基于 Unity 的多人在线对战游戏开发

白天　　刘宇飞

摘要—论文摘要是对文章内容不加诠释和评论的简单陈述。一般控制在 200 字左右，建议在论文全部完成后再动手写摘要。

I. 引言

本项目旨在解决 Unity 多人在线对战游戏开发的问题，联机系统为 C/S 模式，使用 Socket 技术独立开发。该游戏是一个基于物理的足球游戏，打开游戏并设定好端口号后即开启了服务端，其他人则只需输入专用服务器的 IP 地址和端口号便可加入游戏，即使是中途加入当前场上的状态也会同步过来；每位连接到服务器的玩家都会在一定规则下被分配到队伍，将球踢入其他队伍球门会加分，踢入自己队伍球门会扣分；玩家在游戏中有着丰富多样的策略，可以通过身体来带球，可以用四个角上的能量棒去“踢”球，还可以通过旋转能量棒改变球的走向，甚至借助蓄力来发动必杀技，一转局势；因为完全基于物理，玩家的移动等操作皆是通过施加力来实现的，不同物体的物理材质也有差异，如运用得当，可完成多次反弹进门等高难度操作；小地图、碰撞效果、蓄力显示等将给予玩家非常直观的反馈。

“独乐乐不如众乐乐”，多人游戏与单人游戏的快乐程度是在不同层次上的，不管是棋牌类的斗地主、麻将，还是竞技类的足球、篮球，亦或是流行的电子游戏《魔兽争霸》《英雄联盟》都为玩家们带来了单人时无法获得的快乐，许多人还会去观看其他人直播游玩多人游戏，从中获得欢愉。多人游戏也极大拓宽了一个游戏的丰富度，一个规则设置和维护得当的游戏，玩家可以游玩数年也不会感到厌倦。从开发的角度上，将一款游戏做成联机游戏的难度也与纯本地游戏是不可同日而语的，从建立连接、收发数据包到同步状态，还要考虑延迟、反作弊等各种复杂问题，Unity 本身也没有内置合适的联网组件，这些对开发者来说都是一种考验。

我们在开题之初便对常用的 Unity 联网游戏实现方式进行了调研。首先，Unity 内置的 Unity Networking（UNet）已经被官方宣布为过时，并将在近两年彻底停止维护，从 Unity 中移除 [?]；官方用于取代 UNet 的新联网组件——基于 ECS 架构的 Unity NetCode，最新版本刚发布到 0.2.0，只是预览测试用的版本，无法正式投入实际开发；对于第三方的解决方案，Mirror 算是基于 UNet 的改进版，但每个连接仅支持一个客户端；SmartFoxServer 知名度低，国内外的文档都比较少；Photon 为 Client/Client 的通信模式，并非我们需要的 Client/Server 模式 [?]，ET 框架是一个组件系统，和我们熟悉的面向对象差异较大。综合考虑各个因素，小组成员决定通过较为底层的 Socket 直接开发我们希望实现的网络联机系统，相关代码完全透明，参考微软提供的.NET 文档，有着高度的自定义性，也不用担心因第三方封装带来的未知 bug。

我们从最基础的连接和收发包开始，先建立一个控制台的服务器和 Unity 中的客户端，接下来将服务器迁移到 Unity 中，随后再将二者合到一个项目中，共用场景。与此同时，本地也有一个测试场景，用来完成上线前的调试工作。网络部分，我们专注于场景中物体的创建、销毁、同步……玩家角色作为游戏中最重要的部分之一，从基础的移动逐渐丰富到能蓄力、能施展必杀技，有些效果的实现可能会对联网部分提出新的要求，此时便会一边开发网络部分，一边在本地开发效果部分。最后，还有一些用来增加游戏表现力的后期处理和辅助开发的实用代码。

II. 相关工作（飞）

A. Unity

Unity 是一个跨平台的游戏引擎，可用于创建三维、二维、VR、AR 游戏，有着可视化编辑、操作简单、文档全面等优点，易于开发学习，深受大众的喜爱。《城市: 天际线》《奥日与黑暗森林》《茶杯头》《人类一败涂地》等知名游戏皆是用 Unity 开发的。

B. C# Socket

套接字是支持 TCP/IP 协议的网络通信的基本操作单元。可以将套接字看作不同主机间的进程进行双向通信的端点，它构成了单个主机内及整个网络间的编程界面。套接字存在于通信域中，各种进程使用这个相同的域用 Internet 协议来进行相互之间的通信。[?]

