Els jocs multijugador

PID_00270024

Rubén Mondéjar Andreu
Carles Pairot Gavaldà

Temps mínim de dedicació recomanat: 3 hores

Universitat Oberta de Catalunya

Rubén Mondéjar Andreu **Carles Pairot Gavaldà**

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Javier Cánovas Izquierdo (2020)

Primera edició: febrer 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Índex

Introducció

En aquest mòdul didàctic presentem les nocions bàsiques dins del món dels videojocs multijugador, tant des de la perspectiva conceptual com les eines necessàries per crear-ne un.

En primer lloc, aprofundirem en els videojocs que incorporen aquesta modalitat, ja sigui parcialment o totalment. Per a això, ens centrarem en el punt de vista històric, tot repassant breument el seu origen i la seva evolució en les diferents èpoques en què van ser llançats.

A continuació, veurem breument com va ser aquesta evolució en altres dispositius, com ara videoconsoles, mòbils i tauletes. Aquest repàs servirà per introduir alguns conceptes bàsics a través d'exemples de jocs que en el moment van ser populars i innovadors.

Finalment, i com a primer contacte des d'un vessant més pràctic, es proposa un projecte que consisteix a implementar un joc multijugador en mode local a pantalla partida. Per fer-ho, es comença a partir d'un joc preexistent que es modificarà amb el propòsit d'afegir-hi aquesta funcionalitat.

Objectius

En aquest mòdul didàctic es presenten els coneixements necessaris per assolir els següents objectius:

1. Conèixer què és un videojoc multijugador tant local com en línia.

2. Estudiar l'evolució i les innovacions dels videojocs multijugador.

3. Programar un joc en mode multijugador local amb el motor Unity.

1. L'evolució dels jocs multijugador

En aquest primer apartat estudiarem la interacció que es produeix entre diferents jugadors, ja sigui en un mateix dispositiu (per exemple, ordinadors o videoconsoles) o entre diferents dispositius a través d'una xarxa.

Amb aquesta interacció simultània, podem habilitar diferents jugadors perquè participin de manera col·laborativa o competitiva en un mateix videojoc.

Quan parlem de jocs multijugador, ens referim a aquells que disposen de qualsevol modalitat en què es té en compte la interacció de dos o més jugadors al mateix temps. Aquesta interacció pot ser física en un sol dispositiu, a través de diferents dispositius mitjançant cables o una connexió sense fils, o a través de serveis en línia o un altre tipus de xarxa amb persones que hi estiguin connectades. La interacció pot ser en temps real (interacció simultània) o per torns (només un jugador realitza les accions mentre la resta s'espera).

El fet d'afegir una opció multijugador és un factor important a tenir en compte en el disseny d'un videojoc. Des de la perspectiva del desenvolupament, té implicacions importants en el seu disseny i el codi que s'ha de generar. Però des de la perspectiva del màrqueting, també pot augmentar el seu temps de vida, és a dir, el temps que els jugadors hi segueixen interessats i actius. Per tant, pot augmentar el valor que l'usuari percep d'un joc.

Abans o després?

La inclusió de la modalitat multijugador s'ha de definir des del començament del projecte.

Tot seguit veurem les dues maneres d'integrar aquesta opció en un videojoc, tant la modalitat en què els jugadors es connecten a la mateixa màquina (mode local), com la modalitat en què cadascun ho fa des del seu dispositiu (mode remot).

També cal remarcar que hi ha videojocs que permeten una modalitat mixta entre multijugador local i multijugador remot, tot permetent que els jugadors d'un mateix dispositiu competeixin contra d'altres a distància, com per exemple un dos contra dos en alguns jocs *FIFA* d'Electronic Arts, o incorporar dos jugadors en el mateix equip en una sessió de *deathmatch* en algunes versions de la saga *Halo*.

1.1. El multijugador local

És evident que el mecanisme més simple per crear un joc multijugador és permetre que diferents jugadors puguin participar simultàniament a través del mateix dispositiu. Aquesta modalitat de joc ha estat molt important des del començament en diferents plataformes i sovint ha ofert un valor afegit als jocs d'un sol jugador.

Així, alguns dels primers jocs de la història ja incorporaven l'opció de joc multijugador en mode local. En realitat era una simplificació per evitar que fos l'ordinador que dirigís les accions del nostre oponent. Alguns exemples de jocs primerencs són de *Tennis for Two* (1958), *Spacewar!* (1962) o l'arxiconegut *Pong* (1972).

1.1.1. Ordinadors domèstics

Es coneix com a **computadora domèstica** o **ordinador domèstic** la segona generació de computadores, que van sortir a la venda amb el naixement de l'Altair 8800 en un període de temps que s'estén fins a principis dels noranta. Això inclou totes les computadores de 8 bits i la primera remesa d'equips amb CPU de 16 bits.

Figura 1. *Gauntlet* (1985)

Aquests ordinadors són considerats els pares de les videoconsoles que arribarien *a posteriori* a ocupar el seu lloc i, per tant, els primers a arribar a les llars. El seu enfocament familiar convidava els jugadors a comprar més dispositius d'entrada, com ara *joysticks* o comandaments per poder jugar a jocs multijugador.

1.1.2. Màquines *arcade*

Durant les èpoques d'or dels anys vuitanta i principis dels noranta, les màquines *arcade* van fomentar el multijugador local amb títols de dos jugadors (*Pong*) o fins i tot de quatre (*Gauntlet*) a les sales recreatives de l'època.

Figura 2. *Street Fighter II, The World Warrior*

Figura 3. Màquina recreativa de *Beast Bursters* per a tres jugadors simultanis

Aquestes màquines oferien diferents dispositius d'entrada per a cada jugador i, òbviament, els gèneres que més s'hi prestaven eren els jocs de lluita (saga *Street Fighter*), els *beat'em-up* (*Teenage Mutant Ninja Turtles*) i altres de molt menys populars en altres plataformes, com els *shooter* sobre raïls (per exemple, *Virtua Cop*).

Figura 4. *Teenage Mutant Ninja Turtles*

En els últims temps, aquestes màquines es connectaven mitjançant xarxes per poder fer partides multijugador en xarxa, com *Sega Rally* o *Mario Kart,* tot oferint la possibilitat de competir amb jugadors de la mateixa sala recreativa en diferents màquines.

1.1.3. Videoconsoles

Una **videoconsola** de **consola de videojocs** és un sistema electrònic d'entreteniment per a la llar que executa videojocs continguts en diferents tipus d'emmagatzematge.

Figura 5. *Super Mario Kart*

Evidentment, les videoconsoles han estat durant molt de temps la referència a les llars dels jugadors, amb la instauració des dels seus inicis del multijugador local com una manera de jugar amb la família o els amics. Des de l'Atari fins a la Wii U, podem veure clarament com s'ha fomentat aquest factor, tot passant per grans referents com la sèrie *Mario Kart* o el *shooter Goldeneye*, un dels grans avantatges del qual va ser que permetia jugar fins a quatre jugadors alhora.

Figura 6. *Goldeneye* per a Nintendo 64

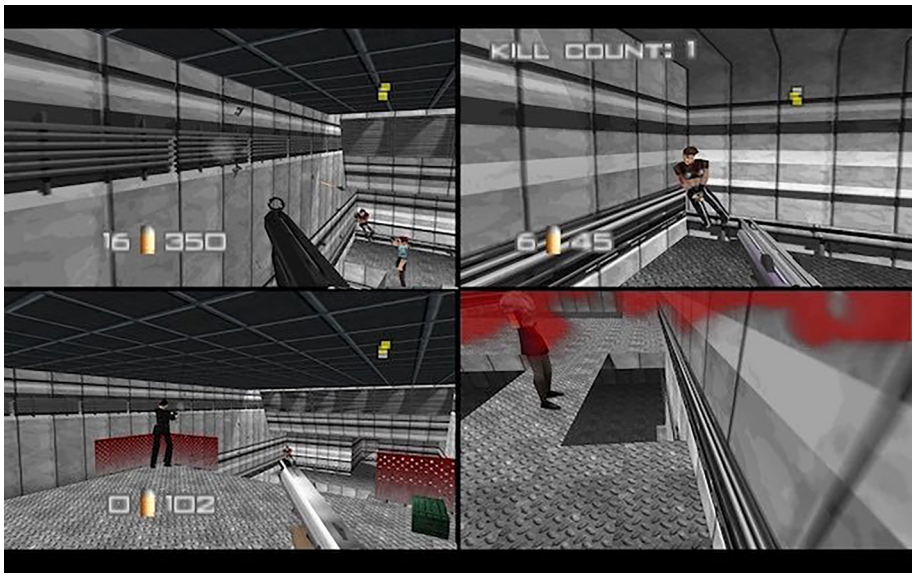


Figura 7. Super Nintendo MultiTap

Un detall a tenir en compte en aquesta evolució de consoles de sobretaula: fins al llançament de la Nintendo 64, les consoles van donar suport a quatre jugadors alhora, requerint *hubs* per a comandaments coneguts com a *multitap,* que permetien jugar a dos jugadors és simultàniament en títols com *Superman Bomberman,* per als quals sí que hi havia màquines que el suportaven, en aquella època.

