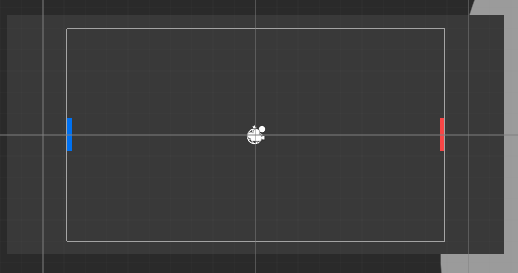
1. **Pong:**

Having fully tested the capabilities of the Socket API, we are nearing the original objective we sought: making a multiplayer game playable via LAN. Making a game is no easy task, which is why we used the Unity Engine, a powerful tool that is used by amateurs and professionals alike to build creative games and ideas. It uses the C# language as well as a component system.

Let’s start by briefly talking about the Unity Editor:

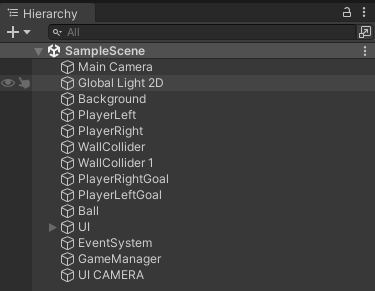
The editor is structured into:

1. The scene view:



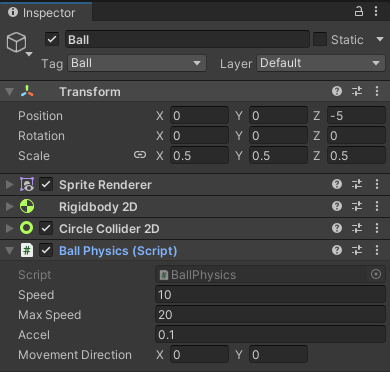
It allows direct control over all the elements inside a specific scene.

1. The hierarchy:



It contains all the objects of the scene, as well as how they are related, one being the child of another. It is also possible to add objects through it.

1. The inspector:

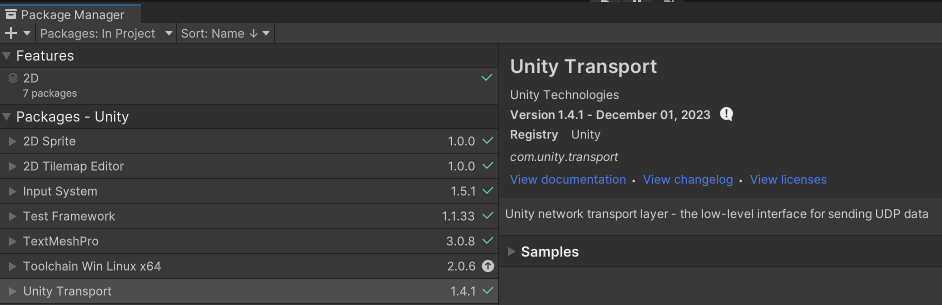


It allows the modification of component settings. The example shown here is that of a ball, it has a sprite renderer to show its image on screen and handle its color, a Rigidbody2D and a CircleCollider2D that go hand in hand in allowing collision detection and physics simulation. It also has a BallPhysics script attached to it, this is a custom script written in C# as shown by the script in parenthesis. Finally, all objects have a Transform component that handles their position, rotation, and scale in the world.

Any property added to a component can be directly exposed in the inspector to allow easy modification and customization.

1. There is also a console window for debugging, a project window to browse the scripts and assets (images, settings, audio files…) of the project/game.

Unity has a lot of features built-in, but networking is not supported by default. To add it to our game, we have to access the Package Manager and download Unity’s Transport Package:



You may have noticed a particular abbreviation whilst looking at the figure, the Unity Transport does not use the TCP protocol! It sends its data using UDP. This is logical when you think about the nature of a game. It is an amalgam of objects constantly moving around the scene, effects, sounds, collision happening all at once, every frame. If we use TCP, the sender will make the data arrives to the receiver, but we usually want the sending and receiving to be in real-time, so UDP is the better choice, we can lose some data on the way, what’s important is the responsiveness of the controls.

This is it for our quick overview of the Unity Editor. To reiterate, we create objects, to which we attach components or scripts to control their behavior, and to use networking capabilities, we install the unity transport package to the project.

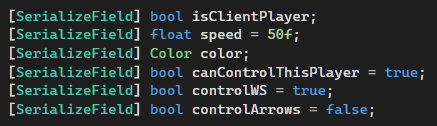
In what follows, we will review the code of the game and how the communication occurs. It is important to note that some methods will be briefly summarized and we will focus mainly on the networking aspect of the game.

