基于 socket 通信实现的简易多人游戏

计算机系 171860508 张天昀

2019 年 5 月 25 日

1 背景与软件介绍

通过通信来实现一个多人游戏的灵感来自于现在商场内常见的游戏设施：一块大显示屏上展示 2D 的游戏界面，消费者通过扫描二维码即可通过 HTML5 或 react 实现的 instant application 加入游戏，关闭程序即可退出。游戏软件通过实现一个 D-Pad（或方向键等按钮），或通过读取设备姿态来让玩家操控；同时玩家对游戏状态的获取完全通过面前的显示屏进行，即手机屏幕上并不会包含过多的游戏内容。

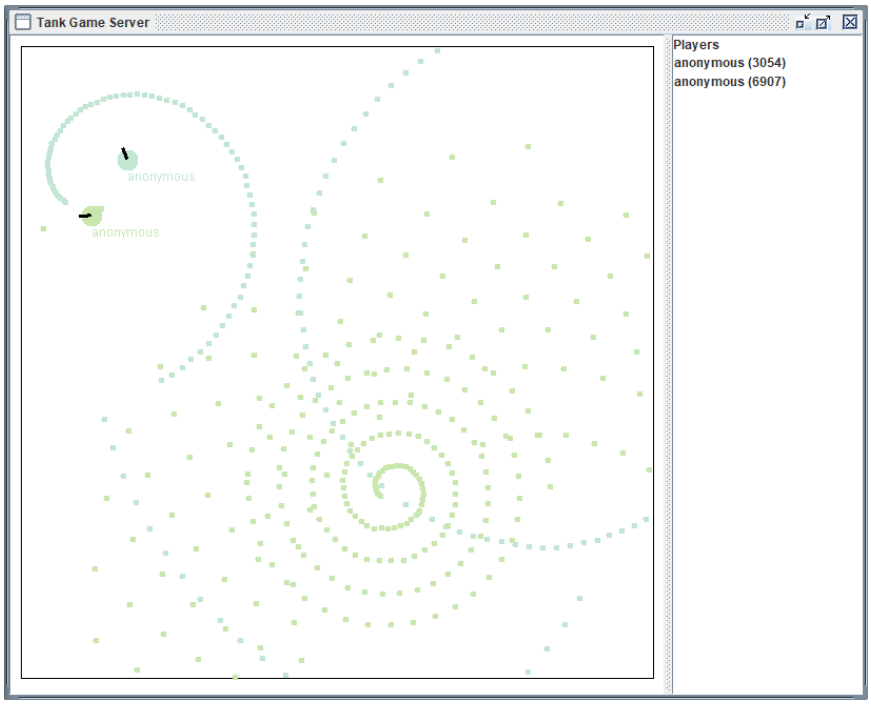

图 1: 服务端运行截图（左侧为游戏画面，右侧为玩家列表）

基于这个想法，我实现了一个基于 socket 通信的简单的“坦克大战”游戏：

- 游戏规则：玩家操纵坦克在 2D 平面上移动，可以前进、后退、调整方向、发射炮弹；炮弹击中其他玩家则得分。
- 客户端/服务端分工：游戏的画面通过服务端 GUI 显示，客户端只有简单的文字状态和输入按钮。客户端通过屏幕按钮、设备传感器读取数据。
- 数据通信：客户端通过 socket 以每 20 毫秒 1 次的频率向服务端发送指令，同时服务端收到指令后将处理后玩家的状态返回给客户端。

2 软件结构与交互过程

2.1 客户端

客户端共有两层界面，分别为设置服务端地址的 `MainActivity` 和游戏界面 `GameActivity`。

在 `MainActivity` 中，用户可以设置服务端的地址、端口号以及连接时使用的昵称，如图 2所示。

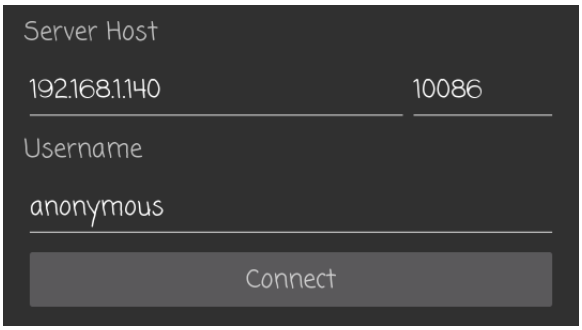

图 2: 客户端 `MainActivity` 界面

点击连接后，客户端启动 `GameActivity`，显示游戏界面，同时创建通信线程与指定地址的服务器端建立联系。游戏界面的主要内容如图 3，包括客户端的 UUID、服务端和客户端的状态（以文本方式展现），和操纵游戏使用的按钮（界面主体）。

