Assignment 1: Simplest Client-Server

1. Get Set Up

If you are not already using gitlab or github, you are now.

Create two Git repos, for your Server and Client projects. Name them TicTacToeClient and TicTacToeServer since tic tac toe is the final boss in this course.

Using Git Desktop (or cmd line), clone your repos so that you have access to two project folders being managed by Git.

Create a Unity project in both of your git repo folders.

In both projects, go to Window->Package Manager, in “Packages:Unity Registry” search for “Unity Transport” and install.

Get the simplest server and client components from:

https://gitlab.com/netcode-in-games-campaign-manual/simplestclient/-/blob/main/Assets/NetworkClient.cs

https://gitlab.com/netcode-in-games-campaign-manual/simplestserver/-/blob/main/Assets/NetworkServer.cs

Add the NetworkClient script to your client project. Add the NetworkServer into your server project.

In each project, create a gameobject and attach the appropriate network script to it.

Push an initial commit on both of your projects.

Link me as a collaborator on both of your projects.

https://gitlab.com/FernandoTeaches

https://github.com/WindJesterFernando

2. Initial Demonstration

Open NetworkClient.cs, find the string constant named IPAdrees and then edit it to your local IP Address. To discover your local IP, on Windows, use ipconfig in the windows terminal or, if you are on Mac, via your apple icon -> Sys Pref -> Network menu options.

Open your server program and run it. Now run your client program. Does it connect (and stay connected)? When you focus on your client program and hit the 'a' key, do you get a hello from your client in your server debug log?

Is there a delay in your debug log msgs? Do they not show up until you select the project? By default unity does not run in the background. To change this, go to Project Settings -> Player -> Resolution and Presentation and then toggle on the Run in Background option.

3. The "out" Parameter Modifier

What is the "out" keyword parameter modifier in C#? What does it do?
Why do we need to use the out keyword parameter mod with int values but not when we pass class instances?

The out keyword lets methods return multiple values via parameters, requiring the method to assign a value before exiting. For int (value type), out is needed to modify the original variable since only a copy is passed. For class instances (reference types), the method works on the original data directly, so out is unnecessary.

4. Reading Code

The cornerstone of this course is the code contained with NetworkClient.cs and NetworkServer.cs. You are being challenged not just to understand the gist of it, but instead, seek to understand every line of it.

Reading code is a skill to get good at in and of itself. As you may have intuitively deduced, contrary to the way we read a book, the process of reading code is not as linear. Thus the question opens, how does one best read through code? Consider the following approach which is optimized to reduce cognitive overload, stress.

1. Scan through the code, look for any keywords or syntax that you do not understand. These are easily isolated and can initially be figured out. Answer the following: what

keywords/syntax are not immediately known to you and (after some research) what do they do?

NetworkDriver.EndSend - In Unity's multiplayer network system, this is the key function used to complete the sending of a packet. It helps ensure that the send operation initiated by BeginSend is completed correctly.

DataStreamWriter.WriteInt - In Unity's network transport API, it is used to write an int value to a stream buffer to be sent over the network. It returns a deferred handle that allows you to overwrite the written value later if needed. This method ensures efficient serialization of multiplayer communications.

DataStreamWriter.WriteBytes - In Unity's network transport API, you can write raw byte data to a data stream buffer to be sent over the network. You provide an array of bytes and the number of bytes to write. This is useful when transferring serialized objects, custom messages, or other binary data during a multiplayer session.

NativeList - In Unity, this is a dynamic and resizable collection designed for use with Unity's job system and burst compiler. It stores elements in native memory, which has better performance and lower overhead than typical managed collections such as List<T>.

NetworkDriver.ScheduleUpdate - This is a method in the Unity Networking API that handles incoming and outgoing network messages. It schedules the driver to update its state, managing tasks such as handling connection events, processing received packets, and preparing outgoing data. This is critical for maintaining network communications and ensuring smooth gameplay in multiplayer scenarios.

NativeArray - In Unity, this is a fixed-size contiguous memory array designed for high-performance applications. It allows developers to store data in native memory, resulting in better performance in multithreaded and parallel jobs, especially when using the job system or burst compiler. Unlike managed arrays, you must explicitly allocate and dispose NativeArray to manage memory correctly. It provides methods to efficiently read, write, and copy data.

2. Scan through the code and look for external classes, structures, enumerations and function calls. Read the documentation on them. This should take a while. Answer for each of the following:
    2.1. What is the NativeList struct?