We will begin by showing a little demo showcasing a local game of Pong.

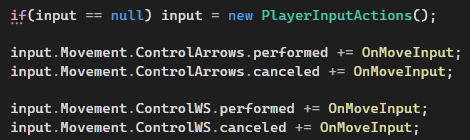


So… what is Pong? As you can see in the video, Pong is a game where each player controls one paddle, and has to reflect the ball back to the other player to score points.

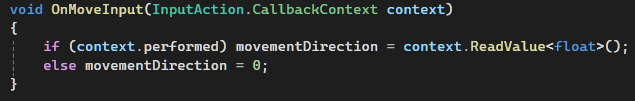
Now let’s take a look at the code of the game itself starting from the paddle/player control script.



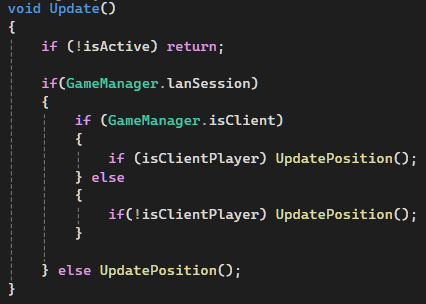
The player script has these properties marked with the attribute SerializeField. This exposes a private variable to the inspector. So the user, can decide if the paddle is that of the client player, the speed at which the paddle moves, its color, and if it uses the arrows or W and D for the movement input.



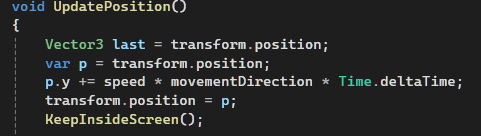
To handle the input, the script subscribes to the event triggered by the PlayerInputActions.



This is the function that takes the callback of the input, if the input is performed, we get the direction of movement (either up or down), and if it is canceled, we make sure the direction is set to 0.

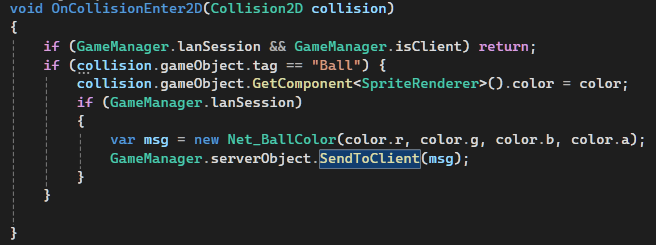


The update function is a function that runs every frame by the engine on each MonoBehavior (the inherited class of every component). Here, the Update function checks if we are playing locally or on LAN and decides if it should UpdatePosition of the paddle.



Every frame, UpdatePosition will take the movement direction and modify the y position of the transform of the paddle. Time.deltaTime is used to smooth the movement as the framerate can vary between machines, this ensures that the object moves the right amount vertically each frame. KeepInsideScreen is used to keep the paddle within the bounds of the playable screen.

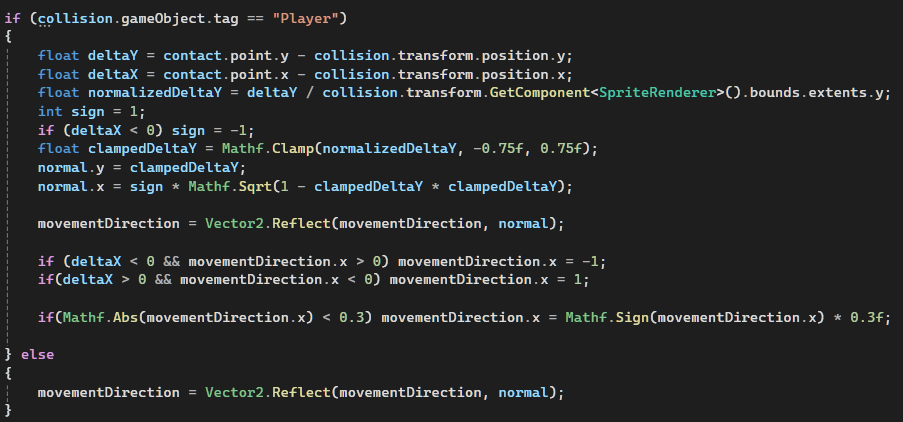
Another function, OnCollisionEnter2D is called whenever an object equipped with a rigidbody2d and a collider2d enter in collision:



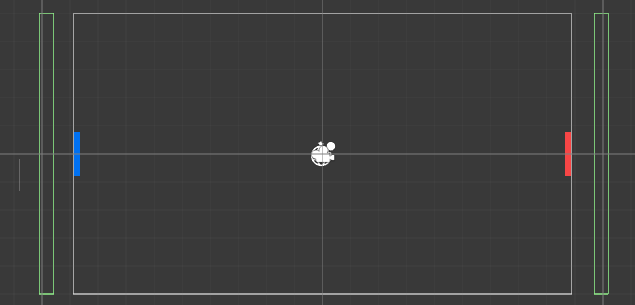
Here, if the object the paddle collided with is the ball, we change the ball’s color to the paddle’s color by accessing the ball’s SpriteRenderer and changing its color property.

