Computación Gráfica

Desarrollo de juegos multijugador en Unity

Máster Universitario en Ingeniería Informática

Curso 2020/2021

MÁSTER UNIVERSITARIO
INGENIERÍA INFORMÁTICA

VNiVERSiDAD
D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Luis Blázquez Miñambres

Contenido

Tabla de ilustraciones

1. Introducción

En este documento se recoge el informe acerca del trabajo teórico práctico de la asignatura "Computación Gráfica" del Máster en Ingeniería Informática por la Universidad de Salamanca en el Curso 2020-2021, consistente en el análisis del modo de creación, configuración y desarrollo de videojuegos en línea o servicios multijugador que ofrece la herramienta Unity, junto con un pequeño ejemplo de funcionamiento.

2. Material de ejemplo

En este apartado, se explicará brevemente un ejemplo sencillo de un videojuego en 2D con una nave que dispara a naves enemigas, desarrollado y enfocado principalmente para ser jugado en multijugador o en línea a partir de los aspectos desarrollados anteriormente.

2.1. Contexto

El multijugador es un modo de juego relacionado con el área de los videojuegos, en el que dos o más jugadores pueden cooperar con un clan (o equipo) o competir (generalmente) para jugar el mismo juego al mismo tiempo . Este tipo de juegos multijugador se pueden jugar dentro de la misma red (conocido como conexión en red LAN) o desde redes distintas a través de un servidor de juegos de Internet [1].

Los juegos multijugador pueden proporcionar evidencia cronológica de dos fuentes principales: el cliente y el servidor. La evidencia de la línea de tiempo viene en forma de cuándo alguien jugó uno de estos juegos, pero en algunos casos, cuánto jugó una persona es igualmente importante. En las siguientes secciones, mostraremos cómo determinar cuándo alguien está jugando y cuánto tiempo pasa en línea.

Los juegos multijugador de Internet son juegos que requieren una conexión a Internet para jugar. Estos son solo juegos en línea, que pueden ser jugados en su navegador sin comprar ningún tipo de software de forma externa. El servidor de juegos al que se conecta el usuario lo proporciona el servicio de juegos, como MSN Gaming Zone, ¡Yahoo !. Gaming o Pogo [2], solo por nombrar algunos sitios de juegos de Internet como ejemplos.

Para utilizar estos servicios, normalmente el usuario o jugador debe crear una cuenta en el juego e iniciar sesión para utilizarla, de forma que se tenga identificada de alguna manera su IP con su nombre de usuario a modo identificativo.

Este tipo de juegos en línea brindan a los jugadores ocasionales la oportunidad de unirse a la comunidad de jugadores sin tener que pasar varias horas al día jugando juegos al mismo tiempo. Los servicios los juegos multijugadores permiten realizar un seguimiento de las estadísticas de los jugadores, proporcionando foros y salas de chat en el juego para comunicarse con otros jugadores del mismo juego. También se pueden crear listas de amigos como parte del servicio de juegos.

En este informe se relatará la posición de la herramienta Unity frente a todos estos conceptos y como poder desarrollarlos de forma simplificada.

2.2. Configuración de sala de juego

Para cumplir el objetivo del ejemplo utilizado, que es mostrar el funcionamiento de un juego multijugador utilizando Unity se habilitará un componente en el menú principal del juego para conectarse al servidor o sala de juego del multijugador.

Para ello, en Unity se necesita un GameObject al que se le añadirán los componentes NetworkManager y NetworkMangerHUD, para gestionar la conexión multijugador a nivel de red. El aspecto del GameObject quedaría como se muestra en la **¡Error! No se encuentra el origen de la referencia.**. Este objeto será el que se encargue de mantener y gestionar la conexión de diferentes conexiones al juego y sincronizar los GameObject de la escena entre todos los clientes. Mientras que el NetworkMangerHUD simplemente mostrará una interfaz o HUD para los jugadores que se conecten al juego.

En este ejemplo se utilizarán las opciones de LAN Host y LAN Client. Los juegos multijugador de Unity funcionan de la siguiente manera: primero, un jugador inicia un juego como anfitrión (seleccionando LAN Host). Un host funciona como cliente y servidor al mismo tiempo. Luego, otros jugadores pueden conectarse a este host como clientes (seleccionando Cliente LAN). El cliente se comunica con el servidor, pero no ejecuta ningún código de solo servidor. Entonces, para probar este juego, en los casos de uso posteriores se abrirán dos instancias, una como Host y otra como Cliente.

Sin embargo, no se puede abrir dos instancias del juego en Unity Editor. Para hacerlo, habrá que crear el juego y ejecutar la primera instancia desde el archivo ejecutable generado. La segunda instancia se puede ejecutar desde Unity Editor (en el modo de reproducción).