Figura 8. *Super Bomberman*

1.2. El multijugador en xarxa

Tecnologies de xarxa

La inclusió de tecnologies de xarxa en un joc s'ha convertit en un element clau per incrementar-ne les vendes i la qualitat.

L'origen dels jocs multijugador en xarxa moderns són els sistemes *mainframe* universitaris de la dècada dels setanta. No obstant això, fins a l'arribada de l'accés a Internet a mitjan dècada dels noranta no van començar a destacar de veritat. El potencial que ofereix Internet com a eina de comunicació global ha revolucionat l'ús de les tecnologies, principalment perquè proporciona accés a un 81 % de la població dels països més desenvolupats. Els videojocs no són una excepció. Els jocs en xarxa han tingut una evolució molt clara i paral·lela a les innovacions en telecomunicacions, i és molt important analitzar cada etapa relacionada amb les tecnologies emergents en el seu moment.

Per tant, depenent de l'època, cada joc ha ofert diferents opcions multijugador, des de les petites xarxes locals fins a les xarxes d'àmbit global com Internet, que han permès a jugadors de diferents parts del món jugar al mateix videojoc simultàniament.

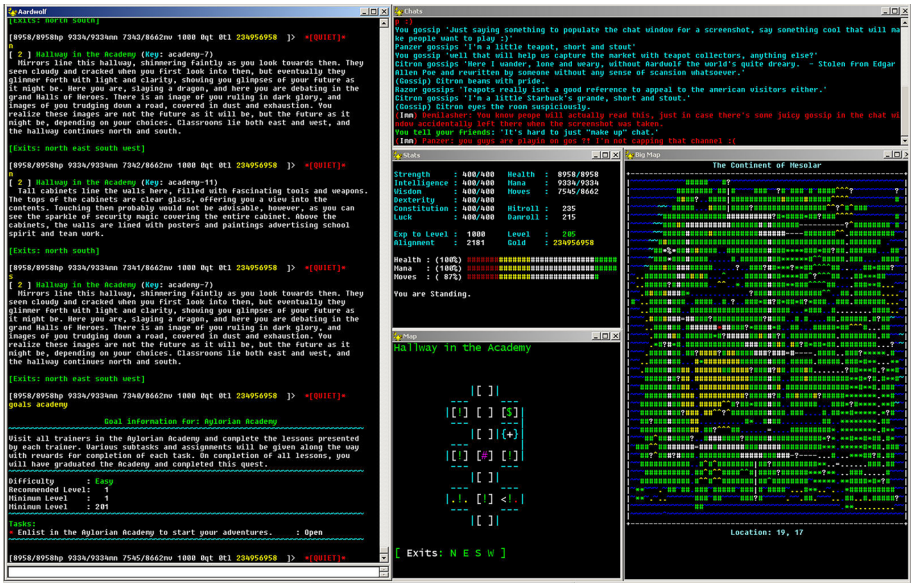
1.2.1. Masmorres multiusuari

Les masmorres multiusuari o MUD (de l'anglès *Multi-User Dungeon*) són jocs multijugador en els quals diferents jugadors estan connectats simultàniament al mateix món virtual a través d'un terminal.

El funcionament d'aquests jocs exigeix la creació d'un avatar per part del jugador i, tot explorant un món, ha de desenvolupar les estadístiques associades.

Aquest tipus de joc es va fer popular als *mainframes* de les principals universitats. El terme MUD va néixer amb el joc homònim del 1978, creat per Rob Trushaw a la Universitat d'Essex. D'alguna manera, els MUD poden ser considerats una primera versió d'ordinador del joc de rol *Dungeons and Dragons* (Dracs i Masmorres), encara que no tots els MUD siguin necessàriament jocs de rol.

Figura 9. MUD *Aardwolf* (1996)



Quan els ordinadors personals es van fer més potents, els fabricants de maquinari van començar a oferir mòdems que permetien que dos ordinadors es comuniquessin entre ells a través de línies telefòniques estàndard. Encara que les taxes de transmissió fossin extraordinàriament lentes comparades amb els estàndards moderns, això va permetre jugar als MUD fora de l'àmbit universitari.

1.2.2. Jocs en LAN

A principis dels anys noranta va aparèixer la possibilitat d'utilitzar les connexions en xarxa dels ordinadors personals per facilitar un nou tipus de videojocs. En aquesta època, l'accés a la xarxa es feia mitjançant mòdems o a través d'una LAN (xarxa d'àrea local, o *Local Area Network*) mitjançant el protocol IPX de Novell, fet que limitava la quantitat d'informació que es podia transferir entre ordinadors.

El protocol IPX (*Internet Packet eXchange*) va ser un dels primers que es van utilitzar per interconnectar ordinadors en una xarxa d'àrea local. Al final va acabar desapareixent perquè no era un protocol òptim per comunicar un gran nombre d'ordinadors, a diferència del TCP/IP.

El primer joc que va oferir suport multijugador LAN va ser *Doom* (1993), tot establint les bases dels jocs en xarxa moderns. La versió inicial d'aquest *shooter* en primera persona d'Id Software suportava fins a quatre jugadors en una sola sessió de joc, amb l'opció de jugar cooperativament o en modalitat competitiva, batejada com a *deathmatch*.

Doom

Cal destacar que *Doom* va ser un joc d'acció amb un ritme ràpid, que exigia l'aplicació de molts conceptes clau de comunicació en xarxa.

Per descomptat, aquestes tècniques han evolucionat molt des de 1993, però la influència de *Doom* encara és molt notable avui, ja que moltes de les seves mecàniques han influït en jocs posteriors. Cal assenyalar que amb Doom es va començar a parlar de «motor del joc» (*game engine*) com a element que va servir per separar la tecnologia subjacent del joc en si. Més endavant, moltes empreses com Epic Games van llicenciar la seva tecnologia reutilitzable perquè els dissenyadors de videojocs poguessin centrar-se en els jocs i no en la tecnologia. L'exemple més conegut és l'Unreal Engine.

Lectura recomanada

Per a més detalls sobre la història i la creació de *Doom,* recomanem el llibre *Masters of Doom* (2003), llistat a la bibliografia d'aquest mòdul didàctic.

Molts jocs multiintèrpret amb suport per a LAN també admetien altres modalitats, com per exemple la connexió per mòdem. Durant molts anys, la gran majoria de jocs en xarxa va seguir donant suport al joc en LAN. Això va comportar l'aparició d'esdeveniments com les LAN *parties,* en les quals els jugadors es reunien físicament en un lloc concret per connectar els seus ordinadors i així jugar dins de la mateixa xarxa d'àrea local.

Figura 10. Mode multijugador en LAN de *Doom* (1993)

La primera aproximació de les videoconsoles al món dels videojocs multijugador en xarxa es va produir amb la introducció de la consola portàtil Nintendo Game Boy. La consola incorporava un port sèrie que permetia connectar fins a 4 mecanismes amb el cable Game Link, i alguns dels jocs més populars (com per exemple *Tetris*) oferien la possibilitat de competir amb altres usuaris.

Tanmateix, el desplegament de xarxes d'àrea local per a la interconnexió de consoles no és el cas més habitual, ateses les seves característiques de dispositiu per a la llar.

Figura 11. Cable Game Link

Tot i que encara hi ha jocs multijugador en xarxa amb suport per a LAN, la tendència els darrers anys, tant dels sistemes com dels desenvolupadors, sembla centrar-se exclusivament en la modalitat de multijugador en línia.

1.2.3. Jocs en línia

En un joc en línia (*online*), els jugadors es connecten entre si a través d'una gran xarxa d'ordinadors separats geogràficament. Avui, els jocs en línia són sinònim de jocs a Internet, però el terme *en línia* és una mica més ampli i pot incloure algunes de les xarxes anteriors que, inicialment, no es connectaven a Internet.

Wide Area Network (WAN)

Una xarxa d'àrea àmplia o WAN és una xarxa d'ordinadors que uneix diverses xarxes locals, encara que els seus membres no siguin tots en una mateixa ubicació física. Moltes WAN es construeixen per part d'organitzacions o empreses per a fer-ne un ús privat, i altres són instal·lades per proveïdors d'Internet per facilitar connexió als seus clients.

A mesura que Internet es va començar a estendre a finals de 1990, els jocs en línia van seguir el mateix camí ascendent. Alguns dels jocs més populars aquells anys van ser *Quake*, d'Id Software (1996) i *Unreal*, d'Epic Games (1998). Ambdós casos, jocs de trets en primera persona (FPS, *First Person Shooters*), com *Doom*.

