Java《双人单机+局域网联机 Q 版泡泡堂》）项目开发报告

1.项目概述

1.1 基本内容：

本项目是一个基于 Java 和 JavaFX，利用 FXGL 游戏引擎进行开发的双人游戏，模仿童年 4399 游戏泡泡堂。游戏基本逻辑是玩家通过放置炸弹、拾取道具击败其他玩家从而获得胜利。本项目游戏包含双人单机和双人联机模式，游戏界面友好，操作简便，旨在通过本项目熟练应用 Java 的基础语法，加深对网络编程的理解以及初步学习体会游戏开发 Entity-Component-System 的概念。

1.2 主要参加人员

组长：戴竹涵

成员：吴悦 李天悦 谢依潼

1.3 业绩权重说明：均分（1 1 1 1）

1.4 工作任务的分解和人员分工

戴竹涵：负责道具的使用和生成、可破坏障碍物的生成、联机模式的实现以及部分音效的处理；

吴悦：负责人物行走、地图绘制以及不可破坏障碍物的生成，以及单机游戏大厅界面的设计；

李天悦：负责游戏的计分与结算系统，游戏中和结束界面的 UI 设计，以及主要音效添加；

谢依潼：负责菜单 UI 的设计和帮助文档的编写；

1.5 知识点应用等基本说明

知识点应用：JAVA 基础语法、JAVA 网络编程及多线程编程的基础应用

创新点 ：采用在 GITHUB 上开源的基于 JAVAFX 开发的 JAVA 游戏引擎 FXGL

技术难点：

1、在游戏引擎对单应用多模式游戏与联机通信的支持一般（或者说因引擎框架限制导致实现部分效果变得较为困难）的情况下实现单机与联机游戏 大 厅并将用户在联机大厅的选择传入游戏本体类；

2、联机时保证双方地图随机道具人物行走与攻击爆出道具等游戏元素的一致以及对各种断

连的考虑合处理。

2.基本功能

《双人单机+局域网联机 Q 版泡泡堂》是一款面向 2 名玩家的对战游戏，支持单机和联机两种模式。

- **单机模式：**两位玩家可以共享同一台电脑的键盘进行游戏。玩家 1 通过“WSAD”键控制角色移动，并通过空格键放置炸弹；而玩家 2 则通过“↑ ↓← →”键进行移动，并通过回车键放置炸弹。
- **联机模式：**游戏的联机模式支持局域网内的联机游戏，房主可以创建房间，而其他玩家则可以通过输入局域网 IP 地址加入房间进行游戏。在联机对战中，玩家通过“WSAD”键控制角色移动，并通过空格键放置炸弹。
- **规则总概：**本游戏提供了四个不同的角色和三张地图供玩家选择。单机和联机模式开始前，玩家都可以进行选择。每位玩家的初始生命值为 3，游戏的时间限制为 180 秒。在游戏过程中，玩家可以捡起各种道具，如鞋子可以加速玩家的移动速度，炸弹可以增加连续放置的炸弹数量，红瓶子可以扩大炸弹的爆炸范围，而蓝瓶子则可以减少炸弹的伤害范围，但不会低于初始的十字伤害。游戏界面的左侧会实时显示每个玩家的生命值和道具数量，并会随着生命值的减少及道具的获取和使用而变化。同时游戏界面也会显示游戏的剩余时间。游戏会对胜负进行判定，在时间结束前存活的一方获胜，如果时间结束时两个玩家均仍存活，则判定为平局。

3.系统设计与实现

3.1 系统架构设计

①采用技术和平台

Java：作为项目的主要编程语言，Java 提供了强大的面向对象编程能力，易于处理游戏中的多样化逻辑和数据结构。Java 的广泛应用和稳定性也为项目的开发和维护提供了便利。

JavaFX：用于开发富客户端应用程序的框架。在本项目中，JavaFX 主要用于构建用户界面，如游戏窗口、菜单和游戏中的各种交互元素。JavaFX 的高度可定制性和响应式设计能力使得游戏界面更加直观和吸引人。

FXGL 游戏引擎：一个基于 JavaFX 的轻量级游戏开发库。它简化了许多常见游戏开发任务，如动画处理、物理模拟、场景管理等。FXGL 的使用使得项目能够专注于游戏逻辑和特性的实现，而不必从头开始处理底层的游戏开发细节。此外，FXGL 引擎提供了对网络编程支持的 API，支持项目实现局域网联机模式。

其中 FXGL 的 github 链接如下：

https://github.com/AlmasB/FXGL/blob/dev/README_CN.md

局域网联机主要参考的 FXGL 文档链接如下：

https://github.com/AlmasB/FXGL/wiki/Multiplayer-and-Networking-(FXGL-11)

另外项目团队学习 FXGL 的视频教程链接如下：

https://www.youtube.com/watch?v=-sXkHTQKszk

https://www.bilibili.com/video/BV1di4y1r7eG/?spm_id_from=333.337.search-card.all.click

