《黄金矿工》小游戏实验报告

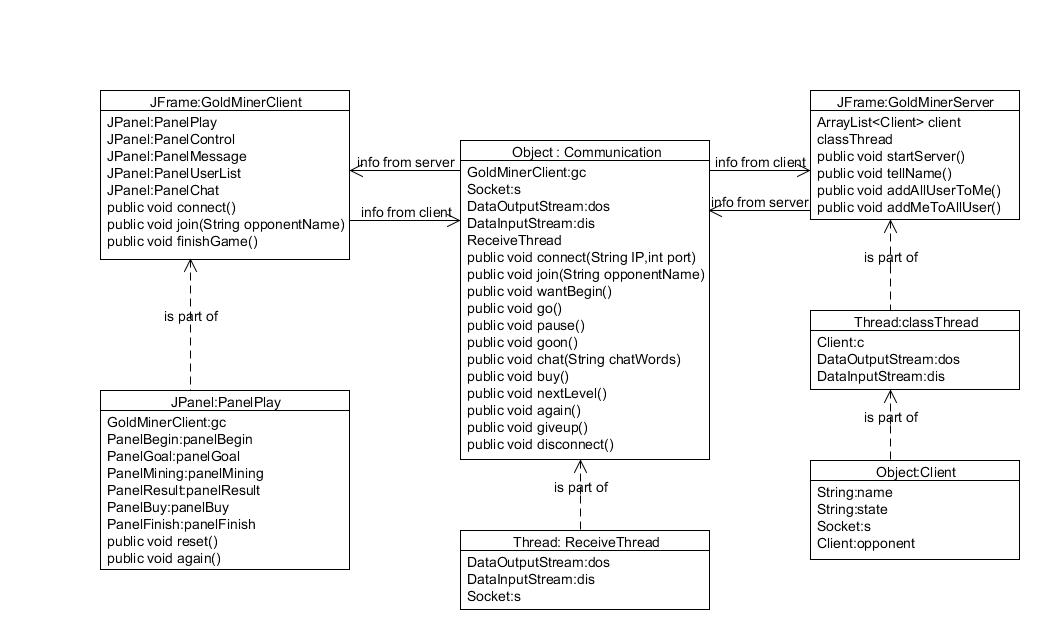
生65 林芳云 2016030040

## 一、类和模块设计

游戏中共出现三种类：

1. 窗体类JFrame,包括GoldMinerClient和GoldMinerServer，负责展现服务器、客户端界面以及提供程序运行的入口。
2. 信息类，包括Command类和Communication类。

Command类中包含了所有服务器和客户端间通讯时的消息字符串。

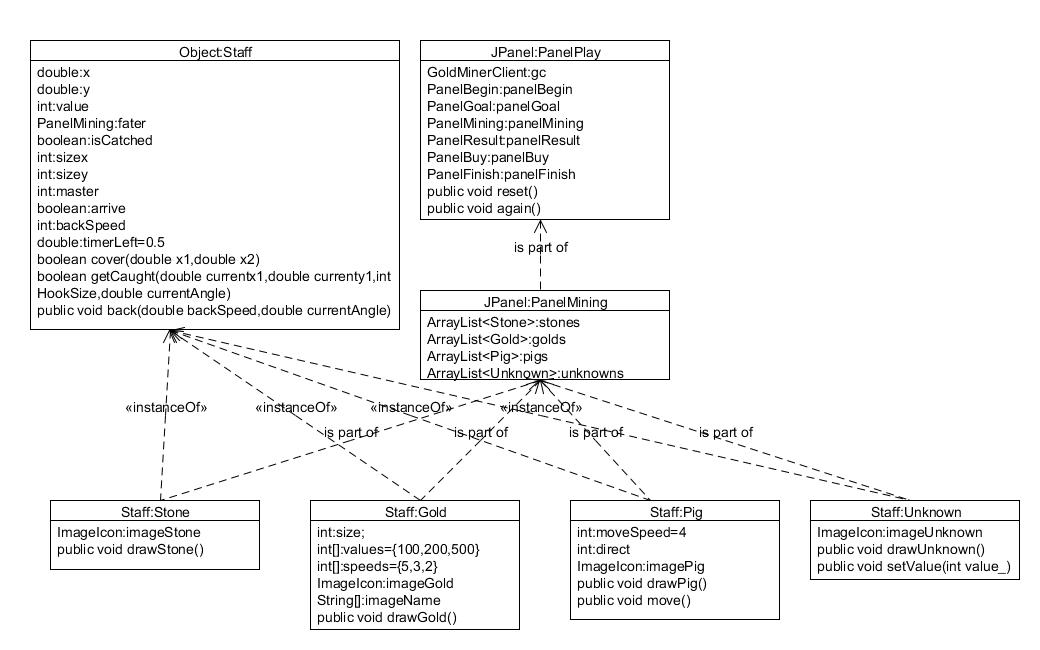
Comminication负责为服务器处理接收到的信息、帮助服务器向外发送信息。窗体类和信息类之间的关系如下：

