

# UE5.1 制作联网多人射击游戏学习笔记

版本:

UE5.1.1

Windows 10

原课程网址:

<https://www.udemy.com/course/unreal-engine-5-cpp-multiplayer-shooter/#overview>

## 第一部分 制作会话相关的插件

### 005 局域网连接

创建 C++ 第三人称游戏模板，并在 maps 文件夹中创建一个默认关卡大厅 Lobby。

创建三个函数分别创建大厅 (OpenLobby)，连接游戏 (CallOpenLevel, CallClientTravel)，并使用 UFUNCTION(BlueprintCallable) 宏标记以便在蓝图中调用。

具体如下：

角色的.h 文件中添加如下代码：

```
public:
    UFUNCTION(BlueprintCallable)
    void OpenLobby();

    /**
    *Join a game.
    *@include "Kismet/GameplayStatics.h"
    *@param: Address is host IP address.
    */
    UFUNCTION(BlueprintCallable)
    void CallOpenLevel(const FString & Address);

    UFUNCTION(BlueprintCallable)
    void CallClientTravel(const FString& Address);
```

.cpp 文件中添加如下代码：

```
void AMPtestingCharacter::OpenLobby()
{
    UWorld* World = GetWorld();
    if (World)
    {
        World->ServerTravel("/Game/ThirdPerson/Maps/Lobby?listen");
    }
}

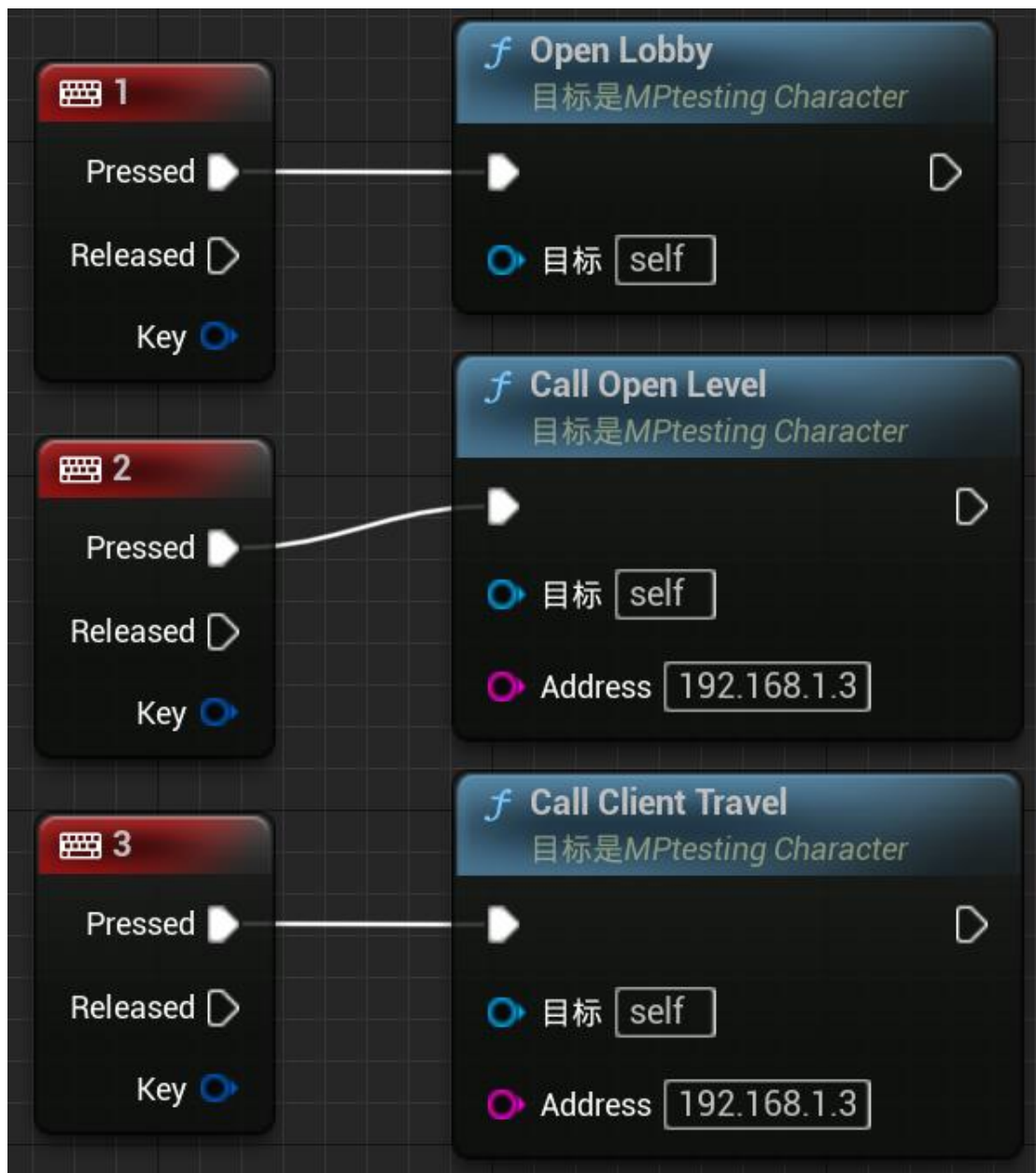
void AMPtestingCharacter::CallOpenLevel(const FString& Address)
```

```

{
    UGameplayStatics::OpenLevel(this,*Address);
}
void AMPtestingCharacter::CallClientTravel(const FString& Address)
{
    APlayerController* PlayerController =
    GetGameInstance()->GetFirstLocalPlayerController();
    if (PlayerController)
    {
        PlayerController->ClientTravel(Address,ETravelType::TRAVEL_Absolute);
    }
}

```

UE 中，打开角色蓝图，并且如下连接：



## 006 在线子系统

在线子系统官方文档:

<https://docs.unrealengine.com/5.1/zh-CN/online-subsystem-in-unreal-engine/>

问: 我们怎么找到需要加入的游戏呢?

两种方法:

专用服务器: 一种方法是使用一个专门的服务器记录运行游戏。但是服务器需要成本, 人越多成本越高。

监听服务器: 第二种方法是使用监听服务器。以一个玩家作为 host, 其他玩家通过 IP 地址加入 host。我们在玩多人游戏的时候并不会去专门记别人的 IP 地址, 登录的时候往往只有一个简单的 play 按钮。利用监听服务器为了实现这个功能就会用到监听服务器。

如果你想要自己实现这些功能, 为了实现安全性保障, 维护好友系统之类的功能, 你所需要的精力甚至超过了游戏的编程, 但如果你需要让你的游戏能够在 Xbox 或者 steam 这些平台上运行, 你又需要去学习 Xbox 和 steam 这些平台所提供的网络服务。

UE 在线子系统: UE 有一套自己的在线服务系统, 根据你的游戏需要, 你可以自己配置需要的在线游戏所需的功能, UE 引擎在平台层实现了针对各个平台的细节, 所以对于游戏开发而言只需要了解 UE 的在线子系统, 实现所需的功能而不再需要去关注各个平台上的具体实现 (如果要了解游戏引擎的结构, 可以参考 games104 课程)。

## 007 在线会话

在线子系统: 用来连接在线服务, 处理平台提供的服务接口

平台服务: 好友, 成就, 游戏会话等

会话接口: 用于创建, 管理以及销毁在线会话; 用于查找会话或者进行匹配。

会话: 是一个服务器上运行的游戏的实例, 玩家可以自由加入, 或者只能通过邀请加入。

会话的生命周期:

创建会话, 等待加入, 注册玩家, 会话启动, 游玩, 结束会话, 注销玩家, 更新或销毁会话。

我们需要关注的会话接口功能:

CreateSession(), FindSessions(), JoinSession(), StartSession(), DestorySession()

GamePlan:

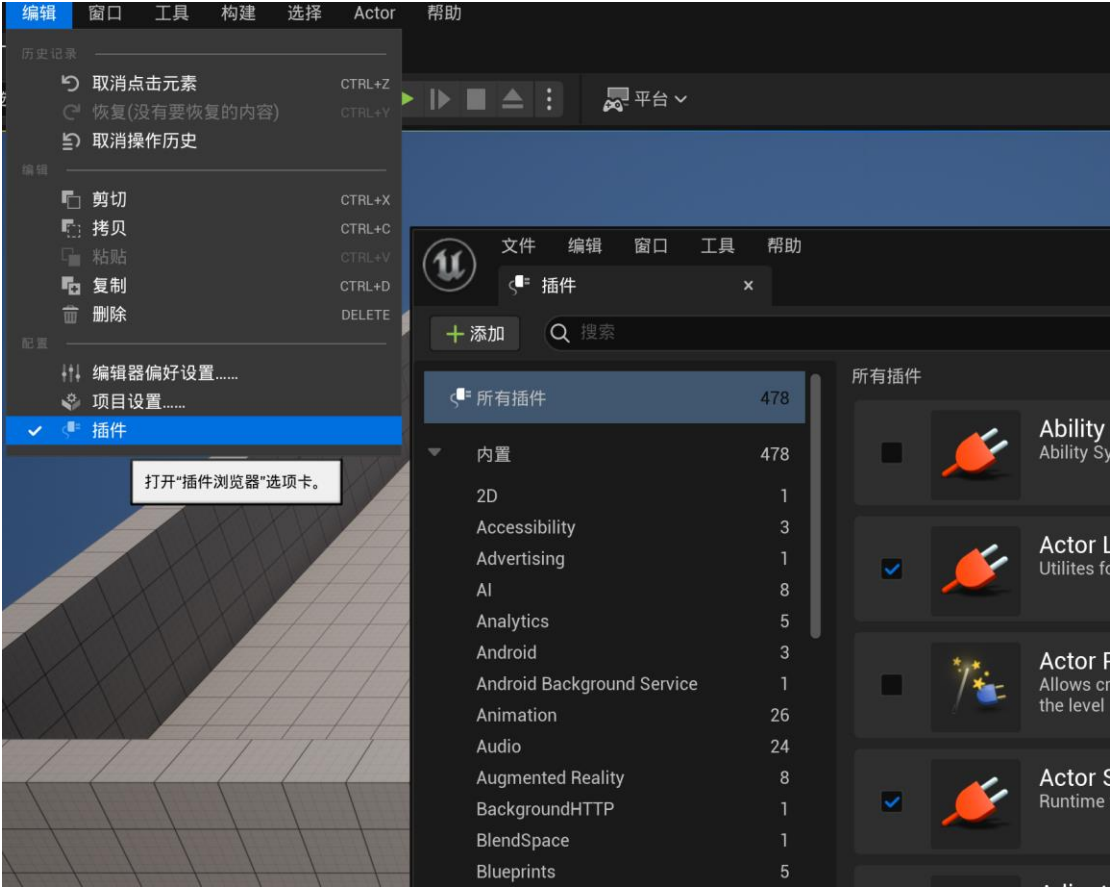
Class for handing Sessions:

Host: CreateSession()=> OpenLobby()

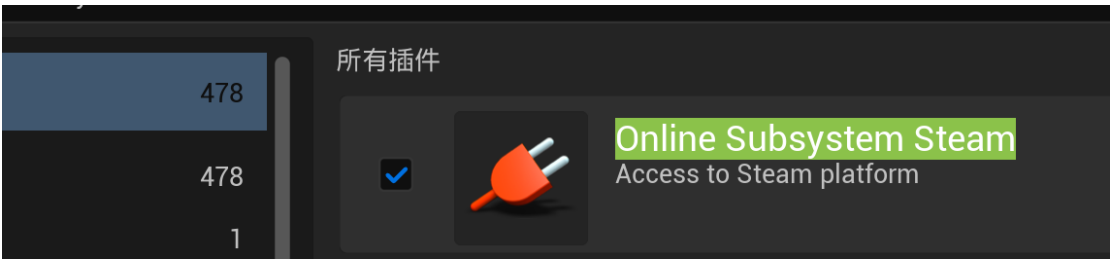
Join: FindSessions()=> JoinSession()=> CallClientTravel

