doi:10.16018/j.cnki.cn32-1650/n.201803004

基于 unity 局域网同步与多角色玩家的设计与实现

干建松
(江苏财经职业技术学院 现代教育技术中心,江苏 淮安　223003)

摘要:通过 Unity 2017 的 Unet 模块,使用三维游戏引擎 Unity 的 NetworkManager、NetworkIdentefier、NetworkTransform 等组件,设计并实现局域网同步与多玩家角色场景;通过分析 NetworkMessage、NetworkServer,用 C#程序脚本完整地实现多角色玩家的选择功能;最后于文章结尾处提出了解决网络延迟的初步方法。

关键词:局域网;网络游戏;网络同步;Unity;Unet
中图分类号:TP317　　**文献标识码**:A　　**文章编号**:1671 - 5322(2018)03 - 0019 - 05

在高代价、长周期的商业化开发之前,做一个非常完善的网络游戏系统不仅需要大量的投入,而且开发成功后还需要有足够大的市场,否则将面临着极大的开发风险。为了降低开发风险,事先做一个基于局域网的演示程序(DEMO),既能达到展示目的,又能节省时间,降低开发成本。

Unity 软件是一个轻松创建多平台三维交互、建筑可视化、实时三维场景等类型的综合型专业游戏开发引擎。作为三维游戏引擎和虚拟现实(VR)软件的急先锋,其制作的 VR 产品的覆盖率达到 85%;用 Unity 制作多玩家游戏的用户,通过 NetworkManager 组件,可以将网络游戏开发直接通过服务器端计算,并同步给所有客户端。

目前,国内外许多学者在 Unity3D 平台游戏开发方面做了大量的研究工作,如伍传敏等[1]基于 Unity3D 完成了第一人称射击游戏的设计与开发,张典华等[2]基于 Unity3D 实现了多平台兼容的三维空战游戏,潘志庚等[3]则研究了 Unity3D 与 Kinect 整合数据技术在体感游戏中的应用价值。本文主要通过 Unet 和 C#编程实现局域网客户端游戏角色的选择和网络同步。

1　系统概述

在 C/S 网络结构中,服务器保存着所有玩家和敌人的对象(Object),客户端拥有服务器端场景相同的环境,并且每个客户端都有一个由客户端操控的角色(LocalPlayer)流程。

图 1 为多人游戏场景的消息机制。图 1 中客户端(Client)执行本地操作后,所有命令都在服务器中处理,之后再将处理结果同步到各个 Client 中[4]。

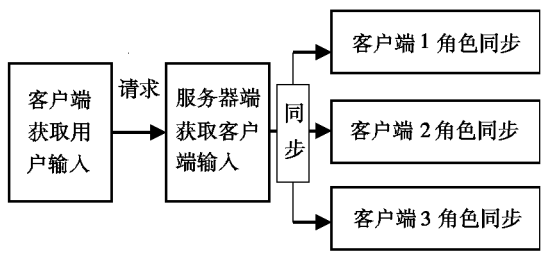

图 1　多人游戏场景消息机制
Fig. 1　Multiplayer game scene messaging mechanism

2　设计与实现

2.1　创建工程

制作地面,导入坦克 Tank,并设计控制脚本 moveCtrl.cs。

```
public class MoveCtrl:MonoBehaviour {
//移动和旋转速度分别为 2m/s 和 30°/s
float moveSpeed = 2.0f;
```

收稿日期:2018-07-08
基金项目:江苏省现代教育技术研究重点课题(2016-R-49468);江苏省大学生创新项目(201713753006Y)
作者简介:干建松(1976—),男,江苏淮安人,副教授,硕士,主要研究方向为虚拟现实、数字媒体。

```
float rotateSpeed = 30.0f;
//设定二维数组 V 存储移动速度和旋转角度
int [ ] v = new int[2]{0,0};
void Update( ){
Move( );
}
void Move( ){
//按下 W、S 键分别前进 1 和 -1,其它键为 0
if (Input. GetKey (KeyCode. W)) {
v[0] = 1;
} else if (Input. GetKey (KeyCode. S)) {
v[0] = -1;
} else {
v[0] = 0;
}
}
//设按下 A、D 键旋转速度分别为 1 和 -1,其它键为 0
if (Input. GetKey (KeyCode. A)) {
v[1] = -1;
} else if (Input. GetKey (KeyCode. D)) {
v[1] = 1;
} else {
v[1] = 0;
}
//按下 WASD 键实现玩家(坦克)的自由移动
transform. Translate (Vector3. forward * v[0] * moveSpeed * Time. deltaTime);
transform. Rotate (Vector3. up * v[1] * rotateSpeed * Time. deltaTime);
}
}
```

