和终止点的坐标。绘制连接线后，更新计数器并显示连接次数，然后线程休眠500毫秒以便玩家观察连接效果。最后，重新绘制游戏面板，将连接的两个方块标记为空白状态，并检查是否胜利。

判断每次消除过后是否已经胜利

```
//GamePanel
private void isWin() {
    // 如果已经消除的方块数量等于总方块数量
    if(count==n*n){
        // 弹出一个对话框，提示玩家游戏完成
        String msg = "再来一轮吧";
        int type = JOptionPane.YES_NO_OPTION;
        String title = "恭喜你完成了游戏！";
        int choice = JOptionPane.showConfirmDialog(null, msg,title,type);
        // 如果玩家选择了“否”，则退出游戏
        if(choice==1){
            System.exit(0);
        }
        // 如果玩家选择了“是”，则开始新的游戏
        else if(choice == 0){
            startNewGame();
        }
    }
}
```

该函数用于判断游戏是否胜利。当已经消除的方块数量等于总方块数量时，弹出一个对话框提示玩家游戏完成，并根据玩家的选择进行相应的操作。如果玩家选择了“否”，则退出游戏；如果玩家选择了“是”，则开始新的游戏。

开始新游戏startNewgame

```
//GamePanel
public void startNewGame() {
    count = 0; // 重置游戏计数器为0
    mapUtil = new Map(10, n); // 生成新的游戏地图
    map = mapUtil.getMap(); // 获取生成的地图数组
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            map1[i][j] = map[i][j]; // 复制地图到备份数组
        }
    }
```

```
    count1 = count; // 备份游戏计数器

    isClick = false; // 重置点击状态为未点击
    clickId = -1; // 重置点击ID为-1
    clickX = -1; // 重置点击X坐标为-1
    clickY = -1; // 重置点击Y坐标为-1
    linkMethod = -1; // 重置连连看方法为-1
    GameClient.textField.setText(count + ""); // 更新游戏计数器显示
    repaint(); // 重新绘制游戏界面
}
```

对鼠标按下事件的处理

```
//GamePanel
public void mousePressed(MouseEvent e) {
    Graphics g = this.getGraphics(); // 获取画笔对象

    // 获取鼠标点击的坐标
    int x = e.getX() - leftX;
    int y = e.getY() - leftY;

    // 判断点击是否在游戏区域内
    if (x < 0 || y < 0 || x > 500 || y > 500)
        return;

    // 计算点击的方块在地图中的行列索引
    int i = y / 50;
    int j = x / 50;

    // 生成点击操作记录
    String operation = "CLICK " + i + " " + j;
    operationRecord.add(operation);

    // 判断是否已经有方块被点击
    if (isClick) {
        // 判断点击的方块是否为空白方块
        if (map[i][j] != BLANK_STATE) {
            // 判断点击的方块是否与上一个点击的方块相同
            if (map[i][j] == clickId) {
                // 判断点击的方块是否与上一个点击的方块相同位置
                if (i == clickX && j == clickY)
                    return;

                // 判断点击的方块与上一个点击的方块是否可以连通
                if (verticalLink(clickX, clickY, i, j) || horizontalLink(clickX,
clickY, i, j) || oneCornerLink(clickX, clickY, i, j) || twoCornerLink(clickX,
clickY, i, j)) {
                    // 绘制选中的方块
                    drawSelectedBlock(j * 50 + leftX, i * 50 + leftY, g);
                    // 绘制连线
                    drawLink(clickX, clickY, i, j);
                    isClick = false; // 取消方块点击状态
```

```
                } else {
                    clickId = map[i][j]; // 更新点击的方块ID
                    clearSelectBlock(clickX, clickY, g); // 清除上一个选中的方块
                    clickX = i; // 更新点击的方块行索引
                    clickY = j; // 更新点击的方块列索引
                    drawSelectedBlock(j * 50 + leftX, i * 50 + leftY, g); // 绘制
选中的方块
                }
            } else {
                clickId = map[i][j]; // 更新点击的方块ID
                clearSelectBlock(clickX, clickY, g); // 清除上一个选中的方块
                clickX = i; // 更新点击的方块行索引
                clickY = j; // 更新点击的方块列索引
                drawSelectedBlock(j * 50 + leftX, i * 50 + leftY, g); // 绘制选中
的方块
            }
        }
    } else {
        // 判断点击的方块是否为空白方块
        if (map[i][j] != BLANK_STATE) {
            clickId = map[i][j]; // 更新点击的方块ID
            isClick = true; // 设置方块点击状态为true
            clickX = i; // 更新点击的方块行索引
            clickY = j; // 更新点击的方块列索引
            drawSelectedBlock(j * 50 + leftX, i * 50 + leftY, g); // 绘制选中的方块
        }
    }
}
```

`GameClient` **类:**

在属性中设置了三个按钮和一个文本框

```
//GameClient
JButton button1 = new JButton("重新开始");
JButton button2 = new JButton("退出");
JButton button3 = new JButton("replay");
static JTextField textField = new JTextField(10);
```

构造函数

```
//GameClient
public GameClient(){
    JLabel label1 = new JLabel("已消去方块数量: ");
    JLabel label2=new JLabel("按D提示");
    JPanel panel = new JPanel(new BorderLayout());
    textField.setEditable(false);
```

```
    loadState();
    System.out.println(gameState.getCount());
    panel2=new GamePanel(gameState);
    textField.setText(gameState.getCount());

    panel2.setLayout(new BorderLayout());
    panel.setLayout(new FlowLayout());
    panel.add(label1);
    panel.add(textField);
    panel.add(button1);
    panel.add(button2);
    panel.add(button3);
    panel.add(label2);
    this.getContentPane().setLayout(new BorderLayout());
    this.getContentPane().add(panel,BorderLayout.SOUTH);
    this.getContentPane().add(panel2,BorderLayout.CENTER);
    this.setSize(600,630);
    //this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            saveState();
            dispose();
        }
    });

    this.setTitle("连连看游戏");
    this.setVisible(true);
    button1.setEnabled(true);
    button2.setEnabled(true);
    button3.setEnabled(true);

    button1.addMouseListener(new MouseAdapter(){
        public void mouseClicked(MouseEvent e){
            textField.setText("0");
            panel2.startNewGame();
        }
    });

    button2.addMouseListener(new MouseAdapter(){
        public void mouseClicked(MouseEvent e){
            saveState();
            System.exit(0);
        }
    });

    button3.addMouseListener(new MouseAdapter(){
        public void mouseClicked(MouseEvent e){
            textField.setText("0");
            panel2.replay();
        }
    });
}
```

```
/*具体功能如下：

1. 创建标签label1和label2，分别用于显示已消去方块数量和提示信息。
2. 创建面板panel，并使用BorderLayout布局管理器。
3. 设置textField为不可编辑状态。
4. 调用loadState()方法加载游戏状态。
5. 在控制台打印游戏状态的方块数量。
6. 创建GamePanel对象panel2，并将游戏状态传入。
7. 将游戏状态的方块数量设置为textField的文本。
8. 设置panel2和panel的布局管理器分别为BorderLayout和FlowLayout。
9. 将label1、textField、button1、button2、button3和label2添加到panel中。
10. 设置GameClient的内容面板布局管理器为BorderLayout，并将panel和panel2分别添加到内容面板的南部和中部。
11. 设置GameClient的窗口大小为600x630。
12. 注册窗口关闭事件，当窗口关闭时调用saveState()方法保存游戏状态，并释放资源。
13. 设置GameClient的标题为"连连看游戏"，并将窗口设置为可见状态。
14. 启用button1、button2和button3的鼠标事件监听器。
15. 当点击button1时，将textField的文本设置为"0"，并调用panel2的startNewGame()方法开始新游戏。
16. 当点击button2时，调用saveState()方法保存游戏状态，并退出程序。
17. 当点击button3时，将textField的文本设置为"0"，并调用panel2的replay()方法重新开始游戏。
*/
```