图中仅列出部分关键的成员变量和成员函数。

1. 物品类

物品指的是所有在游戏中可能会被矿工挖到的物品。

物品类和窗体类的关系如下：

其中，istanceOf表示子类,is part of表示成员变量。

Staff表示一般的物品，Stone,Gold,Pig,Unknown都继承自它。

1. 其他

TimeDialog类：用于构造定时对话框。

Sound类：用于播放音乐

NewButton类：制作形状任意的按钮

这三个类与其他类关系并不密切，用途单一，故归为其他类。

## 二、功能点实现

### 1、自动生成物品布局

通过二维数组表示每关物品的坐标，第一维为关数，第二维为x或y坐标。

例如，石头的布局可以通过stoneX和stoneY两个二维数组实现（在PanelMining中定义）：

**int**[][] stoneX= {

{w/5,w\*4/5,w/2,w\*2/3},

{w/7,w\*8/9,w\*2/7,w\*4/9,w\*5/9,w\*7/8},

{w\*2/3,w\*2/3,w\*3/4,w\*7/8},

{w/3,w\*3/5,w\*9/10,w/20},

{w/7,w\*5/6,w/3,w\*37/60},

{w/20,w/4,w/6,w\*5/6},

{w/7,w\*8/9,w\*2/7,w\*4/9,w\*7/8},

};//石头的横坐标

**int**[][] stoneY= {

{h/3,h/3,h\*8/9,h\*2/3},

{h\*4/9,h/4,h\*16/27,h\*2/3,h\*2/3,h\*9/10},

{h/2,h\*3/4,h\*11/20,h\*3/4},

{h\*9/14,h/2,h\*9/10,h\*8/9},

{h\*3/5,h\*3/4,h/2,h\*3/4},

{h/2,h\*9/10,h/4,h\*8/9},

{h\*4/9,h/4,h\*16/27,h\*2/3,h\*9/10},

};//石头的纵坐标

金子、未知物的设置类似。对于猪，由于猪只从画面的两端进入并且在整关中y坐标不变，所以其坐标表示为：

**int**[][] pigX= {

{0,1,0},

{0,0,1},

{0,1,1},

{0,0,1},

{0,0,0},

{0,0,1},

{1,1,1}

};

其中0表示从左侧进入，1表示从右侧进入。

为了让游戏能不断玩下去，地图从第4关后开始每4关一循环。

**if** (level<=3)

{

**for** (**int** i=0;i<stoneX[level-1].length;i++)

{

Stone stone=**new** Stone(**this**,stoneX[level-1][i],stoneY[level-1][i]);

stones.add(stone);

}

//...省略其他物品设置

**else**

{

**for** (**int** i=0;i<stoneX[level%4+3].length;i++)

{

Stone stone=**new** Stone(**this**,stoneX[level%4+3][i],stoneY[level%4+3][i]);

stones.add(stone);

}

//...省略其他物品设置

}

### 2、按照重力摆动钩子

在PanelMining中添加成员变量Timer，每当游戏开始时启动，每过0.1秒更新钩子和x轴夹角，在paintComponent函数中重绘钩子。

定时器的实现：

timer=**new** Timer(100,**this**);