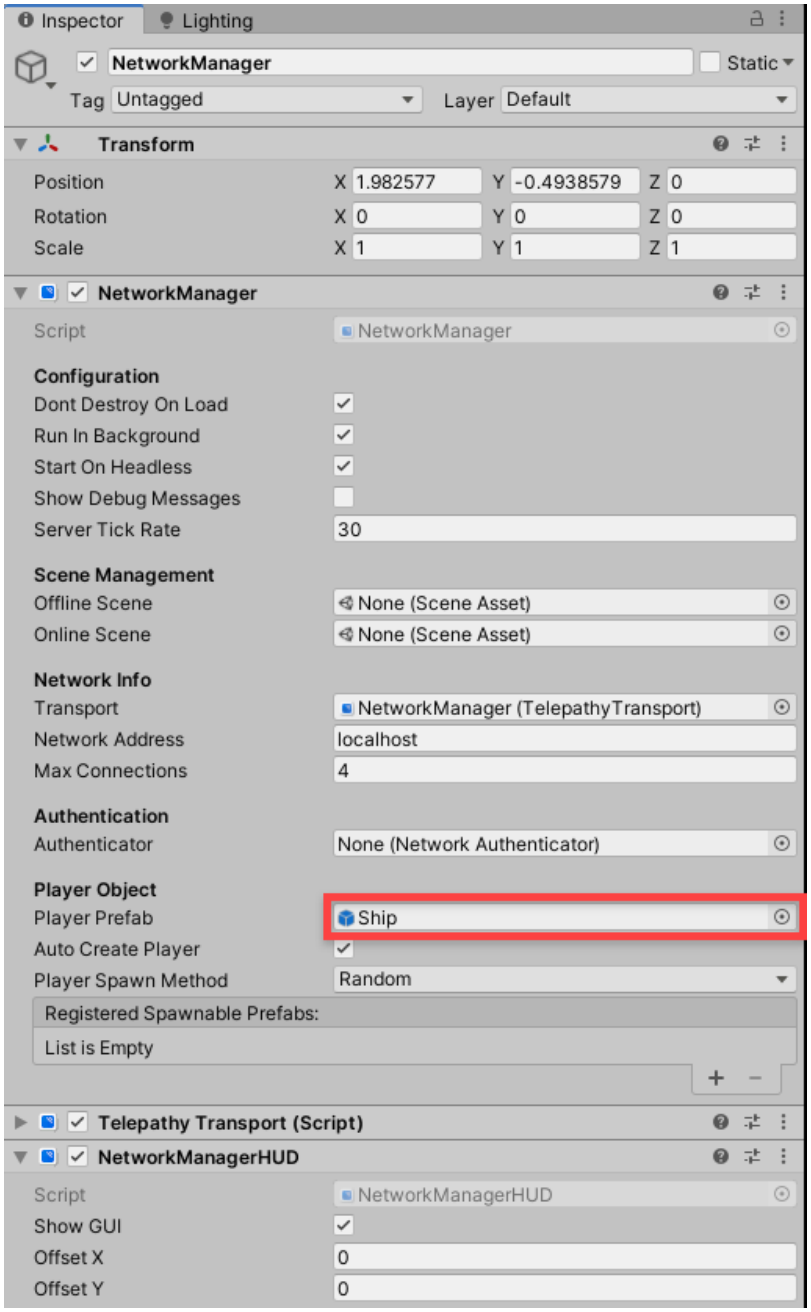

Ilustración 1: Configuración de red del juego

2.3. Diseño del juego y jugabilidad

En este apartado se detalla la jugabilidad del ejemplo planteado , incidiendo en las características específicas de la nave que actuará como jugador principal y de los enemigos que habrá que combatir dentro del entorno.

2.3.1. Jugador principal

Una vez configurado el NetworkManager, se pasa a crear objetos del juego que serán administrados por él mismo. El primero que se va a crear es la nave del jugador principal. A partir del juego base, la nave espacial solo se mueve horizontalmente en la pantalla. Más adelante, se agregará la capacidad de disparar balas y causar daños. Lo primero será crear un nuevo GameObject llamado “Ship” y convertilo en un *prefab* Esto se puede ver en la imagen del apartado anterior Ilustración 1. Por otro lado, la siguiente Ilustración 2, muestra los componentes prediseñados de “Ship”, que se explicarán a continuación.

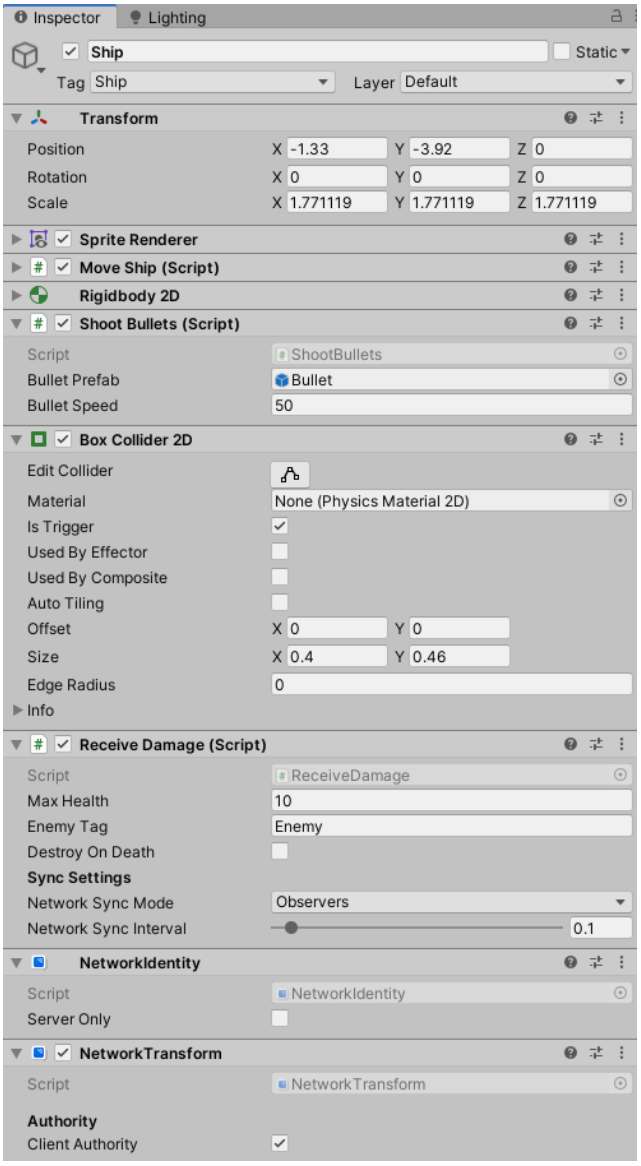

Ilustración 2: Inspector con elementos del jugador principal

El reproductor usa el componente de red de duplicación de Unity para enviar el objeto. Para que NetworkManager administre los GameObject del juego, hay que añadir un componente NetworkIdentity. Nuevamente, dado que la nave del jugador principal será controlada por el jugador, se habilitará la casilla de verificación "Permiso de jugador local". A su vez, el componente NetworkTransform es responsable de actualizar la posición de la nave en el servidor y en todos los clientes. De lo contrario, si se mueve la nave en una pantalla, su posición no se actualizará en las otras pantallas del resto de jugadores. NetworkIdentity y NetworkTransform son dos componentes necesarios para la funcionalidad del juego multijugador en todos los GameObject que se muevan por la escena.