保存游戏状态saveState

```
//GameClient
public void saveState(){
    try{
        // 创建一个GameState对象，将文本框和面板的地图数据传入
        gameState=new GameState(textField.getText(), panel2.get_Map());

        // 创建一个文件输出流，将游戏状态对象写入文件gamestate.txt
        FileOutputStream fileout=new FileOutputStream("gamestate.txt");
        ObjectOutputStream out =new ObjectOutputStream(fileout);
        out.writeObject(gameState);
        out.close();
        fileout.close();

        System.out.println("saved");
    }catch (Exception e){
        e.printStackTrace();
    }

    try (Socket socket = new Socket(SERVER_HOST, SERVER_PORT);
         ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
         ObjectInputStream in = new ObjectInputStream(socket.getInputStream())
    ) {
        // 向服务器发送游戏状态数据
        out.writeObject(gameState);
        out.flush();

        // 接收服务器的响应
```

```
            String response = (String) in.readObject();
            System.out.println(response);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
```

该函数首先创建一个`GameState`对象，将文本框和面板的地图数据传入，然后将该对象写入文件`gamestate.txt`，实现了保存游戏状态的功能。 接着，函数创建一个Socket对象，连接到指定的服务器主机和端口。然后创建一个对象输出流和对象输入流，用于向服务器发送游戏状态数据和接收服务器的响应。 最后，函数将游戏状态数据写入对象输出流，并刷新输出流。然后从对象输入流中读取服务器的响应，并打印出来。 如果在保存游戏状态或与服务器通信的过程中出现异常，将打印异常信息。