**public** **void** startTimer()

{

System.***out***.println("mine starttimer:"+System.*currentTimeMillis*());

startTime=**new** Date();//开始时间

timer.start();

timeLeft=60;

}

**public** **void** actionPerformed(ActionEvent e)

{

//调整钩子的角度

**if** (isBacking1==**false**&&isSearching1==**false**)

{

currentAngle1+=direct1\*Math.***PI***/40;

**if** (currentAngle1>=Math.***PI***||currentAngle1<=0)

direct1=-direct1;

}

**if** (isBacking2==**false**&&isSearching2==**false**)

{

currentAngle2+=direct2\*Math.***PI***/40;

**if** (currentAngle2>=Math.***PI***||currentAngle2<=0)

direct2=-direct2;

}

｝

每过0.1秒，钩子转动4.5°。当角度小于0或180°，向反方向转动。

### 3、计分界面

在paintComponent中添加：

g2.drawString("$"+panelPlay.score1, x1, y1+60);

g2.drawString("$"+panelPlay.score2, x2, y2+60);

这两句在人物的旁边画上他们的分数。（X1,y1）为玩家自己的坐标，（x2,y2）为伙伴的坐标。

在actionPerformed中添加：

**if** (isBacking1==**true**&&

currentx1<x1+20

&&currentx1>x1-20

&&currenty1<y1+20

&&currenty1>y1-20)

{

isBacking1=**false**;

**int** value1=loadValue(1);

panelPlay.score1+=value1;

currentLoad1="";

}

这句话用于更新panelScore1。isBacking1是一个boolean型变量，表示玩家是否正在撤回。后面的判断条件用于判断是否到达出发点。如果玩家正在撤回且达到了出发点，通过loadValue函数得到玩家手中物品的价值。如果没有物品，返回0。通过函数返回值更新panelScore1.

下面是loadValue函数的实现：

**public** **int** loadValue(**int** num)

{

String word=(num==1)?currentLoad1:currentLoad2;

String other=(num==1)?currentLoad2:currentLoad1;

**if** (word.equals(""))

**return** 0;

String[] words=word.split(":");

String[] otherwords=other.split(":");

String type=words[0];//种类

**int** i=Integer.*parseInt*(words[1]);//编号

**int** value = 0;

**if** (words[0].equals("gold"))

{

value=golds.get(i).value;

//修改对方的currentload编号

**if** (!other.equals("")&&otherwords[0].equals("gold"))

{

String otherType=otherwords[0];//对方的种类

**int** j=Integer.*parseInt*(otherwords[1]);//对方的编号

**if** (j>i)

{

**if** (num==1)

currentLoad2="gold:"+(j-1);

**else**

currentLoad1="gold:"+(j-1);

}

}

golds.remove(i);

}

//...省略对石头、未知物品、猪的相同操作

**return** value;

}

currentLoad1是一个字符串，格式为”种类：编号“。在玩家抓到一样物品的时候，物品的属性自动赋给currentLoad1，具体实现略。currentLoad2同理。

函数前半部分判断了当前回到原点的是己方玩家还是对方玩家、通过split函数拆分出字符串中的物品种类、物品编号，从而获取物品价值，最后将物品从ArrayList中删去。但是注意此时对方如果也正在拖曳一个物品，那么这个物品的编号就要改变了：如果被删去的物品在对方手中物品后面，那么没有影响；反之，编号-=1。

在物品到达原点时，在玩家身边显示物品价值0.5秒。

在paintComponent中添加：

**if** (timeScore1-timeLeft<=0.5&&timeScore1>=timeLeft&&dispScore1>0)

g2.drawString("$"+dispScore1, x1-50, y1);

**if** (timeScore2-timeLeft<=0.5&&timeScore2>=timeLeft&&dispScore2>0)

g2.drawString("$"+dispScore2, x2+50, y2);

dispScore1是玩家1返回出发点时手中的物品价值。在玩家抓住物品时自动获取。

timeScore1是玩家1身边开始显示这个物体价值时的剩余时间。在玩家返回原点时自动获取。

timeLeft是当前剩余时间。

注意：空手而归不需要显示，所以应当加上dispScore>0的判断。

### 4、倒计时界面

在paintcomponent中添加：

g2.drawString("Time:"+(**int**)timeLeft, 500, 20);

timeLeft是double型变量，表示当前时间，在timer启动时初始化为60（见前面startTimer函数），每次actionPerformed就减去0.1。