Cabe mencionar que se debe activar los permisos del cliente (Client Authority) en el componente NetworkTransform. Ahora, para lidiar con el movimiento y la colisión, se deberán agregar RigidBody2D y BoxCollider2D. Además, BoxCollider2D será un disparador (establezca Triggers en verdadero), porque no queremos que las colisiones afecten la física de la nave. Finalmente, agregaremos un script MoveShip, que tendrá un parámetro de Velocidad. Más adelante se agregarán otros scripts, pero nada más. El script MoveShip es muy simple: en el método FixedUpdate, se obtendrá el movimiento del eje horizontal y se configurará la velocidad de la nave en consecuencia.

Sin embargo, hay dos cosas muy importantes relacionadas con la red que deben explicarse. En primer lugar, normalmente todos los scripts de los juegos de Unity heredan MonoBehaviour para usar su API. Sin embargo, para utilizar la API de red, el script debe heredar NetworkBehaviour en lugar de MonoBehaviour. Debe incluir el espacio de nombres Networking (utilizando UnityEngine.Networking) para hacer esto. De manera similar, en un juego multijugador de Unity, el mismo código se ejecuta en todas las instancias del juego (anfitrión y cliente). Para permitir que los jugadores controlen solo su propia nave espacial, no todas las naves espaciales en el juego, habrá que agregar una condición If al comienzo del método FixedUpdate para verificar si se trata de un jugador local.

2.3.2. Disparos

Lo siguiente que se va a agregar al juego es permitir que la nave espacial dispare balas. Nuevamente, estas viñetas deben sincronizarse entre todas las instancias del juego.

Primero, habrá que crear el GameObject *prefab* “Bullet” . Para administrar en la red, al igual que en una nave *prefab*, se necesitan los componentes NetworkIdentiy y NetworkTransform. Sin embargo, una vez que se crea la bala, el juego no necesita expandir su posición en la red porque el motor de física actualizará la posición. Por lo tanto, se cambiará la velocidad de reenvío de la red a 0 en la transición de la red para evitar la sobrecarga de la red al disparar tantas balas entre los jugadores que haya en escena.

Además, la bala tendrá cierta velocidad y debería chocar con el enemigo más tarde. Por lo tanto, se agregarán los componentes RigidBody2D y CircleCollider2D a la casa prefabricada.

El script “ShootBullets”, al igual que el anterior script mencionado referente al jugador principal, también es NetworkBehavior. En el método de actualización, verificará si el jugador local ha presionado la barra espaciadora y, si es así, llamará a un método para disparar la bala. Este método creará una instancia de una nueva bala, establecerá su velocidad y la destruirá después de un segundo (cuando la bala haya abandonado la pantalla).

Nuevamente, hay algunos conceptos de red importantes que deben explicarse. Primero, hay una marca [Command] encima del método CmdShoot. La etiqueta y el "Cmd" al principio del nombre del método lo convierten en un método especial llamado Command. En general, un comando es un método que se ejecuta en el servidor, aunque haya sido llamado en el cliente. En este caso, cuando el jugador local dispara una bala, el juego enviará una solicitud

de comando al servidor, y el servidor ejecutará el método en lugar de llamar al método en el cliente.

Además, hay una llamada a NetworkServer.Spawn en el método CmdShoot. El método Spawn es responsable de crear viñetas en todas las instancias del juego. Por lo tanto, lo que tiene que hacer CmdShoot es crear una viñeta en el servidor y luego el servidor replica la viñeta en todos los clientes. Hay que tener en cuenta que esto solo es posible porque CmdShoot es un comando y no un método normal.

La representación del componente de la bala se puede observar en la Ilustración 3.

