Los juegos multijugador

PID_00243557

Rubén Mondéjar Andreu

Tiempo mínimo de dedicación recomendado: 3 horas

Universitat Oberta de Catalunya

Índice

Introducción

En este módulo didáctico presentamos las nociones básicas del mundo de los videojuegos multijugador, tanto desde la perspectiva conceptual como de las herramientas necesarias para crear uno.

En primer lugar, nos adentraremos en los videojuegos que incorporan esta modalidad, ya sea parcial o completamente. Para ello, nos centraremos en el punto de vista histórico, repasando brevemente su origen y evolución en las diferentes épocas que fueron lanzados.

Seguidamente, veremos brevemente cómo fue esta evolución en otros dispositivos, como son las videoconsolas, los móviles y las tabletas. Este repaso servirá para introducir algunos conceptos básicos por medio de ejemplos de juegos populares e innovadores en su momento.

Para finalizar, y como primera toma de contacto desde una vertiente más práctica, se propone un proyecto consistente en llevar a cabo, paso a paso, un juego multijugador en modo local a pantalla partida. Para ello, se parte de un juego preexistente, que será modificado con el propósito de añadir esta funcionalidad.

Objetivos

En este módulo didáctico se presenta a los estudiantes los conocimientos necesarios para conseguir los objetivos siguientes:

1. Conocer qué es un videojuego multijugador, tanto local como en línea.

2. Estudiar la evolución y las innovaciones de los videojuegos multijugador.

3. Programar un juego en modo multijugador local mediante el motor Unity.

1. La evolución de los juegos multijugador

En este primer apartado vamos a estudiar la interacción que se produce entre diferentes jugadores, ya sea en un mismo dispositivo (por ejemplo, ordenadores o videoconsolas) o entre distintos dispositivos a través de una red.

Mediante esta interacción simultánea, podemos habilitar a varios jugadores para que participen de forma colaborativa o competitiva en un mismo videojuego.

Cuando hablamos de juegos multijugador, nos referimos a aquellos que poseen cualquier modalidad en la que se tiene en cuenta la interacción de dos o más jugadores al mismo tiempo. Esta puede ser tanto de manera física en un único dispositivo, a través de diferentes dispositivos mediante cables o conexión inalámbrica o mediante servicios en línea u otro tipo de red con personas conectadas a la misma. La interacción puede ser en tiempo real (interacción simultánea) o por turnos (tan solo un jugador realiza las acciones mientras el resto esperan).

El hecho de añadir una opción multijugador es un factor importante a tener en cuenta en el diseño de un videojuego. Desde la perspectiva del desarrollo, tiene implicaciones importantes en su diseño y el código que hay que generar. Pero desde la perspectiva del marketing, esto también puede aumentar su tiempo de vida, es decir, el tiempo que los jugadores continúan interesados y activos en él. Por tanto, puede aumentar el valor que el usuario percibe de un juego.

A continuación, veremos las dos formas de integrar este modo en un videojuego, tanto la modalidad donde los jugadores se encuentran conectados a la misma máquina (modo local), como la modalidad donde cada uno lo hace desde su dispositivo (modo remoto).

¿Antes o después?

La inclusión de la modalidad multijugador se debe definir ya desde el inicio del proyecto.

También cabe destacar que hay videojuegos, aunque sean en un porcentaje bajo, que permiten una modalidad mixta entre multijugador local y multijugador remoto, permitiendo a jugadores en un mismo dispositivo competir contra otros jugadores a distancia, como por ejemplo un dos contra dos en algunos juegos *FIFA* de Electronics Arts o incorporar dos jugadores en el mismo equipo en una sesión de *deathmatch* en algunas entregas de la saga Halo.

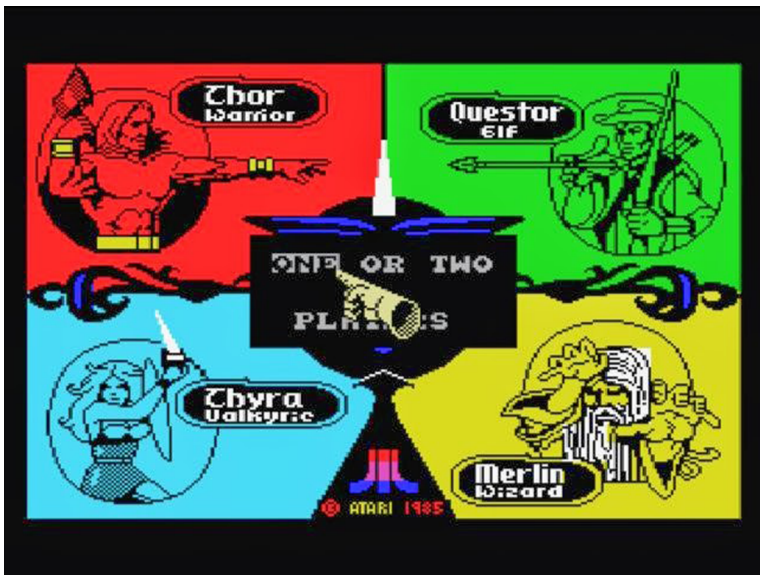
1.1. El multijugador local

Es obvio que el mecanismo más simple de crear un juego multijugador es permitir que diferentes jugadores puedan participar simultáneamente a través del mismo dispositivo. Esta modalidad de juego ha sido muy importante desde los inicios en diferentes plataformas y a menudo ofrecía un valor añadido sobre los juegos de un solo jugador.

Así pues, algunos de los primeros juegos de la historia ya incorporaban la opción de juego multijugador en modo local. En realidad, se trataba de una simplificación, para no tener que contemplar que fuera el ordenador quien dirigiera las acciones de nuestro oponente. Algunos ejemplos de juegos muy primerizos son *Tennis for Two* (1958), *Spacewar!* (1962) o el archiconocido *Pong* (1972).

1.1.1. Ordenadores domésticos

Se denomina *computadora doméstica* u *ordenador doméstico* a la segunda generación de computadoras, que entraron en el mercado con el nacimiento del Altair 8800, en un lapso de tiempo que se extiende hasta principios de la década de los noventa. Esto engloba a todas las computadoras de 8 bits y a la primera remesa de equipos con CPU de 16 bits.



Gauntlet (1985)

Estos ordenadores son considerados los padres de las videoconsolas que vendrían a posteriori a ocupar su lugar y por ende los primeros en llegar a los hogares. Su enfoque familiar invitaba a los jugadores a comprar más dispositivos de entrada, como *joysticks* o mandos para poder jugar a juegos multijugador.

1.1.2. Máquinas Arcade

Durante las épocas doradas de los años ochenta y principios de los noventa, las máquinas Arcade fomentaron el multijugador local con títulos a dos jugadores (*Pong*) o incluso a cuatro (*Gauntlet*) en los salones recreativos de la época.

Máquina recreativa de Beast Bursters para tres jugadores simultáneos

Street Fighter II The World Warrior

Estas máquinas ofrecían diferentes dispositivos de entrada para cada jugador y, obviamente, los géneros que más se prestaban para ello eran los juegos de lucha (saga *Street Fighter*), los *beat'emup* (*Teenage Mutant Ninja Turtles*) y otros mucho menos populares en otras plataformas, como los *shooter* sobre raíles (por ejemplo, *Virtua Cop*).

Teenage Mutant Ninja Turtles

En los últimos tiempos, estas máquinas se conectaban usando redes para poder hacer partidas multijugador en red, como por ejemplo *Sega Rally* o *Mario Kart*, ofreciendo la posibilidad de competir con jugadores del mismo salón recreativo en diferentes máquinas.

1.1.3. Videoconsolas

Una **videoconsola** o **consola de videojuegos** es un sistema electrónico de entretenimiento para el hogar que ejecuta videojuegos contenidos en diferentes tipos de almacenamiento.

Super Mario Kart

Claramente, las videoconsolas han sido durante más años la referencia en los hogares de los jugadores, instaurando desde sus inicios el multijugador local como una forma de jugar en familia o con amigos. Desde la Atari hasta la Wii U, podemos ver claramente cómo se ha fomentado este factor, pasando por grandes hitos, como la serie *Mario Kart* o el *shooter Goldeneye*, una de cuyas grandes bazas fue poder jugar hasta cuatro jugadores al mismo tiempo.