### 5、当前钱数、目标钱数界面

在paintcomponent中添加：

//画关数

g2.drawString("Level:"+level, 500, 40);

//画钱数和目标金钱

g2.drawString("Money:"+"$"+panelPlay.score, 20, 20);

g2.drawString("Goal:"+"$"+panelPlay.panelGoal.score,20,40);

### 6、键盘按键响应

新建键盘监听器keyProcessor

为客户端窗体添加键盘监听器。

panelPlay.gc.addKeyListener(keyProcessor);

KeyAdapter keyProcessor=**new** KeyAdapter() {

**public** **void** keyPressed(KeyEvent e) {

//我按下

**if** (e.getKeyCode()==KeyEvent.***VK\_DOWN***) {

//我放钩子

**if** (isBacking1==**false**)

{

**if** (isSearching1==**false**)

{

//通知对方我放钩子

panelPlay.gc.c.go();

//放音乐

}

}

}

}

};

isSearching判断是否在伸长钩子准备拉物体。如果按键方既没有在伸长钩子，也没有回撤钩子，就是在原地摆动，此时发送消息给服务器，服务器将这一消息通知对方和自己（为了保持同步）。

### 7、碰撞检测

#### 碰壁检测：

**if** (currentx2<=10||currentx2>=imageMining.getIconWidth()-10||

currenty2>=imageMining.getIconHeight()-10)

{

//回收钩子1

isSearching2=**false**;

isBacking2=**true**;

backSpeed2=5\*panelPlay.speedTime;

}

#### 碰到物体检测：

**for** (**int** i=0;i<stones.size();i++)

{

**if** (stones.get(i).getCaught(currentx1,currenty1,HookSize,currentAngle1)

&&stones.get(i).isCatched==**false**

&&isBacking1==**false**)

{

//...

isSearching1=**false**;

isBacking1=**true**;

backSpeed1=panelPlay.speedTime;

//...

}

}

Staff类的getCaught函数用于判断物品是否被抓到：

**boolean** getCaught(**double** currentx1,**double** currenty1,**int** hookSize,**double** currentAngle)

{

**double** tempx=currentx1+hookSize\*1.5\*Math.*cos*(currentAngle);

**double** tempy=currenty1+hookSize\*1.5\*Math.*sin*(currentAngle);

**if** ((tempx>=x //最左边

&&tempx<=x+sizex) //最右边

&&tempy>=y //最上面

&&tempy<=y+sizey) //最下面

**return** **true**;

**return** **false**;

}

其中(currentX1,currentY1)是钩子和线连接点的坐标，hookSize是钩子的大小，currentAngle是钩线和x轴的夹角，(x,y)是物品图片的左上角坐标，(sizex,sizey)是物品图片的宽和高。整个判断条件表示钩子是否进入了物品图片的矩形区。

不过这个判断条件对于较小的物品存在漏判的可能，可能钩子在0.1秒内从物品上穿过去了。但是目前没有遇到这种情况。

### 8、网络编程

大致思路：在客户端加入一个成员变量Commincation类，服务端加入一个线程类。

Communication接受客户端主动发出的消息，如连接服务器，放钩子，放弃游戏等；处理来自服务端的消息。

服务端的线程类接收来自客户端的消息并处理。

#### 客户端

##### ①客户端主动发送的消息

连接服务器：c.connect(IP, GoldMinerServer.***TCP\_PORT***);

邀请好友加入对战：c.join(opponentName);

在开始界面点击开始按钮：panelPlay.gc.c.wantBegin();

在游戏界面放钩子：panelPlay.gc.c.go();

在游戏界面点击暂停按钮：panelPlay.gc.c.pause();

在游戏界面点击继续按钮：panelPlay.gc.c.goon();

在购物界面买东西：panelPlay.gc.c.buy();