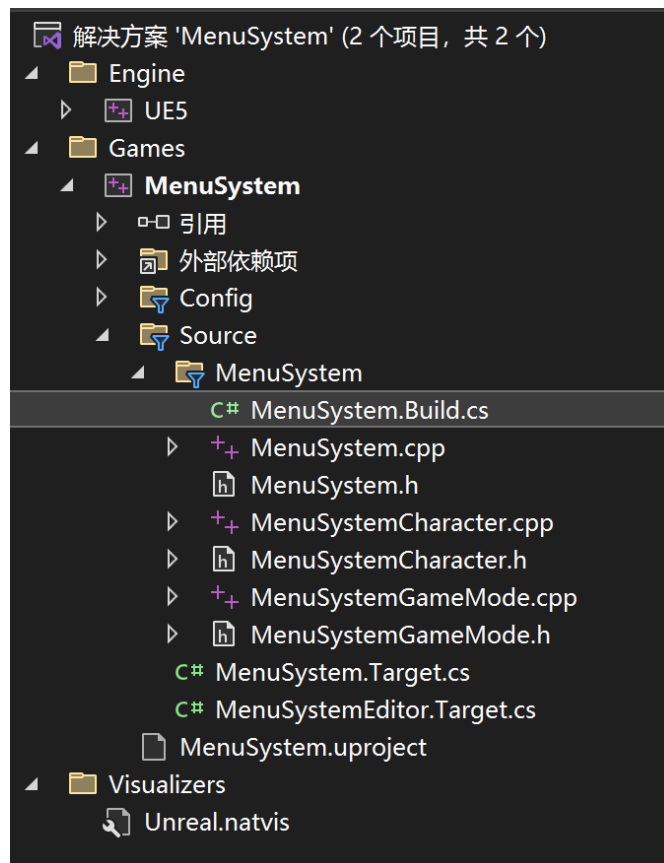
# 008 为 steam 配置工程

创建一个新的 C++第三人称工程，命名为 MenuSystem  
接下来为 steam 配置在线子系统。点击编辑中的插件，



之后搜索下图插件并激活，按提示重启 UE。





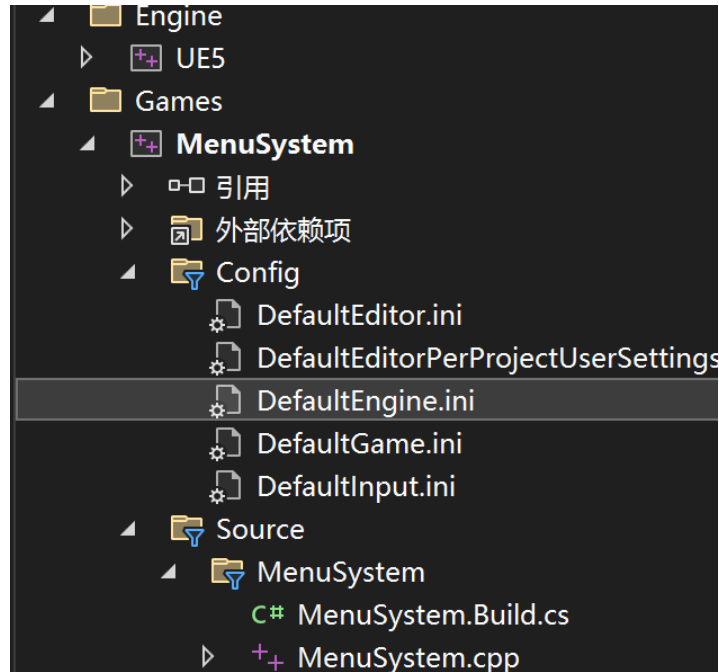
之后打开 VS，选择 Build 文件 MenuSystem.Build.cs

在 `PublicDependencyModuleNames.AddRange` 之后添加两个 module

```
PublicDependencyModuleNames.AddRange(new string[] {  
    "Core",  
    "CoreUObject",  
    "Engine",  
    "InputCore",  
    "HeadMountedDisplay",  
    "EnhancedInput",  
    "OnlineSubsystemSteam",  
    "OnlineSubsystem"  
});
```

`OnlineSubsystem` 是全体的在线子系统

`OnlineSubsystemSteam` 是针对 steam 平台的在线子系统，`OnlineSubsystem` 将利用它来接入 steam



编译通过后找到配置文件 DefaultEngine.ini 并将 <https://docs.unrealengine.com/5.1/zh-CN/online-subsystem-steam-interface-in-unreal-engine/> 中“完成的设置”中的代码复制进来并保存。

之后关闭 VS 和工程文件，找到工程所在的文件夹删除如下三个文件夹。

	.vs	2023/4/12 18:49	文件夹	
<input checked="" type="checkbox"/>	Binaries	2023/4/12 18:49	文件夹	
	Config	2023/4/12 19:14	文件夹	
	Content	2023/4/12 18:54	文件夹	
	DerivedDataCache	2023/4/12 18:50	文件夹	
<input checked="" type="checkbox"/>	Intermediate	2023/4/12 18:54	文件夹	
<input checked="" type="checkbox"/>	Saved	2023/4/12 19:05	文件夹	
	Source	2023/4/12 18:49	文件夹	
	.vsconfig	2023/4/12 18:49	VSCONFIG 文件	1 KB
	MenuSystem.sln	2023/4/12 18:49	Visual Studio Soluti...	5 KB
	MenuSystem.uproject	2023/4/12 18:53	Unreal Engine Proj...	1 KB

之后右键点击.uproject 文件, 生成 VS 项目, 然后再双击.uproject 文件选择 yes 以生成 Binaries 文件夹。

至此，项目可以成功连接 steam 了。

## 009 连接在线子系统

教程里说到需要把会话相关的代码打包起来，以供反复使用，但是为了简单，这次直接写在角色类里面。

在 VS 中打开 character.h 和.cpp 文件。

在.h 文件中的添加一个指针：

```
public:
    // Pointer to the online session interface
    //class IOnlineSessionPtr OnlineSessionInterface;
    TSharedPtr<class IOnlineSession, ESPMode::ThreadSafe> OnlineSessionInterface;
```

有关 `IOnlineSession` 类的说明可以参考文档：

<https://docs.unrealengine.com/5.1/en-US/API/Plugins/OnlineSubsystem/Interfaces/IOnlineSession/>

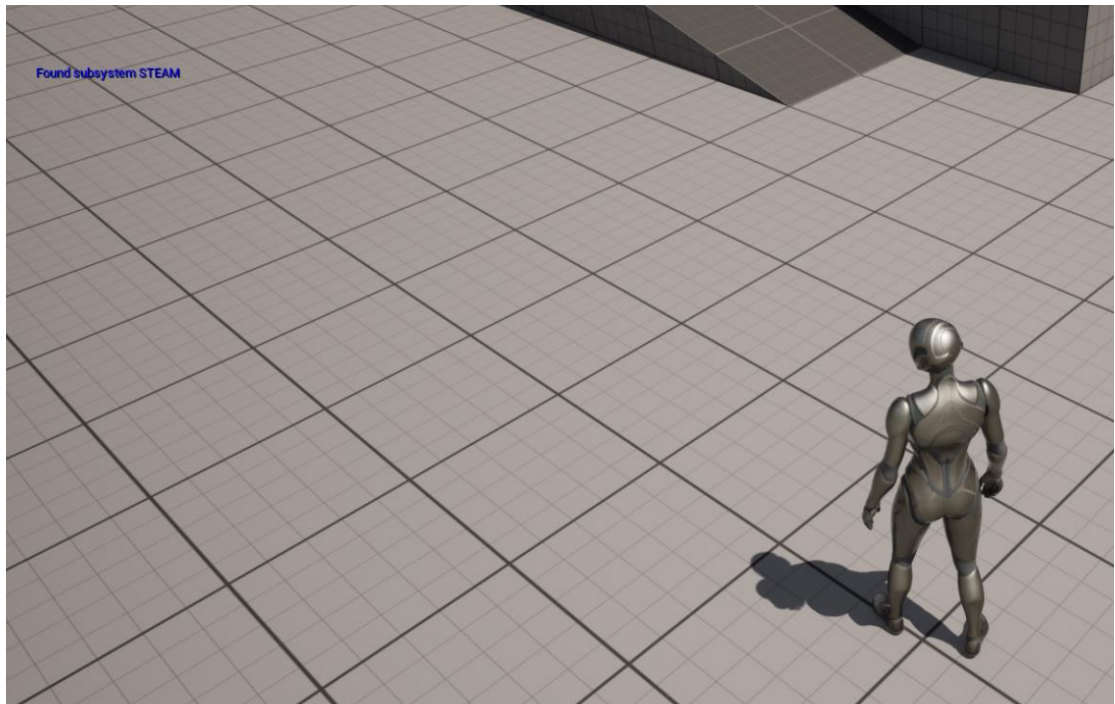
在.cpp 文件中的构造函数末尾添加代码：

获取在线子系统，并且显示出在线服务的名称。

```
IOnlineSubsystem* OnlineSubsystem = IOnlineSubsystem::Get();
if (OnlineSubsystem)
{
    OnlineSessionInterface = OnlineSubsystem->GetSessionInterface();
    if (GEngine)
    {
        GEngine->AddOnScreenDebugMessage(
            -1,
            15. f,
            FColor::Blue,
            FString::Printf(TEXT("Found subsystem %s"),
                *OnlineSubsystem->GetSubsystemName().ToString())
        );
    }
}
```

`IOnlineSubsystem` 类的说明可以参考文档

<https://docs.unrealengine.com/5.1/en-US/API/Plugins/OnlineSubsystem/IOnlineSubsystem/>



完成代码后直接在引擎里运行游戏（play in editor 也就是 PIE）不会链接上在线服务（无论是使用 Standalone, Listen 或者 Delicated Server），完成打包后运行会显示成功连接（因为这个消息是在角色的构造函数中打印的）。

PS：编译阶段报 2 进制代码错误可以按之前的方式处理

	.vs	2023/4/12 18:49	文件夹	
<input checked="" type="checkbox"/>	Binaries	2023/4/12 18:49	文件夹	
	Config	2023/4/12 19:14	文件夹	
	Content	2023/4/12 18:54	文件夹	
	DerivedDataCache	2023/4/12 18:50	文件夹	
<input checked="" type="checkbox"/>	Intermediate	2023/4/12 18:54	文件夹	
<input checked="" type="checkbox"/>	Saved	2023/4/12 19:05	文件夹	
	Source	2023/4/12 18:49	文件夹	
	.vsconfig	2023/4/12 18:49	VSCONFIG 文件	1 KB
	MenuSystem.sln	2023/4/12 18:49	Visual Studio Soluti...	5 KB
	MenuSystem.uproject	2023/4/12 18:53	Unreal Engine Proj...	1 KB

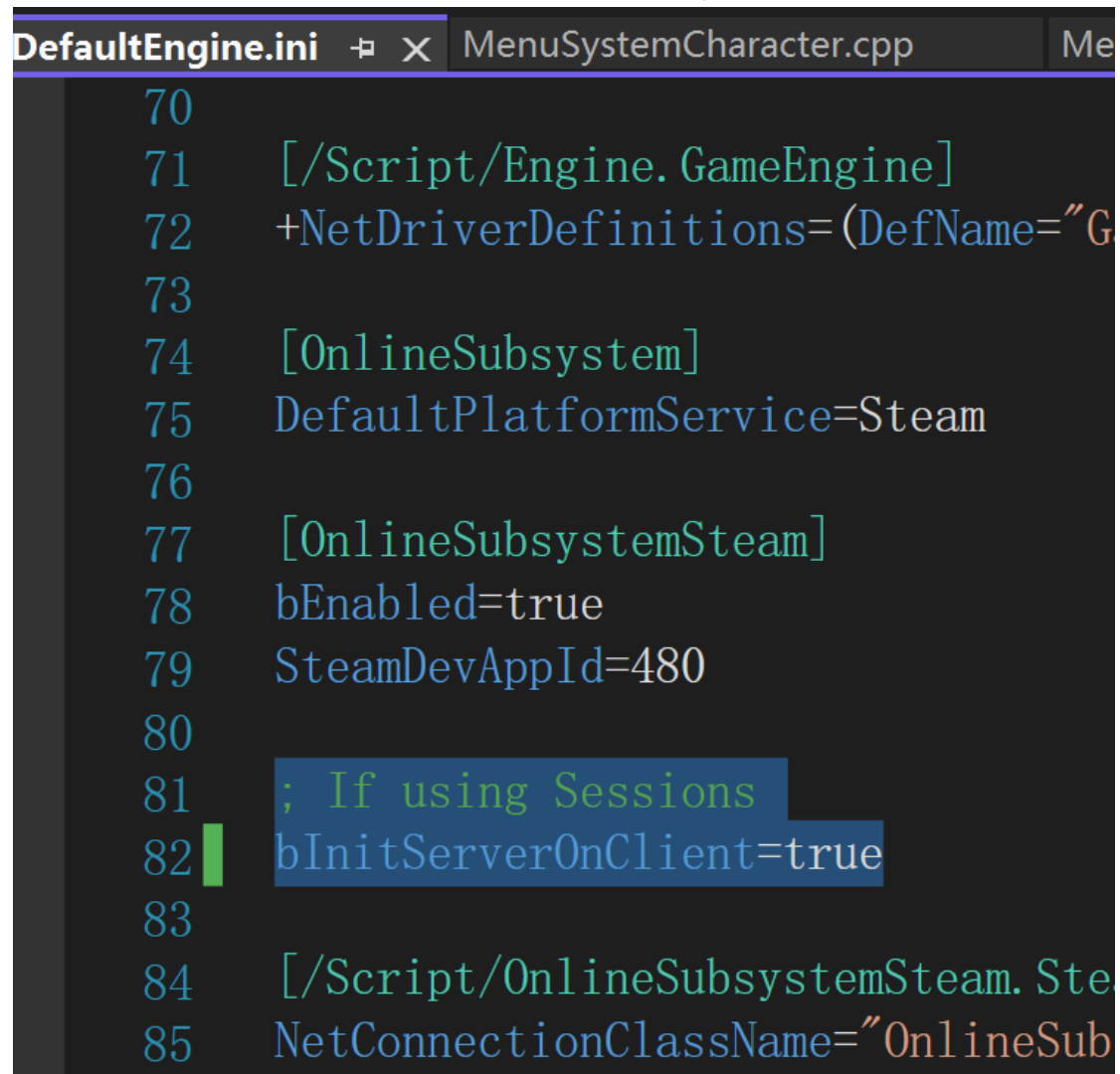
删除这仨文件夹

PS：我自己使用 VS 运行 UE 的时候，PIE 运行触发了中断，会导致 UE 编辑器卡住，直接在 VS 里跳过中断就可以继续运行。



## 010 创建一个会话

要创建会话在 UE5.1.1 中首先需要去到.ini 配置文件，取消掉对 session 的注释，由于教程视频是 5.0.0，这一步在视频的结尾才出现，我以为是自己哪个步骤写错了，花了一个多小时才想到是这个问题，然后解决了之后才看到视频结尾有 QA 环节，瞬间无语（笑哭）。



```
70
71  [/Script/Engine.GameEngine]
72  +NetDriverDefinitions=(DefName="G
73
74  [OnlineSubsystem]
75  DefaultPlatformService=Steam
76
77  [OnlineSubsystemSteam]
78  bEnabled=true
79  SteamDevAppId=480
80
81  ; If using Sessions
82  bInitServerOnClient=true
83
84  [/Script/OnlineSubsystemSteam.Ste
85  NetConnectionClassName="OnlineSub
```