Super Nintendo MultiTap

Un detalle a tener en cuenta en esta evolución en las consolas de sobremesa fue que hasta la salida de la Nintendo 64, las consolas dieron soporte a cuatro jugadores a la vez, requiriendo *hubs* para mandos conocidos como *multitap*, que permitían jugar a dos jugadores más simultáneamente a títulos como *Super Bomberman*, que provenía de las Arcade, donde sí había máquinas que lo soportaban en la época.

Super Bomberman

1.2. El multijugador en red

Tecnologías de red

La inclusión de tecnologías de red en un juego se ha convertido en un elemento clave para incrementar las ventas y la calidad del mismo.

El origen de los juegos multijugador en red modernos proviene de los sistemas *mainframe* universitarios de la década de los setenta. Sin embargo, no fue hasta la llegada del acceso a Internet a mediados de los noventa cuando empezó a destacar verdaderamente. El potencial ofrecido por Internet como herramienta de comunicación global ha revolucionado el uso de las tecnologías, principalmente porque proporciona acceso a un 70 % de la población de los países más desarrollados. Los videojuegos no son una excepción. Los juegos en red han tenido una evolución muy clara paralelamente a las innovaciones en telecomunicaciones, y resulta muy interesante analizar cada etapa relacionada con las tecnologías emergentes en su momento.

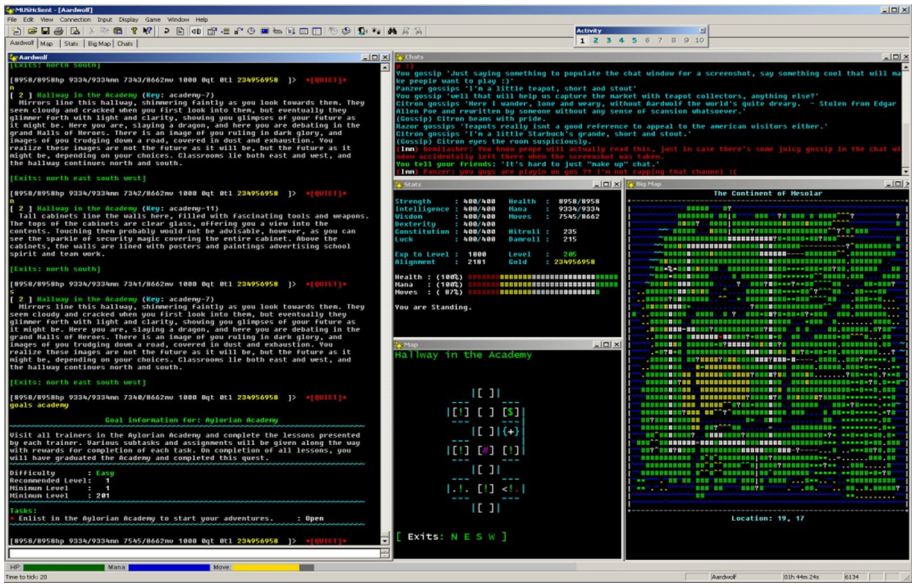
Por tanto, dependiendo de la época, cada juego ha ofrecido diferentes opciones para el multijugador, desde las pequeñas redes locales hasta las redes de ámbito global, que han permitido a jugadores de diferentes partes del mundo jugar al mismo videojuego simultáneamente.

1.2.1. Mazmorras multiusuario

Las **mazmorras multiusuario** o **MUD** (del inglés *Multi-User Dungeon*) son juegos multijugador donde diversos jugadores se encuentran conectados al mismo mundo virtual al mismo tiempo a través de una terminal.

El funcionamiento de estos juegos requiere la creación de un avatar por parte del jugador y, mediante la exploración de un mundo, debe desarrollar las estadísticas asociadas.

Este tipo de juego se hizo popular en los *mainframes* de las principales universidades. El término *MUD* se originó en el juego del mismo nombre de 1978, creado por Rob Trushaw en la Universidad de Essex. En cierto modo, los MUD pueden ser considerados como una primera versión de ordenador del juego de rol *Dungeons and Dragons* (*Dragones y Mazmorras*), aunque no todos los MUD son necesariamente juegos de rol.



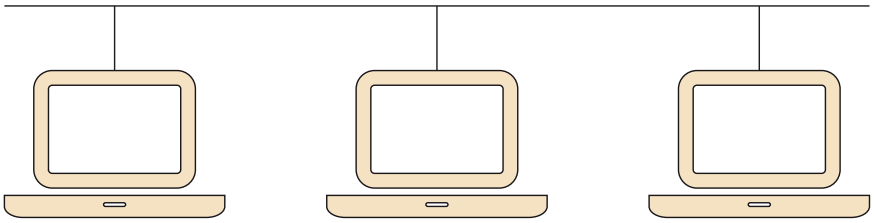
MUD Aardwolf (1996)

Cuando los ordenadores personales se hicieron más potentes, los fabricantes de hardware comenzaron a ofrecer módems que permitían que dos ordenadores se comunicaran entre sí a través de líneas telefónicas estándar. Aunque las tasas de transmisión fueron extraordinariamente lentas comparadas con los estándares modernos, esto permitió a los MUD ser jugados fuera del ámbito universitario.

1.2.2. Juegos en LAN

A principio de los años noventa apareció la posibilidad de utilizar las conexiones de red de los ordenadores personales para proporcionar un nuevo tipo de videojuegos. En esa época el acceso a la red se realizaba mediante el uso de módems o a través de una LAN (red de área local, o *Local Area Network* en inglés) mediante el protocolo IPX de Novell, lo que limitaba la cantidad de información que se podía transferir entre ordenadores.

Esquema de la arquitectura LAN



El protocolo IPX (*Internet Packet eXchange*) fue uno de los primeros que se utilizó para interconectar ordenadores en una red de área local. Al final acabó desapareciendo porque no era un protocolo óptimo para comunicar un gran número de ordenadores, al contrario que TCP/IP.

El primer juego en dar soporte multijugador LAN fue *Doom* (1993), estableciendo las bases de los juegos en red modernos. La versión inicial de este *shooter* en primera persona de id Software soportaba hasta cuatro jugadores en una sola sesión de juego, con la opción de jugar de forma cooperativa o en una competitiva bautizada como *deathmatch*.

Doom

Hay que destacar que *Doom* fue un juego de acción con un ritmo rápido, que requería de la aplicación de varios de los conceptos clave de comunicación en red.

Por supuesto, estas técnicas han evolucionado mucho desde 1993, pero la influencia de *Doom* sigue siendo muy notoria.

Muchos juegos multijugador con soporte para LAN también admitían otras modalidades, como por ejemplo la conexión por módem. Durante muchos años, la gran mayoría de juegos en red siguió dando soporte al juego en LAN. Esto condujo a la aparición de eventos como las *LAN Party*, en las que los jugadores se reunían en un lugar y conectaban sus ordenadores para jugar a juegos en red.

Lectura recomendada

Para mayor detalle sobre la historia y la creación de *Doom*, recomendamos el libro *Masters of Doom* (2003), listado en la bibliografía de este módulo didáctico.

Modo multijugador en LAN de *Doom* (1993)

Game Link Cable

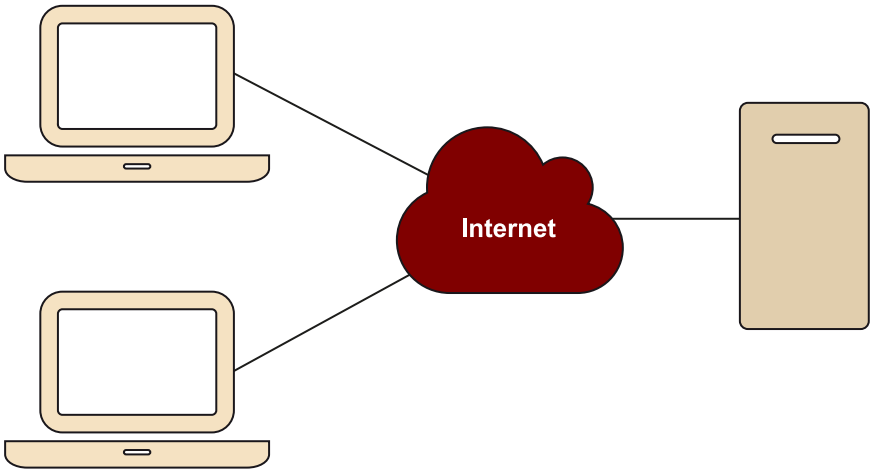
La primera aproximación de las videoconsolas en el mundo de los videojuegos multijugador en red se produjo con la introducción de la consola portátil Nintendo Game Boy. La consola llevaba incorporado un puerto serie que permitía conectar hasta cuatro dispositivos mediante el *Game Link Cable*, y algunos de los juegos más populares (como por ejemplo el *Tetris*) ofrecían la posibilidad de competir con otros usuarios. De todos modos, el despliegue de redes de área local para la interconexión de consolas no es el caso más habitual, por sus características de dispositivo para el hogar.

