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 is a keyword in C# which is used to pass the arguments to a method by reference, allowing the method to modify the original variable directly. It is generally used when a method returns multiple values. For the value types this includes int, float, bool, struct, ect. When you pass them to a method, C# passes a copy of the value, not the original. This means changes made to the parameter inside the method wont affect the original variable outside the method. This is good when you want the method to initialize or modify the value of the original variable directly.

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?

   ushort - a data type that is used to declare variables that can store unsigned 16-bit (2-byte) integers. It can hold values ranging from 0 to 65,525. It is an alias for the System.UInt16 type in the .NET framework.

   System.byte - A built in value type that represents an unsigned 8-bit integer. It is part of the .NET framework and belongs to the System namespace. Since it is unsigned, it can only store non-negative values. In C#, the byte keyword is an alias for the System.Byte so both can be used interchangeably. A common use of the System.Byte is to represent an array of bytes, for example, when working with file streams of encoding text.

   NetworkDriver.EndSend - In Unity's multiplayer networking system, this is a critical function used to finalize the sending of packets. It helps ensure that the correct completion of a send operation initiated with BeginSend.

   DataStreamWriter.WriteBytes -In Unity's networking transport API allows you to write raw byte data into the data stream buffer for sending across the network. You provide a byte array along with the number of bytes to write. This is useful when transmitting serialized objects, custom messages, or other binary data during multiplayer sessions.

   DataStreamWriter.WriteInt - In Unity's networking transport API this is used to write an int value to the stream buffer for sending over the network. It returns a deferred handle, allowing you to overwrite the written value later if needed. This method ensures efficient serialization for multiplayer communication.

NetworkDriver.BeginSend - In Unity's transport API is used to initiate the process of sending data over the network. It prepares a DataStreamWriter where you can write the content to be sent. After writing the data, you finalize the send operation using EndSend. This method ensures that the network driver allocates the necessary resources for sending, preventing overflows or conflicts when handling packets.

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, offering better performance and lower overhead than typical managed collections like List<T>. Since it operates in unmanaged memory, you must dispose of it manually to avoid memory leaks.

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, enabling better performance in multithreaded and parallel jobs, especially when working with the job system or burst compiler. Unlike managed arrays, you must explicitly allocate and dispose of NativeArray to manage memory properly. It provides methods for reading, writing, and copying data efficiently.

NetworkDriver.ScheduleUpdate - This is a method in Unity's networking transport API that processes incoming and outgoing network messages. It schedules the driver to update its state, managing tasks like handling connection events, processing received packets, and preparing outgoing data. This is essential for maintaining network communication and ensuring smooth gameplay in multiplayer scenarios.

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?
   Already looked into this one, however:
   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, offering better performance and lower overhead than typical managed collections like List<T>. Since it operates in unmanaged memory, you must dispose of it manually to avoid memory leaks.

   2.2. What is the NetworkDriver struct?
   The NetworkDriver is the main API with which users interact with the unity transport package (cool). It can be thought of as a socket with extra features.

2.3. **What is the NetworkConnection struct?**
This is public representation of a connection. This is obtained by calling Accept() (on servers) or Connect(NetworkEndpoint) (on clients) and acts as a handle to the communication session with a remote peer.

2.4. **What is the NetworkPipeline struct?**
This is the identifier for a network pipeline obtained with CreatePipeline(Type[]).

2.5. **What is the FragmentationPipelineStage struct?**
Pipeline stage that can be used to fragment large packets into MTU-sized chunks. Use this stage when defining pipelines that need to send packets larger than ~1400 bytes.

2.6. **What is the ReliableSequencedPipelineStage struct?**
This pipeline stage can be used to ensure that packets sent through it will be delivered, and will be delivered in order. This is done by sending acknowledgements for received packets, and resending packets that have not been acknowledged in a while.

2.7. **What is the NetworkDriver's CreatePipeline() function?**
This is to create a new pipeline from stage types. A pipeline consists of various processing stages (like serialization, compression, or encryption) that handle data packets as they are sent or received. By defining a custom pipeline, you can control how data is processed at each stage, optimizing performance and ensuring that specific requirements are met for your network communications.

2.8. **What is the NetworkEndPoint struct?**
Just learned that “NetworkEndPoint” (P being capital) is actually obsolete. Now we use “NetworkEndpoint”, which by definition is the representation of an endpoint on the network. Typically, this means an IP address and a port number, and the API provides means to make working with this kind of endpoint easier.

2.9. **What is the NetworkEndPoint.Parse() function?**
The “Parse” is the provided IP address and port. Prefer this method when parsing IP addresses and ports that are known to be good (e.g hardcoded values.)

2.10. **What is the NetworkDriver's ScheduleUpdate() function?**
This schedules an update job. This job will process incoming packets and check timeouts (queuing up the relevant events to be consumed by PopEvent(out NetworkConnection, out DataStreamWriter) and Accept()) and will send any packets queued with EndSend(DataStreamWriter). This job should generally be scheduled once per tick.

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

This is a function in Unity's networking transport API that schedules the driver to process network events and then waits for the operation to complete. This is useful for ensuring that all pending network activities are handled before proceeding with other logic, maintaining synchronization in networked gameplay.

2.12. What is the NetworkEvent.Type enum?
This is an enum that represents various types of network events that can occur during communication. It includes values such as:
Empty: No event occurred.
Connect: A new connection was established.
Disconnect: A connection was terminated.
Data: Data has been received.

2.13. What is the NetworkConnection's PopEvent() function?
This pops the next available event on the connection.

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.**

   I think I only have one:
   1. What additional error handling could be implemented to improve the server?

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!