②系统类图

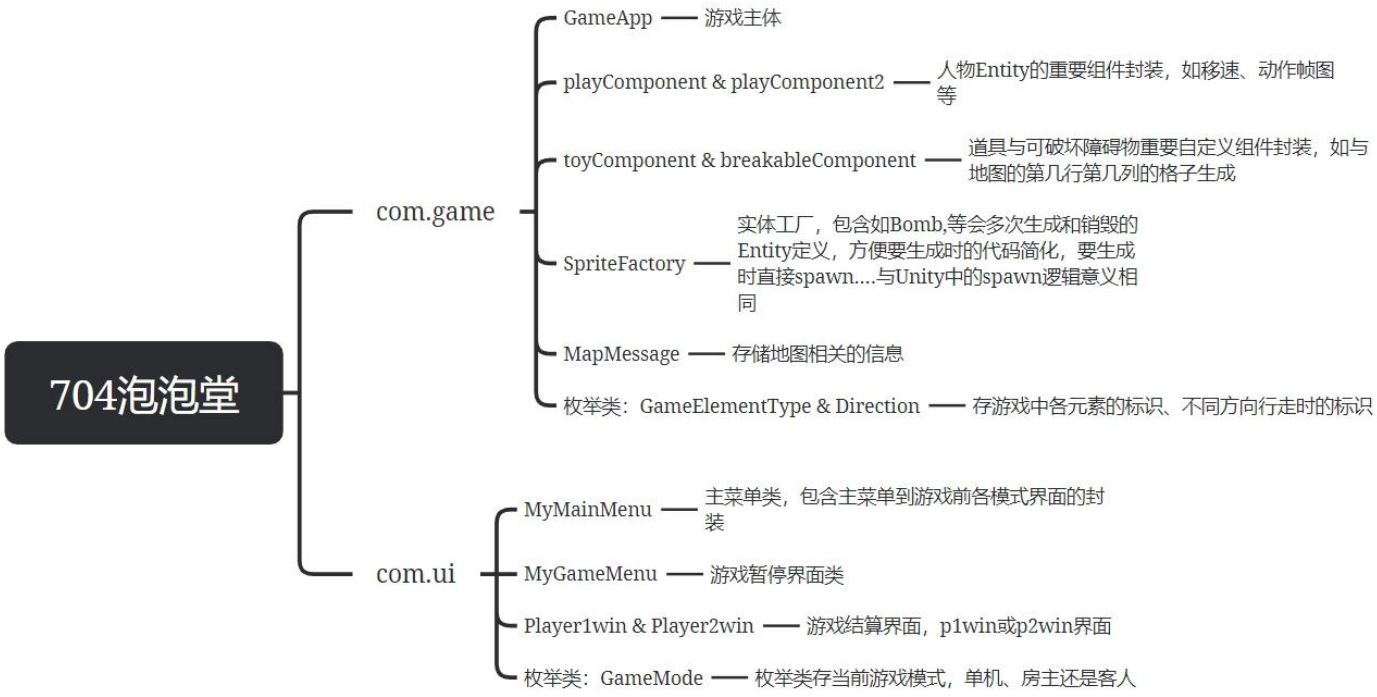

图 3-1-1 系统类图

3.2 功能详细设计

3.2.1 道具、可破坏障碍物的生成与交互、联机模式

- 可破坏障碍物、道具的生成与交互

①可破坏障碍物生成：为了避免游戏逻辑上的 bug，可破坏障碍物的生成位置和总数被设计为确定的；同时为了给玩家增加地图的随机感，三种地图分别具有两种可破坏障碍物的贴图，每局地图生成时采用随机数来决定确定位置要用的的障碍物贴图，避免重复地图的枯燥。

②可破坏障碍物交互：玩家可以通过投掷炸弹破坏可破坏障碍物，处于炸弹范围内的可破坏障碍物将被移除并在此局内不再生成，破坏障碍物时有一定几率爆出四种道具内的任意一种。

③道具生成：道具有两种生成方式，一种是于间隔 7s 游戏时间后于地图无障碍物且无可破坏障碍物且无已生成道具的格子根据随机数落在的区间生成四种道具内的任意一种；第二种生成方式则是玩家破坏可破坏障碍物时爆出，为增加游戏性，爆出的道具设计为有一定概率被此炸弹伤害移除。另外一局游戏最多有 20 个道具同时呈现在地图上，超过 20 个后将不会再以任何方式生成道具。

④道具交互：道具于游戏中的交互可分为两类——被拾起与被破坏。其中玩家行走时碰到道具则会被视为拾起道具，四种道具的功能简介如下：鞋子增加玩家的移动速度，至一定数量后标识为 full 继续捡起道具不会再增加玩家速度；炸弹可以增加玩家可连续放置的炸弹数量；红瓶子可以扩大炸弹的爆炸范围，一次向外扩大一圈十字伤害，有效上限为 2 个增幅；蓝瓶子减少炸弹的伤害范围，下限不会低于初始的上下左右各一格的十字伤害。道具除由破坏可破坏障碍物生成时有非常短暂的无敌时间，其余时候皆可被炸弹伤害移除。