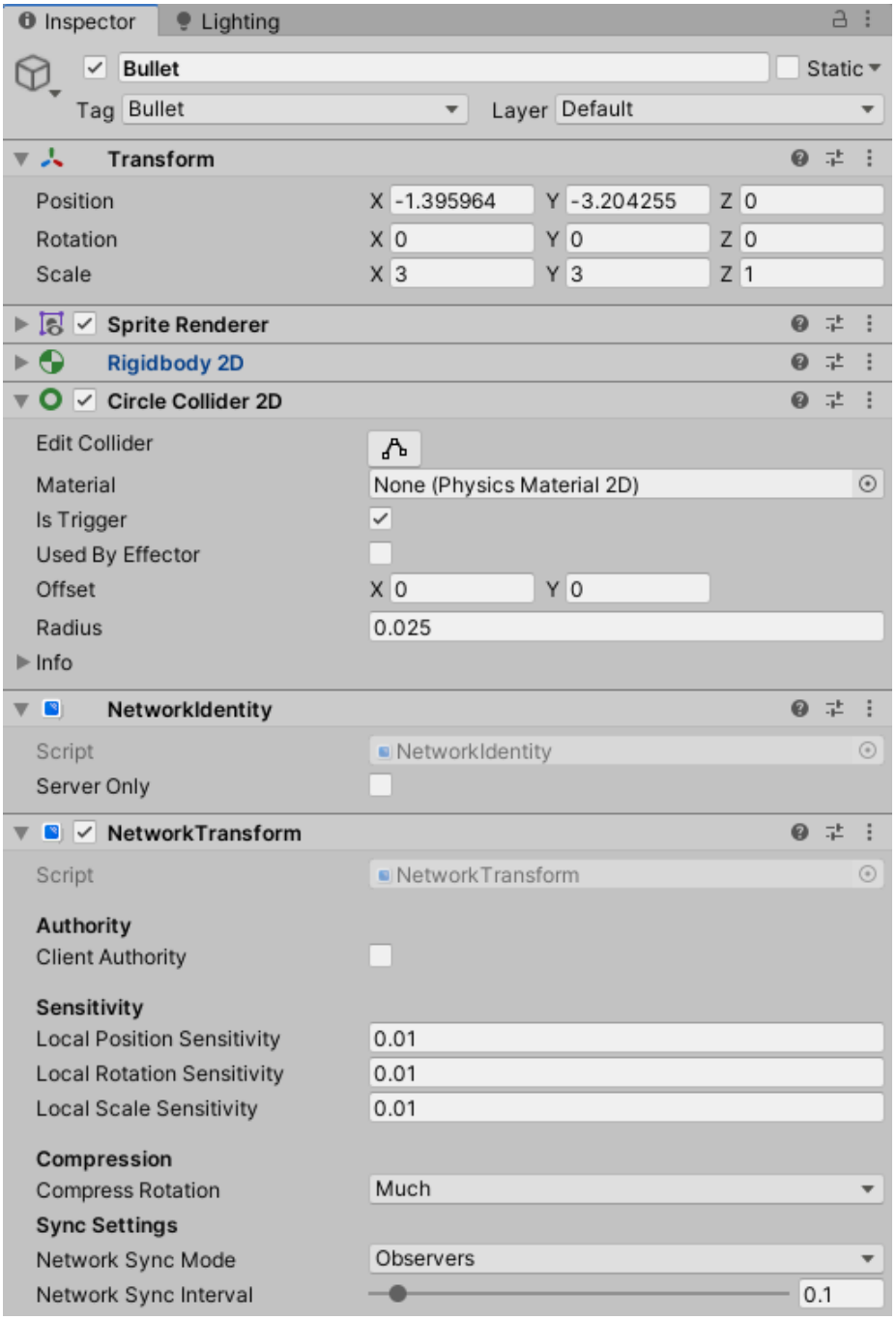

Ilustración 3: Inspector con elementos del objeto de disparo

Finalmente, hay que indicarle al componente al componente que administra la red que puede generar balas. Se hará agregando la viñeta prefabricada en la lista de Prefabricados de generación registrados como se puede ver en la Ilustración 4.

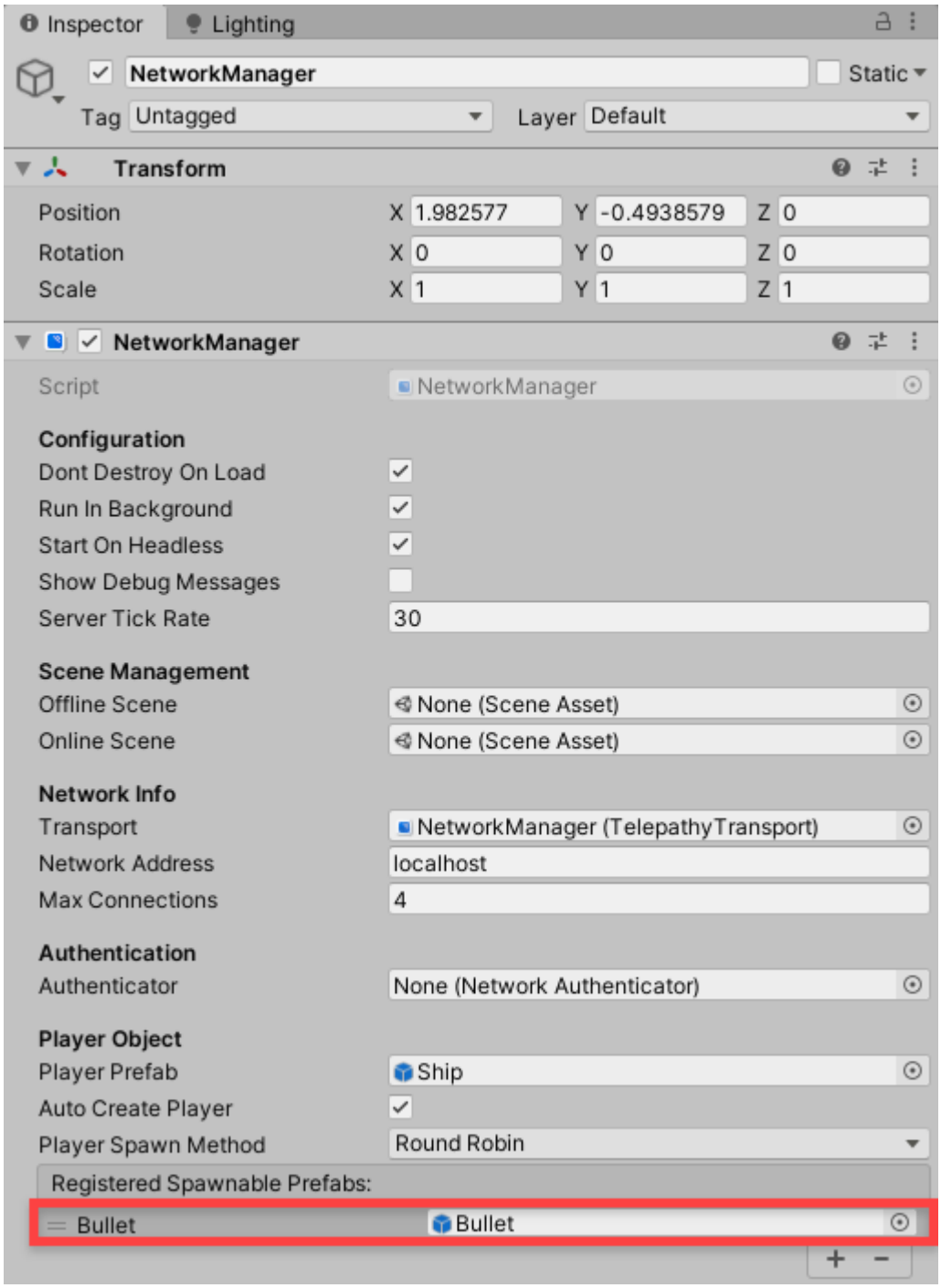

Ilustración 4: Indicación al NetworkManager de la generación de disparos

2.3.3. Enemigos

El siguiente paso del juego es añadir a los enemigos y *spawnearlos* o generarlos en la escena.

