

# 游戏对战平台

[关键词] 聊天室 小游戏 JAVA

## 1 介绍

### 1.1 基本功能

聊天室功能：

提供用户之间的简单聊天通信，并且对于游戏得分进行排名，为用户提供联机游戏的平台。

实现游戏：

推箱子：一款家喻户晓的益智小游戏，玩家控制搬运工上下左右移动，来将箱子推到指定地点。

飞行射击：一款操纵飞行角色发射炮弹攻击敌机的小游戏。

对对碰：一款经典的消除类游戏，玩家只要通过点击人物来使人物之间互相换位，连成 3 个以上的人物来消除得分。

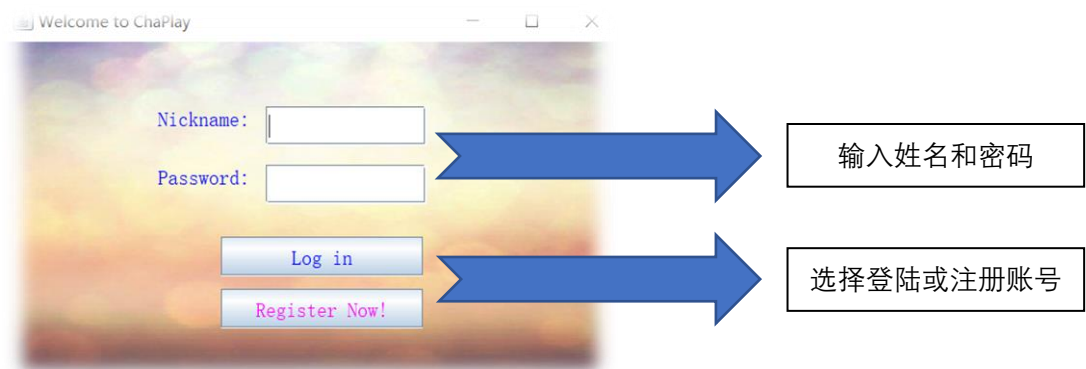
弹弹堂：一款回合制射击类竞技游戏，玩家双方发射炮弹攻击对方，直到一方血量为零。

## 2 平台原理

### 2.1 登录界面

#### 2.1.1 主要功能

登录界面类似 QQ 的登录界面，用户需要输入用户名和密码与服务器进行连接。其中服务器会进行判定是否重复登陆，用户名是否有效（即是否注册）等返回登陆状态信息。



如果首次登陆平台，需要点击注册（“Register Now!”）按钮跳转到注册界面

进行注册。

### 2.1.2 主要方法

`login_succ(String name, String password)`方法会根据用户输入的用户名和密码 and 数据库进行连接，进行用户名和数据库数据之间的配对，返回登陆的信息，判定是否登陆成功。

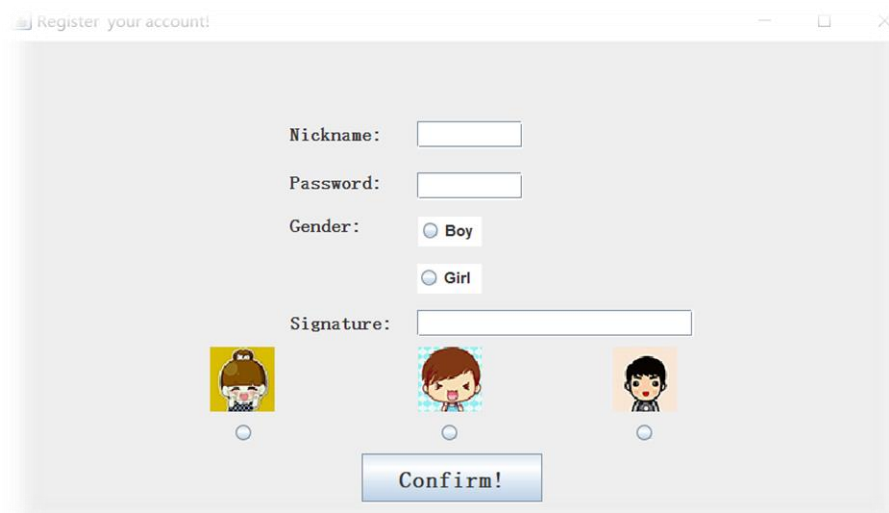
```
int login_succ(String name, String password);  
void log_in();
```

`log_in()`方法实现发起登陆的操作，首先建立用户的套接字接口与服务器进行连接，并可以得知服务器返回的连接情况，判定登陆的状态。

## 2.2 注册界面

### 2.2.1 主要功能

用户需要输入用户名、密码、性别、个性签名并且选择自己的头像。然后数据库建立对应表项完成注册。



### 2.2.2 主要方法

```
int register_succ();
```

`register_succ()`方法将用户输入的数据发送到数据库，完成比较“用户名”是否重复，各个栏目的填写内容是否超过规定长度的比较，并将信息反馈给用户。

## 2.3 平台主界面

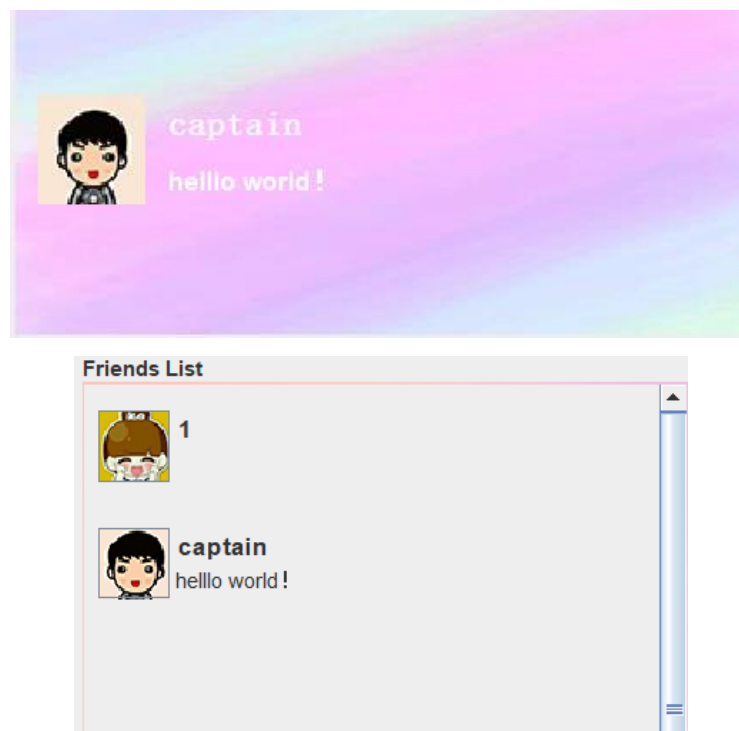
平台主界面是游戏平台的核心界面，其中分为 4 个主要部分：个人信息与好友栏、聊天室、游戏借口与排名和菜单栏。



## 2.3.1 个人信息与好友栏

### 2.3.1.1 主要功能

个人信息栏将呈现自己的注册信息，包括姓名和个性签名，在成功添加好友之后，好友列表中将加入好友的信息列表，点击好友头像便可以进行基本的聊天通信。



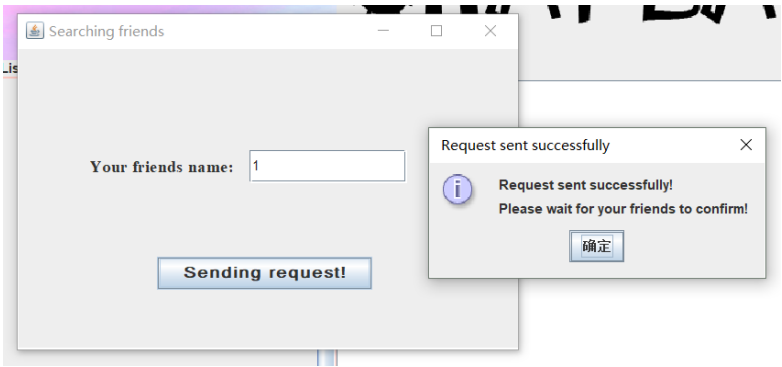
### 2.3.1.2 主要方法

```
public boolean friendsFind(String name);
public search_friends();
public requestItem(int pic, String name);
```