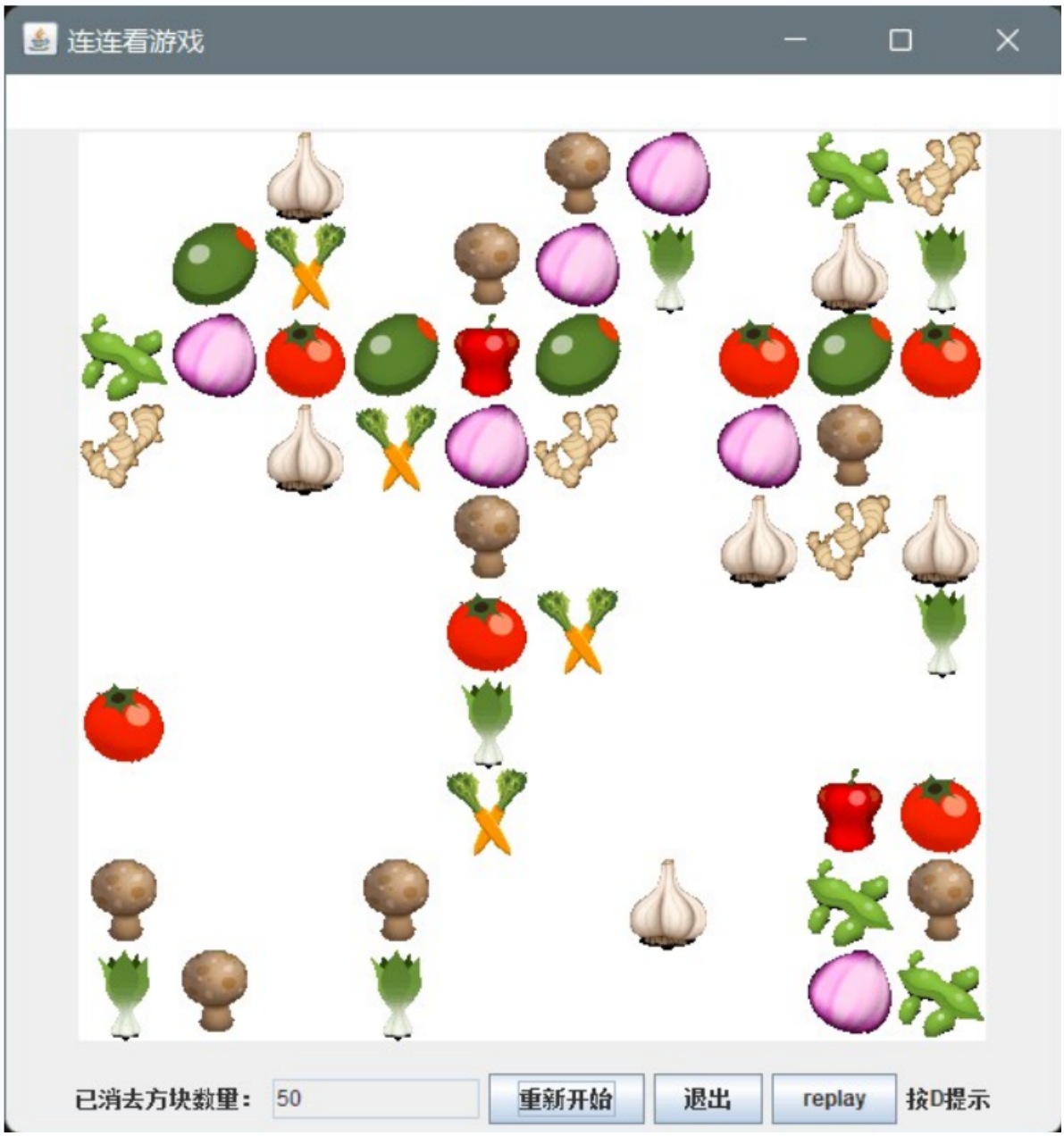
读取游戏状态

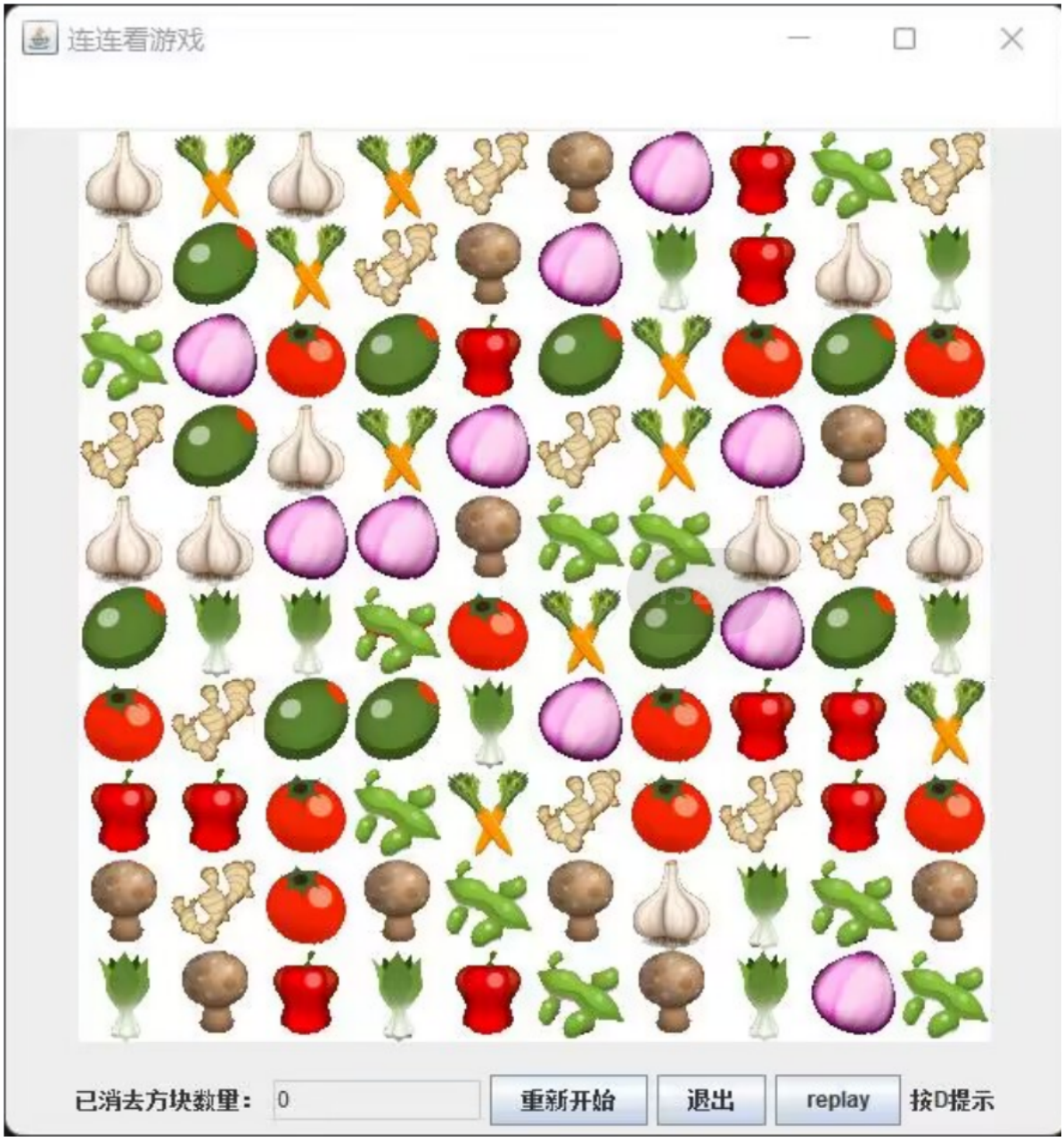
```
//GameClient
public void loadState(){
    try{
        // 创建一个文件输入流，用于读取保存游戏状态的文件
        FileInputStream filein=new FileInputStream("gamestate.txt");
        // 创建一个对象输入流，用于读取对象数据
        ObjectInputStream in=new ObjectInputStream(filein);
        // 从文件中读取游戏状态对象，并将其赋值给gameState变量
        gameState=(GameState)in.readObject();
        // 关闭输入流
        in.close();
        // 关闭文件输入流
        filein.close();
        // 输出提示信息，表示游戏状态加载成功
        System.out.println("load");
    }catch (Exception e){
        // 输出异常信息
        e.printStackTrace();
    }
}
```