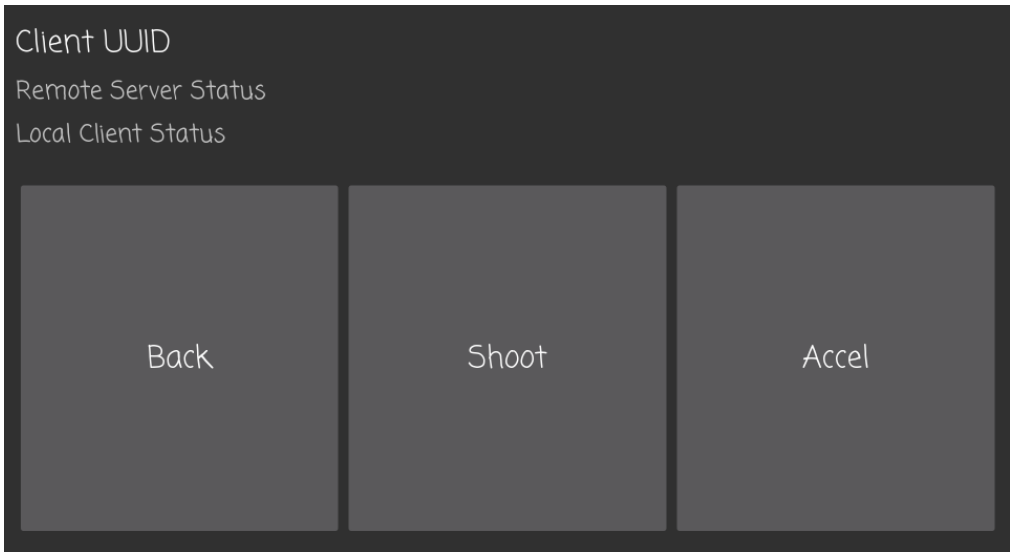

图 3: 客户端 `GameActivity` 界面

客户端以每秒 50 次的频率向服务端报告玩家的操作，操作内容包括三个按钮的点击情况、以及通过陀螺仪等传感器判断出的设备倾角。同时，客户端接收从服务端返回的玩家状态，如果玩家被炮弹击中（坐标被重置）则进行震动提醒。具体的通信方式在后文介绍。

2.2 服务端

服务器端由两个主要类组成：分别是图形界面 `Graphics` 和通信模块 `Server`，如图 4。服务端启动后会创建这两个类对应的线程，定期与客户端通信并绘制画面。