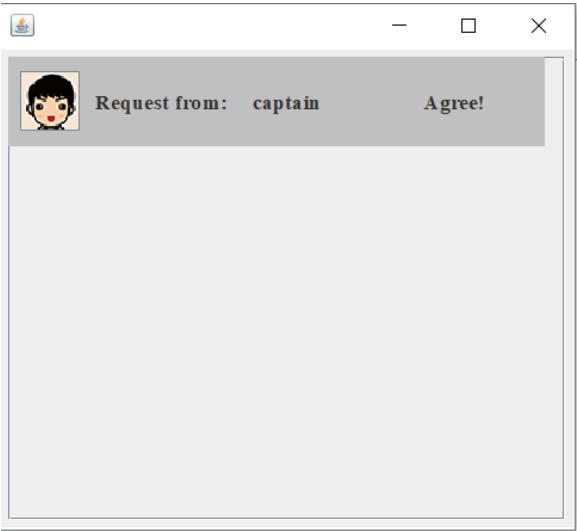
`public search_friends()`是 `public class search_friends` 的构造函数，其中将创建如下图所示的 `JFrame`，用户将通过姓名搜索好友。

`public boolean friendsFind(String name);`向数据库发起好友申请请求，数据库首先判断好友是否存在，如果存在将会通过服务器向目标用户发送好友申请请求，并且返回是否成功发送请求的信息。当发送请求成功时，将弹出下图所示的对话

框告知用户。



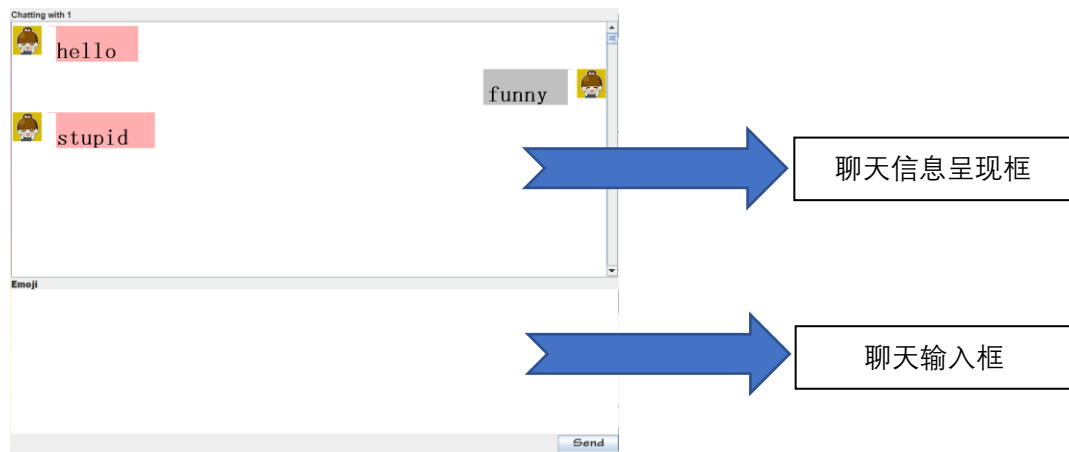
`public requestItem(int pic, String name);`是 `public class requestItem` 的构造函数，如果用户从服务器接受到好友申请的请求信息时，将在这一窗口显示待通过检验的好友的信息，用户可以选择“Agree”或“Refuse”完成申请。如果选择“Agree”将呈现下图所示的请求条目表明好友申请通过，接下来将在好友列表中新建好友条目呈现新好友的信息。同时，向服务器发送通过申请的信息，随后服务器将相应好友通过信息发送给发起申请的用户，并更新其好友列表条目和数据库相关好友数量信息。



## 2.3.2 聊天室

### 2.3.2.1 主要功能

用户通过点击好友列表的好友头像框选择进行聊天的好友，在信息呈现框将显示历史聊天记录，用户通过在信息输入框输入聊天信息，并点击“Send”按钮或“Ctrl”键发送消息。



### 2.3.2.2 主要数据结构与方法

```
public static ArrayList<RandomAccessFile> fileList = new ArrayList<>();
static class friendList extends person_info;
public void init_chat();
public static void setChat_window(int index);
public static void addTextMessage(String messages, int fla, int pic, int count);
public static int getStringHeight(String str, Font font);
public static int getStringWidth(String str, Font font);
JBubble(int height, int width, ArrayList<String> msg, int issell, Image img, Dimension d);
```

fileList 中为该用户和所有玩家的聊天记录文件，在 init\_chat()方法中初始化聊天界面（包括信息呈现框、信息输入框和发送按钮），并将 fileList 中的聊天内容读入到 friendList 中对应好友的 msgList 中记录消息，便于切换聊天对象时可以将聊天记录输出到信息呈现框中。

setChat\_window(int index)方法将用于切换聊天窗口，其消息呈现框清空，并将 friendsList 中对应好友 msgList 中聊天记录内容呈现在屏幕上。

用户通过在消息输入框的消息编辑，并点击“Send”按钮或“Ctrl”键发送信息。发送的消息记录在“friendsName.txt”的本地聊天记录文件中。

同时消息的发送采用气泡的形式发送并呈现在屏幕上。通过调用 getStringHeight(String str, Font font)和 getStringWidth(String str, Font font)方法获取发送消息的高度和宽度，然后调用 addTextMessage(String messages, int fla, int pic, int count)方法新建 JBubble（定义在 JBubble.java 中）对象，并加到屏幕呈现框中，完成消息的发送。

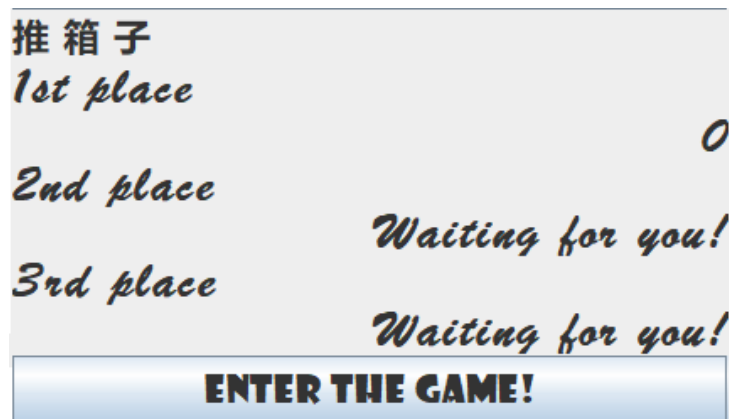
### 2.3.3 游戏接口与排名

#### 2.3.3.1 主要功能

在游戏接口这一段总共有 3 种信息：游戏名、游戏排名和游戏入口，下图为推箱子的入口的信息。其中目前的第一名为一名叫做“0”的玩家。

用户从平台接收排名的最近信息并且更新相应的排名的玩家名称。

提供进入游戏的按钮，用户点击按钮即可运行相应的游戏并且获得得分。



## 2.3.4 菜单栏

### 2.3.4.1 主要功能

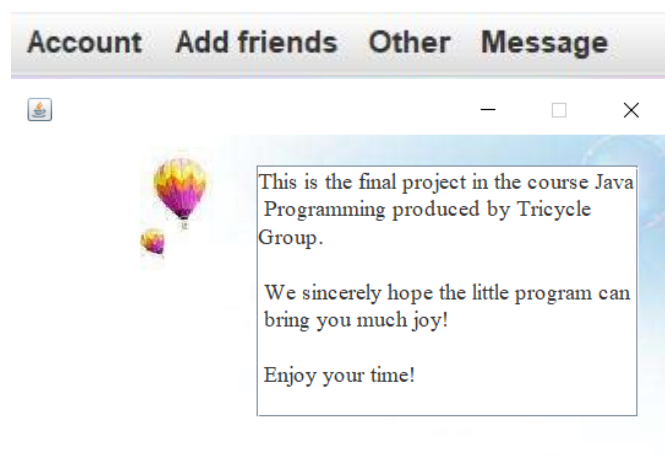
菜单分为一下 4 个部分：Account，Add friends，Other 和 Message，如下图  
所示。