Encara que sembli que un joc en línia pot ser implementat de la mateixa manera que un joc en LAN, en realitat, cal assenyalar que la quantitat de temps que necessiten les dades per viatjar a través de la xarxa normalment és més gran. Per tant, la incorporació de la connectivitat via Internet va suposar un nou pas en l'evolució dels jocs multijugador des del punt de vista de la seva arquitectura.

Figura 12. *Quake* (1996)

Latència

Una consideració molt important és la **latència** o quantitat de temps que les dades necessiten per viatjar per la xarxa.

De fet, com a anècdota interessant, la versió inicial de *Quake* no estava realment dissenyada per treballar mitjançant connexió a Internet. Fins al llançament d'una actualització, concretament el pedaç *QuakeWorld*, la modalitat multijugador no era fiable a través d'Internet atesa la latència de les comunicacions. Els mètodes per compensar la latència es tractaran al llarg de l'assignatura.

En el camp de les videoconsoles, a poc a poc van anar apareixent alguns intents d'afegir components de maquinari de xarxa a mesura que proliferaven els serveis de banda ampla. La Sega Dreamcast (1999) tenia un adaptador de xarxa i va ser una de les primeres a provar de capitalitzar el creixent mercat dels jocs en línia.

En aquells moments, però, els serveis d'Internet encara eren massa limitats a escala mundial i no es comptava amb serveis de banda ampla com els actuals. Dreamcast utilitzava connexions via telefònica per proporcionar el servei de joc en línia, és a dir, el sistema incloïa un mòdem, i per això el seu èxit va ser molt limitat.

La innovació del joc en línia que proposava Sega no va ser suficient per conquerir el mercat de les videoconsoles. Així doncs, Sega va ser relegada a un segon pla i semblava que el seu lloc l'ocuparia Microsoft, amb el qual s'havia associat amb la inclusió del seu sistema operatiu en la seva última consola.

Figura 13. Xbox Live des del *dashboard* de la primera Xbox

Aquest va ser un punt d'inflexió important, perquè amb l'arribada dels serveis en línia de mà de Microsoft amb l'Xbox Live (2002) en la seva primera consola d'escriptori, la Xbox original, havia començat l'era dels serveis multijugador.

A mesura que la proliferació dels videojocs s'ha anat estenent per l'entorn mòbil, els jocs multijugador han crescut de la mateixa manera en aquest entorn on, a causa de les característiques del dispositiu, el multijugador local no té gaire sentit (si no és que parlem de jocs per a tauletes). Molts dels jocs multijugador en aquestes plataformes són asíncrons, normalment estan basats en torns i no requereixen la transmissió de dades en temps real.

En aquest model, als jugadors se'ls notifica quan arriba el seu torn i disposen d'un temps considerable per fer el seu moviment. Un exemple d'un joc mòbil conegut que utilitza la modalitat de multijugador asíncrona és paraules amb *Words with Friends* (2009). Des d'un punt de vista tècnic, un joc en xarxa asíncron és més senzill d'implementar que no pas un en temps real.

Jocs asíncrons

Òbviament, el model asíncron ha existit des del començament dels jocs multijugador de xarxa, com per exemple amb *Space Crusade*.

Originàriament, el model asíncron es va utilitzar en jocs per a mòbils –per la necessitat de fiabilitat en xarxes mòbils, comparada amb les connexions per cable. No obstant això, amb la proliferació de dispositius i les millores de les xarxes Wi-Fi, ja és habitual veure jocs en temps real en aquests dispositius. Un exemple d'un joc per a mòbils que aprofita la comunicació en xarxa en temps real és *Hearthstone: Heroes of Warcraft* (2014).

Suport mòbil

Cal destacar que els darrers anys també s'han creat serveis en línia (iOS Game Center o Google Play Games) que donen suport específic als jocs multijugador per a dispositius mòbils.

1.2.4. Jocs multijugador massius

El següent pas en l'evolució dels jocs en xarxa va arribar amb l'aparició dels primers jocs en línia persistents cap al 1995, una evolució clara dels MUD. Fins i tot avui, la majoria dels jocs multijugador en línia es limiten a un petit nombre de jugadors per partida, amb un número de jugadors admesos d'entre dos i seixanta-quatre. No obstant això, no centenars, sinó milers de jugadors poden participar en una sola sessió de joc en línia persistent.

Originàriament, aquests jocs combinaven una part dels dos punts anteriors: tenien un servidor central al qual els usuaris es connectaven per jugar i, a més, disposaven d'un client gràfic que permetia interactuar intuïtivament amb l'entorn i amb altres jugadors. Aquesta mena de jocs van ser anomenats multijugador massius (*Massively Multiplayer Games*).

Aquest tipus de jocs no es poden jugar de manera local, perquè bona part de la lògica del joc es troba en els servidors i requereixen d'una connexió permanent amb el servidor cada vegada que es vol jugar una partida. A més, aquests servidors es distribueixen en zones, per poder donar servei a cada ubicació per separat, encara que també solen disposar de servidors centralitzats.

Això té repercussió, entre altres coses, en el cost de manteniment d'aquesta infraestructura de servidor. Per això, els jocs persistents van introduir un nou model de negoci per als videojocs, basat en subscripcions mensuals.

Figura 14. *World of Warcraft*

La majoria d'aquests jocs pertany al gènere dels MMORPG (*Massively Multiuser Online Role Playing Game*) i, en molts sentits, els MMORPG es poden considerar l'evolució gràfica de les masmorres multiusuari. Com a exemples paradigmàtics, podem destacar *Ultima Online* (1997), que va ser el primer a establir-se i el que més temps porta en el mercat, i el *World of Warcraft* (2004), que va batre tots els rècords en nombre d'usuaris subscrits en superar els 12 milions de jugadors de tot el món el 2010.

Arquitectura d'un MMO

L'arquitectura d'un MMO és un repte tècnicament complex, que va més enllà del seu propi desenvolupament, i que requereix coneixements de *backend* i arquitectures de xarxes, així com de gestió de servidors.

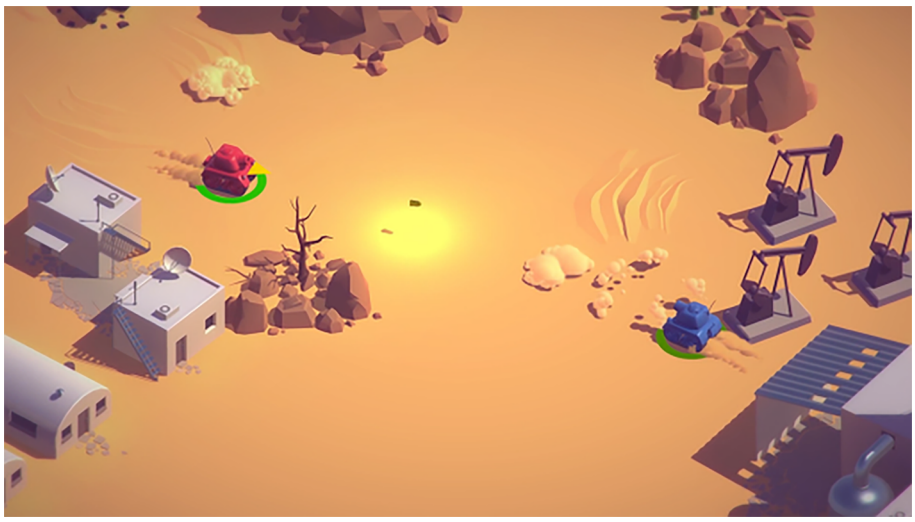
2. Projecte: *Tanks!* local

En bona part, els jocs multijugador locals poden ser programats de la mateixa manera que els jocs d'un sol jugador. Les úniques diferències solen ser múltiples punts de vista i/o el suport a múltiples dispositius d'entrada. Atès que la programació de jocs multijugador locals és molt similar als jocs d'un sol jugador, el nostre primer objectiu en aquesta assignatura serà desenvolupar un joc d'aquest tipus.

Vegeu també

Al final de l'apartat hi ha les solucions a tots els reptes.

Figura 15. Tutorial de *Tanks!*



En aquest apartat revisarem dos punts clau per poder oferir aquesta modalitat en els nostres jocs: la gestió de les múltiples entrades i estàncies de jugador, així com l'aplicació de la tècnica de *split screen* (pantalla partida), tot proporcionant a cada jugador la seva pròpia càmera i el seu espai de pantalla.

Per a això, treballarem a partir d'un exemple oficial d'Unity3D (*Tanks!*), al qual hem incorporat un mode de multijugador local bàsic.

Codi de l'exemple

Podeu trobar aquest exemple a l'Asset Store d'Unity.

