河南科技学院
2022 届本科毕业论文（设计）

题目：基于 Unity3D+Golang 的网络 TPS 设计与实现

学　　号：20181515101

姓　　名：杨勐奇

专　　业：计算机科学与技术

学　　院：信息工程学院

指导教师：黄勇（讲师）

完成时间：2022 年 4 月

摘 要

本文主要使用Unity3D引擎进行开发，通过Unity3D开发了TPS游戏，TPS游戏分为客户端和服务器两部分,客户端实现从UI登录界面到游戏大厅的创建，以及多名玩家同时进入到一个房间开始同一局游戏，游戏所有的连接方式均为TCP连接，一些玩家通过UI做出的操作均使用RPC调用服务端(比如登录注册)。服务器部分使用Golang语言进行开发，主要包含了连接的管理，以及对客户端发送过来的数据包进行处理比如是否进行转发操作，游戏逻辑的处理主要包含了玩家的登录注册请求和一些攻击伤害的服务端处理,数据持久化使用MongoDB数据库。

关键词: C#，网络游戏，Golang，Unity3D

ABSTRACT

The article is mainly developed using the Unity3D engine, Through Unity3D to develop the TPS client, from the UI login interface to the creation of the game lobby, and multiple players enter a room at the same time to start the same game, all the connection methods of the game are TCP connection, and some players use RPC to call the server (such as login registration) for operations made through the UI. The back-end part is developed using the Golang language, which mainly includes the management of the connection, and the processing of packets sent by the client such as whether to forward the operation, the processing of the game logic mainly includes the player's login registration request and some attack damage server-side processing, data persistence using MongoDB database, which is a document database is more suitable for storing untable structured data such as gamers.

Keywords: C#; Online games; Golang; Unity3D;

目 录

1 概述

互联网的高速发展游戏已经成为很多人的交友方式之一，Unity 制作的多人在线战争游戏，使用键盘和鼠标作为控件，其设计和执行方式结合了第一人称和第三人称战争游戏的特点。每个玩家必须在游戏中使用自己的账号注册后登录，然后加入一个满是其他玩家的大厅，然后选择自己的类别和团队，开始一轮游戏。射击游戏在作品中渗透多元化的文化输出，引导他人在游戏中更加生动形象的学习各种知识[1]。在游戏中每支队伍最多有 6 名成员，比赛开始时，每支队伍都试图在对敌方士兵的比赛中达到最大的歼灭率，并获得最好的总分，直到回合结束。在回合结束时，玩家可以选择离开游戏，加入另一个大厅或留在同一个大厅并开始玩另一轮游戏。

1.1 多人游戏

多人游戏玩家可以在同一时间和其他玩家在线上进行对战，一个玩家的行为会影响其他玩家，以及游戏的结果总的来说是整场比赛。多人游戏可以在本地或通过网络进行。当所有玩家都连接到同一网络(本地服务器)时，使用“本地”(所有的计算机连接到同一路由器)不需要互联网连接。如今，实现本地多人连接的方式已经不像以前那么流行了，但现在仍在使用有些情况下，比如网吧里的电脑，私人比赛，或者仅仅是朋友在网上玩同一所房子。第二种选择是使用网络技术通过互联网连接，所有玩家或客户端都连接到负责同步它们的服务器或主机可以同时玩同一个游戏。大多数多人游戏类型——例如射击游戏，策略、角色扮演和比赛——提供多人游戏选项；然而，在这个项目中将专注于射击类。

多人游戏广受欢迎，成功的原因是当有其他人与之竞争时，通常会发现游戏更具竞争力和趣味性而不是使用人工智能。人工智能或 npc（非玩家角色）是游戏开发行业仍然需要大量工作和巨大改进；单人游戏中的玩家通常会研究 NPC 的行为，并试图利用它，特别是在 npc 的设计很糟糕，在这种情况下，玩家很容易学会如何对抗它的动作很快，游戏对玩家来说不再具有挑战性。

以多人游戏为例，玩家可以预测游戏中会发生的一些策略但是，在大多数情况下无法准确预测敌方玩家的移动。

人类玩家能够决定自己的行为，从错误中学习，尝试不同的方法战略，并为一个团队或个人运作；这就是为什么在智能达到同样的潜力之前，它不可能是一个合适的替代品。当然，这是多人游戏之所以受欢迎，并不是唯一的原因：在多人游戏中，玩家可以和自己的朋友一起加入游戏，作为一个团队一起玩，这会让玩家的每一场游戏都更加愉快。排名系统有助于多人游戏的普及，也使玩家之间

更具挑战性，更能激励与队伍中的其他玩家竞争，使玩家渴望根据自己的技能水平达到可能的最高级别。最成功的多人游戏会组织顶级玩家和团队的比赛。相互对抗以赢得奖品和认可，鼓励粉丝和玩家成为在游戏中成功。这通常会导致更多的新玩家和老玩家加入游戏更频繁地比赛，努力获得更高的排名。

1.2 射击游戏

射击类游戏是动作类游戏的一个子类，玩家通常被描绘成一个有武器的士兵他手中的武器、小工具和其他配件在帮助他消灭敌人。多人射击游戏目前有多种不同的游戏模式，比如团队模式死亡竞赛，死亡竞赛，统治，夺取旗帜，战斗皇家，等等。本项目的游戏模式是团队死亡竞赛，两个对立的团队在同一个平台上相互竞争选定的地图上，一支队伍的玩家试图淘汰别的玩家直到一个队明确获胜为止。射击游戏的摄像机视图有时可能会有所不同，但最常见的情况是是第一人称射击手（FPS）和第三人称射击手（TPS）。还有其他一些选择，比如 top 向下或固定摄像头俯瞰地图，但它们不像前两个那样受欢迎。第一人称 person shooter 是一种射击游戏类型，游戏摄像头代表玩家的眼睛，并且以他的双手为中心，双手通常握着他的主要武器和配件。概念这类游戏旨在让玩家体验真实的战斗/战争，并将场景描述为尽可能现实。着重实现角色控制和三人称视角跟随、基于 Unity 物理系统射线的 AI 机器人控制等。本游戏搭建逼真的三维游戏场景，以射击加闯关为游戏剧情，采用触控式的控制操作，给玩家带来更加灵活的操控、更易沉浸其中的游戏乐趣[4]。

1.3 游戏引擎

从零开始创建游戏是一项极其困难的任务，因为从一开始程序员需要有各种各样的工具供使用；简单地说，需要一种方法要将图形投影到屏幕上，模拟物理，实现代码/逻辑、声音、动画，显示用户界面元素以及使游戏尽可能完整所涉及的更多方面。信息技术很明显，如果开发者想在不使用外部工具的情况下自己制作游戏，首先需要预先创建这些组件中的每一个，这既困难又耗时任务即使是一个有经验的开发人员，也至少需要几年的时间才能成功能够继续游戏的实际开发。为了避免这个问题，开发者和公司使用游戏引擎。游戏引擎是一个软件开发环境，一个编辑程序，为用户提供所有创建游戏所需的工具。游戏引擎通常提供 3d 世界编辑空间（环境）：渲染引擎、物理引擎、实现代码、动画、声音和动画的工具更多的游戏功能。正如前面提到的，创建游戏引擎是一项极具挑战性的任务，比创造一个真正的游戏要难很多倍，这就是独立游戏的原因开发者、较小的工作室甚至大公司通常决定使用第三方游戏引擎发展。然而，也有一些公司选择开发自己的游戏引擎，避免需要适应其他人可能做出的改变对于引擎来说。游

戏引擎是公司的财产,不是免费的,因此是免费的其他开发者要想获得这些服务,他们必须购买引擎或支付订阅费鉴于开发该软件的游戏工作室愿意将其出售。两个最受欢迎的免费游戏引擎目前市场上有 Unity3D 和 Unreal Engine,它们都有各自的优势和优势弱点。Unity3D 游戏引擎提供了友好的用户界面,并且支持跨平台开发。用 Unity3D 开发的游戏可通过 Unity3D 打包程序打包成不同平台的安装包,从而实现跨平台开发[3]。

