BlasterLearning

Multiplayer Plugin

1.概述

OnlineSubsystem多人插件子系统的整体的思路就是使用**Steam**平台进行联机，实现则是通过虚幻5引擎的在线子系统来实现。虚幻5的在线子系统封装了不同平台（如 **Steam、Epic**）的底层网络功能接口，使开发者通过统一接口实现多人联机功能，无需直接调用底层 **API**。这就有点类似**Spring**当中**Mybatis**一样，我们可以简单的使用**Mybatis**就可以做到之前**JDBC**那样去和数据库进行交互，但省去了当初直接使用JDBC的复杂过程。并且由于是统一接口，所以也不需要因为平台的不同还得去对代码和逻辑进行修改，跨平台性很高。

2.实现

2.1模块配置

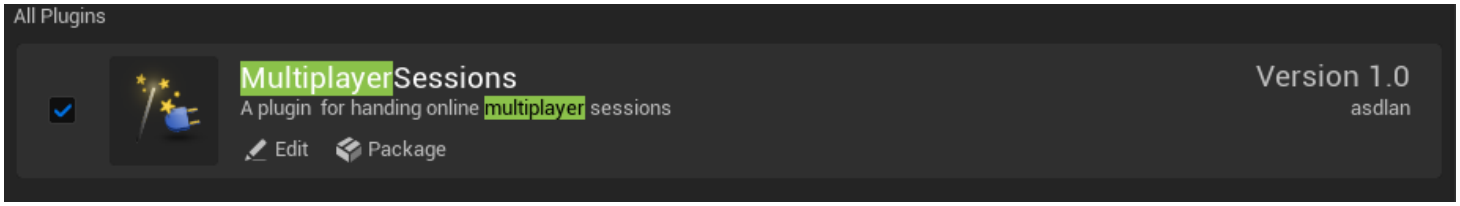
首先我们需要在引擎当中添加插件。

重启引擎，这样我们现在的项目现在就包含了在线子系统模块了，为接下来创建我们自己的在线子系统插件做准备。

2.2创建在线会话插件模块

MultiplayerSessions多人会话模块是依赖于虚幻5的多人在线子系统模块，用来实现整个游戏多人联机的部分。包括的功能有创建多人菜单界面，加入会话、创建会话、搜索会话、销毁会话等主要功能。和游戏主题分开，以插件形式使用，使得后续在其他项目可以直接引入并使用。

首先创建模块。



再在插件的uplugin和build中配置插件的依赖模块和声明模块。

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

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

在ini文件中配置使用Steam联机功能

```
[/Script/Engine.GameEngine]
+NetDriverDefinitions=(DefName="GameNetDriver",DriverClassName="OnlineSubsystemSteam.SteamNetDr

[OnlineSubsystem]
DefaultPlatformService=Steam

[OnlineSubsystemSteam]
bEnabled=true
SteamDevAppId=480
bInitServerOnClient=true

[/Script/OnlineSubsystemSteam.SteamNetDriver]
NetConnectionClassName="OnlineSubsystemSteam.SteamNetConnection"
```

这样必要的设置就完成了，重新编译项目成功就代表可以直接使用**MultiplayerSessions**插件去实现自定义的多人会话功能了。

2.3代码功能实现

2.3.1功能类的创建

MultiplayerSessions的两个功能类，**Menu**和**MultiplayerSessionsSubsystem**类。

1. **Menu**继承自**UserWidget**类，负责游戏菜单界面和功能按键的实现，是蓝图中**Widget**小组件的父类。对界面按钮进行动态事件绑定，初始化和自定义销毁方法。定义委托的回调函数并对自定义委托的回调函数进行绑定。
2. **MultiplayerSessionsSubsystem**类继承自**UGameInstanceSubsystem**类，用来实现会话相关方面的功能方法和会话接口委托的定义和回调函数的绑定。

2.3.2 Menu类

```
 UFUNCTION(BlueprintCallable)
 void MenuSetup(int32 NumOfPublicConnections = 4, FString TypeOfMatch = FString(TEXT("FreeForAl

protected:
 virtual bool Initialize() override;
 virtual void NativeDestruct() override;

 //
 //多人在线子系统的自定义委托的回调函数
 //
 UFUNCTION()
 void OnCreateSession(bool bWasSuccessful);
 void OnFindSessions(const TArray<FOnlineSessionSearchResult>& SessionResults, bool bWasSuccess
 void OnJoinSession(EOnJoinSessionCompleteResult::Type Result);
 UFUNCTION()
 void OnDestorySession(bool bWasSuccessful);
 UFUNCTION()
 void OnStartSession(bool bWasSuccessful);

private:

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

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

 UFUNCTION()
 void HostButtonClicked();

 UFUNCTION()
 void JoinButtonClicked();

 void MenuTearDown();

 //用来处理所有在线会话功能的子系统
 class UMultiplayerSessionsSubsystem* MultiplayerSessionsSubsystem;

 UPROPERTY(BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
 int32 NumPublicConnections{4};
```

```
UPROPERTY(BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
FString MatchType{TEXT("FreeForAll")};
FString PathToLobby{TEXT("")};
```

菜单初始化方法MenuSetup

```
void UMenu::MenuSetup(int32 NumOfPublicConnections, FString TypeOfMatch, FString LobbyPath)
{
 PathToLobby = FString::Printf(TEXT("%s?listen"), *LobbyPath);
 NumPublicConnections = NumOfPublicConnections;
 MatchType = TypeOfMatch;
 AddToViewport();
 SetVisibility(ESlateVisibility::Visible);
 SetFocus();

 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:
  MultiplayerSessionsSubsystem->MultiplayerOnFindSessionsComplete.AddUObject(this, &ThisClass::O
  MultiplayerSessionsSubsystem->MultiplayerOnJoinSessionComplete.AddUObject(this, &ThisClass::On
  MultiplayerSessionsSubsystem->MultiplayerOnDestorySessionComplete.AddDynamic(this, &ThisClass
  MultiplayerSessionsSubsystem->MultiplayerOnStartSessionComplete.AddDynamic(this, &ThisClass::O
 }
}
```

- `MenuSetup` 方法的作用是对菜单小组件进行初始化，接收大厅路径，连接人数，匹配模式，并把小组件添加到视窗，设置可见性，并聚焦。

- 通过获取 `World` 得到玩家控制器 `PlayerController` 。设置界面焦点,显示鼠标光标不锁定鼠标。仅让玩家和UI进行交互。
- 获取游戏实例，并从游戏实例当中得到 `MuliplayerSessionsSubsystem` 子系统，并把当中的委托绑定其回调函数。

初始化函数Initialize

```
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;
}
```

- Initialize方法在Widget小组件中多用于在首次创建之后，展示到视窗之前调用，重载Initialize方法对一些小组件的参数进行初始化。此项目则是对两个按钮的动态事件点击事件绑定。

NativeDestruct

```
void UMenu::NativeDestruct()
{
 MenuTearDown();
 Super::NativeDestruct();
}
```

- 在widget被销毁时执行清理操作，重载此方法调用MenuTearDown方法执行自定义的清理操作。并调用父类NativeDestruct。

MenuTearDown方法

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

- 用于从父类中移除小组件。
- 切换输入模式到仅游戏模式，关闭所有UI输入。
- 设置鼠标为不可见。

OnCreateSession方法

```
void UMenu::OnCreateSession(bool bWasSuccessful)
{
 if (bWasSuccessful)
 {
  UWorld* World = GetWorld();
  if (World)
  {
   World->ServerTravel(PathToLobby);
  }
 }
 else
 {
  if (GEngine)
  {
   GEngine->AddOnScreenDebugMessage(
    -1,
    15.f,
    FColor::Red,
    FString(TEXT("Failed to create session!"))
   );
  }
  HostButton->SetIsEnabled(true);
 }
}
```

- 当创建会话成功返回true时，回调函数会被触发。游戏跳转到大厅地图。
- 如果传回false输出创建会话失败，并设置按钮可重新按下。

OnFindSessions方法

```
void UMenu::OnFindSessions(const TArray<FOnlineSessionSearchResult>& SessionResults, bool bWasSu
{
 if (MultiplayerSessionsSubsystem == nullptr)
 {
  return;
 }
 for (auto Result : SessionResults)
 {
  FString Id = Result.GetSessionIdStr();
  FString User = Result.Session.OwningUserName;
  FString SettingsValue;
  Result.Session.SessionSettings.Get(FName("MatchType"), SettingsValue);
  if (SettingsValue == MatchType)
  {
   MultiplayerSessionsSubsystem->JoinSession(Result);
   return;
  }
 }
 if (!bWasSuccessful || SessionResults.Num() == 0)
 {
  JoinButton->SetIsEnabled(true);
 }
}
```

- 循环遍历的得到的会话结果列表，并且和MatchType进行匹配，如果成功则调用加入会话并传入此结果的值。
- 如果不成功或者会话列表为空，则重新设置加入按钮。

OnJoinSession方法

```
void UMenu::OnJoinSession(EOnJoinSessionCompleteResult::Type Result)
{
 IOnlineSubsystem* Subsystem = IOnlineSubsystem::Get();
 if (Subsystem)
 {
  IOnlineSessionPtr SessionInterface = Subsystem->GetSessionInterface();
  if (SessionInterface.IsValid())
  {
   FString Address;
   if (!SessionInterface->GetResolvedConnectString(NAME_GameSession, Address))
   {
    return;
   }

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

 if (Result != EOnJoinSessionCompleteResult::Success)
 {
  JoinButton->SetIsEnabled(true);
 }
}
```

- 得到子系统实例，从子系统实例中获取会话接口，并通过会话名称获取到IP地址。
- 如果成功获取IP地址，那就会调用客户端跳转，进入游戏关卡当中。
- 如果返回结果不是成功代表加入会话失败，重新设置加入按钮。

HostButtonClicked方法

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

- 点击按钮事件被触发则调用点击方法，设置按钮不可按下，防止重复操作。
- 调用创建会话方法并传入相关的游戏类型和连接人数。

JoinButtonClicked方法

```
void UMenu::JoinButtonClicked()
{
 JoinButton->SetIsEnabled(false);
 if (MultiplayerSessionsSubsystem)
 {
  MultiplayerSessionsSubsystem->FindSessions(20000);
 }
}
```

- 点击按钮事件被触发则调用点击方法，设置按钮不可按下，防止重复操作。
- 调用查找会话方法，并限制结果为20000。

2.3.3 MultiplayerSessionsSubsystem类

```
public:
 UMultiplayerSessionsSubsystem();

 //
 //为了处理会话功能，Menu 类将调用这些
 //
 void CreateSession(int32 NumPubilcConnections, FString MatchType);
 void FindSessions(int32 MaxSearchResults);
 void JoinSession(const FOnlineSessionSearchResult& SessionResult);
 void DestorySession();
 void StartSession();

 //
 //为 Menu 类声明的自定义委托，用于绑定回调函数
 //
 FMultiplayerOnCreateSessionComplete MultiplayerOnCreateSessionComplete;
 FMultiplayerOnFindSessionsComplete MultiplayerOnFindSessionsComplete;
 FMultiplayerOnJoinSessionComplete MultiplayerOnJoinSessionComplete;
 FMultiplayerOnDestorySessionComplete MultiplayerOnDestorySessionComplete;
 FMultiplayerOnStartSessionComplete MultiplayerOnStartSessionComplete;

 int32 DesiredNumPublicConnections{};
 FString DesiredMatchType{};

protected:

 //
 //将添加到在线会话接口委托列表中的委托的内部回调
 //这些回调函数不需要在类外部调用
 //
 void OnCreateSessionComplete(FName SessionName, bool bWasSuccessful);
 void OnFindSessionsComplete(bool bWasSuccessful);
 void OnJoinSessionComplete(FName SessionName, EOnJoinSessionCompleteResult::Type Result);
 void OnDestorySessionComplete(FName SessionName, bool bWasSuccessful);
 void OnStartSessionComplete(FName SessionName, bool bWasSuccessful);

private:
 IOnlineSessionPtr SessionInterface;
 TSharedPtr<FOnlineSessionSettings> LastSessionSettings;
 TSharedPtr<FOnlineSessionSearch> LastSessionSearch;
```

```
//
//添加到在线会话接口委托列表中
//把 MultiplayerSessionsSubsystem 的内部回调绑定到这些委托上
//
FOnCreateSessionCompleteDelegate CreateSessionCompleteDelegate;
FDelegateHandle CreateSessionCompleteDelegateHandle;
FOnFindSessionsCompleteDelegate FindSessionsCompleteDelegate;
FDelegateHandle FindSessionsCompleteDelegateHandle;
FOnJoinSessionCompleteDelegate JoinSessionCompleteDelegate;
FDelegateHandle JoinSessionCompleteDelegateHandle;
FOnDestroySessionCompleteDelegate DestorySessionCompleteDelegate;
FDelegateHandle DestorySessionCompleteDelegateHandle;
FOnStartSessionCompleteDelegate StartSessionCompleteDelegate;
FDelegateHandle StartSessionCompleteDelegateHandle;

bool bCreateSessionOnDestory{ false };
int32 LastNumPublicConnections;
FString LastMatchType;
```

UMultiplayerSessionsSubsystem构造函数

```
UMultiplayerSessionsSubsystem::UMultiplayerSessionsSubsystem() :
 CreateSessionCompleteDelegate(FOnCreateSessionCompleteDelegate::CreateUObject(this, &ThisClass
 FindSessionsCompleteDelegate(FOnFindSessionsCompleteDelegate::CreateUObject(this, &ThisClass::C
 JoinSessionCompleteDelegate(FOnJoinSessionCompleteDelegate::CreateUObject(this, &ThisClass::On
 DestorySessionCompleteDelegate(FOnDestroySessionCompleteDelegate::CreateUObject(this, &ThisCla
 StartSessionCompleteDelegate(FOnStartSessionCompleteDelegate::CreateUObject(this, &ThisClass::C
{\
 IOnlineSubsystem* Subsystem = IOnlineSubsystem::Get();
 if (Subsystem)
 {
  SessionInterface = Subsystem->GetSessionInterface();
 }
}
```

