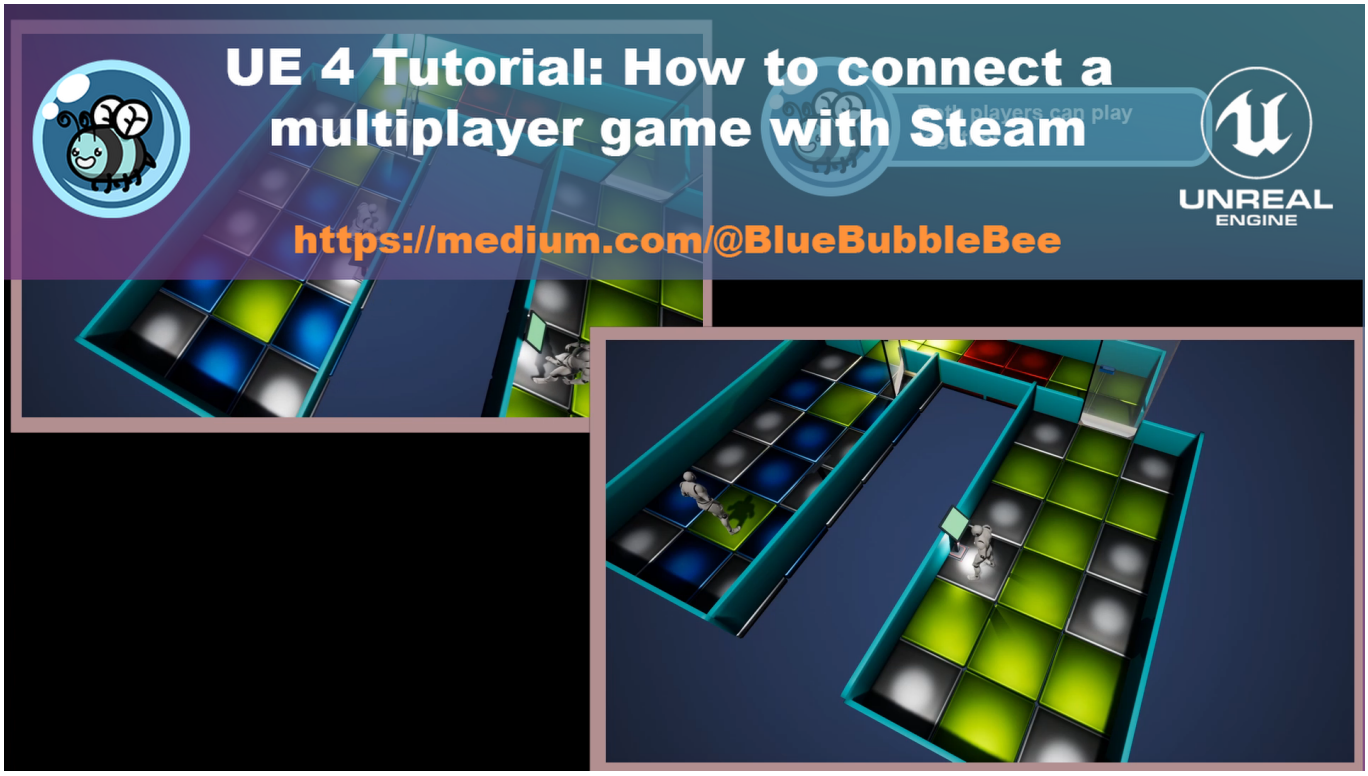
UE4 Tutorial: How to Connect a Multiplayer Game with Steam

Connect an existing project with Steam Tools and play with your friends

BlueBubbleBee Follow
Dec 15, 2019 · 8 min read ★



bluebubblebee/UE4_CoopPuzzleGame

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

. . .

Prepare the project for Steam

First step. Add the OnlineSubsystem and OnlineSubsystemSteam module to the dependencies build file.

This is a c# file named NameOfYourProject.Build.cs

In my case is **CoopPuzzleGame.Build.cs**

What it does is to include a new module on our project, in this case the OnlineSubsystem, and this will allows us to get access to extra header files in the code.