Primero, se necesitará un nuevo GameObject "Enemy", que se convertirá en *prefab*. Entonces, cree un nuevo GameObject llamado Enemy y conviértalo en un prefabricado. Al igual que las naves, los enemigos tendrán un Rigidbody2D y BoxCollider2D para manejar movimientos y colisiones. Además, al igual que los otros componentes de la escena, necesitará NetworkIdentity y NetworkTransform, que será manejado por NetworkManager

Ahora, se creará un GameObject llamado “EnemySpawner”. “EnemySpawner” también tendrá NetworkIdentity, pero ahora se activará el campo "Solo servidor" en el componente. De esta forma, el generador solo existirá en el servidor, porque se pretende que no se puedan crear enemigos en cada cliente. Además, tendrá el script “SpawnEnemies”, que generará enemigos a intervalos regulares (los parámetros son la prefabricación del enemigo, el intervalo de generación y la velocidad del enemigo). Esto se puede observar en la Ilustración 5.

En este script hay que tener en cuenta que se utilizará un nuevo método de Unity: OnStartServer. Este método es muy similar a OnStart, la única diferencia es que solo se llama para el servidor. Cuando esto sucede, se llamará a InovkeRepeating cada 1 segundo para llamar al método SpawnEnemy (según spawnInterval).

El método SpawnEnemy creará una instancia de un nuevo enemigo en una ubicación aleatoria y usará NetworkServer.Spawn para copiarlo entre todas las instancias del juego. Finalmente, el enemigo será destruido en 10 segundos tal y como se observa en la Ilustración 6.

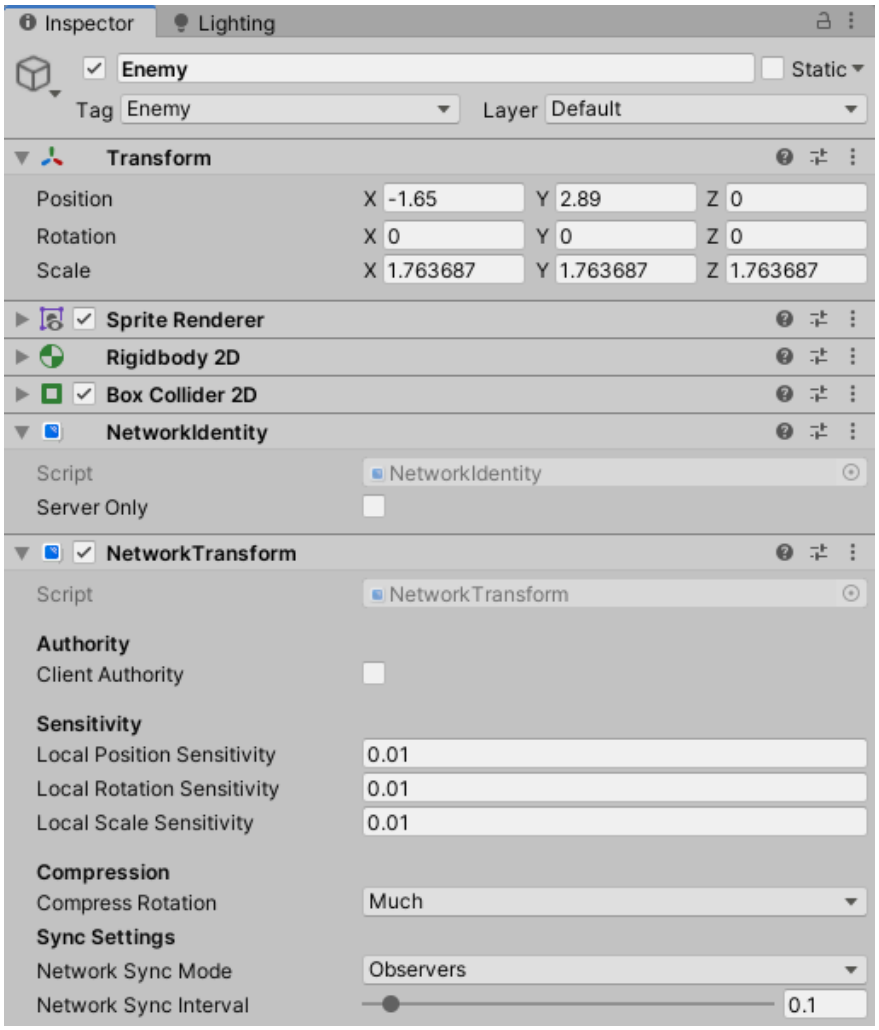

Ilustración 5: Inspector con elementos de los enemigos

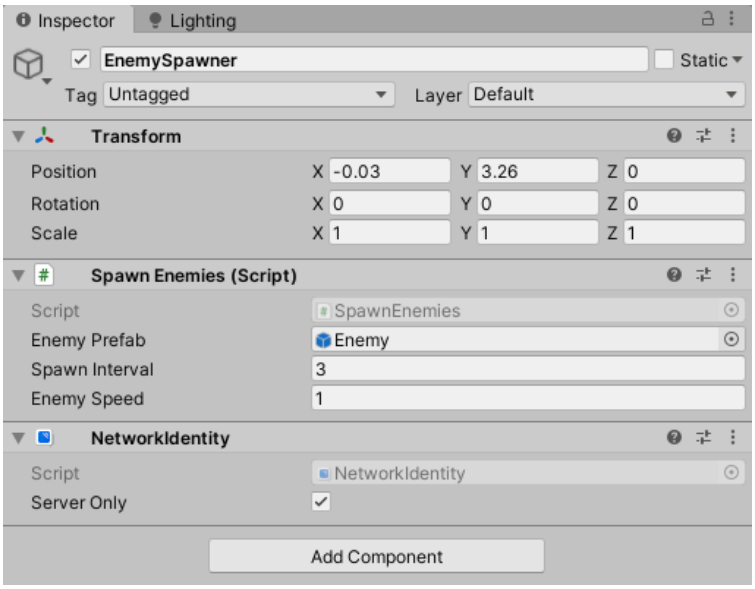

Ilustración 6: Inspector con elementos del generador de enemigos

Antes de jugar, se debe agregar el objeto *prefab* "Enemy" a la lista de prefabricados de generación registrados como se observa en la Ilustración 7.

