Travelling in Multiplayer

An overview of how travelling works in multiplayer.

Seamless and non-seamless travel

In **Unreal Engine (UE)**, there are two main ways to travel: **seamless** and **non-seamless**. The main difference, is that **seamless** travel is a non-blocking operation, while **non-seamless** will be a blocking call.

When a client executes a non-seamless travel, the client will disconnect from the server and then re-connect to the same server, which will have the new map ready to load.

It is recommended that Unreal Engine multiplayer games use seamless travel when possible. It will generally result in a smoother experience, and will avoid any issues that can occur during the reconnection process.

There are three ways in which a non-seamless travel must occur:

- When loading a map for the first time
- When connecting to a server for the first time as a client
- When you want to end a multiplayer game, and start a new one

There are three main function that drive travelling: `UEngine::Browse`, `UWorld::ServerTravel`, and `APlayerController::ClientTravel`. These can be a bit confusing when trying to figure out which one to use, so here are some guidelines that should help:

`UEngine::Browse`

- Is like a hard reset when loading a new map.
- Will always result in a non-seamless travel.
- Will result in the server disconnecting current clients before travelling to the destination map.
- Clients will disconnect from current server.
- Dedicated server cannot travel to other servers, so the map must be local (cannot be URL).

`UWorld::ServerTravel`

- For the server only.
- Will jump the server to a new world/level.
- All connected clients will follow.
- This is the way multiplayer games travel from map to map, and the server is the one in charge to call this function.
- The server will call `APlayerController::ClientTravel` for all client players that are connected.

`APlayerController::ClientTravel`

- If called from a client, will travel to a new server
- If called from a server, will instruct the particular client to travel to the new map (but stay connected to the current server)

Enabling Seamless Travel

To enable seamless travel, you need to setup a transition map. This is configured through the `UGameMapsSettings::TransitionMap` property. By default this property is empty, and if your game leaves this property empty, an empty map will be created for the transition map.

The reason the transition map exists, is that there must always be a world loaded (which holds the map), so we can't free the old map before loading the new one. Since maps can be very large, it would be a bad idea to have the old and new map in memory at the same time, so this is where the transition map comes in.

So now we can travel from the current map to the transition map, and then from there we can travel to the final map. Since the transition map is very small, it doesn't add much extra overhead while it overlaps the current and final map.

Once you have the transition map setup, you set `AGameModeBase::bUseSeamlessTravel` to true, and from there seamless travel should work!

Seamless Travel Flow

Here is the general flow when executing seamless travel:

1. Mark actors that will persist to the transition level (more below)
2. Travel to the transition level
3. Mark actors that will persist to the final level (more below)
4. Travel to the final level

Persisting Actors across Seamless Travel

When using seamless travel, it's possible to carry over (persist) actors from the current level to the new one. This is useful for certain actors, like inventory items, players, etc.

By default, these actors will persist automatically:

- The `GameMode` actor (server only)
    - Any actors further added via `AGameModeBase::GetSeamlessTravelActorList`
- All Controllers that have a valid `PlayerState` (server only)
- All `PlayerControllers` (server only)
- All local `PlayerControllers` (server and client)

    - Any actors further added via `APlayerController::GetSeamlessTravelActorList` called on local `PlayerControllers`