Account 中提供切换账号和退出平台的入口。

Add friends 中提供添加好友的操作,进而调用 Search\_friends 中的相关方法。

Other 中有“关于”界面的接口，关于界面中简要的介绍我们小组和我们的  
project。

Message 中有 Friend request 接口，点击可以查看是否有玩家向我发送好友申  
请。



## 2.4 客户端

### 2.4.1 主要功能

客户端的主要功能是和服务端建议唯一的定向的输入输出对接，然后向服务  
器发送请求，进而进一步与其他用户进行交互。

## 2.4.2 主要数据结构与方法

```
private ServerSocket serverSocket;
private ExecutorService exec;
private Map<String, PrintWriter> storeInfo;
List<usrPoint> tuixiangzi;
List<usrPoint> ddt;
List<usrPoint> duiduipeng;
List<usrPoint> feiji;
public ChatServer();
private synchronized void putIn(String key,PrintWriter value);
class usrPoint;
private synchronized void sendToAll(String message);
private synchronized void sendToSomeone(String name, String message);
public void start();
class ServerClient implements Runnable
```

clientSocket将与服务器端口之间形成套接字连接,调用start(String nickname)方法,其中start方法有调用checkIfLog(String nickname)判断是否重复登陆,如果没有,则登陆成功,初始化pw、br,并通过pw向服务器发送信息,br从服务器中获取信息。

exec通过创建一个ListenerServer线程监听来自服务器的消息,并且解析信息。接受到的信息分为一下内容:

- 1) 发送好友申请: 以“Request”开头;
- 2) 同意好友申请: 以“Agree by”开头;
- 3) 发起弹弹堂联机对战申请: 以“CONNECT”开头;
- 4) 同意进行联机对战: 以“ACCEPT”开头;
- 5) 拒绝联机对战: 以“REFUSE”开头;
- 6) 发送实施游戏信息: 以“DDTANK”开头;
- 7) 发送聊天消息

客户端通过sendMsg(String fName, String msg)向用户fName发送消息msg,首先向服务器发送相应信息,然后服务器解析信息,将信息发送给fName。同时调用固定的sendGameMsg(String str)发送实时的游戏信息。

## 2.5 服务器

### 2.5.1 主要功能

服务器主要功能为接受来自客户端的信息,进行解析然后发送给其他用户,或者在本地进行存储;或者获取最新的游戏排名,发不给所有的玩家。

### 2.5.2 主要数据结构与方法

在构造函数ChatServer()中,初始化服务器套接字接口serverSocket,并调用start()开始监听客户端的连接。同时新建TimerTask,每隔3s更新一次游戏得分的排名。用户游戏的排名被存储在4个List当中,每次TimerTask中进行List的

得分排序，将前三名的玩家姓名通过 `sendToAll(String message)` 发送给所有的玩家。

在监听客户端的连接的时候，如果建立连接，则在线程池 `exec` 中建立 `ServerClient` 线程，监听来自客户端的信息，并协助客户端与其他用户的交互，服务器调用 `putIn(String Key, PrintWriter value)` 将用户的昵称和输出流建立键值对放入 `Map` 中。服务器解析的消息有以下几类：

- 1) 私聊信息：以 “@fName” 开头；
- 2) 判断好友是否上线：以 “\$\$\$” 开头；
- 3) 好友申请：以 “@friends request from” 开头；
- 4) 游戏联机申请：以 “CONNECT@” 开头；
- 5) 同意联机申请：以 “ACCEPT@” 开头；
- 6) 拒绝联机申请：以 “REFUSE@” 开头；
- 7) 实时游戏信息：以 “DDTANK@” 开头；
- 8) 游戏的分更新：以 “SCORE@” 开头

## 3 游戏原理

### 3.1 推箱子

#### 3.1.1 推箱子游戏介绍

推箱子是一款来自日本的古老游戏，目的是训练玩家的逻辑思考能力。在一个狭小的仓库中，要求玩家控制角色把木箱放到指定位置。

#### 3.1.2 推箱子功能

运行游戏载入相应的地图，屏幕中出现一名推箱子的工人，其周围是围墙、人可以走的通道、几个可以移动的箱子和箱子放置的目的地。玩家通过按上下左右键控制工人推箱子，当所有的箱子都推到了目的地后出现过关信息，并显示下一关。如果推错了，玩家通过单击鼠标右键可以撤销上次的移动操作。游戏界面如下所示：





### 3.1.3 推箱子实现原理

[整体结构]

```
public class Box_Pusher
{
    static class MapFactory {}
    static class Map {}
    static class GameFrame {}
}
```

[常量定义说明]

```
final byte BLANK = 0, WALL = 1, BOX = 2, BOXONEND = 3, END = 4,
MANDOWN = 5, MANLEFT = 6, MANRIGHT = 7, MANUP = 8, GRASS = 9,
MANDOWNONEND = 10, MANLEFTONEND = 11,
MANRIGHTONEND = 12, MANUPONEND = 13;
```

BLANK: 空格; WALL: 墙壁; BOX: 箱子; END: 目的地; BOXONEND: 箱子在目的地; MANDOWN: 朝向为下的角色; MANLEFT: 朝向为左的角色; MANRIGHT: 朝向为右的角色; MANUP: 朝向为上的角色; GRASS: 可以走动的通道; MANDOWNONEND: 朝向为下且在终点的角色; MANUPONEND: 朝向为上且在终点的角色; MANLEFTONEND: 朝向为左且在终点的角色; MANRIGHTONEND: 朝向为右且在终点的角色。

[具体实现]

核心代码说明:

MapFactory 定义了地图数据类。三维数组 `byte[][][] sampleMap`, 以此描述推箱子每个关卡的内容。

Map 定义了地图类, 用于保存当前的游戏状态, 保存人的位置和游戏地图当前状态。撤销启动时, 恢复地图的操作也通过这个类获取需要的人的位置、地图的当前状态和关卡数来实现。

GameFrame 是游戏面板类, 完成游戏的界面刷新显示, 以及相应鼠标键盘的

相关事件。以下为 GameFrame() 中核心函数的相关说明：

在构造方法 GameFrame() 中，调用 initMap() 方法来初始化本关游戏地图，清空悔棋信息列表 list。

initMap() 会调用 getMapSizeAndPosition() 方法获取游戏区域大小以及显示游戏的左上角位置，还会调用 getManPosition() 方法获取角色当前位置。

getPic() 加载要显示的图片。

grassOrEnd(byte man) 判断人所在的位置是通道还是目的地。

moveUp(), moveDown(), moveLeft(), moveRight() 是该游戏的主要逻辑，实现角色的上下左右移动。推箱子游戏的特殊性，决定了角色移动涉及 3 个位置，即角色的当前位置，角色要移动到的位置和箱子要到达的位置。以向上移动 moveUp() 为例进行说明：

向上移动涉及的 3 个位置为角色的当前位置 (map[row][column])，角色的上一位的位置 p1 (map[row-1][column]) 和上上一位的位置 p2 (map[row-2][column])。首先判断 p1 处的图像类型，这里的图像类型可能为 BOX BOXONEND WALL GRASS END。