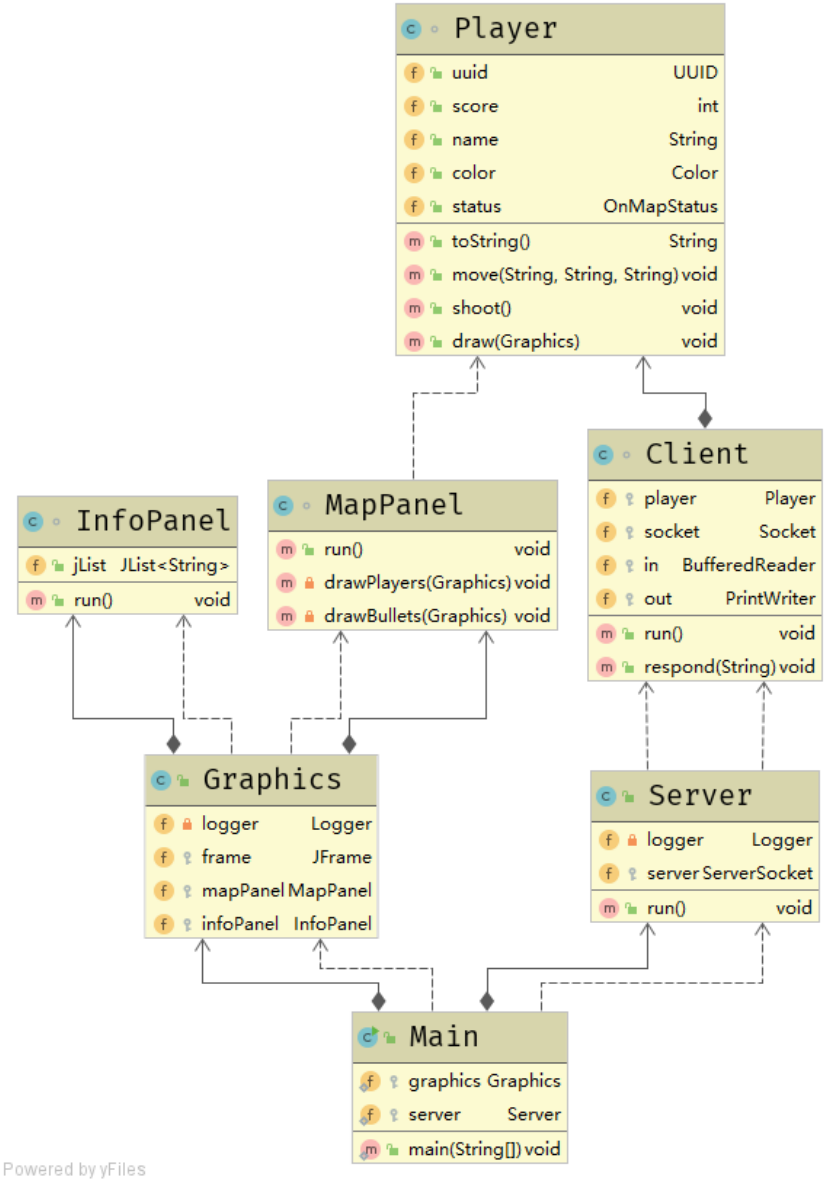

图 4: 服务端 class 关系图

`Graphics` 模块以每秒 40 次的频率重绘显示画面。整个显示界面分为 `MapPanel` 和 `InfoPanel` 两个部分，分别显示游戏地图和玩家列表。每次绘制画面时，都会从 `Server` 所持有的客户端列表中读取信息，调用 `drawPlayers(Graphics)` 和 `drawBullets(Graphics)` 方法将玩家和子弹绘制到地图画面的指定位置。

`Server` 模块维护一个客户端列表，保留对应的用户状态等信息，并处理与客户端之间的通信。每一个客户端 `Client` 实体持有一个 `Player` 实体，体现了玩家与客户端的一一对应的关系。`Client` 实现了 Java 的 `Runnable` 抽象类，创建后即作为线程在后台运行，维持与客户端的通信。当通信主动结束或丢失时，线程停止运行，实体自动销毁。

在服务端，游戏的两个主要部分：玩家（坦克）和炮弹以异步的方式进行更新。每当服务器端收到玩家的指令数据，就会更新玩家的位置、方向；而每当 `Graphics` 更新画面，所有的炮弹就会移动到下一个位置。这样即使玩家没有任何输入，炮弹依然会沿着之前的轨道移动。同时，当炮弹击中其他玩家或者触碰到地图边缘时，就会从炮弹列表被删除，并根据规则给对应的玩家增加得分。

2.3 数据格式

为了避免复杂的数据处理（如使用 JSON 等标记语言），在客户端与服务端交互时使用的数据格式参考了证件机读码的设计，即按预先设计的顺序将数据排列好后，两组数据之间插入 < 符号表示分隔，这样就可以把多个数据组合成一个字符串进行传输。接收端收到后可以通过调用 Java 的 `split("<")` 函数将收到的数据分割成原始组成部分，进而进行处理。

客户端想要建立连接时，发送 `CON<$username` 指令，服务端存储收到的用户名称，返回一个生成的 UUID，表示建立连接成功。此后，客户端每次向服务端发送的数据共有 4 个部分，其代表的含义如下：

- 第一部分表示此次指令的类型，`ACT` 表示玩家操作，`BYE` 表示主动断开连接。
- 第二部分表示玩家的操作，`ACC` 表示前进，`BAK` 表示倒退，`SHT` 表示射击，`NUL` 表示无操作。
- 后两部分组合起来，表示设备的倾角。如 `NEG<030` 表示设备向右（顺时针）旋转了 30 度。服务端可以通过这个角度来更改玩家的移动方向。

例如，客户端发出向前直线移动指令时，发出的数据为 `ACT<ACC<POS<000`。

同样的，服务端向客户端返回的数据也为 4 部分，

- 第一部分固定为 `RES`，表示这是来自服务端的回应。
- 第二部分根据用户操作而异，如果用户操作合法，则返回对应的操作指令，否则此部分为 `INV`。
- 后两部分组合起来表示用户当前在游戏地图中的二维坐标 (x, y)，坐标分别用两个三位整数表示。