- 联机模式

联机模式的规则同单机模式双人相似，不同主要在于：地图确定为三号地图雪人；规定 HOST 为 p1,CLIENT 为 p2；双方都采用 WASD 和 SPACE 进行游戏操作；HOST 拥有较高权限负责开始游戏并拥有暂停游戏结束游戏回到主界面的权限。

①HOST 与 CLIENT 的连接：联机模式采取 HOST 和 CLIENT 的交互模式：选择创建房间则被视为 HOST，创建房间进入游戏大厅后将会显示“你创建了房间”的消息并显示你创建房间时的 ip 地址，HOST 需要等待 CLIENT 进入房间才能开始选择角色进入下一步准备游戏开始的步骤；若选择加入房间则被视为 CLIENT，加入房间需要输入房主所开房间的 ip 地址，其中设置了正则

表达式判断输入 ip 的合法性，连接中若无法连接则会弹出错误窗口显示连接错误信息，若连接空房间或超时则会进入空房间，无法进行除回退上一界面的任一操作，若连接成功则会进入 HOST 房间并收到“你已进入房间”的信息并将此条信息发送给 HOST，采用 TCP 连接。

②HOST 与 CLIENT 在联机游戏大厅内做游戏准备逻辑设计：双方进入游戏后可在左下方的聊天框进行聊天，连接成功后 HOST 可以开始选角色，此时 CLIENT 不可选角色并提示请等待 HOST 选取角色，HOST 选取好角色后 p2 界面中 p1 选取的角色将亮起且不可被选择，select 按钮亮起表示 p2 可以开始选择角色，p2 选择好角色后 p1 界面相应被选择的人物亮起，“START”按钮亮起表示可以开始游戏，点击开始游戏后双方将同时进入游戏。

③HOST 与 CLIENT 在游戏中的交互：双方进入游戏后如单机模式一样 p1 生成在左上角，p2 生成在右上角，采用局部数据传输的方式同步两边信息，人物行走与炸弹放置动作采取双边传输，而道具的生成则被设置为 HOST 单边传输来保证画面的一致和道具不被重复生成，一方生命值降到 0 时进行游戏输赢判断，同时显示游戏结束界面，连接断开，按 RESTART 按钮回到游戏主菜单。

3.2.2 人物行走、地图、不可破坏障碍物的生成逻辑、单机模式游戏大厅界面设计

-人物行走、地图、不可破坏障碍物的生成逻辑：

对于游戏地图，我们用一个二维数组来存储地图的实时状态。元素值为 0 代表当前地块为空，可以进行人物移动或炸弹放置；元素值为 1 代表当前地块上有障碍物，角色和道具都无法位于该地块上;元素值为 2 则代表当前地块有道具生成。地图状态需要根据玩家交互实时更新。

在游戏主体中，设置[不可破坏障碍物],[可破坏障碍物],[人物],[道具],[炸弹]五类可交互元素，对于每一类的功能实现如下：

不可破坏障碍物：是提前设置好的、状态不会更改的障碍物，二维数组中不可破坏障碍物所在区域的值永远为 1。简称“死障碍”。

可破坏障碍物：是游戏初始化开始时放置上去的障碍物，每张地图障碍物初始位置固定，但在游戏过程中可以被破坏。可破坏障碍物被破坏后，二维数组对应元素值需要更改为 0(空地)。简称“活障碍”。

人物：玩家操控的主要角色，通过读取键盘输入来控制人物行走、放置炸弹等互动操作。人物数量设置为 2，即需要两位玩家。人物通过读取键盘输入来判断行走方向，根据不同方向播放相应的动作帧图，产生相应的动画效果。人物初始的生命值为 3，每次被炸弹伤害后生命值减一，生命值首先达到 0 者判定为输，另一方则胜利。

道具：设置四类道具，分别为炸弹、红瓶、蓝瓶、鞋子，效果分别为：增加该玩家最多同时可放置炸弹数、增加炸弹威力、减少炸弹威、提高移动速度。每个道具的个数存放在玩家类的成员变量里，根据其数值对实现对应的道具效果。

炸弹：是实现玩家间交互、玩家与地图交互的主要媒介，可以破坏地图、破坏道具、造成伤害。

游戏逻辑的实现重点在于五个元素间的交互，列表格如下（列对行的作用）

	死障碍	活障碍	人物	道具	炸弹
死障碍			阻挡人物行走	阻止道具生成在该处	影响炸弹爆炸后的波及范围
活障碍			阻挡人物行走	阻止道具生成在该处	被炸弹破坏
人物	人物不能继续朝该方向行走			人物可以拾起道具	人物可以放置炸弹
道具			修改人物的部分属性		
炸弹		炸弹破坏活障碍	炸弹使人物损失生命值	炸弹破坏道具	

人物位置坐标的改变与否是通过目标位置是否存在障碍物实现的。如果不存在障碍物，则根据速度改变相应的人物坐标，达到行走效果；如果存在障碍物则改变坐标后再使坐标回退，从而在视觉上达到人物“原地踏步”的效果。