- (1) 如果 p1 是墙 WALL，因为不能穿墙，所以不处理直接返回。
- (2) 如果 p1 是 BOX 或 BOXONEND，则判断 p2，如果 p2 是 END 或 GRASS：保存当前的游戏进度于 ArrayList 的 list 变量中，用于撤销；判断 p2 是否是目的地 END，如果是则 p2 状态设置为 BOXONEND，否则为 BOX；人前进一步，如果 p1 是 BOX，则 p1 状态设置为 MANUP，否则 p1 状态设置为 MANUPONEND；将角色刚才的位置设置成 GRASS 或 END；修改人的位置在 map 数组中的行坐标 row。
- (3) 如果 p1 是 GRASS 或者 END：保存当前的游戏进度于 ArrayList 的 list 变量中，用于撤销；判断 p1 是否是目的地 END，如果不是则 p1 状态设置为 MANUP，否则 p1 状态设置为 MANUPONEND；将角色刚才的位置设置成 GRASS 或 END；修改人的位置在 map 数组中的行坐标 row。

isFinished() 判断是否所有的箱子都在终点上，从而判断是否过关。

undo() 方法用于撤销，具体原理是往前查找 ArrayList 的 list 的信息进行还原地图。

nextGrade() 实现下一关的初始化，并调用 repaint() 方法显示游戏界面。

### 3.1.4 推箱子游戏中遇到的问题及解决

[问题] 在游戏制作的后期，为了增加游戏的美观度而增添了背景图片的绘制，但因此该游戏在进行过程中难免地出现了屏幕闪烁现象。

[解决方案] 双缓冲技术。

一个动画在运行的时候，如果图像的切换是在屏幕上完成的，并且图像切换频率高于一定值，则可能会造成屏幕的闪烁，消除动画闪烁现象的最佳方法是使用双缓冲技术。

双缓冲技术在屏幕外做一个图像缓冲区，事先在这个缓冲区内绘制图像，然后再将这个图像送到屏幕上，这样一来，虽然动画中的图像切换很频繁，但是双缓冲技术很好地避免了在屏幕上进行消除和刷新时候的处理工作所带来的屏幕闪烁情况。

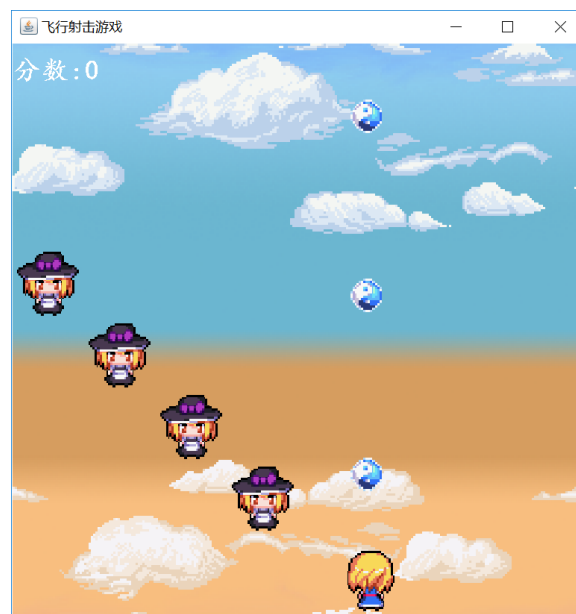
在该程序中的实现如下：

```
private Image iBuffer = null;
private Graphics gBuffer = null;
...
public void paint(Graphics g)
{
    gBuffer = iBuffer.getGraphics();
    //之后是通过 gBuffer 在 iBuffer 这一图像缓冲区上绘制图像
    g.drawImage(iBuffer, 0, 0, this); //最后把 iBuffer 画在屏幕上
}
```

## 3.2 飞行射击

### 3.2.1 飞行射击游戏介绍

飞行射击游戏，又称为雷电游戏，因其操作简单、节奏明快，作为纵轴射击的经典之作。玩家可以通过上下左右键盘控制自己的飞机移动，当玩家飞机的子弹碰到地方飞机，敌方飞机会出现爆炸效果，当玩家飞机被敌方飞机碰到，玩家飞机也会出现爆炸效果，并且游戏结束。飞行射击游戏的运行界面如下：



### 3.2.2 飞行射击游戏原理

[整体结构]

```
public class Thunder extends JFrame
{
    class Bullet{}
    class Enemy{}
    class GamePanel extends JPanel{}
    public Thunder(){}
}
```

```
}
```

[具体实现]

**Bullet** 是子弹类, 描述了单个子弹的坐标等属性, 且自带画图方法绘制。

**Enemy** 是敌机类, 描述了单个敌机的位置, 是否死亡等属性, 自带画图方法。

**GamePanel** 是游戏的主程序, 完成游戏的界面刷新显示, 以及相应键盘的相关事件。以下为 **GamePanel()** 中核心原理的相关说明:

有关敌机和子弹的实现: 游戏中使用到的敌机、子弹均采用对用的类实现。因为子弹的数量很多, 敌机的数量也会很多, 所以每一个子弹、每一个敌机都要用一个对象来实现。但是如果频繁地创建子弹对象与敌机对象, 这样会造成内存泄漏等严重问题。因此该游戏在 **GamePanel** 中定义了两个数组 **mEnemy[]** 和 **mBullet[]** 并且设定这两个数组的大小是 4 和 15, 意思为在屏幕上最多显示 4 个敌机和 15 个子弹。此外, 还维护了 **reserveBullet** 和 **flyingBullet** 两个 **ArrayList** 对象来实现子弹逻辑。如果某个敌机对象被子弹击中, 或者超出屏幕底边, 这时候可以对这个对象进行属性的重置, 让他重新出现在上方的战场上, 子弹对象同理。

**init()** 初始化分数、玩家飞机坐标等属性, 加载游戏所需要的图片, 创建敌机和子弹类对象并设置当前时间。

**reinit()** 用于重新开始游戏的情形, 重新初始化玩家的属性。

**updateBg()** 更新游戏逻辑, 实现了:

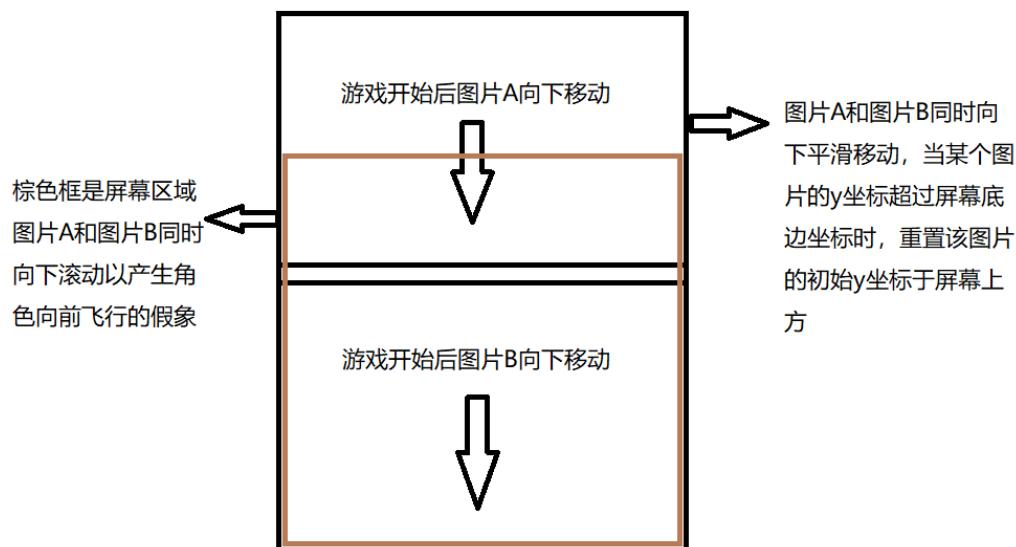
- (1) 背景图片滚动
- (2) 更新子弹的位置坐标
- (3) 确定是否要发射新的子弹
- (4) 更新敌机位置
- (5) 调用 **Collision** 进行碰撞检测