在购物界面点击进入下一关：panelPlay.gc.c.nextLevel();

在游戏结束界面选择重玩：panelPlay.gc.c.again();

在游戏结束界面选择退出游戏：panelPlay.gc.c.exit();

游戏过程中进行策略协商：c.chat(msg);

从服务器上退出：c.disconnect();

##### ②Communcation类对绑定的客户端的以上消息进行处理：

**public** **void** connect(String IP,**int** port)//为负责的客户连接指定端口和IP的服务器

{

**try** {

s=**new** Socket(IP,port);//创建一个和服务器此端口通信的本地socket

//创建接受消息的线程

**new** ReceiveThread(s).start();

dis=**new** DataInputStream(s.getInputStream());

dos=**new** DataOutputStream(s.getOutputStream());

} **catch** (UnknownHostException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

Communication类对于消息创建一个DataOutputStream类，用于向外输出消息；一个DataInputStream类，用于接收消息。通过dos.writeUTF()向外发送消息，通过dis.readUTF()向内读取消息。

为了区分不同类型的消息，需要在消息头部加上一个标示消息种类的字符串，例如用户通话时，在用户发送的消息前面加上”chat”，便不会和别的类型的消息混淆。所有这样用于标识的字符串放在Command类中。

##### ③Commincation将在头部添加了类型标示的消息传给了服务端。

##### ④Communication开一个线程ReceiveThread，接收来自服务端的信息并处理

**public** **void** run()

{

**while**(**true**)

{

**try** {

dis=**new** DataInputStream(s.getInputStream());

dos=**new** DataOutputStream(s.getOutputStream());

msg=dis.readUTF();

String[] words=msg.split(":");

//如果服务器告知了用户的名字

**if** (words[0].equals(Command.***TELLNAME***))

{

gc.myName=words[1];

gc.userList.userList.add(gc.myName+":ready");

gc.message.messageArea.append("My name:"+gc.myName+"\n");

}

//以下省略十多个else if

｝

通过dis接收来自服务器的消息并转换为字符串，通过split函数分隔出头部（头部是一种服务器为不同类型的消息加上的标示，在服务端中会介绍），根据不同的头部调用不同的消息处理函数。

在消息处理函数中，一般对客户端界面进行操作。例如，当服务端传来的消息的头部是Command.DELETE，标示要删除以头部后面的字符串为名字的客户。Commincation会通知和自己绑定的GoldMinerClient对象，将这个客户从客户端界面的UserList上删去。

#### 服务端

##### ①服务器启动

**public** **void** startServer()

{

**try** {

ss=**new** ServerSocket(***TCP\_PORT***);

**while**(**true**)

{

Socket s=ss.accept();

sockets.add(s);

*clientNum*++;

*clientNameNum*++;

Client c=**new** Client("Player"+*clientNameNum*,s);//客户的名字是"player1"的格式

clients.add(c);//将新客户加入客户列表

lStatus.setText("连接数："+*clientNum*);

//获取客户的IP地址

String msg=s.getInetAddress().getHostAddress()+"Player"+*clientNameNum*+"\n";

taMessage.append(msg);

tellName(c);//将名字告知给客户

addAllUserToMe(c);//将所有老客户信息通知新客户

addMeToAllUser(c);//将新客户信息通知所有老客户

**new** ClientThread(c).start();//为每个客户启动一个消息处理线程

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

ss是服务器类的ServerSocket类型的成员变量。每有一个客户端发送连接请求，服务器就通过accept函数连接一个新的客户端，创建一个新的Client对象用于保存客户信息，更新服务器界面，将服务器上所有的用户信息返回给新用户，将新用户的信息告知给所有老用户，同时新开一个线程ClientThread，用于处理这个客户端传来的所有请求。在main函数中调用startServer()函数，服务器开始工作。

##### ②ClientThread线程

这个线程用于处理所有客户端（实际上是Commication类）传来的消息，需要先将其读入并转换为字符串，然后根据头部确定消息类型，调用不同的处理函数。

**class** ClientThread **extends** Thread{

**private** Client c;//处理来自谁的消息

**private** DataInputStream dis;

**private** DataOutputStream dos;

**public** **void** run()

{

**while**(**true**)

{

**try** {

dis=**new** DataInputStream(c.s.getInputStream());