2.2 组件设置

2.2.1 设计原理

Network Identity:使用 UNet 生成的物体都需要挂上这个组件,它会分配给物体一个 NetID。通过 NetID 识别本地物体的操作,而其他 NetID 不响应本地操作。

Network Transform:用来同步物体的 Transform。默认同步的方向是从服务器到客户端。

2.2.2 设计过程

(1)给上一步创建的坦克依次挂上 Network Identity 和 Network Transform 组件;

(2)将 Hierarchy 中的坦克拖至 Project 面板中,生成 prefab,如图 2;

图 2 Tank 生成 Prefab

Fig. 2 Prefab that generates Tank

图 2 Tank 生成 Prefab (3)删除 Hierarchy 中原来的坦克对象,保存场景。

2.3 添加 NetManager、UnetHud、UnetTransform 和 UnetStartposition

2.3.1 设计原理

NetworkManager:管理网络多人游戏的状态,实现创建、运行和调试。游戏物体由同一预制件派生,需要挂上 2.2.1 中的角色控制脚本 MoveCtrl. cs。NetworkManager 配置如图 3 所示。

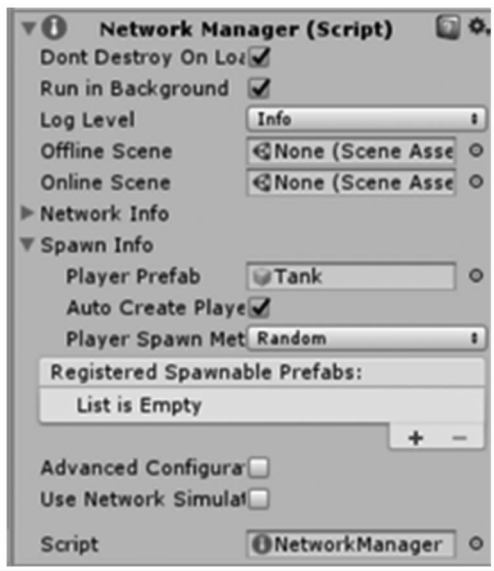

图 3 NetworkManager 配置图

Fig. 3 Configuration diagram of Network Manager

NetworkManagerHUD:提供一个简单的用户界面(UI),显示游戏运行控制网络状态。一般使用 LAN Host 和 LAN Client 功能分别建立主机和客户端,并跟 NetworkManager 一起工作,如图 4。

图 4　NetworkManagerHUD 生成界面

Fig. 4　Generating interface of Network Manager HUD

NetworkStartPosition:控制玩家产生位置(position)的组件,可以挂在场景内任意一个物体上。NetworkManager 寻找场景中 NetworkStartPosition 对象,并在该位置产生玩家。

NetworkBehaviour:将本地玩家(localPlayer)的位置发布给服务器(Host),服务器将本地玩家发过来的消息同步给所有客户端。

LocalPlayer:本地客户端所派生的游戏角色,其所有权由 NetworkManager 分配。当某一客户端成功连接到服务器时,该客户端创造出来的实例(Instance)被标记为 LocalPlayer,而其他的实例则不会。

[Command]:服务器向所有客户端发送更新命令。客户端将本地玩家(LocalPlayer)本地输入的数据请求服务器端运行,并同步至所有客户端。在本地玩家移动后,重复上述动作,即将移动后的数据上传给服务器,经服务器处理后进行分发,并同步至所有客户端。

2.3.2　设计过程

(1)新建一个平面,并合理设置其 X 轴和 Z 轴的绽放比例,此处分别为 5 和 5。

(2)新建一个空物体并将其命名为 GoNetworkManager,调整其位置至 Plane 中心或其他合适位置,再依次通过【Compnent】、【Network】菜单添加以上 3 个组件,然后将 2.2 生成的玩家预制体拖到 NetworkManager 中 Spawn Info 的 Player Prefab 框中,结果如图 3。