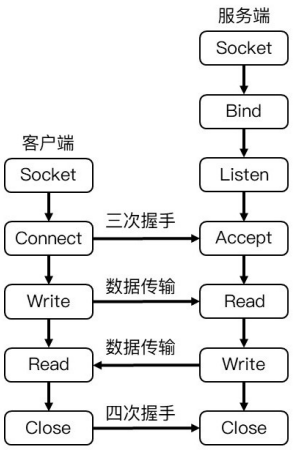


图 1. Socket 通信的基本流程

C. Addressable Asset System

Addressable Asset System 提供了一种按地址加载 Assets 的简单方式，相比 Resources 和 Asset Bundle 更加方便与灵活，能够进行自动化仓储管理和内存管理，只需要一个地址便可以从任意地方加载，默认的所有操作都是异步操作，可以添加事件监听。无需开发者自行收集与管理资源的依赖关系，且提供可视化界面，极其强大。

D. Post Processing

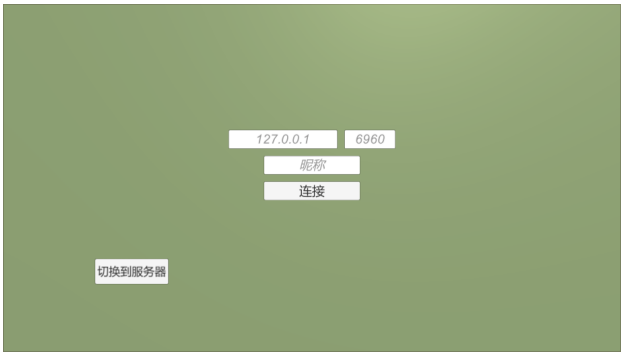
Post Processing 是一个后期效果增强组件，可以在较少的时间内实现辉光、色彩分离、胶片颗粒、景深、运动模糊、镜头畸变、色彩调整等各种常用特效的效果，为整个游戏增光添彩，也减少了自己编写 shader 的困难。

III. 实践过程

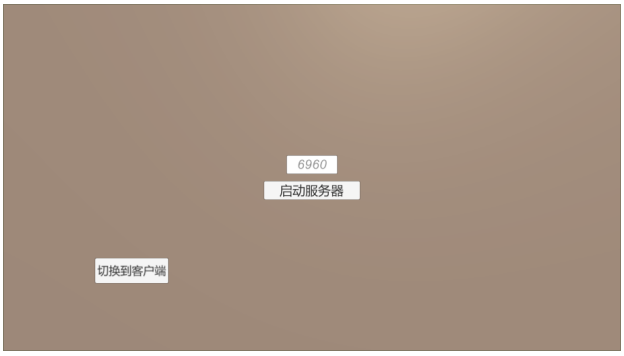
A. Socket 通信

本次实践使用 Socket 技术完成服务端（Server）与客户端（Client）间的网络通信。因为游戏基于 Unity，故底层使用了.Net 框架 System.Net（特别是 System.Net.Socket）命名空间下的内容。与 Socket 通信相关的程序架构，借鉴了网络教程 [?]，并在其基础上加以改进，使之更能满足开发和游戏运行时的需求。

为了方便复用代码逻辑及 Unity 资源，本次实践中将服务端与客户端置于同一项目，共享部分资源（例如一些游戏场景、Prefab、代码逻辑等）。最终构建出的程序可由用户手动选择充当服务端或客户端。



(a) 成为客户端



(b) 成为服务端

图 2. 选择成为客户端或服务端

1) 连接与断开:

本项目在服务端同客户端通信的过程中，同时用到了 TCP 与 UDP 两种传输协议。服务端开启后，会监听其端口上的 TCP 连接请求和 UDP 报文；客户端向指定套接字连接的过程中，会先尝试同服务端建立 TCP 连接，通过 TCP 连接简单交换信息（如客户端 ID 和玩家昵称等）后再向服务端发送 UDP 报文，让服务端对客户端信息形成记录，为之后进行 UDP 通信做准备。

本项目中的所有代码逻辑均处于 BallScripts 命名空间下，并根据性质进一步细分。服务端逻辑在 BallScripts.Servers 命名空间；客户端逻辑在 BallScripts.Clients 命名空间。两命名空间中均包含 Client 类，内含内部类 TCP、UDP，负责处理一个