String msg=dis.readUTF();

String[] words=msg.split(":");

**if** (words[0].equals(Command.***JOIN***))//客户端邀请一名玩家

{

String opponentName=words[1];

**for** (**int** i=0;i<clients.size();i++)

{

**if** (clients.get(i).name.equals(opponentName))

{

dos=**new** DataOutputStream(clients.get(i).s.getOutputStream());

dos.writeUTF(Command.***JOIN***+":"+c.name);

}

}

}

//省略以下十多个else if

}

Client c用于保存该消息来源客户端的信息。

通过dis.readUTF()将字节流转换为字符串，通过split函数分割出头部，辨认消息的类型words[0].根据消息不同，调用不同的处理函数。

在处理函数中，一般有以下2种操作：向c的伙伴输出信息；向服务器上所有的用户广播信息（例如C从服务器端退出了，需要所有用户将其从UserList上删除）。

### 9、声音的实现

创建一个SoundPlayer类，主要实现通过文件名加载音乐功能loadSound(String filename)和播放音乐功能。

完整的类如下：

**public** **class** SoundPlayer {

File file;

AudioInputStream stream;

AudioFormat format;

DataLine.Info info;

Clip clip;

SoundPlayer(){

}

**public** **void** loadSound(String filename)

{

file=**new** File(filename);

**try** {

stream=AudioSystem.*getAudioInputStream*(file);

} **catch** (UnsupportedAudioFileException | IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

format=stream.getFormat();

}

**public** **void** playSound()

{

info=**new** DataLine.Info(Clip.**class**, format);

**try** {

clip=(Clip)AudioSystem.*getLine*(info);

} **catch** (LineUnavailableException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

clip.open(stream);

} **catch** (LineUnavailableException | IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

clip.start();

}

}

AudioInputStream类用于将文件转换为音频输入流。

AudioFormat用于获取音频格式。

DataLine.info用于设置音频行信息。

Clip clip建立音频行

Clip.open(stream)将音频数据读入音频行。

Clip.start()播放音频行。

### 10、定时对话框的实现

在用户邀请自己选中的伙伴时，伙伴的界面上会出现一个100秒的定时对话框，提示用户有人想和他一起挖矿。如果用户没有在100秒之内作出选择，默认为拒绝，这样可以避免有的用户中途离开了电脑但是没有退出服务器导致一直不响应请求的现象发生。

完整的类如下：

public class TimeDialog {

private String message=null;

private int seconds=0;//可思考时间

private JLabel label=new JLabel();

private JButton confirm;//接受邀请

private JButton cancel;//不接受邀请

private JDialog dialog=null;

int result=-5;//接受为0，,拒绝为1，超时为-5

public int showDialog(JFrame father,String message\_,int sec)

{

this.message=message\_;//玩家邀请您，您是否同意？

seconds=sec;

label.setText(message\_);

label.setBounds(80,6,200,20);

ScheduledExecutorService s=Executors.newSingleThreadScheduledExecutor();

confirm=new JButton("接受");

confirm.setBounds(100,40,60,20);

//如果按下接受键，result=0,对话框消失

confirm.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

result=0;

TimeDialog.this.dialog.dispose();

}

});

cancel=new JButton("拒绝");

cancel.setBounds(190,40,60,20);

//如果按下“拒绝”键，result=1,对话框消失

cancel.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

result=1;

TimeDialog.this.dialog.dispose();

}

});

//绘制对话框

dialog=new JDialog(father,true);//对话框是模态框，父窗口是father

dialog.setTitle("提示：本窗口将在"+seconds+"秒后关闭");

dialog.setLayout(null);

dialog.add(label);//玩家邀请您，您是否同意？

dialog.add(confirm);

dialog.add(cancel);

//runnable任务按指定的时间间隔重复执行

s.scheduleAtFixedRate(new Runnable() {

public void run()

{

TimeDialog.this.seconds--;

if (TimeDialog.this.seconds==0)