各元素之间的交互判断是通过碰撞监听实现的。为每个可交互的元素的实体(entity)设置一个碰撞箱，通过 FXGL 中提供的各类碰撞方法实现交互逻辑。具体实现详见代码。

-单机模式游戏大厅界面设计：

单机模式游戏大厅界面重点功能为游戏大厅中的角色选择——实现选择逻辑和用户操作的合法性判断。主要实现功能为：初始状态下，角色图片、select 和 start 按钮为灰色的不可点击状态。玩家 1 首先点击角色图片(透明按钮)选择控制角色,角色图片更改为彩色,select 按钮亮起并更改为可点击状态。玩家点击 select 按钮确定所选角色，该角色则不可被第二个玩家选择。第二个玩家和地图选择操作同理。只有在两个玩家都已经选择了角色和地图后，

start 按钮才会亮起，玩家才可以开始游戏。

3.2.3 游戏的计分与结算系统、游戏中和结束界面的 UI 设计、及主要音效添加

-游戏计分功能：游戏计分功能和 p1,p2 角色在地图上的触发的事件绑定，负责初始化人物的生命值，移动速度，炸弹数量，爆炸伤害范围等信息，随着游戏过程中人物与周围实体的碰撞，如人物与爆炸形成的火焰发生碰撞后，人物的生命减一，在 GameVars 里面和屏幕上显示的血量绑定，在画面左边的人物信息栏进行实时的显示。

-游戏结算系统：创建三个子界面，分别对应三种游戏结束情况，即：p2 血量清零，p1 胜利，p1 血量清零，p2 胜利，游戏时间结束，二者血量都为清零，即为平局。游戏阶段界面中若有输赢则对两名玩家的状态分别显示，赢家用王冠标志状态，输者的图标变成黑白色，鼠标移动到二者头像上时会自行播放音效。结算菜单有两个功能性按钮，即退出游戏或者返回主菜单。

-音效添加：主要体现在切换场景时对应背景音乐改变，在点击可点击按钮，比如选择角色，确定角色时能够发出对应音效方便玩家确认点击效果。不同按钮之间对应的音效不同。鼠标移动到按钮上也会有对应音效，以丰富玩家的游玩体验。游戏地图内所操纵角色放下炸弹，炸弹爆炸，被炸弹火焰伤害，拾取到道具气泡时都会产生对应音效，实现游戏效果。

-游戏中及结束界面的 ui 设计：游戏中及结束界面的 ui 设计的主要操作是创建在固定位置的 entity 实体，用指定的被禁图片进行贴图，在中间的位置留出放置地图的空间，在后续工作中与组内成员设计实现的地图块耦合。

3.2.4 菜单 UI 的设计和帮助文档的编写

-菜单界面 UI 设计：在主界面设置四个按钮，分别为跳转单机模式，跳转联机模式，查看帮助和退出功能，并贴上相对应的图像，当鼠标移动到各个选项上方时，按钮图像切换为字体周围发光的图像，移开后发光消失，以此呈现鼠标移至按钮上，按钮发光的效果。编写帮助文档，在游戏内介绍游戏基本玩法和功能，点击主界面菜单的 HELP 即可跳转查看帮助文档，在文档左下角可点击 MENU 按钮跳转回主界面。

3.3 编码实现

编码实现重点在游戏主体类 GameApp 的实现、主菜单类的实现、联机模式的实现以及以人物组件类的实现为例展示 Component 类的作用。

3.3.1 GameApp 类: 游戏主体

利用 FXGL，若制作非常简单的小游戏只需要这一个类，在这个类里，FXGL 提供几个可覆写

的方法：initUI, initPhysics, initSettings 等来供玩家编写自己游戏相关功能的代码，使用时最好打印执行时所在的线程方便设计游戏代码的放置位置。

本项目本体即在覆写这几个方法的基础上通过封装众多功能函数与独立出其他类来实现，此类代码逻辑简概截图如下：

```
public class GameApp extends GameApplication{
    39 usages
    private MyMainMenu myMainMenu;
```

图 3-3-1-1 类继承自 FXGL 提供的重要类 GameApplication

```
//游戏窗口大小设置、图标设置等
no usages
@Override
protected void initSettings(GameSettings settings){...}
//游戏变量初始化
no usages
@Override
protected void initGameVars(Map<String, Object> vars){...}
//游戏主体逻辑
no usages
@Override
protected void initGame(){...}
//游戏过程中处理碰撞等，由FXGL的物理引擎支持
no usages
@Override
protected void initPhysics() {...}
//初始化游戏过程中的UI
no usages
@Override
protected void initUI(){...}
//游戏刷新函数，帧率与电脑刷新率有关
no usages
@Override
protected void onUpdate(double tpf){...}
```