Aunque siguen habiendo juegos multijugador en red con soporte para LAN, la tendencia en los últimos años, tanto de los sistemas como de los desarrolladores, parece enfocarse exclusivamente en la modalidad de multijugador en línea.

1.2.3. Juegos en línea

En un juego en línea (*online*), los jugadores se conectan entre sí a través de una gran red de ordenadores separados geográficamente. Hoy en día, los juegos en línea son sinónimo de juegos en Internet, aunque el término *en línea* es un poco más amplio y puede incluir algunas de las redes anteriores que, inicialmente, no se conectaban a Internet.

Esquema de la arquitectura WAN



A medida que Internet comenzó a despegar a finales de 1990, los juegos en línea siguieron el mismo camino ascendente. Algunos de los juegos populares en aquellos años fueron *Quake*, de id Software (1996), y *Unreal*, de Epic Game (1998). En ambos casos, se trata de juegos de disparos en primera persona (FPS, *First Person Shooter*), al igual que *Doom*.

Aunque pueda parecer que un juego en línea puede ser implementado de la misma forma que un juego LAN, no es así. Por ello, la incorporación de conectividad vía Internet supuso un nuevo paso en la evolución de los juegos multijugador desde el punto de vista de su arquitectura.

Quake (1996)

De hecho, como anécdota interesante, la versión inicial de *Quake* en realidad no estaba diseñada para trabajar mediante una conexión a Internet. Hasta el lanzamiento de una actualización, concretamente el parche *QuakeWorld*, la modalidad multijugador no era fiable a través de Internet. Los métodos para compensar la latencia los trataremos a lo largo de la asignatura.

Latencia

Una consideración muy importante es la **latencia**, o la cantidad de tiempo que necesitan los datos para viajar por la red.

En el campo de las videoconsolas poco a poco fueron apareciendo algunas aproximaciones para añadir componentes hardware de red, a medida que fueron proliferando los servicios de banda ancha. La Sega Dreamcast (1999) dispuso de un adaptador de red y fue una de las primeras en intentar capitalizar el creciente mercado de los juegos en línea.

Sin embargo, en aquella época los servicios de Internet eran demasiado limitados a nivel mundial y no se contaba con los servicios de banda ancha actuales. La Dreamcast utilizaba conexiones vía telefónica para dar el servicio de juego en línea, de manera que el sistema incluía un módem, por lo que su éxito fue muy limitado.

El destino de Dreamcast

La innovación del juego en línea que proponía Sega no fue suficiente para conquistar el mercado de videoconsolas.

Xbox Live desde el Dashboard de la primera Xbox

Este fue un punto de inflexión importante, ya que con la llegada de los servicios en línea iniciado por Microsoft con Xbox Live (2002) en su primera consola de sobremesa, la Xbox original, se dio inicio a la era de los servicios multijugador.

Windows en consolas

Como reemplazo en el mundo de las consolas, Sega dejó paso a Microsoft, con quien se había asociado usando el sistema operativo (Windows CE) de esta en su última consola.

A medida que la proliferación de los videojuegos se ha ido ampliado en el panorama móvil, los juegos multijugador han crecido por igual en este entorno, donde, por las características del dispositivo, el multijugador local no tiene demasiado sentido (a menos que hablemos de juegos para tableta). Muchos de los juegos multijugador en estas plataformas son asíncronos, típicamente basados en turnos, que no requieren la transmisión en tiempo real de los datos.

Juegos asíncronos

Obviamente, el modelo asíncrono ha existido desde el principio de los juegos multijugador en red, como por ejemplo *Space Crusade*.

En este modelo, los jugadores son notificados cuando llega su turno, y tienen una gran cantidad de tiempo para hacer su movimiento. Un ejemplo de juego conocido para móviles que utiliza la modalidad de multijugador asincrónico es *Words with Friends* (2009). Desde un punto de vista técnico, un juego en red asíncrono es más sencillo de implementar uno en tiempo real.

Originalmente, se utilizó el modelo asíncrono en juegos para móviles por la necesidad de fiabilidad en las redes móviles, comparada con las conexiones por cable. Sin embargo, con la proliferación de dispositivos y mejoras en redes Wi-Fi, es ya común ver juegos en tiempo real en estos dispositivos. Un ejemplo de un juego para móviles que se aprovecha de la comunicación de red en tiempo real es *Hearthstone: Heroes of Warcraft* (2014).

Soporte para móvil

Cabe destacar que en los últimos años también se han creado plataformas de servicios en línea (iOS Game Center o Google Play Games) que dan soporte a los juegos multijugador para dispositivos móviles.

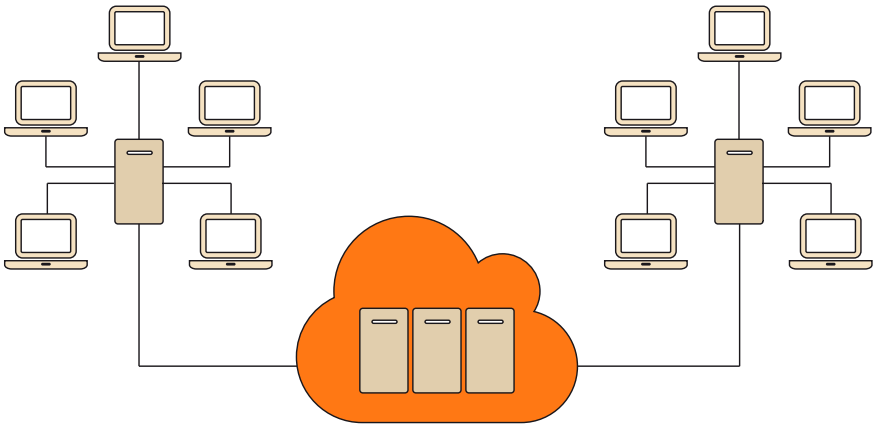
1.2.4. Juegos multijugador masivos

El siguiente paso en la evolución de los juegos de red se produjo con la aparición de los primeros juegos en línea persistentes alrededor de 1995, clara evolución de los MUD. Incluso hoy en día, la mayoría de los juegos multijugador en línea están limitados a un pequeño número de jugadores por partida, con un número de jugadores admitidos entre dos y sesenta y cuatro. Sin embargo, no cientos, sino miles de jugadores pueden participar en una sola sesión de juego en línea persistente.

En su origen, estos juegos combinaban parte de los dos puntos anteriores: tenían un servidor central donde los usuarios se conectaban para jugar, y, además, disponían de un cliente gráfico que permitía interactuar de forma intuitiva con el entorno y los otros jugadores. Este tipo de juegos fueron llamados *Massively Multiplayer Games* (juegos multijugador masivos).

Este tipo de juegos no pueden jugarse de forma local, debido a que la mayor parte de la lógica del juego se encuentra en los servidores, y requieren una conexión permanente con el servidor cada vez que queramos jugar una partida. Además, estos servidores se encuentran distribuidos por zonas, para poder dar servicio a cada localización por separado, aunque también disponen de servidores centralizados.

Esquema de la arquitectura Cloud



Arquitectura de un MMO

La arquitectura de un MMO es un desafío técnicamente complejo, que va más allá del propio desarrollo del mismo y requiere conocimientos de *backend* y arquitecturas de redes y servidores.