**UtilRandom(int botton, int top)** 生成 [botton, top) 的随机数, 用于指定刷新出的敌机初始横坐标。

**Collision()** 检测碰撞, 检测子弹是否和敌机碰撞以及自机是否和敌机碰撞, **Collision()** 会调用 **crash** 方法检测两个物体是否碰撞, **crash** 方法的原理很简单, 其可以判断图形中两个矩形是否相交。(这里没有采用像素碰撞检测是因为两种检测方法在该游戏中相差不大, 且矩形碰撞检测有着更好的性能)。

**emitBullet()** 发射炮弹, 每次发射之前都检测 **reserveBullet** 是否为空, 如果不是, 则从中取出一个对象放入 **flyingBullet**, 并且将该对象“发射”出去。**Thunder()** 进行 **JFrame** 的初始化工作。

### 3.2.3 地图滚动原理的实现



另外，具体的动画特效实现原理详见 3.3.4 动画实现部分，飞行射击、对对碰、弹弹堂的动画原理原则上是一样的。

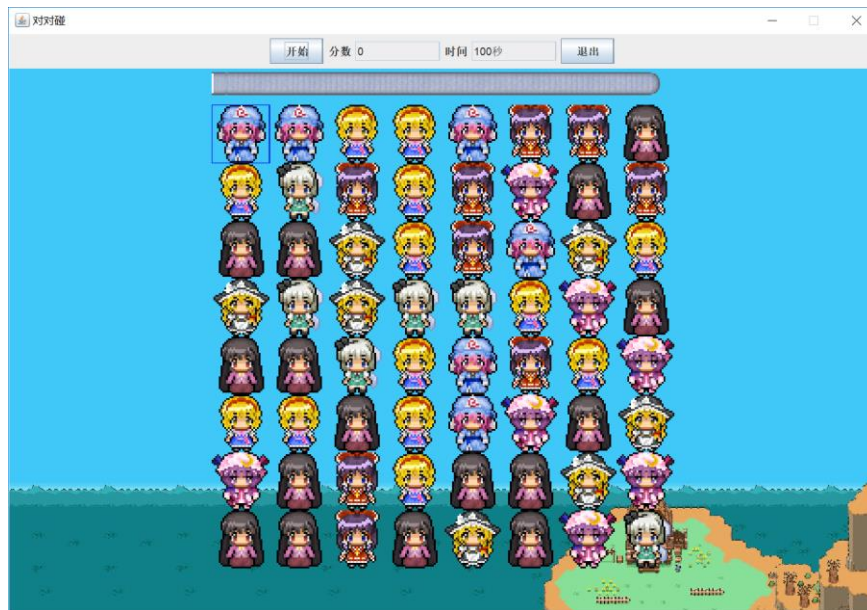
### 3.3 对对碰

#### 3.3.1 对对碰游戏介绍

对对碰是一款经典的消除类游戏，玩家只要通过点击人物来使人物之间互相换位，连成 3 个以上的人物来消除得分。

#### 3.3.2 对对碰游戏功能

只需要鼠标单击进行操作，玩家点击两个相邻的人物，使他们交换位置，如果交换位置后有三个以上的连续人物则可以将他们消除，消除的人物由上方的砖块补充，且如果消除人物补充完后某处依然有三个以上连续的人物，则可以继续消除（即连锁反应），消除人物越多，玩家分数越高。游戏界面如下图所示：



### 3.3.3 对对碰实现原理

[整体核心结构]

```
public class Touch_animation extends JFrame
{
    private JPanel backPanel = new JPanel();
    private RoomPanel roomPanel = new RoomPanel();
    private int pictures[][] = new int[matrixR][matrixC];
    private int cClicked, rClicked; //第一次单击按钮的坐标
    ...
    public Touch_animation() {}
    public void init() {}
    private void initImage() {}
    private boolean isThreeLinked(int y, int x, int[]kinds){}
    private void removeLinked(int y, int x, int kind){}
    private boolean globalSearch(int flag){}
    private void callAnimation() {}
    private void downPictures(){ }
    private void swapPictures(int x, int y){}
    private void testExchanging(){ }
    private void initPictureMatrix(){ }
    private void upsetPictureMatrix(){ }
    class RoomPanel extends JPanel{}
}
```

[具体实现]

游戏面板由一个 `JFrame` 的继承类和两个 `JPanel`（或其继承类）构成。`Touch_animation` 继承了 `JFrame` 构成面板底层,采用 `BorderLayout` 布局,`backPanel` 位于 `Touch_animation` 的 `NORTH`, 负责放置“开始”和“退出”按钮以及显示分数和

时间的 TextField。roomPanel 位于 Touch\_animation 的 CENTER 没有添加组件，仅仅用于绘制图片、人物和进度条。int[][] pictures 是该游戏的核心变量，它存储了游戏矩阵的信息，每个方格的数字代表其存储哪种人物，信息的表示是 0~6，共 7 种人物类型。

init()用于各项初始化，作用在游戏刚开始以及游戏重新开始的初始化。

initImage()加载背景图片、小人物图片以及动画特效的图片。

isThreeLinked(int y, int x, int[]kinds)判断(x,y)坐标处是否有 3 个以上的相同人物组，如果有则返回 true，否则返回 false，原理采用基础的横向/纵向顺序检索。另外，为了提高效率，需要在检测出可消除的行/列的同时，知道是消除一行还是一列，还是行和列。由于 Java 没有指针，因此这里直接采用数组 int[]kinds 来记录结果，只启用 kinds[0]，kinds[0] 为 1 表示横向消除，10 表示纵向消除 11 表示横纵都消除。

removeLinked(int y, int x, int kind)是在 isThreeLinked 被调用后并且返回 true 的基础之上使用的，作用是消除(x,y)坐标处三个以上的连续行/列甚至行列都消除。kind 作为输入，也就是 isThreeLinked 写入的 kinds[0]的值，指定是消除一行、一列还是行列都消除。

globalSearch(int flag)全局扫描人物矩阵，如果有消除点则返回 true，否则返回 false。flag 为 1 时仅仅扫描，flag 为 0 时，在扫描的基础上进行消除。靠调用 isThreeLinked 检测是否有消除点，调用 removeLinked 进行消除。

callAnimation()具体见 3.3.4 动画实现

downPictures()实现消除后“空白”处由上方的人物“下坠”补全。

swapPictures(int x, int y)实现鼠标点击相邻方块的交换。这里传入的参数 x 和 y 已经是第二次点击的坐标，第一次点击的人物横纵坐标会被存入 cClicked 和 rClicked 两个变量中，直到第二次点击面板时才会将点击坐标(x,y)作为参数调用 swapPictures。在该函数中，x 和 y 会先被转换成 cAbs 和 rAbs，即选中的人物横纵坐标，之后判断两次点击的人物坐标是否相邻，如果相邻则尝试交换并调用 testExchanging 判断交换后是否会产生消除。

testExchanging()测试交换结果，如果可以消除则进行消除。

initPictureMatrix()补全 pictures 矩阵的“空白”处，用随机数填上。

upsetPictureMatrix()将 pictures 矩阵所有格重置为某个随机数。

RoomPanel 覆盖了 JPanel 的 paint()方法，绘制背景、人物和进度条。

### 3.3.4 动画实现

这是本游戏比较特殊的地方，也是为 3.4 弹弹堂的实现奠定基础之处。核心相关字段和方法如下：

```
private boolean isAnimation; //是否处于动画状态
private boolean isExchanging; //是否处于交换状态
private boolean isExplosion; //是否处于爆炸状态
private boolean isFalling; //是否处于下降状态
private Timer animation; //动画的定时器

//爆炸特效有关字段
private BufferedImage explosionImage[]; //所有图片
```

```

private final int explosionCnt = 25; //一共 25 帧
private int explosionCur; //当前的帧数
private boolean explosion[][]; //爆炸矩阵

//交换动画有关字段
private final int changeCnt = 32;
private boolean changeBack;
private int changeCur; //已经交换的步数
private int sr, sc, dr, dc; //交换图标的坐标
private int pictureY[][] = new int[matrixR][matrixC];
private int pictureX[][] = new int[matrixR][matrixC];

private void callAnimation() {}

