

Section 3 - Steam Multiplayer

These are the slides that accompany the Unreal Multiplayer course.

Looking for something, try searching [our GitHub repo.](#)

Enjoy your stay!

*Sam & Ben
GameDev.tv*

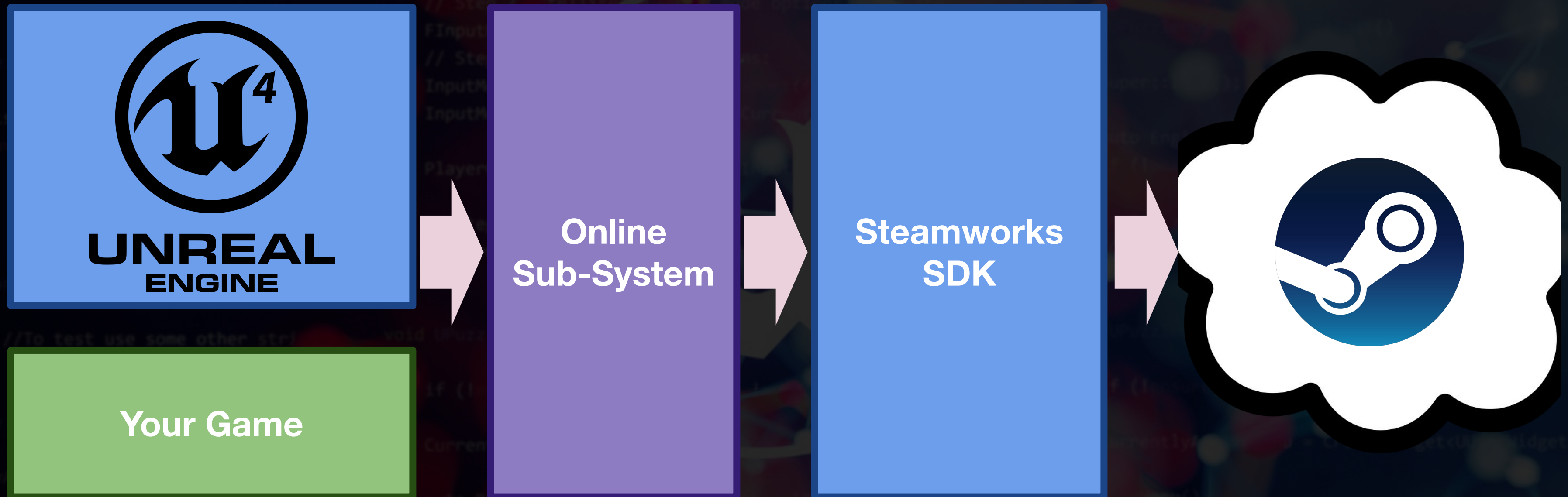
Live Google Slides at bit.ly/multiplayerslides



Introduction to Steam Multiplayer

Getting the Steamworks SDK

Unreal and Steam



Live Google Slides at bit.ly/multiplayerslides

Building SpaceWar In Visual Studio

Get It Running

- Make sure DirectX SDK is installed.
- Find the include and library paths.
- Add to Visual Studio.
- Build and run.



Building SpaceWar In XCode

Testing Steam Lobbies

Find A Test Partner

- Check in with your previous partner.
- Write a post on the forum.
- Respond to posts on the forum.

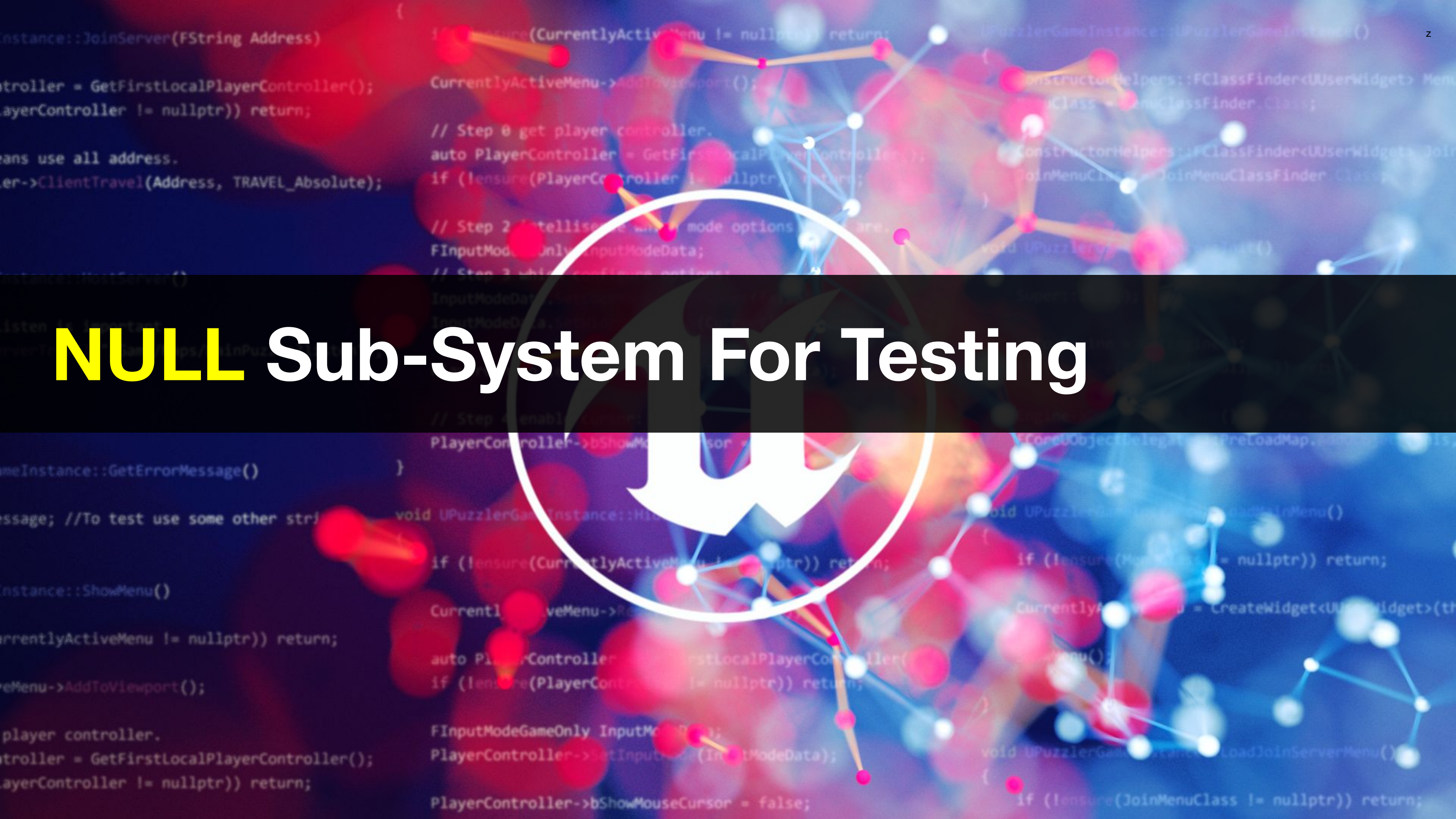


The Online Sub-System

Get The OSS Pointer

- Use the **Get()** static.
- Check for null.
- Print the result.





NULL Sub-System For Testing

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
// All players use all address.  
PlayerController->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
Listen is important
```

```
ServerToClientMap[Address] = UPuzzlerGameInstance::JoinServerMenuClass;
```

```
Instance::GetErrorMessage()
```

```
Message; //To test use some other string
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
Menu->AddToViewport();
```

```
Player controller.
```

```
Controller = GetFirstLocalPlayerController();
```

```
PlayerController != nullptr)) return;
```

```
if (ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisenet which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which window options
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
// Step 4 enable cursor
```

```
PlayerController->bShowMouseCursor = true;
```

```
}
```

```
void UPuzzlerGameInstance::HideMenu()
```

```
if (ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (ensure(PlayerController != nullptr)) return;
```

```
FInputModeGameOnly InputModeData;
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
ConstructorHelpers::FClassFinder<UserWidget> MenuClass;
```

```
MenuClass = MenuClassFinder.Class;
```

```
ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
void UPuzzlerGameInstance::LoadMainMenu()
```

```
{
```

```
if (ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
LoadJoinServerMenu();
```

```
}
```

```
CoreObjectDelegates::PreLoadMap.Add(this, &UPuzzlerGameInstance::PreLoadMap);
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

```
{
```

```
if (ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
LoadJoinServerMenu();
```

```
}
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

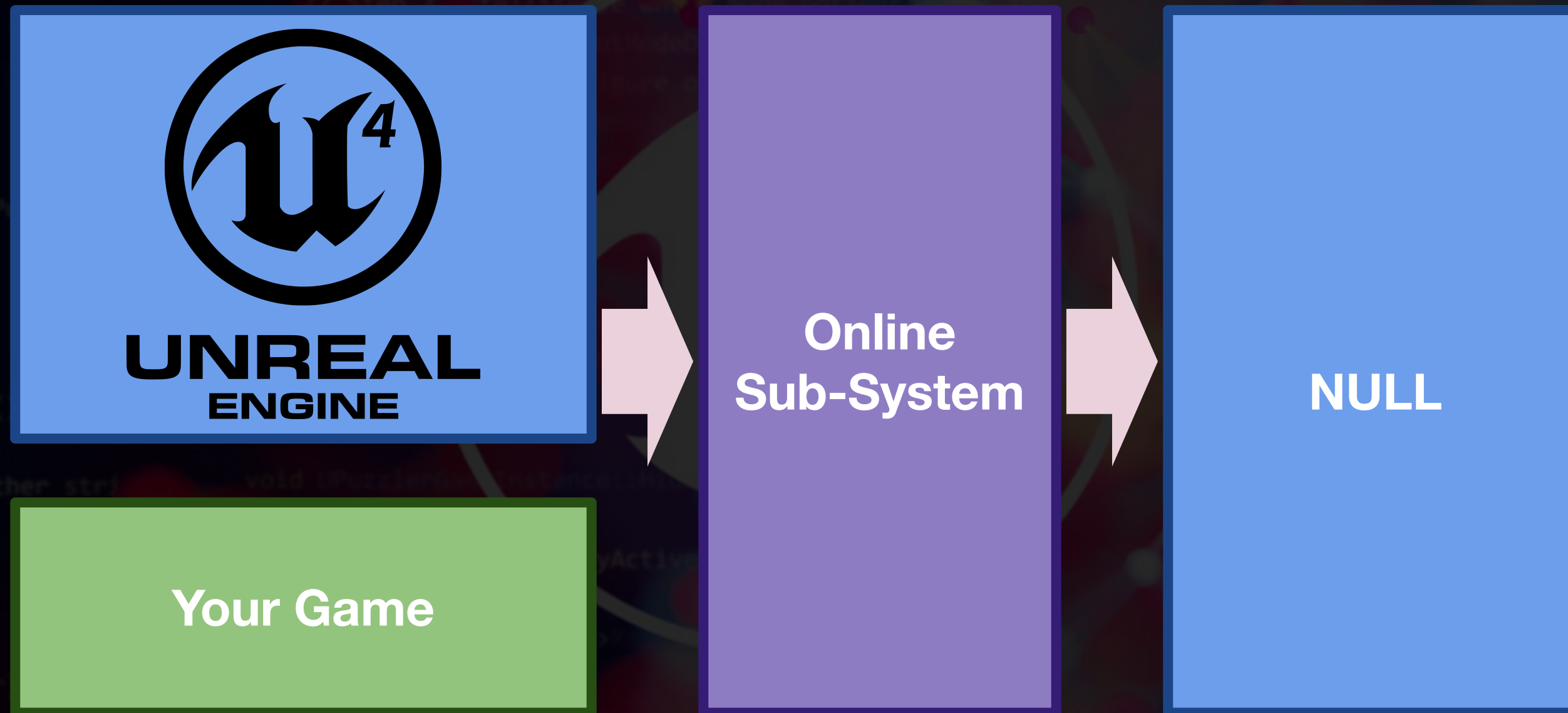
```
{
```