Esto repercute, entre otras cosas, en el coste de mantenimiento de esta infraestructura de servidores. Por esta razón, los juegos persistentes introdujeron un nuevo modelo de negocio para los videojuegos basado en suscripciones mensuales.

World of Warcraft

La gran mayoría de estos juegos pertenecen al género de los MMORPG (*Massively Multiuser Online Role Playing Game*), y en muchos sentidos, los MMORPG pueden ser considerados como la evolución gráfica de las mazmorras multiusuario. Como ejemplos paradigmáticos podemos destacar el *Ultima Online* (1997), que fue el primero en establecerse y el que más tiempo lleva en el mercado, y el *World of Warcraft* (2004), que ha batido todos los récords sobre número de usuarios suscritos al superar los doce milllones de jugadores de todo el mundo.

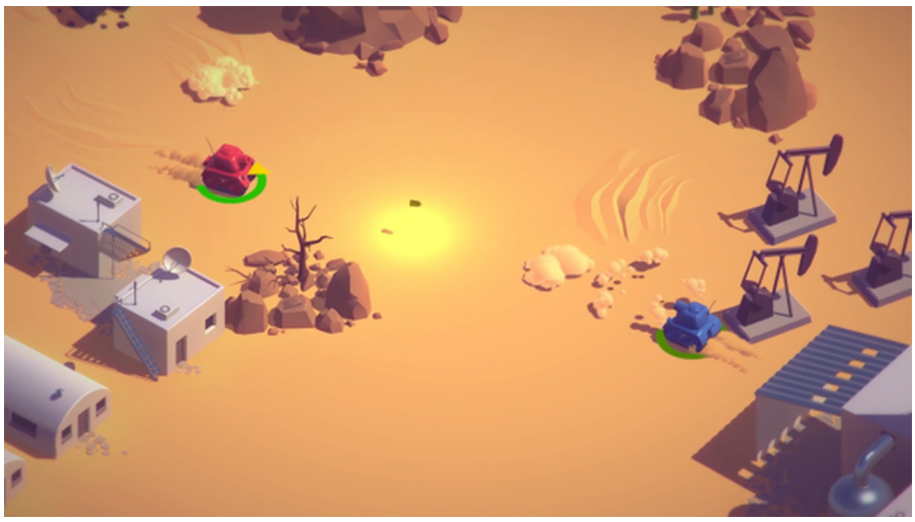
2. Proyecto: *Tanks!* local

En su mayor parte, los juegos multijugador locales pueden ser programados de la misma manera que los juegos de un solo jugador. Las únicas diferencias son típicamente múltiples puntos de vista y/o el apoyo a múltiples dispositivos de entrada. Dado que la programación de los juegos multijugador locales es muy similar a los juegos de un solo jugador, nuestro primer objetivo en esta asignatura será desarrollar un juego de este tipo.

Ved también

Al final del capítulo están las soluciones a todos los retos.

Tutorial de *Tanks!*



Código del ejemplo

Podéis encontrar este ejemplo en la Asset Store de Unity.

En este apartado revisaremos dos puntos clave para poder ofrecer esta modalidad en nuestros juegos: gestionar las múltiples entradas y estancias de jugador, así como aplicar la técnica de *split screen* (pantalla partida), proporcionando a cada jugador su propia cámara y espacio de pantalla.

Para ello trabajaremos a partir de un ejemplo oficial de Unity (*Tanks!*), al cual le hemos incorporado un modo de multijugador local básico.

2.1. Gestión de múltiples dispositivos de entrada

El requisito indispensable para un juego multijugador local es permitir a los jugadores utilizar cada uno un dispositivo de entrada distinto. Estos dispositivos no tienen por qué estar separados físicamente, como podría ser el ejemplo de juegos que destinan diferentes teclas de un teclado para cada jugador o en dispositivos móviles donde cada zona de la pantalla está destinada a recibir las entradas de un jugador.

Distintas entradas

Aunque por lo general y por ergonomía se prefiere la utilización de diferentes tipos de entradas, estas no tienen por qué ser del mismo tipo, por ejemplo, un jugador usando el teclado y otro un mando.

De cara a desarrollar un juego multijugador local, es importante que a nivel de programación seamos capaces de desarrollar el código necesario para que la entrada del usuario sea lo más genérica posible y permita ser configurada para dotarla de la mayor versatilidad.

Estas opciones no son triviales, si tenemos en cuenta la cantidad de diferentes dispositivos de entrada y de fabricantes, así como su comportamiento en los diferentes sistemas operativos, donde, por ejemplo, un mando de Xbox 360 no se comporta igual en la consola, en un ordenador con Windows o en un ordenador con MacOS, debido a los *drivers* y al mapeo de botones en cada uno, así como el acceso a las opciones avanzadas o no de este en cada entorno.

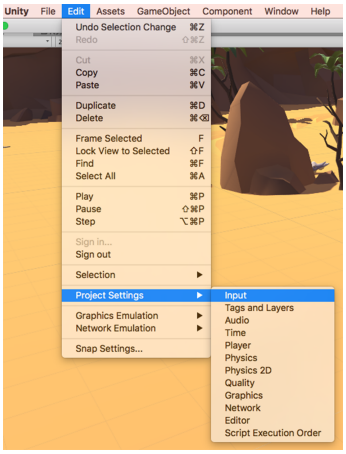
Mapping de botones del *pad* de Xbox 360 en Windows y en Mac

2.1.1. Ejemplo

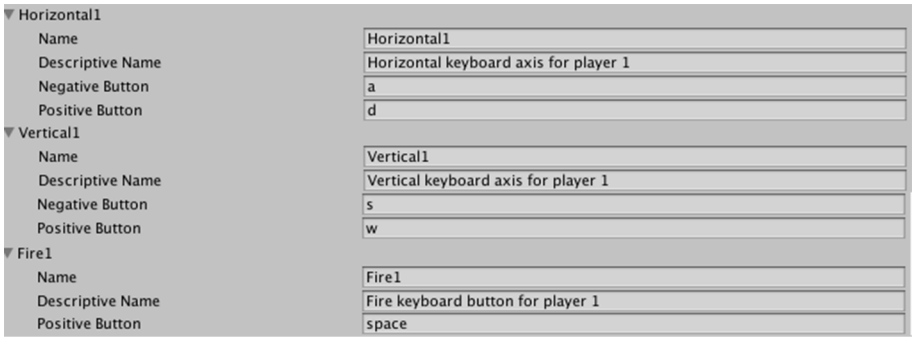
Dentro del código del proyecto, en la carpeta *complete/scripts* podéis encontrar los scripts *TankMovement* y *TankShooting* que permiten utilizar las diferentes entradas genéricas de botones que provienen de la configuración del juego añadiendo en tiempo de ejecución el número de jugador instanciado.

```
1.      m_MovementAxisName = "Vertical" + m_PlayerNumber;
2.      m_TurnAxisName = "Horizontal" + m_PlayerNumber;
3.      m_FireButton = "Fire" + m_PlayerNumber;
```

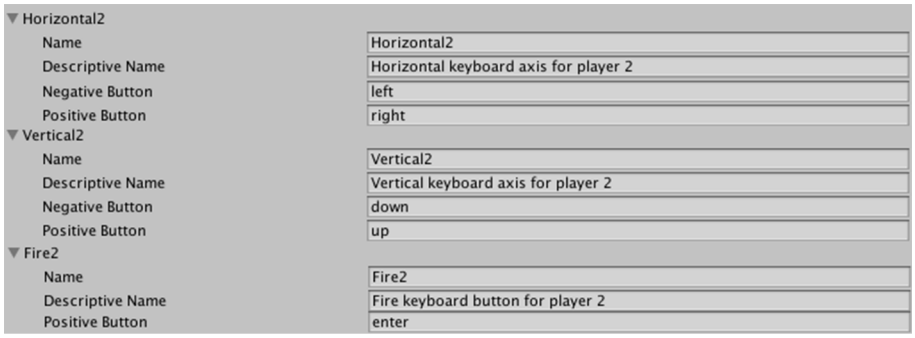
Como podemos ver en este apartado de configuración, en este proyecto tenemos ya definidos los controles de ambos jugadores.