图 3-3-1-2 实现游戏逻辑使用的各覆写方法

3.3.2 MyMainMenu 类：主菜单 UI 代码实现

FXGL 的有默认的主菜单界面，想要自定义的主菜单界面，需要另开一个类继承 FXGLMenu

类并选择标识是自定义主菜单而非游戏中暂停界面菜单，菜单界面中为了实现各个游戏前界面的跳转，保证不脱离 FXGL 的框架采用将每个界面用函数封装，跳转时清屏调用下一个界面函数的方式来实现伪界面跳转，其中注意在 originMenu 函数里封装主菜单界面时要同时将所用到的用于选择的一些标识变量恢复到默认状态，代码实现逻辑截图如下：

```
public class MyMainMenu extends FXGLMenu {
    4 usages
    private GameMode gameMode;
    1 usage
    public static int flag=0;//控制音乐播放
    14 usages
    private AtomicBoolean m_flag;
    3 usages
    private Vector<Double> p1BOMB = new Vector<>();
    3 usages
    private int bld1 = 3;
```

图 3-3-2-1 类创建，只截取了部分成员变量

```
//构造函数，初始主菜单
public MyMainMenu() throws FileNotFoundException {...}
//帮助文档查看界面
2 usages
public void read_help() throws FileNotFoundException {...}
//单机游戏大厅界面
2 usages
public void consoleMode() throws FileNotFoundException {...}
//联机模式选择创建或加入房间界面
4 usages
public void netMode(){...}
//联机模式创建房间界面
2 usages
public void crtRoom(){...}
//联机模式加入房间界面
2 usages
public  void  joinRoom(){...}
//联机模式游戏大厅
2 usages
public void gameHall_net() throws FileNotFoundException {...}
//保存初始主菜单，要回来的
5 usages
public void originMenu() throws FileNotFoundException {...}
```

图 1-3-2-2 各界面封装函数

MyGameMenu 同理，为对 FXGLGameMenu 的重写。

3.3.3 联机模式实现

①房主创建房间的实现：

在 MyMainMenu 类的 crtRoom 函数中，默认房主名为 HOST，点击创建房间后检验房间是否可创建代码截图如下：

```
btn_crtRoom.setOnAction(event->{
    stopAndPlaybtn_moveon(); //按钮声音
    //避免检验时影响界面线程另开一个executor
    ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.submit(() -> {
        try {
            //检验ip是否可用
            serverIP = InetAddress.getLocalHost();
            System.out.println(serverIP);
            Platform.runLater(() -> {
                //ip可用，准备创建房间，进行界面跳转
                getContentRoot().getChildren().clear();
                p1name = textField1.getText();
                System.out.println(p1name);
                try {
                    gameHall_net();
                } catch (FileNotFoundException e) {
                    throw new RuntimeException(e);
                }
            });
        } catch (UnknownHostException e) {
            throw new RuntimeException(e);
        }
    });
    executor.shutdown();
```

图 3-3-3-1 创建房间前置检验

房间创建成功后进入新界面开启 FXGL 的网络服务，创建一个 TCPserver 并等待连接，设置监

听连接的事件监听器以及监听来自 client 的信息的信息监听器，代码截图如下：

```
if (isServer) {
    //聊天框添加房主创建房间信息
    lb_choosinginfo.setText("Waiting for p2 to join the room...");
    String first = p1name + "[You]创建了房间(ip address: " + serverIP + ")";
    chatListView.getItems().add(first);
    //开启TCPserver可以接受连接了
    server = getNetService().newTCPServer( port: 8889);
    server.startAsync();
    //启动监听连接，一旦连接上让角色选择等按钮变为可用并立即
    server.setOnConnected(connection -> {
        System.out.println("a client!");
        btn_role1.setOnAction(e -> {stopAndPlaybtn_moveon();handleButtonClick1(btn_role1,btnSELECT);});
        btn_role2.setOnAction(e -> {stopAndPlaybtn_moveon();handleButtonClick2(btn_role2,btnSELECT);});
        btn_role3.setOnAction(e -> {stopAndPlaybtn_moveon();handleButtonClick3(btn_role3,btnSELECT);});
        btn_role4.setOnAction(e -> {stopAndPlaybtn_moveon();handleButtonClick4(btn_role4, btnSELECT);});
        btnSELECT.setOnAction(e -> {...});
        btnSTART.setOnAction(e -> {...}); //开始游戏
        //监听来自client的信息
        connection.addMessageHandlerFX((conn, message) -> {...});
    });
```

图 3-3-3-2 房主开启等待连接和接收信息代码

②客户加入房间实现：

在 MyMainMenu 类的 joinRoom 函数中，默认客户昵称为 player2，输入房间 ip 地址后先检验房间 ip 是否合法再检验房间是否可以连接，若无法连接则会弹出错误窗口显示连接错误信息，若连接空房间或超时则会进入空房间，无法进行除回退上一界面的任一操作，若可连接则跳转至联机游戏大厅界面开启 FXGL 的网络服务，创建一个 TCPclient 并连接房间，设置监听连接的事件监听器以及监听来自 server 的信息的信息监听器，连接一旦成功则接受来自 HOST 的房间信息并发送我已进入房间的信息给 sever，代码截图如下：

```
private static final Pattern PATTERN_IP = Pattern.compile( regex: "^(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.){3}([01]?\\d\\d?|2[0-4]\\d|25[0-5])$");
1 usage
private boolean checkIP(String text)
{
    return PATTERN_IP.matcher(text).matches() ? true : false;
}

//ip不合法不可以连接房间
tf_serverip.textProperty().addListener(e -> {
    serverIP_string = tf_serverip.getText().trim();
    if (!checkIP(serverIP_string) && !serverIP_string.isEmpty()){
        label.setVisible(true);
        button1.setDisable(true);
    }else if(serverIP_string.isEmpty()){
        label.setVisible(false);
    }else {
        label.setVisible(false);
        button1.setDisable(false);
    }
```