2.1. Gestió de múltiples dispositius d'entrada

El requisit indispensable per a un joc multijugador local és permetre als jugadors que cadascun d'ells utilitzi un dispositiu d'entrada diferent. Aquests dispositius no necessàriament han d'estar separats físicament, com podria ser l'exemple de jocs que destinen diferents tecles d'un teclat per a cada jugador o en dispositius mòbils, en què cada zona de la pantalla està destinada a rebre entrades d'un jugador.

Diferents entrades

Encara que, normalment i per ergonomia, es prefereix l'ús de diferents tipus d'entrades, aquestes no han de ser del mateix tipus necessàriament, per exemple, un jugador utilitza el teclat i un altre un comandament.

Per tal de desenvolupar un joc multijugador local, és important que, pel que fa a la programació, siguem capaços de desenvolupar el codi necessari perquè l'entrada de l'usuari sigui el més genèrica possible i permeti ser configurada per dotar-la de la màxima versatilitat possible.

Aquestes opcions no són trivials, si tenim en compte la quantitat de dispositius diferents d'entrada i de fabricants, així com el seu comportament en els diferents sistemes operatius en els quals, per exemple, un comandament de Xbox 360 no té el mateix comportament en la consola, en un ordinador amb Windows o en un ordinador amb MacOS a causa dels *drivers* i l'assignació de botons en cadascun, així com l'accés a les seves opcions avançades, o no, en cada entorn.

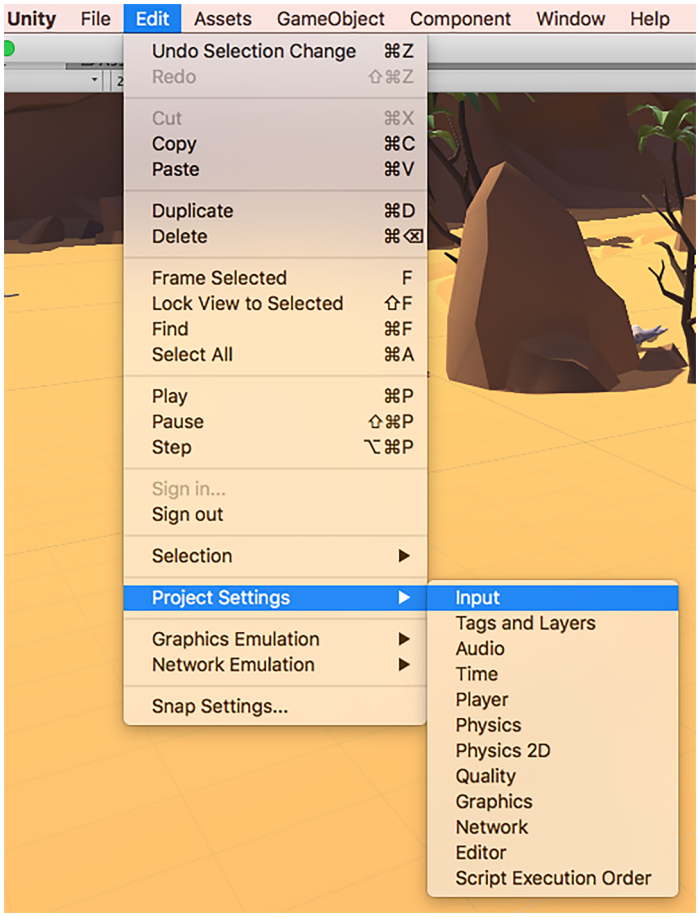
Figura 16. *Mapping* de botons del *pad* de *Xbox* 360 en Windows i MacOS

2.1.1. Exemple

Dins del codi del projecte, a la carpeta *_Completed-Assets/Scripts/Tank*, podeu trobar els *scripts TankMovement* i *TankShooting* que permeten utilitzar les diferents entrades genèriques dels botons que provenen de la configuració del joc, tot afegint en temps d'execució el nombre de jugadors que s'instancien.

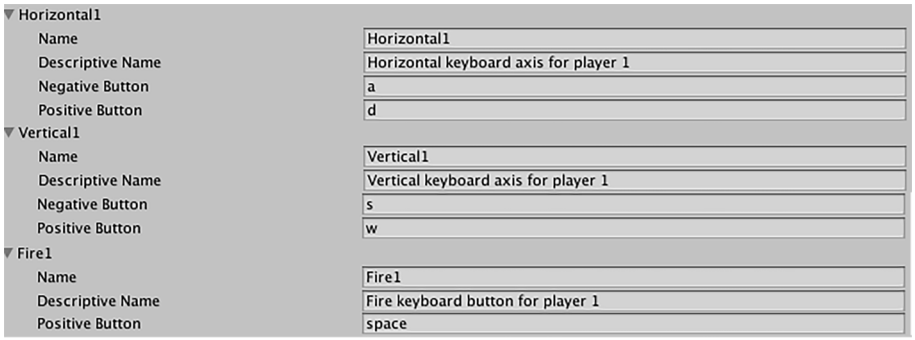
```
// The axes names are based on the player number.
m_MovementAxisName = "Vertical" + m_PlayerNumber;
m_TurnAxisName = "Horizontal" + m_PlayerNumber;

// The fire axis is based on the player number.
m_FireButton = "Fire" + m_PlayerNumber;
```

Figura 17. Configuració de dispositius d'entrada al menú *Edit* > *Project Settings* > *Input* d'Unity



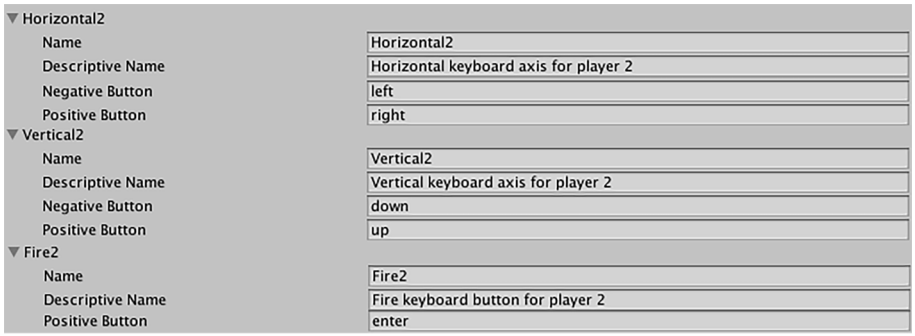
Com podem veure en aquest apartat de configuració, en aquest projecte ja tenim definits els controls d'ambdós jugadors.

Figura 18. Propietats dels dispositius d'entrada del jugador 1



Per exemple, amb la propietat *Horizontal1* es defineixen els dos botons de moviment (incrementar o disminuir) horitzontal del jugador 1. D'altra banda, la propietat *Fire2* defineix el botó de disparament del jugador 2.

Figura 19. Propietats dels dispositius d'entrada del jugador 2



2.1.2. Integració en el projecte

No indicar els botons concrets en el nostre codi i cenyir-nos a l'API d'Unity ens permet ser més versàtils i derivar els problemes dels diferents dispositius a fases més avançades de desenvolupament, així com valorar l'ús d'altres dispositius o *assets* molt elaborats que ens proporciona aquesta funcionalitat i problemàtiques ja resoltes.

Repte 1

Habiliteu una nova acció secundària (*AltFire*) perquè cada jugador pugui disparar alternativament amb un altre botó a escollir per vosaltres.

En executar el joc tal com es presenta a l'escena *_Complet-Game*, podeu comprovar el funcionament d'aquest punt inicial.

Repte 2

Modifiqueu l'entrada d'acció secundària (*AltFire*) perquè el tanc dispari projectils de color vermell i que siguin, a més, un 50% més ràpids.

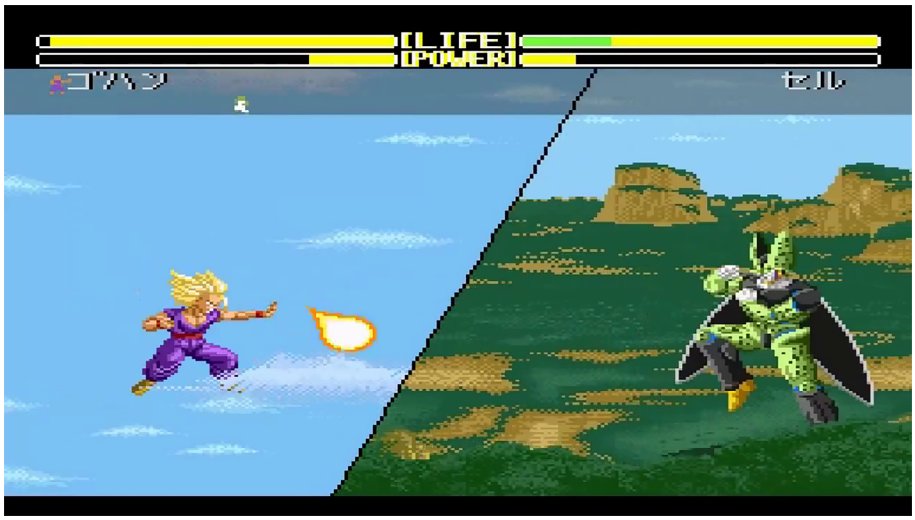
CompleteShell

Els projectils del joc estan gestionats pel *prefab CompleteShell*. El color pot ser controlat a través del seu *shader*.