- 为会话相关的委托绑定回调函数。
- 获得会话接口对象。

CreateSession方法

```
void UMultiplayerSessionsSubsystem::CreateSession(int32 NumPubilcConnections, FString MatchType)
{
 DesiredNumPublicConnections = NumPubilcConnections;
 DesiredMatchType = MatchType;
 if (!SessionInterface.IsValid())
 {
  return;
 }

 auto ExistingSesion = SessionInterface->GetNamedSession(NAME_GameSession);
 if (ExistingSesion != nullptr)
 {
  bCreateSessionOnDestory = true;
  LastNumPublicConnections = NumPubilcConnections;
  LastMatchType = MatchType;

  DestorySession();
 }
 //将委托存储在 FDelegateHandle 中，以便我们可以稍后将其从委托列表中移除。
 CreateSessionCompleteDelegateHandle = SessionInterface->AddOnCreateSessionCompleteDelegate_Hand

 LastSessionSettings = MakeShareable(new FOnlineSessionSettings());
 LastSessionSettings->bIsLANMatch = IOnlineSubsystem::Get()->GetSubsystemName() == "NULL" ? true
 LastSessionSettings->NumPublicConnections = NumPubilcConnections;
 LastSessionSettings->bAllowJoinInProgress = true;
 LastSessionSettings->bAllowJoinViaPresence = true;
 LastSessionSettings->bShouldAdvertise = true;
 LastSessionSettings->bUsesPresence = true;
 LastSessionSettings->bUseLobbiesIfAvailable = true;
 LastSessionSettings->Set(FName("MatchType"), MatchType, EOnlineDataAdvertisementType::ViaOnline
 LastSessionSettings->BuildUniqueId = 1;

 const ULocalPlayer* LocalPlayer = GetWorld()->GetFirstLocalPlayerFromController();
 if (!SessionInterface->CreateSession(*LocalPlayer->GetPreferredUniqueNetId(), NAME_GameSession,
 {
  SessionInterface->ClearOnCreateSessionCompleteDelegate_Handle(CreateSessionCompleteDelegateHan

  //广播我们自己的自定义委托
  MultiplayerOnCreateSessionComplete.Broadcast(false);
 }
}
```

- 存储传入的连接数量和游戏类型，验证游戏实例。
- 判断要创建的会话是否存在，如果存在销毁会话。
- 对绑定的委托在会话接口中进行注册，存储注册之后得到的句柄。
- 设置要创建的会话的各种设置。
- 调用会话接口创建会话。判断返回值进行删除委托和广播自定义委托操作。

FindSessions方法

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

 FindSessionsCompleteDelegateHandle = SessionInterface->AddOnFindSessionsCompleteDelegate_Handle

 LastSessionSearch = MakeShareable(new FOnlineSessionSearch());
 LastSessionSearch->MaxSearchResults = MaxSearchResults;
 LastSessionSearch->bIsLanQuery = IOnlineSubsystem::Get()->GetSubsystemName() == "NULL" ? true
 LastSessionSearch->QuerySettings.Set(SEARCH_PRESENCE, true, EOnlineComparisonOp::Equals);

 const ULocalPlayer* LocalPlayer = GetWorld()->GetFirstLocalPlayerFromController();
 if (!SessionInterface->FindSessions(*LocalPlayer->GetPreferredUniqueNetId(), LastSessionSearch
 {
  SessionInterface->ClearOnFindSessionsCompleteDelegate_Handle(FindSessionsCompleteDelegateHandl

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

- 注册查找会话委托。
- 设置查询会话条件。
- 调用查找会话方法并按照查询条件查询。
- 根据结果进行删除句柄和携带查询结果广播自定义委托。

JoinSession方法

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

 JoinSessionCompleteDelegateHandle = SessionInterface->AddOnJoinSessionCompleteDelegate_Handle(

 const ULocalPlayer* LocalPlayer = GetWorld()->GetFirstLocalPlayerFromController();
 if (!SessionInterface->JoinSession(*LocalPlayer->GetPreferredUniqueNetId(), NAME_GameSession, S
 {
  SessionInterface->ClearOnJoinSessionCompleteDelegate_Handle(JoinSessionCompleteDelegateHandle)

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

- 如果不存在会话实例直接广播委托错误。
- 注册委托并保存句柄。
- 调用加入会话方法，如果加入会话失败则删除注册的委托并广播自定义委托。

DestorySession方法

```
void UMultiplayerSessionsSubsystem::DestorySession()
{
 if (!SessionInterface.IsValid())
 {
  MultiplayerOnDestorySessionComplete.Broadcast(false);
  return;
 }

 DestorySessionCompleteDelegateHandle =  SessionInterface->AddOnDestroySessionCompleteDelegate_H

 if (!SessionInterface->DestroySession(NAME_GameSession))
 {
  SessionInterface->ClearOnDestroySessionCompleteDelegate_Handle(DestorySessionCompleteDelegateH
  MultiplayerOnDestorySessionComplete.Broadcast(false);
 }
}
```