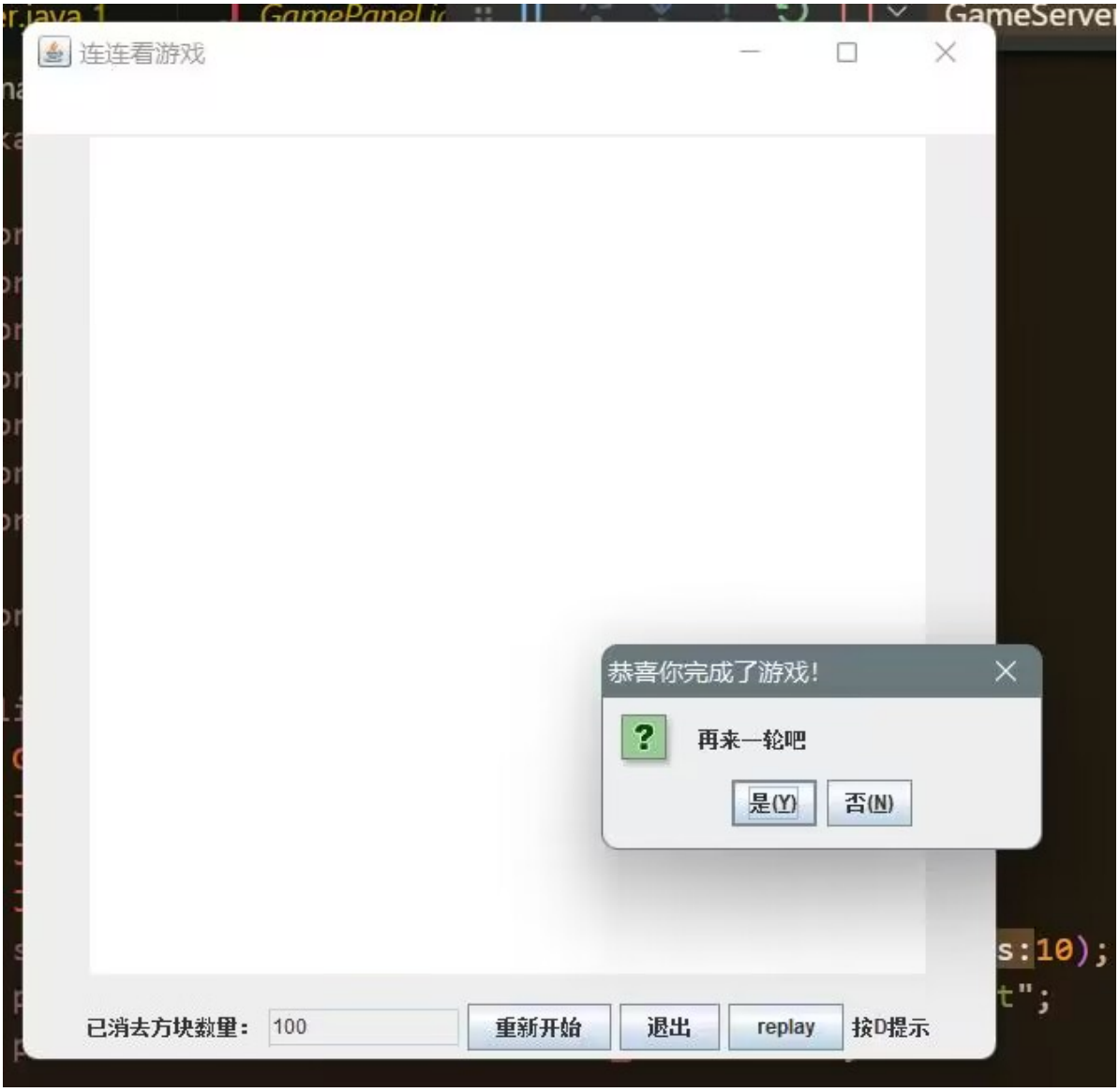
该函数用于加载游戏状态，从保存游戏状态的文件中读取游戏状态对象，并将其赋值给`gameState`变量。具体实现步骤如下： 1. 创建一个文件输入流，用于读取保存游戏状态的文件。 2. 创建一个对象输入流，用于读取对象数据。 3. 从文件中读取游戏状态对象，并将其赋值给`gameState`变量。 4. 关闭输入流。 5. 关闭文件输入流。 6. 输出提示信息，表示游戏状态加载成功。 7. 如果在加载过程中出现异常，打印异常信息。