在创建会话之前，我们要知道会话是怎么工作的。

Delegate: 一个 delegate 对象储存了一系列函数的引用，它可以向这些绑定了的函数（通常称为回调(callback)函数）进行广播，每个收到广播的函数会执行。

会话接口（Session Interface）调用 CreateSession() 会向服务器（Steam）发送请求，服务器会调用自己的 SessionCreated，并向会话接口返回结果。会话接口会创建一个 Delegate List 来处理这些事情。

在角色.h 中添加以下代码：

```
protected:
    UFUNCTION(BlueprintCallable)
```

```

void CreateGameSession();

void OnCreateSessionComplete(FName SessionName, bool bWasSuccessful);
private:
    FOnCreateSessionCompleteDelegate CreateSessionCompleteDelegate;

```

其中, CreateGameSession() 是支持蓝图调用的创建会话的方法。OnCreateSessionComplete 是我们的 Delegate 对象 CreateSessionCompleteDelegate 将要绑定的回调函数。

在角色.cpp 文件中, 由于 Delegate 是一个未初始化的对象, 我们需要现在类的构造函数中使用列表初始化:

```

AMenuSystemCharacter::AMenuSystemCharacter() :
    CreateSessionCompleteDelegate(FOnCreateSessionCompleteDelegate::CreateUObject(this,
&ThisClass::OnCreateSessionComplete))

```

初始化过程中绑定了回调函数。

之后我们完成 CreateGameSession 函数和 OnCreateSessionComplete 函数:

```

void AMenuSystemCharacter::CreateGameSession()
{
    // called when pressing the 1 key
    if (!OnlineSessionInterface.IsValid())
    {
        return;
    }

    auto ExistingSession = OnlineSessionInterface->GetNamedSession(NAME_GameSession);
    if (ExistingSession != nullptr)
    {
        OnlineSessionInterface->DestroySession(NAME_GameSession);
    }
    // 为创建的会话添加delegate
    OnlineSessionInterface->AddOnCreateSessionCompleteDelegate_Handle(CreateSessionCompleteDelegate);
    // 设置会话的config
    TSharedPtr<FOnlineSessionSettings> SessionSettings = MakeShareable(new FOnlineSessionSettings());
    SessionSettings->bIsLANMatch = false;
    SessionSettings->NumPublicConnections = 4;
    SessionSettings->bAllowJoinInProgress = true;
    SessionSettings->bAllowJoinViaPresence = true;
    SessionSettings->bShouldAdvertise = true;
    SessionSettings->bUsesPresence = true;
    const ULocalPlayer* LocalPlayer = GetWorld()->GetFirstLocalPlayerFromController();
    OnlineSessionInterface->CreateSession(*LocalPlayer->GetPreferredUniqueNetId(), NAME_

```

```

GameSession,*SessionSettings);
}

void AMenuSystemCharacter::OnCreateSessionComplete(FName SessionName, bool
bWasSuccessful)
{
    if (bWasSuccessful)
    {
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Blue,
                FString::Printf(TEXT("Created session: %s , Num of session
is: %d"),*SessionName.ToString(), OnlineSessionInterface->GetNumSessions())
                );
        }
    }
    else
    {
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Red,
                FString(TEXT("Failed to create session!"))
                );
        }
    }
}

```

最后，我们在 UE 编辑器中连接角色蓝图



之后打包游戏，启动后按 1，就可以看到会话建立成功。

## 011 设置加入游戏会话

这一部分内容和上一部分很相似，在.h 文件中创建一个 Find 的 delegate 对象，以及一个 FindSession 的蓝图可调用函数，再给 delegate 创建一个回调函数，最后由于 Find 的结果需要在回调函数里使用，再添加一个智能指针。

.h 中添加的代码为

```
protected:
    UFUNCTION(BlueprintCallable)
    void JoinGameSession();

    void OnFindSessionsComplete(bool bWasSuccessful);
private:

    FOnFindSessionsCompleteDelegate FindSessionsCompleteDelegate;
    TSharedPtr<FOnlineSessionSearch> SessionSearch;
```

.cpp 中同样在初始化列表里初始化 Delegate 对象：

```
AMenuSystemCharacter::AMenuSystemCharacter():
    CreateSessionCompleteDelegate(FOnCreateSessionCompleteDelegate::CreateUObject(this,
    &ThisClass::OnCreateSessionComplete)),
    FindSessionsCompleteDelegate(FOnFindSessionsCompleteDelegate::CreateUObject(this, &T
    hisClass::OnFindSessionsComplete))
```

之后完成 Find 函数和回调函数即可：

```
void AMenuSystemCharacter::JoinGameSession()
{
    // Find Game sessions
    if (!OnlineSessionInterface.IsValid())
    {
        return;
    }

    SessionSearch = MakeShareable(new FOnlineSessionSearch());
    SessionSearch->MaxSearchResults = 10000;
    SessionSearch->bIsLanQuery = false;
    SessionSearch->QuerySettings.Set(SEARCH_PRESENCE, true, EOnlineComparisonOp::Equals);

    OnlineSessionInterface->AddOnFindSessionsCompleteDelegate_Handle(FindSessionsComple
    teDelegate);
    const ULocalPlayer* LocalPlayer = GetWorld()->GetFirstLocalPlayerFromController();
    OnlineSessionInterface->FindSessions(*LocalPlayer->GetPreferredUniqueNetId(), Sessio
    nSearch.ToSharedRef());
```

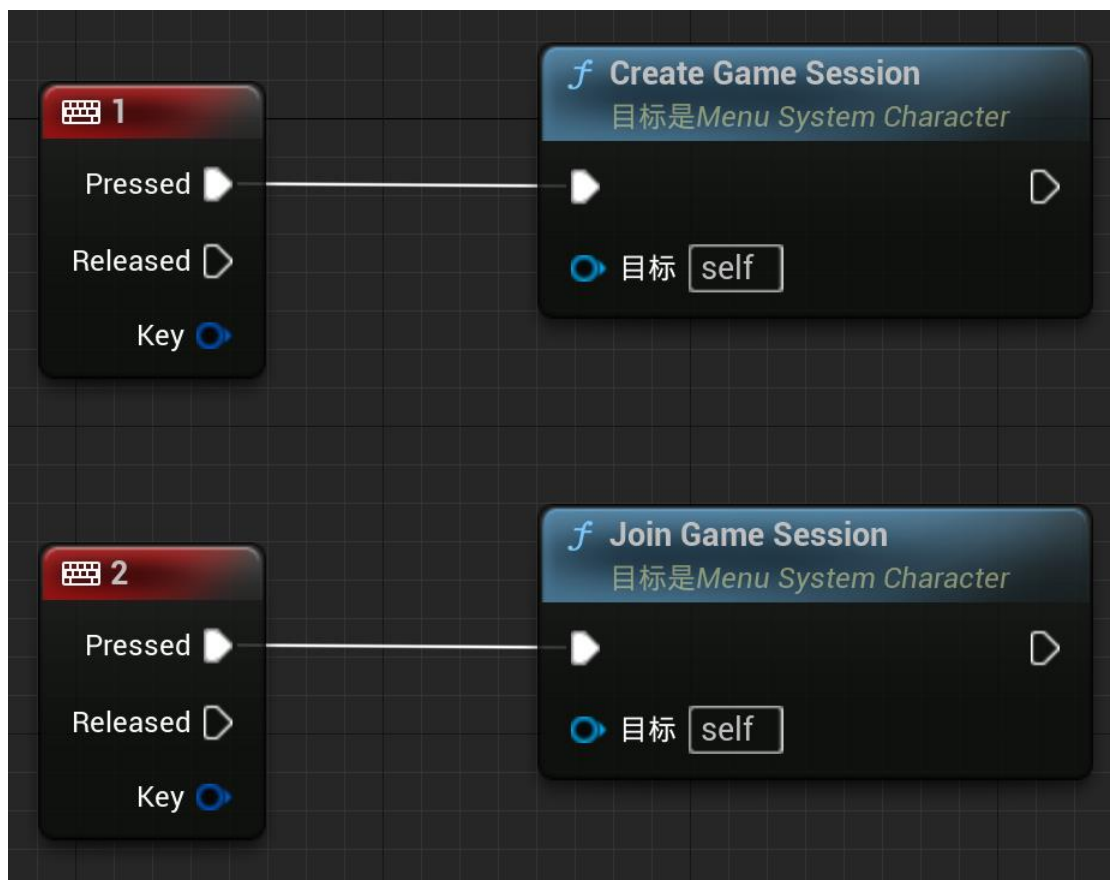
```

}

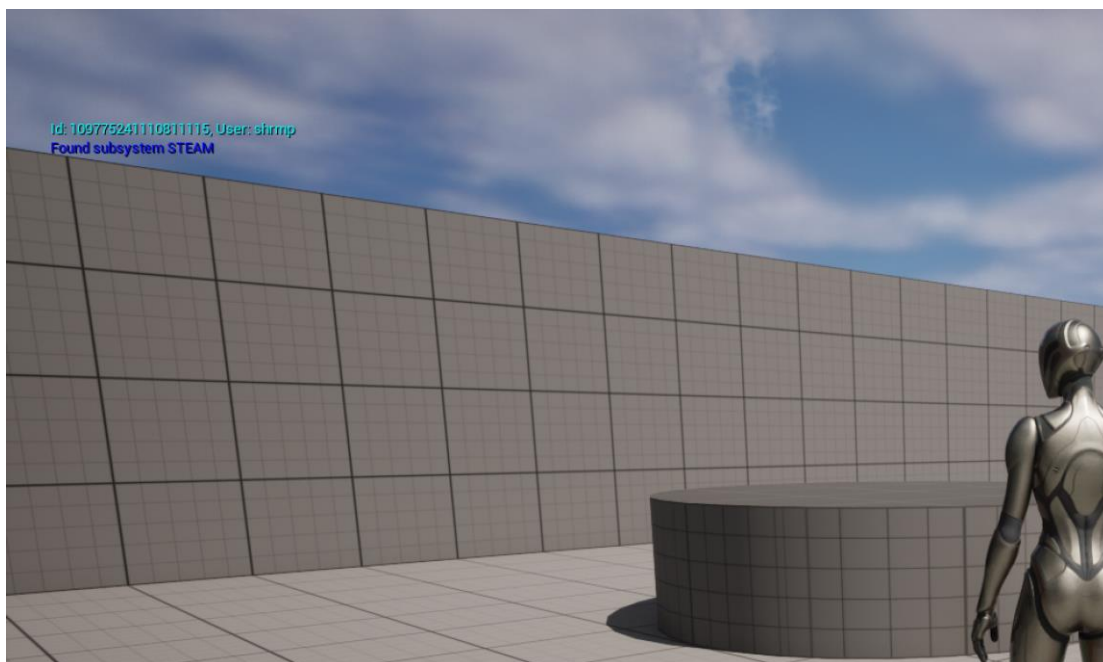
void AMenuSystemCharacter::OnFindSessionsComplete(bool bWasSuccessful)
{
    for (auto Result : SessionSearch->SearchResults) {
        FString Id = Result.GetSessionIdStr();
        FString User = Result.Session.OwningUserName;
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Cyan,
                FString::Printf(TEXT("Id: %s, User: %s"), *Id, *User)
            );
        }
    }
}
}

```

最后在蓝图里调用 Find 函数：



完成后打包游戏即可。



验证的时候需要两台设备分别登录两个 steam 账号，然后打开游戏，第一个客户端按 1 键创建会话，之后第二个客户端按 2 键即可查找到客户端 1 创建的会话，并通过 debug 打印出来。

