Multiplayer Lobby System

Documentation

1. Introduction

Overview

The **Multiplayer Lobby System** is a modular, lightweight, and easy to use framework for creating multiplayer experiences in Unreal Engine. It provides a complete multiplayer flow, from the **Main Menu** to the **Lobby**, and finally into the **Gameplay** level, allowing developers to quickly integrate multiplayer functionality into their projects with minimal setup.

The system includes built-in **Host** and **Join Session** functionality, allowing players to easily create or join multiplayer sessions through an intuitive interface. Once all players have joined the lobby, non-host players can use the **Ready Up** system to indicate they are ready to begin the match. The host can start the game once all connected players are ready or, if enabled, force the match to start regardless of player readiness, ensuring all players remain synchronized before transitioning to the gameplay level.

To provide a smooth multiplayer experience, the system uses **Seamless Server Travel** to transition players from the lobby to the gameplay level while preserving player connections.

A fully modular **Multiplayer Chat System** is also included. The chat system is independent of the lobby and can be easily integrated into any multiplayer level or project, making it suitable for both lobby communication and in game messaging.

The lobby also includes:

- Host and Join Session functionality.
- Ready Up system for all non-host players.
- Host controls to kick players from the lobby.
- Replicated game settings that automatically synchronize to every connected player and are carried over to the gameplay level.
- A **Profile Menu** that allows players to customize their display name, which is synchronized across the multiplayer session.

Designed to be simple, flexible, and modular, the Multiplayer Lobby System can be easily customized and integrated into new or existing Unreal Engine projects, providing a solid foundation for multiplayer games with minimal setup.

2. Project Structure

The Multiplayer Lobby System is organized into three main levels, each representing a different stage of the multiplayer flow. Every level contains its own set of Blueprints responsible for handling the functionality specific to that stage. This separation keeps the project modular, organized, and easy to understand.

While the project includes several helper and utility Blueprints for smaller tasks, the majority of the core multiplayer logic is implemented within the **Lobby** Blueprints. These Blueprints manage the overall multiplayer flow, including player synchronization, session management, lobby state, game settings, ready status, and the transition into gameplay.

The project is divided into the following sections:

- **Main Menu** – Handles session hosting, session joining, profile management, and navigation to the lobby.
- **Lobby** – Contains the core multiplayer logic, including player management, Ready Up, replicated game settings, player synchronization, chat, kicking players, and seamless travel to the gameplay level.
- **Gameplay** – Receives the replicated game settings from the lobby and begins the multiplayer match using the synchronized data.

3. Blueprint Overview

3.1 Main Menu

Blueprint Locations	Content/MultiplayerLobbySystem/Blueprints/Game/MainMenu , Content/MultiplayerLobbySystem/Blueprints/Player
UI Location	Content/MultiplayerLobbySystem/Blueprints/UserInterface/Widget/MainMenu

Overview

The **Main Menu** Blueprints are the entry point of the Multiplayer Lobby System. It is responsible for displaying the Main Menu user interface when the level is loaded and performing the minimal initialization required for the menu.

Unlike the Lobby Blueprints, the Main Menu Blueprint intentionally contains very little gameplay logic. Most multiplayer systems, including session management, player synchronization, and lobby functionality, are implemented within dedicated UI widgets and the Lobby Blueprints, keeping the Main Menu simple and easy to maintain.

Responsibilities

- Creates and displays the Main Menu UI.
- Performs the minimal setup required for the Main Menu level.
- Displays network error messages received from the Player Character.

Notes

- The Main Menu Blueprints are intentionally lightweight.
- Most multiplayer functionality is handled by the Lobby Blueprints.
- The network error message widget is created by the **Main Menu Player Character**. Any network error information is received from the **Game Instance**, which passes the required data to the Player Character for displaying the appropriate message to the player.

Blueprint Index

Blueprint	Purpose
BP_MainMenuGameMode , BP_MainMenuGameState , BP_MainMenuPlayerState	Default Unreal Engine gameplay framework Blueprints used by the Main Menu level. These Blueprints contain no custom logic and exist primarily to define the level's Game Mode, Game State, and Player State. They can be extended with custom functionality if needed.
BP_MainMenuPlayerController	Creates and displays the Main Menu widget when the level starts.
WBP_MainMenu, WBP_JoinSession, WBP_HostSession	Main Menu user interface containing the logic for hosting sessions, searching for sessions, joining sessions, and managing the main menu workflow.
WBP_SessionSearchResult	Displays the information for a single session in the session search results list and handles interaction with that session entry.