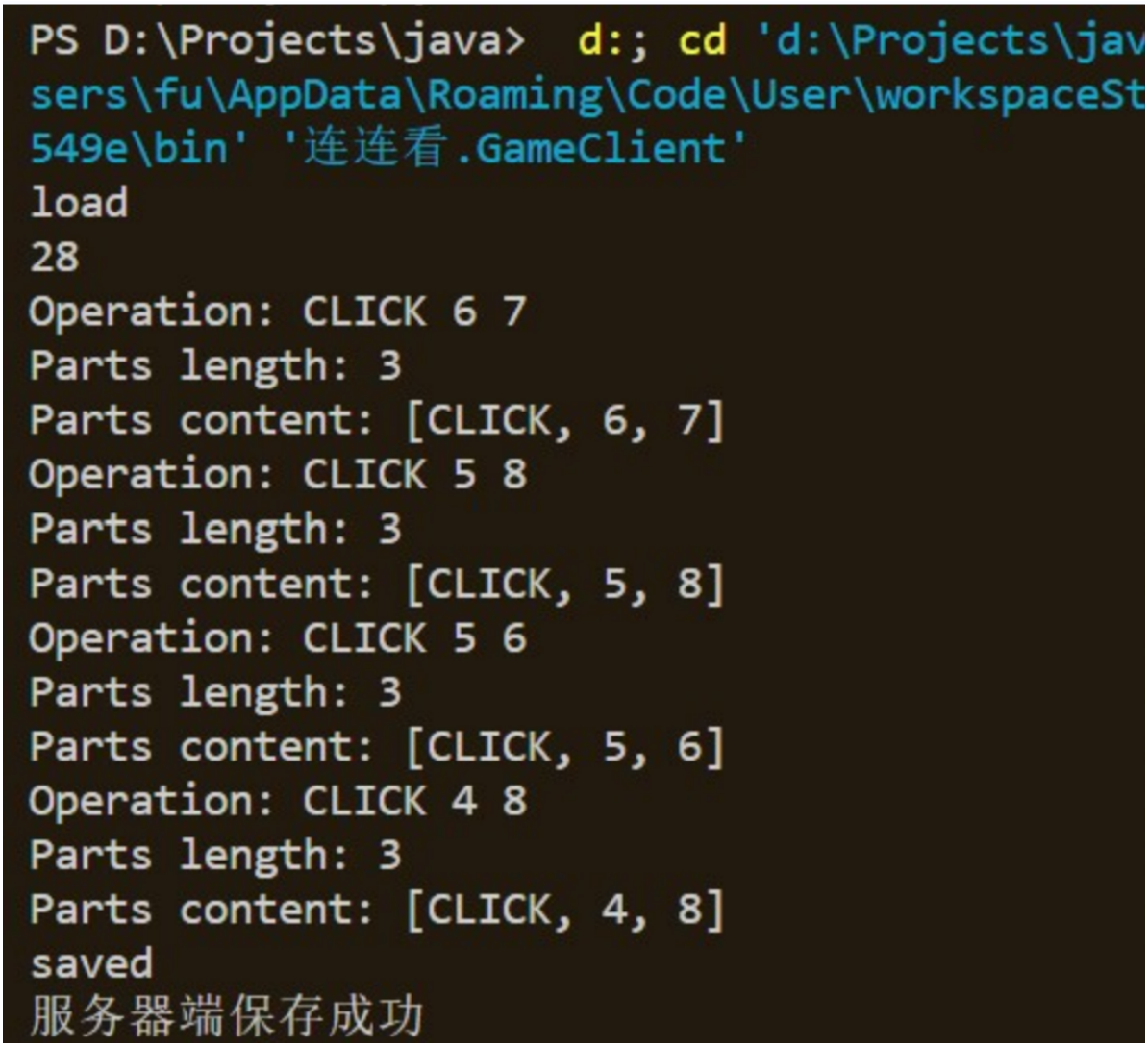
最后在`GameClient`类中调用主函数`main`即可运行游戏。

界面展示




连连看游戏
已消去方块数量：
0
重新开始
退出
replay
按D提示


连连看游戏
恭喜你完成了游戏!
再来一轮吧
是(Y)
否(N)
已消去方块数量:
100
重新开始
退出
replay
按D提示



不足与感想

不足：

1.游戏中未设计难度等级选择；

2.游戏中未设计计时器，使倒数结束时结束游戏；

3.游戏界面不能自适应屏幕大小，拉动边框就会导致界面在原处不动；

4.游戏未设计用户登录和注册功能，还有排行榜的功能；

感想： 通过这个项目，我深刻体会到了Java语言的强大和灵活性。 在制作连连看游戏的过程中，我首先进行了游戏界面的设计和布局。通过使用Java的Swing库，我能够轻松地创建各种图形界面组件，如按钮、标签和面板。这使得游戏界面看起来非常美观和直观。 接下来，我实现了游戏的逻辑部分。通过使用Java的面向对象编程特性，我创建了一个游戏类，其中包含了游戏的初始化、开始、结束等方法。我还创建了一个二维数组来表示游戏的棋盘，并使用随机数生成器来随机生成棋盘上的图案。通过使用循

环和条件语句，我能够实现游戏规则的判断和消除。 在整个制作过程中，我遇到了一些挑战和困难。例如，如何判断两个图案是否可以相连，如何实现图案的消除和移动等。但是通过不断地学习和思考，我最终找到了解决方案，并成功地实现了这些功能。 通过完成这个连连看游戏的制作，我不仅提高了自己的编程技能，还加深了对Java语言的理解和应用。我相信，通过不断地学习和实践，我将能够在Java编程领域取得更大的成就。