在 Unity3D 和虚幻引擎之间,Unity3D 被选为游戏引擎开发 TPS,主要是因为 Unity3D 在早期阶段对开发人员来说更容易。

其主要优点如下:

对于初学者来说,这是一个相对简单的游戏引擎,因为它的简单和模块化的用户界面和插入工具。

目前,它的商店、资产商店和其他一些商店中有大量免费或付费资产其中大部分是由 Unity 创建的,但大部分是独立 Unity 开发者和建模师。

它与所有可用操作系统兼容,包括 Windows、Mac、Linux、Android 和 IOS。它与目前市场上的每一款新游戏机都兼容,包括 PlayStation 4、Xbox 和任天堂交换机。

它支持大多数新的技术,包括虚拟现实、增强现实、混合现实。它有大量的教程,因为有大量的开发者和工作室使用它。

Unity 与许多重要公司关系密切,这意味着他们正在合作,或者将在未来合作,为引擎添加更多功能。例如,Unity 就是一个例子已经发布了两个新的渲染管道,并宣布了一个新的和改进的物理。

Unity3D 是一个非常灵活和强大的工具,不仅用于创建游戏,还用于似乎无穷无尽的其他用途,如创建模拟器、增强现实应用程序、android 应用或全息投影。它与其他应用程序开发相比的主要优势像 Android Studio 这样的程序是如何轻松地实现 3D 元素并使应用程序更具沉浸感。

UI 系统主要处理人机交互和场景切换,通过异步累加配合异步加载实现了场景的快速切换和 2D 场景与 3D 场景的叠加显示[2]。应用程序主要处理与用户交互的界面(简称 UI)组装在画布实体中。Unity Engine 实体是一个二维的、明确定义的空间,用户界面资产位于其中放置。画布实体上显示的资产包括:文本框、按钮、下拉列表下拉菜单和图表及其所有功能。这就考虑到了结构和易用性对开发者和用户的操作。

C#: Unity 引擎默认支持的语言,C#是一种面向对象的编译类型语言,它为程序提供了清晰的结构并允许代码被重用,自带 GC(垃圾回收),从而降低了开发成本,适合游戏脚本开发。

Golang: Google 开源的编译型语言，主要应对现代多核 CPU 的高并发处理，自带 GC(垃圾回收)，非常适合游戏服务器这种高并发的处理。

远程过程调用是一种进程间通信技术，用于基于客户端-服务器的应用程序。它也称为子程序调用或函数调用。客户端有一个请求消息，RPC 将其翻译并发送到服务器。此请求可能是对远程服务器的过程或函数调用。当服务器接收到请求时，它会将所需的响应发送回客户端。客户端在服务器处理调用时被阻塞，只有在服务器完成后才恢复执行。

远程过程调用中的事件顺序如下：

客户端存根由客户端调用。

客户端存根进行系统调用以将消息发送到服务器并将参数放入消息中。

消息由客户端的操作系统从客户端发送到服务器。

服务器操作系统将消息传递给服务器存根。

MongoDB 是一个基于文档的 NoSQL 数据库系统，用于处理大量数据。成为 NoSQL 意味着它的数据存储和检索结构不同于传统的 SQL 数据库，后者使用表格关系。MongoDB 使用文件收集系统，这些文件可能大小和内容各不相同。本文中使用 MongoDB 的目的是从检索到的数据中得出的并被 TPSGame 应用程序利用。

2 系统概要设计

2.1 系统需求分析

功能需求:通过对目前商业大部分 FPS(第一人称射击游戏)的调研和分析，该系统确定实现的功能模块包括登陆注册、游戏大厅管理、玩家在线管理、游戏玩法。

系统架构:系统采用 C/S 架构(客户端/服务器)是常用的两层架构来搭建项目，其次服务器端按照功能模块分，降低项目代码的耦合性，使代码逻辑结构更加清晰，易于项目的维护和版本升级，也增加了程序的可移植性，Unity 也支持一套代码多端导出，服务端代码不需要丝毫变动。系统前台使用 Unity3D 和 UnityUI 搭建和开发，后端基于 GoWorld 和 MongoDB 搭建开发环境和存储数据，之后服务器只要对外暴露服务端口即可。

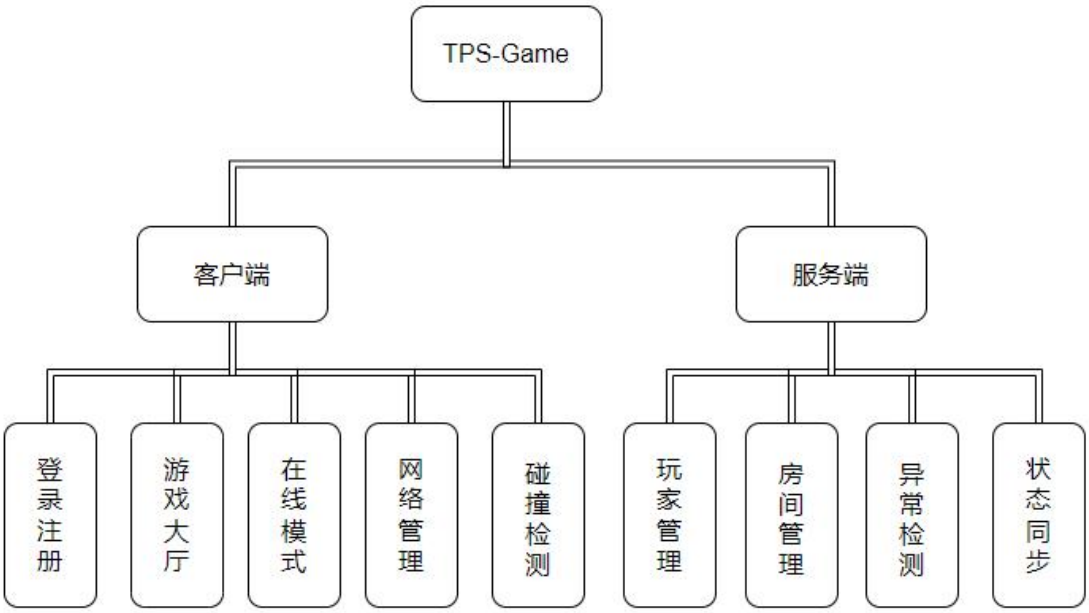

图 2-1 系统结构图

2.2 总体流程设计

在登录注册部分游戏客户端和服务端是根据 RPC 调用进行账号密码的传递和检测，注册时会先查询 MongoDB 是否有账号密码，如果存在则返回 ERROR；如果不存在则生成玩家的唯一 ID 进行入库返回 OK；之后用户即可登录，登录和注册类似，也是先进行 RPC 调用，MongoDB 进行数据的校验，正确且存在则返回账号的唯一 ID 并在服务端生成 Player(登录之前为随机生成的 Entity 登录之后会删除，并将客户端控制权交给新生成的 Player)。

游戏房间管理则完全在服务端 Game 部分，通过维护一个 map 管理所有的房间(为了统一房间 ID 为 0 作为大厅并且不设置人数上限),普通房间人数上线为 12 人并且存储在一个 slice 内(存储 Player 唯一的 ID)；如果有玩家死亡则删除 slice 内的 Player，直到最后一人胜利；如果房间内 slice 为 0，且 5 分钟内没有玩家加入，则服务端会自动删除该房间，防止一直占用服务端资源；每个房间还会存在一个房主进行游戏的开始，如果房主退出房间则权限会自动交给下一名玩家，这些步骤都会在服务端去完成，这是为了防止外挂的产生。