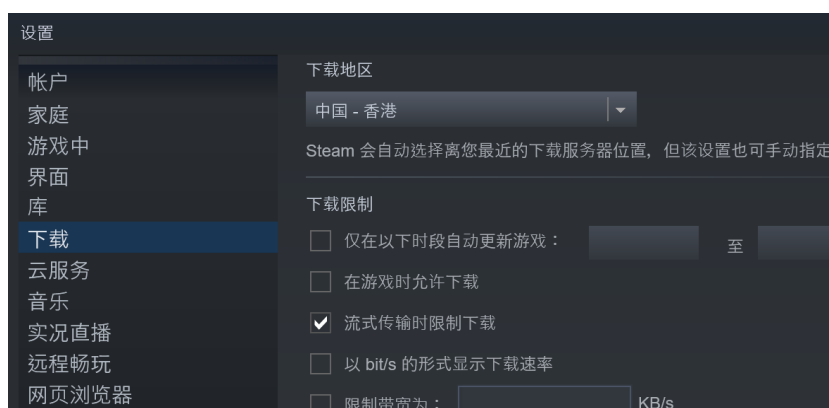
## Q1

steam 中 480 是开发者使用的作品 id，那么在大家都是 480 的情况下，是怎么找到另一个相同代码的客户端的？

猜测是 480 这个代码之下有一个哈希表？然后整个工程文件打包的时候会生成哈希值，来进行匹配。

经过 013 之后发现应该就是单纯的全部都能搜索到。

## 012 steam 区域



当你在 steam 进行测试的时候, 你需要保证你的两个 steam 账号的下载地区是同一个地区。

## 013 加入会话



这一节又弄了一晚上，坑还是很多的。

首先，教程里没有提到的，需要在【项目设置】的【打包】里添加 Lobby 地图和初始地图，否则按 1 之后不能跳转到 Lobby 中



其次在最后加入之前要确保 012 中所提到的两个账号区域一致，不然无法加入。  
最后需要参考 ISSUE2 中的设置，否则可能无法加入。

整体上来说和之前 010,011 差不多，创建一个 Join 的 Delegate 对象，绑定回调函数。

在.h 中：

```
private:
FOnJoinSessionCompleteDelegate JoinSessionCompleteDelegate;
protected:
void OnJoinSessionComplete(FName SessionName, EOnJoinSessionCompleteResult::Type Result);
```

在.cpp 中：

首先是构造函数：

```
AMenuSystemCharacter::AMenuSystemCharacter() :
    CreateSessionCompleteDelegate(FOnCreateSessionCompleteDelegate::CreateUObject(this,
&ThisClass::OnCreateSessionComplete)),
    FindSessionsCompleteDelegate(FOnFindSessionsCompleteDelegate::CreateUObject(this, &ThisClass::OnFindSessionsComplete)),
```

```
JoinSessionCompleteDelegate(FOnJoinSessionCompleteDelegate::CreateUObject(this, &ThisClass::OnJoinSessionComplete))
```

在 createsession 函数中需要增加设置，用来后面进行加入时的验证匹配：

```
SessionSettings->Set(FName("MatchType"), FString("FreeForAll"),
EOnlineDataAdvertisementType::ViaOnlineServiceAndPing);
```

之前的 2 个回调函数中均有改变，在 Create 的回调函数中，需要添加创建之后转移到大厅的代码，并且设置 MatchType 以便让其他客户端能够顺利的加入游戏中。在 Find 中，会对循环里的 Result 进行匹配判断以加入正确的游戏，在匹配之后会调用加入会话，加入完成后会激活 Join 的回调函数，加入游戏。

```
void AMenuSystemCharacter::OnCreateSessionComplete(FName SessionName, bool
bWasSuccessful)
{
    if (bWasSuccessful)
    {
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Blue,
                FString::Printf(TEXT("Created session: %s , Num of session
is: %d"), *SessionName.ToString(), OnlineSessionInterface->GetNumSessions())
            );
        }

        UWorld* World = GetWorld();
        if (World)
        {
            bool IsTravelSuccessful =
World->ServerTravel(FString("/Game/ThirdPerson/Maps/Lobby?listen"));
            int TravelResult = IsTravelSuccessful ? 1 : 0;
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Purple,
                FString::Printf(TEXT("TravelResult is: %d"), TravelResult)
            );
        }
    }
    else
    {
```



```

        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Red,
                FString(TEXT("Failed to create session!"))
            );
        }
    }
}

```

以及

```

void AMenuSystemCharacter::OnFindSessionsComplete(bool bWasSuccessful)
{
    if (!OnlineSessionInterface.IsValid())
    {
        return;
    }

    for (auto Result : SessionSearch->SearchResults) {
        FString Id = Result.GetSessionIdStr();
        FString User = Result.Session.OwningUserName;

        FString MatchType;
        Result.Session.SessionSettings.Get(FName("MatchType"), MatchType);

        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Cyan,
                FString::Printf(TEXT("Id: %s, User: %s, MatchType: %s"), *Id,
                *User, *MatchType)
            );
        }

        if (MatchType == FString("FreeForAll"))
        {
            if (GEngine)
            {
                GEngine->AddOnScreenDebugMessage(
                    -1,
                    15.f,

```

```

        FColor::Cyan,
        FString::Printf(TEXT("Joining Match Type: %s"), *MatchType)
    );
}

OnlineSessionInterface->AddOnJoinSessionCompleteDelegate_Handle(JoinSessionComplete
Delegate);

    const ULocalPlayer* LocalPlayer =
    GetWorld()->GetFirstLocalPlayerFromController();

    OnlineSessionInterface->JoinSession(*LocalPlayer->GetPreferredUniqueNetId(), NAME_Ga
meSession, Result);
}
}
}

void AMenuSystemCharacter::OnJoinSessionComplete(FName SessionName,
EOnJoinSessionCompleteResult::Type Result)
{
    if (!OnlineSessionInterface.IsValid())
    {
        return;
    }

    FString Address;
    if (OnlineSessionInterface->GetResolvedConnectString(NAME_GameSession, Address))
    {
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Yellow,
                FString::Printf(TEXT("Connect string: %s"), *Address)
            );
        }

        APlayerController* PlayerController =
        GetGameInstance()->GetFirstLocalPlayerController();
        if (PlayerController)
        {
            PlayerController->ClientTravel(Address, ETravelType::TRAVEL_Absolute);
        }
    }
}
}

```

## Q2

一些默认的函数执行顺序是什么样的？

## 014 创建一个插件

UE 插件由一个或多个模块组成，每个模块有如下特点：

1. 一段独立的 C++ 代码
2. 包含一个 build 文件 (.Build.cs)
3. 只包含代码，不包含资产
4. 封装（每个模块有一个明显的功能）
5. 结构比较好
6. 我们的项目本身也是一个模块，一个模块能被单独编译

插件可以直接在 UE 编辑器的插件管理里很方便的启用或禁用。更重要的原因是可以重复使用。

当我们通过插件管理器启用插件时，会同时改变.uproject 工程文件。



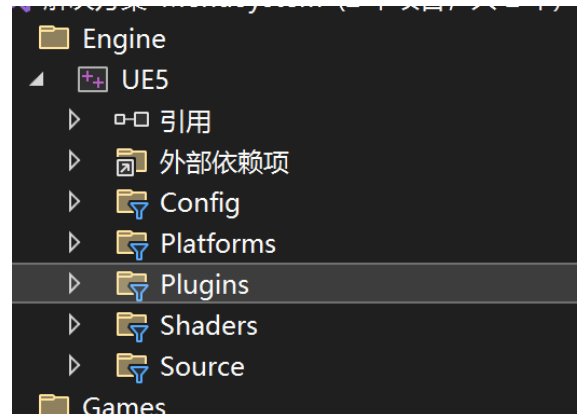
```
MenuSystem.uproject  x DefaultGame.ini  MenuSystemCharacter.cp
架构: <未选择架构>
1  {
2      "FileVersion": 3,
3      "EngineAssociation": "5.1",
4      "Category": "",
5      "Description": "",
6      "Modules": [
7          {
8              "Name": "MenuSystem",
9              "Type": "Runtime",
10             "LoadingPhase": "Default"
11         }
12     ],
13     "Plugins": [
14         {
15             "Name": "ModelingToolsEditorMode",
16             "Enabled": true,
17             "TargetAllowList": [
18                 "Editor"
19             ]
20         },
21         {
22             "Name": "OnlineSubsystemSteam",
23             "Enabled": true
24         }
25     ]
26 }
```

可以看到之前启用了的 Steam 的子系统插件。

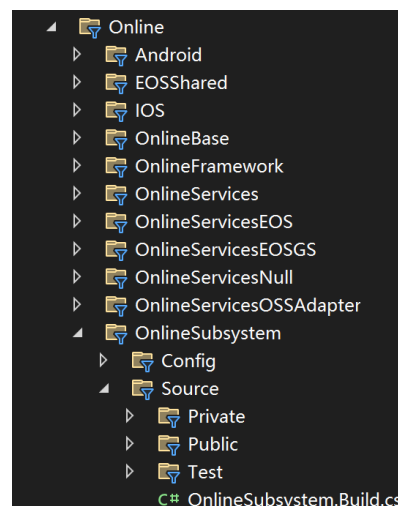
插件也可以依赖别的插件，这些插件会成为.uplugin 文件。但是 UE 的插件是有层级关系的（引擎层，独立层，项目专属层），除了可以依赖同级别的插件外，高级的插件可以依赖低级的插件，但是低级的插件不能依赖高级的插件。

例如游戏插件可以依赖与引擎插件，但是引擎插件不能依赖游戏插件。

引擎层的插件在 VS 资源管理器中可以看到



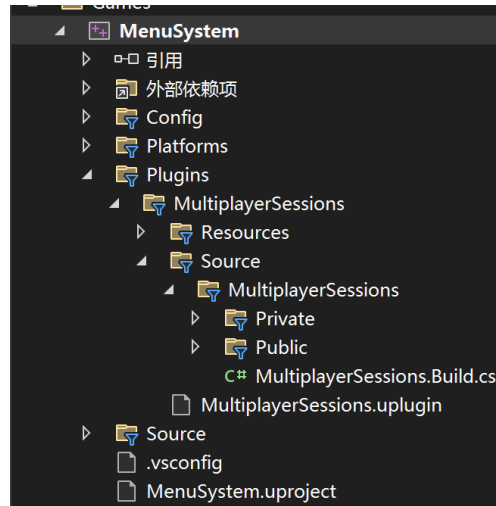
这次创建的插件就依赖于引擎插件中的在线子系统插件。



打开插件管理器之后点左上角的加号，创建一个空白插件。



之后就能看到在项目中多了一个插件的文件夹



接下来就要给插件添加依赖。

打开.uplugin 文件

在 modules[···]之后添加代码：

```
,
    "Plugins": [
        {
            "Name": "OnlineSubsystem",
            "Enabled": true
        },
        {
            "Name": "OnlineSubsystemSteam",
            "Enabled": true
        }
    ]
}
```

接下来在 MultiplayerSessions.Build.cs 文件的

PublicDependencyModuleNames.AddRange中 添加两项依赖

```
    "OnlineSubsystem",
    "OnlineSubsystemSteam",
```

然后编译，完成之后能看到项目文件夹中出现了插件文件夹。

yerCourse > MenuSystem >		
名称		修改日期
.vs		2023/4/12 18:49
Binaries		2023/4/13 0:13
Build		2023/4/13 23:42
Config		2023/4/13 23:23
Content		2023/4/14 0:56
DerivedDataCache		2023/4/12 18:50
Intermediate		2023/4/14 0:56
Platforms		2023/4/13 22:43
Plugins		2023/4/14 0:56
Saved	创建日期: 2023/4/14 0:56	2023/4/14 0:56
Source	大小: 49.3 MB	2023/4/12 18:49
.vsconfig	文件夹: MultiplayerSessions	2023/4/12 18:49
MenuSystem.sln		2023/4/12 18:49
MenuSystem.userproj		2023/4/12 18:53

## 015 创建我们自己的子系统

会话系统的父类应该选择什么？

一个不错的选择是 Game Instance 类，他有如下特性：

1. 在游戏创建是被创建
2. 直到游戏结束才会销毁
3. 在关卡跳转之间存在（单例？）

因此，游戏实例可以维持我们在关卡转换之间的多人会话需求，然而游戏实例是为了一些基础的功能创建的,因为多人会话功能并不是游戏实例创建的目的,我们希望有一个类有 game instance 类类似的性质，教程中使用的是 game instance subsystem。

<https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Subsystems/>

1. After `UGameInstance` is created, an instance called `UMyGamesSubsystem` is also created.
2. When `UGameInstance` initializes, `Initialize()` will be called on the subsystem.
3. When `UGameInstance` is shut down, `Deinitialize()` will be called on the subsystem.
4. At this point, the reference to the subsystem is dropped, and the subsystem is garbage-collected if there are no more references to it.