2.2. Pantalla partida (*split screen*)

Aquesta opció també pot ser dinàmica en jocs, especialment en els jocs amb perspectiva bidimensional, tot utilitzant-la com a recurs en cas que els jugadors s'allunyin prou com perquè el zoom no arribi a mostrar-los alhora a la pantalla d'una manera satisfactòria.

Figura 20. *DBZ Butoden 2* (1993)



Exemples d'aquesta variant podrien ser l'aplicada a la saga *Dragon Ball Butoden* de Super Nintendo o en jocs de les múltiples sagues *Lego*, en què es potencia el multijugador local.

Figura 21. *Lego Marvel Super Heroes* (2013)

En la nostra implementació buscarem el mateix efecte per potenciar l'ús de la pantalla completa quan els jugadors es trobin a poca distància, però en el moment en què es detecti que se supera aquesta distància, activarem la segona pantalla, tot dividint- la horitzontalment.

Una millora que veiem en aquest tipus de jocs és que la divisió de la pantalla és obliqua en funció de la posició entre els jugadors. Deixarem aquesta millora per als reptes d'aquest mòdul.

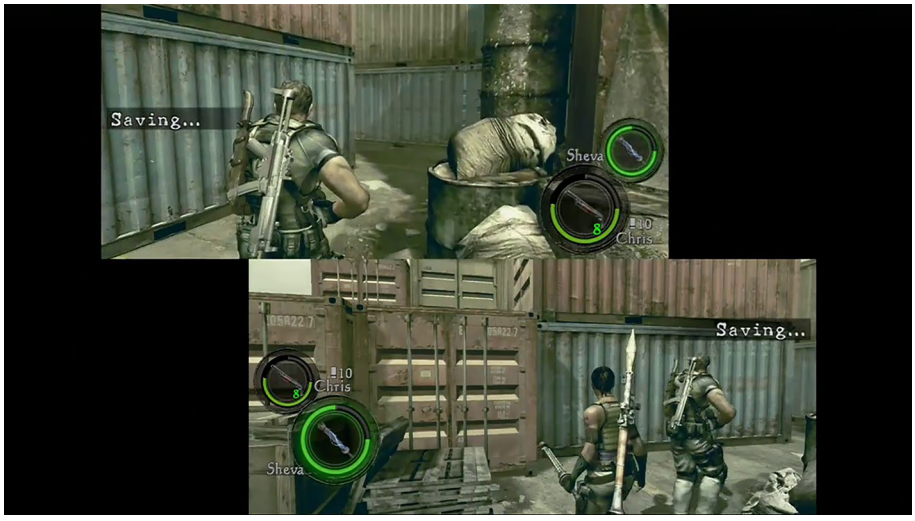
Figura 22. Ulleres 3D complementàries

Contingut complementari

Aprofitant la tecnologia de les ulleres 3D complementàries, cada jugador pot veure el joc a pantalla completa.

Com a curiositat, és interessant saber que no sempre s'ha permès als jugadors obtenir el màxim possible de pantalla mitjançant aquesta tècnica, sinó que hi va haver alguns casos en què es va mantenir la proporció de la pantalla individual amb l'*aspect ratio* original (per exemple, 16:9) com va ser el cas del motor marc *MT Framework* (v1.4), de Capcom, amb jocs com *Resident Evil 5* o *Lost Planet 2*.

Figura 23. Resultat amb la pantalla partida del mode cooperatiu a *Resident Evil 5*



2.2.1. Gestió de càmeres per a pantalla partida

Tot i que *a priori* ens pugui semblar complicat, realment és fàcil crear un efecte de pantalla partida gràcies a la propietat *viewport rectangle* de cada càmera.

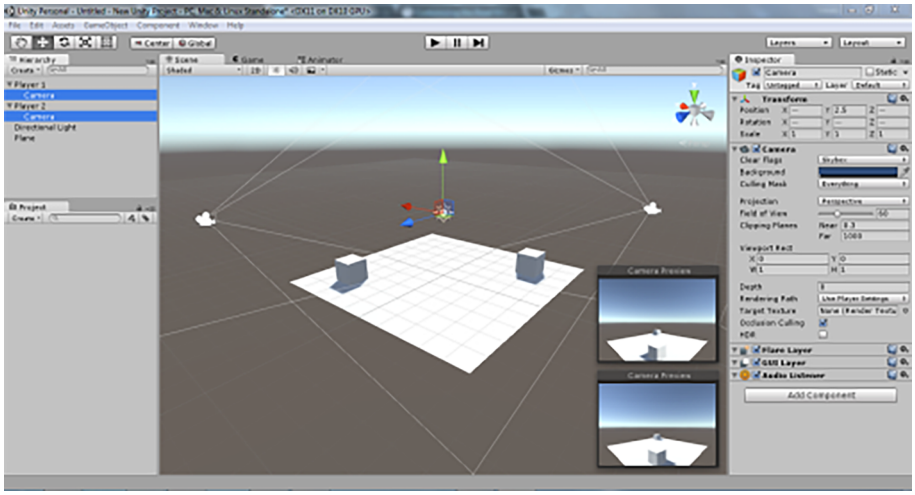
Concretament, si tenim dues càmeres, hem de revisar els valors d'amplitud (W) i d'alçada (H) de cada *viewport rectangle*. Això ens permet, per exemple, que la càmera del primer jugador es mostri des del la meitat de la pantalla fins a la part superior, mentre que la càmera del segon jugador comença a la part inferior i s'atura a mig camí per sobre de la pantalla.

Viewport rectangle

S'utilitza per especificar a quina part de la pantalla es dibuixarà la vista de la càmera on està definida.

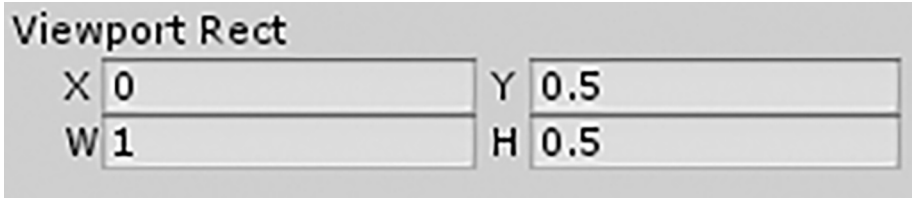
Posarem en pràctica aquesta idea començant amb un exemple molt inicial en Unity, on només hem de crear dos *GameObject*, com per exemple dos cubs, que representen els jugadors, tot associant una càmera a cadascun d'ells.

Figura 24. Creació de *GameoOject* en Unity



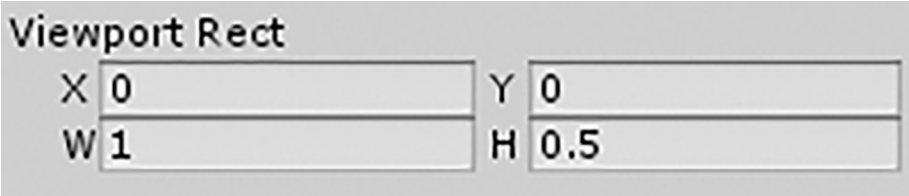
A continuació, donarem els valors (X = 0, Y = 0.5, W = 1, H = 0.5) al *Viewport Rect* de la càmera del jugador 1.

Figura 25. *Viewport Rect* de la càmera del jugador 1



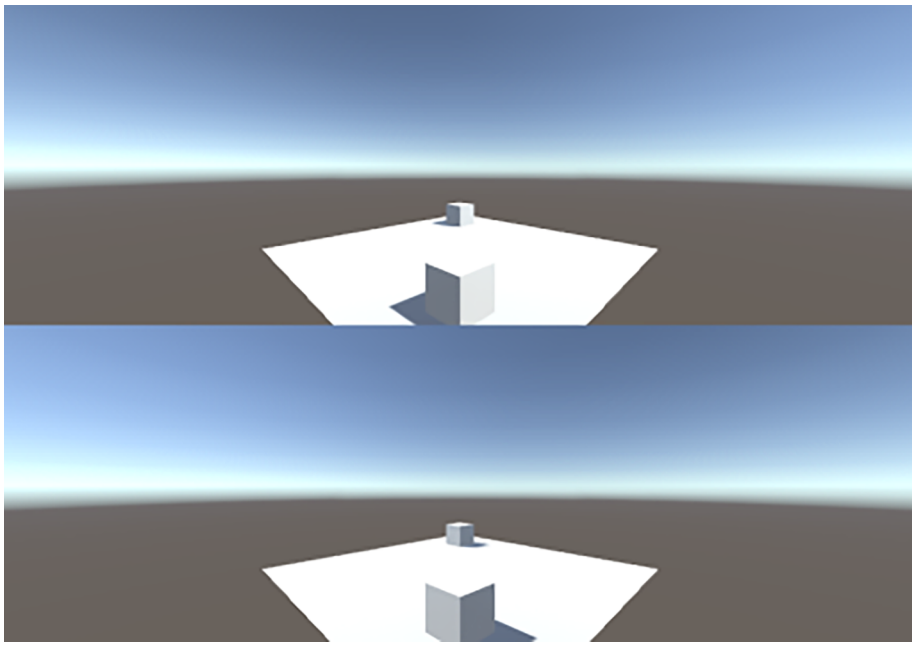
El següent pas és configurar el *Viewport Rect* perquè les dimensions de la càmera del jugador siguin (X = 0, Y = 0, W = 1, H = 0.5). Finalment, hem de donar els valors (X = 0 , Y = 0, W = 1, H = 0.5) al *Viewport Rect* de la càmera del jugador 2.