A modo de ejemplo, podemos ver en la configuración de dispositivos de entrada en el menú de Edit > Project Settings > Input.



Por ejemplo, con la propiedad *Horizontal1* se definen los dos botones de movimiento (incrementar o decrementar) del movimiento horizontal del Jugador 1, o por otro lado, la propiedad de *Fire2* define el botón de disparo del Jugador 2.



2.1.2. Integración en el proyecto

No indicar los botones concretos en nuestro código y ceñirse a la API de Unity nos permite ser más versátiles y derivar los problemas de los diferentes dispositivos a fases más avanzadas del desarrollo, así como valorar el uso de otros dispositivos o *assets* muy elaborados que nos proporciona esta funcionalidad y problemáticas ya resueltas.

Reto 1: Habilitad una nueva acción secundaria (*AltFire*) para que cada jugador pueda disparar alternativamente con otro botón a elegir por vosotros.

Ejecutando el juego tal cual se os presenta la escena *Complete*, podéis comprobar el funcionamiento de este punto inicial.

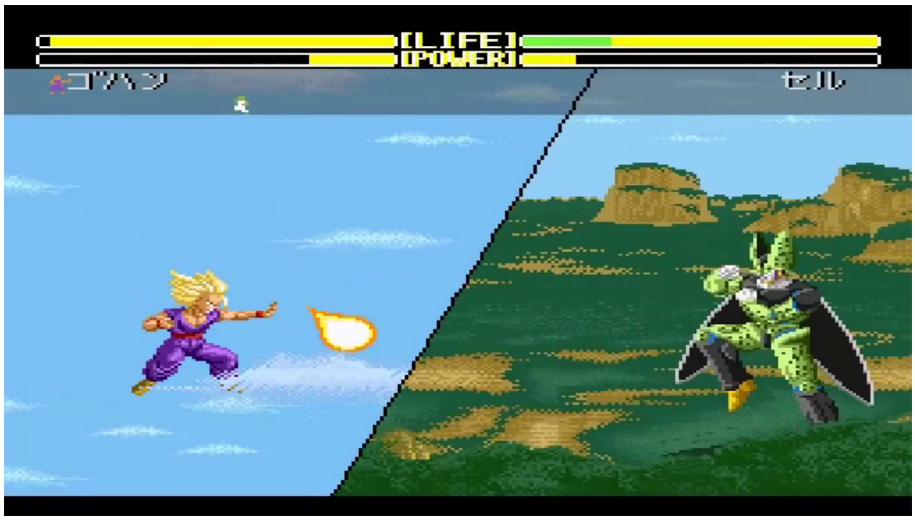
Reto 2: Modificad la entrada de acción secundaria (*AltFire*) para que los disparos del tanque sean proyectiles de color rojo y que además sean un 50 % más rápidos.

CompleteShell

Los proyectiles del juego son gestionados por el *prefab CompleteShell*. El color se puede controlar a través de su *Shader*.

2.2. Pantalla partida (*split screen*)

Esta opción también puede ser dinámica en juegos, especialmente en los juegos con perspectiva bidimensional, utilizándolo como recurso en el caso de que los jugadores se alejen lo suficiente para que el zoom no baste para mostrarlos a la vez en pantalla de forma satisfactoria.



DBZ Butoden 2 (1993)

Ejemplos de esta variante podrían ser la aplicada en la saga *Dragon Ball Butoden* de Super Nintendo o en juegos de las múltiples sagas *Lego*, donde se potencia el multijugador local.

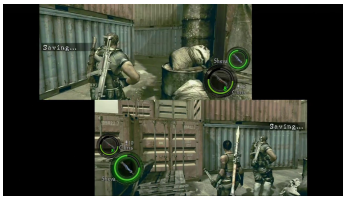
Lego Marvel Super Heroes (2013)

Aprovechando la tecnología de gafas 3D complementarias, permite a cada jugador ver a pantalla completa el juego.

En nuestra implementación buscaremos el mismo efecto para potenciar el uso de la pantalla completa cuando los jugadores se encuentren a una corta distancia, pero en el momento en que se detecte que se supera esta distancia, activaremos la segunda pantalla, dividiendo horizontalmente la pantalla.

Una mejora que vemos en este tipo de juegos es que la división de la pantalla es oblicua dependiendo de la posición entre los jugadores. Dejaremos esta mejora para los retos de este módulo.

Como curiosidad, es interesante saber que no siempre se ha permitido a los jugadores obtener el máximo de pantalla posible usando esta técnica, sino que hubo algunos casos en que se mantuvo la proporción de la pantalla individual con el *aspect ratio original* (por ejemplo, 16:9), como fue el caso del motor MT Framework (v1.4) de Capcom en juegos como *Resident Evil 5* o *Lost Planet 2*.



Resultado con la pantalla partida del modo cooperativo de *Resident Evil 5*.

2.2.1. Gestión de cámaras para pantalla partida

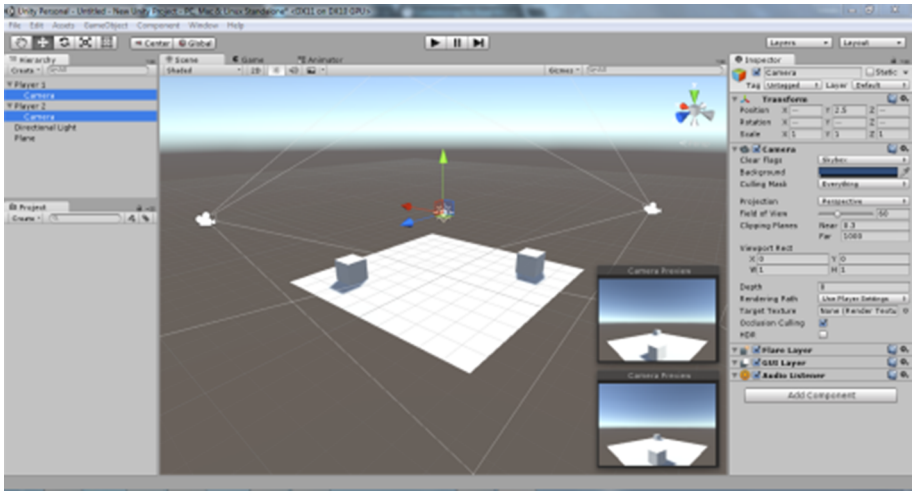
Aunque *a priori* nos pueda parecer complicado, en realidad es fácil crear un efecto de pantalla partida gracias a la propiedad *Viewport Rectangle* de cada cámara.

Concretamente, teniendo dos cámaras debemos revisar los valores de amplitud (W) y altura (H) de cada *Viewport Rectangle*. Esto nos permite, por ejemplo, que la cámara del primer jugador se muestre desde la mitad de la pantalla hasta la parte superior, mientras que la cámara del segundo jugador comienza en la parte inferior y para a medio camino encima de la pantalla.

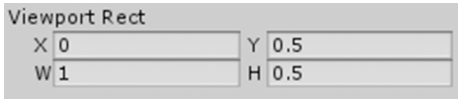
Viewport Rectangle

Se usa para especificar en qué parte de la pantalla se dibujará la vista de la cámara donde está definida.

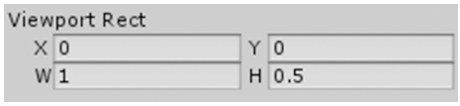
Vamos a poner en práctica esta idea, ya partiendo de un ejemplo muy inicial en Unity, donde tan solo debemos crear dos *GameObject*, como por ejemplo dos cubos, que representen a los jugadores, asociando una cámara a cada uno de ellos.