## Reasons to Use Subsystems

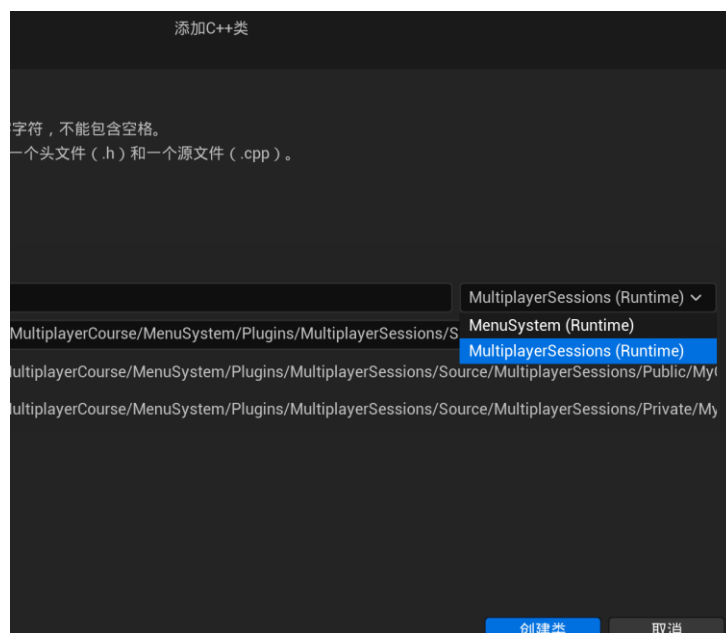
There are several reasons to use programming subsystems, including the following:

- Subsystems save programming time.

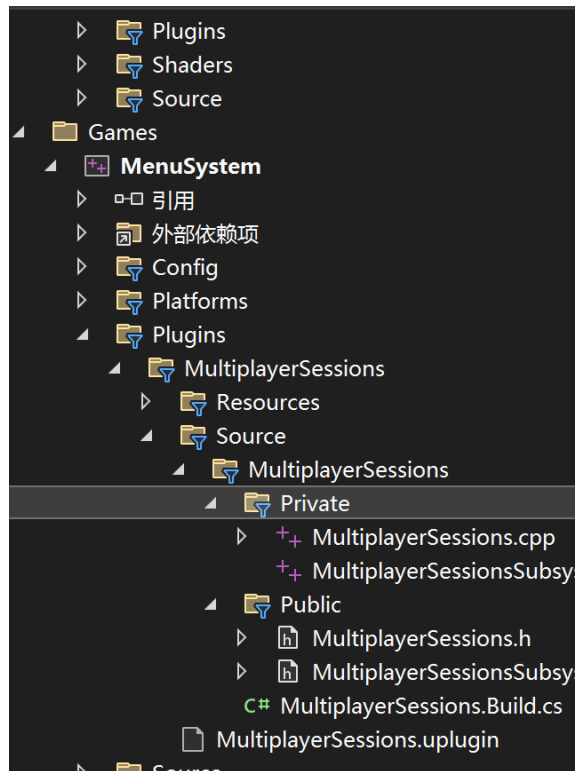
- Subsystems help you avoid overriding engine classes.
- Subsystems help you avoid adding more API on an already busy class.
- Subsystems enable access to Blueprints through user friendly typed nodes.
- Subsystems enable access to Python scripts for editor scripting, or for writing test code.
- Subsystems provide modularity and consistency in the codebase.

Subsystems are particularly useful when creating plugins. You do not need to have instructions about the code needed to make the plugin work. The user can just add the plugin to the game, and you know exactly when the plugin will be instanced and initialized. As a result, you can focus on how to use the API and the functionality provided in UE4.

打开 UE，创建一个新的类，继承自 `game instance subsystem`，并且选择为我们新创建的插件模块添加这个类。



点击创建然后我们能看到，VS 资源管理器中我们的会话插件里出现了两个文件夹，里面有我们创建的类。



如果有红色波浪线的错误，删除三个文件夹：

	.vs	2023/4/12 18:49	文件夹	
<input checked="" type="checkbox"/>	Binaries	2023/4/13 0:13	文件夹	
	Build	2023/4/13 23:42	文件夹	
	Config	2023/4/13 23:23	文件夹	
	Content	2023/4/14 22:51	文件夹	
	DerivedDataCache	2023/4/12 18:50	文件夹	
<input checked="" type="checkbox"/>	Intermediate	2023/4/14 22:51	文件夹	
	Platforms	2023/4/13 22:43	文件夹	
	Plugins	2023/4/14 0:56	文件夹	
<input checked="" type="checkbox"/>	Saved	2023/4/14 22:51	文件夹	
	Source	2023/4/12 18:49	文件夹	
	.vsconfig	2023/4/12 18:49	VSCONFIG 文件	1 k
	MenuSystem.sln	2023/4/12 18:49	Visual Studio Soluti...	5 k
	MenuSystem.uproject	2023/4/12 18:53	Unreal Engine Proj...	1 k

以及删除插件中的两个文件夹

<input checked="" type="checkbox"/>	Binaries	2023/4/14 0:56	文件夹	
	Content	2023/4/14 22:51	文件夹	
<input checked="" type="checkbox"/>	Intermediate	2023/4/14 0:56	文件夹	
	Resources	2023/4/14 0:56	文件夹	
	Source	2023/4/14 0:56	文件夹	
	MultiplayerSessions.uplugin	2023/4/14 1:03	UPLUGIN 文件	1 K

之后右键项目工程文件，生成 vs 项目，再左键双击生成二进制文件。

在.h 文件中我们加载 sessioninterface 的头文件，然后添加代码：

public:



```

    UMultiplayerSessionsSubsystem();
protected:

private:
    IOnlineSessionPtr SessionInterface;
在.cpp 文件中我们在构造函数中初始化 sessioninterface:
#include "OnlineSubsystem.h"
UMultiplayerSessionsSubsystem::UMultiplayerSessionsSubsystem()
{
    IOnlineSubsystem* Subsystem = IOnlineSubsystem::Get();
    if (Subsystem)
    {
        SessionInterface = Subsystem->GetSessionInterface();
    }
}

```

后续 Delegate 对象的添加将放在接下来的 016 中。

## 016 会话界面委托

相比于之前的 delegate 的添加，这一章节多出了 delegate 的 handle 成员变量，用于移除不需要的 delegate，handle 是前调用过的 AddOnCreateSessionCompleteDelegate\_Handle() 类似的函数返回，为了实现完整的会话过程，使用了 5 个 delegate，直接上代码：

.h 文件：

```

UCLASS()
class MULTIPLAYERSESSIONS_API UMultiplayerSessionsSubsystem : public
UGameInstanceSubsystem
{
    GENERATED_BODY()
public:
    UMultiplayerSessionsSubsystem();

    /**
     * To handle session functionality. The Menu class will call these
     */
    void CreateSession(int32 NumPublicConnections, FString MatchType);
    void FindSessions(int32 MaxSearchResults);
    void JoinSession(const FOnlineSessionSearchResult& SessionResult);
    void DestorySession();
    void StartSession();
}

```

```

protected:
    /**
     * Internal callbacks for the delegates we'll add to the Online session interface
     delegate list.
     * This don't need to be called outside this class.
     */
    void OnCreateSessionComplete(FName SessionName, bool bWasSuccessful);
    void OnFindSessionsComplete(bool bWasSuccessful);
    void OnJoinSessionComplete(FName SessionName, EOnJoinSessionCompleteResult::Type
Result);
    void OnDestroySessionComplete(FName SessionName, bool bWasSuccessful);
    void OnStartSessionComplete(FName SessionName, bool bWasSuccessful);
private:
    IOnlineSessionPtr SessionInterface;

    /**
     * To add to the Online Session Interface delegate list.
     * We'll bind our MultiplayerSessionsSubsystem internal callbacks to these.
     */
    FOnCreateSessionCompleteDelegate CreateSessionCompleteDelegate;
    FDelegateHandle CreateSessionCompleteDelegateHandle;
    FOnFindSessionsCompleteDelegate FindSessionsCompleteDelegate;
    FDelegateHandle FindSessionsCompleteDelegateHandle;
    FOnJoinSessionCompleteDelegate JoinSessionCompleteDelegate;
    FDelegateHandle JoinSessionCompleteDelegateHandle;
    FOnDestroySessionCompleteDelegate DestroySessionCompleteDelegate;
    FDelegateHandle DestroySessionCompleteDelegateHandle;
    FOnStartSessionCompleteDelegate StartSessionCompleteDelegate;
    FDelegateHandle StartSessionCompleteDelegateHandle;
};

```

.cpp 文件中需要在构造函数中初始化 5 个 delegate。

```

UMultiplayerSessionsSubsystem::UMultiplayerSessionsSubsystem():
    CreateSessionCompleteDelegate(FOnCreateSessionCompleteDelegate::CreateUObject(this,
&ThisClass::OnCreateSessionComplete)),
    FindSessionsCompleteDelegate(FOnFindSessionsCompleteDelegate::CreateUObject(this, &T
hisClass::OnFindSessionsComplete)),
    JoinSessionCompleteDelegate(FOnJoinSessionCompleteDelegate::CreateUObject(this, &Thi
sClass::OnJoinSessionComplete)),
    DestroySessionCompleteDelegate(FOnDestroySessionCompleteDelegate::CreateUObject(thi
s, &ThisClass::OnDestroySessionComplete)),
    StartSessionCompleteDelegate(FOnStartSessionCompleteDelegate::CreateUObject(this, &T
hisClass::OnStartSessionComplete))
{
    IOnlineSubsystem* Subsystem = IOnlineSubsystem::Get();

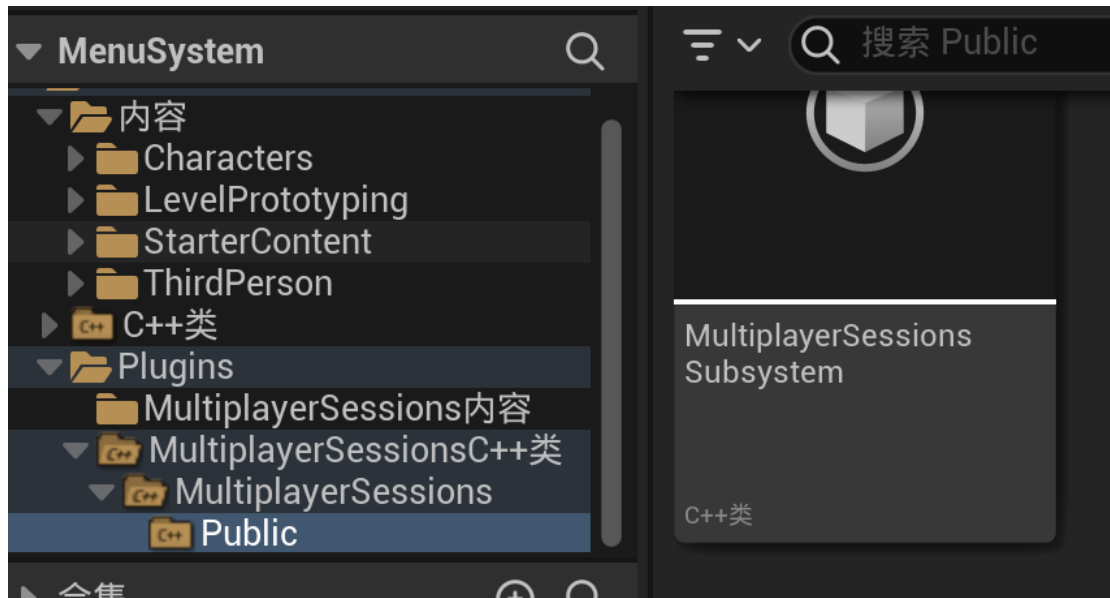
```

```
if (Subsystem)
{
    SessionInterface = Subsystem->GetSessionInterface();
}
}
```

## 017 菜单类

我们会创建一个菜单并把它添加到我们的插件中，这个菜单可以方便我们加入会话，这样以后我们就不需要重复的创建他，此外我们还希望能够设置我们插件的参数，比如最大加入人数。

打开 UE 编辑器，在插件的 public 文件夹中添加一个继承与 UserWidget 的类。



### Warning 1

我添加类的时候出错了，导致我整个工程文件用不了了，一不小心吧 source 文件删了，然后还直接清理了回收站，写了几天的代码直接没了。大家一定要好好备份。没办法只能去 gitHub 上拷一份相同进度的代码下来继续了。教程里使用的是 5.0 的版本，而我用的是 5.1 的版本，所以不能兼容，我也不想换成 5.0 的版本的 UE，幸好这只是个开头，而且我也记录了整个学习得过程，就当是复习好了，重新写一遍。

重新写了一遍，这次只花了半个小时就写到这里，但是创建 widget 类还是发生了相同的错误。动态库的连接错误，直接用空项目添加 Menu 是没有问题的，有点无语，猜测是不是少了依赖项。

K

## error LINK2019 when create c++ class user widget in MultiplayerSession plugin

25

Kuroneko · Lecture 17 · 11 months ago

Hello, I got a error LINK2019 when create user widget:

```
Module.MultiplayerSessions.gen.cpp.obj : error LNK2019: unresolved external symbol "__declspec(dllimport) public:
__cdecl UUserWidget::UUserWidget(class FVTableHelper &)" (__imp_??
0UUserWidget@@QEAA@AEAVFVTableHelper@@@Z), referenced in function "public: __cdecl UMenu::UMenu(class
FVTableHelper &)" (??0UMenu@@QEAA@AEAVFVTableHelper@@@Z)
```

```
Module.MultiplayerSessions.gen.cpp.obj : error LNK2019: unresolved external symbol "__declspec(dllimport) class
UClass * __cdecl Z_Construct_UClass_UUserWidget(void)" (__imp_?
Z_Construct_UClass_UUserWidget@@YAPEAVUClass@@XZ), referenced in function "void __cdecl `dynamic
initializer for 'public: static class UObject * (__cdecl*const * const
Z_Construct_UClass_UMenu_Statics::DependentSingletons)(void)'(void)" (??_E?
DependentSingletons@Z_Construct_UClass_UMenu_Statics@@2QBQ6APEAVUObject@@XZB@@YAXXZ)
```

```
E:\ue4\MultiplayerCourse\MenuSystem\Plugins\MultiplayerSessions\Binaries\Win64\UnrealEditor-
MultiplayerSessions.patch_0.exe : fatal error LNK1120: 104 unresolved external command
```

Failed to link patch (0.000s) (Exit code: 0x460)

Does anyone know what happend?