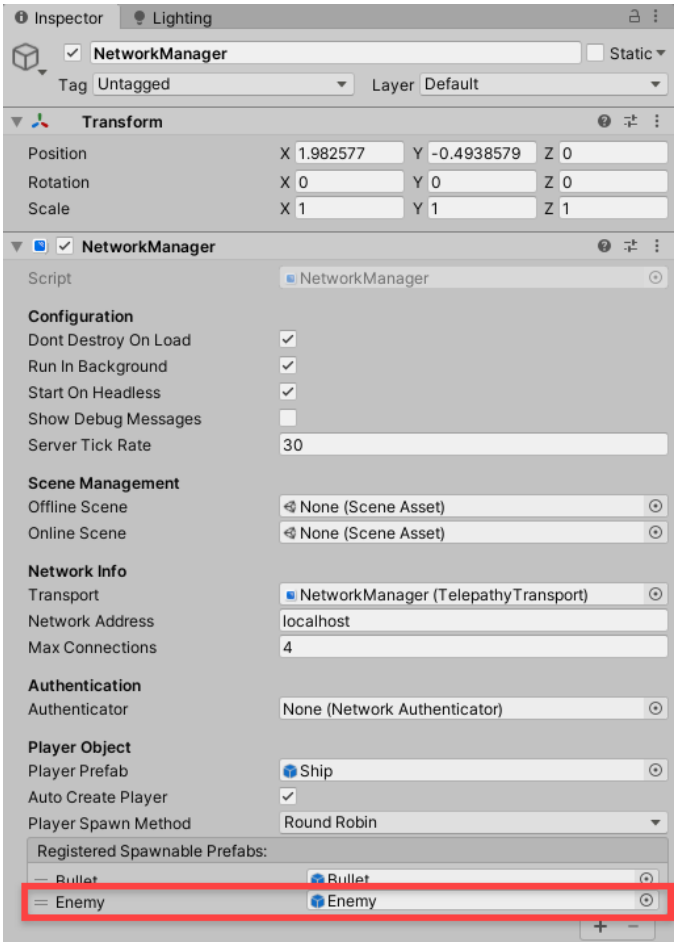

Ilustración 7: Indicación al NetworkManager de la generación de enemigos

2.4. Casos de uso

En el ejemplo planteado, una vez configurado y construido el ejecutable, se presentan tres casos de uso para jugar online , que se presentarán a continuación teniendo tres conceptos , ya mencionados, en cuenta: Host, Cliente, Servidor. Para ello se utilizarán dos instancias de este ejecutable del juego en el mismo ordenador

2.4.1. LAN Host + LAN Client

En este caso de uso, ejecutadas dos ventanas con el mismo juego de ejemplo, donde en una de las instancias del ejecutable se actuará como usuario o cliente Host de la sala de juego y el otro usuario se conectará como cliente a la sala de juego del host.

El usuario Host es aquel que, dentro de una sala de juego, mantiene la conexión de la sala de juego con las prestaciones y configuración de su propia conexión para el resto de los usuarios que se conecten. Forma parte del juego como otro usuario corriente solo que, en este caso, el mantenimiento y viabilidad de la conexión de la partida irá en función de la conexión de este usuario, que actuará también como servidor de la partida a la que se conectarán el resto de los jugadores. En cualquier caso, el cliente Host no tiene por qué ser el creador y administrador del servidor, pero en este caso será así. En cuanto al usuario cliente, este se conectará a la sala de juego del servidor levantado por el cliente Host como un usuario corriente.

Cabe mencionar en este tipo de conexión que, si el usuario Host sale de la partida o se cae su conexión, se cerrará automáticamente la sala de juego. En caso de otros juegos más modernos lo que ocurriría sería que se elegiría un nuevo usuario o jugador host de entre los jugadores conectados dentro de la sala de juegos.

A continuación, en la Ilustración 8, se puede ver la opción elegida para este caso de uso. Y en la Ilustración 9, se puede observar que, efectivamente hay dos jugadores (y tantos como se quiera) en la sala de juego en la que la pantalla de la izquierda corresponderá a la instancia del jugador Host y la de la derecha a la de un jugador cliente cualquiera.

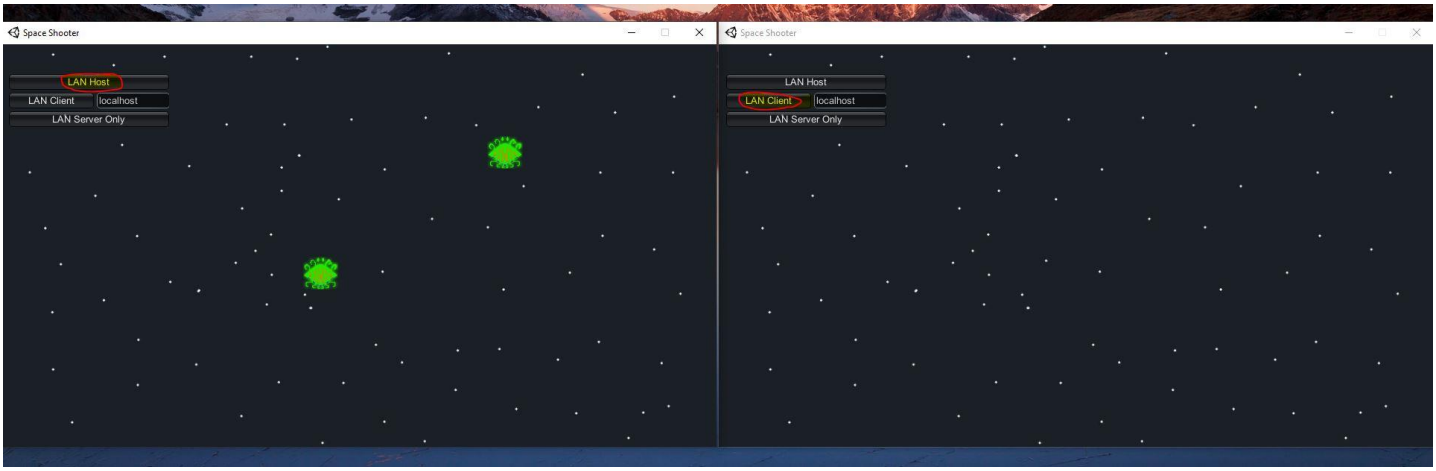

Ilustración 8: Menú principal con las opciones Host y Client elegidas

Cabe también mencionar que, solo se podrá levantar un jugador Host dentro de una misma red (en este caso la local *localhost*) ya que también actúa como servidor de la sala de juego. Si se desease levantar otro servidor por medio de un jugador Host se indicará dentro del menú principal del juego, como el de la Ilustración 8, en la barra para indicar otra IP o dominio.