客户端的 TCP、UDP 相关逻辑。服务端还存在 Server 类，统筹多个 Client 对象；而客户端的 Client 为单例类，管理自身的连接及通信。服务端存在多个 Client 对象，意味着服务端同每个客户端都保持独立的 TCP 连接。但对于 UDP，服务端使用统一的逻辑发送、接收数据，并根据 Client 对象中记录的客户端 ID 及套接字端点来确定要交由哪个 Client 对象做具体处理。

无论是服务端还是客户端，都存在主动断开连接的代码逻辑，即服务端可以主动断开同任意客户端的连接，客户端也可以主动断开同服务端的连接。一方断开时，另一方稍后也会自行关闭连接。

2) 数据包:

服务端、客户端间传输的内容，本质上是一些字节序列，虽然便于传输，却不方便理解与处理。为此，引入 BallScripts.Utils 命名空间下的 Packet 类，作为服务端和客户端共同的一种约定，将字节序列包装成使用起来更方便的数据包。

一个 Packet 对象让代码能够写入或读取一些常见的数据类型（如 int、float、string、Vector3 等），从而灵活地组装或使用字节序列。此外，在构建字节序列并发送的过程中，数据包能够自动为字节序列补充长度信息；如果数据包从客户端发送，则还会包含其客户端 ID。

程序执行到任何逻辑，但凡需要网络通信，便要构建并发送数据包。这些数据包携带的数据往往不同，需要一种标识来加以区分，让接收方知道这些数据包分别代表何种逻辑。这种标识可以通过枚举实现，来自服务端的数据包使用 ServerPackets 枚举；来自客户端的数据包使用 ClientPackets 枚举。标识会以整数的形式附在数据包内。

3) 发送、接收:

服务端和客户端，各提供一组专门的方法来发送、接收不同标识的数据包。

创建并发送数据包的逻辑，在服务端位于 ServerSend 类，在客户端位于 ClientSend 类。两个类中包含若干公有（public）、静态（static）方法，一种持有特定标识的数据包至少会对应其中一个。根据各方法中使用的不同逻辑，数据包会被写入不同的标识、数据，并通过 TCP 或 UDP 发送。在服务端，有时还允许指定数据包的发送对象，可以是所有已连接的客户端或某些特定客户端。

接收并处理数据包的逻辑，在服务端位于 ServerHandle 类，在客户端位于 ClientHandle 类。类中同样有若干公有、静态方法，这些方法的名称与数据包持有的各种标识相对应。例如，在客户端，ClientHandle 中的各方法，名称同 ServerPackets 中的枚举项对应。这些方法的参数列表也被统一规定，在程序初始化时以 ServerPackets 项为键、存储着 ClientHandle 方法的委托为值，纳入字典中统一管理。这样程序收到数据包后，根据其标识即可在字典中找到相应的处理逻辑，进而通过得到的逻辑处理数据包。大致相似的布置和流程在服务端中也存在。

为了不让接收数据包的逻辑同各种游戏逻辑大量耦合，又分别设置了 ServerLogic 类与 ClientLogic 类，用于放置一些真正和游戏逻辑有关的方法，部分方法传入从数据包中得到的数据，运用它们更新场景信息、执行游戏逻辑乃至发送新的数据包。

B. 场景物体

1) 资源管理:

2) 创建:

3) 销毁:

4) 同步:

5) 物理:

C. 玩家角色

1) 移动:

2) 蓄力:

3) 必杀技:

4) 动画:

5) 队伍、球门和得分:

D. 游戏效果

1) 玩家名字:

2) 小地图:

3) 后期处理:

4) shader（天）:

E. 实用代码

1) 单例爷爷:

2) 线程管理: 本节需要详细介绍本文提出/设计的方法。

公式示例：

$$a + b = \gamma \tag{1}$$

式 (1) 是一个演示用的公式。

表格示例。表格的代码可以实现在

表 I
表格

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

http://www.tablesgenerator.com/ 上设计好，然后复制过来。表格 I是一个示例。

图片的示例：图 1是一个点。在表格和图中使用

图 3. Example of a figure caption.

label 可以指定标签，便于后续使用 ref 命令引用。

正文中通过使用 cite 命令引用参考文献。

IV. 实验结果 (飞)

本节题目可以自拟。

主要负责成果展示。

V. 结论

对整个工作做一个总结，得出结论，并展望未来。

未来，我们可以进一步提升游戏性，加入三角、圆形、五角星等新的拥有不同大招的角色，引入场地道具等系统，甚至可以去做新的场地、多个球门、吃鸡模式；性能方面也有较大优化的空间，当前服务器必须以图形界面运行，如果将其改为纯控制台，或许就可以真正部署到运行 Linux 系统的服务器上。