```
if (ensure(JoinMenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, JoinMenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

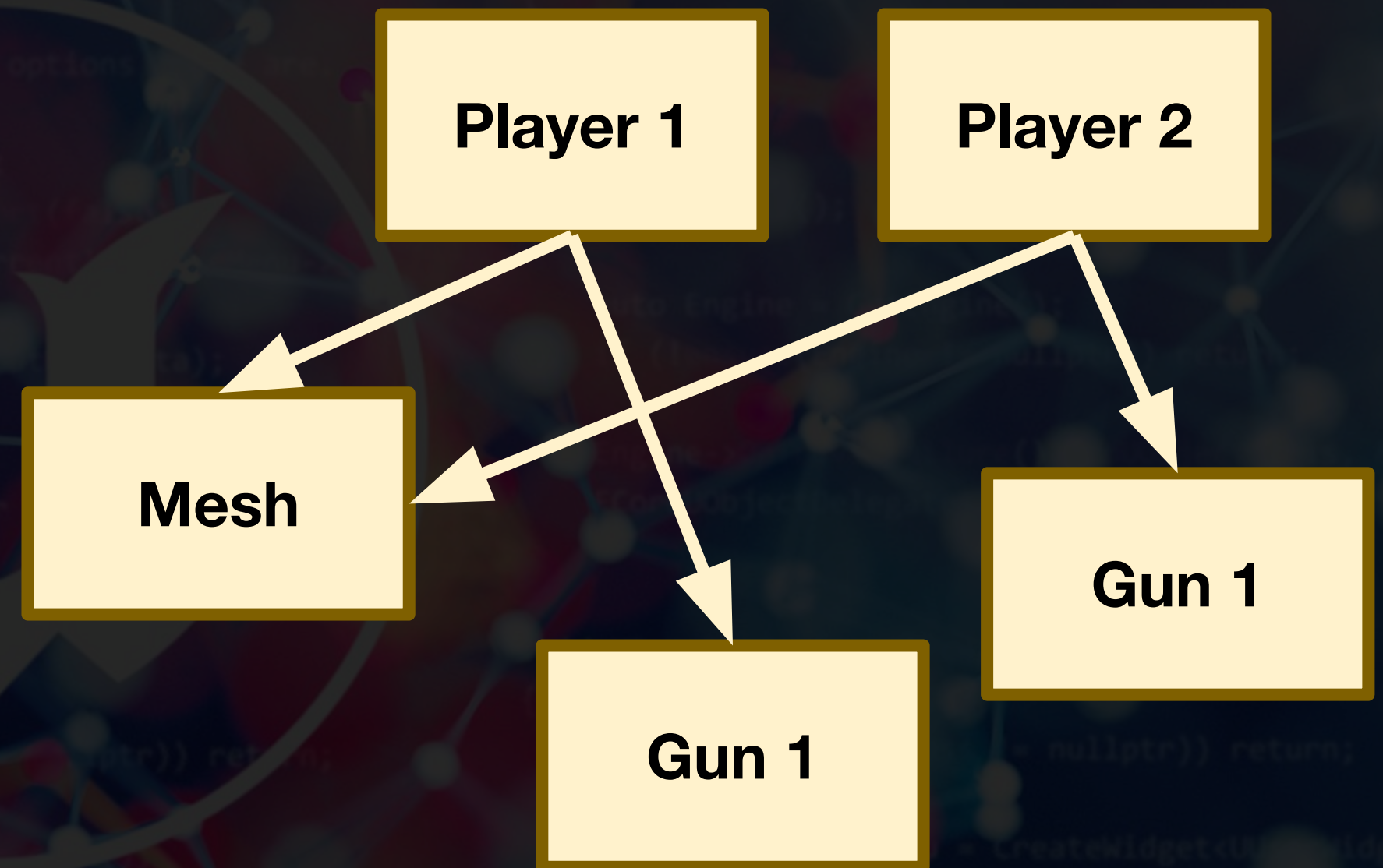
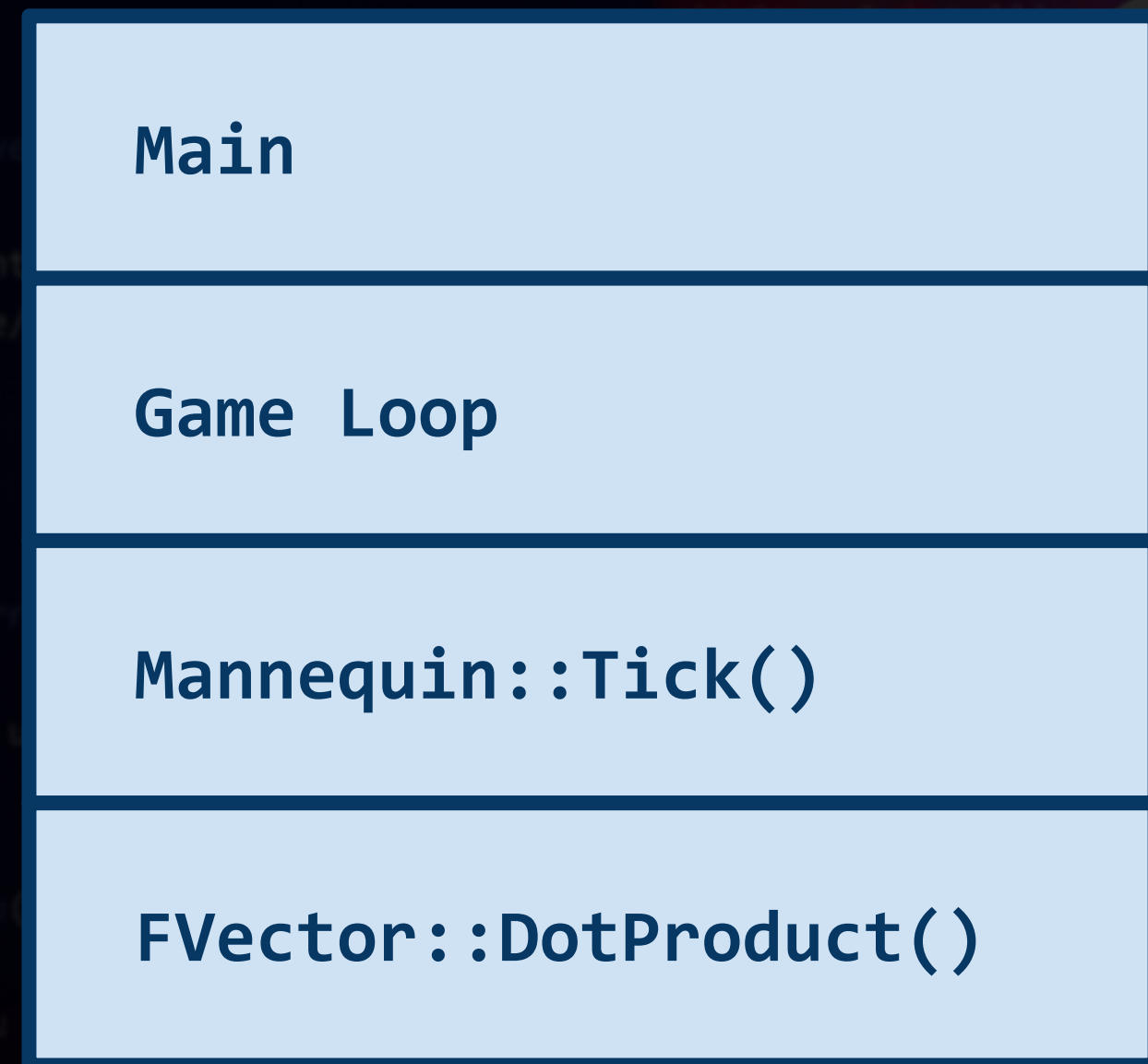