源代码

(https://github.com/adtureven/LianLianKan)

```
//MapFactory
package 连连看;

public class MapFactory {

    static int[][] map;
    public MapFactory(){

    }

    public static int[][] getMap(int n){
        map = new int[n][n];//生成n*n地图

        //初始化地图信息为空
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                map[i][j] = -1;
            }
        }
        return map;
    }
}
```

```
//Map
package 连连看;

import java.util.ArrayList;

public class Map {
    private final int[][] map;
    private final int count;
    private final int n;
```

```
    public Map(int count,int n){//一共有count种不同的图案,n行n列
        map = MapFactory.getMap(n);//获取n行n列的数组
        this.count = count;
        this.n = n;
    }
    public int[][] getMap(){
        ArrayList<Integer> list = new ArrayList<Integer>();//先将等量图片ID添加到
list中
        for(int i=0;i<n*n/10;i++){
            for(int j=0;j<count;j++){
                list.add(j);
            }
        }
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                int index = (int) (Math.random()*list.size());//从list中随机取一个
图片ID，并将其添加到数组中，再从list中删除掉它
                map[i][j] = list.get(index);
                list.remove(index);
            }
        }
        return map;//返回一个图片随机生成的地图数组
    }
    public int[][] getResetMap(){//获取再次打乱后的地图信息

        ArrayList<Integer> list = new ArrayList<Integer>();//list用来存储原先的地图
信息

        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(map[i][j]!=-1)//如果(x,y)处的图片ID不为-1，那么将该图片id添加到list
                    list.add(map[i][j]);
                map[i][j]=-1;
            }
        }
        //将原先地图上剩余的未消去的图片打乱
        while(!list.isEmpty()){
            int index = (int) (Math.random()*list.size());//从list中随机取一个图片
ID，并将其添加到数组中，再从list中删除掉它
            boolean flag = false;
            while(!flag){
                int i = (int) (Math.random()*n);//获取随机的地图行列
                int j = (int) (Math.random()*n);
                if(map[i][j]==-1){//如果该位置无图片
                    map[i][j] = list.get(index);
                    list.remove(index);
                    flag = true;
                }
            }
        }
        return map;
    }
}
```

```java
//GameState
package 连连看;

import java.io.Serializable;

public class GameState implements Serializable {
    private String count;
    private int[][] map;
    GameState(String count,int[][]map){
        this.count=count;
        this.map=map;
    }
    public String getCount(){
        return count;
    }
    public int[][] G_getMap(){
        return map;
    }
}
```

```java
//GamePanel
package 连连看;

import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Arrays;

import javax.swing.*;
import java.util.List;



public class GamePanel extends JPanel implements ActionListener
,MouseListener,KeyListener{

    private Image[] pics;
    private final int n;

    //private  int k=0;

    private int[][] map;

    private int[][] map1=new int[10][10];
    //private int[][] map2=new int[10][10];


    private final int leftX = 40;
    private final int leftY = 32;
    private boolean isClick = false;
    private int clickId,clickX,clickY;
```

```
    private int linkMethod;
    private Node z1,z2;
    private Map mapUtil;
    public int count = 0;
    public int count1 = 0;
    private int k=0;

    //private List<String> operationRecord;
    private List<String> operationRecord = new ArrayList<>();

    //public GameState gameState;


    public static final int LINKBYHORIZONTAL = 1,LINKBYVERTICAL =
2,LINKBYONECORNER = 3,LINKBYTWOCORNER = 4;
    public static final int BLANK_STATE = -1;

    public GamePanel(int count){
        setSize(600, 600);
        n = 10;
        mapUtil = new Map(count, n);
        map = mapUtil.getMap();
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                map1[i][j]=map[i][j];
            }
        }
        count1=count;

        this.setVisible(true);
        this.addKeyListener(this);
        this.addMouseListener(this);
        this.setFocusable(true);
        getPics();
        repaint();
    }
    public GamePanel(GameState gameState){
        setSize(600, 600);
        n=10;
        count=Integer.parseInt(gameState.getCount());
        count1=count;
        map=gameState.G_getMap();
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                map1[i][j]=map[i][j];
            }
        }

        this.setVisible(true);
        this.addKeyListener(this);
        this.addMouseListener(this);
        this.setFocusable(true);
        getPics();
        repaint();
    }
```

```
    private void getPics() {
        pics = new Image[10];
        for(int i=0;i<=9;i++){
            pics[i] = Toolkit.getDefaultToolkit().getImage("pic"+(i+1)+".png");
        }
    }

    public void paint(Graphics g){
        g.clearRect(0, 0, 800, 30);

        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(map[i][j]!=BLANK_STATE){
                    g.drawImage(pics[map[i]
[j]],leftX+j*50,leftY+i*50,50,50,this);
                }else{
                    g.clearRect(leftX+j*50,leftY+i*50,50,50);
                }
            }
        }
        k=1;

    }
    //水平方向的消除
    private boolean horizontalLink(int clickX1, int clickY1, int clickX2, int
clickY2) {

        if(clickY1>clickY2){
            int temp1 = clickX1;
            int temp2 = clickY1;
            clickX1 = clickX2;
            clickY1 = clickY2;
            clickX2 = temp1;
            clickY2 = temp2;
        }

        if(clickX1==clickX2){

            for(int i=clickY1+1;i<clickY2;i++){
                if(map[clickX1][i]!=BLANK_STATE){
                    return false;
                }
            }

            linkMethod = LINKBYHORIZONTAL;
            return true;
        }


        return false;
    }

    //垂直方向的消除
```

```
    private boolean verticalLink(int clickX1, int clickY1, int clickX2, int
clickY2) {

        if(clickX1>clickX2){
            int temp1 = clickX1;
            int temp2 = clickY1;
            clickX1 = clickX2;
            clickY1 = clickY2;
            clickX2 = temp1;
            clickY2 = temp2;
        }

        if(clickY1==clickY2){

            for(int i=clickX1+1;i<clickX2;i++){
                if(map[i][clickY1]!=BLANK_STATE){
                    return false;
                }
            }

            linkMethod = LINKBYVERTICAL;

            return true;
        }
        return false;
    }

    //一个折角的消除
    private boolean oneCornerLink(int clickX1, int clickY1, int clickX2, int
clickY2) {

        if(clickY1>clickY2){
            int temp1 = clickX1;
            int temp2 = clickY1;
            clickX1 = clickX2;
            clickY1 = clickY2;
            clickX2 = temp1;
            clickY2 = temp2;
        }

        if(clickX1<clickX2){

            if(map[clickX1][clickY2]==BLANK_STATE && horizontalLink(clickX1,
clickY1, clickX1, clickY2)&&verticalLink(clickX2,clickY2,clickX1,clickY2)){
                linkMethod = LINKBYONECORNER;
                z1 = new Node(clickX1, clickY2);
                return true;
            }


            if(map[clickX2][clickY1]==BLANK_STATE&&horizontalLink(clickX2,
clickY2, clickX2, clickY1)&&verticalLink(clickX1,clickY1,clickX2, clickY1)){
                linkMethod = LINKBYONECORNER;
                z1 = new Node(clickX2, clickY1);
                return true;
```

```
            }

        }else{

            if(map[clickX2][clickY1]==BLANK_STATE&&horizontalLink(clickX2,
clickY2, clickX2, clickY1)&&verticalLink(clickX1,clickY1,clickX2, clickY1)){
                linkMethod = LINKBYONECORNER;
                z1 = new Node(clickX2, clickY1);
                return true;
            }

            if(map[clickX1][clickY2]==BLANK_STATE&&horizontalLink(clickX1,
clickY1, clickX1, clickY2)&&verticalLink(clickX2,clickY2,clickX1, clickY2)){
                linkMethod = LINKBYONECORNER;
                z1 = new Node(clickX1, clickY2);
                return true;
            }

        }

        return false;
    }

    //两个折角的消除
    private boolean twoCornerLink(int clickX1, int clickY1, int clickX2, int
clickY2) {

        for(int i=clickX1-1;i>=-1;i--){

            if(i==-1&&throughVerticalLink(clickX2, clickY2, true)){
                z1 = new Node(-1, clickY1);
                z2 = new Node(-1, clickY2);
                linkMethod = LINKBYTWOCORNER;
                return true;
            }

            if(i>=0&&map[i][clickY1]==BLANK_STATE){

                if(oneCornerLink(i, clickY1, clickX2, clickY2)){
                    linkMethod = LINKBYTWOCORNER;
                    z1 = new Node(i, clickY1);
                    z2 = new Node(i, clickY2);
                    return true;
                }


            }else{
                break;
            }

        }

        for(int i=clickX1+1;i<=n;i++){

            if(i==n&&throughVerticalLink(clickX2, clickY2, false)){
```

```
                z1 = new Node(n, clickY1);
                z2 = new Node(n, clickY2);
                linkMethod = LINKBYTWOCORNER;
                return true;
            }

            if(i!=n&&map[i][clickY1]==BLANK_STATE){

                if(oneCornerLink(i, clickY1, clickX2, clickY2)){
                    linkMethod = LINKBYTWOCORNER;
                    z1 = new Node(i, clickY1);
                    z2 = new Node(i, clickY2);
                    return true;
                }

            }else{
                break;
            }

        }


        for(int i=clickY1-1;i>=-1;i--){

            if(i==-1&&throughHorizontalLink(clickX2, clickY2, true)){
                linkMethod = LINKBYTWOCORNER;
                z1 = new Node(clickX1, -1);
                z2 = new Node(clickX2, -1);
                return true;
            }


            if(i!=-1&&map[clickX1][i]==BLANK_STATE){

                if(oneCornerLink(clickX1, i, clickX2, clickY2)){
                    linkMethod = LINKBYTWOCORNER;
                    z1 = new Node(clickX1, i);
                    z2 = new Node(clickX2, i);
                    return true;
                }

            }else{
                break;
            }

        }

        for(int i=clickY1+1;i<=n;i++){

            if(i==n&&throughHorizontalLink(clickX2, clickY2, false)){
                z1 = new Node(clickX1, n);
                z2 = new Node(clickX2, n);
                linkMethod = LINKBYTWOCORNER;
                return true;
            }
```

```
            if(i!=n&&map[clickX1][i]==BLANK_STATE){

                if(oneCornerLink(clickX1, i, clickX2, clickY2)){
                    linkMethod = LINKBYTWOCORNER;
                    z1 = new Node(clickX1, i);
                    z2 = new Node(clickX2, i);
                    return true;
                }

            }else{
                break;
            }

        }


        return false;
    }



    private boolean throughHorizontalLink(int clickX, int clickY,boolean flag){

        if(flag){

            for(int i=clickY-1;i>=0;i--){
                if(map[clickX][i]!=BLANK_STATE){
                    return false;
                }
            }

        }else{

            for(int i=clickY+1;i<n;i++){
                if(map[clickX][i]!=BLANK_STATE){
                    return false;
                }
            }

        }

        return true;
    }

    private boolean throughVerticalLink(int clickX,int clickY,boolean flag){

        if(flag){

            for(int i=clickX-1;i>=0;i--){
                if(map[i][clickY]!=BLANK_STATE){
                    return false;
                }
            }
```

```
        }else{

            for(int i=clickX+1;i<n;i++){
                if(map[i][clickY]!=BLANK_STATE){
                    return false;
                }
            }

        }


        return true;
    }

    private void drawSelectedBlock(int x, int y, Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        BasicStroke s = new BasicStroke(1);
        g2.setStroke(s);
        g2.setColor(Color.RED);
        g.drawRect(x+1, y+1, 48, 48);
    }

    @SuppressWarnings("static-access")
    private void drawLink(int x1, int y1, int x2, int y2) {

        Graphics g = this.getGraphics();
        Point p1 = new Point(y1*50+leftX+25,x1*50+leftY+25);
        Point p2 = new Point(y2*50+leftX+25,x2*50+leftY+25);
        if(linkMethod == LINKBYHORIZONTAL || linkMethod == LINKBYVERTICAL){
            g.drawLine(p1.x, p1.y,p2.x, p2.y);

        }else if(linkMethod ==LINKBYONECORNER){
            Point point_z1 = new Point(z1.y*50+leftX+25,z1.x*50+leftY+25);
            g.drawLine(p1.x, p1.y,point_z1.x, point_z1.y);
            g.drawLine(p2.x, p2.y,point_z1.x, point_z1.y);

        }else{
            Point point_z1 = new Point(z1.y*50+leftX+25,z1.x*50+leftY+25);
            Point point_z2 = new Point(z2.y*50+leftX+25,z2.x*50+leftY+25);

            if(p1.x!=point_z1.x&&p1.y!=point_z1.y){
                Point temp;
                temp = point_z1;
                point_z1 = point_z2;
                point_z2 = temp;
            }

            g.drawLine(p1.x, p1.y, point_z1.x, point_z1.y);
            g.drawLine(p2.x, p2.y, point_z2.x, point_z2.y);
            g.drawLine(point_z1.x,point_z1.y, point_z2.x, point_z2.y);

        }

        count+=2;
        GameClient.textField.setText(count+"");
```

```
        try {
            Thread.currentThread().sleep(500);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        repaint();
        map[x1][y1] = BLANK_STATE;
        map[x2][y2] = BLANK_STATE;
        isWin();
    }


    public void clearSelectBlock(int i,int j,Graphics g){
        g.clearRect(j*50+leftX, i*50+leftY, 50, 50);
        g.drawImage(pics[map[i][j]],leftX+j*50,leftY+i*50,50,50,this);
    }

    public int[][]get_Map(){
        return map;
    }

    private boolean find2Block() {


        if(isClick){
            clearSelectBlock(clickX, clickY, this.getGraphics());
            isClick = false;
        }

        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){

                if(map[i][j]==BLANK_STATE){
                    continue;
                }

                for(int p=i;p<n;p++){
                    for(int q=0;q<n;q++){
                        if(map[p][q]!=map[i][j]||(p==i&&q==j)){
                            continue;
                        }

                        if(verticalLink(p,q,i,j)||horizontalLink(p,q,i,j)

 ||oneCornerLink(p,q,i,j)||twoCornerLink(p,q,i,j)){
                            drawSelectedBlock(j*50+leftX, i*50+leftY,
this.getGraphics());
                            drawSelectedBlock(q*50+leftX, p*50+leftY,
this.getGraphics());
                            drawLink(p, q, i, j);
                            repaint();
                            return true;
                        }
```

```
                }
            }
        }
    }

    isWin();

    return false;
}


private void isWin() {

    if(count==n*n){
        String msg = "再来一轮吧";
        int type = JOptionPane.YES_NO_OPTION;
        String title = "恭喜你完成了游戏！";
        int choice = JOptionPane.showConfirmDialog(null, msg,title,type);
        if(choice==1){
            System.exit(0);
        }else if(choice == 0){
            startNewGame();
        }
    }

}


public void startNewGame() {
    // TODO Auto-generated method stub
    count = 0;
    mapUtil = new Map(10,n);
    map = mapUtil.getMap();
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            map1[i][j]=map[i][j];
        }
    }
    count1=count;

    isClick = false;
    clickId = -1;
    clickX = -1;
    clickY = -1;
    linkMethod = -1;
    GameClient.textField.setText(count+"");
    repaint();
}

public void replay() {
    // 重置地图和计数器
    //resetMap();
    int count2=0;
    map=map1;
    count=count1;
```