- 会话实例不存在则广播自定义委托。
- 调用销毁会话方法，如果返回为fasle，删除注册的委托，并广播自定义委托。

OnCreateSessionComplete回调函数

```
void UMultiplayerSessionsSubsystem::OnCreateSessionComplete(FName SessionName, bool bWasSuccessf
{
 if (SessionInterface)
 {
  SessionInterface->ClearOnCreateSessionCompleteDelegate_Handle(CreateSessionCompleteDelegateHar
 }

 MultiplayerOnCreateSessionComplete.Broadcast(bWasSuccessful);
}
```

- 清除注册的委托并广播自定义委托。

OnFindSessionsComplete回调函数

```
void UMultiplayerSessionsSubsystem::OnFindSessionsComplete(bool bWasSuccessful)
{

 if (SessionInterface)
 {
  SessionInterface->ClearOnFindSessionsCompleteDelegate_Handle(FindSessionsCompleteDelegateHandl
 }

 if (LastSessionSearch->SearchResults.Num() <= 0)
 {
  MultiplayerOnFindSessionsComplete.Broadcast(TArray<FOnlineSessionSearchResult>(), false);
  return;
 }

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

- 清除注册的委托，根据查询结果个数判断广播是否携带参数。

OnJoinSessionComplete回调函数

```
void UMultiplayerSessionsSubsystem::OnJoinSessionComplete(FName SessionName, EOnJoinSessionCompl
{
 if (SessionInterface)
 {
  SessionInterface->ClearOnJoinSessionCompleteDelegate_Handle(JoinSessionCompleteDelegateHandle)
 }