例如，客户端发出向前移动指令，服务器执行后返回 `RES<MOV<114<514`。

2.4 交互流程

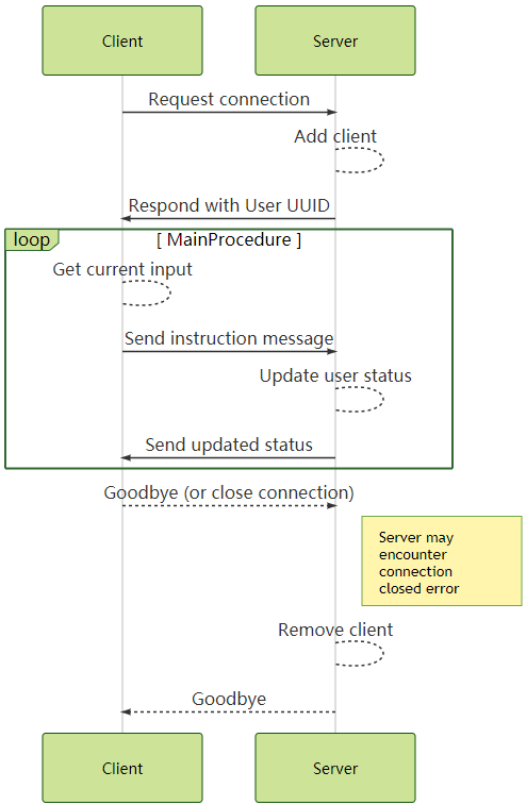

图 5: 交互时序图

如时序图（图 5）所示，整个程序的生命周期如下：

1. 客户端向服务端发送连接请求，同时传输用户昵称；
2. 服务端收到请求，创建连接对象并生成 UUID 传送给客户端。
3. 进入主循环：
   - 客户端将当前用户输入（按钮、设备姿态）整合后发送给服务端；
   - 服务端根据收到的指令执行操作，更新玩家的状态；
   - 服务端操作结束后将用户的最新状态发送回客户端；
   - 客户端更新显示内容，等待下一次循环。
4. 客户端主动提出结束连接，或直接关闭连接；
5. 服务端响应结束请求或检测到连接已关闭的错误，关闭连接并删除对应客户端对象。

3 遇到的困难与解决方案

3.1 画面闪烁

在制作服务端图形界面时出现了画面闪烁的问题。经过查询资料发现，由于游戏画面需要以 40FPS 的速率重新绘制，每次绘制时都会将画面先清空（即调用 `update(g)`）后再重新绘制图形，会导致画面出现闪烁。原有的代码为

```
java.awt.Graphics g = getGraphics();
while (true) {
    // wait for next frame
    update(g);
    drawPlayers(g);
    drawBullets(g);
}
```

为了解决画面闪烁的问题，需要使用双缓冲的策略，即创建一层相同大小的图像 `buffer` 作为缓冲区，每次重绘界面时将所有内容绘制到 `buffer` 中，绘制完后一次性将 `buffer` 绘制到界面的实际缓存中。修改后的代码如下

```
java.awt.Graphics g = getGraphics();
Image buffer = createImage(this.getWidth(), this.getHeight());
java.awt.Graphics g2 = buffer.getGraphics();
while (true) {
    // wait for next frame
    update(g2);
    drawPlayers(g2);
    drawBullets(g2);

    g.drawImage(buffer, 0, 0, null);
}
```

3.2 连接中断

在 socket 连接因某种原因断开时，连接时打开的输入、输出流也都会关闭，因此如果继续尝试读入会产生空指针的错误（见时序图图 5 中的黄色标记）。

通常情况下，如果手机客户端退出，系统仍然会将 socket 连接保持在后台；但如果客户端进程被关闭，那么 socket 连接就会被系统关闭。此时服务端的输入、输出流就会被改变为空指针。

当服务端与客户端交互中出现错误时，`try` 语句块将会停止执行，在我的实现中一律认为客户端已经终止连接，随后将客户端从列表中移除，结束线程。

```
void playerMainThread() {
    try {
        while (connection_is_alive) {
            // play game
        }
    } expect (Exception e) {
        // handle error if not 'connection reset'
    }
    // remove player
}
```

3.3 多线程并发问题

服务端的界面绘制、客户端通信都是并发执行的，多线程对资源的访问可能产生竞争，因此我使用了以下的方式来避免竞争导致的问题：

- 对于图形界面模块 `Graphics` 来说，所有的对于客户端和玩家信息的操作只有读没有写，因此可以放心的访问。
- 当用户进行游戏操作时，只会修改自身的 `Player` 数据，不会访问其他线程对应的资源，也就不会产生竞争。
- 当用户发起连接或者断开连接时，需要对客户端列表进行修改。此时可能会影响到其他线程的执行。为了解决此问题，我使用了 Java 提供的并发数据结构 `java.util.concurrent.CopyOnWriteArrayList`，此结构的列表在修改时会自动复制一份来保持数据的完整性。同时，把修改客户端列表的函数修改为使用 `synchronized` 同步的函数，防止竞争状态的出现。

4 自我评价

- Java 的开发使用了面向对象设计的方法，游戏参数从代码中剥离出来，通过一个单独的静态类控制，可以方便的进行修改来改变游戏体验。
- 所有的玩家操作均通过异步执行的方式完成，如果客户端或服务端线程出现暂时阻塞，阻塞恢复后会依次处理缓冲区接收到的数据，所以软件的稳定性基本依赖于 `BufferedReader` 的稳定性。
- 由于实现时间有限，加上缺乏软件开发的经验，我对软件正确性和鲁棒性的测试并不充分，因此可能会出现没有在测试中遇到的错误，进而导致软件运行异常。