2.4　位置同步

2.4.1　调试运行

运行两个游戏窗口,分别设置为 Host 和 Client,发现 Host 端的角色移动,client 端的角色同步更新;但反过来却不行,即客户端(client)角色移动后,主机端(Host)角色并不同步更新。原因是 Unet 是服务器权威的,本地客户端和服务器共享同一个场景,数据不会从客户端向服务器同步,这个方向上的操作叫做命令(Command),用法为在代码前增加[command],见 2.3.1。

2.4.2　设计过程

(1)网络环境。MoveCtrl.cs 中的 MoveCtrl 类需要继承 NetworkBehaviour,而继承这个类之前需要添加命名空间 UnityEngine.Networking。步骤 2.2.1 的文件更改后为:

```
using UnityEngine.Networking;
public class MoveCtrl : NetworkBehaviour
```

(2) 位置同步。将步骤 2.1 中的函数 Move()内容位置放在另一函数[CmdMove()]中,并将其设置在[command]标签中,在客户端调用、服务器执行后,同步到所有客户端。

步骤 2.2.1 中玩家控制部分更改为:

```
//客户端向服务器端发送的命令,在服务器端执行
[Command]
void CmdMove(int[]v){
transform.Translate (Vector3.forward * v[0] * moveSpeed * Time.deltaTime);
transform.Rotate (Vector3.up * v[1] * rotateSpeed * Time.deltaTime);
}
// Update is called once per frame
void Update () {
//isLocalPlayer 实现本地玩家控制本地输入
//否则网络里所有的客户端都会执行
if (isLocalPlayer) {
Move ();
}
}
void Move(){
//移动判断过程同 2.2.1
//CmdMove 函数传递移动和旋转
//并且把命令传送给服务器执行后
```

```
//同步到所有客户端
CmdMove(v);
}
}
```

运行结果如图 5。

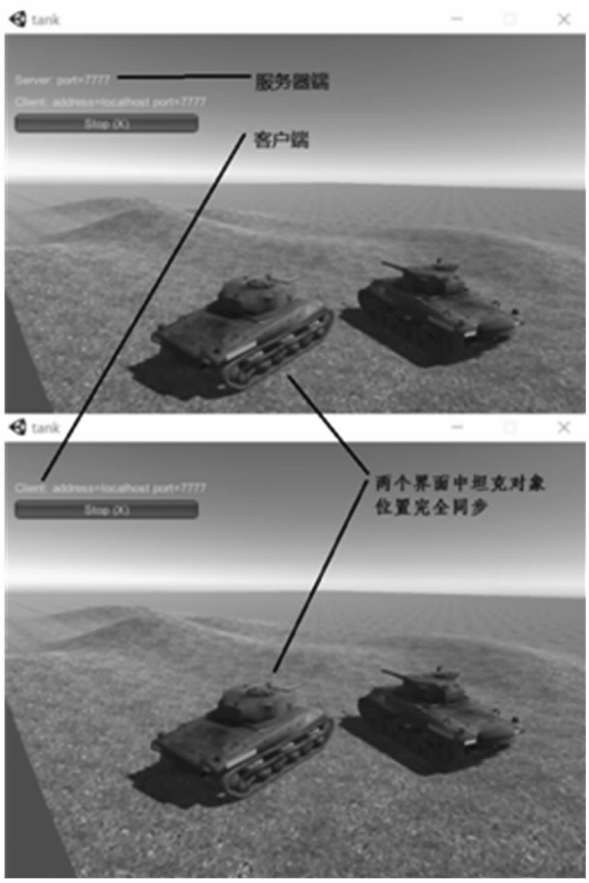

图 5 坦克位置运行同步

Fig. 5 Synchronization run of Tank position

2.5 多角色设计

如果需要增加多角色选择功能,还需要完成以下设计。

2.5.1 设计原理

OnServerAddPlayer:可以动态添加玩家角色。2.2.1 中的 NetworkManager 用于管理预制体实例化的网络对象,如果需要产生多角色、动态化的玩家游戏,必须在 ClientScene 中注册。这可以用 ClientScene. RegisterPrefab( ) 函数,也可以由 NetworkManager 将预制体添加到列表中自动注册。本文是通过调用继承自 NetworkManager 的默认方法动态选择玩家角色的产生。