图 3-3-3-3 检验 ip 是否合法且可达

```
//开启TCPClient
client = getNetService().newTCPClient(serverIP_string, port: 8889);
client.connectAsync();
client.setOnConnected(connection -> {
    btn_role1.setDisable(true);
    btn_role2.setDisable(true);
    btn_role3.setDisable(true);
    btn_role4.setDisable(true);
    btn_role1.setOnAction(e -> handleButtonClick1(btn_role1,btnSELECT));
    btn_role2.setOnAction(e -> handleButtonClick2(btn_role2,btnSELECT));
    btn_role3.setOnAction(e -> handleButtonClick3(btn_role3,btnSELECT));
    btn_role4.setOnAction(e -> handleButtonClick4(btn_role4, btnSELECT));
    btnSELECT.setOnAction(e -> {handlebtnSELECT_guest(btn_role1, btn_role2, btn_role3, btn_role4, btnSELECT, btnSTART); btn_Click.play();});
    boolean isConnected = true; //一旦连接成功，就给sever发包：p2的nickname,连接成功的信息
    var bundle = new Bundle( name: "client");
    bundle.put("connecting",isConnected);
    bundle.put("p2name",p2name);
    client.broadcast(bundle);
    //监听来自server的信息
    connection.addMessageHandlerFX((conn, message) -> {...});
});
```

图 3-3-3-4 客户加入房间并接受发送信息代码

③HOST 和 CLIENT 信息收发实现：

信息收发较多，仅以联机大厅聊天实现和游戏中 client 同步 server 人物向上走实现为例子。

- 联机大厅聊天信息收发代码截图

```
private void sendMessage() {
    String message = messageField.getText();
    if (!message.isEmpty()) {
        if(isServer){
            //server模式,先让自己能看到自己发的信息
            chatListView.getItems().add(p1name+"[You]: " + message);
            String mess = p1name+"[Host]: " + message;
            messageField.clear();
            //打包这条信息
            var bundle = new Bundle( name: "msg");
            bundle.put("p1msg",mess);
            //发给client
            server.broadcast(bundle);
        }else {
            chatListView.getItems().add(p2name+"[You]: " + message);
            String mess = p2name+"[Guest]: " + message;
            messageField.clear();
            //同理server发信息
            var bundle = new Bundle( name: "msg");
            bundle.put("p2msg",mess);
            client.broadcast(bundle);
        }
    }
}
```

图 3-3-3-5 send 按钮连的函数来发送信息

```
if(message.get("p2msg")!=null){
    String p2msg = message.get("p2msg");
    chatListView.getItems().add(p2msg);
}
```

```
if(message.get("p1msg")!=null){
    String p1msg = message.get("p1msg");
    chatListView.getItems().add(p1msg);
}
```

图 3-3-3-6 server 与 client 接受信息（于 connection.addMessageHandlerFX((conn, message)->{}里)

- 游戏中 client 同步 server 人物向上走代码截图

```
FXGL.getInput().addAction(moveup = new UserAction( name: "moveup") {
    no usages
    @Override
    protected void onAction() {
    if(gameMode ==GameMode.LOCAL)
    {...}
    if(gameMode ==GameMode.SERVER)
    {
        playercomponent1.moveUp();
        double x = player1.getX();
        double y = player1.getY();
        //将p1向上走动作标签和对应坐标打包发送给client
        var bundle = new Bundle( name: "p1move");
        bundle.put("p1move", 0);
        bundle.put("p1move_x", x);
        bundle.put("p1move_y", y);
        server.broadcast(bundle);
    }
    if(gameMode ==GameMode.CLIENT)
    {...}
}
},KeyCode.W);
```

图 3-3-3-7 GameApp 类内：server 向上走，同时广播自己向上走信息

```
//收到的p1/p2行动
2 usages
private Vector<Pair<Integer, Pair<Double,Double>>> p1moves = new Vector<>();
1 usage
public Vector<Pair<Integer, Pair<Double, Double>>> getP1moves() { return p1moves; }

if(message.get("p1move")!=null){
    int dir = message.get("p1move");
    double x = message.get("p1move_x");
    double y = message.get("p1move_y");
    Pair<Double,Double> xy = new Pair<>(x,y);
    Pair<Integer,Pair<Double,Double>> p1move = new Pair<>(dir,xy);
    p1moves.add(p1move);
}
```