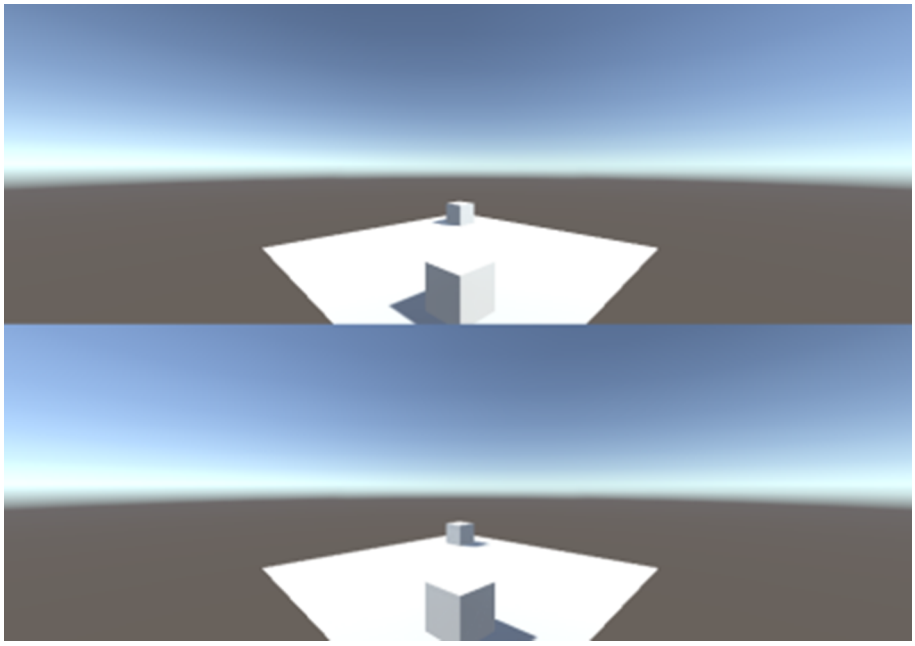
A continuación, vamos a darle los valores (X = 0, Y = 0.5, W = 1, H = 0.5) al *Viewport Rect* de la cámara del Jugador 1.



El siguiente paso es configurar el *Viewport Rect* para que las dimensiones de la cámara del jugador sean (X = 0, Y = 0, W = 1, H = 0.5). Finalmente, debemos darle los valores (X = 0, Y = 0, W = 1, H = 0.5) al *Viewport Rect* de la cámara del Jugador 2.



Y así es como, gracias a la redimensión de lo que muestra cada cámara de cada jugador, conseguimos obtener el resultado esperado en cualquiera de nuestros proyectos:



Resultado final al aplicar la técnica de pantalla partida

Esta misma técnica se puede aplicar tal cual a nuestro proyecto de lucha de tanques, de modo que ambos jugadores tengan su zona de pantalla individual.

Reto 3: Haced que cada *Viewport* de cada jugador sea proporcional a la pantalla original, alineada la pantalla del primer jugador a la izquierda y la del segundo jugador a la derecha, como fue el caso del motor MT Framework (por ejemplo, *Resident Evil 6*).

2.2.2. Pantalla partida desplegada dinámicamente

Una vez entendemos el concepto básico y sabemos dónde debemos modificar un juego para aplicarle esta técnica, pasamos a realizar la modificación del ejemplo que estamos estudiando en este apartado.

Antes de empezar, debemos considerar que es más adecuado realizar la modificación del proyecto vía código en tiempo de ejecución que en el editor, como hemos visto anteriormente. De esta forma, tenemos una solución más

flexible, podemos activarla cuando queramos, o también si programamos el *script* para dividir la pantalla para dos jugadores, podemos hacer lo mismo para cuatro jugadores.

En el tutorial original en el que nos basamos ya existe el *script GameManager*, en el que, mediante la función *SpawnAllTanks()*, se realiza el *spawn* de todos los tanques. El primer paso es modificar esta función para que tenga acceso a la cámara principal y única hasta el momento:

```
1.      Camera mainCam =
2.         GameObject.Find ("Main Camera").GetComponent<Camera>();
```

La idea es usarla para realizar réplicas para cada uno de los jugadores mediante una función llamada *AddCamera(..)*, que implementaremos más tarde y que recibe como parámetro el número de jugador y la cámara original:

```
3.      AddCamera(i, mainCam);
```

Y finalmente desconectamos la cámara inicial:

```
4.      mainCam.gameObject.SetActive(false);
```

El código final quedaría de la siguiente forma:

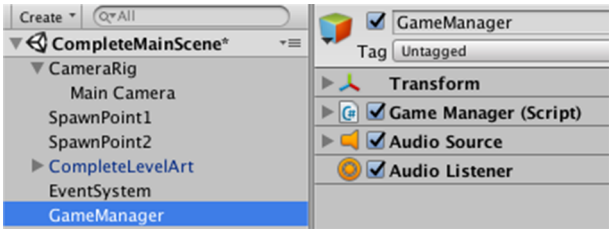
```
1.      private void SpawnAllTanks()  {
2.          Camera mainCam =
3.            GameObject.Find ("Main Camera").GetComponent<Camera>();
4.
5.          for (int i = 0; i < m_Tanks.Length; i++) {
6.              m_Tanks[i].m_Instance =
7.                  Instantiate(m_TankPrefab,
8.                  m_Tanks[i].m_SpawnPoint.position,
9.                  m_Tanks[i].m_SpawnPoint.rotation)
10.                 as GameObject;
11.             m_Tanks[i].m_PlayerNumber = i + 1;
12.             m_Tanks [i].Setup();
13.             AddCamera(i, mainCam);
14.         }
15.         mainCam.gameObject.SetActive(false);
16.     }
```

Reto 4: Activad la técnica de pantalla partida cuando los tanques estén a un mínimo de separación y desactivadla cuando se reduzca la distancia por debajo del mismo límite.

La función *AddCamera(..)* nos va a permitir replicar la cámara principal y además modificar el *Viewport* para darle a la visión del jugador la proporción de pantalla que le pertoque. En este caso, simplemente aplicamos la división horizontal al aplicar de forma fija el *Split Screen* para dos jugadores:

```
1.      private void AddCamera(int i, Camera mainCam) {
2.
3.          GameObject childCam = new GameObject( "Camera"+(i+1));
4.          Camera newCam = childCam.AddComponent<Camera>();
5.          newCam.CopyFrom(mainCam);
6.          childCam.transform.parent =
7.              m_Tanks[i].m_Instance.transform;
8.
9.          if (i==0) newCam.rect = new Rect(0.0f, 0.5f, 1.0f, 0.5f);
10.         else newCam.rect = new Rect(0.0f, 0.0f, 1.0f, 0.5f);
11.
12.     }
```

Un último detalle a tener en cuenta al realizar este proceso es que debemos cambiar el componente de *AudioListener* (que debe ser único y existir) de emplazamiento (actualmente en la cámara principal) para que este no se duplique al clonarla. Una opción sencilla es moverlo al *GameManager*:



Al ejecutar de nuevo, podremos ver que el resultado es el esperado, con la pantalla superior destinada a la cámara del Jugador 1 y la inferior al Jugador 2:

Reto 5: Haced que cada cámara esté enfocando al tanque de cada jugador y lo siga haciendo cuando estos se muevan.

Hacer la modificación vía *script* y no vía editor, al igual que el resto del ejemplo, permite que nuestra ampliación de este proyecto se mantenga suficientemente genérica para poder ser extensible y fácilmente mantenible.

Número de jugadores

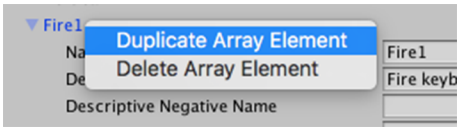
En caso de querer habilitar el juego para un número distinto de jugadores deberemos revisar de nuevo la función *AddCamera(..)*, así como la lógica del juego involucrada para poder dar soporte a este nuevo requerimiento.

Reto 6: Generad el proyecto correctamente para ejecutarlo como una aplicación en vuestro sistema operativo y modificad los controles de entrada del jugador.

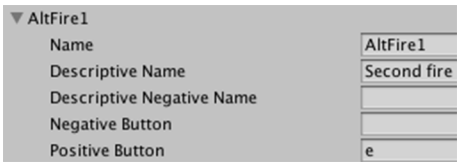
2.2.3. Soluciones a los retos propuestos

1) Reto 1

Como hemos visto, desde el editor de entradas podemos añadir una nueva entrada para detectar el nuevo tipo de disparo, que podemos llamar *AltFire1* y *AltFire2* (respectivamente).



De esta forma tendremos este *input* disponible y preconfigurado, con lo cual nuestro trabajo aquí consiste en hacer que al detectar esta acción se ejecute el comportamiento esperado.