Testing The OSS



Live Google Slides at bit.ly/multiplayerslides

Memory Management In C++

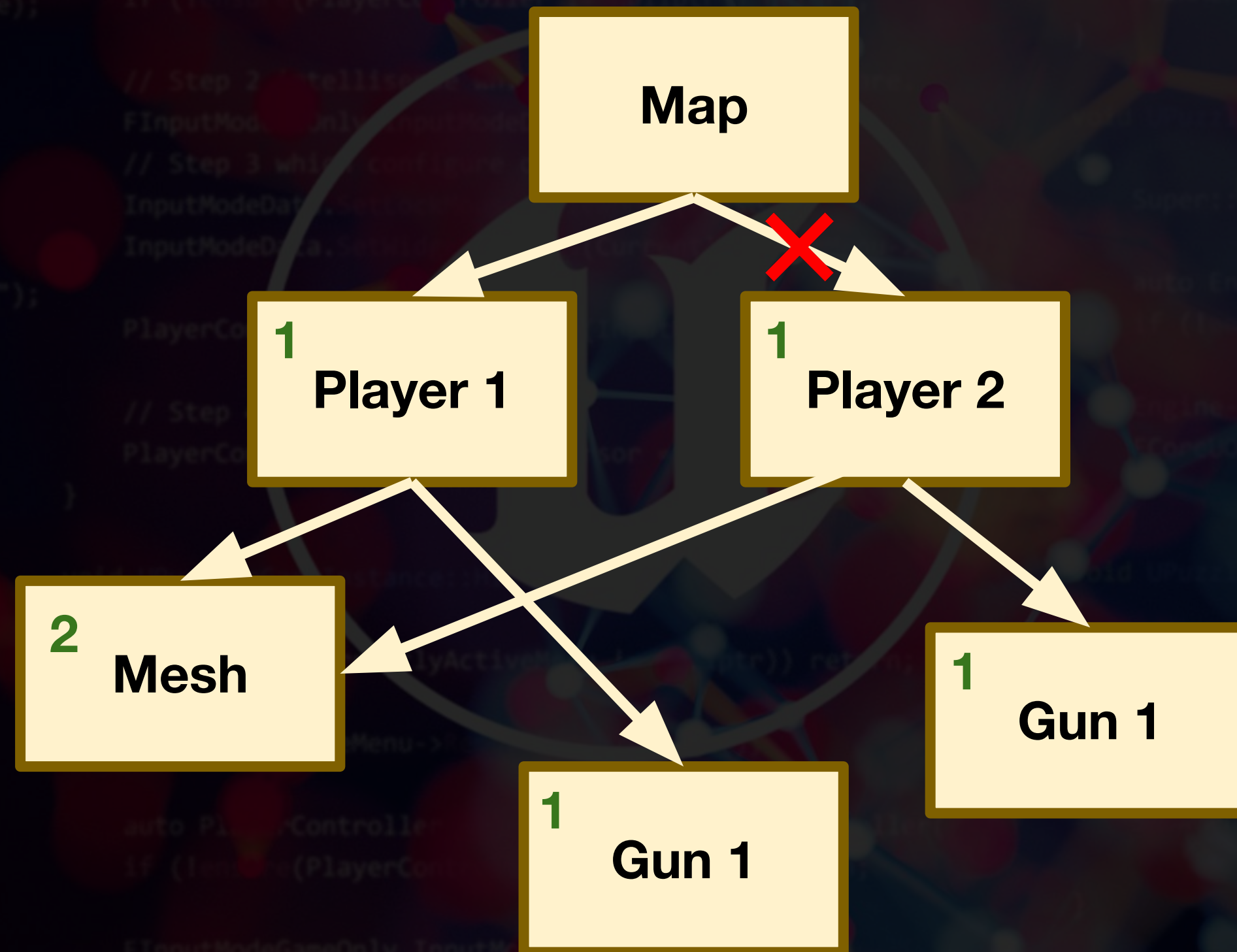
Stack vs Heap



Live Google Slides at bit.ly/multiplayerslides



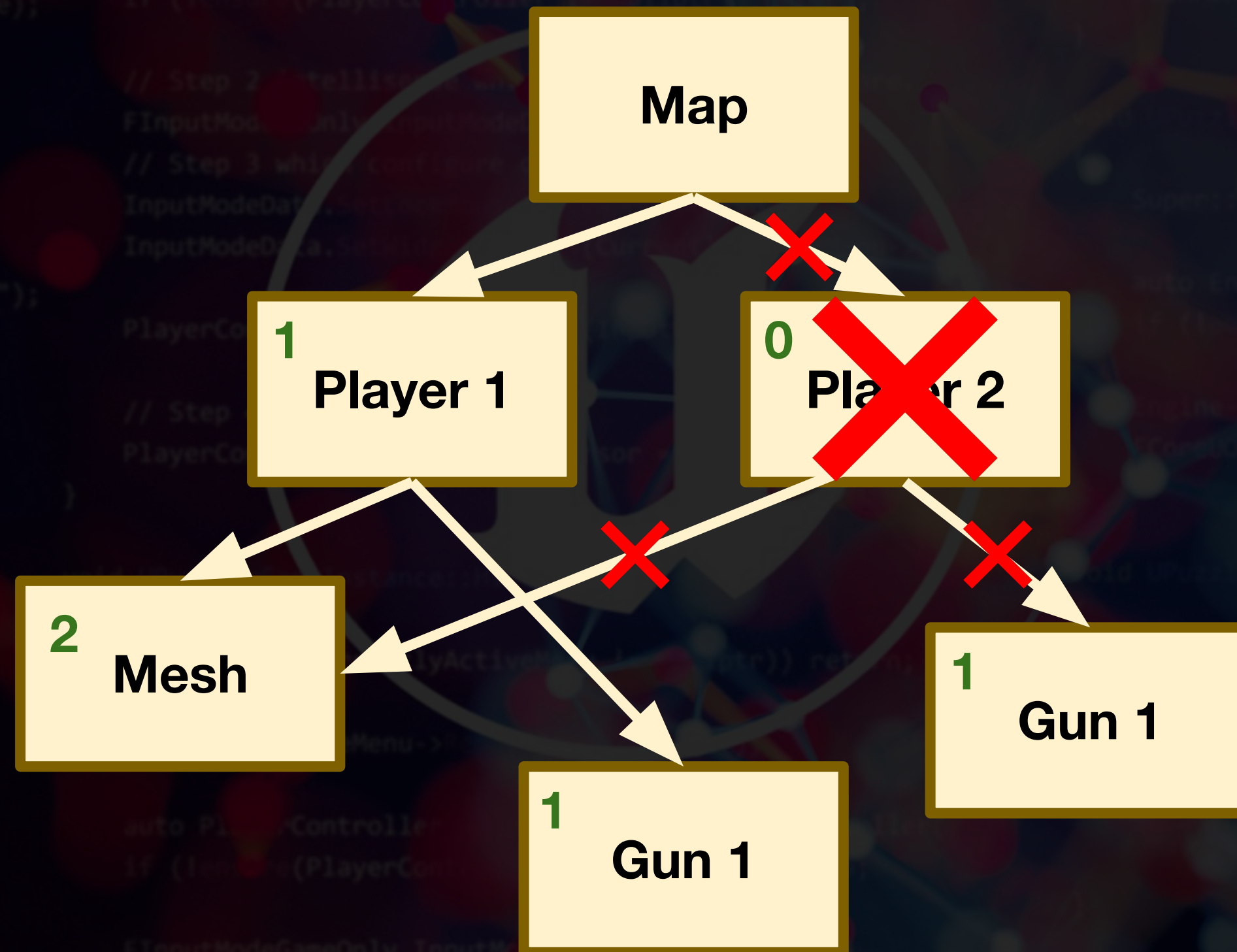
Reference Counting



Live Google Slides at bit.ly/multiplayerslides



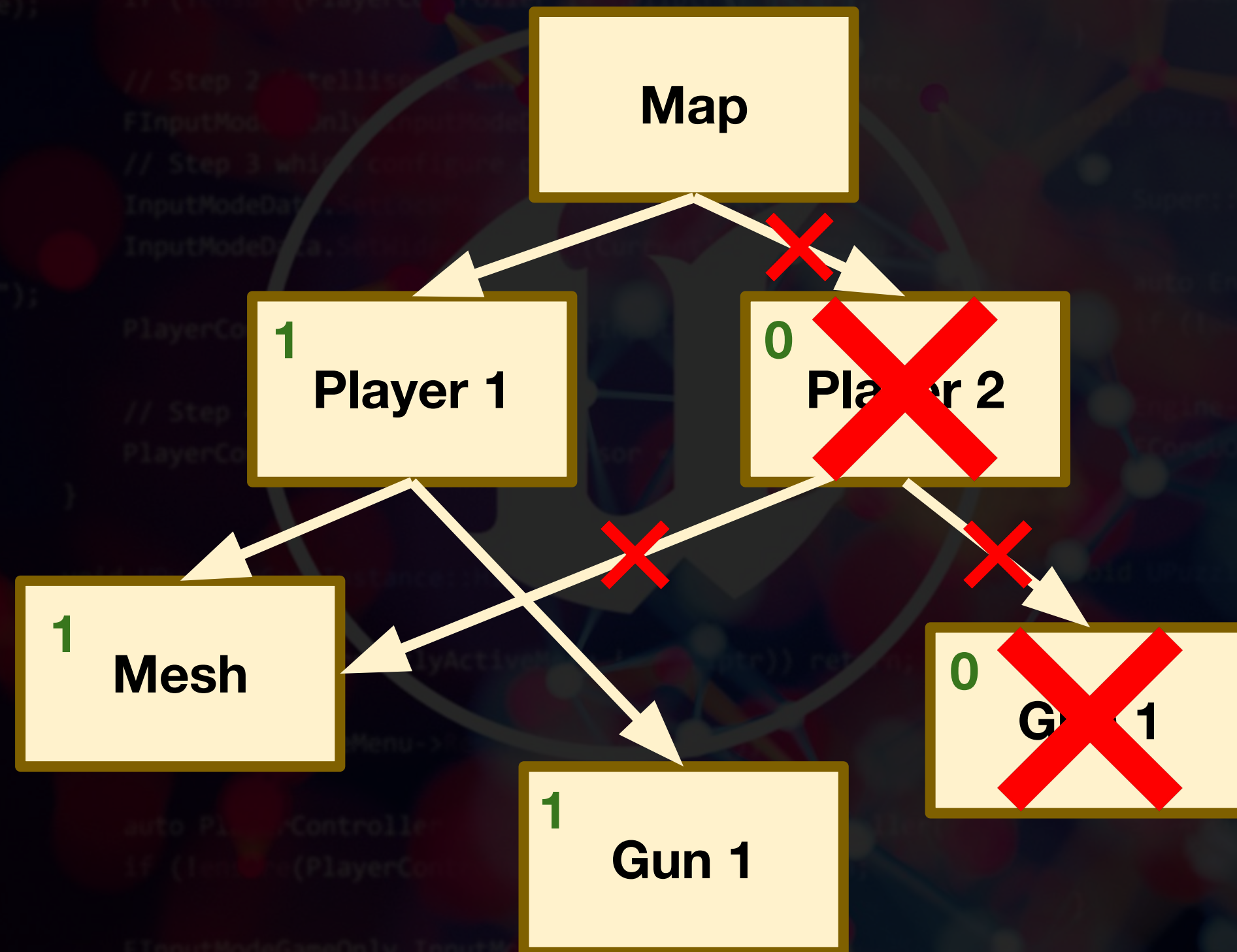
Reference Counting



Live Google Slides at bit.ly/multiplayerslides

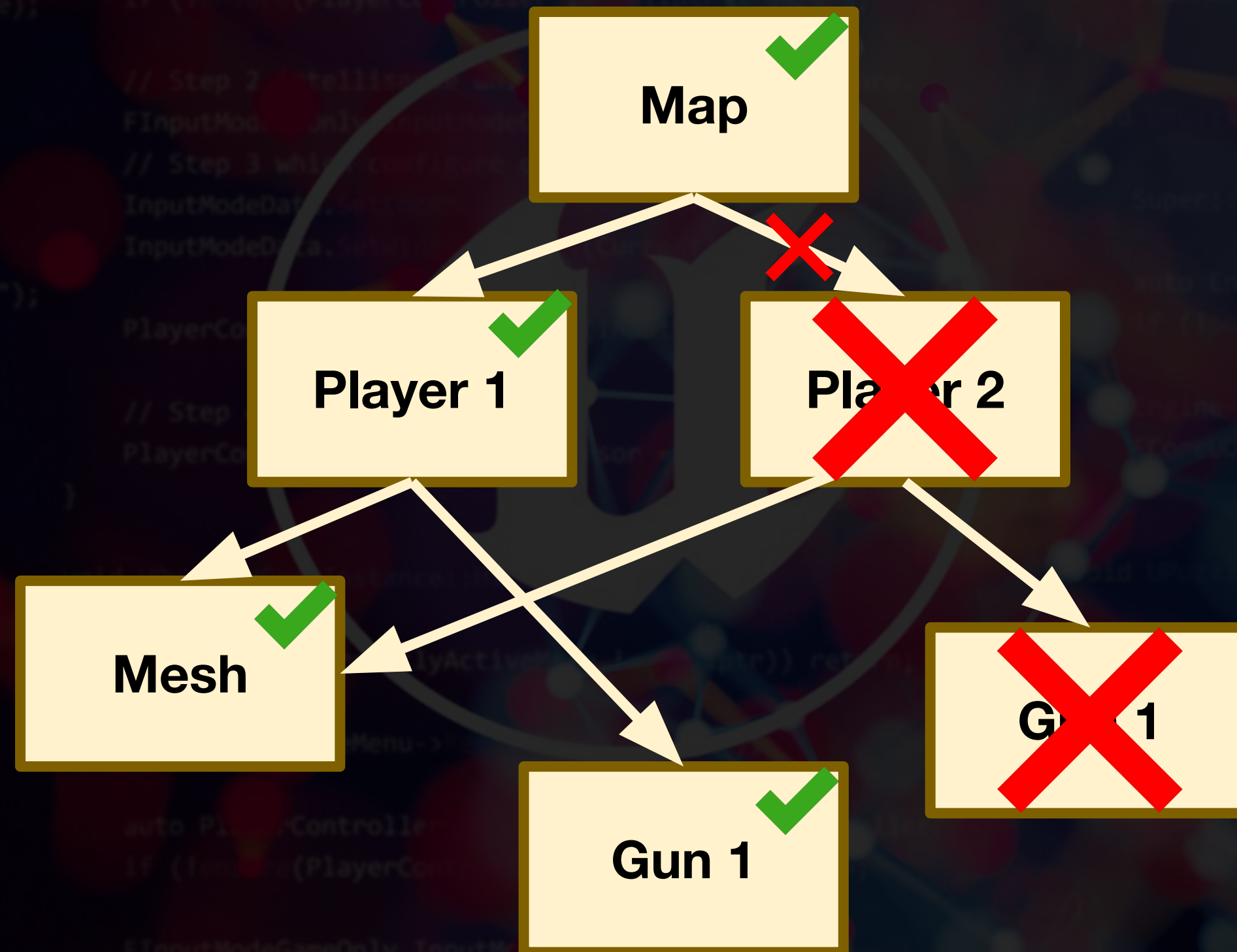


Reference Counting



Live Google Slides at bit.ly/multiplayerslides

Garbage Collection



Live Google Slides at bit.ly/multiplayerslides



How Does Unreal Do It?

1. How do we reference count an object?
2. How does Unreal know what objects to scan for garbage collection?
3. How does Unreal know which pointers it should follow?



Reference Counting In Unreal

- Use a **TSharedPtr<AActor>**
- Constructing increments the count
- Destructing decrements the count.



Garbage Collection in Unreal

- All **UObjects** are automatically in the “set”
- Unreal starts from the “root set”
- Unreal walks all the **UPROPERTY** pointer
- Any **UObject** not found can be deleted.



Creating Online Sessions

Create A Session On Host

- Use player zero.
- Use any name for the session.
- Use default settings for now.
- Only go into the game if we succeed.



Destroying Online Sessions

Finding Online Sessions

Print On Find Complete

- Find the type of the delegate.
- Register for callbacks.
- Print and test.



Query Parameters & Session Settings

Print The Sessions Found

- Found session are in the session search.
- Use the **TArray** docs for iterating.
- What happens if you use default settings.





Lists Of Widgets With ScrollBox

Populate The List