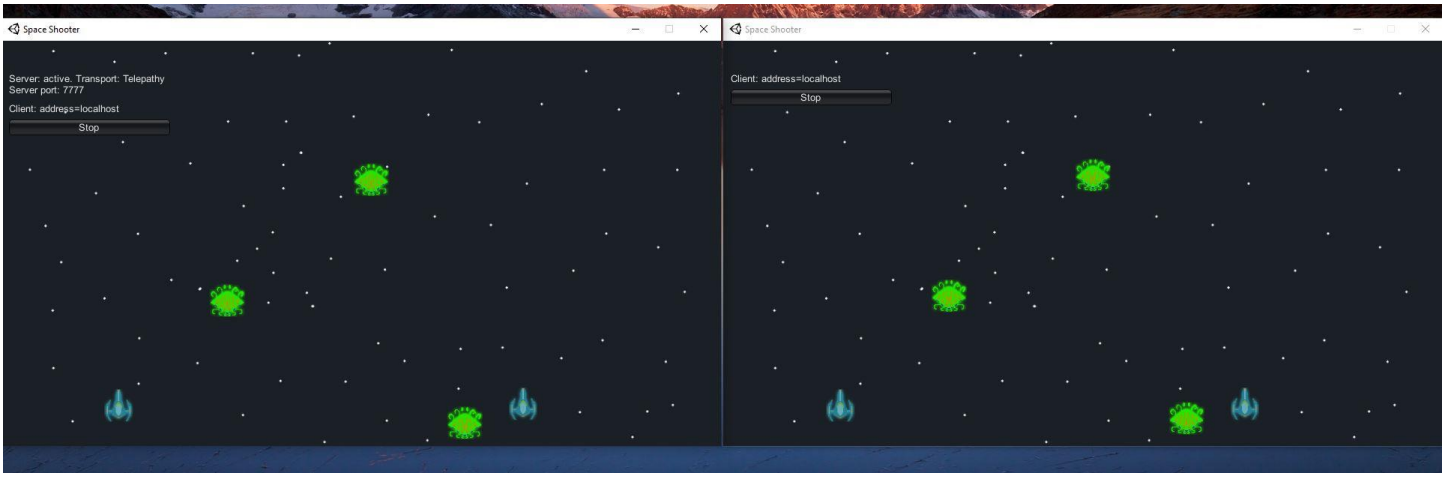

Ilustración 9: Demostración del juego con dos jugadores: un Host y un Cliente

2.4.2. LAN Server + LAN Client

En este caso de uso, ejecutadas dos ventanas con el mismo juego de ejemplo, en una de las instancias del ejecutable se actuará únicamente como servidor de la sesión o sala de juego y el otro usuario se conectará como cliente a la sala de juego del host.

En este caso, con la diferencia del anterior, no habrá dos jugadores involucrados en el juego, sino que una de las dos instancias actuará únicamente como la parte servidor que mantendrá y gestionará la conexión de usuarios cliente a la sala de juego sin participar en él. En cuanto al usuario cliente, este se conectará a la sala de juego del servidor levantado por el cliente Host como un usuario corriente. Al igual que en el caso anterior, se podrán conectar tantos usuarios clientes como se quiera.

A continuación, en la Ilustración 10, se puede ver la opción elegida para este caso de uso. Y en la Ilustración 11, se puede observar que únicamente hay un jugador involucrado en el juego (y tantos como se quiera) ya que la otra instancia del juego solamente actúa como servidor de la sala de juego.