图 3-3-3-8 MyMainMenu 类内：client 接受 p1 移动信息，存储

```
//接受p1移动消息
Vector<Pair<Integer, Pair<Double,Double>>> p1moves = myMainMenu.getP1moves();
if(p1moves.size()>0){
    int dir = p1moves.firstElement().getKey();
    double x = p1moves.firstElement().getValue().getKey();
    double y = p1moves.firstElement().getValue().getValue();
    if(dir == 0){
        playercomponent1.moveUp();
        player1.setPosition(x,y);
    }else if(dir== 1){
        playercomponent1.moveDown();
        player1.setPosition(x,y);
    }else if(dir== 2){
        playercomponent1.moveLeft();
        player1.setPosition(x,y);
    }else if(dir== 3){
        playercomponent1.moveRight();
        player1.setPosition(x,y);
    }
    p1moves.remove( index: 0);
}
```

图 3-3-3-8 GameApp 类 OnUpdate 函数内：通过 get 方法收到此信息，使 client 界面 p1 移动

3.3.4 playerComponent 类

人物类的部分主要代码：

```
public class playerComponent extends Component {
    11 usages
    public int INIT_SPEED = 40;
    2 usages
    private Point2D playerCenter;
    12 usages
    public int speed;
    5 usages
    public LocalTimer unbeatTimer;
    2 usages
    public Duration unbeatDealy = Duration.seconds( v: 1.4); //受伤后的无敌时间
    6 usages
    public int myBomb = 1; //初始可一次性放1个炸弹
    8 usages
    public int addFire = 0; //初始炸弹威力范围为十字
    5 usages
    public int mylife = 3;
```

图 3-3-4-1 人物类创建，省去部分初始化人物属性的变量声明

```java
public void moveUp(){
    if (isMoving)
        return;
    isMoving = true;
    dir = Direction.UP;
    speed = -INIT_SPEED;
    velocity.set((float) (0), (float) (speed));
    move();
}
3 usages
public void moveDown(){
    if (isMoving)
        return;
    isMoving = true;
    dir = Direction.DOWN;
    speed = INIT_SPEED;
    velocity.set((float) (0), (float) (speed));
    move();
}
```

```java
private void move() {
    List<Entity> blockList;
    blockList = blocks.get().getEntitiesCopy();
    int length = Math.round(velocity.length());
    velocity.normalizeLocal();
    for (int i = 0; i < length; i++) {
        //System.out.println("速度为:" + length);
        entity.translate( dx: velocity.x / 60,  dy: velocity.y / 60);
        boolean collision = false;
        for (int j = 0; j < blockList.size(); j++) {
            if (blockList.get(j).getBoundingBoxComponent().isCollidingWith(bbox)) {//碰撞判断，bbox直接能自动匹配对应实体
                collision = true;
                break;
            }
        }
        //运动，遇到障碍物回退
        if (collision) {
            entity.translate(-velocity.x, -velocity.y);
            break;
        }

    }
}
```

图 3-3-4-2 人物类中设置人物行走的部分代码展示

3.3.5 其余部分功能代码展示

角色选择部分代码：

```
private void handleButtonClick1(Button btn_role1, Button btnSELECT) {
    if (!btn_role1.isDisabled()) {
        isChoosing = 1;
        selectBT = true;

        ColorAdjust colorful = new ColorAdjust();
        colorful.setSaturation(0);
        role1_image.setEffect(colorful);  //角色一被选中，设成彩色

        ColorAdjust grey = new ColorAdjust();
        grey.setSaturation(-1);  //其他角色设成灰色
        role2_image.setEffect(grey);
        role3_image.setEffect(grey);
        role4_image.setEffect(grey);
        if (isChoosing != 0) {
            btnSELECT_image.setEffect(colorful);
            btnSELECT.setDisable(false);
        }
        if (player2Flag) {
            role2_image.setEffect(colorful);
        }
        if (player3Flag) {
            role3_image.setEffect(colorful);
        }
        if (player4Flag) {
            role4_image.setEffect(colorful);
        }
        System.out.println("角色一已被选中");
    }
}
```

```
private void handleButtonClick5(Button map_bt1, Button btnSELECT2) {
    if (!map_bt1.isDisabled()) {
        isChoosingMap = 1;
        ColorAdjust colorAdjustt = new ColorAdjust();
        colorAdjustt.setSaturation(0);
        ColorAdjust colorAdjust = new ColorAdjust();
        colorAdjust.setSaturation(-1);
        snow_image.setEffect(colorAdjust);
        grass_image.setEffect(colorAdjust);
        castle_image.setEffect(colorAdjustt);
        if (isChoosingMap != 0) {
            btnSELECT_image2.setEffect(colorAdjustt);
            btnSELECT2.setDisable(false);
        }
        System.out.println("地图一已被选中"+"地图的选择数是"+isChoosingMap);
        MyMap=1;
    }
}
```