- Create a new C++ widget.
- Create the BP child.
- Create this widget from **MainMenu.cpp**.
- Add a few test ones as children.



Populating The Server List

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
rans use all address.  
er->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
listen is important.
```

```
ServerTo
```

```
meInstance::GetErrorMessage()
```

```
essage; //To test use some other str
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
reMenu->AddToViewport();
```

```
player controller.
```

```
Controller = GetFirstLocalPlayerController();
```

```
PlayerController != nullptr)) return;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisece which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which confides options;
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
// Step 4 enable cursor;
```

```
PlayerController->bShowMouseCursor = true;
```

```
PlayerController->bShowMouseCursor = true;
```

```
PlayerController->bShowMouseCursor = true;
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->bShowMouseCursor = false;
```

```
PlayerController->bShowMouseCursor = false;
```

```
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
ConstructorHelpers::FClassFinder<UserWidget> Menu
```

```
MenuClass = MenuClassFinder.Class;
```

```
ConstructorHelpers::FClassFinder<UserWidget> Join
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
void UPuzzlerGameInstance::HitMenu()
```


Requesting A Refresh

- Add to the MenuInterface.
- Trigger a FindSessions.
- When that returns, set the servers.



Selecting A Server

Set The **SelectedIndex**

- Handle the button callback
- Call the **SelectIndex()**
- Test the printed value.



Joining A Session