That is basically how the paddle works, minus the networking handling.

As for the ball, the idea is relatively the same as that of the paddle, but in this case, the ball cannot be controlled, it takes a random direction at the start of a round, and upon hitting a wall, it is reflected. When it collides with a paddle however, the reflection is weighted based on where the ball touches the paddle:

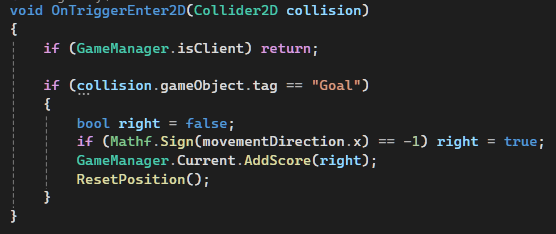


For the scoring to work, there are invisible trigger behind each player, if the ball enters one of the triggers, the score is added to the correct side:

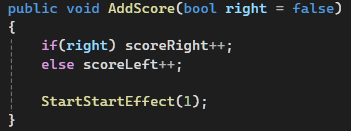


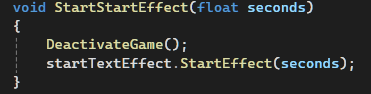
The triggers’ colliders are the green boxes.

When the ball enters a trigger, the OnTriggerEnter function is called:



As you can see, the script handling the addition of the score is the GameManager script. Let’s check some of its functions:





This handle adding the score to the correct player as well as showing the counter on the screen after pausing the game.

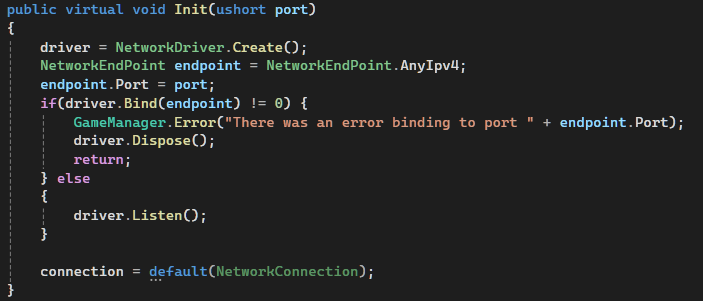
The GameManager is a Singleton, meaning it only has one static instance over the span of the game’s life, it also prepares the server/client settings.

Finally, let’s delve into how the networking is handled in the game.

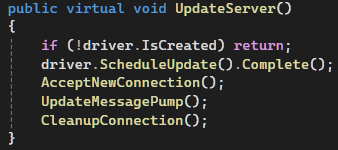
We will first look at the server:



Instead of a socket, we use a NetworkDriver and a NetworkConnection:



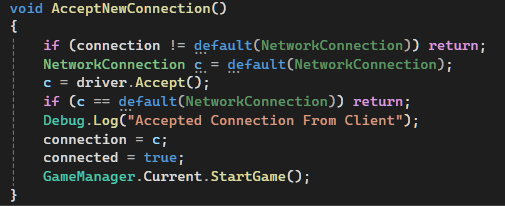
We start by creating a driver, and initializing our endpoint using the port passed as an argument. Then we bind to that point, if it returns 0, then there was a problem, else we start listening for a connection.



The update function of the server calls UpdateServer each frame.

CleanupConnection removes any lost connection.

AcceptNewConnection: (server can accept many clients, if it is in game, those clients are added to a queue to wait for their turn to play)



If no connection is ongoing, and we listen to a new one, we accept it and store it in our NetworkConnection, and then we can start the game.

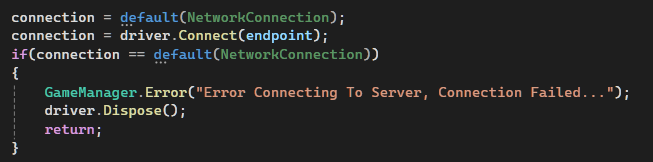
UpdateMessagePump:

This is the function that receives the data from the client:



We first pop any event from the driver, and read the OpCode of the message, and based on that we execute the command sent. If we were disconnected from the client, we end the game.