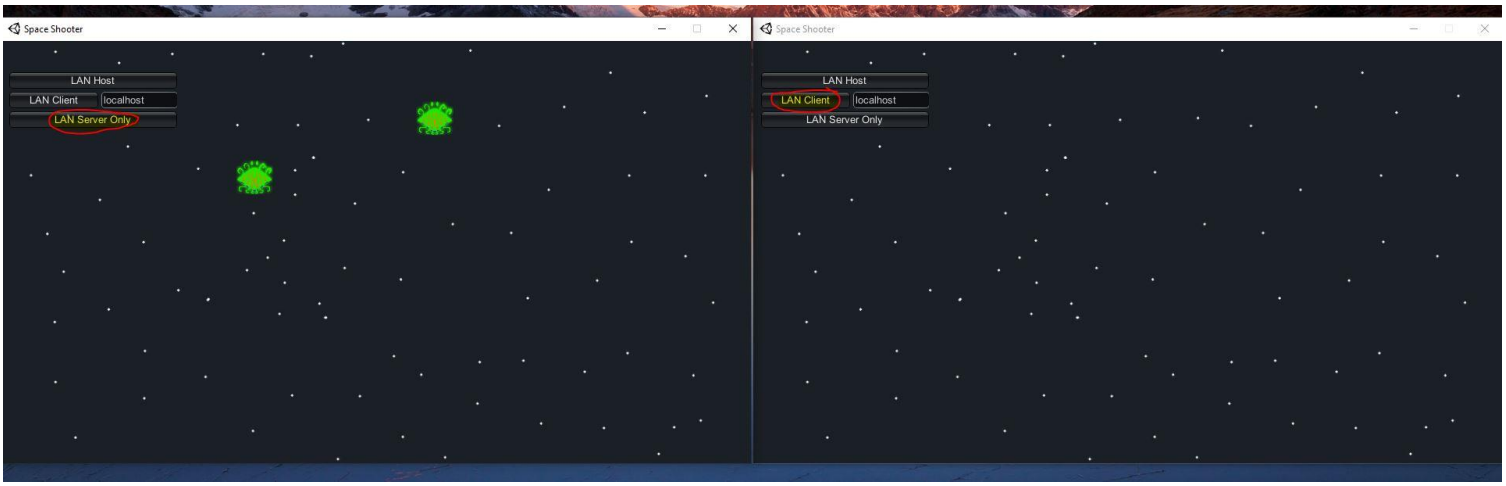

Ilustración 10: Menú principal con las opciones Server y Client elegidas

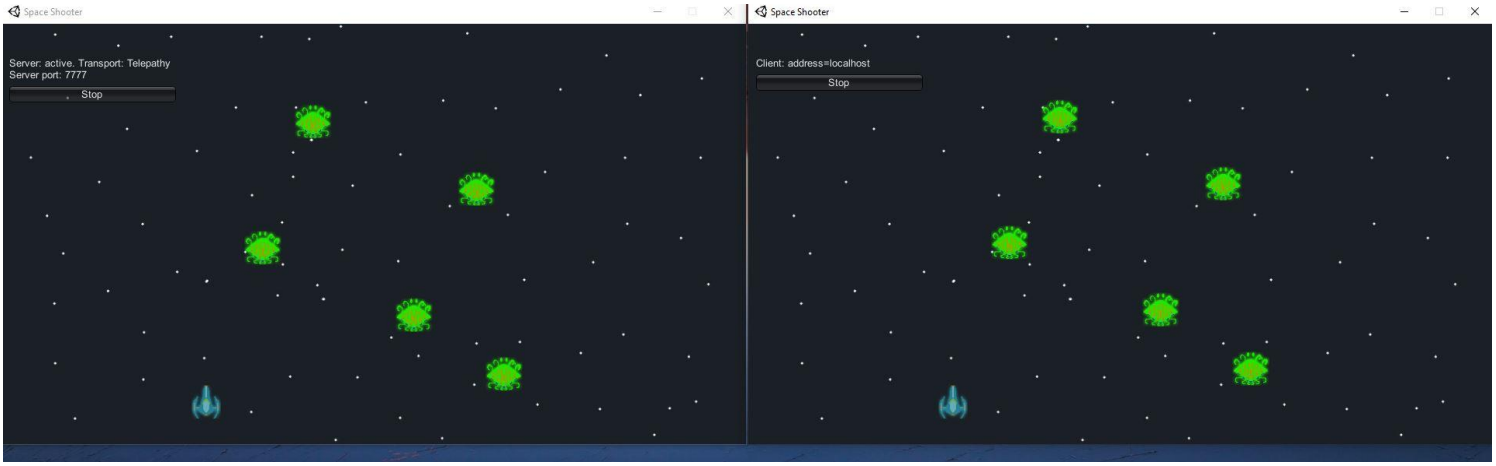

Ilustración 11: Demostración del juego con dos jugadores: un Servidor y un Cliente

Cabe también mencionar que, se podrán levantar tantas instancias de servidor del juego como se desee indicando un puerto distinto en caso de ser de la misma red, o una IP o nombre de dominio distinto en caso de ser redes distintas.

2.4.3. LAN Client + LAN Client

En este caso de uso, ejecutadas dos ventanas con el mismo juego de ejemplo, donde en ambas instancias del juego se encuentran dos clientes dentro de la misma red. En este caso, como se puede observar en la Ilustración 12, al no haber ninguna instancia del juego que funcione como servidor que aloje la sala de juego. Ambos clientes intentan conectarse a *localhost* por medio del puerto asignado a la sala de juego, pero sin éxito debido a lo explicado anteriormente.

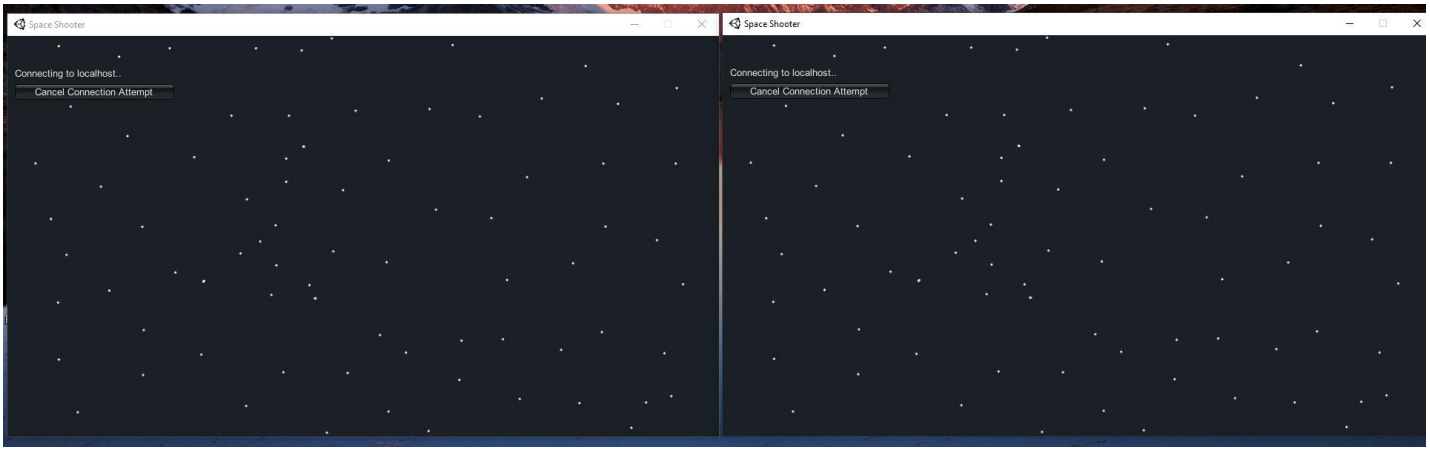

Ilustración 12: Demostración del juego con dos jugadores Cliente

Bibliografía

[1] «Webopedia,» [En línea]. Available: https://www.webopedia.com/definitions/multiplayer/.

[2] «Pogo,» [En línea]. Available: https://www.pogo.com/.