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
// All players use all address.  
PlayerController->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
Listen is important.
```

```
ServerTravel(Address);
```

```
Instance::GetErrorMessage()
```

```
Message; //To test use some other string
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
Menu->AddToViewport();
```

```
Player controller.
```

```
Controller = GetFirstLocalPlayerController();
```

```
PlayerController != nullptr)) return;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisenet which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which configures options.
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputMode(CurrentlyActiveMenu->GetInputMode());
```

```
// Step 4 enable cursor.
```

```
PlayerController->bShowMouseCursor = true;
```

```
}
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
FInputModeGameOnly InputModeData;
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
ConstructorHelpers::FClassFinder<UserWidget> MenuClass;
```

```
MenuClass = MenuClassFinder.Class;
```

```
ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
void UPuzzlerGameInstance::LoadMainMenu()
```

```
{
```

```
if (!ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

```
{
```

```
if (!ensure(JoinMenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, JoinMenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
void UPuzzlerGameInstance::LoadPreLoadMapMenu()
```

```
{
```

```
if (!ensure(PreloadMenuClass != nullptr)) return;
```

```
PreloadMenu = CreateWidget<UserWidget>(this, PreloadMenuClass);
```

```
PreloadMenu->AddToViewport();
```

```
void UPuzzlerGameInstance::LoadMainMenu()
```

```
{
```

```
if (!ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

```
{
```

```
if (!ensure(JoinMenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, JoinMenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
void UPuzzlerGameInstance::LoadPreLoadMapMenu()
```

```
{
```


Join A Session

- Handle the join callback.
- Get the connect string.
- Do a **ClientTravel**.
- Test!



Enabling The Steam OSS

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
// All players use all address.  
PlayerController->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
Listen is important.
```

```
ServerToClient(FString Message);
```

```
Instance::GetErrorMessage()
```

```
Message; //To test use some other string
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
Menu->AddToViewport();
```

```
Player controller.
```

```
Controller = GetFirstLocalPlayerController();
```

```
PlayerController != nullptr)) return;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisenet which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which configures options.
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
// Step 4 enable cursor.
```

```
PlayerController->bShowMouseCursor = true;
```

```
PlayerController->bShowMouseCursor = true;
```

```
PlayerController->bShowMouseCursor = true;
```

```
PlayerController->bShowMouseCursor = true;
```

```
PlayerController->bShowMouseCursor = true;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->SetInputMode(FInputModeGameOnly);
```

```
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```


Debug With A Pal

- Go and request a testing buddy.
- Share your repo with them.
- Take it in turns.
- Investigate the logs.
- HINT: The issue is in finding the session.