Figura 26. *Viewport Rect* de la càmera del jugador 2



I així és com, gràcies a la redimensió del que mostra cada càmera de cada jugador, aconseguim obtenir el resultat esperat en qualsevol dels nostres projectes.

Figura 27. Resultat final en aplicar la tècnica de la pantalla partida



Aquesta mateixa tècnica es pot aplicar directament al nostre projecte de combat de tancs, de manera que els dos jugadors tenen la seva àrea de pantalla individual.

Repte 3

Feu que el *Viewport* de cada jugador sigui proporcional a la pantalla original, amb la pantalla del primer jugador alineada a l'esquerra i la del segon jugador a la dreta, com amb el cas del motor MT Framework (per exemple, *Resident Evil 6*).

2.2.2. Pantalla partida desplegada dinàmicament

Un cop entenem el concepte bàsic i sabem on hauríem de modificar un joc per aplicar-hi aquesta tècnica, passem a fer la modificació de l'exemple que estem estudiant en aquest apartat.

Abans de començar, hem de considerar que és més adient fer la modificació del projecte a través del codi en temps d'execució que en l'editor, com hem vist anteriorment. D'aquesta manera, tenim una solució més flexible, podem activar-la quan vulguem —o també, si programem l'*script* per dividir la pantalla per a dos jugadors, podem fer el mateix per a quatre jugadors.

En el tutorial original sobre en el qual ens basem, ja existeix l'*script GameManager* en el qual, mitjançant la funció *SpawnAllTanks()*, es realitza l'*spawn* de tots els tancs. El primer pas és modificar aquesta funció perquè tingui accés a la càmera principal i única fins ara:

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

La idea és utilitzar-la per realitzar rèpliques per a cadascun dels jugadors mitjançant una funció anomenada *AddCamera(..)*, que implementarem més endavant i que rep com a paràmetre el número de jugador i la càmera original:

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

I, finalment, desconnectem la càmera inicial:

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

El codi final quedaria de la manera següent:

```
private void SpawnAllTanks()  {
     Camera mainCam =
         GameObject.Find ("Main Camera").GetComponent<Camera>();

     for (int i = 0; i < m_Tanks.Length; i++) {
          m_Tanks[i].m_Instance = Instantiate(m_TankPrefab,
             m_Tanks[i].m_SpawnPoint.position,
             m_Tanks[i].m_SpawnPoint.rotation) as GameObject;

           m_Tanks[i].m_PlayerNumber = i + 1;
          m_Tanks [i].Setup ();
          AddCamera(i, mainCam);
     }
     mainCam.gameObject.SetActive(false);
 }
```

Repte 4

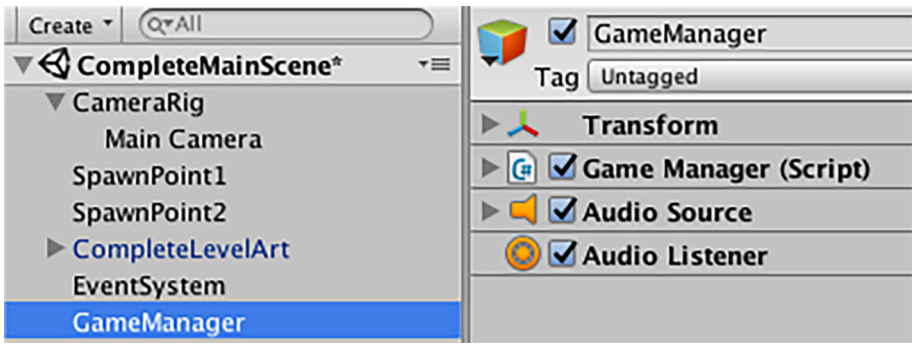
Activeu la tècnica de pantalla partida quan els tancs siguin a un mínim de separació i desactiveu-la quan es redueixi la distància per sota del mateix límit.

Sobre la funció *AddCamera(..)*, ens permetrà replicar la càmera principal i a més modificar el *Viewport* per donar a la vista del jugador la proporció de pantalla que li correspongui. En aquest cas, simplement apliquem la divisió horitzontal en aplicar de manera fixa l'*split screen* per a dos jugadors:

```
private void AddCamera(int i, Camera mainCam) {
     GameObject childCam = new GameObject( "Camera"+(i+1) );
     Camera newCam = childCam.AddComponent<Camera>();
     newCam.CopyFrom(mainCam);

     childCam.transform.parent = m_Tanks[i].m_Instance.transform;
     if (i==0) newCam.rect = new Rect (0.0f, 0.5f, 1.0f, 0.5f);
     else newCam.rect = new Rect (0.0f, 0.0f, 1.0f, 0.5f);
 }
```

Un últim detall a tenir en compte en realitzar aquest procés és que hem de canviar el component d'*AudioListener* (que ha de ser únic i existir) d'ubicació (actualment a la càmera principal) perquè no es dupliqui en clonar-la. Una opció senzilla és moure'l al *GameManager*:

Figura 28. GameManager



Quan es torna a executar es pot veure que el resultat és el que s'espera, amb la pantalla superior destinada a la càmera de la jugador 1 i la pantalla inferior a la jugador 2:

Figura 29. Resultat a la pantalla superior i la pantalla inferior

Repte 5

Feu que cada càmera enfoqui el tanc de cada jugador i que ho segueixi fent quan aquests es moguin.

Fer la modificació via *script* i no a través de l'editor, com amb la resta de l'exemple, permet que la nostra ampliació d'aquest projecte es mantingui prou genèrica com per ser extensible i fàcil de mantenir.

Nombre de jugadors

En cas de voler habilitar el joc per a un nombre diferent de jugadors haurem de revisar novament la funció *AddCamera(..)*, així com la lògica del joc involucrada per poder donar suport a aquest nou requeriment.

Repte 6

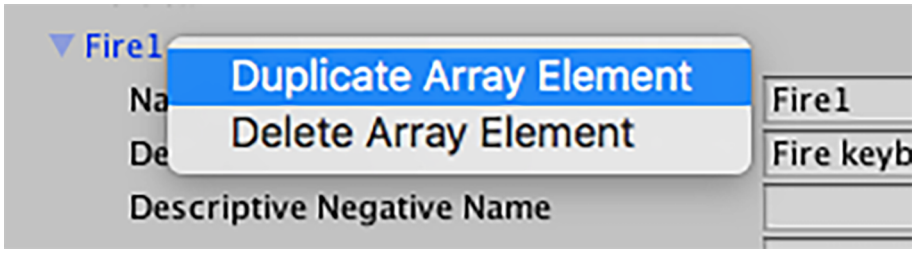
Genereu el projecte correctament per executar-lo com a aplicació al vostre sistema operatiu i modifiqueu els controls d'entrada del jugador.

2.2.3. Solucions als reptes proposats

Repte 1

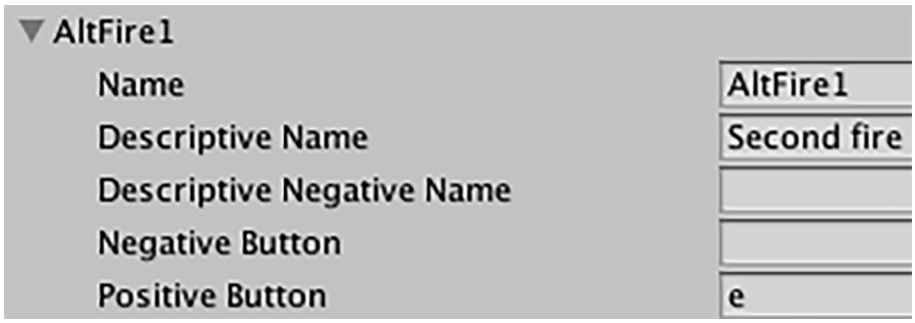
Com hem vist, des de l'editor d'entrades podem afegir una nova entrada per detectar el nou tipus de disparament, que podem anomenar *AltFire1* i *AltFire2* (respectivament).