TimeDialog.this.dialog.dispose();//超时，对话框消失

else

dialog.setTitle("本窗口将在"+seconds+"秒后关闭");

}

},1,1,TimeUnit.SECONDS);

dialog.pack();

dialog.setSize(new Dimension(350,100));

dialog.setLocationRelativeTo(father);

dialog.setVisible(true);

return result;

}

}

ScheduledExecutorService s=Executors.newSingleThreadScheduledExecutor();

s.scheduleAtFixedRate(new Runnable() {

public void run()

{

TimeDialog.this.seconds--;

if (TimeDialog.this.seconds==0)

TimeDialog.this.dialog.dispose();//超时，对话框消失

else

dialog.setTitle("本窗口将在"+seconds+"秒后关闭");

}

},1,1,TimeUnit.SECONDS);

｝

这两句实现了定时功能。第一句提供了一个能够按照时间执行任务的类。第二句调用了这个类的方法，第一个参数是要执行的人物（在1之前的整个Runnable对象都是），第2个参数是首次执行的延迟时间，第3个参数是一次执行终止和下一次执行开始之间的延迟，第4个参数是上述两个参数的时间单位。这里，每1秒执行一次任务，任务是将剩余时间减1，如果剩余时间小于0，对话框消失。

### 11、NewButton的实现

NewButton是一种自定义按钮，其形状随着按钮上的图片改变。

具体实现如下：

**public** **class** NewButton **extends** JButton{

ImageIcon img;

**public** NewButton(String icon){

**super**();

**this**.img = **new** ImageIcon(icon);

setBorderPainted(**false**);

setContentAreaFilled(**false**);

setOpaque(**false**);

setSize(img.getIconWidth(),img.getIconHeight());

}

@Override

**public** **void** paintComponent(Graphics g){

**if**(**this**.getModel().isPressed())

{

g.drawImage(img.getImage(),1,1,**this**);

}

**else**

{

g.drawImage(img.getImage(),0,0,**this**);

}

**super**.paintComponent(g);

}

}

setBorderPainted(**false**); 按钮的边框不可见

setContentAreaFilled(**false**);按钮上不被图片覆盖的部分不可见

paintComponent中，当按钮被按下的时候，图片会往右下角有一个微小位移，模仿了按钮被按下的视觉效果。

### 12、读写文件

在客户端新建两个文件，分别用于存放该客户的单次成绩以及该客户参与过的所有双人游戏中最高五次对应的伙伴。

新建两个类SingleRecord和DoubleRecord,用于存放成绩记录。

SingleRecord类定义如下：

**public** **class** SingleRecord {

**int** score;

Date date;//游戏结束时间

**public** SingleRecord(**int** score\_,Date date\_)

{

score=score\_;

date=date\_;

}

**public** **int** getScore()

{

**return** score;

}

**public** Date getDate()

{

**return** date;

}

}

DoubleRecord比SingleRecord多一个opName变量用于存放伙伴的信息。直接继承SingleRecord类然后添加String opName变量即可。

在客户端新建两个列表ArrayList<SingleRecord> scoreSingleRecord和ArrayList<DoubleRecord> scoreDoubleRecord。

新建比较器，用于比较两条成绩记录的先后。按照成绩从高到低排，如果成绩相同，按照时间从前往后排。

**class** MyComparator **implements** Comparator<SingleRecord>{

@Override

**public** **int** compare(SingleRecord s1, SingleRecord s2) {

// **TODO** Auto-generated method stub

**if**(s1.getScore()-s2.getScore()!=0)

{

**return** s2.getScore()-s1.getScore();

}

**else**

{

**return** s1.getDate().getTime()<s2.getDate().getTime()?-1:1;

}

}

}

并通过Collections.sort(scoreSingleRecord,new MyComparator())将比较器绑定到上述链表。

每次游戏结束时，更新链表、更新文件。在PanelFinish中添加updateRecord函数

**public** **void** updateRecord()

{

//记录成绩

Date date=**new** Date();

SimpleDateFormat f=**new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

**try** {

date=f.parse(f.format(**new** Date()));