Row Selection In Lists

Set The Row Colour

- Choose the colours you want
- Consider the states: **Hovered** and **Selected**
- How many colours will you need?



Displaying Search Result Properties

Update The UI

- Fill out the rest of the data.
- Bind the necessary components.
- Set the components from the struct.



Debugging The Search Results

Layout The Text

- Equal spacing in each row
- No overlapping
- Add row headers.



Custom Session Settings

Set A Custom Server Name

- Model the UI from the join menu.
- Need to enter name in a text box.
- Send the text value up to create session.



Continued: Custom Session Settings

GameMode And Multiplayer

Counting The Players

- Implement **PostLogin** and **Logout**
- Use these methods to count the players
- Log a message when you have 3.
- Where is the **GameMode**?



Enabling Seamless Travel

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
// Ensures all address.  
PlayerController->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
Listen is important.
```

```
ServerToClient(FString Message)
```

```
Instance::GetErrorMessage()
```

```
Message; //To test use some other string
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
Menu->AddToViewport();
```

```
Player controller.
```

```
Controller = GetFirstLocalPlayerController();
```

```
PlayerController != nullptr)) return;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisenet which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which configures options.
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
// Step 4 enable cursor.
```

```
PlayerController->bShowMouseCursor = true;
```

```
}
```

```
void UPuzzlerGameInstance::HideMenu()
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
FInputModeGameOnly InputModeData;
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
ConstructorHelpers::FClassFinder<UserWidget> MenuClass;
```

```
MenuClass = MenuClassFinder.Class;
```

```
ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
void UPuzzlerGameInstance::LoadMainMenu()
```

```
{
```

```
Super::LoadMainMenu();
```

```
auto Engine = GetEngine();
```

```
if (!ensure(Engine != nullptr)) return;
```

```
Engine->SetInputMode(FInputModeGameOnly);
```

```
CoreObjectDelegates::PreLoadMap.AddStatic(this, &UPuzzlerGameInstance::PreLoadMap);
```

```
void UPuzzlerGameInstance::PreLoadMap()
```

```
{
```

```
if (!ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
LoadMainMenu();
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

```
{
```

```
if (!ensure(JoinMenuClass != nullptr)) return;
```


Add A Transition Map

- Read the documentation
- Add a custom transition map
- Display “Loading...” text.



Debugging Engine Code

Debug The NULL Subsystem

- Does host or client call **RegisterPlayer**?
- Does the **NumOpenPublicConnections** decrement?
- If not, why not?



Starting A Session

```
Instance::JoinServer(FString Address)  
  
Controller = GetFirstLocalPlayerController();  
PlayerController != nullptr)) return;  
  
// All players use all address.  
PlayerController->ClientTravel(Address, TRAVEL_Absolute);
```

```
Instance::HostServer()
```

```
Listen is important.
```

```
ServerToClientMessage
```

```
Instance::GetErrorMessage()
```

```
Message; //To test use some other string
```

```
Instance::ShowMenu()
```

```
CurrentlyActiveMenu != nullptr)) return;
```

```
Menu->AddToViewport();
```

```
Player controller.
```

```
Controller = GetFirstLocalPlayerController();
```

```
PlayerController != nullptr)) return;
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
// Step 0 get player controller.
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
// Step 2 Intellisenet which mode options are.
```

```
FInputModeGameOnly InputModeData;
```

```
// Step 3 which window options
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
InputModeData.SetInputMode(FInputModeGameOnly);
```

```
// Step 4 enable cursor
```

```
PlayerController->bShowMouseCursor = true;
```

```
}
```

```
void UPuzzlerGameInstance::HitMenu()
```

```
if (!ensure(CurrentlyActiveMenu != nullptr)) return;
```

```
CurrentlyActiveMenu->RemoveFromParent();
```

```
auto PlayerController = GetFirstLocalPlayerController();
```

```
if (!ensure(PlayerController != nullptr)) return;
```

```
FInputModeGameOnly InputModeData;
```

```
PlayerController->SetInputMode(InputModeData);
```

```
PlayerController->bShowMouseCursor = false;
```

```
UPuzzlerGameInstance::UPuzzlerGameInstance()
```

```
ConstructorHelpers::FClassFinder<UserWidget> MenuClass;
```

```
MenuClass = MenuClassFinder.Class;
```

```
ConstructorHelpers::FClassFinder<UserWidget> JoinMenuClass;
```

```
JoinMenuClass = JoinMenuClassFinder.Class;
```

```
void UPuzzlerGameInstance::LoadMainMenu()
```

```
{
```

```
if (!ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
LoadJoinServerMenu();
```

```
}
```

```
CoreObjectDelegates::PreLoadMap.AddStatic(this, &UPuzzlerGameInstance::PreLoadMap);
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

```
{
```

```
if (!ensure(MenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, MenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
LoadMainMenu();
```

```
}
```

```
void UPuzzlerGameInstance::LoadJoinServerMenu()
```

```
{
```

```
if (!ensure(JoinMenuClass != nullptr)) return;
```

```
CurrentlyActiveMenu = CreateWidget<UserWidget>(this, JoinMenuClass);
```

```
CurrentlyActiveMenu->AddToViewport();
```

```
}
```


Start The Game On Timer

- Use the timer manager
- Wait for **X** players to join
- Get the game starting after **Y** seconds
- Call **StartSession** on the **SessionInterface**
- Make sure others can't join.



Handling Network Errors

Steam Multiplayer Wrap-up