12 replies

Follow replies

K

Kuroneko

★ Answer

11 months ago

98

I solved this by adding "UMG" module to MultiplayerSessions.build.cs:

```
PublicDependencyModuleNames.AddRange(
    new string[]
    {
        "Core",
        "OnlineSubsystem",
        "OnlineSubsystemSteam",
        "UMG",
        // ... add other public dependencies that you statically link with here ...
    }
);
```

WB

Willian

44 months ago

2

折腾了一晚上，都没弄好，最后在教程下面看到了别人也有一样的问题，要在.cs 文件上加上依赖。2点半了，明天起来再学。

.h 中添加如下代码:

```
public:
    UFUNCTION(BlueprintCallable)
    void MenuSetup();
```

.cpp 中添加如下代码，用来设置 UI 控件，添加到视口，获取玩家控制器，并且修改控制器输入模式，显示鼠标等:

```
void UMenu::MenuSetup()
{
    AddToViewport();
    SetVisibility(ESlateVisibility::Visible);
    bIsFocusable = true;

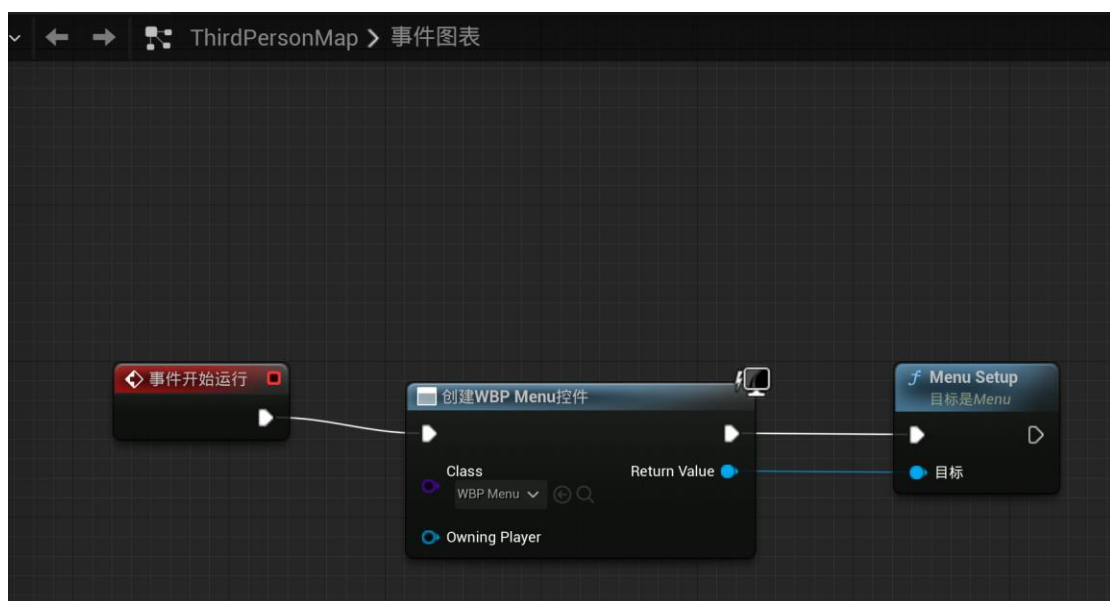
    UWorld* World = GetWorld();
    if (World)
    {
```

```

APlayerController* PlayerController = World->GetFirstPlayerController();
if (PlayerController)
{
    FInputModeUIOnly InputModeData;
    InputModeData.SetWidgetToFocus(TakeWidget());
    InputModeData.SetLockMouseToViewportBehavior(EMouseLockMode::DoNotLock);
    PlayerController->SetInputMode(InputModeData);
    PlayerController->SetShowMouseCursor(true);
}
}
}

```

最后在 UE 编辑器中 MutliplayerSessions content 中添加一个 widget 蓝图类，设置两个按钮分别命名为 JoinBotton 和 HostBotton。并在关卡蓝图中调用。



最后把控件的父类设置为 Menu 类即可。

## 018 访问我们的子系统

在 menu.h 中添加如下代码:

```
protected:
    virtual bool Initialize() override;

private:
    UPROPERTY(meta = (BindWidget))
    class UButton* HostButton;

    UPROPERTY(meta = (BindWidget))
    UButton* JoinButton;

    UFUNCTION()
    void HostButtonClicked();

    UFUNCTION()
    void JoinButtonClicked();

    // The subsystem designed to handle all online session functionality
    class UMultiplayerSessionsSubsystem* MultiplayerSessionsSubsystem;
```

分别是重载基类的 initailize 函数, 创建两个与 UI 两个按钮对应的按钮指针, 宏 UPROPERTY(meta = (BindWidget)) 的作用是让调用了 menu 类作为父类的子类中的同名按钮直接绑定上这个指针。HostButtonClicked() 和 JoinButtonClicked() 分别是按钮按下后的回调函数。最后由于需要连接我们的会话系统, 还需要一个会话系统的指针。

在.cpp 文件中, 首先添加头文件:

```
#include "MultiplayerSessionsSubsystem.h"
#include "Components/Button.h"
```

之后再构造函数尾部获取会话系统:

```
UGameInstance* GameInstance = GetGameInstance();
if (GameInstance)
{
    MultiplayerSessionsSubsystem =
GameInstance->GetSubsystem<UMultiplayerSessionsSubsystem>();
}
```

最后实现回调函数, 并绑定回调函数。

```
bool UMenu::Initialize()
{
    if (!Super::Initialize())
    {
        return false;
    }
}
```

```

        if (HostButton)
        {
            HostButton->OnClicked.AddDynamic(this, &ThisClass::HostButtonClicked);
        }
        if (JoinButton)
        {
            JoinButton->OnClicked.AddDynamic(this, &ThisClass::JoinButtonClicked);
        }

        return true;
    }

void UMenu::HostButtonClicked()
{
    if (GEngine)
    {
        GEngine->AddOnScreenDebugMessage(
            -1,
            15.f,
            FColor::Yellow,
            FString(TEXT("Host Button Clicked")))
    };
}

if (MultiplayerSessionsSubsystem)
{
    MultiplayerSessionsSubsystem->CreateSession(4, FString("FreeForAll"));
}

}

void UMenu::JoinButtonClicked()
{
    if (GEngine)
    {
        GEngine->AddOnScreenDebugMessage(
            -1,
            15.f,
            FColor::Yellow,
            FString(TEXT("Join Button Clicked")))
    };
}

}

```



## Q3

Super 的作用是什么。调用父类被重载了的方法的接口？

## 019 创建会话

### ISSUE 1

本节中使用的

```
void UMenu::OnLevelRemovedFromWorld(ULevel* InLevel, UWorld* InWorld)
```

函数似乎在UE5.1的版本下被移除了，我参考一下讨论，将其换成了

```
virtual void NativeDestruct() override;
```

链接：

<https://forums.unrealengine.com/t/where-is-uuserwidget-onlevelremovedfromworld-in-5-1/692215>

官方教程里用了相同的方法。

## 正文

在上一节中我们完成了 createSession，但是我们希望创建会话之后能够直接进入，所以在回调函数中创建会话之后还需要添加一段转换场景的代码：

```
void UMenu::HostButtonClicked()
{
    if (GEngine)
    {
        GEngine->AddOnScreenDebugMessage(
            -1,
            15.f,
            FColor::Yellow,
            FString(TEXT("Host Button Clicked")))
    };
    if (MultiplayerSessionsSubsystem)
    {
        MultiplayerSessionsSubsystem->CreateSession(NumPublicConnections, MatchType);
        UWorld* World = GetWorld();
        if (World)
        {
            World->ServerTravel("/Game/ThirdPerson/Maps/Lobby?listen");
        }
    }
}
```

```

    }
}
}

```

但是我们发现，这样做进入到 Lobby 之后仍然不能行动，因为我们之前修改了输入模式为 UIOnly，所以我们需要在添加一个函数，用来修改输入模式：

.h 中：

```
private:
```

```
void MenuTearDown();
```

.cpp 中：

```
void UMenu::MenuTearDown()
```

```

{
    RemoveFromParent();
    UWorld* World = GetWorld();
    if (World)
    {
        APlayerController* PlayerController = World->GetFirstPlayerController();
        if (PlayerController)
        {
            FInputModeGameOnly InputModeData;
            PlayerController->SetInputMode(InputModeData);
            PlayerController->SetShowMouseCursor(false);
        }
    }
}

```

那么我们什么时候调用这个函数呢？作者给出的方案是在关卡转换的时候调用该函数，但是由于会碰到上面的 ISSUE 1 所以我们换成了 NativeDestruct() 函数，代码如下：

.h 中：

```
protected:
```

```
virtual void NativeDestruct() override;
```

.cpp 中：

```
void UMenu::NativeDestruct()
```

```

{
    MenuTearDown();
    Super::NativeDestruct();
}

```

此外为了提高代码的灵活性，将创建会话时的参数设置为成员变量，再将 setup 改为参数列表含两个默认参数的结构，并对成员变量初始化。

.h:

```
public:
```

```
UFUNCTION(BlueprintCallable)
```

```

void MenuSetup(int32 NumberOfPublicConnections = 4, FString TypeOfMatch =
FString(TEXT("FreeForAll")));

```

```
private:
```

```

    int32 NumPublicConnections{4};
    FString MatchType{ TEXT("FreeForAll") };
.cpp:
void UMenu::MenuSetup(int32 NumberOfPublicConnections, FString TypeOfMatch)
{
    NumPublicConnections = NumberOfPublicConnections;
    MatchType = TypeOfMatch;
    AddToViewport();
    SetVisibility(ESlateVisibility::Visible);
    bIsFocusable = true;

    UWorld* World = GetWorld();
    if (World)
    {
        APlayerController* PlayerController = World->GetFirstPlayerController();
        if (PlayerController)
        {
            FInputModeUIOnly InputModeData;
            InputModeData.SetWidgetToFocus(TakeWidget());
            InputModeData.SetLockMouseToViewportBehavior(EMouseLockMode::DoNotLock);
            PlayerController->SetInputMode(InputModeData);
            PlayerController->SetShowMouseCursor(true);
        }
    }
    UGameInstance* GameInstance = GetGameInstance();
    if (GameInstance)
    {
        MultiplayerSessionsSubsystem =
GameInstance->GetSubsystem<UMultiplayerSessionsSubsystem>();
    }
}

```

以及:

```

void UMenu::HostButtonClicked()
{
    if (GEngine)
    {
        GEngine->AddOnScreenDebugMessage(
            -1,
            15.f,
            FColor::Yellow,
            FString(TEXT("Host Button Clicked")))
    };
    if (MultiplayerSessionsSubsystem)

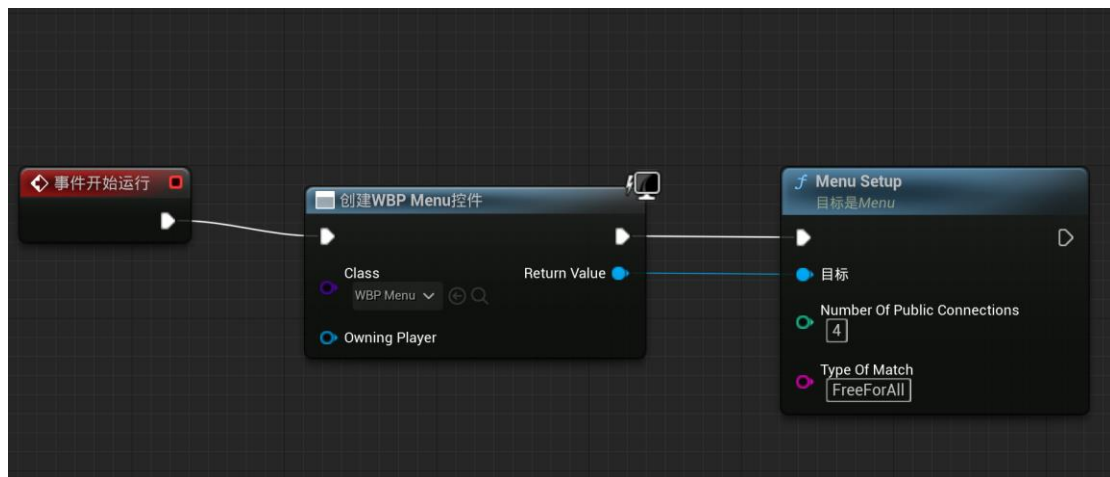
```

```

{
    MultiplayerSessionsSubsystem->CreateSession(NumPublicConnections, MatchType);
    UWorld* World = GetWorld();
    if (World)
    {
        World->ServerTravel("/Game/ThirdPerson/Maps/Lobby?listen");
    }
}
}