```

简要地说，`isAnimation = isExchanging | isExplosion | isFalling`(几个字段的含义见以上注释)。动画进行和游戏逻辑不冲突的关键在于：每次要进行游戏逻辑时都判断一下 `isAnimation` 是否为 `true`，如果不是则屏蔽游戏逻辑，直到动画结束，否则继续游戏逻辑。

爆炸特效的实现原理就是最基本的“动”画原理，按照一定频率在某个地方播放一段连续的帧产生动画错觉。这里的 `explosion` 矩阵用于指示哪些人物所在的位置被消除，即需要产生爆炸特效。爆炸特效的帧播放依靠于 `private Timer animation` 这一动画的定时器，该定时器会在爆炸特效被启动的时候启动，每隔 0.01 秒切换一帧，然后调用 `repaint()` 重新绘制面板，直到帧数被播放完。

交换动画的实现原理类似爆炸特效的基本原理，也是按照一定的频率，在图片移动路径上不断打印图片，从而实现两张图片相向“移动”到对方位置的视觉效果。这里的 `pictureY` 和 `pictureX` 存储了所有人物图片在 `roomPanel` 的 `y` 坐标和 `x` 坐标，`roomPanel` 的 `paint()` 函数会根据以上两个矩阵绘制每个人物，而图片交换动画的实现就是每隔一段时间更改“移动”的两张人物的 `y` 和 `x` 的值，从而实现动画效果。交换动画会在 `swapPictures` 中通过指定 `isExchanging=true` 以及 `sr,sc,dr,dc` 等字段来初始化和启动交换“动画”。

`callAnimation()` 是用来检测全局是否产生了消除，如果产生了消除，则设定 `isExplosion`、`isAnimation` 为 `true` 后启动爆炸动画。

## 3.4 弹弹堂

### 3.4.1 弹弹堂游戏介绍

一款回合制射击类竞技游戏，玩家双方发射炮弹攻击对方，直到一方血量为零，游戏结束。弹弹堂游戏原理实现灵感许多都是来自于以上的小游戏，例如：推箱子游戏中使用二维数组存储地图，弹弹堂也使用这一方法设定障碍地图；飞行射击游戏使用多线程实现背景滚动和子弹飞行，以及对对碰也使用了多线程来实现爆炸特效，这也用在了弹弹堂弹道绘制、角色动态和爆炸特效等方面。

### 3.4.2 弹弹堂游戏功能



该游戏有两个模式：联机对战和单击模式。联机对战：通过游戏大厅的联网和好友在线对战；单击模式：不需要联网，可以选择双人对战，也可以选择和电脑 AI 对战。

进入游戏后会显示菜单，有四个选项“双人对战”、“对战 AI”、“联机对战”、“退出游戏”，通过“↑”“↓”切换当前选项，“Enter”键确定选项。

游戏操作：

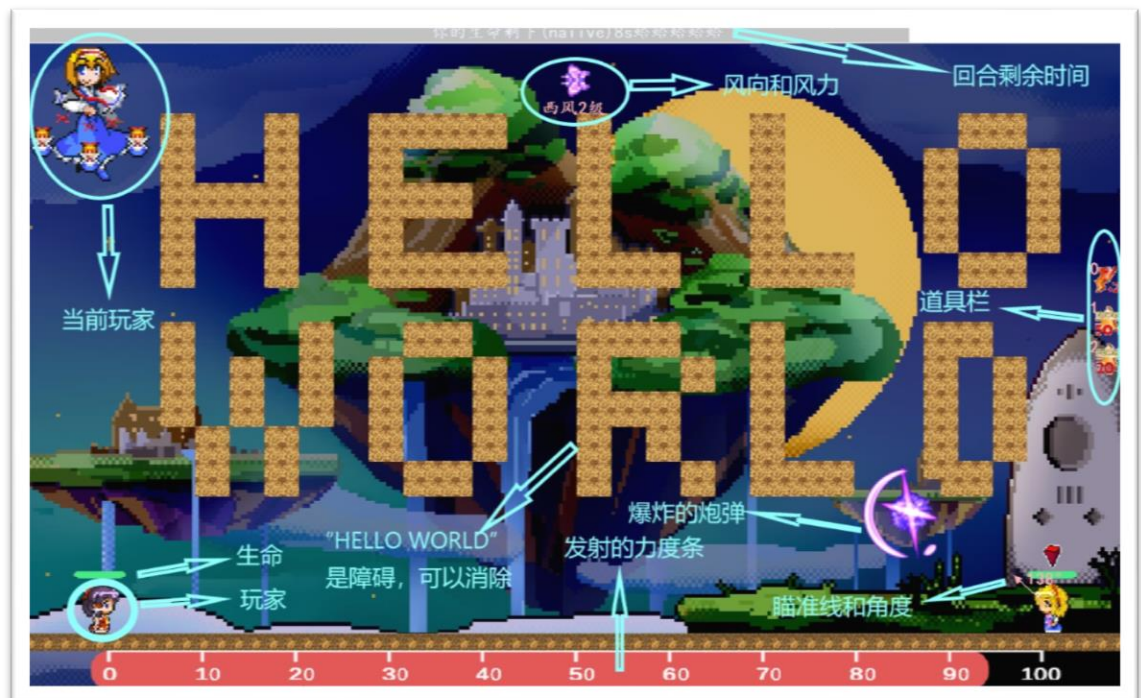
“←”、“→”控制角色左右移动

滑动鼠标控制瞄准角度

数字键 0~2 选择使用的道具

点击鼠标控制炮弹发射的力度，按住的时间越久，炮弹发射的距离越远。在最下面的“力度”刻度条，会显示当前空格操作所产生的力度。当不小心按过了应有刻度时应持续按下直到 100 时会向回反刻度此时按得时间越久炮弹发射越近。