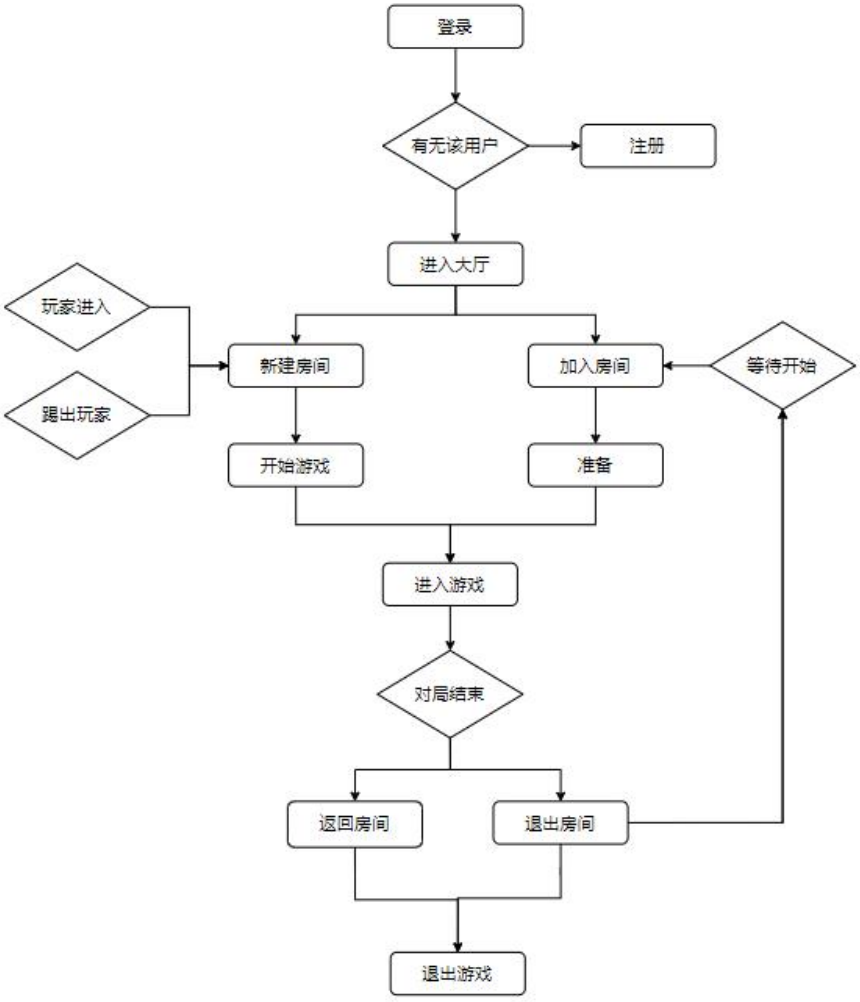

图 2-2 系统流程图

游戏内部管理主要做了三件事情玩家位置和状态的同步、玩家之间攻击的碰撞检测、异常数据的检测(防止外挂的一种手段);玩家之间的位置同步仅仅是同步当前 Space 房间里面的玩家位置，并且是每帧都要同步，每帧都通过 tcp 连接像服务端同步玩家 x、y、z 轴的坐标信息，服务端收到数据包并广播到房间里，当玩家改变状态时(比如:Idle->Move)则是通过 RPC 调用，服务端然后发送 RPC 调用同步改变每个客户端相应 Player 的状态；玩家之间的碰撞检测也类似与玩家状态改变，当检测到碰撞后会调用服务端的 RPC 服务判断伤害值，被攻击到的玩家血量大于 0 则正常扣血，并且通过 RPC 服务同步到相应客户端，如果血量小于等于 0 则通过 RPC 服务同步到客户端玩家死亡，并且服务端要删除对应的 Player，同时判断一下游戏是否符合结束条件，不符合则不做处理；异常数据的检测主要就是对数值类型的数据做检测，例如收到客户端玩家移动的数据包检测玩家当前的位置和数据包同步的位置做插值计算，如果插值过大的数据一定是有问题的，服务端对数据进行修改再同步发送出去，伤害值也是同理。

3 数据库设计

3.1 数据库需求分析

数据库用于存储系统所需要处理的数据，该系统的数据库存储的数据包括用户的个人信息、账号等级信息、持有武器信息，各个部分数据存在多种关系。

当用户喜欢一个游戏时，用户通常想要创建一个账号，通过这种方式获得了丰富的功能，比如记录用户的记录，用户的表现，用户已经玩了多长时间的游戏，或者能够保持一个朋友名单。大型商业游戏通常有大型数据库，可以存储所有玩家的账户，以及每个账户的详细信息，玩家的进度和游戏中的货币。这些数据库大多是用 SQL 编写的，并填充有大型表包含所有需要的细节；然而，在例子中，没有必要建立如此庞大的数据库，虽然在 Unity3D 中使用 SQL 构建一个相对容易。

通过分析该系统的需求构建数据库表存储物理模型，确定每个表的字段名和数据类型，在保证数据查询效率的情况下参考三范式对数据库进行设计，减少了数据存储的冗余。数据库设计的优劣在很大程度上影响数据的存储安全和查询速度。针对该设计需要两张表，第一张表用来存储玩家的详情信息(Player 表)主要存储一些 user_id、create_time、state、lv、name 字段信息，第二张表主要存储用户密码信息(PassWord 表)主要存储 user_id、password 字段信息。

3.2 数据表设计

数据字典是指对系统数据库数据进行建模，包括对数据结构、数据属性、数据类型、数据项、数据关系、数据存储等属性进行定义和描述，把抽象的概念模型转化为可供数据库操作的物理模型。数据字典以表格的形式构建，描述了数据库数据存储的方式，如表 3.1、3.2 所示。

表 3.1 玩家详情表 —— Player

属性	类型	是否为空	长度	描述
id	int	NOT NULL	11	主键，自增
user_id	string	NOT NULL	11	玩家唯一标识
create_time	datetime	NOT NULL		创建时间
state	tinyint	Default 0	1	玩家状态
lv	int	NOT NULL	3	玩家等级
name	string	NOT NULL	10	玩家昵称

表 3.2 用户密码表 —— PassWord

属性	类型	是否为空	长度	描述
user_id	string	NOT NULL	11	玩家唯一标识
password	string	NOT NULL	10	玩家账号密码

4 系统功能模块设计

4.1 登录场景设计

TPS 背后的主要思想是开发一款多人游戏，因此实施基于账号的系统是不可避免的。每个多人游戏都有玩家帐号；有些游戏可能会选择以访客玩家身份登录。

系统采用面向服务的设计思想,在游戏客户端使用 C#语言编写脚本完成动作和控制,引入有限状态机、寻路系统、战斗系统等提升游戏的体验[5]。

登录注册部分主要是通过一个 UI 界面去实现的，登录主要实现方式是玩家点击登录按钮会触发 OnLogin 函数，函数内部通过 username 和 password 获取到玩家的输入进行简单的数据格式校验，然后发送 RPC 请求到后端进行验证；注册的逻辑和登录相同仅仅是调用的 RPC 服务不相同。登录完成之后客户端和服务端都会新建一个 Player 对象用来存储玩家的一些信息，直到玩家下线这个对象才会被删除。

核心代码:

```
public void OnLogin() {
        // 登录
        Debug.Log ("Login " + usernameInput.text + ", " + passwordInput.text);
        string username = usernameInput.text.Trim ();       // 获取账号密码
        string password = passwordInput.text;
        if (username == "") {                                // 校验数据格式
            showMessage ("Username is empty!");
            return ;
        }
       ...
       // 调用 RPC 服务登录
       GoWorld.ClientOwner.CallServer ("Login", username, password);
    }
```

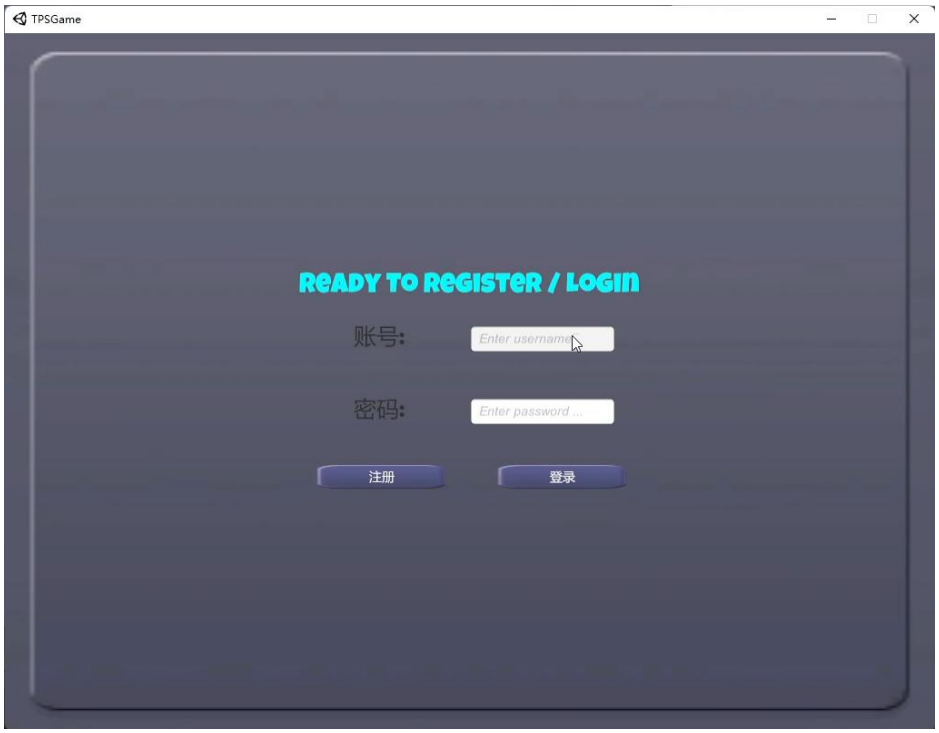

图 4-1 登陆注册

4.2 游戏大厅设计

玩家成功登录其帐户后，立即转到大厅菜单。为了让玩家能够选择想要进入的游戏房间，大厅是必要的；也许玩家会选择一个 ping 更少的服务器，或者选择一个朋友拥有的房间进行团队比赛。游戏细节例如：游戏选项、玩家数量、玩家颜色、所需团队等。用户界面（UI）为由脚本控制，该脚本还控制所有玩家的联网操作：在更简单地说，大厅 Admin 决定游戏规则。大厅有两个功能：它可以在本地/手动模式下使用在线服务器。

这部分实现是被动的实现方式，即服务器主动向客户端推送 JSON 数据，服务端维护了一个房间的 List 当有人新建房间时会主动往这个 List 里添加数据，之后服务器会触发一个 RPC 调用，把 List 数据发送给所有在线的客户端，之后通过 UpdateRoomButton 函数对 JSON 数据进行渲染。

核心代码:

```
// 获取所有房间的列表数据
int num = int.Parse(jo["RoomNum"].ToString());
if (num <= 0) return;
for (int d = 0; d < Content.transform.childCount; d++)
```

```
    {
        Destroy(Content.transform.GetChild(d).gameObject);
    }
    // 通过遍历渲染房间 UI 列表
    for (int tmp_I = 0; tmp_I < num; tmp_I++)
    {
        ...
        button.GetComponent<Button>().onClick.AddListener(delegate
{IntoRoom(button.name);});
        button.transform.SetParent(Content.transform);
        Debug.Log(jo.ToString());
    }
}
```

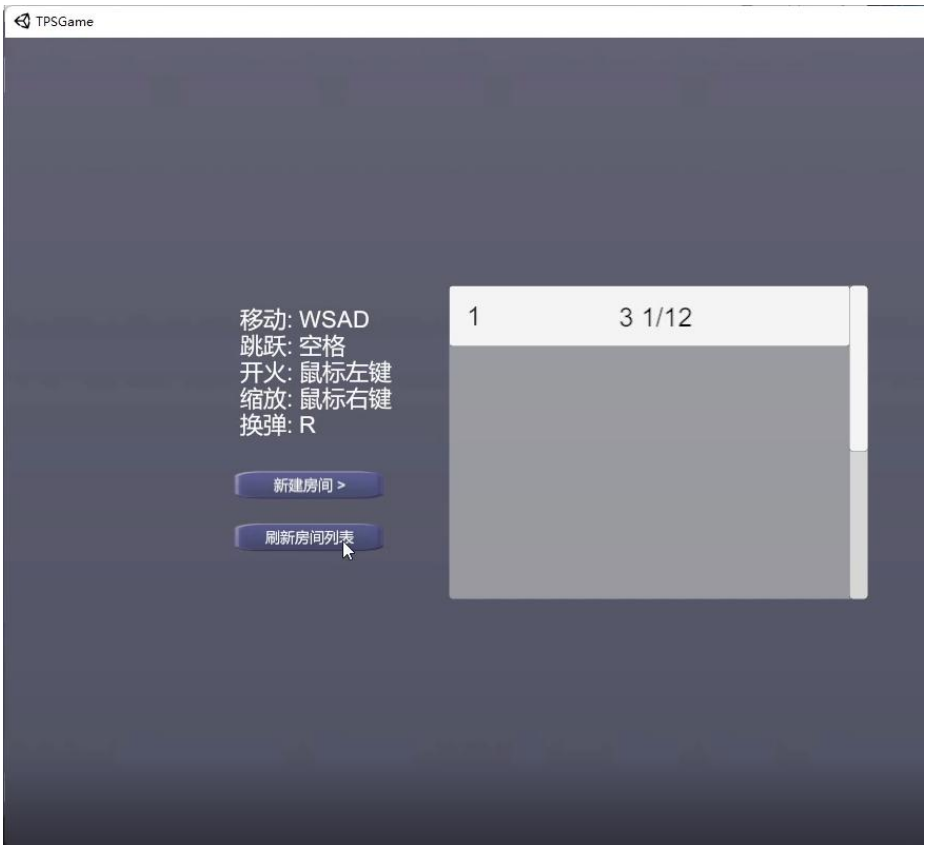

图 4-2 游戏大厅

4.3 在线模式

在线模式，用户不是在本地创建一个房间，而是创建一个服务器房间任何使用游戏连接互联网的人都可以看到并加入的房间。在 Unity 页面中打开，需要登录，然后设置游戏的多人模式选项，例如游戏房间中的最大玩家数。

在线模式里面房间(space)通过后端服务进行维护，每当玩家进入某个房间时会触发一个 RPC 调用，这个 RPC 调用会传递两个 ID(玩家 ID、房间 ID)，后端收到后会根据玩家 ID 将对象实体存放到对应的房间里，之后服务器会通过 RPC 服务通知到所有在当前房间的玩家，客户端通过 UpdateRoomPlayerList 函数将得到的 JSON 数据进行解析渲染到 UI 界面上。

核心代码:

```
private void UpdateRoomPlayerList()        // 玩家进入房间时调用
{
    Debug.Log(joRoomPlayers.ToString());
// 获取房间的 JSON 数据
    int num = int.Parse(joRoomPlayers["Num"].ToString());
    if (num <= 0) return;
    for (int d = 0; d < One.transform.childCount; d++)
    {
        Destroy(One.transform.GetChild(d).gameObject);
    }
    ...
    // 遍历 JSON 数据并且渲染在 UI 上
    for (int tmp_I = 0; tmp_I < num; tmp_I++)
    {
        if (AccountID + "(房主)" ==
joRoomPlayers["Players"][tmp_I]["Id"].ToString() ||
            AccountID + "(房主)(我)" ==
joRoomPlayers["Players"][tmp_I]["Id"].ToString())
          ...
        }
    }
}
```

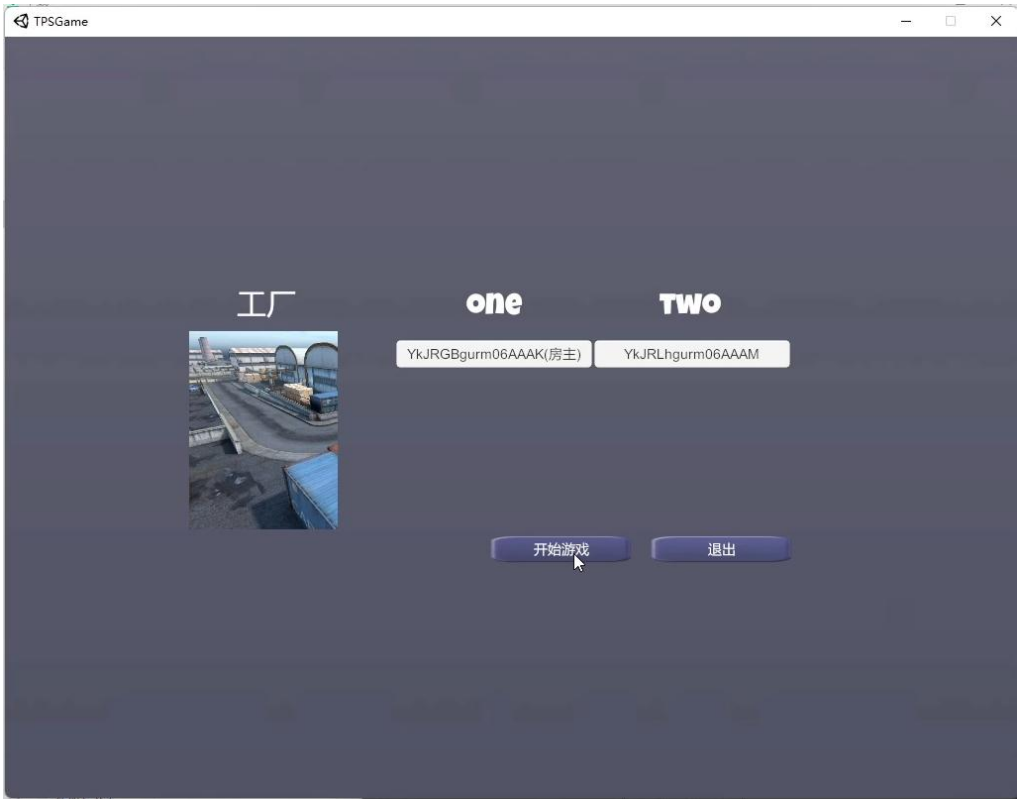

图 4-3 游戏房间

4.4 游戏逻辑处理

网络游戏玩家是带有各种脚本和其他游戏对象为不同的目的而连接到它的对象。在游戏开始时，所有附加到玩家开始并行运行，相互通信，这使得服务器解释这个项目相当困难。在以下步骤中，将尝试将项目分为较小的部分，并为每一个部分的工作方式进行简要解释。

(1).创建玩家:玩家登录之后服务器和客户端都要创建一个玩家对象。

(2).进入房间:当玩家之间进入同一个房间之后，服务器就要通知当前房间里的所有客户端都创建一份房间内玩家的对象。

(3).游戏开始:游戏开始之后服务器将实时同步每个玩家的状态，以及检测每位玩家是否被淘汰，淘汰之后可以通知所有客户端对该玩家做出一些处理。

(4).游戏结束:游戏结束之后玩家可以选择返回大厅或者直接退出游戏。

核心代码:

```
private void Update()            // 每帧调用更新角色状态
{
    if (!carriedWeapon) return;
    AmmoCounttextlabel.text = carriedWeapon.GetCurrentAmmo.ToString()+" / "+
    carriedWeapon.GetCurrentMaxAmmoCarried.ToString();
```

```
    if (Input.GetMouseButton(0) || serviceIsFire)    // 玩家控制射击
    {
        serviceIsFire = true;
        carriedWeapon.HoldTrigger();
    }
    // 瞄准
    if (Input.GetMouseButtonDown(1))            // 玩家取消瞄准
    {
        carriedWeapon.Aiming(true);             // 更改 UI 显示
    }
    if (Input.GetMouseButtonUp(1))              // 玩家开启瞄准
    {
        Quaternion rot = ak47Camera.transform.rotation;
        PlayerFollowCamera.SetActive(true);     // 改变玩家视角
        PlayerFollowCamera.transform.rotation = rot;
        ak47Camera.SetActive(false);
        carriedWeapon.Aiming(false);            // 更改 UI 显示
}
```

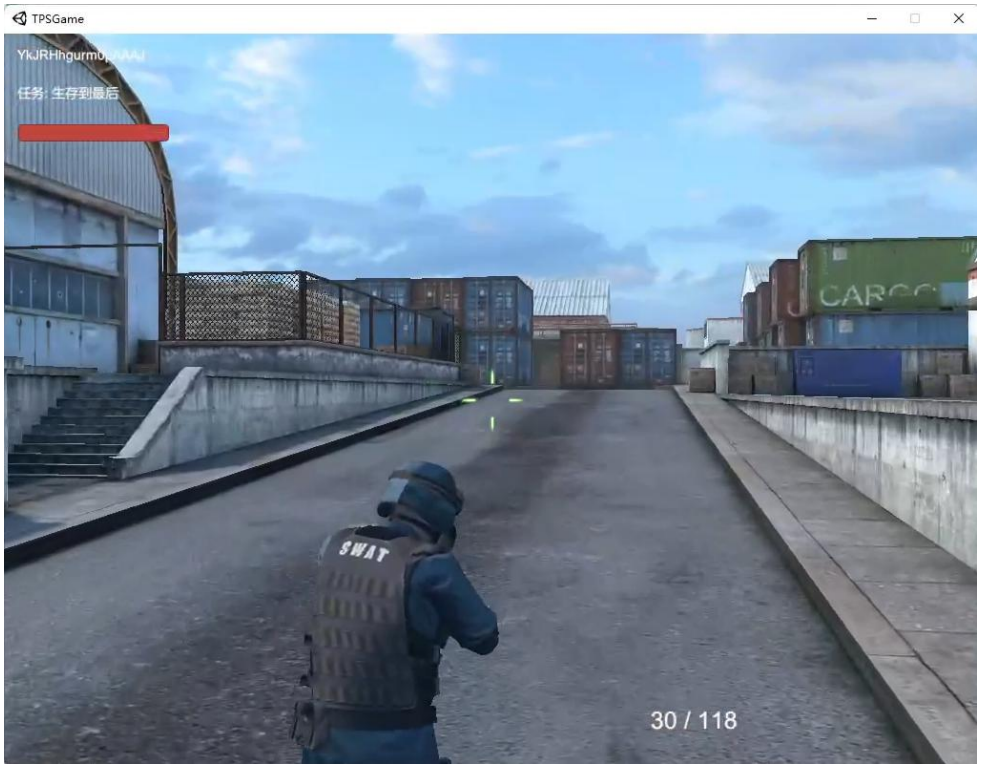

图 4-4 游戏运行

4.5 游戏核心模块

该游戏的逻辑与现代多人射击游戏类似，是一种尝试模拟两个团队之间的现代战争场景。这个游戏的玩家由一个能够以各种方式移动并发射子弹或使用其它小玩意以消灭敌人。玩家的目标是获得最多的淘汰率。有可能造成死亡。当然，个人得分很重要，但玩家的帮助也很重要，队友之间作为一个整体发挥作用，并以最佳方式获得胜利。而一支满是狙击手的队伍在远距离得分时表现强劲，但在近距离战斗时表现明显较弱搏斗。

这一部分比较核心的就是碰撞检测的实现，当玩家点击鼠标左键之后会触发一个 Shoot 的 RPC 服务，这个 RPC 服务会携带三个参数(玩家的 YAW、PITCH、枪口的 xyz 坐标)，服务端会加上玩家的 ID 和这三个参数同步给所有在当前房间的玩家,客户端收到数据包进行解析调用 CreateBullet 函数生成子弹并通过三个参数调整子弹的发射方向和位置，通过 Physics.Raycast 可以获取到子弹是否碰撞到物体，假如子弹碰撞到物体，要通过 GetComponentInParent 获取到 Player 的对象 ID，至此不仅知道子弹碰撞到的是玩家还知道碰撞到了是哪位玩家，如果没有碰到玩家获取到 Player 的对象为空，之后需要做的就是调用 ShootHit 这个 RPC 服务，这个服务传递两个参数(碰撞到的玩家 ID、玩家 body)，服务器收到这个 RPC 服务之后会根据玩家 ID 根据不同的 body 部分做出不同的扣血操作，当血量扣为 0 或者小于 0 时，触发玩家死亡，某位玩家死亡会同步给房间内的所有玩家对其对象进行 delete 删除，否则的话仅仅对扣血的玩家做出 RPC 响应即可，这个响应传递扣完血的数据。

核心代码:

```
private void CreateBullet()
{
    // 绘制射击出的子弹
    gunLine.SetPosition (0, gunBarrelEnd.transform.position);
    if(Physics.Raycast (gunBarrelEnd.transform.position,Muzzlepoint.forward,
out shootHit, range, shootableMask))
    ...
    // 碰撞检测
    if (Physics.Raycast(Muzzlepoint.position, Muzzlepoint.forward, out
shootHit, range, shootableMask))
    {
        // 判断是身体还是头部
        string id =
```

```
shootHit.collider.gameObject.GetComponentInParent<Player>().ID;
            Debug.Log("击中玩家：  "+id);
            string body = "1";
            if (shootHit.collider.gameObject.name == "tou")
            {
                body = "0";
            }
            cli.CallServer ("ShootHit", id, body);
        }
}
```

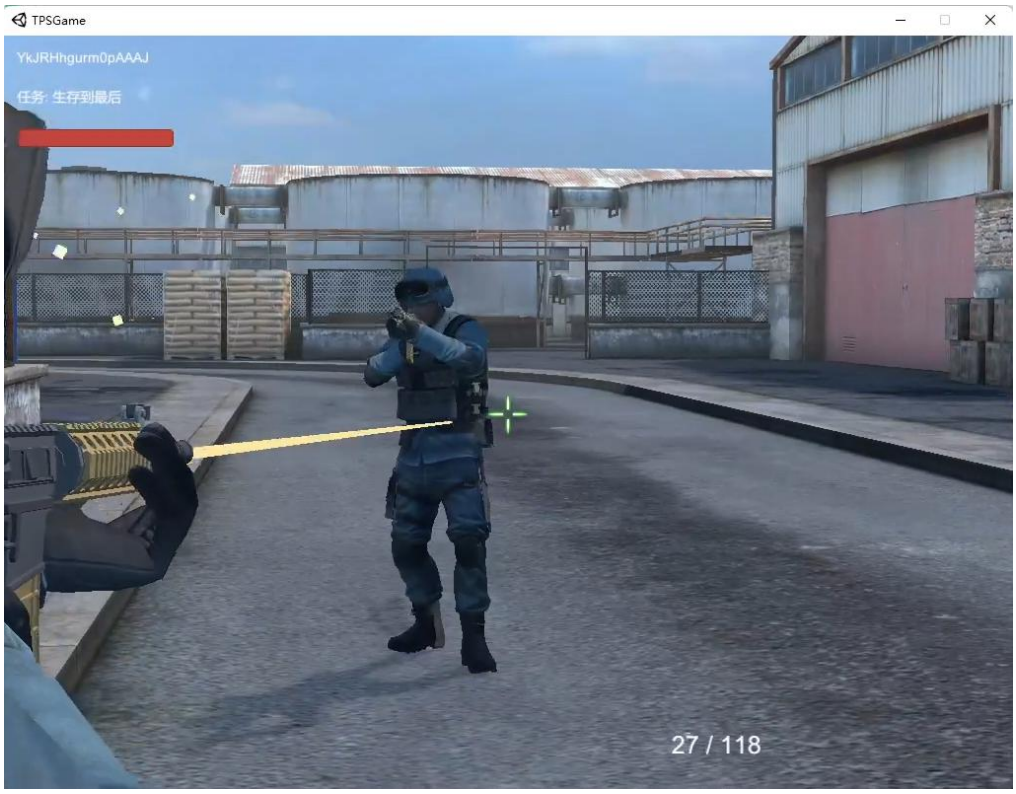

图 4-5 攻击碰撞

4.6 游戏结束处理

游戏结束条件多种多样根据玩家创建的 Space 进行确定，控制游戏结束主要在服务端，例如生存模式(胜利条件只剩一名 Player 生存(HP>0))：全局有 12 个玩家，每当一个玩家死亡之后(HP<=0)服务端就要判断当前这个 Space 空间游戏是否需要结束，如果只剩一名玩家则结束游戏并且让最后一名玩家胜利，否则当前被击败的玩家显示 OUT 并在服务端删除该 Player。

核心代码:

```
public void Fail()
{
// 取消玩家的控制权
   pla.enabled = false;
// 遍历删除对象节省内存
   for (int d = 0; d < HealthBar.transform.childCount; d++) {
      Destroy(HealthBar.transform.GetChild(d).gameObject);
   }
// 失败界面展示
   Canvasdead.SetActive(true);
   PlayerArmature.SetActive(false);
   Cursor.visible = true;
   Debug.Log ("Player is destroyed");
}
```

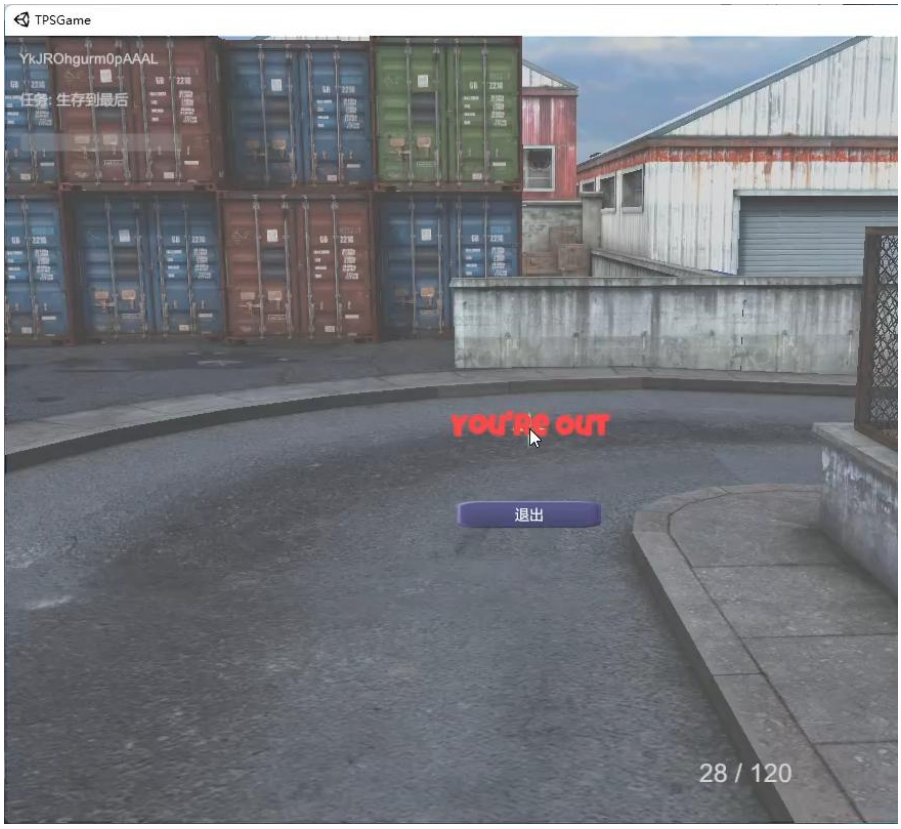

图 4-6 游戏结束

4.7 网络连接

网络可能是该项目最重要的方面，因为在多人游戏中，每个用户正在运行游戏实例或游戏副本，并且由服务器同步所有让用户的游戏实例看起来就像在同一个游戏实例中玩一样游戏简单地说，当一个玩家在游戏实例中移动时，其他玩家连接到同一台服务器上，必须看到玩家在其实例中执行完全相同的动作，这需要实时完成。时机是一个关键因素，例如，如果一个玩家射击他的武器必须在完全相同的时间在网络上同步；否则，如果操作是延迟传输后，服务器将失去同步，从而改变其操作的原始结果。

网络部分采用的是 tcp 长连接的方式，当玩家打开客户端界面时会自动通过 tcp 的三次握手连接到服务器，连接成功之后界面的提示信息会更改为注册或登录，之后直到玩家主动关闭客户端为止这条 tcp 连接会一直保持数据包的通讯。

数据包的格式主要分为三类，第一类为位置同步数据包它包含每个玩家的 x，y，z 位置信息和玩家的唯一 ID；第二类数据包为状态同步包，包含玩家当前的状态信息和玩家的唯一 ID；第三类数据包是信息获取数据包，主要是做一些获取数据列表以及玩家对 UI 界面的操作，比如加入房间和获取房间列表。

5 服务器功能模块设计(服务器)

首先介绍一下服务端组成部分：Entity 实体对象在游戏中代表玩家、怪物、NPC 之类的对象，用 Space 空间代表游戏中的场景，当用户登录之后将游戏客户端控制权交给一个 Player 对象，之后就可以使 Entity 或 Player 在不同的 Space 空间之间跳转实现房间切换。这样的抽象使得游戏的网络通信模式较为统一，能够在框架层做更多的功能，顶层逻辑无需关心数据同步，能提高游戏开发效率。并且使整个服务端结构变得很简单，容易扩展。

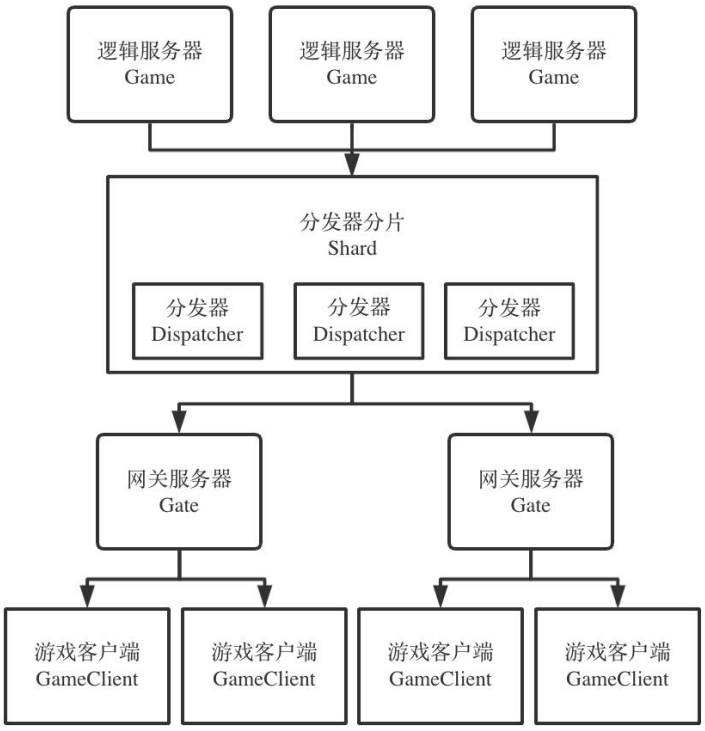

图 5-1 服务端架构

服务端共分为三个线程，分别是逻辑服务器 Game 负责游戏具体的逻辑处理，也是 Entity/Player/Space 活动的地方；dispatcher 负责消息分发；Gate 网关主要就是管理游戏客户端的连接。

5.1 服务端连接管理

连接管理主要在服务端的Gate部分,Gate启动时默认是一个(至少要有一个),有多个Gate的情况下服务端将采用负载均衡的方式进行连接的处理(目前内置两种方式：轮询方式、哈希方式)；

Gate 的核心部分在于 GateService，所用功能用 GateService.Run 进行启动，默认启动 Tcp 的连接方式，启动后 Gate 会进入一个循环，实际上就是处理来自客户端的消息和来自 Dispatcher 的消息以及处理定时器的消息。

当客户端连接后，服务端会新建 ClientProxy，即客户端代理，用它来存储客户端所需的信息，只有知道这些信息才能够将实体与连接关联。如下的结构中，最重要的信息是代表连接的 GoWorldConnect 和代表控制实体的 clientid。

一个完整的客户端连接过程:首先客户端连接时 gate 会创建一个 clientProxy 结构体，创建完成后会选一个 dispatcher 发送新客户端连接的信息。发送的数据包括消息类型 MT_MOTIFY_CLIENT_CONNECTED，以及实体 ID(gate 生成的一个唯一 ID)。Dispater 在收到 gate 的消息后，会随机选取一个 game，发送有客户端

登录的消息。game，当它收到 dispatch 关于客户端连接的消息后，game 就会创建一个 BootEntity，也就是 Account。这里面代码的 MakeGameClient 以及赋值的内容其实只是让 entity 记录和它关联的客户端连在哪个 game 上。

核心代码:

```
type ClientProxy struct {
    clientid         common.ClientID                    // 客户端 ID
    clientSyncInfo clientSyncInfo                  // 客户端锁
    heartbeatTime   time.Time                          // 心跳包时间
    ownerEntityID   common.EntityID                    // 所有者实体 ID
}
// 主要流程 for 循环一直处理不同的消息类型
func (gs *GateService) mainRoutine() {
    for {
      select {
      case pkt := <-gs.clientPacketQueue: // 客户端数据包队列
        ...
      case pkt := <-gs.dispatcherClientPacketQueue://调度程序数据包队列
          op := opmon.StartOperation("GateServiceHandlePacket")
          ...
          break
        case <-gs.ticker:
          gs.tryFlushPendingSyncPackets()
          break
        }
        post.Tick()
    }
}
```

5.2 服务端消息处理

dispatcher 分发器的任务就是分发消息，dispatcher 也是分布式结构，可以开启多个。每个 dispatcher 的职能相同，仅仅为分解压力，dispatcher 开启监听，game 和 gate 都连接上它。分发器的任务是消息分发，它需要知道每个连接体的信息。对于每个连接进来的 game，dispatcher 会保存如下的结构，主要记录 game 的 id 等。对于每个连接进来的 gate，会保存 dispatcherClientProxy 的结构，记录 gateid。dispatcher 还必须包含实体在哪个 game 的信息，假如通过 rpc 去调

用其他 game 的实体，就必须要知道实体所在的位置。所以对游戏服中的 entity，dispatch 要保存的信息封装在如下的 entityDispatchInfo，记录了它所在游戏服的 Id。所以每个 Dispatch 至少要保存所有的 game、所有的 gate 和一些 entity 的信息，并且 entity 以 id 为索引，能够很快查找到各个 entity 在哪个 game。解析创建实体的过程。由自定义代码在 game 中创建实体，例如调用 createEntitySomeWhere 让 goaworld 在指定的 game 创建指定类型的实体。createEntitySomeWhere 会发送信息给某一个 dispatch，goworld 会根据 entity 的 id 做哈希选择一个 dispatch 去发送。当 dispatch 收到消息后，会选择要转发的 game，如果参数 gameid 是 0，它就随机选取一个 game，否则转发给参数指定的 game。在 game 收到消息后，它最终会调用 EntityManager 中创建实体的方法，代码如下。它会根据要创建的类型反射创建 entity，然后将它放到 nilSpace 中。最后再通知 dispatcher 说有 entity 被成功创建。当 dispatch 收到后，做些后处理即可完成实体的创建。

核心代码:

```
type gameDispatchInfo struct {
    gameid              uint16                    // 游戏 ID
    clientProxy         *dispatcherClientProxy  // 客户端代理
    isBlocked           bool                      // 是否阻塞
    blockUntilTime      time.Time
    pendingPacketQueue []*netutil.Packet  // 未处理的数据包队列
    isBanBootEntity     bool
    lbcheapentry        *lbcheapentry
}
func (edi *entityDispatchInfo) dispatchPacket(pkt *netutil.Packet) {
    if edi.blockUntilTime.IsZero() {
        // 处理数据为 0 的情况
        dispatcherService.dispatchPacketToGame(edi.gameid, pkt)
    }
```

5.3 服务端游戏逻辑处理

Game 具体实现分为四个部分，每一部分分别完成了相应的功能;第一部分 Account 它包含了一个字符串存储用户名称、一个 Entity 结构体存储实体信息、一个 bool 值表示用户是否在登录过程，Account 实现了 Entity 主要包含了 Login 和 Register 的 func，以及获取房间列表的 RoomList 的 func；第二部分 Player 包含一些玩家的属性字段比如血量、状态、位置等信息，SetAction 和 Shoot 为