3.2 Lobby

Blueprint Locations	Content/MultiplayerLobbySystem/Blueprints/Game/Lobby, Content/MultiplayerLobbySystem/Blueprints/Player
UI Location	Content/MultiplayerLobbySystem/Blueprints/UserInterface/Widget/Lobby

Overview

The **Lobby Blueprints** are the core of the Multiplayer Lobby System and contain the majority of the project's multiplayer functionality. Most of the systems, including player synchronization, session flow, replicated game settings, Ready Up, multiplayer chat, player management, and seamless travel, are implemented within these Blueprints.

The Lobby Blueprints are divided into **two parts**:

- **Core Blueprints** – Contain all the underlying logic for the Multiplayer Lobby System. These Blueprints are **abstract** and are not intended to be placed directly in a level or assigned as gameplay framework classes. They implement the complete multiplayer framework used by the system.

- **Child Blueprints** – Concrete implementations of the Core Blueprints that are assigned to the level and gameplay framework (Game Mode, Game State, Player Controller, etc.). These Blueprints are intentionally lightweight and primarily exist to expose the Core Blueprints for use within the project. They can also be extended with custom logic to suit the needs of your game.

When overriding Blueprint events in child classes, always **call the parent implementation** for the required events. The **parent implementation** contains the **core multiplayer** logic used by the system, and failing to call it may prevent certain features or networking functionality from working correctly.

Most customization should be performed within the child Blueprints, allowing the core multiplayer framework to remain unchanged while providing flexibility for project-specific features and integrations.

Responsibilities

- Manage the complete multiplayer lobby workflow.
- Synchronize player information across all connected clients.
- Handle player joining and leaving the lobby.
- Manage the Ready Up system and validate player readiness.
- Allow the host to start the match when all players are ready.
- Manage and replicate gameplay settings to all connected players.

- Transfer gameplay settings from the lobby to the gameplay level.
- Handle seamless server travel from the lobby to the gameplay level.
- Manage multiplayer chat and player communication.
- Provide the core framework that all lobby related gameplay classes inherit from.

Notes

- The **Core Lobby Blueprints** are **abstract** and are **not intended to be used directly** (Located in **Content/MultiplayerLobbySystem/Blueprints/Game/Lobby/Core**).
- Always use the provided **child Blueprints** when assigning Game Mode, Game State, Player Controller, Player State, and other gameplay framework classes (Located in **Content/MultiplayerLobbySystem/Blueprints/Game/Lobby**).
- When overriding Blueprint events in the child Blueprints, **always call the Parent implementation**. Failing to do so will prevent the core multiplayer logic from executing correctly.
- The child Blueprints are intentionally lightweight and are the recommended place for project specific customization and integration.
- Most systems within the Multiplayer Lobby System depend on the Lobby Blueprints. Modifying the Core Blueprints directly is not recommended unless you intend to change the underlying framework behaviour.

Blueprint Index

Blueprint	Purpose
BP_LobbyGameModeCore	**Abstract core Game Mode** for the Multiplayer Lobby System. Contains the core Game Mode implementation and enables **Seamless Travel**, allowing players to transition from the Lobby to the Gameplay level while preserving their network connections. Intended to be inherited rather than used directly.
BP_LobbyGameStateCore	**Abstract core Game State** responsible for storing and replicating the gameplay settings structure to all connected players. It also updates the replicated game settings in the **Game Instance** on the server, allowing the settings to persist across seamless level transitions and be accessed throughout the multiplayer flow. Intended to be inherited rather than used directly.
BP_LobbyPlayerControllerCore	**Abstract core Player Controller** that contains the primary multiplayer controller logic. It routes server RPCs,