} **catch** (ParseException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

SingleRecord s=**new** SingleRecord(panelPlay.score1,date);

DoubleRecord d=**new** DoubleRecord(panelPlay.score,date,panelPlay.gc.opName);

panelPlay.gc.singleList.add(s);

panelPlay.gc.doubleList.add(d);

//更新文件

**try** {

PrintStream out=**new** PrintStream(panelPlay.gc.scoreSingleRecord);

**for** (**int** i=0;i<min(5,panelPlay.gc.singleList.size());i++)

out.println(panelPlay.gc.singleList.get(i).score+" "+panelPlay.gc.singleList.get(i).date+"\n");

out=**new** PrintStream(panelPlay.gc.scoreDoubleRecord);

**for** (**int** i=0;i<min(5,panelPlay.gc.doubleList.size());i++)

out.println(panelPlay.gc.doubleList.get(i).score+" "+panelPlay.gc.doubleList.get(i).date+" with "

+panelPlay.gc.doubleList.get(i).opName+"\n");

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

PrintStream将字符串写入文件。

由于用户的成绩记录可能不足5条，所以在刷新文件时输入条目的数量时用户成绩记录数和5的最小值。

在PanelFinish界面添加“Single Record”按钮和”Double Record”按钮。点击后弹出一个列举了用户单人成绩或双人成绩的txt文件。

弹出txt文件的功能可以通过如下操作实现：

Runtime.*getRuntime*().exec("C:\\WINDOWS\\system32\\notepad.exe single.txt");

表示启动notepad.exe程序打开single.txt文件。由于每个windows终端的notepad.exe默认路径都是相同的，所以绝对路径并不会失效。

## 三、操作说明

1、双击GoldMinerServer.jar，启动服务端

2、双击GoldMinerClient.jar，启动客户端。

3、双击客户端界面底部的“连接主机”按钮，请求连接服务器。

4、在客户端界面右侧顶部的UserList中可以看到所有在线玩家，单击其中一个，邀请其加入对战。

5、收到邀请的玩家的窗口弹出“XXX邀请您挖矿，您是否同意？”选择“否“，则另一方弹出对话框：“XXX拒绝了您的邀请”；选择“是”，双方的界面同时出现开始界面。

开始界面如图：



1. 一方点击“开始”按钮，则双方同时进入游戏。出现告知本关目标的界面，界面3秒后消失，出现挖矿界面。
2. 挖矿界面如图所示：



（游戏中界面还应当有猪，没有力气:10字样）

如果当前玩家是主动发起邀请的一方，则其坐在左边，操作左边的钩子；反之，其坐在右边，操作右边的钩子。

两方玩家通过按向下键放钩子。抓到物品后或碰壁后钩子返回，回到原点时人物身边会显示物品的价值0.5秒。

1. 本关结束后，出现结果通知界面。如下图所示：



如果分数不达标，会出现“你没有达到目标分”的字样

1. 结果通知界面出现3秒。如果过关，进入购物界面，如下图所示：



此时，双方玩家可以在画面右侧底部的chat here^对话框中进行协商是否买升力水。如果买水，需要花费154元，但是可以提升力气。当一方点击升力水或者“下一关”按钮后，进入下一关。

1. 如果双方分数总和不达标，出现结束界面。

玩家可以进行协商是否重玩。

任意一方点击Exit都可以退出游戏，但是仍然在线。

任意一方点击Again可以重玩，游戏进入告知分数的界面，从第一关重新开始。

一方点击“Single Record”可以查看当前自己的5次最高单人成绩。如果不足五次，则看到所有成绩从高到低、相同成绩按照日期从前到后排序。

一方点击“Doubld Record”可以查看当前自己和别人的5次最高成绩，如果不足5次，则看到所有成绩从高到低、相同成绩按照日期从前往后排序。



1. 点击客户端界面底部的“放弃游戏”按钮，可以中途退出游戏，另一方将自动结束游戏。
2. 点击客户端界面底部的“关闭程序”按钮，可以退出服务器。
3. 点击服务器底部的“关闭服务器”按钮，服务器将关闭，所有数据清空。