只要的 func;第三部分 MySpace 自定义 TPS 的空间，核心功能 CheckForDestroy 该 func 检测空间自身是否需要删除；第四部分 SpaceService 核心功能用来管理所有 Space 房间，并且通过 IsVictory 来判断当前房间游戏是否需要结束，并通过 EnterSpace 来使 Player 新建或进入对应的 Space 房间。

核心代码:

```
// setDefaultAttrs 设置玩家的一些默认属性
func (a *Player) setDefaultAttrs() {
    a.Attrs.SetDefaultStr("name", "noname")
    a.Attrs.SetDefaultInt("hp", 100)
    a.Attrs.SetDefaultInt("hpmax", 100)
    a.Attrs.SetDefaultStr("action", "idle")
    a.SetClientSyncing(true)
}
// ShootHit_Client 射击  body 0=头部 1=身体
func (a *Player) ShootHit_Client(id string,  body string) {
    a.CallAllClients("Shoot")                    // 调用射击 RPC
    victim := a.Space.GetEntity(common.EntityID(id))
    if victim == nil {
        gwlog.Warnf("Shoot %s, but monster not found", id)
        return
    }
```

5.4 对象实体

entity 定义的代码如下，最重要的属性有代表唯一编号的 ID，代表实体在哪个场景中的 Space，代表实体属性的 Attrs，Attrs 保存着 DescribeEntityType 中用 DefineAttr 定义的属性。其中的 client 是一个 GameClient 成员，若玩家对该 entity 拥有控制权，那么它身上会有 GameClient 结构，记录连接在哪个 gate 等信息。接着看看 entity 从场景跳转到另一个场景的过程。逻辑层调用 EnterSpace 后，game 会发一条请求迁移的信息给 dispatcher，指明哪个实体要迁移到哪个 space 的哪个位置，第一步是请求 space 是在哪个 game。diapatch 收到信息后，回应要进入的 space 是在哪个 game。game 收到后，知道要往哪个 game 迁移，于是又发送迁移请求给 dispatcher。dispatcher 收到后，会 block 要迁移的 entity，也就是先缓存所有对该 entity 的消息队列，然后通知 game，让它可以做迁移准备了。game 收到回应后，会生成迁移消息，迁移消息包括该 entity 所有需要保存的数据，比如 attrs 里面的数据，然后销毁它，再把迁移数据发送给 dispatch。

dispatch 收到后，将迁移消息转发给新 space 所在的 game。新的 game 会恢复 entity 的信息，完成迁移。restoreEntity 除了恢复数据，还会把 entity 放入新 space 中。

核心代码:

```
type Entity struct {
    ID                  common.EntityID       // 全局唯一 ID
    TypeName            string                // 类型名称
    I                   IEntity
    V                   reflect.Value
    destroyed           bool
    typeDesc            *EntityTypeDesc
    Space               *Space                // 所在空间
    Position            Vector3               // 三维空间位置
    InterestedIn        EntitySet
    InterestedBy        EntitySet
    aoi                 aoi.AOI
    yaw                 Yaw                  // 偏航
    rawTimers           map[*timer.Timer]struct{}
    timers              map[EntityTimerID]*entityTimerInfo
    lastTimerId         EntityTimerID
    client              *GameClient           // 客户端实体
    syncingFromClient   bool
    Attrs               *MapAttr
    syncInfoFlag        syncInfoFlag
}
```

6 系统测试

系统测试是在系统需求规范或功能需求规范或两者的上下文中对整个系统进行的。系统测试测试系统的设计和行为以及客户的期望。

6.1 系统测试的类型

本系统采用黑盒测试：

黑盒测试是一种软件测试，用于测试软件产品或应用程序的功能性、稳定性和可靠性。

6.2 测试环境

操作系统：Windows 10，Centos7

内存：服务器 2G，客户端 16G

数据库：MongoDB

6.3 系统测试条目及其结果

表 6.1 功能测试条目及其结果

功能	使用数据	设计目标	描述
登录	用户名、密码正确	进入游戏	进入游戏
登录	信息输入错误	提示	提示用户错误原因
进入房间	房间未满	进入房间	进入房间
进入房间	房间已满	进入失败	提示用户房间已满
玩家死亡	血量大于 0	不做处理	玩家正常继续游戏
玩家死亡	血量归 0	当前玩家结束游戏	禁用玩家操作
游戏结束	房间内玩家大于 1	不做处理	房间正常继续游戏
游戏结束	房间内玩家等于 1	当前房间游戏结束	最后一名玩家胜利

7 结论

总之，TPS 是一款独立的多人射击游戏。这个其背后的理念是努力以最少的成本实施尽可能多的机制可能的并发症和错误。本文从架构和游戏开发的基础知识去全面讲解了一个计算机游戏程序的开发过程，一个计算机程序如何分层解耦，以及如何处理网络连接。当然，这不是一个成熟的游戏，可以被视为对 AAA 公司构成严重威胁，但作为个人项目，它已经达到并超过了最初设定的目标。同时学习开发一款游戏也是对个人计算机知识的一个整合，因为这其中牵涉到这知识面非常之广，不但包含编程语言也涉及了计算机图形学甚至是线性代数等等，促使所有知识融会贯通。最重要的因素是这些技术的发展方式能够以尽可能少的必要更改和返工支持大量不同的添加。该项目有很多机制，旨在支持大量额外投资不需要进行根本性重建的内容。并且能对目前的一些计算机技术和游戏行业有一定的了解。

参考文献

[1] 徐军,张子墨.基于 Unity3d 射击游戏的设计及其核心功能实现[J].福建电脑.2018(7)

[2] 郭东方.基于 Unity3D 坦克战争游戏的设计与实现[D].河北科技大学.2018

[3] 李智鹏.基于 Unity3D 引擎的空中战机游戏设计与实现[D].吉林大学.2016

[4] 董涛.张瑛.基于 Unity3D 的第三视角射击类手游设计与实现[J].通讯世界.2019(11)

[5] 唐迪.基于 Unity3D 引擎的第一人称射击游戏设计与实现[D].电子科技大学.2021

[6] 盛剑涛.基于 Unity 引擎的角色扮演类手游的设计与实现[D].华中科技大学.2019

[7] 张俊,廖金巧.基于 Unity3D 的手机版 FPS 射击游戏设计与开发[J].赤峰学院学报.自然科学版.2016(15)

[8] 冯旭,伊程毅,何元生.基于 Unity 三维游戏引擎的消防科普宣传系统[A].2020 中国消防协会科学技术年会论文集[C].2020

[9] 贺苗元.基于 Unity3D 引擎的虚拟室内漫游的研究设计与应用实现[D].内蒙古大学.2015

[10]韩大鹏.基于 Unity3D 引擎的手机游戏客户端的研究与实现[D].西安电子科技大学.2014

[11]林佳一.《Unity3D 应用开发》课程教学的探索和实践[J].现代计算机.2020(2)

[12]肖俊.基于 Unity3D 的跨平台手机网络游戏的研究与实现[J].计算机产品与流通.2018(2)

[13]唐捷.基于 UNITY3D 的小游戏开发[J].电脑编程技巧与维护.2016(23)

[14]邓珍荣,黄文明,张敬伟,李丽芳.基于 J2ME 手机坦克大战游戏设计及实现[A].广西计算机学会 2008 年年会.论文集[C].2008

[15]Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, et al. Unity: A General Platform for Intelligent

[16]M. McPartland and M. Gallagher, “Reinforcement learning in first person shooter games,” in IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, 2011, pp. 43 - 56.

致谢

回望我的大学生活，充实而又圆满。一直引导着我前行，也让我的大学生活有了明确的方向感，这都离不开我的导师—黄勇老师和李学勇老师的悉心指导，在毕业设计和写论文期间，我与黄老师多次交流确定设计和需求，黄老师给我提出了很多宝贵的建议。感谢黄老师的帮助和关心才能使我的大学生活丰富多彩。