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

Online Subsystem Documentation:

Online Subsystem

The Online Subsystem and its interfaces provide a common way to access the functionality of online services such as...

docs.unrealengine.com

Add the Steam plugin

Edit menu, Plugins, we have to enable Online Subsystem Steam plugin

Open DefaultEngine.ini under your the configs folder on your project and add the next sections at the bottom:

```
[/Script/Engine.GameEngine]
+NetDriverDefinitions=
(DefName="GameNetDriver",DriverClassName="OnlineSubsystemSteam.SteamN
etDriver",DriverClassNameFallback="OnlineSubsystemUtils.IpNetDriver")

[OnlineSubsystem]
DefaultPlatformService=Steam

[OnlineSubsystemSteam]
bEnabled=true
SteamDevAppId=480

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

[Script/Engine.GameEngine]: This enables the net driver

[OnlineSubsystem]: Sets the online system, which in this case is steam

[OnlineSubsystemSteam]: Enable the steam system and sets the app id.
The steam app Id is the online id that you can use to test and develop for free.
If you want your app id, you need to be a developer on Steam, open an account and pay the $100 fee.

With a custom app ID, you can set achievements, cloud settings and so on.

[/Script/OnlineSubsystemSteam.SteamNetDriver]: This is the net driver which allows to have a direct connection to steam and not bother with creating a custom API.

The magic happens under the hood for us, so we don't have to worry about this part. Thank Unreal you are amazing.

At this point, it is important to close visual studio and unreal editor and open everything again.

Check the documentation to extend information:

Online Subsystem Steam

Contributors: Valve The Online Subsystem Steam API enables you to ship Unreal Engine 4 (UE4) applications to Valve's...

docs.unrealengine.com

. . .

In order to run a game from Steam, you can't do it from the editor, it has to be outside of the editor.

We have to use the command prompt of windows for testing and Steam running.

We have to run the next command twice to allow 2 players and check that we are connecting with steam:

```
"D:\Software\Unreal\Epic
Games\UE_4.22\Engine\Binaries\Win64\UE4Editor.exe"
"D:\CoopPuzzleGame\CoopPuzzleGame.uproject" -game -log
```

- The first part of the command indicates the location on your pc where Unreal Editor is installed
- The second part indicates the project location

- **game -log** simply means that we run an instance of the game with logs included.

If the game runs, we can continue with the fun part, the code.

. . .

Handling online sessions

I am sure you have played some online games, and you are more than used to go through the online menus, create sessions, wait for friends to connect, matchmaking, and all of that…

That's what we will implement in this section, creating, hosting, find and joint to sessions and with a bit of luck play with a friend.
Let's go.

Preparing Game Instance

The best place to handle this implementation is on the **GameInstance**, which is globally accessible class that can store any data you want to be carried between levels, sounds good ha?

First, we need a GameInstance extending from **GameInstance**, which I called **CooopPuzzleGameInstance**.

This GameInstance need to be set on the Project Settings, under Maps & Modes, there is a Game Instance section:

The **GameInstance** will have 2 jobs, handling sessions (create, destroy, find and join) and respond and send messages to a menu.

Let's start with the Main Menu

This part is simple, a couple of buttons to create or join menu, and a list with all the sessions that a player can join.

It will have a reference to an **USessionMenuInterface** interface to send messages to any other class that implements the interface, in this case, the **CooopPuzzleGameInstance.**

The buttons on the menu will call methods on the interface such as join a game session, create it… and the **GameInstance** will perform the implementation.

USessionMenuInterface methods:

```
virtual void Host(FString ServerName) = 0;

virtual void JoinSession(uint32 Index) = 0;

virtual void EndSession() = 0;
```

The main menu contains a pointer reference to the interface **USessionMenuInterface.**

. . .

CooopPuzzleGameInstance logic

This is the entry point, the logic starts here. This class contains a **BlueprintCallable** event called **LoadMainMenu**, which is called from the level blueprint.

This method creates a new MainMenu widget and sets the **USessionMenuInterface** interface, which **CooopPuzzleGameInstance** implements:

Setting up the Load Main Menu

The **CooopPuzzleGameInstance** waits for messages received from the **MainMenu,** for example, create a session or join a session.

The **MainMenu** class extends from UUserWidget and has references to the elements on the widget blueprint. For example, it contains a reference to the button new session:

```
UPROPERTY(meta = (BindWidget))
class UButton* NewSessionButton;UPROPERTY(meta = (BindWidget))
```

On the method initialize of the Main Menu, the buttons subscribe to dynamic event when a button is pressed.

For example, the NewSessionButton has an event called **OnNewSessionPressed:**