```

最后打开 UE 编辑器，就能看到关卡蓝图中的函数调用中多出了两个默认参数，这也方便我们进行开发。



## 020 回调子系统函数

这一章节主要做了三件事：

1. 创建一个自定义的 delegate
2. 绑定一个菜单的回调函数
3. 将转移到大厅的功能放在了菜单回调函数中。

从本节开始鉴于代码量的增加，代码将上传到 github:

<https://github.com/qyz3112746/UE5MutliplayerFPS.git>

### 创建自定义委托 (custom delegate)

我们知道在我们的 session interface 和 session subsystem 之间我们是通过 delegate 类来实现通知服务完成的功能的，但是我们的 menu 类并不能知道 session subsystem 中自己发送的请求 GreateSession 的完成情况，所以我们需要创建一个自定义的委托来绑定一个回调函数实现通知的功能。

在 MultipalyerSessionsSubsystem.h 文件中我们要申明一个 custom delegate 的类：

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FMultiplayerOnCreateSessionComplete, bool, bWasSuccessful);
```

委托类的申明通过宏来执行，`DYNAMIC_MULTICAST` 意味着这是一个动态多播的委托，动态委托只能传递的参数都需要能够被编译成蓝图，`_OneParam` 意味着他的回调函数需要接受一个参数，在这里我们用一个 bool 类型的变量来告知 Menu 类 CreateSession 的结果。

创建申明之后，在 UMultiplayerSessionsSubsystem 类中添加一个委托的成员变量：

```
public:
    /**
     * Our own custom delegates for the Menu class to bind callback to
     */
    FMultiplayerOnCreateSessionComplete MultiplayerOnCreateSessionComplete;
```

之后在 Menu.h 中声明一个回调函数，由于是动态委托，需要使用 UFUNCTION() 宏标记回调函数：

```
/**
 * Callbacks for the custom delegates on the MultiplayerSessionsSubsystem
 */
UFUNCTION()
void OnCreateSession(bool bWasSuccessful);
```

后，在 Menu.cpp 的 Setup 方法中绑定回调函数：

```
if (MultiplayerSessionsSubsystem)
{
    MultiplayerSessionsSubsystem->MultiplayerOnCreateSessionComplete.AddDynamic(this,
    &ThisClass::OnCreateSession);
```

```
}
```

回调函数的具体实现如下:

```
void UMenu::OnCreateSession(bool bWasSuccessful)
{
    if (bWasSuccessful)
    {
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Yellow,
                FString(TEXT("Session created successfully!"))
            );
        }

        UWorld* World = GetWorld();
        if (World)
        {
            World->ServerTravel("/Game/ThirdPerson/Maps/Lobby?listen");
        }
    }
    else
    {
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                -1,
                15.f,
                FColor::Red,
                FString(TEXT("Failed to create Session!"))
            );
        }
    }
}
```

这里为了让会话创建完成后在进行地图转移, 我们把之前的地图转移的实现放到了回调函数中。当然, MultipalyerSessionsSubsystem.cpp 中我们也要添加代码实现广播:

```
if (!SessionInterface->CreateSession(*LocalPlayer->GetPreferredUniqueNetId(),
NAME_GameSession, *LastSessionSettings))
{

    SessionInterface->ClearOnCreateSessionCompleteDelegate_Handle(CreateSessionComplete
DelegateHandle);
```

```

    /**
    * Broadcast our own custom delegate
    */
    MultiplayerOnCreateSessionComplete.Broadcast(false);
}

```

委托的回调函数也要进行修改：

```

void UMultiplayerSessionsSubsystem::OnCreateSessionComplete(FName SessionName, bool
bWasSuccessful)
{
    if (SessionInterface)
    {
        SessionInterface->ClearOnCreateSessionCompleteDelegate_Handle(CreateSessionComplete
DelegateHandle);
    }
    MultiplayerOnCreateSessionComplete.Broadcast(bWasSuccessful);
}

```

委托需要在重新编译插件之后才能正确编译，所以我们先把插件中的 Binaries 和 Intermediate 文件夹删除，然后右键项目文件生成 vs 工程，左键双击生成二进制文件。

## 021 更多委托子系统

这一章我们更新了 5 个不同种类的委托：

```
/**
 * Delcaring our own custom delegates for the Menu class to bind callbacks to
 */
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FMultiplayerOnCreateSessionComplete, bool,
bWasSuccessful);
DECLARE_MULTICAST_DELEGATE_TwoParams(FMultiplayerOnFindSessionsComplete, const
TArray<FOnlineSessionSearchResult>& SessionResults, bool bWasSuccessful);
DECLARE_MULTICAST_DELEGATE_OneParam(FMultiplayerOnJoinSessionComplete,
EOnJoinSessionCompleteResult::Type Result);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FMultiplayerOnDestroySessionComplete, bool,
bWasSuccessful);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FMultiplayerOnStartSessionComplete, bool,
bWasSuccessful);
```

其中由于 Find 委托传递的参数中含有不能蓝图化的类 `FOnlineSessionSearchResult` Join 中传递的加入结果同样不能蓝图化，所以这两个委托使用静态委托，其他委托则为动态委托。

```
public:
    /**
     * Our own custom delegates for the Menu class to bind callback to
     */
    FMultiplayerOnCreateSessionComplete MultiplayerOnCreateSessionComplete;
    FMultiplayerOnFindSessionsComplete MultiplayerOnFindSessionsComplete;
    FMultiplayerOnJoinSessionComplete MultiplayerOnJoinSessionComplete;
    FMultiplayerOnDestroySessionComplete MultiplayerOnDestroySessionComplete;
    FMultiplayerOnStartSessionComplete MultiplayerOnStartSessionComplete;
```

在 Menu.h 中添加相应的回调函数，动态委托需要加上宏标记，

```
/**
 * Callbacks for the custom delegates on the MultiplayerSessionsSubsystem
 */
UFUNCTION()
void OnCreateSession(bool bWasSuccessful);
void OnFindSessions(const TArray<FOnlineSessionSearchResult>& SessionResults, bool
bWasSuccessful);
void OnJoinSession(EOnJoinSessionCompleteResult::Type Result);
UFUNCTION()
void OnDestroySession(bool bWasSuccessful);
UFUNCTION()
void OnStartSession(bool bWasSuccessful);
```



在 Menu.cpp 的 setup 函数中统一绑定回调函数，动态委托用 AddDynamic 方法，静态委托使用 AddUObject 方法。

```
if (MultiplayerSessionsSubsystem)
{

    MultiplayerSessionsSubsystem->MultiplayerOnCreateSessionComplete.AddDynamic(this,
    &ThisClass::OnCreateSession);

    MultiplayerSessionsSubsystem->MultiplayerOnFindSessionsComplete.AddUObject(this,
    &ThisClass::OnFindSessions);

    MultiplayerSessionsSubsystem->MultiplayerOnJoinSessionComplete.AddUObject(this,
    &ThisClass::OnJoinSession);

    MultiplayerSessionsSubsystem->MultiplayerOnDestroySessionComplete.AddDynamic(this,
    &ThisClass::OnDestroySession);

    MultiplayerSessionsSubsystem->MultiplayerOnStartSessionComplete.AddDynamic(this,
    &ThisClass::OnStartSession);
}
```

此外还需要注意头文件的添加，以保证顺利编译。

## 022 从菜单加入会话

### ISSUE 2

## Session Settings

In the following video, we will set up our session settings. There is a line that must be added in order to successfully join sessions.

In `UMultiplayerSessionsSubsystem::CreateSession()`, please make sure you set the following variable on `LastSessionSettings`:

```
LastSessionSettings->bUseLobbiesIfAvailable = true;
```

This line is required.

### FINDING SESSION:

在 Menu.cpp 中的 join 按钮的回调函数中调用我们的子系统 FindSessions 方法:

```
void UMenu::JoinButtonClicked()
{
    if (MultiplayerSessionsSubsystem)
    {
        MultiplayerSessionsSubsystem->FindSessions(10000);
    }
}
```

然后, 我们需要在 MultiplayerSessionsSubsystem.cpp 中去实现这个方法:

```
void UMultiplayerSessionsSubsystem::FindSessions(int32 MaxSearchResults)
{
    // Find Game sessions
    if (!SessionInterface.IsValid())
    {
        return;
    }

    LastSessionSearch = MakeShareable(new FOnlineSessionSearch());
    LastSessionSearch->MaxSearchResults = MaxSearchResults;
    LastSessionSearch->bIsLanQuery = IOnlineSubsystem::Get()->GetSubsystemName() ==
    "NULL" ? true : false;
```

```

        LastSessionSearch->QuerySettings.Set(SEARCH_PRESENCE, true,
EOnlineComparisonOp::Equals);

        FindSessionsCompleteDelegateHandle =
SessionInterface->AddOnFindSessionsCompleteDelegate_Handle(FindSessionsCompleteDelegate
);
        const ULocalPlayer* LocalPlayer = GetWorld()->GetFirstLocalPlayerFromController();
        if (!SessionInterface->FindSessions(*LocalPlayer->GetPreferredUniqueNetId(),
LastSessionSearch.ToSharedRef()))
        {

            SessionInterface->ClearOnFindSessionsCompleteDelegate_Handle(FindSessionsCompleteDe
legateHandle);

            MultiplayerOnFindSessionsComplete.Broadcast(TArray<FOnlineSessionSearchResult>(), fa
lse);
        }
    }
}

```

其中，由于我们需要保存搜索结果以便之后使用，我们需要在.h 文件中添加一个成员用来存储该结果：

```
TSharedPtr<FOnlineSessionSearch> LastSessionSearch;
```

然后，实现 FindSessions 委托的回调函数：

```

void UMultiplayerSessionsSubsystem::OnFindSessionsComplete(bool bWasSuccessful)
{
    if (SessionInterface)
    {

        SessionInterface->ClearOnFindSessionsCompleteDelegate_Handle(FindSessionsCompleteDe
legateHandle);

        if (LastSessionSearch->SearchResults.Num() <= 0)
        {

            MultiplayerOnFindSessionsComplete.Broadcast(TArray<FOnlineSessionSearchResult>(),
false);

            return;
        }

        MultiplayerOnFindSessionsComplete.Broadcast(LastSessionSearch->SearchResults,
bWasSuccessful);
    }
}

```

## JOIN SESSION:

由于我们在之前多人在线子系统的委托中通过广播调用了 Menu.cpp 中 Join 的回调函数, 我—我们还需要先实现这个回调函数:

```
void UMenu::OnFindSessions(const TArray<FOnlineSessionSearchResult>& SessionResults,
bool bWasSuccessful)
{
    if (MultiplayerSessionsSubsystem == nullptr)
    {
        return;
    }
    for (auto Result : SessionResults)
    {
        FString SettingValue;
        Result.Session.SessionSettings.Get(FName("MatchType"), SettingValue);
        if (SettingValue == MatchType)
        {
            MultiplayerSessionsSubsystem->JoinSession(Result);
            return;
        }
    }
}
```

在这个函数中如果我们在传递过来的搜索结果中查找到了合适的会话, 我们会调用子系统加入函数加入他:

```
void UMultiplayerSessionsSubsystem::JoinSession(const FOnlineSessionSearchResult&
SessionResult)
{
    if (!SessionInterface.IsValid())
    {
        return;
    }

    MultiplayerOnJoinSessionComplete.Broadcast(EOnJoinSessionCompleteResult::UnknownError);

    JoinSessionCompleteDelegateHandle =
    SessionInterface->AddOnJoinSessionCompleteDelegate_Handle(JoinSessionCompleteDelegate);

    const ULocalPlayer* LocalPlayer = GetWorld()->GetFirstLocalPlayerFromController();
    if (SessionInterface->JoinSession(*LocalPlayer->GetPreferredUniqueNetId(),
NAME_GameSession, SessionResult))
    {
        SessionInterface->ClearOnJoinSessionCompleteDelegate_Handle(JoinSessionCompleteDele
```

```
gateHandle);
```

```
        MultiplayerOnJoinSessionComplete.Broadcast(EOnJoinSessionCompleteResult::UnknownError);  
    }  
}
```

以及实现回调函数:

```
void UMultiplayerSessionsSubsystem::OnJoinSessionComplete(FName SessionName,  
EOnJoinSessionCompleteResult::Type Result)  
{  
    if (SessionInterface)  
    {  
  
        SessionInterface->ClearOnJoinSessionCompleteDelegate_Handle(JoinSessionCompleteDelegateHandle);  
    }  
  
    MultiplayerOnJoinSessionComplete.Broadcast(Result);  
}
```

最后在 Menu 中实现 menu 的回调函数:

```
void UMenu::OnJoinSession(EOnJoinSessionCompleteResult::Type Result)  
{  
    IOnlineSubsystem* Subsystem = IOnlineSubsystem::Get();  
    if (Subsystem)  
    {  
        IOnlineSessionPtr SessionInterface = Subsystem->GetSessionInterface();  
        if (SessionInterface.IsValid())  
        {  
            FString Address;  
            SessionInterface->GetResolvedConnectString(NAME_GameSession, Address);  
  
            APlayerController* PlayerController =  
GetGameInstance()->GetFirstLocalPlayerController();  
            if (PlayerController)  
            {  
                PlayerController->ClientTravel(Address,  
ETravelType::TRAVEL_Absolute);  
            }  
        }  
    }  
}
```

至此我们完成了代码的部分，将游戏打包，在两台电脑上进行联网实验。

## 023 追踪玩家的加入

UE 里的游戏状态分为 2 部分：游戏模式以及游戏状态。

游戏模式包含：

1. 游戏规则
2. 玩家在关卡中的移动
3. 选择出生位置
4. PostLogin(APlayerController\* NewPlayer) 在玩家加入时调用，连接玩家控制器
5. Logout(AController\* Exiting) 玩家退出时调用，解除控制器

游戏状态包含：

1. 客户端状态
2. 非玩家信息状态
3. 玩家状态数组

## 创建一个 GameMode

再 Menusystem 中创建一个继承于游戏模式基类的 LobbyGameMode 类，重载函数：

public:

```
virtual void PostLogin(APlayerController* NewPlayer) override;  
virtual void Logout(AController* Exiting) override;
```

分别在.cpp 中进行实现：

```
void ALobbyGameMode::PostLogin(APlayerController* NewPlayer)  
{  
    Super::PostLogin(NewPlayer);  
  
    if (GameState)  
    {  
        int32 NumberOfPlayers = GameState.Get()->PlayerArray.Num();  
  
        if (GEngine)  
        {  
            GEngine->AddOnScreenDebugMessage(  
                1,  
                60.f,  
                FColor::Yellow,  
                FString::Printf(TEXT("Players in game: %d"), NumberOfPlayers)  
            );  
  
            APlayerState* PlayerState = NewPlayer->GetPlayerState<APlayerState>();  
            if (PlayerState)  
            {
```

```

        FString PlayerName = PlayerState->GetPlayerName();
        GEngine->AddOnScreenDebugMessage(
            -1,
            60.f,
            FColor::Cyan,
            FString::Printf(TEXT("%s has join the game!"), *PlayerName)
        );
    }
}

void ALobbyGameMode::Logout(AController* Exiting)
{
    Super::Logout(Exiting);

    APlayerState* PlayerState = Exiting->GetPlayerState<APlayerState>();

    if (PlayerState)
    {
        int32 NumberOfPlayers = GameState.Get()->PlayerArray.Num();
        if (GEngine)
        {
            GEngine->AddOnScreenDebugMessage(
                1,
                60.f,
                FColor::Yellow,
                FString::Printf(TEXT("Players in game: %d"), NumberOfPlayers - 1)
            );

            FString PlayerName = PlayerState->GetPlayerName();

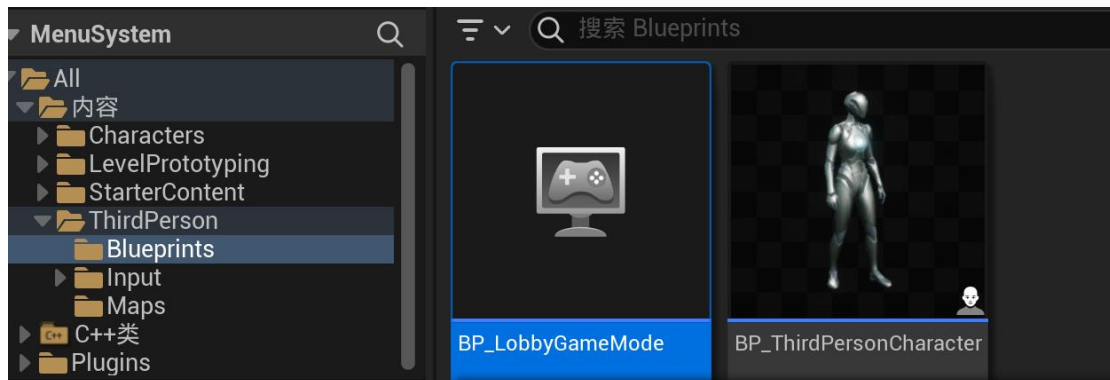
            GEngine->AddOnScreenDebugMessage(
                -1,
                60.f,
                FColor::Cyan,
                FString::Printf(TEXT("%s has exited the game!"), *PlayerName)
            );
        }
    }
}

```

在 `UMultiplayerSessionsSubsystem::CreateSession` 中添加一个设置：  
`LastSessionSettings->BuildUniqueId = 1;`

在配置文件 `DefaultGame.ini` 中设置最大游戏人数：  
`[/Script/Engine.GameSession]`  
`MaxPlayers = 100`

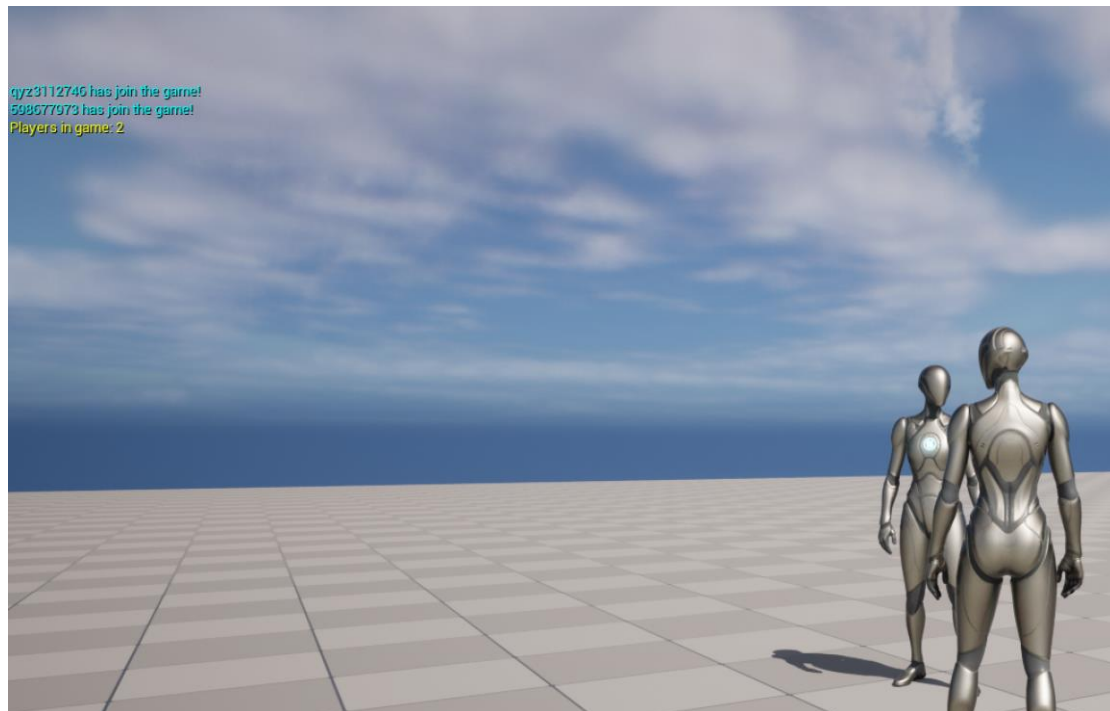
创建基于我们的 gamemode 的蓝图类：



蓝图类的设置中需要把默认的 pawn 换成第三人称蓝图类，因为使用默认类将没有 movement 的复制，再多人情况下看不到别人的运动。



之后，来到 Lobby 关卡，更换关卡的游戏模式类即可。





## 024 大厅的路径

之前我们将大厅的路径设置为了固定值，为了提高灵活性，我们在 MenuSetup 的函数中添加了一个传入的参数：

```
public:
    UFUNCTION(BlueprintCallable)
    void MenuSetup(int32 NumberOfPublicConnections = 4, FString TypeOfMatch =
FString(TEXT("FreeForAll")), FString LobbyPath =
FString(TEXT("/Game/ThirdPerson/Maps/Lobby")));
private:
FString PathToLobby{ TEXT("") };
.cpp修改Setup函数:
void UMenu::MenuSetup(int32 NumberOfPublicConnections, FString TypeOfMatch, FString
LobbyPath)
{
    PathToLobby = FString::Printf(TEXT("%s?listen"), *LobbyPath);
    NumPublicConnections = NumberOfPublicConnections;
    MatchType = TypeOfMatch;
    AddToViewport();
    SetVisibility(ESlateVisibility::Visible);
    bIsFocusable = true;

    UWorld* World = GetWorld();
    if (World)
    {
        APlayerController* PlayerController = World->GetFirstPlayerController();
        if (PlayerController)
        {
            FInputModeUIOnly InputModeData;
            InputModeData.SetWidgetToFocus(TakeWidget());
            InputModeData.SetLockMouseToViewportBehavior(EMouseLockMode::DoNotLock);
            PlayerController->SetInputMode(InputModeData);
            PlayerController->SetShowMouseCursor(true);
        }
    }
    UGameInstance* GameInstance = GetGameInstance();
    if (GameInstance)
    {
        MultiplayerSessionsSubsystem =
GameInstance->GetSubsystem<UMultiplayerSessionsSubsystem>();
    }

    if (MultiplayerSessionsSubsystem)
    {

```

```
MultiplayerSessionsSubsystem->MultiplayerOnCreateSessionComplete.AddDynamic(this,
&ThisClass::OnCreateSession);
```

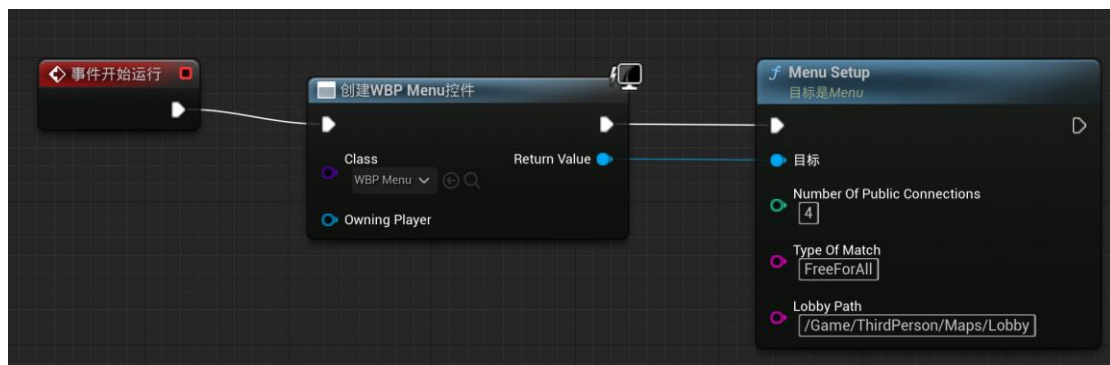
```
MultiplayerSessionsSubsystem->MultiplayerOnFindSessionsComplete.AddUObject(this,
&ThisClass::OnFindSessions);
```

```
MultiplayerSessionsSubsystem->MultiplayerOnJoinSessionComplete.AddUObject(this,
&ThisClass::OnJoinSession);
```

```
MultiplayerSessionsSubsystem->MultiplayerOnDestroySessionComplete.AddDynamic(this,
&ThisClass::OnDestroySession);
```

```
MultiplayerSessionsSubsystem->MultiplayerOnStartSessionComplete.AddDynamic(this,
&ThisClass::OnStartSession);
}
}
```

可以在关卡蓝图中看到变化：



## 025 改进菜单子系统

### 使创建总是可用

我们的子系统中存在一个问题，当我们从一个会话中退出后，创建新的会话时可能会失败，但是过一段时间后又可以成功创建，这是因为创建会话时我们会判断当前是否存在会话，如果存在会删除当前会话，之后创建新的会话。

但是会话的删除由于是网络服务，不是立刻完成的，这就造成了我们创建会话的时候会失败。为了解决这个问题，我们可以使用一个变量来标记是否 destroy 发生在了 create 的后面，如果是的话我们可以在 destroy 的回调函数中处理这个问题。

### 退出键

退出键实现过于简单，直接用蓝图即可，这里不做实现。

### 防止重复按键触发

核心思想就是按下某个按钮后就禁用该按钮，在合适的时候再解除禁用。

```
void UMenu::HostButtonClicked()
{
    HostButton->SetIsEnabled(false);
    if (MultiplayerSessionsSubsystem)
    {
        MultiplayerSessionsSubsystem->CreateSession(NumPublicConnections, MatchType);
    }
}

void UMenu::JoinButtonClicked()
{
    JoinButton->SetIsEnabled(false);
    if (MultiplayerSessionsSubsystem)
    {
        MultiplayerSessionsSubsystem->FindSessions(10000);
    }
}
```

具体实现可以看 github 中的源码。

至此我们完成了插件的制作，，代码中有 2 处错误，在 Blaster 的末尾有更正。