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

        isClick = false;
        clickId = -1;
        clickX = -1;
        clickY = -1;
        linkMethod = -1;
        GameClient.textField.setText(count+"");

        paint(g);
        //revalidate();


        if(k==1){
            // 逐步执行操作记录
            for (String operation : operationRecord) {
                String[] parts = operation.split(" ");
                String action = parts[0];
                System.out.println("Operation: " + operation);
                System.out.println("Parts length: " + parts.length);
                System.out.println("Parts content: " + Arrays.toString(parts));

                int x = Integer.parseInt(parts[1]);
                int y = Integer.parseInt(parts[2]);

                if (action.equals("CLICK")) {
                    // 执行鼠标点击操作
                    if (count2 == 0) {
                        // 第一个点击，记录点击的图像编号和坐标
                        clickId = map[x][y];
                        clickX = x;
                        clickY = y;
                        isClick = true;
                        count2=1;
                    } else {
                        // 第二个点击，判断是否可以连接
                        if(clickId==map[x][y]){
                            if(x==clickX&&y==clickY)
                                continue;
                            if
(verticalLink(clickX,clickY,x,y)||horizontalLink(clickX,clickY,x,y)||oneCornerLi
nk(clickX,clickY,x,y)||twoCornerLink(clickX,clickY,x,y)) {

                                drawSelectedBlock(y*50+leftX,x*50+leftY,g);

 drawSelectedBlock(clickY*50+leftX,clickX*50+leftY,g);
                                drawLink(clickX, clickY, x, y);

                                map[clickX][clickY] = BLANK_STATE;
                                map[x][y] = BLANK_STATE;
                                isClick = false;
                                count2 = 0;
                                paint(g);
                                try {
```

```
                                Thread.currentThread().sleep(500);
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                        else{
                            count=1;
                            isClick = true;
                            clickId = map[x][y];
                            clearSelectBlock(clickX,clickY,g);
                            clickX = x;
                            clickY = y;
                            drawSelectedBlock(y*50+leftX,x*50+leftY,g);
                        }
                    }
                    else{
                        count=1;
                        isClick = true;
                        clickId = map[x][y];
                        clearSelectBlock(clickX,clickY,g);
                        clickX = x;
                        clickY = y;
                        drawSelectedBlock(y*50+leftX,x*50+leftY,g);
                    }
                }
            }
        }
    }
}
public static class Node{
    int x;
    int y;

    public Node(int x,int y){
        this.x = x;
        this.y = y;
    }
}
@Override
public void keyPressed(KeyEvent e) {
    // TODO Auto-generated method stub
    if(e.getKeyCode() == KeyEvent.VK_A){
        map = mapUtil.getResetMap();
        repaint();
    }
    if(e.getKeyCode() == KeyEvent.VK_D){
        find2Block();
        isWin();
    }
}
@Override
public void mousePressed(MouseEvent e) {
    // TODO Auto-generated method stub
    Graphics g = this.getGraphics();
    int x = e.getX()-leftX;
```