游戏界面及相关说明如下图：



游戏相关属性：

#### 1. 角色属性

每个角色有体力、生命值和攻击力三个属性

每回合内，角色走动、使用道具都消耗一定的体力，体力耗尽无法走动或者使用道具。

#### 2. 时间限制

每回合的限定时间是 30 秒，超过时限自动进入下一回合，切换玩家。

#### 3. 风力系统

每回合随机生成一个方向的风力，风力会在水平方向上给子弹施加一定的加速度，从而影响炮弹的弹道轨迹，因此需要玩家注意好控制发射力度和发射角度。

### 3.4.3 弹弹堂实现原理

[核心字段、方法、类]（由于代码量过大，因此该游戏仅介绍关键部分）

```
public class DDtank extends JFrame
{
    private RoomPanel roomPanel = new RoomPanel();
    private final int frequency = 100; //刷新频率
    private Timer timer = new Timer(10,new TimeListener());//定时器
    ...
    private double calAngle = 0; //用于计算的角度
    private int power = 0; //投掷的力度
    ...
    private double bulletAngle;    //子弹的角度
    private double bulletX, bulletY;    //子弹的坐标
    private double bulletVx, bulletVy; //子弹两个方向的速度
    private final double accelerate = 10; //重力加速度
    private boolean bulletFlying = false; //子弹是否在飞
    private final double bias = 0.01; //偏差
    private BufferedImage curBullet; //当前子弹的帧
    ...
    private int turns = 0;    //轮到角色的编号
    private final int roleCnt = 2; //两个玩家
    private Role roles[] = new Role[roleCnt];
    ...
    private Maoyu maoyu; //定义一个 AI，名为毛玉
    ...
    private final int maxWindForce = 5; //风力最大五级
    private int curWindForce = 0; //当前的风力
    ...
    private boolean aiMode; //是否为人机对战
    private boolean netMode; //是否为联网对战
    private int myTurn; //联网对战时玩家被分配的编号（0 或 1）
    public static String strRead; //联网时接受到的 String
    private String playerName; //联网时指定的对方名字

    public DDtank() {}
    public void changeTurns() {} //更换玩家
    public void reinit() {} //重新初始化
    private boolean crash(int objx1, int objy1, int objWidth1, int objHeight1, int
objx2, int objy2, int objWidth2, int objHeight2) {}
    private void parabola() {} //抛物线相关计算
    private void loadMap() {} //加载地图
    //一些监听
    class MyKeyListener implements KeyListener {}
}
```

```

class MyMouseListener implements MouseListener {}
class MyMouseMotionListener implements MouseMotionListener {}
class TimeListener implements ActionListener {}
class RoomPanel extends JPanel {} //自定义面板类
...
private int calPower(double angle, int dx, int dy) //AI 计算力度
private void drawMenu(Graphics2D g2d) {} //绘制菜单
public void emitString(String mode, String str) {} //发送信息
private void parseMessage() {} //解析信息
private class Role {} //角色类
private class Walls {} //障碍类
private class Maoyu {} //名为 Maoyu 的 AI 类
private class SoundPlayer implements Runnable {} //播放音乐
private static class MapFactory {} //地图数据集
}

```

[具体实现]（该游戏按照游戏逻辑分块介绍）

核心部分只有一个底部窗体：

```
public class DDtank extends JFrame
```

和一个顶部面板：

```
private RoomPanel roomPanel = new RoomPanel()。
```

所有的 UI 界面都是使用 RoomPanel 的 paint()函数进行绘制的。

动画特效的帧实现依靠定时器：

```
private Timer timer = new Timer(10,new TimeListener())
```

和其对应的监听事件：

```

class TimeListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        ...//相关属性的计算
        repaint();//绘制游戏图形
    }
}

```

游戏的整体逻辑：

游戏开始运行之后，秒表开始倒计时，如果倒计时到 0 或者玩家发射的炮弹落地后，程序调用 changeTurns 切换玩家，changeTurns 会调用 reinit()重置角色的初始属性（不包括生命值），重新设定风力、重新初始化定时器等。直到有一方玩家生命值归零，游戏结束。

游戏动画逻辑：

利用定时器 timer 和其监听事件，每隔 0.01 秒计算一次游戏的相关属性并进行重新绘图。在 actionPerformed 进行对：回合倒计时、背景切换、角色力度条、体力条、瞄准线、角色位置、炮弹在空中的坐标、是否切换玩家等属性的计算。最后调用 repaint()对上述属性进行绘制。达到每 10ms 刷新一次屏幕的效果，每次绘制不同的帧，以此绘制出动画效果。

角色核心属性：private class Role {}

```

int strength; //体力值
int health;   //生命值
int damage;  //攻击力
int x, y;     //角色的坐标
int face;    //角色的朝向，朝左为-1,朝右为 1
BufferedImage curImage; //当前的帧
BufferedImage personImage[]; //人物的动作图片
BufferedImage figure;    //人物大图

public Role(int _id, int _x, int _y, boolean faceRight, int eI, int bI){} //人物的初始化部分
public void moveLeft(){} //向左移动一步
public void moveRight(){} //向右移动一步
//角色使用道具（一共三种道具）
public void buff_0(){}
public void buff_1(){}
public void buff_2(){}
public void fire(){} //角色发射炮弹
public void motion(){} //人物动态

```

角色可以进行左右移动、使用道具和发射炮弹。其中左右移动是靠 `moveLeft` 和 `moveRight` 实现，每次移动都改变角色的当前帧 `curImage` 和角色的坐标，从而产生角色“走动”的视觉效果。使用道具 `buff_x` 会改变角色的攻击力等属性，但是这些属性都会在 `reinit()` 中被重置回初始值。人物动态 `motion` 意思是长时间不操作角色，程序会让角色的当前帧 `curImage` 变成其他图片，换句话说，就是让角色产生其他的“小动作”，让角色看起来就是“活”的一样。角色的发射子弹详见下一段。

子弹的发射和飞行：

角色发射子弹前可以：通过移动鼠标来调整子弹的发射角度：`calAngle`（通过 `MyMouseMotionListener` 实现），长按鼠标左键控制子弹的发射力度：`power`（通过 `MyMouseListener` 实现，这里设定点击一次即代表开始蓄力，蓄力发射后直到下一回合再次点击左键不能再次发射）。角度规定是角色水平朝向的  $0^{\circ}$  到垂直  $90^{\circ}$ ，只能通过“转身”来完成向后发射子弹。另外，角色蓄力过程中不能移动。

长按鼠标左键后松开，会调用角色相应的 `fire()` 函数，实现角色发射子弹。`fire()` 函数会设置：`bulletFlying=true` 表示子弹处于飞行状态，`bulletX`、`bulletY` 的值表示子弹的初始坐标，`bulletVx`、`bulletVy` 的值表示子弹的水平和垂直方向的初速度。这以后，之前提到的 `actionPerformed` 函数会检测到 `bulletFlying==true`，从而进行抛物线相关属性的计算（这会调用 `parabola` 进行相关计算）。

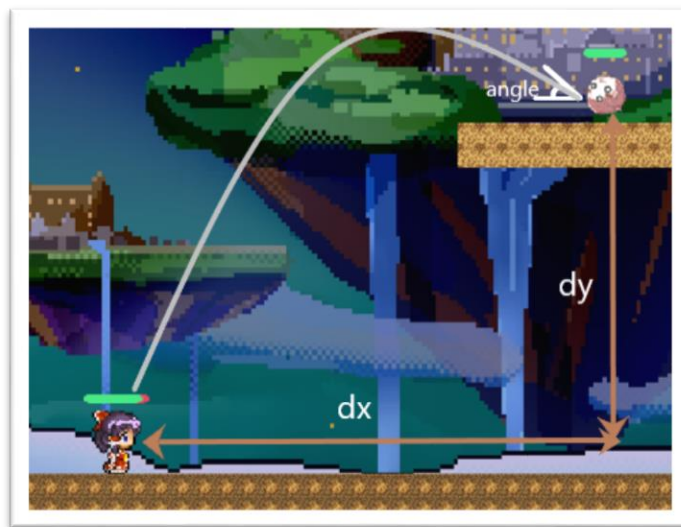
`parabola()` 方法进行抛物线和子弹有关属性的计算，包括子弹下一帧的水平、垂直方向速度，横纵坐标，还会计算子弹朝向和水平方向的角度 `bulletAngle`，从而据此选定相应的子弹图 `curBullet`。最后是调用 `crash` 方法检测子弹是否撞到人或障碍物。这里的 `crash` 原理和第二个游戏：飞行射击游戏中 `crash` 方法的原理一样，这里不再赘述。如果检测到碰到角色，则扣除相应角色的血量并检测剩余血量是否小于 0，如果小于 0 则游戏结束。

游戏 AI 原理:

这个游戏 AI 的计算核心无非是: 根据两个坐标, 如果在攻击范围内, 给出正确的角度和力度。相关的类就是:

```
private class Maoyu
{
    ...//基本属性和角色类似
    public Maoyu(int _x, int _y, int _xmin, int _xmax, int eI, int bi){}
    public void motion(){}
    public void AI(){}
    public void fire(){}
}
```

除了 AI()方法, 其他都和 Role 中的定义类似。如果碰到下图的情况 (AI 就是右上角的小球, 这里要的是根据角度 angle 计算小球攻击小人的力度):