 MultiplayerOnJoinSessionComplete.Broadcast(Result);
}
```

- 清除注册的句柄并广播自定义委托。

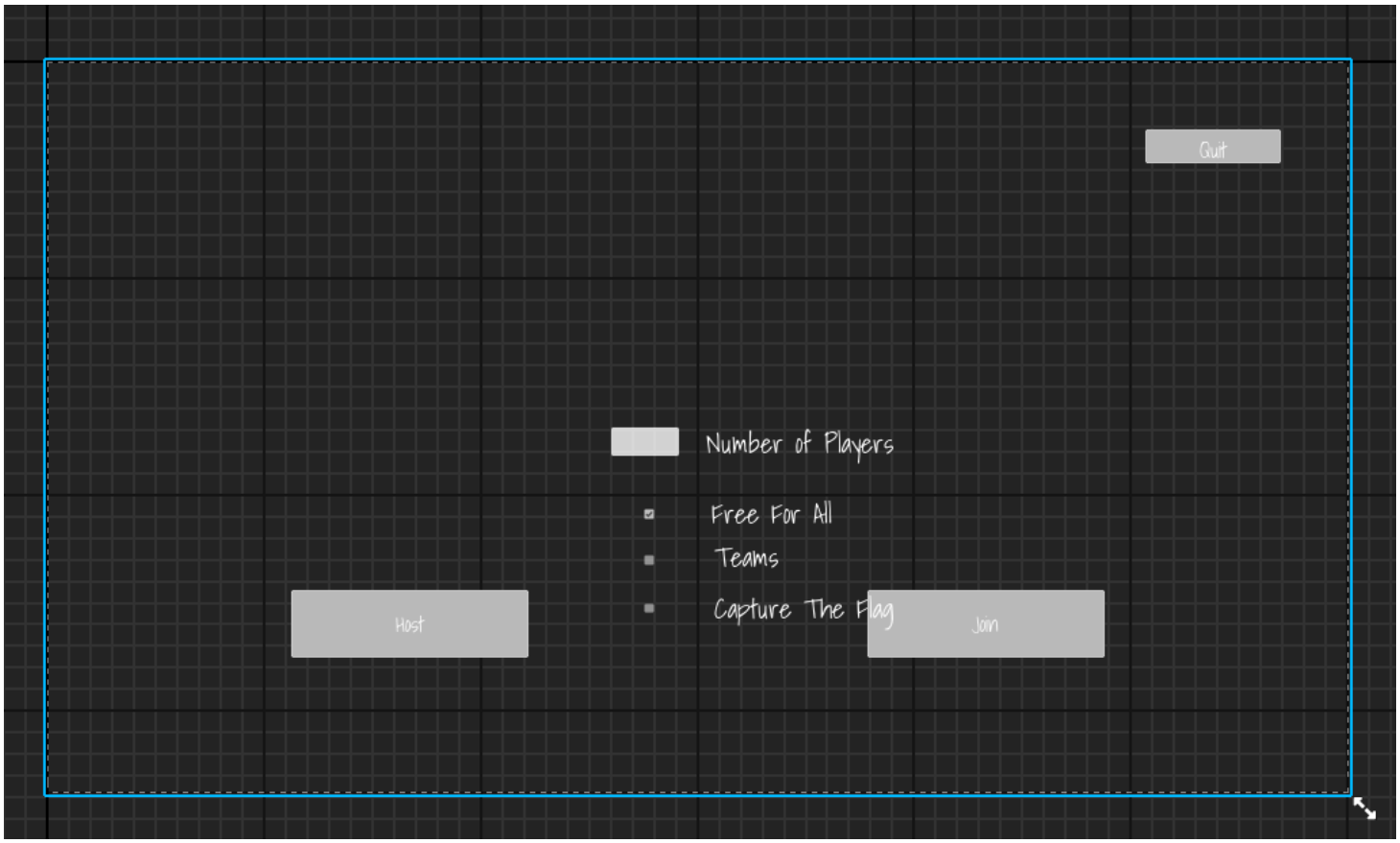
OnDestorySessionComplete回调函数

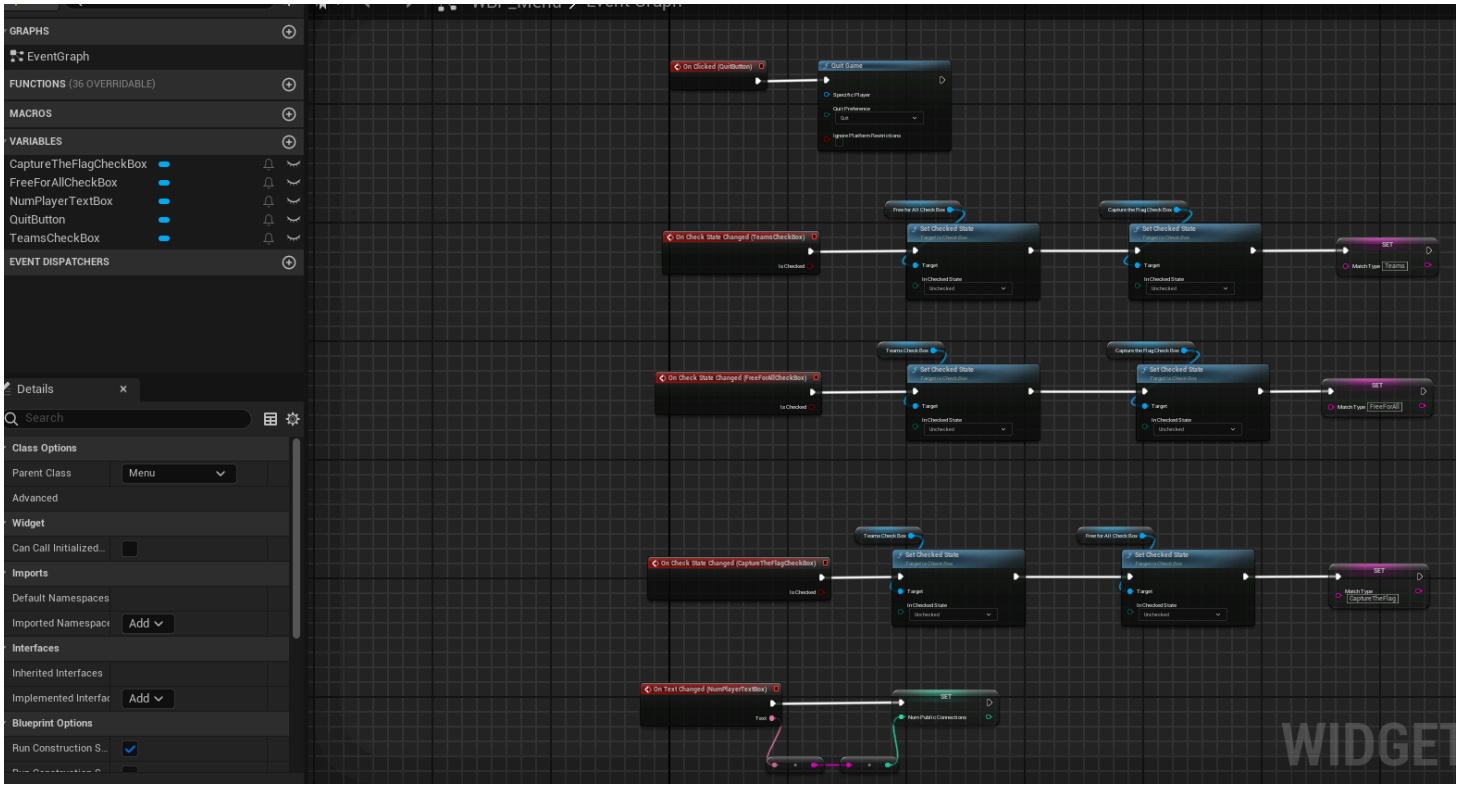
```
void UMultiplayerSessionsSubsystem::OnDestorySessionComplete(FName SessionName, bool bWasSucces
{
 if (SessionInterface)
 {
  SessionInterface->ClearOnDestroySessionCompleteDelegate_Handle(DestorySessionCompleteDelegateH
 }

 if (bWasSuccessful && bCreateSessionOnDestory)
 {
  bCreateSessionOnDestory = false;
  CreateSession(LastNumPublicConnections, LastMatchType);
 }