```
        int y = e.getY()-leftY;
        if(x<0||y<0||x>500||y>500)
            return ;
        int i = y/50;
        int j = x/50;
        String operation = "CLICK " + i + " " + j;
        operationRecord.add(operation);
        if(isClick){
            if(map[i][j]!=BLANK_STATE){
                if(map[i][j]==clickId){
                    if(i==clickX&&j==clickY)
                        return ;


 if(verticalLink(clickX,clickY,i,j)||horizontalLink(clickX,clickY,i,j)||oneCorne
rLink(clickX,clickY,i,j)||twoCornerLink(clickX,clickY,i,j)){
                        drawSelectedBlock(j*50+leftX,i*50+leftY,g);
                        drawLink(clickX,clickY,i,j);
                        isClick = false;
                    }else{
                        clickId = map[i][j];
                        clearSelectBlock(clickX,clickY,g);
                        clickX = i;
                        clickY = j;
                        drawSelectedBlock(j*50+leftX,i*50+leftY,g);
                    }
                }else{
                    clickId = map[i][j];
                    clearSelectBlock(clickX,clickY,g);
                    clickX = i;
                    clickY = j;
                    drawSelectedBlock(j*50+leftX,i*50+leftY,g);
                }
            }
        }else{
            if(map[i][j]!=BLANK_STATE){
                clickId = map[i][j];
                isClick = true;
                clickX = i;
                clickY = j;
                drawSelectedBlock(j*50+leftX,i*50+leftY,g);
            }
        }
    }
    @Override
    public void mouseClicked(MouseEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void mouseReleased(MouseEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void mouseEntered(MouseEvent e) {
        // TODO Auto-generated method stub
```

```
    }
    @Override
    public void mouseExited(MouseEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void keyTyped(KeyEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void keyReleased(KeyEvent e) {
        // TODO Auto-generated method stub
    }
}
```

```
//GameClient
package 连连看;

import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.net.Socket;

import javax.swing.*;

public class GameClient extends JFrame{
    GamePanel panel2 = new GamePanel(10);
    JButton button1 = new JButton("重新开始");
    JButton button2 = new JButton("退出");
    JButton button3 = new JButton("replay");
    static JTextField textField = new JTextField(10);
    private static final String SERVER_HOST = "localhost";
    private static final int SERVER_PORT = 8888;

    public GameClient(){
        JLabel label1 = new JLabel("已消去方块数量: ");
        JLabel label2=new JLabel("按D提示");
        JPanel panel = new JPanel(new BorderLayout());
        textField.setEditable(false);

        loadState();
        System.out.println(gameState.getCount());
        panel2=new GamePanel(gameState);
        textField.setText(gameState.getCount());

        panel2.setLayout(new BorderLayout());
```

```
        panel.setLayout(new FlowLayout());
        panel.add(label1);
        panel.add(textField);
        panel.add(button1);
        panel.add(button2);
        panel.add(button3);
        panel.add(label2);
        this.getContentPane().setLayout(new BorderLayout());
        this.getContentPane().add(panel,BorderLayout.SOUTH);
        this.getContentPane().add(panel2,BorderLayout.CENTER);
        this.setSize(600,630);
        //this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                saveState();
                dispose();
            }
        });

        this.setTitle("连连看游戏");
        this.setVisible(true);
        button1.setEnabled(true);
        button2.setEnabled(true);
        button3.setEnabled(true);

        button1.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                textField.setText("0");
                panel2.startNewGame();
            }
        });

        button2.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                saveState();
                System.exit(0);
            }
        });

        button3.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                textField.setText("0");
                panel2.replay();
            }
        });
    }

    private GameState gameState;
    public void saveState(){
        try{
            gameState=new GameState(textField.getText(), panel2.get_Map());
            FileOutputStream fileout=new FileOutputStream("gamestate.txt");
            ObjectOutputStream out =new ObjectOutputStream(fileout);
            out.writeObject(gameState);
```

```
            out.close();
            fileout.close();
            System.out.println("saved");
        }catch (Exception e){e.printStackTrace();}

        try (Socket socket = new Socket(SERVER_HOST, SERVER_PORT);
             ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new
ObjectInputStream(socket.getInputStream())
        ) {
            //gameState = new GameState(textField.getText(), panel2.get_Map());

            // 向服务器发送游戏状态数据
            out.writeObject(gameState);
            out.flush();

            // 接收服务器的响应
            String response = (String) in.readObject();
            System.out.println(response);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public void loadState(){
        try{
            FileInputStream filein=new FileInputStream("gamestate.txt");
            ObjectInputStream in=new ObjectInputStream(filein);
            gameState=(GameState)in.readObject();
            in.close();
            filein.close();
            System.out.println("load");
        }catch (Exception e){e.printStackTrace();}
    }

    public static void main(String[] args) {
        new GameClient();
    }

}
```

```
//GameServer
package 连连看;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
```

```
public class GameServer {
    private static final int PORT = 8888;

    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(PORT);
            System.out.println("服务器已启动，等待客户端连接...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("客户端已连接: " +
clientSocket.getInetAddress());

                // 创建一个新线程来处理客户端请求
                Thread thread = new Thread(new ClientHandler(clientSocket));
                thread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static class ClientHandler implements Runnable {
        private Socket clientSocket;

        public ClientHandler(Socket clientSocket) {
            this.clientSocket = clientSocket;
        }

        @Override
        public void run() {
            try (
                    ObjectOutputStream out = new
ObjectOutputStream(clientSocket.getOutputStream());
                    ObjectInputStream in = new
ObjectInputStream(clientSocket.getInputStream())
            ) {
                // 从客户端接收游戏状态数据
                GameState gameState = (GameState) in.readObject();

                // 处理游戏状态数据，可以将其保存到数据库或文件中

                FileOutputStream fileout=new FileOutputStream("server.txt");
                ObjectOutputStream out1 =new ObjectOutputStream(fileout);
                out1.writeObject(gameState);
                out1.close();
                fileout.close();

                // 发送响应给客户端
                out.writeObject("服务器端保存成功");
                out.flush();
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            } finally {
                try {
```

```
                    clientSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```