```
NewSessionButton->OnClicked.AddDynamic(this,
&UMainMenu::OnNewSessionPressed);
```

The method OnNewSessionPressed will call the interface and calls the method Host.

Remember, Game Instance implements the interface SessionMenuInterface, so the actual implementation of Host is on the CoopPuzzleGameInstance:

```
void UMainMenu::OnNewSessionPressed()
{
   if (SessionMenuInterface == nullptr) return;
   SessionMenuInterface->Host("CoopPuzzleGameServer");
}
```

The MainMenu implementation is not the important part, the important one is the GameInstance Implementation, so let’s go step by step.

. . .

Initialize CoopPuzzleGameInstance

The first step to start creating sessions is to initialize the CoopPuzzleGameInstance.

On the initialization, we look for any online subsystem that is available, if we have any (in this case it will be Steam), we get a reference pointer of type IOnlineSessionPtr:

```
// Declare of SessionInterface for the subsystem
IOnlineSessionPtr SessionInterface;
```

If that pointer is correct, we will subscribe to events to handle sessions (OnCreateComplete, OnDestroyComplete, OnFindSessionComplete, OnJoinSessionComplete):

```
void UCoopPuzzleGameInstance::Init()
{
   IOnlineSubsystem* SubSystem = IOnlineSubsystem::Get();
   SessionInterface = SubSystem->GetSessionInterface();
   if (SessionInterface.IsValid())
   {

// Subscribe to minimun events to handling sessions
      SessionInterface->OnCreateSessionCompleteDelegates.
AddUObject(this, &UCoopPuzzleGameInstance::OnCreateSessionComplete);

      SessionInterface->OnDestroySessionCompleteDelegates.
AddUObject(this, &UCoopPuzzleGameInstance::OnDestroySessionComplete);

      SessionInterface->OnFindSessionsCompleteDelegates.
AddUObject(this, &UCoopPuzzleGameInstance::OnFindSessionsComplete);

      SessionInterface->OnJoinSessionCompleteDelegates.
AddUObject(this, &UCoopPuzzleGameInstance::OnJoinSessionsComplete);
   }
}
```

- OnCreateSessionCompleteDelegates: Handles when a session has been created
- OnDestroySessionCompleteDelegates: Handles when a session has been destroyed

- OnFindSessionsCompleteDelegates: Handles when we are trying to find existing sessions
- OnJoinSessionsComplete: Handles, when joining a session, has been completed

. . .

Create a Session

When we press the button Host Session, the method Host of the ISessionMenuInterface is called and received by UCoopPuzzleGameInstance.

As a parameter, this function receives the ServerName (any name will do for testing).

Creating a session is simple, we have to first check if we have a valid online subsystem (**IOnlineSubsystem**), in this case, is Steam and after that, initialize the session settings, and finally call **CreateSession** with those settings:

```
if (SessionInterface.IsValid())
{
   FOnlineSessionSettings SessionSettings;

   SessionSettings.bIsLANMatch = false;
   SessionSettings.NumPublicConnections = 2;
   SessionSettings.bShouldAdvertise = true;
   SessionSettings.bUsesPresence = true;
   SessionSettings.Set(SERVER_NAME_SETTINGS_KEY, DesiredServerName,
            EOnlineDataAdvertisementType::ViaOnlineServiceAndPing);

   SessionInterface->CreateSession(0, "CoopPuzzleGameSession",
                                      SessionSettings);
}
```

Check the official documentation to check the settings for a session:

FOnlineSessionSettings

Container for all settings describing a single online session Module OnlineSubsystem Header...

docs.unrealengine.com

The next player will be able to join that session or create a new one.

. . .

Join an existing session

If a player decides to press the button join, the menu interface will show all the available sessions.

Remember, we have a reference to the SessionMenuInterface, so we need to call a method on that interface to populate a list of sessions on the menu from the Game Instance.

There are 2 steps here:

- Create a new TSharedPtr with a SessionSearch
- Populate the list with the result of that search.