El último paso consiste en modificar el *script* de comportamiento del disparo de los tanques (*TankShooting*) para que tenga en cuenta también esta nueva acción:

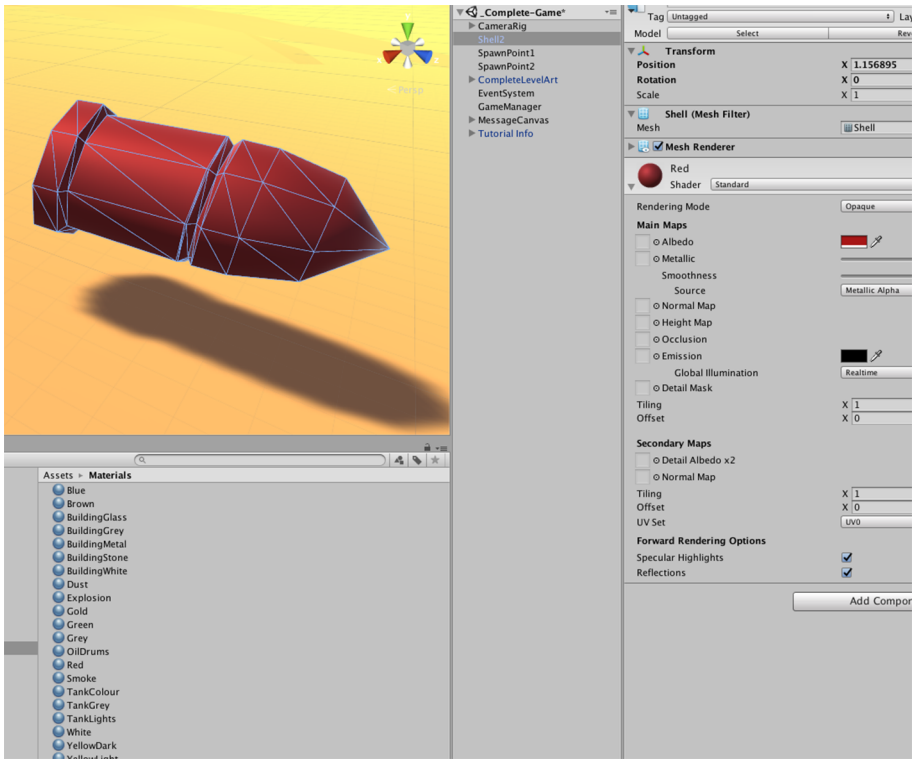
```
1.      private string m_AltFireButton;
2.
3.      private void Start () {
4.          m_AltFireButton = "AltFire" + m_PlayerNumber;
5.      }
```

Y también modificar los accesos a los botones de fuego y centrarlizarlos en una función que deduzca los diferentes estados:

```
6.    private bool FireButton(int mode) {
7.        bool state = false;
8.        switch (mode) {
9.            case 0:
10.               state = Input.GetButtonDown(m_FireButton)
11.                   || Input.GetButtonDown(m_AltFireButton);
12.           break;
13.           case 1:
14.               state = Input.GetButton(m_FireButton)
15.                   || Input.GetButton(m_AltFireButton);
16.           break;
17.           case 2:
18.               state = Input.GetButtonUp(m_FireButton)
19.                   || Input.GetButtonUp(m_AltFireButton);
20.           break;
21.       }
22.       return state;
23.   }
```

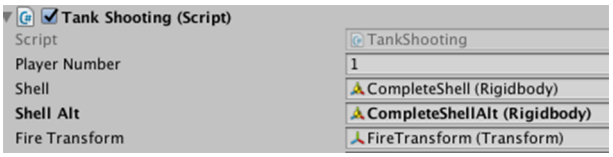
2) Reto 2

Antes de nada vamos a necesitar un nuevo modelo de proyectil rojo, que podemos crear fácilmente a partir del actual duplicando el *prefab* (*CompleteShell*) y cambiando el *shader* actual (*Gold*) por el *shader* de color rojo (*Red*).



A continuación volvemos al *script TankShooting script* para modificar el comportamiento de nuevo. Para ello vamos a necesitar una propiedad para el nuevo *prefab* (*CompleteShellAlt*) y guardar el tipo de disparo que acabamos de realizar:

```
1.      public Rigidbody m_ShellAlt;
2.      private bool m_AltFire;
```



Seguidamente necesitamos modificar la función *FireButton()* para discriminar entre una acción y otra:

```
1.      private bool FireButton(int mode) {
2.          bool action = false;
3.          m_AltFire = false;
4.
5.          switch (mode) {
6.              case 0:
7.                  action = Input.GetButtonDown(m_FireButton)
8.                  m_AltFire = Input.GetButtonDown(m_AltFireButton);
9.              break;
10.             case 1:
11.                 action = Input.GetButton(m_FireButton)
12.                 m_AltFire = Input.GetButton(m_AltFireButton);
13.             break;
14.             case 2:
15.                 action = Input.GetButtonUp(m_FireButton)
16.                 m_AltFire = Input.GetButtonUp(m_AltFireButton);
17.             break;
18.         }
19.         return action||m_AltFire;
20.     }
```

Para finalizar, modificamos levemente la función *Fire()* para que los disparos diferencien entre los tipos de proyectil y la velocidad asociada a cada una de ellas:

```
1.      private void Fire ()  {
2.          m_Fired = true;
3.          Rigidbody shellInstance;
4.
5.          if (m_AltFire) shellInstance =
6.              Instantiate (m_ShellAlt, m_FireTransform.position,
7.                  m_FireTransform.rotation) as Rigidbody;
8.          else
9.              shellInstance = Instantiate (m_Shell,
10.                 m_FireTransform.position, m_FireTransform.rotation)
11.                 as Rigidbody;
12.         shellInstance.velocity =
13.             m_CurrentLaunchForce * m_FireTransform.forward;
14.
15.         if (m_AltFire) shellInstance.velocity *= 1.50f;
16.
17.         m_ShootingAudio.clip = m_FireClip;
18.         m_ShootingAudio.Play();
19.         m_CurrentLaunchForce = m_MinLaunchForce;
20.     }
```

3) Reto 3

Tenemos que recalcular que el *Viewport* de cada jugador sea proporcional a la pantalla original, y la pantalla del primer jugador esté alineada a la izquierda y la del segundo a la derecha, como fue el caso del motor gráfico.

En este caso, partiendo del *aspect ratio* 16:9 debemos recalcular la amplitud y altura de cada pantalla y reconfigurar la función *AddCamera()* de *GameManager*, donde se establecen estos parámetros al crear cada cámara:

```
1.      if (i==0) newCam.rect = new Rect (0.0f, 0.5f, 0.89f, 0.5f);
2.      else newCam.rect = new Rect (0.11f, 0.0f, 0.89f, 0.5f);
```

Al reducir cada cámara a una altura de 0.5, proporcionalmente la longitud debe ser de 0.5 / 9 × 16 = 0.89 para ajustarnos al *aspect ratio* 16:9. Finalmente, hay que alinear la segunda cámara al lado derecho desplazando su posición en el eje X, restando al total (1) la nueva longitud de 0.89.

Aunque el resultado es el esperado, obviamente este cálculo debería ser dinámico y permitir ajustar las dimensiones para diferentes *aspect ratios* que pueda tener el juego.

4) Reto 4

Activar la técnica de pantalla partida cuando los tanques estén a un mínimo de separación y desactivarla cuando se reduzca la distancia por debajo del mismo límite requiere modificar el *script CameraControl* para que tenga en cuenta las diferentes cámaras que hay, tanto la original como las nuevas que hemos añadido por cada tanque.