Figura 30. Editor d'entrades



D'aquesta manera tindrem aquest *input* disponible i preconfigurat, amb la qual cosa la nostra feina aquí consisteix a fer que en detectar aquesta acció s'executi el comportament esperat.

Figura 31. *AltFire1*



L'últim pas consisteix a modificar l'*script* de comportament del disparament dels tancs (*TankShooting*) perquè també tingui en compte aquesta nova acció:

```
    private string m_AltFireButton;

    private void Start () {
           m_AltFireButton = "AltFire" + m_PlayerNumber;
    }
```

I també modificar els accessos als botons de foc i centralitzar-los en un mètode que dedueixi els diferents estats:

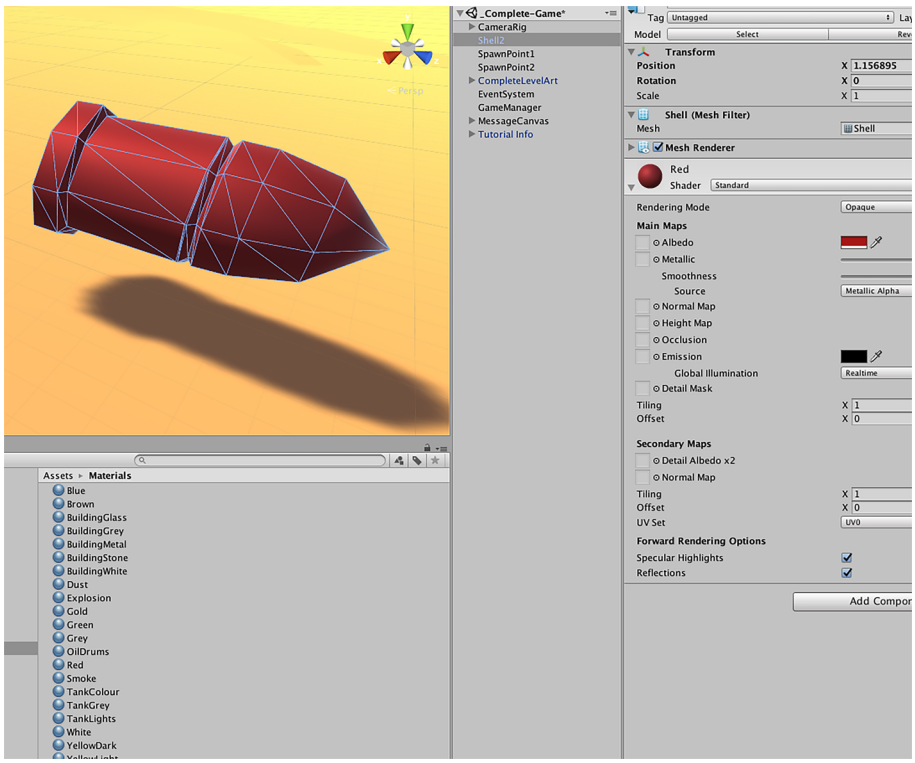
```
    private bool FireButton(int mode) {
           bool state = false;
           switch (mode) {
                case 0:
                    state = Input.GetButtonDown (m_FireButton)
                           || Input.GetButtonDown (m_AltFireButton);
                    break;
                    case 1:
                    state = Input.GetButton (m_FireButton)
                           || Input.GetButton (m_AltFireButton);
                    break;
                    case 2:
                    state =    Input.GetButtonUp (m_FireButton)
```

```
                            || Input.GetButtonUp (m_AltFireButton);
                    break;
                }
                return state;
    }
```

Repte 2

En primer lloc, necessitarem un nou model de projectil vermell, que podem crear fàcilment a partir de l'actual tot duplicant el *prefab* (*CompleteShell*) i canviant el *shader* actual (Gold) pel *shader* de color vermell (Red).

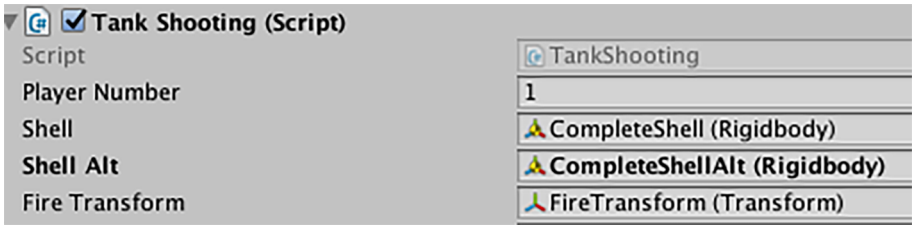
Figura 32. *CompleteShell*



A continuació, tornem a l'*script TankShooting* per modificar de nou el comportament. Per a això, necessitarem una propietat per al nou *prefab* (*CompleteShellAlt*) i desar el tipus de disparament que acabem de realitzar:

```
    public Rigidbody m_ShellAlt;
    private bool m_AltFire;
```

Figura 33. *Script TankShooting*



A continuació necessitem modificar la funció *FireButton()* per discriminar entre una acció i una altra:

```
private bool FireButton(int mode) {
            bool action = false;
            m_AltFire = false;

            switch (mode) {
            case 0:
                action = Input.GetButtonDown (m_FireButton);
                m_AltFire = Input.GetButtonDown (m_AltFireButton);
            break;
            case 1:
                action = Input.GetButton (m_FireButton);
                m_AltFire = Input.GetButton (m_AltFireButton);
            break;
            case 2:
                action = Input.GetButtonUp (m_FireButton);
                m_AltFire = Input.GetButtonUp (m_AltFireButton);
            break;
            }

            return action||m_AltFire;
        }
```

Per acabar, modifiquem lleugerament la funció *Fire()* perquè els disparaments diferenciïn entre els tipus de projectil i la velocitat associada a cadascun d'ells:

```
private void Fire ()  {
     m_Fired = true;
     Rigidbody shellInstance;
     if (m_AltFire) shellInstance =
        Instantiate (m_ShellAlt, m_FireTransform.position, m_FireTransform.rotation)
        as Rigidbody;
     else
          shellInstance =
          Instantiate (m_Shell, m_FireTransform.position, m_FireTransform.rotation)
           as Rigidbody;
     shellInstance.velocity = m_CurrentLaunchForce * m_FireTransform.forward;
     if (m_AltFire)
         shellInstance.velocity *= 1.50f;

     m_ShootingAudio.clip = m_FireClip;
     m_ShootingAudio.Play ();
     m_CurrentLaunchForce = m_MinLaunchForce;
 }
```

Repte 3

Hem de recalcular el *viewport* de cada jugador perquè sigui proporcional a la pantalla original, i la pantalla del primer jugador estigui alineada a l'esquerra i la del segon jugador a la dreta, com amb el cas del motor de gràfics.

En aquest cas, a partir de l'*aspect ratio* de 16:9, hem de recalcular l'amplitud i l'alçada de cada pantalla i tornar a reconfigurar la funció *AddCamera()* del *GameManager*, on s'estableixin aquests paràmetres en crear cada càmera:

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

En reduir cada càmera a una alçada de 0. 5, proporcionalment la longitud ha de ser 0.5/9 * 16 = 0.89 per ajustar-nos a l'*aspect ratio* de 16:9. Finalment, cal alinear la segona càmera ha al costat dret tot desplaçant la seva posició en l'eix X, restant al total (1) la nova longitud de 0.89.

Figura 34. *Aspect ratio*

Tot i que el resultat és l'esperat, òbviament aquest càlcul ha de ser dinàmic i us permetrà ajustar les dimensions de diferents proporcions d' *aspecte* que el joc pot tenir.

Repte 4

Activar la tècnica de pantalla partida quan els tancs siguin a un mínim de separació i desactivar-la quan es redueixi la distància per sota del mateix límit requereix modificar l'*script CameraControl* perquè tingui en compte les diferents càmeres que hi ha, tant l'original com les noves que hem afegit per a cada tanc.