```
// Shareable pointer to the session
TSharedPtr<class FOnlineSessionSearch> SessionSearch;

// Method called from the menu to find sessions
void UCoopPuzzleGameInstance::OpenSessionListMenu()
{
  // Create the pointer
 SessionSearch = MakeShareable(new FOnlineSessionSearch());
   if (SessionSearch.IsValid())
   {
      // Set properties
      SessionSearch->MaxSearchResults = 100;
      SessionSearch->QuerySettings.Set(SEARCH_PRESENCE, true,
                                 EOnlineComparisonOp::Equals);
      SessionInterface->FindSessions(0,
          SessionSearch.ToSharedRef());
    }
}
```

To see more of the properties for the pointer, check the documentation: https://docs.unrealengine.com/en-US/API/Plugins/OnlineSubsystem/FOnlineSessionSearch/index.html

When the search has been completed, there is event delegate waiting on the Game Instance, this is **OnFindSessionsComplete**

This event will loop through all the sessions and will add all the data to an array of FServerData custom struct and send that information to the menu:

```
USTRUCT()
struct FServerData
{
   GENERATED_BODY()
   FString Name;
   uint16 CurrentPlayers;
   uint16 MaxPlayers;
   FString HostUsername;
};
```

The code for the event

```
void UCoopPuzzleGameInstance::OnFindSessionsComplete(bool Success)
{
   if (Success && SessionSearch.IsValid())
   {
      if (SessionSearch->SearchResults.Num() > 0)
      {
         TArray<FServerData> ServerData;
         for (const FOnlineSessionSearchResult& SearchResult :
                                    SessionSearch->SearchResults)
        {
            FServerData Data;
            FString ServerName;
            if
(SearchResult.Session.SessionSettings.Get(SERVER_NAME_SETTINGS_KEY,
ServerName))
             {
                 Data.Name = ServerName;
             }

             // Fill information for the menu

Data.MaxPlayers =
SearchResult.Session.SessionSettings.NumPublicConnections;
             Data.CurrentPlayers = Data.MaxPlayers -
SearchResult.Session.NumOpenPublicConnections;
             Data.HostUsername = SearchResult.Session.OwningUserName;
             ServerData.Add(Data);
      }
```

```
            // Send the information back to the menu
            MainMenu->InitializeSessionsList(ServerData);
        }
    }
```

As you can see, the flow between the menu and the Game Instance goes through the interface.

From the menu, we press the buttons and calls methods on the interface.
The Game Instance receives these methods and acts according to that.

Finally, the Game Instance calls the menu again to send any extra information.

When the previous code is executed we should see on the menu, a list of sessions to joins.

The player will select one and press the Join button to start the game.

. . .

Join a selected session

The menu has a listener on the Join button called OnJoinSelectedSession.
All this code does is to grab the index on the list and call again the interface with that information.

```
// Calling the method Join Session with a specific index
// SelectedScrollIndex is an uint32 representing the row on a scroll
// rect
MenuInterface->JoinSession(SelectedScrollIndex.GetValue());
```

The Game Instance receives the method JoinSession and acts.
There are again 2 parts, find that specific session in our previous SessionSearch:

```
void UEscapeRoomGameInstance::JoinSession(uint32 Index)
{
     // SESSION_NAME is a constant for the sessions
```

```
     // const static FName SESSION_NAME =
TEXT("CoopPuzzleGameSession");

     if (Index < (uint32)(SessionSearch->SearchResults.Num()))
     {
        SessionInterface->JoinSession(0, SESSION_NAME, SessionSearch-
>SearchResults[Index]);
     }
]
```

The specific event for this JoinSession is:

```
SessionInterface->OnJoinSessionCompleteDelegates.AddUObject(this,
&UCoopPuzzleGameInstance::OnJoinSessionsComplete);
```

Finally, the method OnJoinSessionsComplete:

```
void UEscapeRoomGameInstance::OnJoinSessionsComplete(FName
SessionName, EOnJoinSessionCompleteResult::Type Result)
{
    // Get the url for that session first
   FString Url;
   if (!SessionInterface->GetResolvedConnectString(SESSION_NAME,
Url))
   {  return;}

       // The player controller travels to that url on that specific
       // session
      APlayerController* PlayerController =
                    GetFirstLocalPlayerController();
      PlayerController->ClientTravel(Url,
                   ETravelType::TRAVEL_Absolute);
  }
}
```

And that's it. With this, we should be able to connect through steam in 2 different accounts and join an existing session and wow. Play with a friend:

In order to test this properly, you need to export a build from Unreal and execute the game in 2 different machines with 2 different Steam accounts.

Remember this is just a test if you want to go full production with your project you need to get an App ID for your game.
But that doesn't mean you can have your own little game for you and your friends and play together.

. . .

Thanks for reading this **Dev-Blog**.
I hope you enjoyed reading it as much as I did when I wrote it.

You can find me in:

- @Blue Bubble Bee
- *@Cubenary*
- YouTube Channel

Programming C Programming Language Game Development Software Development

Software Engineering