 MultiplayerOnDestorySessionComplete.Broadcast(bWasSuccessful);
}
```

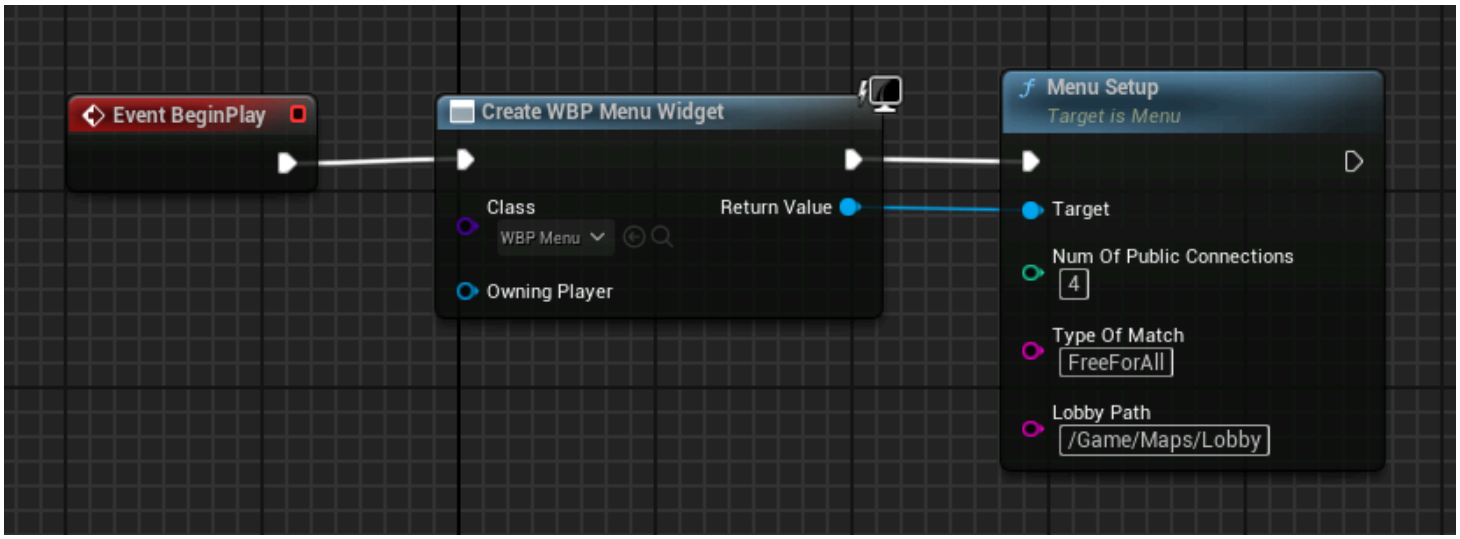
- 清除注册的句柄并广播自定义委托。

2.4 蓝图类

在蓝图中创建一个**UserWidget**类并继承自Menu。并绑定按钮和在地图的蓝图开始事件中创建小组件并调用MenuSetup方法。


Quit
Number of Players
Free For All
Teams
Capture The Flag
Host
Join


GRAPHS
EventGraph
FUNCTIONS (36 OVERRIDABLE)
MACROS
VARIABLES
CaptureTheFlagCheckBox
FreeForAllCheckBox
NumPlayerTextBox
QuitButton
TeamsCheckBox
EVENT DISPATCHERS
Details
Search
Class Options
Parent Class
Menu
Advanced
Widget
Can Call Initialized...
Imports
Default Namespaces
Imported Namespace
Add
Interfaces
Inherited Interfaces
Implemented Interfa
Add
Blueprint Options
Run Construction S...
On Clicked (QuitButton)
Quit Game
On Check State Changed (TeamsCheckBox)
On Check State Changed (FreeForAllCheckBox)
On Check State Changed (CaptureTheFlagCheckBox)
On Text Changed (NumPlayerTextBox)
WIDGET



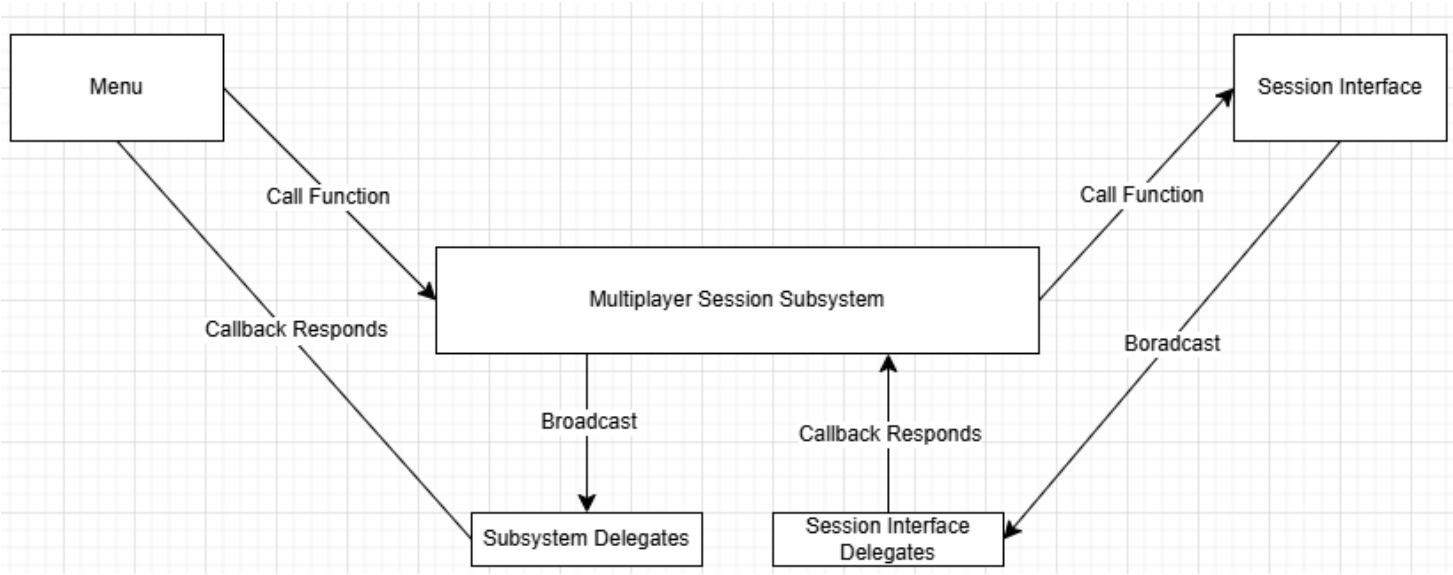
3.理解

委托我的理解就是，虚幻当中在线子系统中关于会话有很多已经有的委托，我们可以通过这些委托绑定回调函数，告诉引擎这些异步操作完成之后我们想让其执行的操作，但绑定的回调函数要符合这些委托绑定的要求，参数必须一致。而绑定之后还需要对这些委托进行注册，也就是在会话接口中的委托列表进行注册，系统就会返回给我们一个句柄，这就类似一个凭证，代表这个委托已经成功注册在系统当中。而如果后续不再想使用这个绑定委托只需去告诉引擎清除这个句柄。引擎就不会再当特定事件发生时委托去调用设置的回调函数。