设横纵坐标初速度为 $V_x$ 和 $V_y$ 加速度为  $g$ :

$$\text{时间 } t \text{ 满足: } \frac{1}{2}gt^2 - V_y \times t = dy$$

$$\text{因此: } t = \frac{V_y + \sqrt{V_y^2 + 2g \, dy}}{g}$$

$$\text{同时满足: } V_x \times t = dx$$

$$\text{因此得出: } V_x \times \frac{V_y + \sqrt{V_y^2 + 2g \, dy}}{g} = dx,$$

$$\text{另一方面: } V_x = power \times \cos(angle), V_y = power \times \sin(angle)$$

将这些带入方程直接计算太繁琐, 因此直接采用让 power 从 1 到 100 的遍历

循环计算  $\frac{V_y + \sqrt{V_y^2 + 2g \, dy}}{g}$  是否与  $dx$  相差在一个范围内, 如果是则返回该 power。

为了简便，这里只把 45°和 135°纳入考虑，程序的实现如下：

```
public void AI()
{
    int dx = x - roles[0].x + (roles[0].width / 2);
    int dy = roles[0].y - y + roles[0].height;
    if (dx > 0)
    {
        power = calPower(leftAngle, dx, dy);
        calAngle = leftAngle;
    }
    else
    {
        power = calPower(rightAngle, dx, dy);
        calAngle = rightAngle;
    }
    fire();
}
```

```
private int calPower(double angle, int dx, int dy){
    .....
    double u = Math.abs(Math.sin(angle));
    double v = Math.abs(Math.cos(angle));
    if (dy > 0)
        for (power = 1; power < 100; power++){
            tmp = power*u + Math.sqrt(power*power*u*u+20*dy);
            tmp *= (power*v);
            if (Math.abs(tmp - Math.abs(10 * dx)) <= bias)
                return power;
        }
    .....//其他情况
}
```

游戏联网：

联网整体逻辑：玩家在菜单中选择“联机对战”选项后，输入对战对方的名字，发送“CONNECT@<对方玩家名字>@”给服务器后，等待服务器为两个玩家搭建连接以及分配角色编号(0或1)。玩家这边收到角色编号之后将其赋值给 myTurn，并设置 netMode=true。

玩家发送信息通过 public void emitString(String mode, String str)，接收信号则通过读取 public static String strRead 实现。

信号协议如下：

CONNECT@<对方玩家名字>@ //向服务器发送请求连接

DDTANK@<对方玩家名字>@<游戏操作> //表示发送到对方程序的操作

SCORE@<游戏名，这里为 DDTANK>@<玩家分数> //发给服务器统计分

数

如果当前对局轮到玩家 (`turns == myTurn`), 则每次玩家的鼠标移动、左右移动、道具使用、发射力度的信息编码成“<游戏操作>”通过 `emitString` 发送到服务器既而传达给对方玩家的程序。如果当前对局是轮到对方玩家 (`turns != myTurn`), 则该玩家的程序隔一段时间就读取一次 `strRead` 直到 `strRead != null`, 然后调用 `parseMessage` 解析其中的<游戏操作>, 然后运行相关函数同步本机玩家的行为。

游戏地图:

`private class Walls{}` 定义了一个方块障碍的属性, 包括是否可消除、是否已经被消除、所在的横纵坐标以及加载的图片等属性。

`private static class MapFactory{}` 的功能和第一个游戏推箱子中的 `MapFactory` 一样, 都是存储地图数据集, 不再赘述。

`loadMap()` 方法能在游戏开始前加载地图, 从而打印出不同的地图样式, 如该游戏样例图中的“HELLO WORLD”的障碍形状。障碍分两种类型, 在 `MapFactory` 的三维数组 `map` 中, 如果值为 1 代表该障碍可以被消除, 如果为 2 代表不能消除, 如果为 0 表示没有障碍。

## 4 数据库搭建

在本次作业中, 我们使用了 MySQL Sever5.5 作为数据库管理系统, 以 `mysql-connector-java-5.1.46-bin.jar` 作为连接数据库的连接器, 在服务器端建立名为 `javaproject` 的数据库以及名为 `user_table` 的表格管理用户信息, 通过 Java 语言访问数据库, 以达到对于用户信息的增删改查, 通过使用 Java 语言中的接口与类, 客户端程序可以访问需要的信息。

### 4.1 MySQL 下载与配置

(1) 在 <https://dev.mysql.com/downloads/> 上下载 MySQL Community Server5.5, 选择 MSI Installer 进行下载, 双击下载得到的 .msi 文件进行安装配置, 其中设置密码为 java2018。

(2) 将安装路径下的 MySQL 文件夹中的 MySQL Server5.5 文件夹中的 bin 的路径添加到 Path 环境变量, 例如: `C:\Program Files\MySQL\MySQL Server 5.5\bin`。

(3) 打开命令行, 输入 `mysql -u root -p` 回车后输入密码 java2018 进入 mysql, 输入: `CREATE DATABASE javaproject;`

创建名为 `project` 的数据库, 可以通过 `SHOW DATABASES;` 检查是否创建成功

输入: `USE javaproject;`

操作 `javaproject` 数据库。

(4) 在上一步骤后输入:

```
CREATE TABLE user_table(  
-> name VARCHAR(30) NOT NULL,  
-> password VARCHAR(30) NOT NULL,  
-> gender INT,  
-> pic INT,  
-> friends_num INT,
```

```

-> signature VARCHAR(100) NOT NULL,
-> friends_name VARCHAR(100) NOT NULL,
-> boxscore INT,
-> thunderscore INT,
-> touchscore INT,
-> ddtankscore INT,
-> PRIMARY KEY(name)
-> )ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

便可创建名为 user\_table 的表格，可通过 SELECT \* from user\_table;来访问表内数据。

## 4.2 代码部分

### 4.2.1 重要变量

name:用户名;  
 password:密码;  
 gender:性别;  
 signature:个性签名;  
 pic:头像;  
 friends\_num:好友数量;  
 friends\_name:好友名字;  
 boxscore:推箱子积分;  
 thunderscore:雷电积分;  
 touchscore:连连看积分;  
 ddtankscore:弹弹堂积分

变量都为私有变量，访问与修改需使用该类所提供的方法，其中因为朋友名字在数据库中的存储形式为逗号分隔的字符串组成，所以有专门给数据库用的两个函数，getstrfriends\_name()与 setstrfriends\_name(String str)，分别用于将好友名字数组 friends\_name 转换为字符串和将数据库中的字符串加载进入数组 friends\_name。

### 4.2.2 重要函数

getConn(): 加载驱动，打开与数据库连接  
 closeAll(): 关闭连接  
 executeQuery(String prepareSql,String []param):接收准备好的 SQL 语句与参数数组，进行对数据库的查询  
 executeUpdate(String prepareSql,String []param): 接收准备好的 SQL 语句与参数数组，进行对数据库条目的修改  
 user\_data.java 用于向平台需要访问数据的地方提供方法，是 data\_base 类的子类，为 data\_base.java 中的函数准备好 SQL 语句与参数数组，提供了五个函数：  
 getUser(): 此函数将数据库中所有的条目读出，创建一个用户列表，该列表可以便于管理员端管理平台的全部用户  
 getUserByName(String name): 通过用户名 name 查询用户，返回一个 person\_info 类的对象，可以方便客户端程序查询用户信息



`addUser(person_info user)`: 向数据库中添加用户  
`editUser(person_info user)`: 修改用户信息  
`delUser(String name)`: 删除用户名为 `name` 的用户

不同的函数用 `SELECT`、`INSERT`、`UPDATE`、`DELETE` 语句以及 `WHERE` 子语句准备 SQL 语句,再调用 `executeQuery` 与 `executeUpdate` 对数据库中的内容进行增删改查。

#### 4.2.3 实现方法

在工程其他需要访问数据库的地方,先创建一个 `user_data` 类的实例,通过 `getConn()`与数据库达成连接,之后通过创建的实例的方法进行增删改查即可。