In Unity, this is a dynamic and resizable collection designed for use with Unity's job system and burst compiler. It stores elements in native memory, which has better performance and lower overhead than typical managed collections such as List<T>.

2.2. What is the NetworkDriver struct?

The NetworkDriver struct in Unity's Netcode is a low-level networking abstraction used to send and receive data over network connections. It manages the underlying transport layer, handles connections, and provides methods for sending and receiving packets in multiplayer applications.

2.3. What is the NetworkConnection struct?

The NetworkConnection struct in Unity's Netcode represents a single connection between a client and a server. It is used to manage and interact with a specific connection, enabling tasks such as sending data to, or receiving data from, a particular client or server.

2.4. What is the NetworkPipeline struct?

The NetworkPipeline struct in Unity's Netcode represents a customizable processing path for network data. It allows you to define how data is handled, such as adding reliability, encryption, or compression layers. A NetworkPipeline is used with a NetworkDriver to process data before it's sent or after it's received, tailoring the communication behavior for specific requirements.

2.5. What is the FragmentationPipelineStage struct?

The FragmentationPipelineStage struct in Unity's Netcode is part of the network pipeline system and is responsible for handling the fragmentation and reassembly of large data packets. When a packet exceeds the maximum transmission unit (MTU) size, this stage breaks it into smaller fragments for transmission. On the receiving end, it reassembles these fragments back into the original packet, ensuring reliable data delivery without exceeding size limits

2.6. What is the ReliableSequencedPipelineStage struct?

The ReliableSequencedPipelineStage struct in Unity's Netcode is part of the network pipeline system, designed to ensure reliable and ordered delivery of data packets. It handles retransmissions for lost packets and guarantees that packets are delivered to the recipient in the exact order they were sent. This

stage is particularly useful for applications requiring strict data integrity, like gameplay-critical messages in multiplayer games.

2.7. What is the NetworkDriver’s CreatePipeline() function?

The NetworkDriver.CreatePipeline() function in Unity's multiplayer networking API creates a network pipeline, which is a configurable data flow path that processes and transfers messages between a client and server. It allows you to add stages like reliability, fragmentation, or encryption to tailor how data is sent.

2.8. What is the NetworkEndPoint struct?

The NetworkEndPoint struct in Unity is part of the Unity Transport layer. It represents the address of a network connection, including the IP address and port number. It is used to define where data is sent or received in a multiplayer game or networked application. The struct supports creating endpoints for both IPv4 and IPv6 addresses.

2.9. What is the NetworkEndPoint.Parse() function?

The NetworkEndPoint.Parse() function in Unity is used to create a NetworkEndPoint by parsing a string representation of an IP address and a port number.

2.10. What is the NetworkDriver’s ScheduleUpdate() function?

The NetworkDriver.ScheduleUpdate() function in Unity schedules a job to process incoming and outgoing network events. It ensures that the NetworkDriver updates its state, including processing data packets, accepting new connections, and maintaining existing ones. This function is commonly used in conjunction with Unity's Job System for efficient handling of network operations in a non-blocking manner.

2.11. What is the NetworkDriver’s ScheduleUpdate().Complete() function?

The NetworkDriver.ScheduleUpdate().Complete() function in Unity is used to complete the job scheduled by ScheduleUpdate(). This ensures that all pending network operations, such as processing incoming/outgoing packets and managing connections, are fully executed before proceeding further in the code. It synchronizes the job execution, making it a blocking operation until the scheduled update is finished.

2.12. What is the NetworkEvent.Type enum?

The NetworkEvent.Type **enum** in Unity is typically used with networking systems, such as Unity's deprecated UNET or other lower-level networking implementations. It is designed to define various types of network events that might occur during network communication.

2.13. What is the NetworkConnection’s PopEvent() function?

The NetworkConnection.PopEvent() function in Unity (commonly found in Unity's low-level networking systems, such as UNET) is used to retrieve the next network event from the connection's event queue.

3. Learn the name of each member variable and function.

4. Read through the code, more or less, linearly. Seek to read each line and understand what it does.

5. Read through the code again and generate a list of any unanswered questions you have.

Note; in some cases, an effective description of a thing can be produced by restating the name of the thing, when this is the case, it is fine to do, but if there is more to say, seek to say it.

Congratulations, transporting data across the internet is a thing you know how to do!