De esta forma este *script* será el encargado de activar y desactivar cada cámara en el momento adecuado. Le añadiremos las propiedades para mantener el modo de **cámara partida** sincronizado, la distancia límite y una distancia extra de histéresis para que en caso de estar bordeando el límite no haya cambios bruscos continuamente.

```
1.      private bool splitMode = true;
2.      public float limitDist = 25.0f;
3.      public float hysteresis = 5.0f;
```

A continuación añadimos una nueva función, llamada *Split()*, para controlar en cada actualización si hay que activar o desactivar el modo:

```
4.      private void FixedUpdate () {
5.          Move();
6.          Zoom();
7.          Split();
8.      }
```

Finalmente, implementamos esta función de forma sencilla, activando o desactivando la cámara principal, que cogerá el relevo en caso de activarla o dejará paso a las otras cámaras en caso de desactivarla:

```
9.      private void Split() {
10.
11.         float distance =
12.             Vector3.Distance(m_Targets[0].position,
13.                  m_Targets[1].position);
14.
15.         if (splitMode && distance < limitDist-hysteresis) {
16.             splitMode = false;
17.             m_Camera.gameObject.SetActive(true);
18.         }
19.         else if (!splitMode && distance > limitDist+hysteresis) {
20.            splitMode = true;
21.            m_Camera.gameObject.SetActive(false);
22.         }
23.     }
```

5) Reto 5

En este caso, vamos a crear un nuevo *script* llamado *CameraControl*, que asociaremos directamente al *prefab CompleteTank* para que de esta forma cada tanque tenga su propio control de cámara asociado y podamos tener la máxima flexibilidad posible para determinar el comportamiento de esta.

```
1.      private Camera m_Camera;
2.
3.      private void FixedUpdate () {
4.          if (m_Camera == null) {
5.              m_Camera = GetComponentInChildren<Camera>();
6.              return;
7.          }
8.          Follow();
9.      }
10.
11.     private void Follow()  {
12.         m_Camera.transform.position = Vector3.Lerp(
13.             m_Camera.transform.position, transform.position,
14.                Time.deltaTime * smooth);
15.     }
```

Recuperando la cámara asociada al tanque y recalculando su posición veremos que conseguimos ver cada tanque en medio de la pantalla. El problema que detectamos en este punto es que la cámara acaba acercándose demasiado a la posición del tanque, hasta llegar a ocupar su posición.

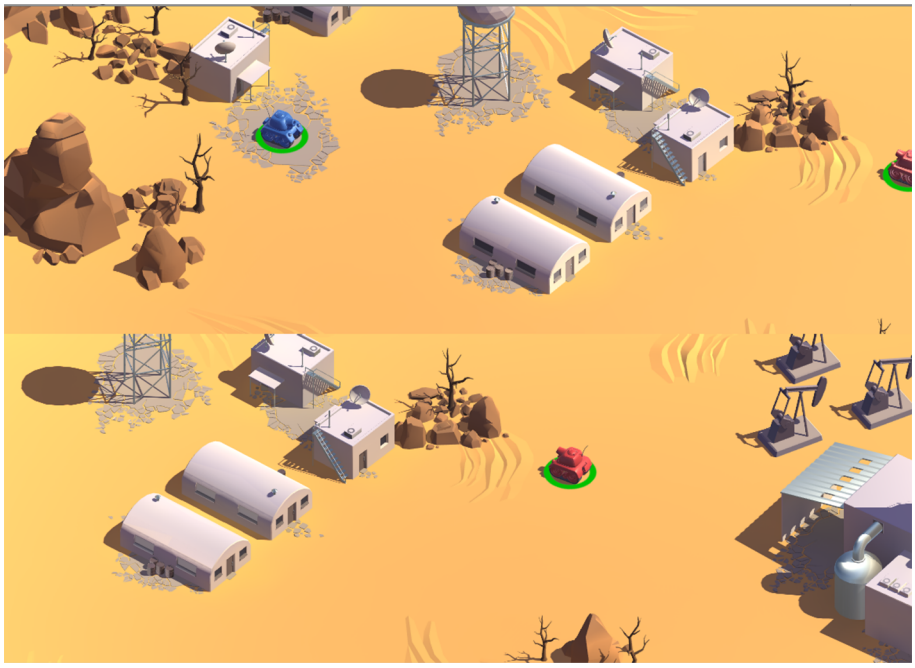
Además, debemos tener en cuenta que se trata de una cámara ortográfica y que la altura debe ser suficiente para que toda la escena entre dentro del *frustum*. Para ello mantendremos un límite de acercamiento entre la cámara y el tanque en el *CameraControl*:

Frustum

Es la región entre los dos planos de la cámara que define el espacio del mundo que aparecerá en la pantalla.

```
1.      private Camera m_Camera;
2.      public float smooth = 0.5f;
3.      public float limitDist = 20.0f;
4.
5.      private void FixedUpdate () {
6.          if (m_Camera == null) {
7.              m_Camera = GetComponentInChildren<Camera>();
8.              return;
9.          }
10.         Follow();
11.     }
12.
13.     private void Follow()  {
14.         float currentDist =
15.             Vector3.Distance(transform.position,
16.                 m_Camera.transform.position);
17.         if (currentDist > limitDist)
18.             m_Camera.transform.position = Vector3.Lerp(
19.                m_Camera.transform.position, transform.position,
20.                   Time.deltaTime * smooth);
21.     }
```

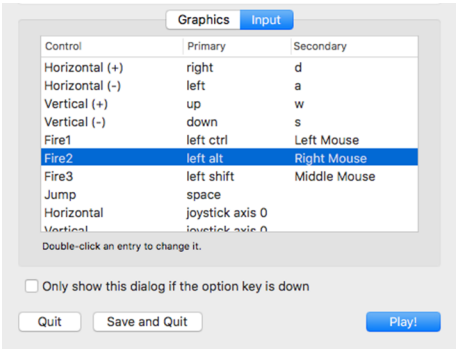
De esta forma, ahora la cámara cambia de posición de forma progresiva, según la posición que adquiera el *GameObject* del jugador, sin acercarse más de lo que le permitimos.



6) Reto 6

Antes de nada, generad el proyecto correctamente para ejecutarlo como una aplicación en vuestro sistema operativo a partir de la escena *CompleteScene*.

Para modificar los controles de entrada una vez generado el juego, Unity nos permite editar estos controles mediante su pantalla de arranque del juego:



Además, desde aquí podéis configurar otros dispositivos externos, diferentes al teclado para jugar al juego. A partir de este punto ya podéis probar el ejemplo completo en local para dos jugadores simultáneos finalizando el primer proyecto de multijugador.

Resumen

En este módulo hemos realizado una primera aproximación al mundo de los videojuegos multijugador a partir de la evolución de esta modalidad en los videojuegos a lo largo de la historia, tanto el multijugador en red como el multiugador local, intentando abarcar los diferentes dispositivos. Por último, hemos visto el desarrollo y las técnicas para el multijugador aplicados a un proyecto práctico.

La evolución lógica del multijugador como modalidad en el mundo de los videojuegos es transversal a todos los dispositivos de ocio que han surgido hasta ahora. Al inicio, ayudando a la popularización de estos, tuvo un papel muy importante el multijugador local, evolucionando luego al multijugador en red.

Hemos hecho hincapié en videojuegos icónicos en esta evolución, dada su aportación e influencia en esta área. Cabe destacar especialmente que la evolución de los videojuegos en el mundo multijugador en red ha estado muy relacionada con la innovación de estudios pioneros en este campo, y acompañada del incremento en capacidades de hardware, así como del uso de las redes tanto locales como globales.

Finalmente, hemos practicado con la modalidad local, con la intención de establecer un buen punto de inicio para ir tomando contacto con el desarrollo de un videojuego multijugador. Concretamente, hemos trabajado un ejemplo en Unity que nos ha permitido estudiar la gestión y modificación de las entradas provenientes de diferentes jugadores, y la implementación de la técnica de pantalla partida, muy popular antaño y que parece irónicamente estar viviendo un nuevo resurgir en estos tiempos de alta conectividad y múltiples posibilidades y facilidades para el juego en línea.

Bibliografía

Alexandre, Thor (2005). *Massively Multiplayer Game Development 2 (Game Development)*. Rockland, MA: Charles River Media, Inc.

Armitage, Grenville; Claypool, Mark; Branch, Philip (2006). *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. Hoboken, NJ: John Wiley & Sons, Ltd.

Barron, Todd (2001). *Multiplayer Game Programming*. Boston, MA: Thomson Course Technology.

Kushner, David (2003). *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. Nueva York: The Random House Publishing Group.

Id Software (1996). *Quake World* [Juego de ordenador en línea].

Young, Vaughan (2005). *Programming a Multiplayer FPS In DirectX*. Hingham, MA: Charles River Media.