2.5.2 设计过程

(1)删除 2. 3. 1 中 GoNetworkManager 上的 NetworkManager 组件。

(2)新建 AddPlayer. cs,添加角色选择功能。

```
//设置使用 Networking 命名空间
using UnityEngine. Networking;
//设置继承自 NetworkManager
public class AddPlayer: NetworkManager{
//按 2.1 中的步骤制作第 2 个 prefab
public GameObject[ ] playerPrefabs;
public int index =0;
public class NetworkMessage:MessageBase{
public int index;
}
//用 OnGUI 写两个选择按钮
//设置坐标排列在 NetworkHUD 界面下面
//分别对应两个游戏角色的选择
void OnGUI( ){
if (GUI. Button (new Rect (10, 140, 100, 20), "坦克 1")) {
index =0;
}
if (GUI. Button (new Rect (10, 170, 100, 20), "坦克 2")) {
index =1;
}
}
//实例化函数
//通过按钮实例化玩家角色(Instantiate)
public override void OnServerAddPlayer(NetworkConnection myConn, short playerControllerId, NetworkReader extraMessageReader){
//定义 message 接受消息
NetworkMessage message = extraMessageReader. ReadMessage < NetworkMessage > ( );
int playerCount = message. index;
//选择实例化的角色顺序
GameObject playerPre = playerPrefabs [playerCount];
//实例化玩家角色位置
GameObject player = (GameObject) Instantiate (playerPre, playerPre. transform. position, Quaternion. identity);
//将玩家角色联网 NetworkServer. AddPlayerForConnection (myConn, player, playerControllerId);
}
```

```
//重写客户端联网函数 OnClientConnect( )
public override void OnClientConnect(NetworkConnection myConn){
NetworkMessage test = new NetworkMessage();
test.index = index;
//调用 OnServerAddPlayer( )函数。
ClientScene.AddPlayer (myConn, 0, test);
}
}
```

2.6　最终运行

经过以上设计后,生成. exe 格式的可执行程序;同时运行两个. exe 文件,并分别设置其 host 和client 属性,如图 4。在两个文件中通过“WASD”键操纵实现运动时,另一文件中的坦克玩家即可实现不同坦克角色并且位置同步更新[5]。

3　小结

本文设计了基于 Unet 的局域网同步游戏,实现了不同玩家可以选择生成并操作本地游戏,且在所有其他客户端实现同步更新。其不足之处表现为:在两个实例窗口运行时,本地客户端控制平滑顺畅,而远程玩家游戏中物体移动时会产生顿挫的现象。原因是基于网络的应用都会因服务器与客户端之间数据传输速度受限而存在一定程度的延迟。

优化这种同步延迟有很多途径,如通过提高 2.2.1 NetworkTransform 组件中 Network Send Rate 的同步数据频率,对延迟有明显的抑制,但不会消灭网络延迟。说明服务器与客户端之间网络的刷新频率对多人游戏效果有很大影响,但并不能完全消除。另外,通过包括内插、外插,以及滤波或者预测的方式,可以进一步提升客户端之间同步的平顺性[6]。

参考文献:

[1] 伍传敏,张帅,邱锦明,等. 基于 Unity3D 的 FPS 游戏设计与开发[J]. 三明学院学报,2012,29(2):35-40.

[2] 张典华,陈一民,李磊. 基于 Unity3D 的多平台三维空战游戏的开发[J]. 计算机技术与发展,2014,24(1):192-195.

[3] 潘志庚,刘从晋,葛莹莹,等. 支持自然交互的虚拟跑步机系统的设计和实现[J]. 系统仿真学报,2017,29(11):2753-2759.

[4] 董健. 基于 Unity 平台的漫游交互系统的设计[J]. 软件工程师,2014,17(11):33-34.

[5] 李慎亮,司占军. 基于 Unity 平台的三维坦克游戏设计与实现[J]. 软件导刊,2015,8:152-154.

[6] 汪瑞. 基于 Unity3D 的多通道下服务器客户端同步[J]. 现代计算机,2015,24(1):26-29.

Design and Implementation of LAN Synchronization and Multi-role Player based on Unity

GAN Jiansong

(Modern Education Technology center of Jiangsu Vocational College of Finance and Economics, Huai'an Jiangsu　223003, China)

Abstract: Through the Unet module of Unity 2017, using the NetworkManager, Network Identer, Network Transform and other components of the three-dimensional game engine Unity, we design and implement the LAN synchronization and multi - player role scene. By analyzing Network Message and Network Server, the multi-role player selection function can be realized completely with C# program script. Finally, a preliminary method for solving network delay is proposed at the end of the article.

Keywords: LAN; network game; network synchronization; Unity; Unet

(责任编辑:李华云)