图 3-3-5-1 角色选择部分代码展示

碰撞检测部分代码：

```
FXGL.getPhysicsWorld().addCollisionHandler(new CollisionHandler(GameType.TOYBOMB,GameType.BOOM) {
    no usages
    @Override
    protected void onCollisionBegin(Entity toyBomb, Entity boom) {
        maxToys++;
        toyCompo = toyBomb.getComponent(toyComponent.class);
        int j = toyCompo.j;int i = toyCompo.i;
        MAPCOPY[j][i] = 0;
        toyBomb.removeFromWorld();
    }
});
FXGL.getPhysicsWorld().addCollisionHandler(new CollisionHandler(GameType.TOYBOTTLE,GameType.BOOM) {
    no usages
    @Override
    protected void onCollisionBegin(Entity toyBottle, Entity boom) {
        maxToys++;
        toyCompo = toyBottle.getComponent(toyComponent.class);
        int j = toyCompo.j;int i = toyCompo.i;
        MAPCOPY[j][i] = 0;
        toyBottle.removeFromWorld();
    }
});

FXGL.getPhysicsWorld().addCollisionHandler(new CollisionHandler(GameType.TOYBOMB,GameType.BOOM) {
    no usages
    @Override
    protected void onCollisionBegin(Entity toyBomb, Entity boom) {
        maxToys++;
        toyCompo = toyBomb.getComponent(toyComponent.class);
        int j = toyCompo.j;int i = toyCompo.i;
        MAPCOPY[j][i] = 0;
        toyBomb.removeFromWorld();
    }
});
FXGL.getPhysicsWorld().addCollisionHandler(new CollisionHandler(GameType.TOYBOTTLE,GameType.BOOM) {
    no usages
    @Override
    protected void onCollisionBegin(Entity toyBottle, Entity boom) {
        maxToys++;
        toyCompo = toyBottle.getComponent(toyComponent.class);
        int j = toyCompo.j;int i = toyCompo.i;
        MAPCOPY[j][i] = 0;
        toyBottle.removeFromWorld();
    }
});
```

图 3-3-5-2 道具被炸破坏代码

（fakebomb 的作用是避免角色和自己放置的炸弹卡在一起）

```
FXGL.getPhysicsWorld().addCollisionHandler(new CollisionHandler(GameType.PLAYER1, GameType.FAKEBOMB) {
    no usages
    @Override//监听碰撞，如果碰撞结束，就把无bbox的fakeBomb类移除掉，原位置替换成有bbox的Bomb类
    protected void onCollisionEnd(Entity player, Entity fakeBomb) {
        System.out.println("碰撞结束");
        if(!fakeBomb.isVisible()){
            return;
        }
        realBomb = FXGL.getGameWorld().spawn( entityName: "bomb",
                new SpawnData(fakeBomb.getX(), fakeBomb.getY()));
        fakeBomb.removeFromWorld();
    }
});
```

图 3-3-5-3 使人物刚放置炸弹时炸弹先是可穿过再编程 realBomb 不可穿过代码

4.系统测试

4.1 测试概览

系统测试是确保《双人单机+局域网联机 Q 版泡泡堂》达到设计和功能要求的重要环节。本项目测试包括功能测试、性能测试和用户接受度测试，以保证游戏在各方面都符合预期。

4.2 功能测试

单机模式测试：进行单机模式测试，检验两位玩家是否能在同一台电脑上顺利游戏，包括角色移动、炸弹放置、道具拾取等基本操作的响应性和准确性，经测试本项目单机模式测试结果良好。

联机模式测试：进行单机模式测试，验证局域网内的联机功能，确保玩家可以通过 IP 地址加入房间，测试房间创建、连接稳定性以及数据同步准确性，经测试本项目联机模式测试结果一般，存在断联处理存在不易修复漏洞、数据同步有时存在较高延迟问题。

游戏规则和逻辑测试：确保所有游戏规则被正确实现，包括计分系统、胜负判定、时间限制等，经测试本项目游戏规则和逻辑测试结果良好。

4.3 性能测试

响应时间：测试游戏操作的响应时间，确保所有动作反馈迅速无明显延迟，经测试本项目单机模式响应时间测试结果良好，联机模式响应时间欠佳。

4.4 用户接受度测试

用户界面友好性：本项目团队收集了测试用户对游戏界面的反馈，确保界面直观易懂，经测试本项目用户界面友好性测试结果良好。

游戏体验：本项目团队收集了测试用户的反馈来评估游戏的整体体验，经测试本项目游戏体验测试结果良好。

5.项目展望

5.1 联机模式的优化：

①由于本项目团队也是初学 FXGL 引擎与网络编程，采取了教程所授的底层是 Socket 的 TCP Server 和 Client，并不适合用作游戏的数据传包，易出现明显延迟，可在熟练后改为 UDP 连接以优化延迟问题。

②联机模式只支持局域网联机，后续可尝试通过内网穿透技术等技术实现异地网络联机。

5.2 游戏功能丰富：

由于时间限制未做成有机器人的 4 人单机模式和 4 人的联机模式，项目团队未来可增加更多功能和内容，如新的游戏模式、新地图、更多道具等，以增强游戏的可玩性和长久吸引力。