	synchronizes player display names, identifies the server host, manages player kick requests, processes the Ready Up system, and routes multicast events such as displaying the loading screen. This Blueprint is used by both the Lobby and Gameplay frameworks to support seamless travel. Intended to be inherited rather than used directly.
BP_LobbyPlayerStateCore	**Abstract core Player State** responsible for managing and replicating player-specific data, including the player's display name, Ready state, and Host status. It also contains the **Multiplayer Chat Component**, which handles sending, receiving, and replicating chat messages as part of the modular Multiplayer Chat System. Additionally, it executes player related multicast events and copies replicated player data to the Gameplay Player State during seamless travel, ensuring player information remains synchronized across levels. Intended to be inherited rather than used directly.
BP_LobbyGameMode, BP_LobbyGameState, BP_LobbyPlayerController, BP_LobbyPlayerState	Concrete implementations of the abstract core Blueprints. They contain the required parent function calls, are mostly empty by default, and are intended for project-specific customization and integration.
BP_LobbyPlayerCharacter	Lobby player character used primarily for UI management. Creates the Lobby HUD and Pause Menu widgets, manages input focus for the chat panel, and handles UI-related interactions. No multiplayer gameplay logic is implemented in this Blueprint
WBP_LobbyHud	Main lobby user interface containing all lobby functionality. Displays the player list, multiplayer chat, game settings panel, and controls for player Ready Up and starting the match. It also allows the host to modify game settings, kick players, and manage the lobby while synchronizing the displayed information for all connected players.
WBP_LobbyPlayerList	Displays information for an individual player within the lobby player list, including the player's display name, host status, and Ready state. Updates automatically as replicated player data changes.
WBP_LobbySettingPallet	Modular widget used to display and organize gameplay setting controls. Designed to allow developers to easily add, remove, or customize game settings within the lobby interface.

3.3 Gameplay

Blueprint Locations	Content/MultiplayerLobbySystem/Blueprints/Game/ Gameplay, Content/MultiplayerLobbySystem/Blueprints/Player

Overview

The **Gameplay** Blueprints provide the foundation for the gameplay level after the transition from the lobby. They intentionally contain minimal built-in logic, serving as a starting point that can be customized to fit your project's needs.

Their primary responsibilities are to initialize the gameplay UI and retrieve the replicated game settings transferred from the lobby through the **Game Instance**. Aside from this, no major gameplay systems are implemented, allowing the gameplay framework to be freely modified and extended.

Responsibilities

- Initialize the gameplay user interface.
- Retrieve the replicated game settings from the Game Instance.
- Apply the transferred gameplay settings as needed.
- Provide a base gameplay framework that can be extended with custom game logic.

Notes

- The Gameplay Blueprints are intentionally lightweight and contain only the functionality required to support the Multiplayer Lobby System.
- The gameplay framework is intended to be customized and extended based on project's requirements.
- The gameplay level must use **BP_LobbyPlayerController** or a child Blueprint that inherits from it. The same Player Controller class must be assigned to both the Lobby and Gameplay levels to maintain compatibility with the Lobby System and ensure Seamless Travel functions correctly.
- All other Gameplay Blueprints can be modified, replaced, or extended as needed without affecting the core Lobby System, provided the required Player Controller inheritance is preserved.

3.4 Other Core blueprints

Blueprint Locations	Content/MultiplayerLobbySystem/Blueprints/Game/GameInstance, Content/MultiplayerLobbySystem/Blueprints/Components
UI Location	Content/MultiplayerLobbySystem/Blueprints/UserInterface/Widget/ChatSystem

Overview

The **Other Core Blueprints** provide functionality that is shared across the entire Multiplayer Lobby System and is not specific to the Main Menu, Lobby, or Gameplay levels. These Blueprints contain reusable systems that support the overall multiplayer framework, allowing data and functionality to persist across level transitions.

The provided **Game Instance** is responsible for managing global multiplayer data that must persist throughout the lifetime of the game. It also includes the **Chat System Component**, allowing the chat system to remain available and function seamlessly across all levels.

Responsibilities

- Handle network and travel error events.
- Store multiplayer data that persists across level transitions.
- Store and provide access to replicated gameplay settings.
- Maintain session-related information that is shared between levels.
- Provide modular gameplay components that can be reused independently.
- Support multiplayer functionality across the Lobby, and Gameplay levels.

Notes

- The **Game Instance** persists for the duration of the game and is used to store data that must remain available across multiple levels.
- Gameplay settings received from the Lobby are stored in the Game Instance on the server and later accessed by the Gameplay level after seamless travel.
- The **Multiplayer Chat Component** is completely modular and can be attached to any **Player State** to provide multiplayer chat functionality.
- The **Multiplayer Chat Component** is independent of the Lobby System and can be attached to any Player State. Combined with the provided **Chat Widget Panel**, it forms a fully modular multiplayer chat system that can be integrated into any multiplayer level or project.

- These Blueprints are shared systems that support the entire Multiplayer Lobby framework rather than a specific level.