而在其中，自定义委托使用了两种委托，**动态多播**和**多播**，动态多播是完全支持虚幻的反射系统和蓝图的，而使用两种的原因是，其中的查询和加入会话需要携带参数，但是这些参数并不是UClass相关的，也就不支持蓝图和反射系统，因此就使用了正常的多播委托，并且如果想用动态多播委托可以把这些参数使用虚幻相关类进行包装，或者自定义返回类型以便在蓝图中使用。但是这样是没有任何意义的，因为并不需要这样做，多播委托就已经可以胜任这个工作了，使用的资源更低，性能更高。动态多播和多播在声明上也有不同，动态多播的参数声明和参数名之间有一个逗号。

而多人插件的整体逻辑就是**Menu**类对象去调用**MultiplayerSessionsSubsystem**类对象中定义的会话相关方面的方法，这些方法会去向会话接口**Session Interface**对象发起相关的会话请求，当这些异步操作完成之后会触发其在引擎中对应的委托，并且我们对这些委托的回调函数也进行了注册和绑定。因此绑定的回调函数会对委托进行响应执行，其中会去广播我们自定义的委托，当自定义委托触发，**Menu**类对象中的回调函数就会对自定义委托进行响应，回调函数被调用作用于**Menu**类对象，使得玩家窗口根据这些回调函数作出改变比如修改UI或者跳转地图等操作。并且委托一旦被触发就会去清除句柄确保一次只执行一次回调函数保证了不会出现重复注册的情况发生，因此不会出现因为没有清除和解绑等造成的错误的函数调用。**MultiplayerSessionsSubsystem**类继承自游戏实例子系统类**GameInstanceSubsystem**也保证了获取到的会话接口和子系统等对象的唯一正确性，并在整个游戏实例当中都是有效的。确保了声明周期和游戏实例的一致,并且在游戏实例的关卡跳转过程中不会出现数据丢失等问题。

4.结构图


Menu
Session Interface
Call Function
Call Function
Multiplayer Session Subsystem
Callback Responds
Boradcast
Broadcast
Callback Responds
Subsystem Delegates
Session Interface Delegates