D'aquesta manera, aquest *script* serà l'encarregat d'activar i desactivar cada càmera en el moment adequat. Hi afegirem propietats per mantenir el mode de **càmera partida**, tot sincronitzant la distància límit i una distància d'histèresi extra perquè en el cas de vorejar el límit no hi ha canvis bruscos contínuament.

```
        private bool splitMode = true;
        public float limitDist = 25.0f;
        public float hysteresis = 5.0f;
```

A continuació, afegim una nova funció anomenada *Split()*,a controlar en cada actualització si cal activar o desactivar el mode:

```
    private void FixedUpdate () {
                Move();
                Zoom();
                Split();
     }
```

Finalment, implementem aquesta funció de manera senzilla, tot activant o desactivant la càmera principal, que agafarà el relleu en cas d'activar-la o deixarà pas a les altres càmeres en cas de desactivar-la:

```
   private void Split() {
            float distance =
                    Vector3.Distance(m_Targets[0].position, m_Targets[1].position);

            if (splitMode && distance < limitDist-hysteresis) {
               splitMode = false;
                 m_Camera.gameObject.SetActive(true);
            }
            else if (!splitMode && distance > limitDist+hysteresis) {
                 splitMode = true;
                    m_Camera.gameObject.SetActive(false);
        }
    }
```

Repte 5

En aquest cas, crearem un nou *script* anomenat *CameraControl*, que associarem directament amb el *prefab CompleteTank* perquè, d'aquesta manera, cada tanc tingui el seu propi control de càmera associada i puguem tenir la màxima flexibilitat possible per determinar-ne el comportament.

```
        private Camera m_Camera;

        private void FixedUpdate () {
            if (m_Camera == null) {
```

```
            m_Camera = GetComponentInChildren<Camera> ();
            return;
        }
        Follow ();
    }

    private void Follow ()  {
            m_Camera.transform.position = Vector3.Lerp (
            m_Camera.transform.position, transform.position,
            Time.deltaTime * smooth);
    }
```

Tot recuperant la càmera associada al tanc i recalculant la seva posició aconseguirem veure cada tanc al mig de la pantalla. El problema que detectem en aquest punt és que la càmera acaba apropant-se massa a la posició del tanc, fins a ocupar la seva posició.

Frustum

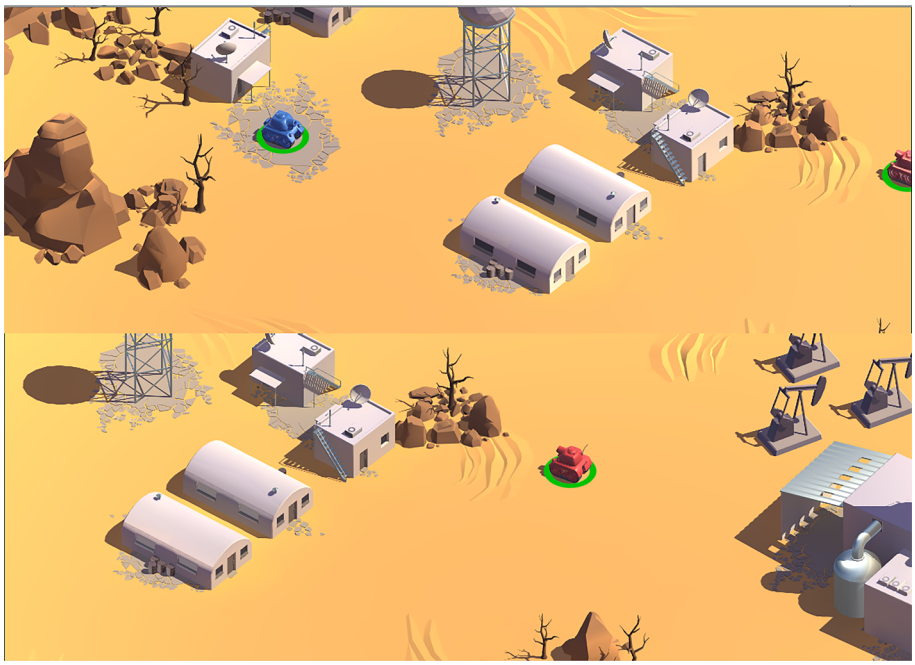
És la regió entre els dos plans de la càmera que defineix l'espai del món que apareixerà a la pantalla.

A més, hem de tenir en compte que es tracta d'una càmera ortogràfica i que l'alçada ha de ser suficient perquè tota escena entri dins el *frustum*. Per a això, mantindrem un límit d'aproximació entre la càmera i el tanc al *CameraControl*:

```
    private Camera m_Camera;
    public float smooth = 0.5f;
    public float limitDist = 20.0f;

    private void FixedUpdate () {
        if (m_Camera == null) {
            m_Camera = GetComponentInChildren<Camera> ();
            return;
        }
        Follow ();
    }

    private void Follow ()  {
             float currentDist =
            Vector3.Distance(transform.position, m_Camera.transform.position);

        if (currentDist > limitDist)
             m_Camera.transform.position = Vector3.Lerp (
             m_Camera.transform.position, transform.position,
             Time.deltaTime * smooth);
    }
```

D'aquesta manera, ara la càmera canvia de posició de manera progressiva, segons la posició que adquireixi el *GameObject* del jugador, sense apropar-se més del que li permetem.

Figura 35. Canvi de posició de la càmera

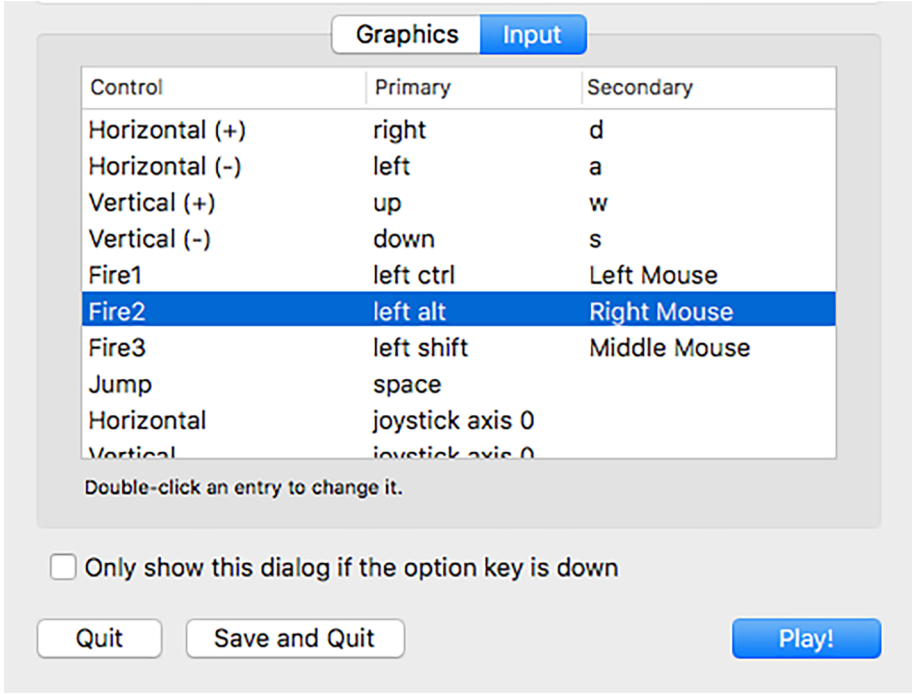


Repte 6

En primer lloc, genereu el projecte correctament per executar-lo com una aplicació al vostre sistema operatiu a partir de l'escena *_Complete-Game*.

Per modificar els controls d'entrada un cop generat el joc, Unity ens permet editar aquests controls mitjançant la seva pantalla d'arrencada del joc.

Figura 36. Pantalla d'arrencada del joc



A més, des d'aquí podeu configurar altres dispositius externs que no siguin el teclat per jugar al joc. A partir d'aquest punt, ja podeu provar l'exemple complet en local per a dos jugadors simultanis, tot completant el primer projecte de joc multijugador.

Resum

En aquest mòdul hem realitzat una primera aproximació al món dels videojocs multijugador, a partir de l'evolució d'aquesta modalitat en els videojocs al llarg de la història, tant el multijugador en xarxa com el multijugador local, tot provant d'englobar els diferents dispositius. Finalment, hem vist el desenvolupament i les tècniques per al multijugador aplicats a un projecte pràctic.

L'evolució lògica del multijugador com a modalitat en el món dels videojocs és transversal a tots els dispositius d'oci que han sorgit fins ara. Al principi, ajudant a la seva popularització, va tenir un paper el multijugador local, que després va evolucionar cap al multijugador en xarxa.

Hem insistit en videojocs icònics en aquesta evolució, atesa la seva aportació i la seva influència en aquesta àrea. Cal destacar especialment que l'evolució dels videojocs en el món multijugador en xarxa ha estat molt relacionada amb la innovació per part d'estudis pioners en aquest camp, i acompanyada de l'increment de les capacitats del maquinari, així com l'ús de xarxes tant locals com globals.

Finalment, hem practicat amb la modalitat local, amb la intenció d'establir un bon punt de partida per familiaritzar-nos amb el desenvolupament d'un videojoc multijugador. Concretament, hem treballat un exemple en Unity que ens ha permès estudiar la gestió i la modificació de les entrades provinents de diferents jugadors, i la implementació de la tècnica de pantalla partida, molt popular en el passat i que sembla irònicament estar vivint un nou ressorgiment en aquests temps d'alta connectivitat i múltiples possibilitats i facilitats per al joc en línia.

Bibliografia

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*. Nova York: The Random House Publishing Group.

Id Software (1996). *Quake World* [joc d'ordinador en línia].

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