Blueprint Index

Blueprint	Purpose
BP_MultiplayerLobbyGameInstanceCore	**Abstract core Game Instance** responsible for handling network and travel error events, storing the maximum player count for the hosted session (due to Blueprint session limitations), and maintaining the replicated gameplay settings received from the Lobby. These settings persist across level transitions and are retrieved by the Gameplay level after seamless travel. Intended to be inherited rather than used directly.
BP_MultiplayerLobbyGameInstance	Child implementation of **BP_MultiplayerLobbyGameInstanceCore** used by the project. This Blueprint is intentionally lightweight and can be extended with project-specific logic while preserving the functionality provided by the core Game Instance.
BPAC_MultiplayerChat	Modular Actor Component that implements the Multiplayer Chat System. It handles sending, receiving, and replicating chat messages between players. The component is attached to the **Player State**, allowing chat functionality to replicate correctly and persist across seamless travel. It can also be attached to any custom Player State, making the chat system reusable in any multiplayer level independently of the Lobby System.

3.5 Utils/Helper Blueprints

Blueprint Locations	Content/MultiplayerLobbySystem/Blueprints/SaveGame, Content/MultiplayerLobbySystem/Blueprints/Utils
UI Location	Content/MultiplayerLobbySystem/Blueprints/UserInterface/Widget/Utils

Overview

The **Utils/Helper Blueprints** contain reusable utility systems and supporting Blueprints used throughout the Multiplayer Lobby System. These Blueprints are responsible for common functionality that is not directly tied to the Main Menu, Lobby, or Gameplay levels, helping keep the core multiplayer framework modular and organized.

This category includes utility widgets, helper functions, Save Game functionality, and miscellaneous systems used across the project. It also contains the **Pause Menu**, **Profile Menu**, **Loading Screen**, and other supporting UI widgets, along with Blueprint Function Libraries that provide reusable functionality for Save Game management, loading screen handling, and other common operations.

Responsibilities

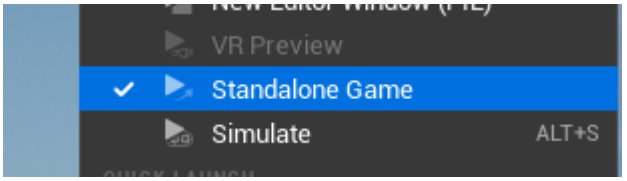
- Manage Save Game creation, loading, and updating.
- Store player preferences and persistent data.
- Provide reusable Blueprint Function Libraries.
- Handle loading screen functionality.
- Provide the Pause Menu and Profile Menu UI.
- Contain miscellaneous helper widgets used throughout the project.
- Support common UI and utility functionality shared across multiple systems.

Notes

- These Blueprints are independent utility systems and are not major part of the core Lobby framework.
- Most helper Blueprints are reusable and can be integrated into other projects with minimal modifications.
- The utility widgets are designed to support the Main Menu, Lobby, and Gameplay systems while remaining modular and easy to customize.
- Blueprint Function Libraries centralize commonly used functionality, reducing duplicated logic throughout the project.

Usage Guidelines

- Ensure replication is enabled for any custom variables or actors that need to synchronize between the server and clients or be visible to other connected players.
- Avoid testing multiplayer using **Play In Editor (PIE),** as Unreal Engine does not support **Seamless Travel** correctly in PIE sessions.
- Always test multiplayer using **Standalone Game** or a **packaged build** to ensure Seamless Travel and networking features function as expected.



- Use multiple Standalone Game instances, separate machines, or multiple instances of the packaged executable when validating multiplayer functionality.
- Perform final testing in a packaged build, as it most accurately represents the behavior of the shipped game.

Support & Customization:

This pack is designed to be a “living” asset and will continue to evolve over time. If you encounter any issues or have questions regarding implementation, please feel free to reach out through our support link or Discord server.

For faster support, we recommend joining our Discord server. Our team is ready to help and answer any questions you may have.

Level up your game's feel and performance today!

⭐⭐⭐⭐⭐ `Your Review means the world to us!`
Great feedback comes back as even better features.
If there are any suggestions or ideas for improvement, feel free to contact us. If possible, we will include your request in future updates. 📰 Follow us for updates and news!
Youtube | Facebook | Email | Linkedin
Website: 300mind.studio

Thank you for choosing the Multiplayer Lobby System. We hope this documentation helps you. Happy Gaming!

Enjoy building your game! 🎮

THANK YOU