The client works almost the same as the server:

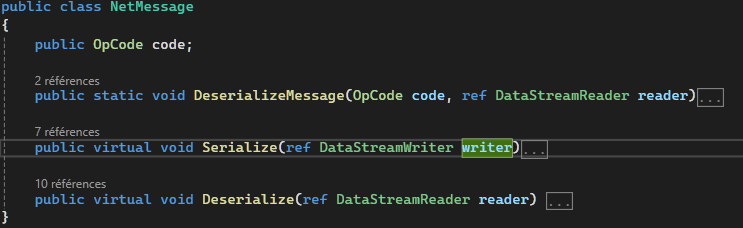


This is how the client connects to the server.

It also updates the messages it receives the same way the server does.

Let us now discuss how the message is executed.

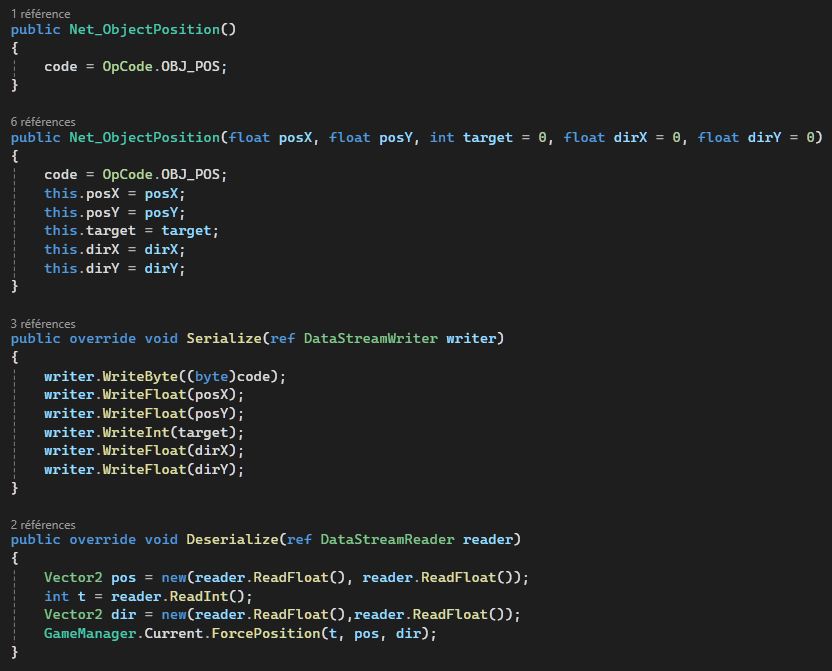
We use the NetMessage Class to deserialize the incoming commands. Every message has the opcode as its first byte, this is used to know the nature of the command to be executed.



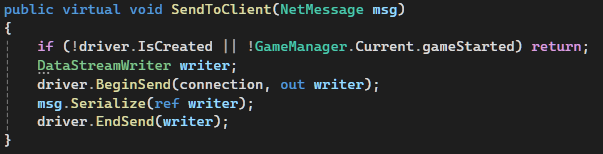
This is the base class of a net message. Serialize is used to send the data after converting them to bytes, Deserialize does the exact opposite for the receiving end.

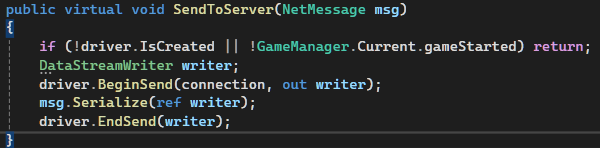
There are many net messages, the net position to control the position of the ball, the net game state to control the flow of the game etc.…

This is an example of the net position:



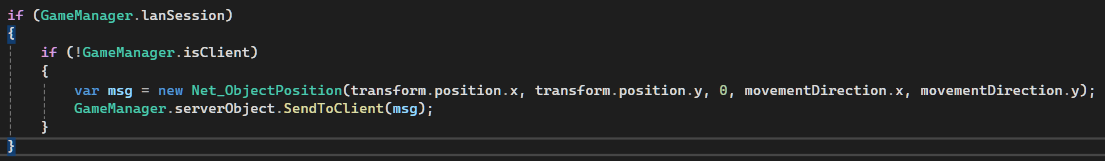
The messages are sent either by the server or the client:



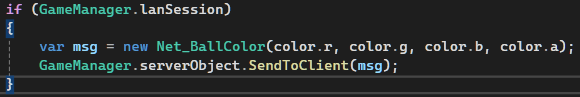


Here are some examples of messages sent between the server and the client:

1. Updating the ball’s position in the client upon reflection:



1. Changing the ball’s color:



1. Updating the paddle position:

