UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Desarrollo de un videojuego multijugador y multiplataforma

MEMORIA PRESENTADA POR:

Eric Torregrosa Blasco

Tutor:

Jordi Joan Linares Pellicer

GRADO DE *INGENIERIA INFORMATICA*

Convocatoria de defensa Septiembre de 2016

ÍNDICE

1.INTRODUCCIÓN

1.1 Estructura del documento

1. Introducción, motivación, propósito y objetivos del proyecto.
   En este capitulo se explica el porque del desarrollo de este proyecto, cuales son los objetivos propuestos y como se ha organizado este documento

2. Estudio y estructura del proyecto
   En este capitulo se justifica los parámetros que hacen posible el desarrollo de este proyecto y se definen los diagramas y esquemas seguidos para el desarrollo del proyecto.

3. Descripción de las herramientas utilizadas
   En esta sección se describen las herramientas utilizadas, poniendo énfasis en por que de la elección de dichas herramientas.
   Se habla sobre Unity 3D.

4. Unity en Red UNET
   Se describen los componentes más importantes sobre el Networking, se enumeran los principales componentes para trabajar en red.

5. Desarrollo del proyecto.
   Este capítulo trata sobre el proyecto realizado, se enumeran y describen los componentes utilizados, las funcionalidades del proyecto y se describe en más profundidad la mecánica del juego.

6. Interfaz de juego
   En esta sección se describen los elementos que componen la interfaz de usuario.

7. Modelos 3D utilizados
   En esta sección se describen los modelos en 3D utilizados en el juego, se describen los personajes principales y el mapa principal.

8. Controles del juego
   Este capítulo describe los controles utilizados en el juego para mover el personaje principal.

9. Resultados
   En esta sección se incluyen capturas de pantalla y videos del proyecto, se describen las características finales y el mercado al que va destinado el producto.

10. Trabajo futuro
    En esta sección se describen elementos y funcionalidades que se desean implementar en un futuro para mejorar y ampliar la aplicación.

11. Conclusiones finales
    En esta sección se enumeran las conclusiones a las que ha llegado el autor al finalizar el proyecto.

12. Bibliografía
    Se muestran las fuentes utilizadas para el desarrollo del proyecto.

13. Anexos
    Se muestran los anexos a los distintos elementos utilizados en este documento.

1.2 Motivación

Mi motivación para realizar el este proyecto ha sido ponerme a prueba a mi mismo en el hecho de crear algo desde cero partiendo de mi creatividad y conocimientos de programación.

El hecho de desarrollar un videojuego en mi caso es muy motivador, ya que es una manera divertida y a la vez productiva de desarrollar aplicaciones.

Partiendo de tener conocimientos previos sobre el desarrollo de videojuegos en Unity 3D, he querido dar un paso más y meterme el mundo del Networking.

Por lo tanto, mi objetivo es tener un producto acabado e interesante, y que este proyecto pueda servir de guía e inspiración para todos aquellos que les entusiasme tanto como a mi los videojuegos y la programación.

1.3 Propósitos y objetivos

El objetivo de este proyecto de final de grado es crear un videojuego de tipo arcade – supervivencia multijugador para dispositivos móviles en 3D.

Donde partiendo de que cada personaje tiene su base y sus torres, así como también cada personaje tiene sus aliados PNJ (Personajes no jugador) que te apoyan al largo de la partida.

El usuario podrá interactuar con su entorno y interactuar en el. El motor de juego ha de permitir exportar el juego a plataformas como PC y Mac, y plataformas móviles como Android o IOS.

Para este proyecto, se ha optado por elegir como plataforma principal Android, el motivo la elección de esta plataforma es la gran cantidad de dispositivos móviles que hay en el mercado con el sistema operativo Android, por la facilidad de publicar la aplicación en el Play Store de Google, y por el ahorro de recursos financieros para para poder publicar esta aplicación.

2. ESTUDIO Y ESTRUCTURA DEL PROYECTO

2.1 Estudio de viabilidad

Para el desarrollo del proyecto no se requiere de una gran infraestructura y los costos de estructura son inexistente.

El material usado para el desarrollo se compone de :

- Ordenador con sistema operativo Windows 10.
- Unity 3D versión 5 o superior, motor de videojuegos multiplataforma.
- Tarjeta gráfica con soporte para DirectX 9.0 y Shader Model 2.0
- Conexión a Internet.
- Dispositivo móvil con Sistema Operativo Android, preferiblemente con 1GB de Memoria RAM y GPU Adreno o superior.

2.2 Diagramas y casos de uso

Diagrama estructura de trabajo

En el siguiente diagrama se define la estructura al realizar el proyecto, y las decisiones que se realizan durante todo el ciclo de vida de la aplicación.

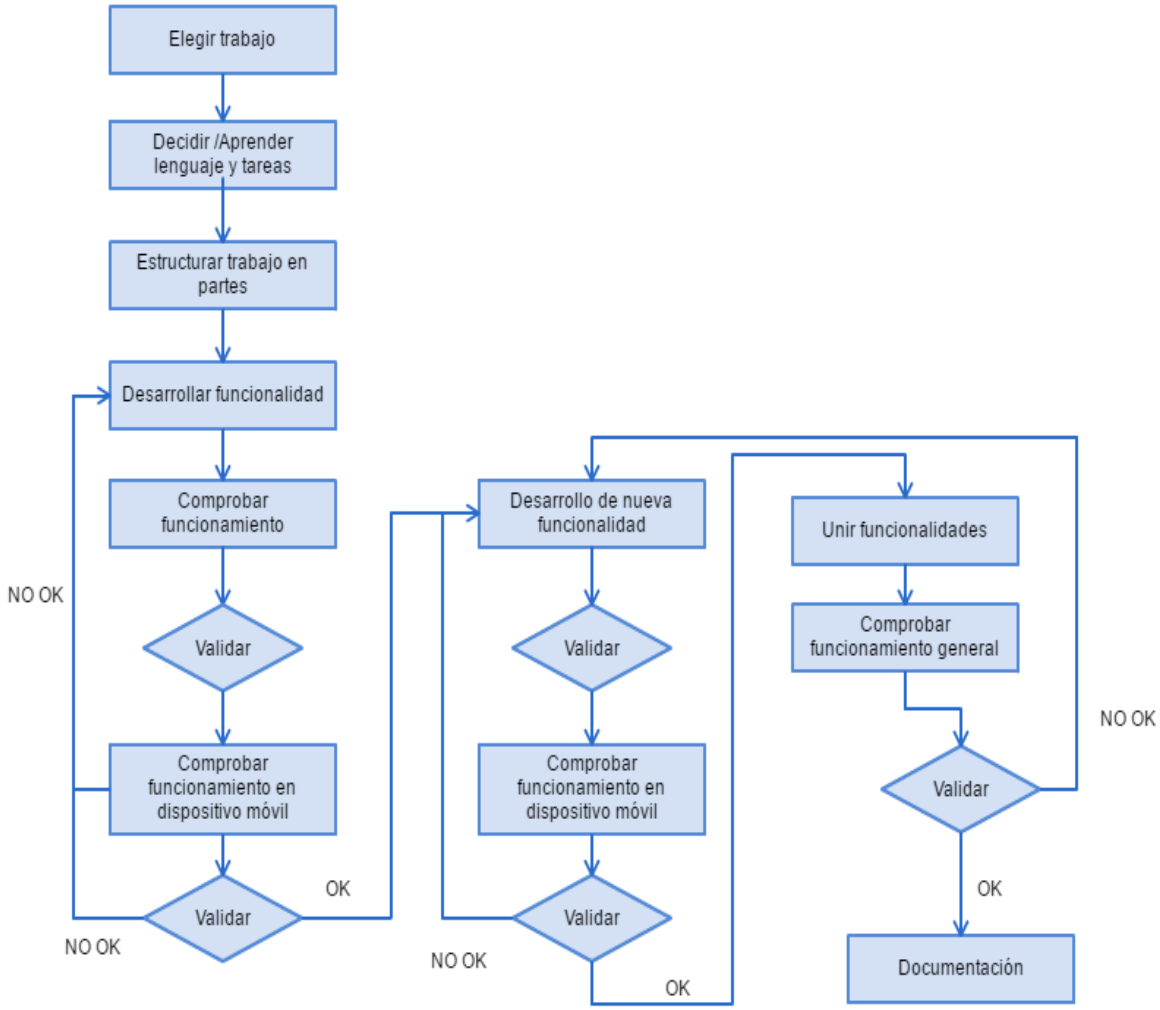


Ilustración 1 Diagrama estructura de trabajo

Diagrama de Networking

En el siguiente diagrama, se muestra como se comporta el juego respecto a si es un servidor/cliente o es un cliente el jugador.

Como se puede apreciar al realizar una función o un comando el cliente necesita comunicarse con el servidor, y este le devuelve una respuesta al cliente.

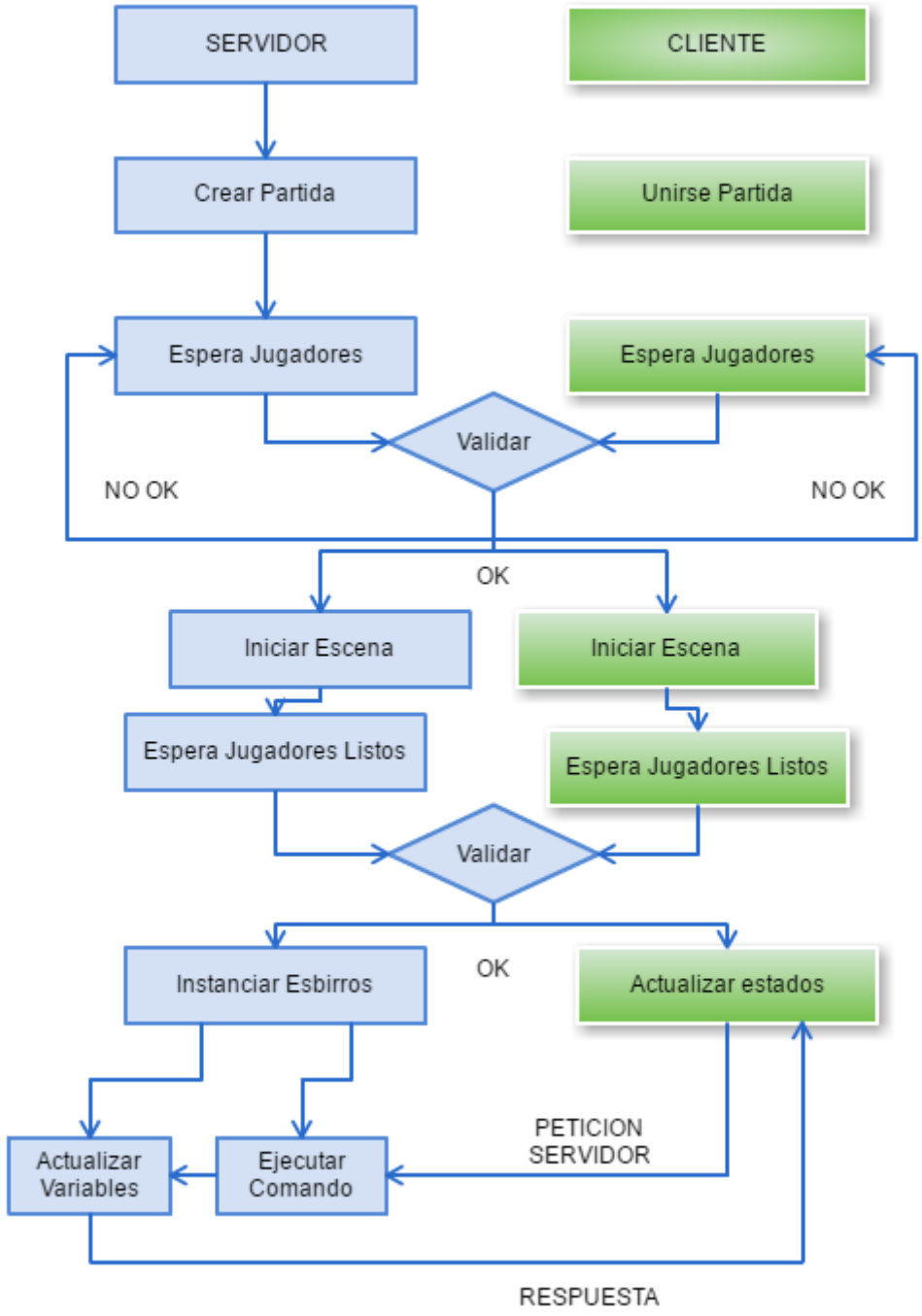


Ilustración 2 Diagraama de Networking

Casos de uso

Los casos de uso describen el comportamiento de un sistema desde el punto de vista de un usuario.

Permiten definir los límites del sistema y las relaciones entre este y el entorno. Podemos decir que son descripciones de las funcionalidades del sistema, independientemente de su implementación.

Los casos de uso están basados en el lenguaje natural, de manera que pueden ser accesibles para todos los usuarios.

Al iniciarse el juego, aparece un menú que contiene dos opciones: Crear partida online o buscar partidas.

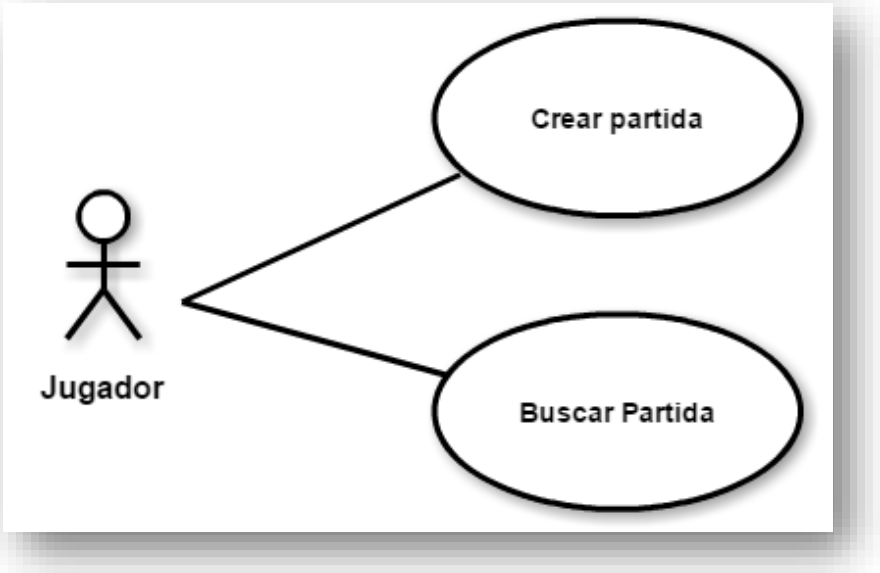


Ilustración 3 Diagrama casos de uso 1

Al pulsar sobre crear partida nos muestra la vista de esperando jugadores con un botón para verificar que estás listo para comenzar la partida.

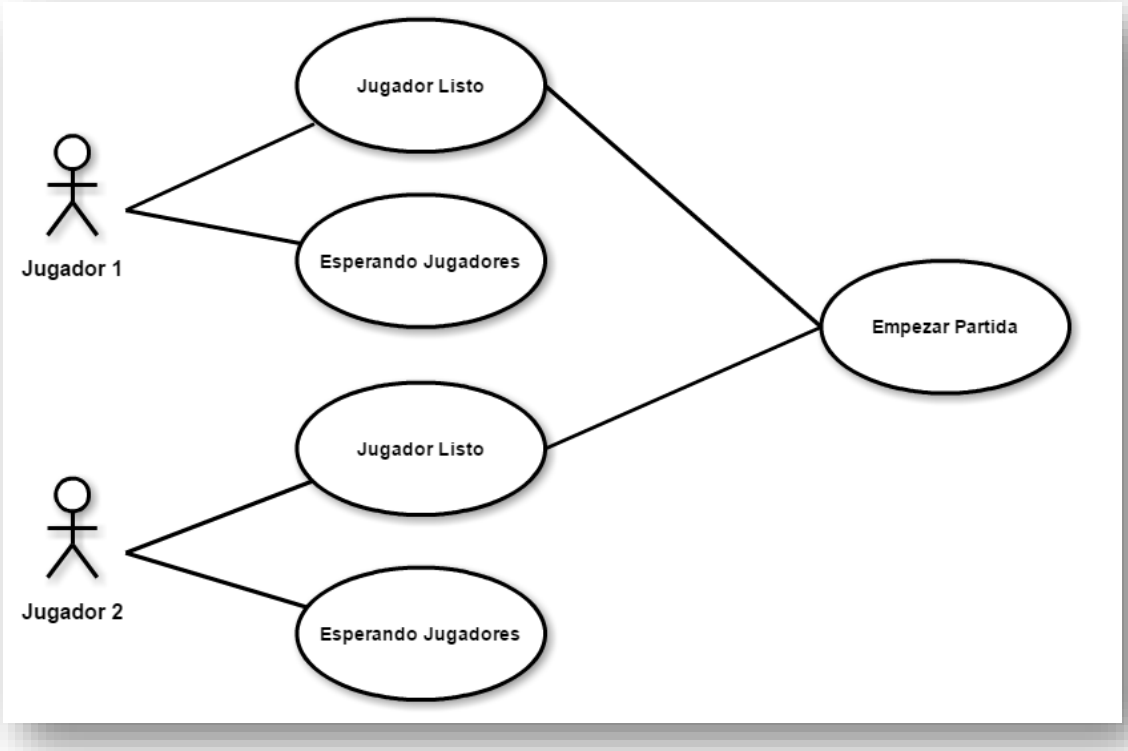
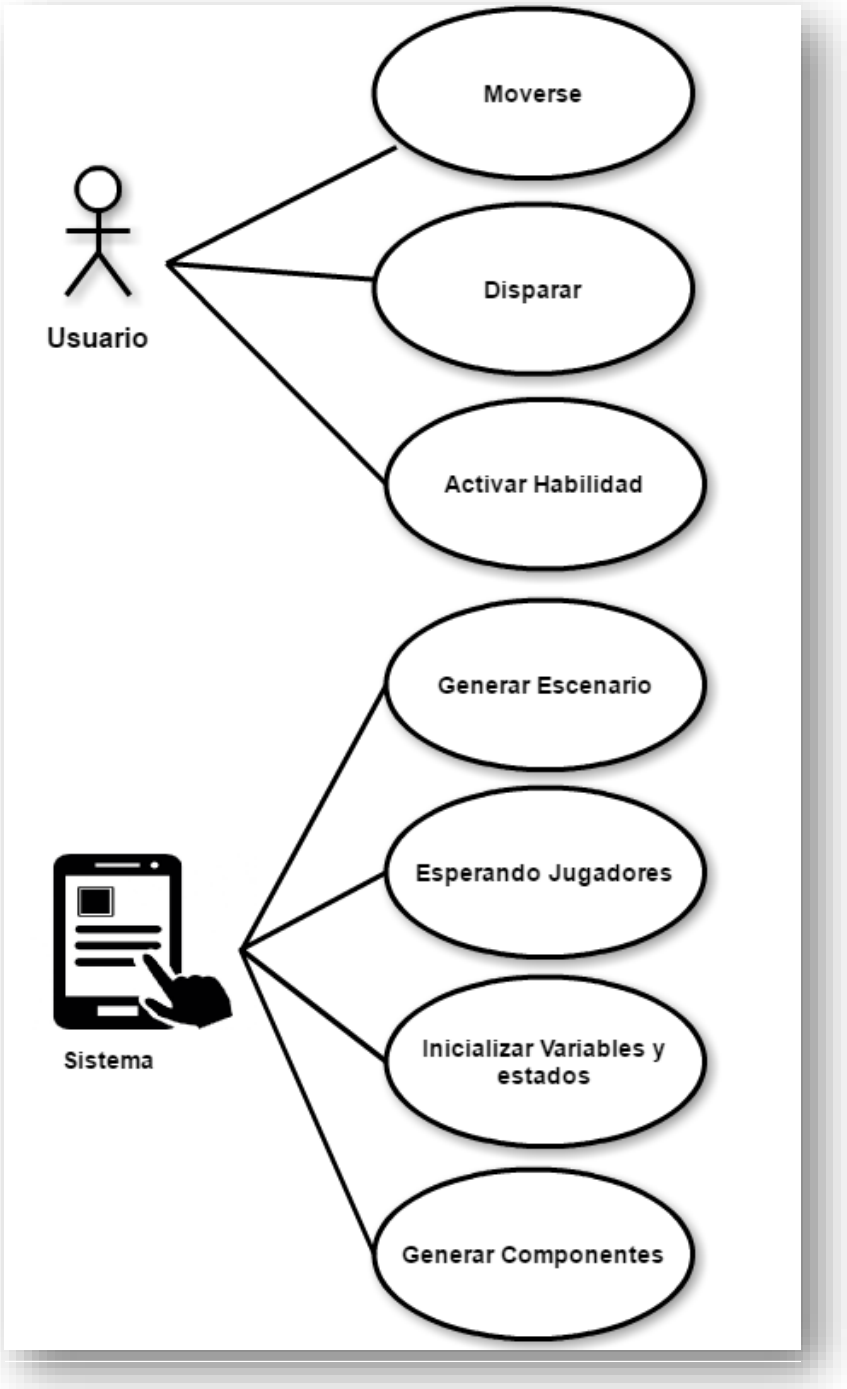


Ilustración 4 Diagrama casos de uso 2

Diagrama de inicio de partida

En el siguiente diagrama se puede ver un caso de uso al iniciar una partida, mostrando el punto de vista del usuario con respecto al del dispositivo.



3. DESCRIPCIÓN DE LAS HERRAMIENTAS MÁS UTILIZADAS

3.1 Sobre Unity 3D

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X y Linux, y permite crear juegos para Windows, OS X, Linux, Xbox 360, PlayStation 3, PlayStation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone.

A partir de su versión 5.4.0 ya no soporta el desarrollo de contenido para navegador a través de su plugin web en su lugar se utiliza WebGL.

Unity tiene dos versiones: Unity Professional (pro) y Unity Personal (gratuita).

3.2 Historia de Unity 3D

La aventura de Unity Tecnhologies empezó en el año 2004 cuando David Helgason, Nicholas Francis y Joachim Ante decidieron dar un vuelco a su compañía de desarrollo de videojuegos tras el fracaso de _GooBall'.

Ilustración 6 Diagrama captura juego _GooBall

El juego no había tenido el éxito esperado pero en su desarrollo habían creado unas herramientas muy potentes que sirvieron como semilla para una idea que rondaba la cabeza del equipo: democratizar el desarrollo de videojuegos.
Crear un motor de videojuegos que pequeñas y grandes empresas pudieran utilizar por igual. Un entorno amigable para programadores, artistas y diseñadores que llegase a diferentes plataformas sin obligar a programar el juego específicamente para cada una de ellas. Sí, una especie de utopía hace diez años y que sin embargo, a día de hoy, se ha convertido en realidad.

El motor llegaría solo a Mac en principio y se presentaría en dos versiones, *Indie* y Profesional. La primera representaba un punto de acceso económico para los pequeños estudios que estaban empezando y no podían permitirse un dispendio considerable, tenía muchas funciones y su precio empezaba en 300 dólares.

La versión Pro, de 1.500 dólares de coste, traía todas las funciones del motor. Es importante señalar que incluso en la versión *Indie* del motor se permitía vender el producto resultante. De momento el motor funcionaba bien pero estaba lejos de ser una alternativa al mismo nivel de los grandes.

En 2008 llegaría el gran salto al aprovechar el lanzamiento del iPhone, la fiebre que desató y la compatibilidad con la plataforma. Un poco más tarde llegaría la versión para Android.

Ilustración 7 Diagrama imagen iphone

Ilustración 8 Diagrama imagen Android

El auge de Unity era imparable. En 2009 la compañía dio el espaldarazo definitivo a su estrategia. La versión *Indie* desaparecía como tal y se convertía en gratuita para todo aquel que quisiera iniciarse en la plataforma. Incluso con esa versión, la gratuita, se permitía distribuir el juego. Los desarrolladores independientes no tardaron en abrazar la idea y convirtieron a Unity en uno de los motores gráficos más usados. Eso sí, todavía faltaba dar un salto de calidad para conquistar a los estudios de desarrollo que exigían más calidad gráfica. Un salto que llegó con la versión 4.0 y que demuestran los acuerdos cerrados con Sony, Microsoft y Nintendo para que el motor sea compatible con sus sistemas. La versión actual es compatible con muchísimas plataformas (PC, Mac, Linux, iOS, Android, BlackBerry, PlayStation, Xbox, Wii, Wii U, Web…) y el sueño de desarrollar una sola vez, darle a publicar y olvidarte de problemas está más cerca que nunca.

3.3 Unity y Networking

Unity nos incorpora una serie de herramientas muy particulares que nos permiten trabajar sobre recursos en red, lo que se denomina Networking.

A partir de la versión 5 de Unity, se liberó una serie de herramientas para el desarrollo de proyectos en red, esta API es llamada Unity UNET.

UNET nos proporciona una serie de elementos que nos permiten trabajar de forma abstracta sobre componentes de Unity ya utilizados para convertirlos en elementos en red.

También nos incorpora un extensión del propio editor y en concreto una clase llamada NetworkBehaviour que nos proporciona todos los elementos, funciones, y demás funcionalidades para el desarrollo de la programación de elementos en red.

3.4 Por que Unity 3D?

La elección de un motor de juego a otro puede influenciar mucho a la hora de decidirse y decantarse por una plataforma u otra, puede que tengamos pensado desarrollar un juego para una plataforma en concreto y en tal caso igual la elección de un motor de juego puede ser decisiva e importante.

En mi caso, me he decantado por elegir Unity 3D por la simplicidad de poder exportar mi proyecto a una plataforma u otra de manera muy rápida y sencilla.

En mi caso puedo estar desarrollando un videojuego para PC y llegado un punto querer adaptarlo y exportarlo para plataformas móviles como Android o IOS, fácilmente con pocos cambios en el diseño y la programación puede hacerlo con Unity 3D.

Otro aspecto por el que me decanto por elegir Unity 3D, es por el coste en licencias, prácticamente con la versión FREE de Unity 3D puedes desarrollar cualquier tipo de videojuego con muy pocas restricciones.

4. UNITY EN RED UNET

4.1 Networking en Unity 3D

4.1.1 API de scripting de Alto nivel

Unity en términos de networking, nos ofrece una API de Scripting de Alto nivel (HLAPI).

Esto nos proporciona acceso a los comandos que cubren la mayoría de requerimientos comunes para juegos multijugador sin la necesidad de preocuparse acerca de los detalles de la implementación de bajo nivel.

La HLAPI nos permite entre otras cosas:

- Controlar el estado de red del juego utilizando un "Network Manager" (Administrador de Red).
- Alojar juegos "hosted (alojados) al cliente", dónde el host es también un cliente jugador.
- Serializar datos utilizando un serializador con propósito general.
- Enviar y recibir mensajes de red.
- Enviar comandos de red de clientes a servidores.
- Realizar llamados de procedimientos remotos (remote procedure calls-RPCs) de servidores a clientes.
- Enviar eventos de red de servidores a clientes.

4.1.2 Integración del Motor y el Editor

El servicio de networking UNET en Unity esta integrado en el propio motor y en el Editor.
Esto nos permite trabajar con componentes y ayudas visuales para construir un juego multijugador, entre otras cosas nos proporciona.

- Un componente de NetworkIdentity para objetos en red.
- Un NetworkBehaviour para scripts en red.
- Una sincronización automática configurable de los transforms de los objetos.
- Una sincronización automática de variables de script.
- Soporte para colocar objetos en red en escenas de Unity.
- Network components

4.1.3 Servicios de Internet

Unity nos ofrece servicios de internet para soportar el juego a través de la producción y la salida del producto, el cual incluye:

- Un servicio de crear partidas.
- Crea partidas publicarlas.
- Listar las partidas disponibles y unirse a ellas.
- Servidor de re-transmisión
- Game-play sobre Internet sin un servidor dedicado.
- Enrutamiento de mensajes para los participantes de las partidas.

NetworkTransport capa de transporte en tiempo real.

Se incluye una Capa de transporto en tiempo real que ofrece:

- Un protocolo basado en un UDP optimizado.
- Un diseño multi-canal para evitar problemas de head-of-line blockin (HOL)
- soporte para una variedad de servicios de calidad (Quality of Service - QoS)) por canal.
- Una topología de red flexible que soporta arquitecturas de persona a persona o de cliente a servidor.

Esquema de la capa de transporte en red

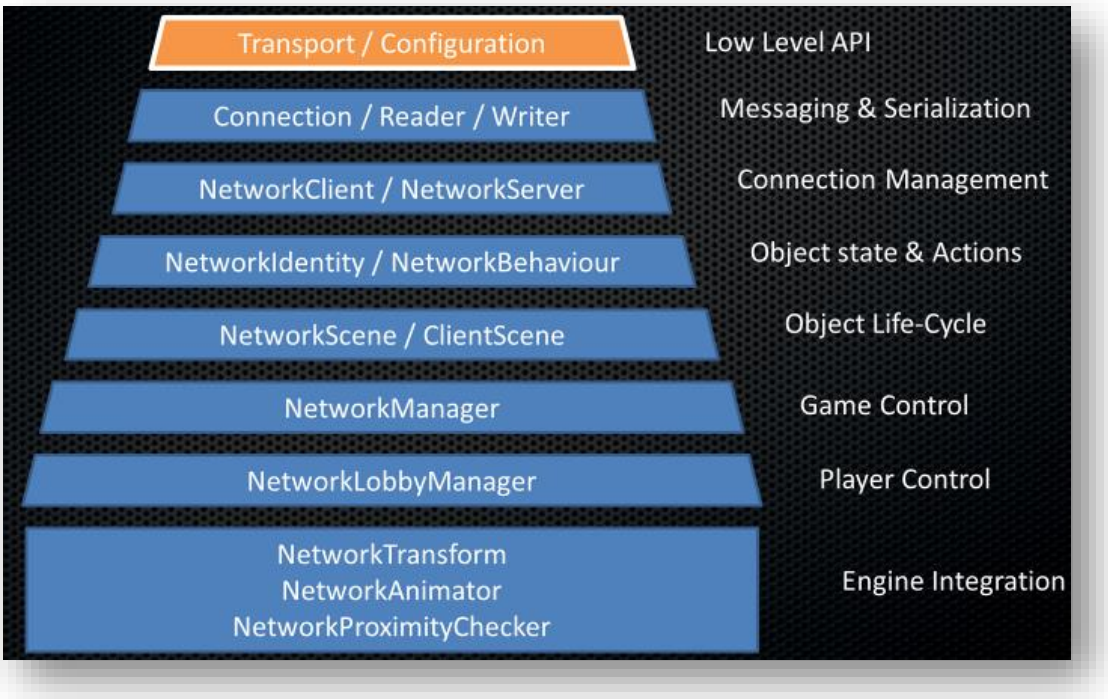


Ilustración 9 Diagrama Esquema de capa de transporte en red

4.1.4 Servidor y Anfitrión

En el sistema de red de Unity, los juegos tienen un servidor y múltiples clientes Dónde no hay un servidor dedicado, en este caso uno de los clientes juega el rol del servidor - nosotros llamamos este cliente el "anfitrión (Host)".

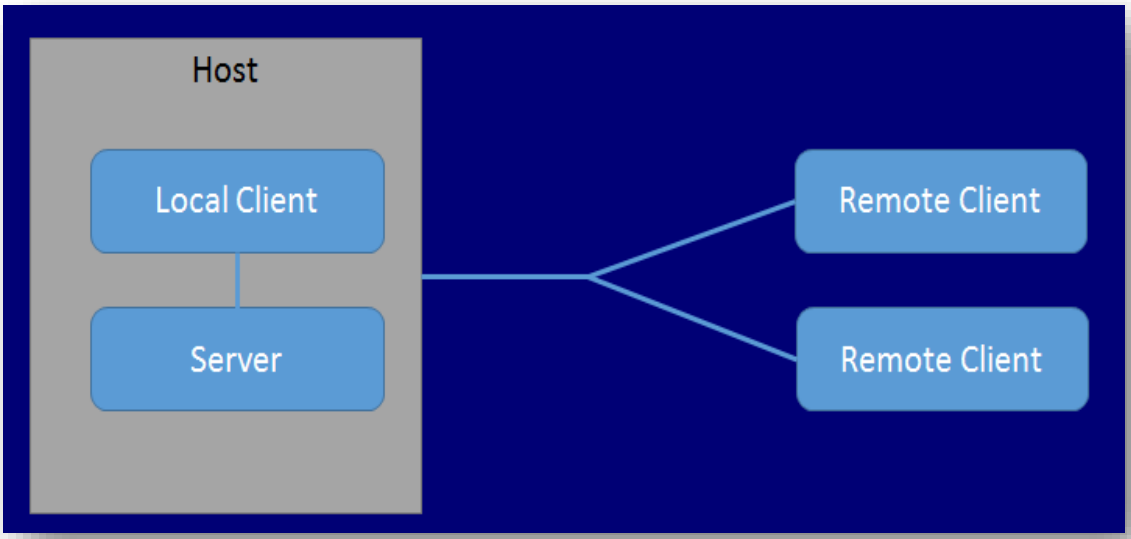


Ilustración 10 Esquema servidor y anfitrión

El anfitrión es un servidor y un cliente en el mismo proceso.

El anfitrión utiliza un tipo especial de cliente llamado LocalClient (cliente local), mientras otros clientes son RemoteClients (clientes remotos).

El LocalClient (cliente local) se comunica al servidor (local) a través de llamadas de función directas y colas de mensajes, ya que está en el mismo proceso. En realidad comparte la escena con el servidor. Los RemoteClientes (clientes remotos) se comunican con el servidor sobre una conexión regular de red.

Esquema de juego en área local

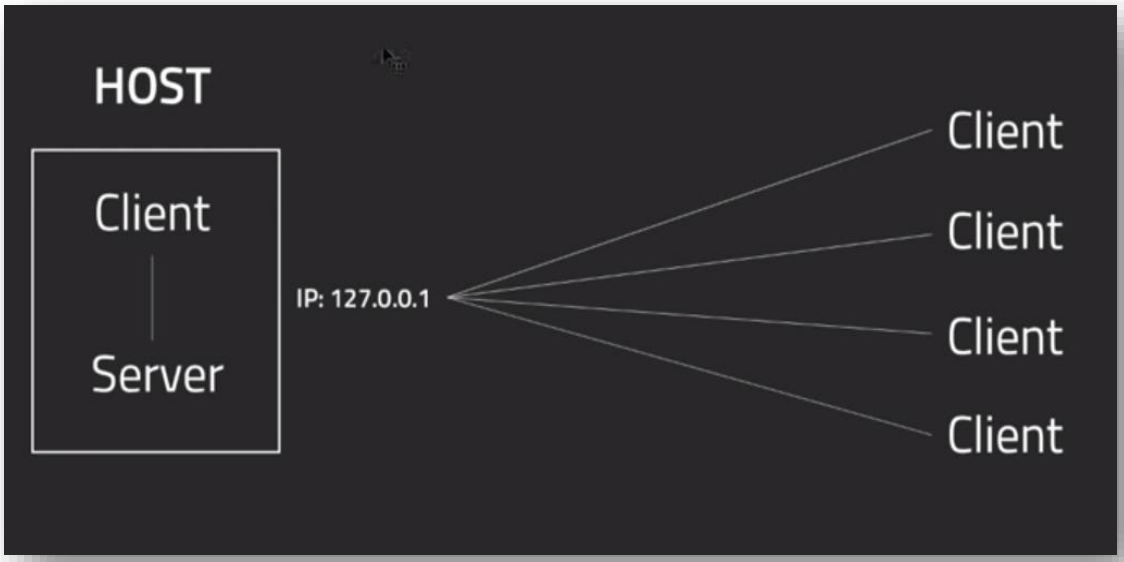


Ilustración 11 Esquema de juego en área local

Esquema de juego por Internet

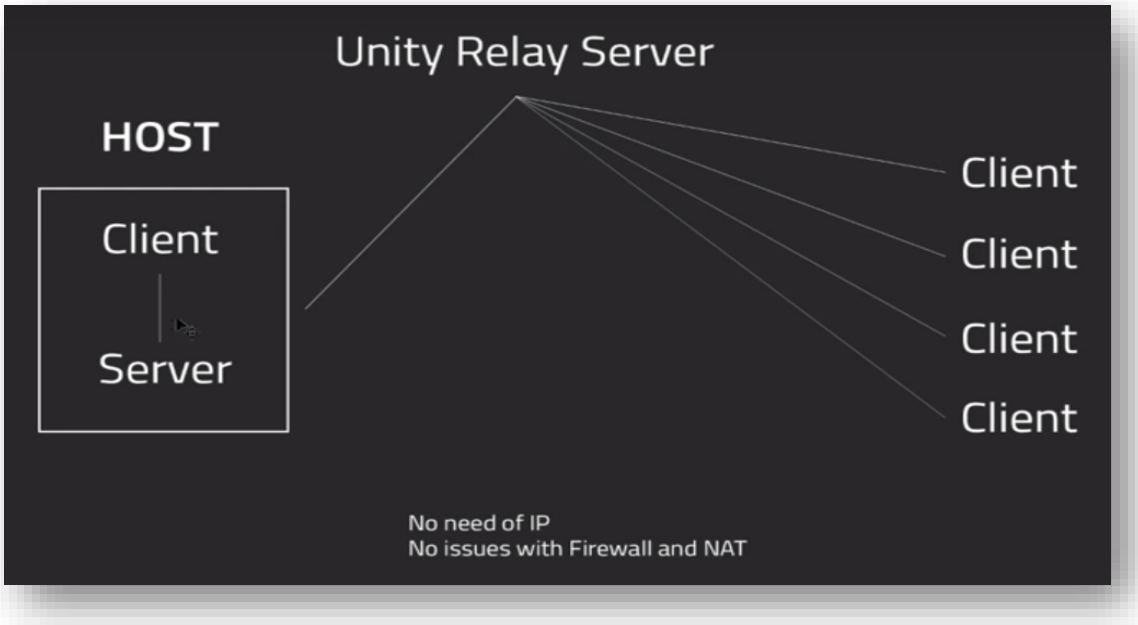


Ilustración 12 Esquema de juego por Internet

Como se puede apreciar en el esquema, el juego por Internet, no necesita de un servidor de juego dedicado, el propio cliente hace de servidor y cliente.

Tampoco necesita conocerse la dirección IP para poder establecer la conexión y comunicación entre clientes y servidor.
No se necesita definir reglas en el Firewall ni configuración adicional para la configuración del NAT.

4.1.5 Instanciación y generación

En Unity, GameObject.Instantiate crea nuevos game objects de Unity. Pero con el sistema de red, los objetos también deben ser “spawned (generados)” para estar activos en la red.

Esto solamente se puede hacer en el servidor y causa los objetos en ser creados en clientes conectados.

Una vez los objetos estén generados, el Spawning System (sistema de generación) hace la administración del ciclo de vida del objeto distribuido y la sincronización de estados.

4.1.6 Jugadores y autoridad

En el sistema de red, los objetos del jugador son especiales. Hay un objeto jugador asociado con cada persona jugando el juego, y los comandos son enrutados a ese objeto.

Una persona no puede invocar comandos en un objeto jugador de otra persona - solamente en él mismo. Por lo que hay un concepto de “mi” objeto jugador. Cuando el jugador es agregado y hay una asociación hecha con una conexión, cuando un jugador es agregado y la asociación es hecha con una conexión, ese objeto jugador se vuelve un objeto “jugador local” en el cliente de ese jugador.

Hay una propiedad isLocalPlayer que es configurada a true, y un callback OnStartLocalPlayer() que es invocado en el objeto en el cliente.

El esquema de abajo muestra dos clientes y sus jugadores locales.

Esquema autoridad sobre servidor

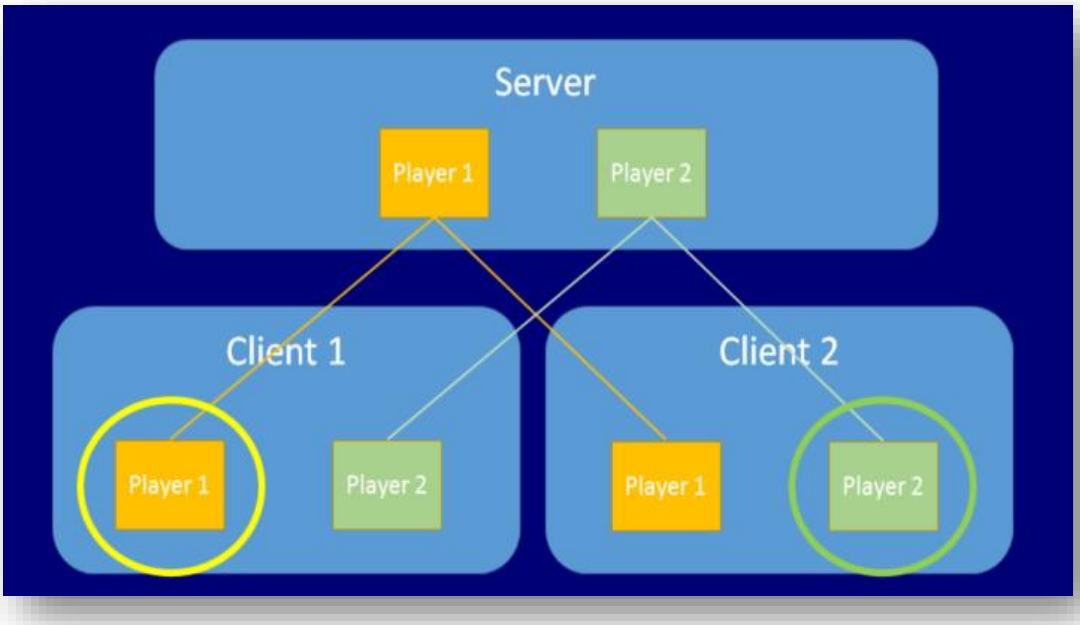


Ilustración 13 Esquema autoridad sobre servidor

Solamente el objeto jugador que es "suyo" tendrá la flag isLocalPlayer configurada. Esto puede ser utilizado para filtrar un procesamiento de input, para manejar anexos de cámara, o hacer otras cosas del lado del cliente que solo deberían ser hechas para su jugador.

En adición a isLocalPlayer, un objeto jugador puede tener su "local authority (autoridad local) ". Esto significa que el objeto jugador en el cliente de su dueño es responsable para el objeto - este tiene autoridad sobre este. Esto es utilizado más común para controlar movimiento, pero puede ser utilizado para otras cosas también. El componente NetworkTransform entiende esto y va a enviar movimiento del cliente si esté está habilitado.

El NetworkIdentity (Identidad de red) tiene una casilla de verificación para configurar LocalPlayerAuthority (Autoridad del jugador local).

Para objetos no jugadores, tal como enemigos, no hay un cliente asociado, por lo que la autoridad reside en el servidor.

Esquema Autoridad sobre objetos no jugadores

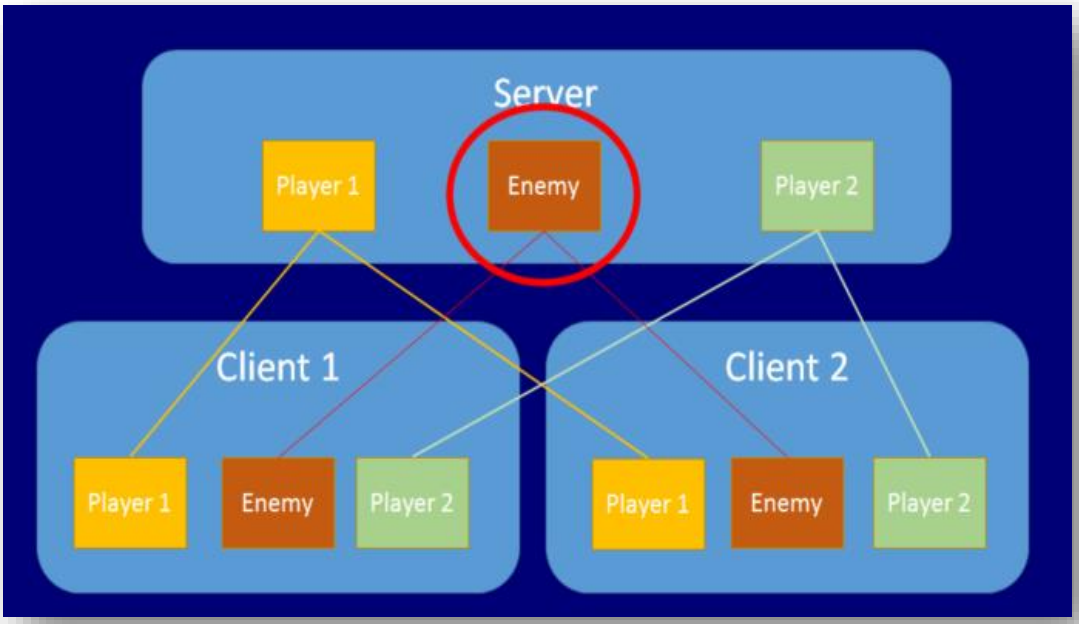


Ilustración 14 Esquema autoridad sobre objetos no jugadores

Hay una propiedad “isAuthority” en el NetworkBehaviour (comportamiento de red) que puede ser utilizado para decir si un objeto tiene autoridad. Por lo que objetos no jugadores tienen autoridad en el servidor, y objetos jugadores con localPlayerAuthority configurado tienen autoridad en el cliente de su dueño.

4.1.7 Autoridad de Cliente para Objetos no jugadores

Con la actualización de Unity 5.2, ya es posible tener autoridad de cliente sobre objetos no jugadores.

Para ello hay dos maneras de efectuarlo:

- Una vez instanciado el objeto, utilizando NetworkServer.SpawnWithLocalAutothority, y pasar la conexión de red del cliente como argumento, para tomar propiedad del objeto.

- Otra manera es utilizar NetwrokItdentity.AssingClientAuthority, asignando la conexión de red del cliente para tomar propiedad sobre el objeto.

Asignar autoridad a un cliente causa OnStartAuthority() en ser llamado en NetworkBehaviours en el objeto, y la propiedad hasAuthority será true. En otros clientes, la propiedad hasAuthority será falsa. Los objetos no-jugadores con autoridad de cliente puede enviar comandos, tal como los jugadores pueden. Estos comandos corren en la instancia del servidor del objeto, no en el jugador asociado con la conexión.

Los objetos no jugadores que tienen autoridad de cliente deben tener LocalPlayerAuthority marcado en su NetworkIdentity.

Este ejemplo de abajo genera un objeto y le asigna autoridad al cliente del jugador en el cual fue generado.

```
[Command]

void CmdSpawn() {

var go = (GameObject)Instantiate(otherPrefab, transform.position + new Vector3(0,1,0),
Quaternion.identity);

   NetworkServer.SpawnWithClientAuthority(go, connectionToClient);

}
```

4.1.8 Propiedades del contexto de la red

Hay propiedades en la clase NetworkBehaviour que le permite a script saber cuál es el contexto de la red de un objeto en red en cualquier momento.

- isServer - true si el objeto está en un servidor (o anfitrión-host) y ha sido generado.
- isClient - true si el objeto está en un cliente, y fue creado por el servidor.
- isLocalPlayer - true si el objeto es un objeto jugador para este cliente.
- isAuthority - true si el objeto es propiedad del proceso local

Estas propiedades pueden ser vistas en la ventana de pre-visualización del objeto en la ventana del Inspector del editor.

4.2 Principales componentes de UNET

4.2.1 Network Transform

Este componente se encarga de la sincronización de un elemento en red, encargándose de sincronizar su estado, así como su posición, rotación y escala. Podemos definir varios valores de configuración, así como el tiempo que tarda en enviar datos, el método de sincronismo, y que es lo que queremos sincronizar.
Entre ellos, se puede sincronizar solo el transform, rigidbody, charactercontroller.

4.2.2 NetworkIdentity

NetworkIdentity

Este componente, consiste básicamente en ser un identificador único de objeto en red.

Cada gameobject o componente que tenga comportamiento en red, debe tener un networkidentity asociado.

En el editor podemos establecer la identidad del componentes estableciendo si es solo de servidor o tiene autoridad local.

-Server Only
-Local Player Authority

4.2.3 NetworkAnimator

Este componente se encarga de la sincronización de un animator en red, al utilizar este componente sobre un modelo, y asociarle su correspondiente animator, automáticamente se encarga de gestionar el sincronismo de animaciones entre clientes y servidor.

De esta manera podemos definir el comportamiento de las animaciones solo en el servidor, y olvidarse en los clientes de su comportamiento, ya que automáticamente se nos actualizara su estado de animación.

4.2.4 SpawnPoint

Este elemento se usa cuando invocamos un Player desde el networklobby, automáticamente dicho player se posiciona en la posición del gameobject que contenga el componente SpawnPoint.

Al configurar nuestro lobbymanager, podemos elegir el modo invocación de los players, estableciendo una cola de prioridad, y asignando a los jugadores en función de quien llega primero (Round Robin) el primer SpawnPoint cercano.

4.2.5 Sincronización de variables y estados en red

Sincronizar variables y estados en red

En UNET tenemos varias maneras de sincronizar estados y variables, entre ellas voy a enumerar las más utilizadas y las que he utilizado en el desarrollo del proyecto.

4.2.5.1 Sincronización de variables

Para poder sincronizar variables primitivas, como enteros, floats y vectores (Vector3), primero que todo tenemos que extender de la clase NetworkBehaviour.
Luego antes de la definición de una variable, debemos añadir su correspondiente etiqueta para enumerar cada variable y decirle a Unity que la variable que estamos usando debe estar sincronizada.

Ejemplo:

```
[syncvar]
public int vida = 0;
```

En este ejemplo estamos declarando una variable de tipo entero llamada vida, y con la etiqueta syncvar la mantenemos sincronizada.

NOTA: Importante, cada destacar el funcionamiento de las variables syncvar, dada una variable sincronizada, mantiene su estado con respecto al servidor, es decir los clientes obtienen el valor de la variable que el servidor posee, y el servidor es el único que puede cambiar su valor sincronizado para que se refleje en los clientes.

Si un cliente cambia directamente el valor de la variable, solo será visible para el localmente, pero no se reflejara en el servidor, en cambio si el servidor cambia el valor de la variable, automáticamente para todos los clientes obtendrán el valor sincronizado con respecto al servidor.

4.2.5.2 Comandos y funciones remotas

En UNET, podemos llamar a funciones de servidor, o funciones a modo de broadcast para todos los clientes.

Esto nos permite tener control sobre varios clientes al mismo tiempo.
Para poder utilizar estos comandos es necesario, que el objeto que contenga el script tenga autoridad para poder ejecutar comandos de servidor, esto se traduce que en que este marcado como LocalServerAutority, de esta manera le damos permisos al objeto para que pueda ejecutar comandos de servidor.

NOTA: aunque un cliente ejecute un comando de servidor, realmente lo que hace es decirle al servidor que invoque este comando, pero en si el cliente no esta ejecutando el comando localmente.

Esto es importante recalcarlo para entender el funcionamiento de los comandos.

Por poner un ejemplo, el cliente ejecuta un comando que llama a una función que se encarga de hacer saltar al personaje.

Al ejecutar la función, esta se ejecutara en el servidor y realizara el comportamiento en el servidor.

Command [Command]

Para invocar una función en el servidor, es necesario añadir la etiqueta [Command] antes de definir la función, también toda función debe tener como nombre el prefijo Cmd_ si no se cumplen estos requisitos la función no se ejecutará en el servidor y no tendrá el comportamiento esperado.

Ejemplo:

```
[Command]
public void Cmd_Disparar(){
        // Codigo de disparar
}
```

ClientRpc [ClientRpc]

Mediante la etiqueta [ClientRpc], podemos llamar a una función en todos los clientes al mismo tiempo, podría definirse como una función de broadcast en red.

Es muy útil para definir y ejecutar un comportamiento de un objeto en concreto en red, y que todos los clientes lo ejecuten simultáneamente.
Cuando se quiera hacer un cambio que se quiere reflejar en todos los clientes, esta etiqueta es muy útil.

Para definirla debemos añadir la etiqueta [ClientRpc] antes de declarar la función y posteriormente definir el nombre de la función con el prefijo Rpc_

Ejemplo

```
[ClientRpc]
public void Rpc_CambiarColor(){
        // código de cambiar color
}
```

Server [Server]

Mediante la etiqueta [Server] definimos el comportamiento de una función para que se ejecute solo, y únicamente en el servidor, aunque un cliente llame a esta función, solo se ejecutara en el servidor.

Para definirla simplemente hay que añadir la etiqueta [Server] antes de declarar la función.

Ejemplo:

```
[Server]
public void FuncionSoloServidor(){
        // código de la función
}
```

5. DESARROLLO DEL PROYECTO

5.1 Estructura del proyecto

Esquema de estructura del proyecto

Se ha definido una estructura por carpetas en el proyecto, agrupando los diferentes elementos en dichos directorios.

- **Escenas**: contiene las escenas del juego.
- **Materials**: contiene los materiales utilizados en el proyecto.
- **Scripts**: contiene varios directorios asociados a los elementos del proyecto, dentro de dichos directorios se agrupan los scripts utilizados en el proyecto.
- **Prefabs**: contiene los prefabs utilizados en el proyecto.
- **Fonts**: contiene los distintos tipos de tipografía utilizados para la interfaz del proyecto.
- **Recursos externos**: contiene todos los assets y componentes externos al proyecto que se han utilizado para el desarrollo, entre ellos contienen elementos descargados de la Asset Store.
- **Sounds**: contiene los sonidos utilizados en el juego.
- **Textures**: contiene las texturas utilizadas en el juego, principalmente para la creación del mapa.
- **Editor**: contiene referencias, componentes y scripts utilizados por el Editor para extender las funcionalidades del propio editor de Unity.
- **Standard Assets**: contiene componentes importados de unity, entre ellos se encuentras, Character Controller, sistemas de particulas, skybox…

5.2 Objetivos del proyecto

Descripción general del juego

- El juego esta pensado en realizar partidas rápidas multijugador jugador contra jugador.
- Las partidas tienen una duración máxima de 3 minutos.
- El objetivo del juego es acabar con las torres del contrincante antes de que el contrincante destruya tus torres.
- Cada jugador tiene asignado un color a su equipo, rojo o azul, disponen de 3 torres aliadas y al largo del tiempo aparecen esbirros aliados que atacan a las torres enemigas.
- Cada torre y esbirro dispone de vida que se disminuye al recibir impactos de proyectiles o ataques de esbirros.
- Cada jugador dispone de una barra de vida, y una barra de energía.
- La barra de vida del jugador disminuye cuando este recibe un impacto, en caso de disminuir su barra de vida por completo, el jugador permanece inmóvil durante 5 segundos y posteriormente es posicionado en su base inicial.
- La barra de energía disminuye cuando el jugador realiza un ataque o una habilidad especial.
- La barra de energía se autor recarga automáticamente en función del tiempo.
- Cada jugador puede disparar proyectil con un coste de energía bajo y puede depositar una bomba con un coste energía más elevado.
- Las torres aliadas disponen de un cañón encima que dispara proyectiles al jugador contrario o a sus esbirros aliados.
- Para forzar que se termine una partida, los esbirros aliados aumentan en función del tiempo, cuando queda poco tiempo se invocan muchos esbirros forzando el fin de la partida.

5.3 Elementos utilizados en red

En el siguiente listado se componentes los elementos del juego utilizados en Red:

Elementos Utilizados en Red	
Elemento	Componente
Jugador	NetworkIdentity NetworkTransform NetworkAnimator NetworkTransformChild
Esbirro	NetworkIdentity NetworkTransform NetworkAnimator
Torres	NetworkIdentity NetworkTransform NetworkAnimator
Proyectiles	NetworkIdentity NetworkTransform
Bomba	NetworkIdentity NetworkTransform

5.4 Componentes en elementos principales

5.4.1 Jugador

Jugador
Transform
Mesh Renderer
Movement Player (Script)
SetupLocalPlayer (Script)
Network Identity
– Local Player Authority
Network Transform
– Sync Character Controller
Animator
Network Animator
Disparar (Script)
Jugador (Script)
Habilidades (Script)
Audio Source
– Sonido disparo
CharacterController
Elementos Hijos al Jugador
-Modelo
– Disparador
Canvas
– Controles
EventSystem
Efecto Sangre
– Particle System

5.4.2 Esbirros

Esbirros
Transform Mesh Renderer NavMeshAgent – AutoRepath Marcado NetworkIdentity NetworkTransform - Sync Transform Box Collider Enemigo Melee (Script) Rigidybody -Freeze Position Y, -Freeze Rotation X – Z Capsule Collider - IsTrigger AudioSource **Elementos Hijos al Esbirro** Modelo -Animator -NetworkAnimator Marca Efecto Sangre Disparador

5.4.3 Torres

Torres
Box Collider Rigidbody Torreta (Script) NetworkIdentity – LocalPlayerAutority AudioSource -HitTower **Hijos:** Elementos Visuales Cañón MeshRenderer EnemigoRango(Script) NetworkIdentity - LocalPlayerAutority Disparador

5.5 Descripción de funcionalidades

5.5.1 Movimiento del jugador

Para el movimiento del jugador se ha utilizado un joystick táctil en la parte inferior izquierda de la pantalla táctil del dispositivo móvil.
El movimiento del jugador se produce al mover el joystick dirigiéndolo con el dedo.

Para el desarrollo del movimiento, se ha programado mediante un script que captura los eventos producidos en el joystick y en función a los valores devueltos, se realiza el movimiento del jugador en una dirección u otra.

Para simplificar y mejorar la experiencia del juego, se ha utilizado una librería de Unity que nos incorpora un joystick o varios joystick, que podremos configurar al gusto, adecuándonos a nuestras necesidades.

La librería usada se llama CInput, para utilizarla, simplemente debemos importar el paquete a nuestro proyecto, y posteriormente asignar el Asset de joystick al jugador.
Para poder trabajar con ella mediante script debemos importar su clase, y utilizar sus métodos para capturar los inputs.

5.5.2 Movimiento en red

Dado que el movimiento en el jugador es una acción que debe producir cada cliente y solo poder ejecutarlo el, debemos adaptar nuestro script para que se comporte como tal en red, y activarlo solo en cada jugador.

Es decir si el jugador Azul es nuestro jugador local, este jugador para nosotros localmente debe tener activado o asignado el script de movimiento, los otros clientes que verán nuestro jugador como una instancia de objeto, no deben tener asignado el script del movimiento.

Esto es debido a que utilizamos métodos de entrada que serían capturados por todos los jugadores, es decir, si un jugador mueve el joystick para mover el personaje, cada personaje (el nuestro y todos los demás) recibirían dicho evento y se moverían todos a la vez causando un comportamiento erróneo.

5.5.3 Activar y desactivar componentes en red

En un juego multijugador autoritario, es muy importante tener claro que componentes y scripts serán locales y cuales no.

Esto se traduce en saber que comportamientos queremos que se produzcan en todas las instancias de juego, y cuales solo localmente.

Para nuestro jugador, es muy importante crear un script que se encargue de gestionar estos componentes, activándolos o desactivándolos según nuestro jugador sea local (es nuestro jugador) o no lo es.

Para ello se ha creado un script llamado SetupLocalPlayer, en este script en primer lugar lo que se hace es comprobar si nuestro jugador es local o no.
Para ello usamos la variable isLocalPlayer, esta variable nos indica si es nuestro jugador o no en resumidas cuentas.

En función de ello activamos componentes como, los scripts de entrada, MovimientoJugador, Animator, Marcar jugador….

5.5.4 Disparar proyectiles

Para poder disparar proyectiles desde nuestro jugador, se ha utilizado un joystick situado en la parte derecha inferior de la pantalla táctil de un dispositivo móvil.

Este joystick al pulsar sobre y el dirigirlo sobre una dirección, se encarga de controlar la rotación del jugador (hacia donde está mirando) y al mismo tiempo empieza a disparar proyectiles en dicha dirección, siempre y cuando disponga de energía suficiente para ello.

Comportamiento de función de disparar

Para disparar cada jugador dispone de un script que captura los eventos del joystick de disparar, y se encarga de invocar una función de servidor que se encarga de instanciar el proyectil en dicha dirección.

La función de disparar debe tener la etiqueta Command y empezar con el prefijo Cmd_ como hemos indicado anteriormente.

Esta función es importante declararla así, ya que solo el servidor tiene autoridad para poder realizar instancias de otros objetos en red.
Nuestro jugador tiene autoridad para poder realizar funciones de comando de servidor, pero realmente el que realiza el trabajo y el proceso de calculo y la instancia del objeto es el servidor.

5.5.5 Proyectiles

Cada proyectil dispone de un NetworkIdentity para que se comporte como un objeto en red.

Los proyectiles disponen de un script de estadísticas, que incluyen su daño base.
También incluyen un tag, etiquetando cada proyectil, esto es para saber quien ha disparado dicho proyectil, y en función de ello realizar daño o no.

Los proyectiles tienen un network transform para sincronizar su posición y rotación en todas las instancias.

Pero tienen un time rate muy bajo, solo se sincroniza la primera vez, es decir su posición inicial, una vez posicionados mediante script se le asigna una fuerza en su dirección base.

Por que no se sincroniza por completo un proyectil en red?

Al sincronizar constantemente un proyectil en red, debe a la latencia de la red en general, el comportamiento del proyectil en el lado de los clientes es bastante pésimo, esto es debido a que el proyectil se desplaza a una velocidad considerable, y debido al retardo de la red, en algunos casos se puede ver como el proyectil se ve a parones o lagueado.

Para solucionar este problema, mi solución consiste en instanciar su posición y posteriormente asignar una fuerza localmente en cada instancia, de esta manera en todos los clientes el efecto se ve bien, y el comportamiento es optimo.

5.5.6 LobbyManager

En nuestro lobbyManager debemos especificar varios valores, entre ellos el numero máximo de jugadores, el jugador a instanciar (se le pasa como prefab) y la escena de juego en red.

También debemos especificar, y esto es muy importante, todos los componentes que se podrán instanciar en Red, todos ellos deberán tener un NetworkIdentity como he mencionado anteriormente.

El lobby Manager se compone de una escena construida mediante la nueva GUI de Unity 5, en la que se disponen de 2 botones, uno para crear partida online (especificando un nombre de partida) y otro para listar las partidas.

Al crear o unirse a una partida, antes de empezar el juego, se puede definir el nombre de nuestro jugador, así como se nos muestra el color de nuestro equipo ya sea Rojo o Azul.
El juego comenzara cargando la escena de juego, en el momento en que ambos jugadores estén listos, para ello, deben pulsar ambos jugadores en el botón de Listo.

Al estar los dos jugadores listos automáticamente se cargara la escena de juego, y esta escena permanecerá a la espera hasta que ambos jugadores tengan cargada la escena por completo y estén listos para iniciar el juego, para ello se muestra un panel informativo, con el nombre y color de cada jugador, mostrando que están esperando a que los jugadores estén listos.

5.5.7 NetworkLobby – MatchMaking

Para poder crear salas de juego, listar las salas, e unirse a una sala, Unity nos proporciona un Lobby Manager con matchmaking incluido para poder configurarlo y establecer y gestionar las conexiones.

En el proyecto, se ha utilizado un Lobby Manager, en el cual se permite, crear partidas, unirse a una partida, todo esto por Internet o para testing crear y unirse a una partida de juego local (Área Local).

Para poder crear partidas por Internet debemos reunir una serie de requisitos.
Inicialmente necesitamos tener una cuenta de Unity activada, y posteriormente vincular nuestro proyecto a un proyecto de Unity.

Una vez vinculado el proyecto, debemos activar la pestaña de Servicios en Red. Esta acción se puede realizar mediante el propio Unity 3D a partir de la versión 5.4, o mediante la web de unity.

Una vez activado el proyecto y lo servicios multijugador, establecemos las configuraciones oportunas. En este caso, el máximo de jugadores en una partida (2).

En la versión personal de Unity se nos proporciona gratuitamente 20CCU, esto es traduce en conexiones concurrentes a nuestro proyecto.
Se pueden establecer mas conexiones y valores, pero para ello debemos contratar servicios Premium de Unity con un coste en base al tráfico efectuado.

5.5.8 Torres Aliadas

Las torres aliadas, son nuestro elemento a defender, o en el lado contrario a atacar, las torres disponen de una barra de vida, que se disminuye al recibir impactos de otro jugador o de sus esbirros.

Las torres disponen a modo defensivo de un cañón, que se encarga de disparar proyectiles al jugador contrario o a sus esbirros más cercanos.

5.5.9 Cañón Torre

El cañón torre se encarga de buscar enemigos en un rango de distancia, cada cierto tiempo actualiza su objetivo, y apunta y dispara proyectiles a dicho objetivo.

Para poder encontrar objetivos, mediante script, se controla la distancia entre los enemigos cercanos, y en caso de encontrar un enemigo dentro de su rango de visión se encarga de dispararle, en caso de perder rango de visión y no tiene objetivos, permanece a la espera hasta encontrar un nuevo objetivo.

5.5.10 Esbirros

Los esbirros son zombies aliados que se instancian automáticamente cada cierto tiempo, y se encargan de atacar a las torres enemigas a corta distancia.

Los esbirros tienen asignado un daño base y una vida.

La vida disminuye cuando el esbirro recibe un impacto de un proyectil enemigo, ignoran los proyectiles de nuestro jugador.

Los esbirros disponen de una marca situada a la altura de sus pies, con forma circular, mostrando el color de nuestro equipo o el equipo contrario, de esta manera podemos visualizar fácilmente los esbirros aliados o esbirros enemigos.

5.5.11 Instanciador de esbirros

Para poder generar los esbirros, es necesario tener un objeto que contenga un script que se encarga de instanciarlos en una posición cada cierto tiempo.
Para ello se han asignado dos gameobjects, uno por cada equipo, que se encargan de instanciar esbirros cada cierto tiempo.

Al instanciar un esbirro, a este se le asigna un objetivo, al cual estos deben encontrar y en tener una distancia considerable pararse y atacarlo.
Los objetivos pueden ser el jugador contrario, o sus torres.

El instanciador de enemigos es un objeto únicamente de servidor, es decir, los clientes no deben tener este objeto, para ello asignamos en su NetworkID la casilla onlyServer, de esta manera nos aseguramos de que el objeto únicamente tendrá instancia en el servidor.

5.5.12 Espera de jugadores listos

Dado el retardo que sufren los clientes con respecto al servidor al cargar una escena, obviamente el servidor siempre cargara la escena más rápidamente que los clientes, por ello es necesario establecer un sincronismo entre ambos jugadores para verificar que ambos estén listos, tengan todos los componentes cargados y la escena de juego lista para empezar la partida.

Para ello se realiza un doble check de variables de estado entre ambos jugadores, cuando un jugador esta listo invoca una función que actualiza un contador de variable de checkeo de listo.

Y los jugadores quedan esperando a que este valor sea igual a 2 (2 jugadores), en caso de que no sea igual a 2, siguen esperando.

Esta función se realiza por red mediante un comando que actualice la variable de estado, y se comprueba su estado localmente en cada instancia de jugador.

5.5.13 Manager Script

Para poder gestionar ciertas estadísticas y valores del juego, es muy recomendable tener una clase o script que contenga todos los valores que requiramos.

Para ello usamos un script llamado Manager, en el guardamos valores como, nuestro jugador local (como Gameobject) jugador contrario, así como las bases aliadas y torres enemigas, contador de torres restantes…

Es una buena manera de obtener rápidamente una referencia a un jugador u otro en cuando nos sea necesario.

6. INTERFAZ DE JUEGO

La Interfaz de usuario se compone de varias vistas, en esta sección se describen las vistas utilizadas en el proyecto, mostrando los elementos más característicos de la interfaz.

6.1 Elementos de la interfaz

6.1.1 Menú Principal

Ilustración 15 Figura Menú Principal

Esta es la pantalla principal, entre ella se puede apreciar un titulo del juego en la parte superior izquierda.

Un botón de volver, que se encarga de regresar a la vista anterior.

Una sección para jugar Online, que se compone de dos botones, en la parte izquierda un botón para crear partida junto con un campo de texto para indicar el nombre de la partida.

En la parte derecha un botón para listar las partidas disponibles.

En la parte inferior hay una sección para jugar localmente, esta opción esta oculta para vistas móviles, y es usada solo para Testing y en desarrollo.

En ella hay dos botones una para crear partida y unirse a ella, y otro para unirse a la partida.

En juego local, las partidas se crean en la dirección de localhost en el puerto 8000, esto se puede cambiar en los ajustes del componente NetworkManager mediante el Editor de Unity.

6.1.2 Listar partidas

Ilustración 16 Figura Menu buscar partidas - no encontradas

Listar Partidas, no se han encontrado partidas.

En esta sección nos muestra las partidas disponibles, en caso de haberlas, junto con un botón para unirse a ellas.

El encabezado de esta vista sigue un patrón idéntico a la vista principal, mostrando un titulo del juego junto con un botón para volver a la a vista anterior.

6.1.3 Vista de espera de jugadores

En la siguiente vista se pueden ver los jugadores conectados, mostrando el nombre del jugador, el color del equipo, junto con un botón para verificar que el jugador esta listo.

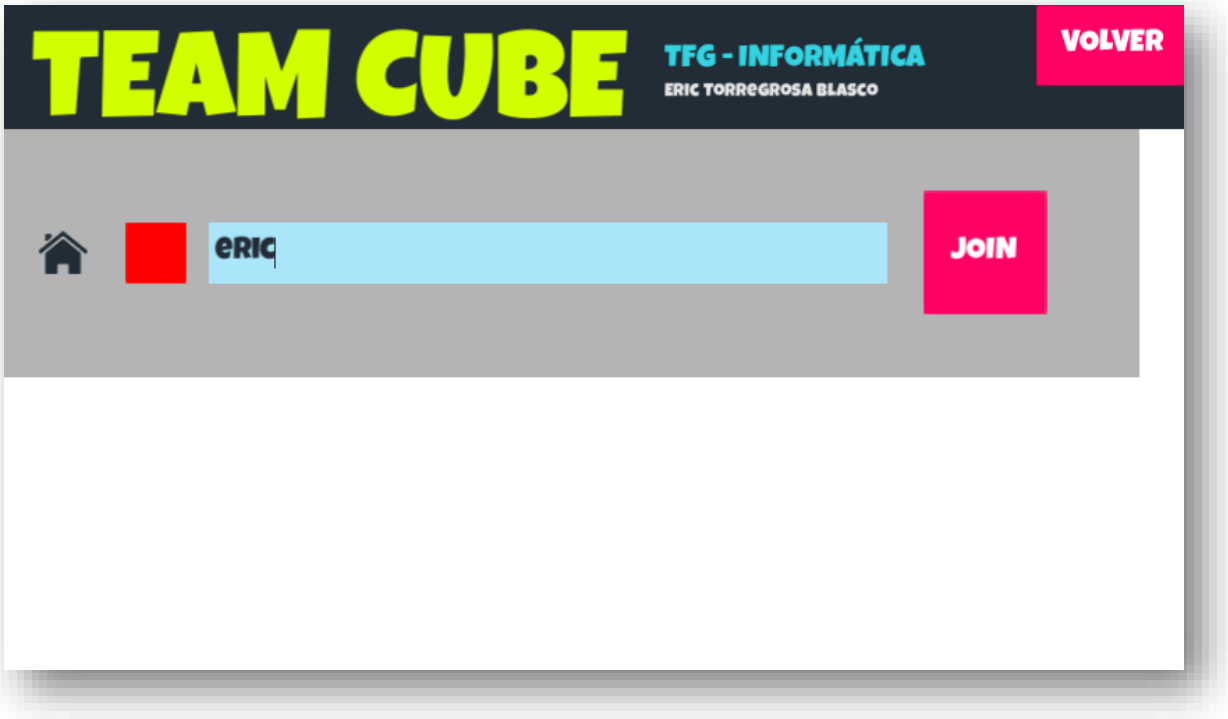


Ilustración 17 Figura Vista de espera de jugadores 1

Panel de juego esperando jugadores listos

En la vista de juego, ya cargada la escena de juego, se muestra una vista junto con el nombre de los jugadores, junto con su color de juego, indicando que se esta esperando a que los jugadores estén listos para comenzar la partida.

6.1.4 Panel de espera de jugadores listos

En la vista de juego, ya cargada la escena de juego, se muestra una vista junto con el nombre de los jugadores, junto con su color de juego, indicando que se esta esperando a que los jugadores estén listos para comenzar la partida.

Ilustración 18 Vista esperando oponente 1

6.1.5 Panel Superior de estadísticas

La interfaz de juego se compone de una interfaz simple situando un panel superior en la parte superior de la pantalla ,en la que se muestran las estadísticas de ambos jugadores.

En la vista de juego, en la parte superior de la pantalla, se muestra un panel con varios elementos indicando el estado de los jugadores, así como el estado de sus torres.

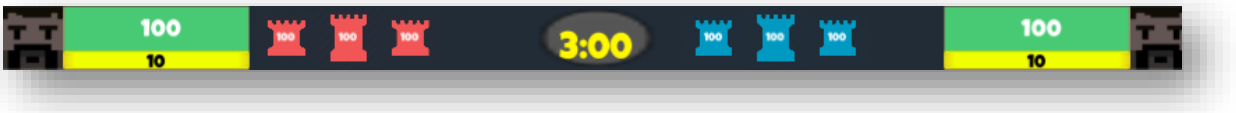


Ilustración 19 Figura vista panel superior de juego

Entre ellas podemos ver:

-Panel con color de equipo.
-Avatar del jugador.
-Nombre del jugador.
-Barra de vida del jugador.
-Barra de energía del jugador.
-Torres del jugador, mostrando su porcentaje de vida.

En la parte izquierda superior se muestran los elementos del jugador local, y en la parte superior derecha los elementos del jugador contrario.

6.1.6 Panel partida ganada

En este panel se muestra el nombre del jugador que ha ganado, este panel solo se muestra la jugador ganador.

Para mostrar el panel se requiere que se halla acabado el tiempo y tengas mas torres que el rival, o destruyas todas las torres del contrincante antes que él.

Ilustración 20 Figura panel ganador 1

6.1.7 Panel partida perdida

En este panel se muestra el nombre del jugador que ha perdido, este panel solo se muestra al jugador que ha perdido la partida.

Para mostrar el panel se requiere que se halla acabado el tiempo y tengas menos torres que el rival, o que tu rival destruya todas las torres del jugador antes que tu a él.

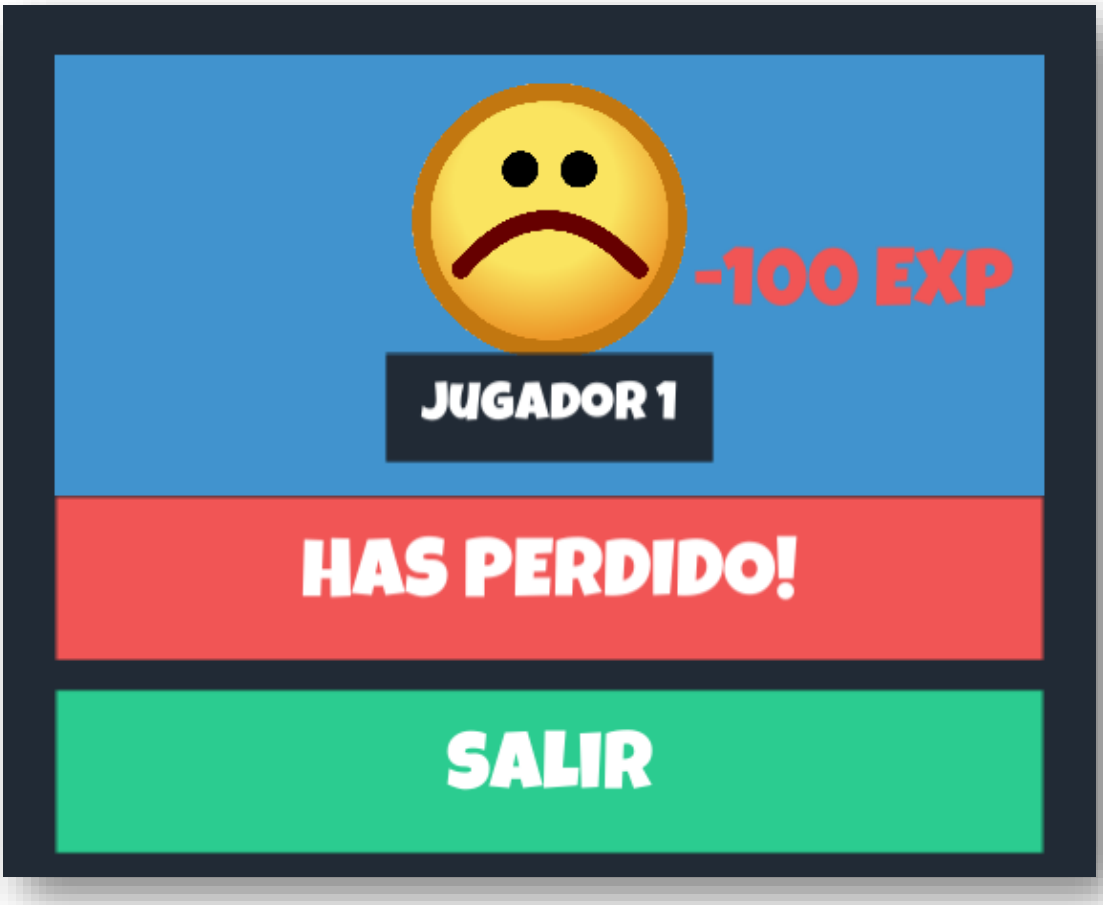


Ilustración 21 Figura panel perdedor

6.2 Actualización de la GUI

Para actualizar los elementos de la GUI, se dispone de un script que se encarga de obtener las referencias a las variables de estado de los diferentes elementos como el jugador, y las torres.

Para la sincronización de la GUI, se deben comprobar las variables de estado, para ello se hace uso de una Corrutina que cada cierto tiempo (1 segundo) comprueba los estados y los asigna a los elementos de la GUI.

Elementos a mostrar en la interfaz de usuario:

-Barra de vida del jugador.
-Barra de energia.
-Vidas de las torretas.
-Tiempo restante de la partida.

7. MODELOS 3D UTILIZADOS

7.1 Personaje Principal

El personaje principal se compone de un modelo 3D de estilo cúbico minimalista, al estilo Minecraft.
Se ha elegido un modelo simple de bajos polígonos.

El modelo incorpora una escopeta, mediante la cual es donde se instancian los proyectiles efectuados por el jugador.

Ilustración 22 Figura Imagen Personaje Principal

7.2 Esbirro 1

Para los esbirros del jugador uno, se han elegido modelos 3D de estilo minimalista, con una apariencia de zombie.

Ilustración 23 Figura imagen Esbirro 1

7.3 Esbirro 2

Para los esbirros del jugador dos, se han elegido modelos 3D de estilo minimalista, con una apariencia de zombie.

Ilustración 24 Figura imágen esbirro 2

7.4 Habilidad 1 TNT

El jugador es capaz de instanciar una bomba bomba, al pulsar sobre el botón de habilidad, esta bomba tiene un coste de 5 puntos de energía, y una vez instanciada en el juego, en 3 segundos estalla causando un daño de área de 50 puntos a todos los elementos que se encuentren en su rango de alcance.

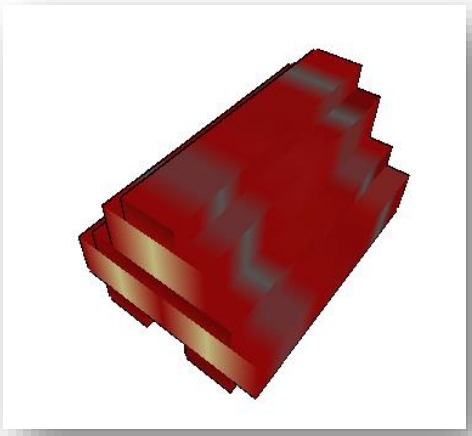


Ilustración 25 Figura Imágen icono Habilidad 1

7.5 Torres Aliadas

Las Torres se componentes de un elemento 3D con forma de torre, junto con un cañón en la parte superior.
El cañón es un elemento que va rotando cuando tiene un objetivo.

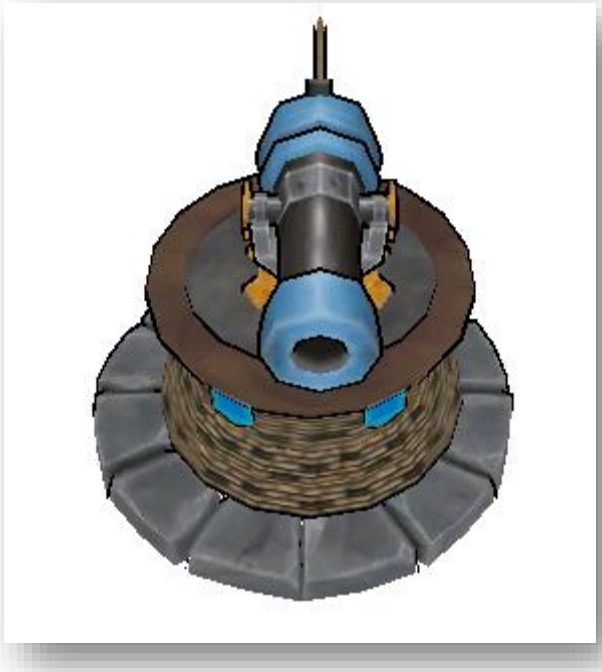


Ilustración 26 Figura Imágen Toreta 1

7.6 Mapa del juego

El mapa del juego esta diseñado en forma de isla flotante, rodeado por un mar.

El mapa se compone de dos zonas simétricas, cada una correspondiente a cada jugador, y estas zonas están unidas mediante dos puentes centrales.

En cada zona del mapa se encuentran ubicadas 3 torres aliadas del color del equipo del jugador.

El mapa incluye elementos estaticos visuales como piedras, camino de piedra, cristales, hierbas y barandillas.

Vista del mapa perspectiva del jugador.

Ilustración 27 Figura vista del mapa 1

Vista del mapa perspectiva aérea

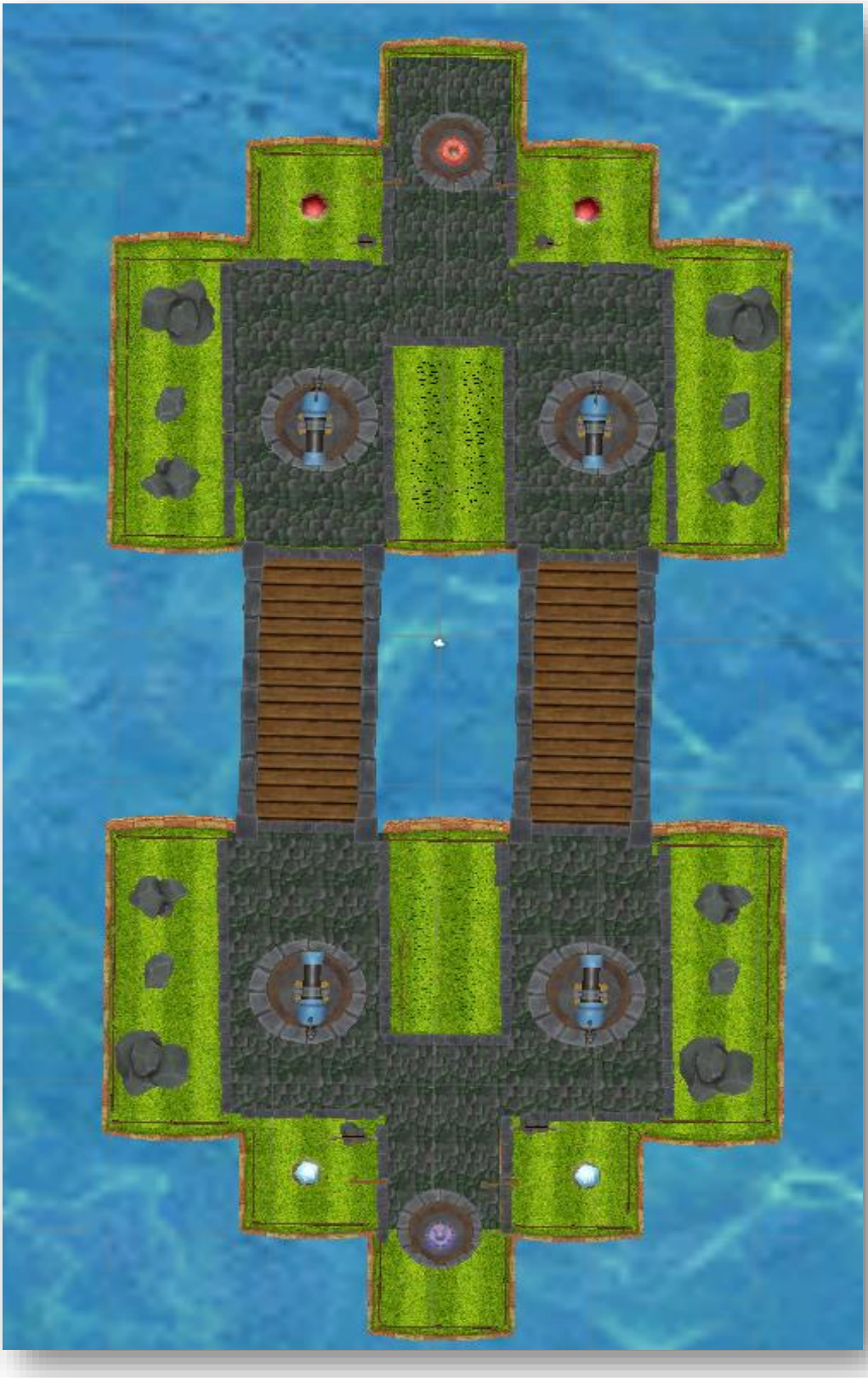


Ilustración 28 Figura vista del mapa 2

Mapa de calor del juego

En este mapa se muestran las áreas por las que pueden desplazarse los esbirros.

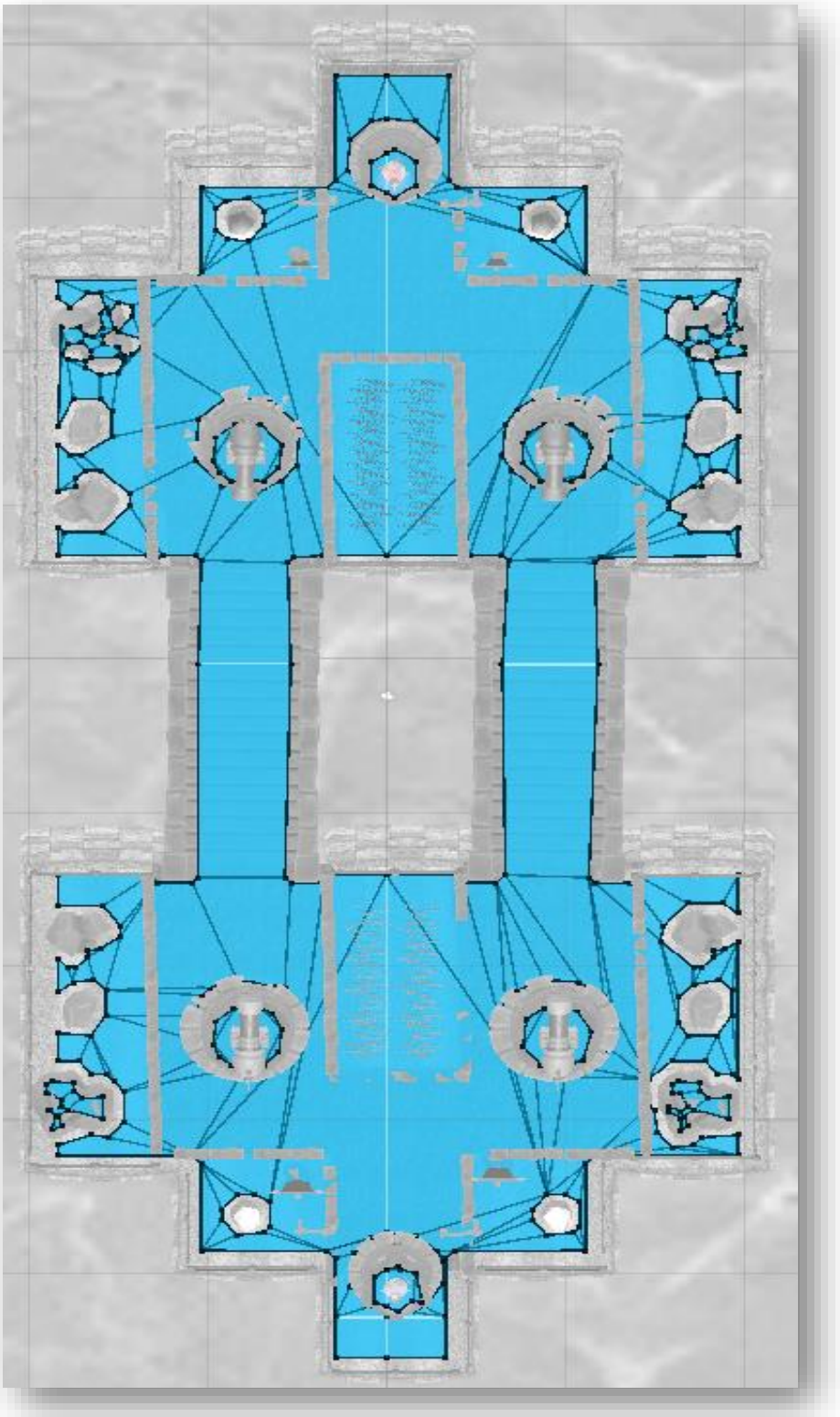


Ilustración 29 Figura vista mapa pathfinding

8. CONTROLES DEL JUEGO

Para controlar nuestro personaje, se han definido dos joystick en la parte inferior de la pantalla, situando cada joystick a cada extremo.

En el joystick izquierdo, podemos controlar la posición del personaje, en el caso de que no se este usando el joystick derecho, la rotación del personaje coincidirá con la dirección a la que se esta moviendo el personaje.

En el joystick derecho, podemos controlar la rotación del personaje, y al mismo tiempo dispara proyectiles en dicha dirección.

Captura de controles de movimiento.

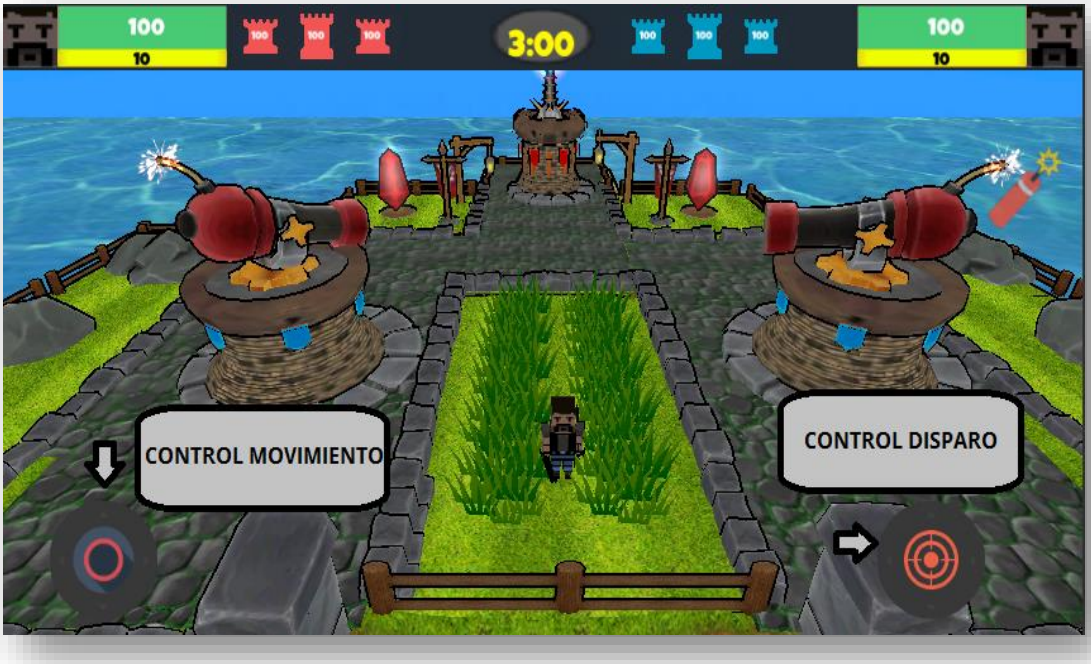


Ilustración 30 Figura imágen controles

9. RESULTADOS

9.1 Características finales de la aplicación

Interfaz agradable y simple.

Crear y unirse a partidas fácilmente.

Libertad en el movimiento del jugador.

Controles simples.

Modo de juego libre, puedes elegir tu propia estrategia de juego, si prefieres jugar a la defensiva, defendiendo tus torres o eres un jugador activo y prefieres la acción y jugar de atacante.

Varios Esbirros con inteligencia artificial propia.

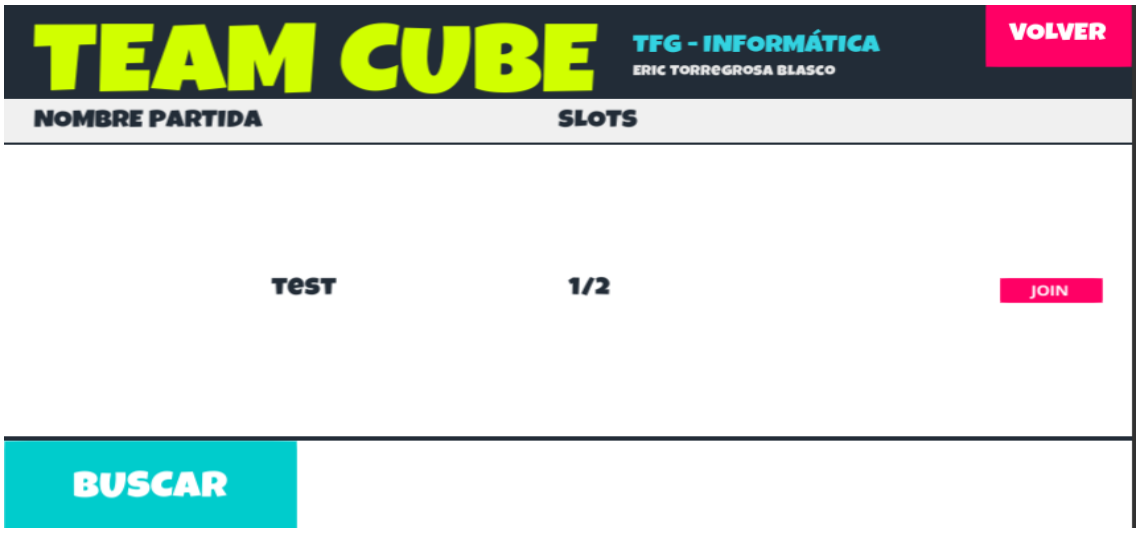
9.2 Descripción de la mecánica y objetivos de la aplicación

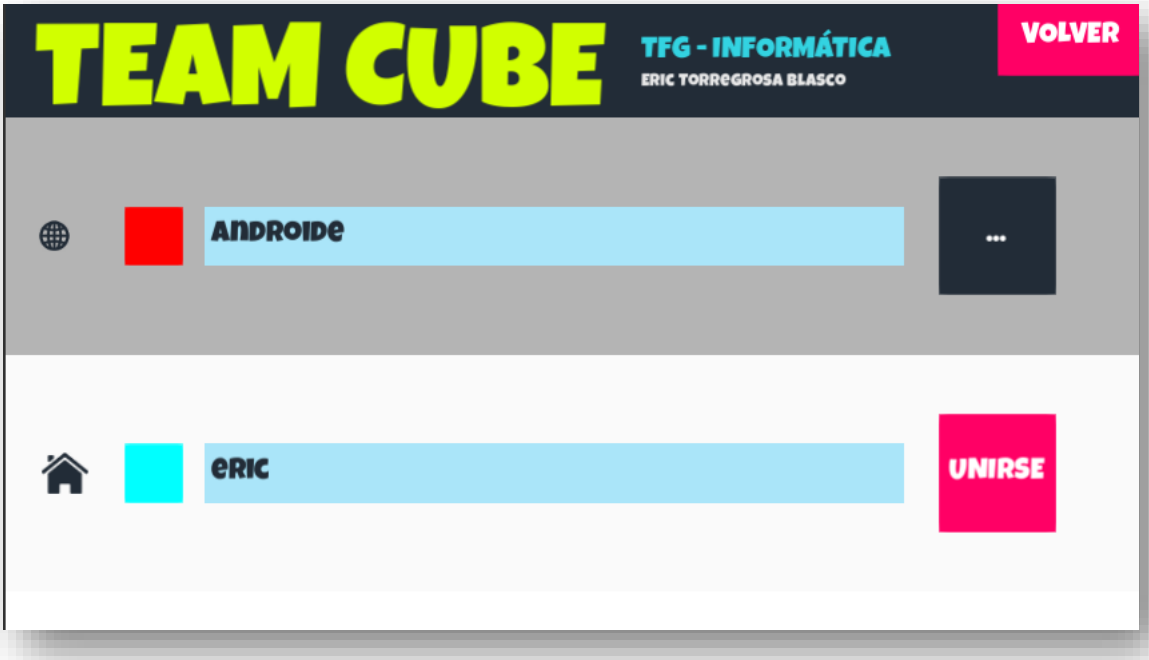
La mecánica del juego es muy simple para el propio usuario, la interfaz de usuario dispone de botones grandes visuales en los que se describe claramente que funcionalidad se realiza.

El objetivo de la aplicación es poder establecer partidas entre dos jugadores por Internet, sin que estos jugadores tengan que preocuparse de la funcionalidad y como se ha programado la aplicación en si.

La aplicación ofrece poder crear y listar partidas y jugar en ellas, creando retos entre los jugadores.

9.3 Capturas de pantalla






TEAM CUBE
ANDROIDE
VS
ERIC
ESPERANDO OPONENTE...


300
10
0:18
300
10
ERIC
ANDROIDE

9.4 Mercado al que va destinado

El juego va destinado para plataformas móviles, inicialmente está pensado lanzarse para la plataforma Android en la tienda de aplicaciones de Play Store de Google.

Dependiendo del éxito de la aplicación se distribuirá hacia más plataformas como APP Store en IOS o Windows Phone.

El juego aún no se va a publicar, dado que hay muchas características y mejoras que quiero realizar, pero en tener una versión mejorada con algunas de las mejoras citadas posteriormente, se publicará la aplicación.

Para monetizar la aplicación se utilizarán Ads de publicidad, concretamente Unity Ads 2.0.

Ilustración 31 Figura máscota Android

Ilustración 32 Figura icono Google Play

10. TRABAJO FUTURO

10.1 Posibles mejoras y actualizaciones

En el futuro se desea ampliar el proyecto en gran medida añadiendo futuras actualizaciones entre las cuales:

- Mayor numero de mapas.
- Nuevos modos de juego.
- Modo cooperativo
- Modo un jugador
- Nuevas armas.
- Más habilidades
- Implementación de los servicios de Google Play Games.
- Añadir amigos.
- Personalización del personaje.
- Sistema de micro pagos para adquirir nuevas skins de personaje.
- Exportar producto a otras plataformas móviles como IOS o Windows Phone.
- Eventos en función del tiempo.
- Sistema de logros y puntuación global.

10.2 Lanzamiento del juego

El juego está previsto lanzarse a finales del año 2016, se distribuirá a través de la tienda de aplicaciones de Google Play Store, y está disponible para todos todos los dispositivos con versión de Android superior a 2.3.

Estará disponible para todos los países.

11. CONCLUSIONES FINALES

El desarrollo de un videojuego multijugador es una tarea compleja en comparación al desarrollo de un videojuego tradicional de un solo jugador.

Para el desarrollo del proyecto, se ha requerido de buscar mucha documentación sobre el sistema de unity en términos de Networking, dado que es una tecnología nueva y que tiene poco tiempo de vida, la documentación en si es muy escasa, para poder realizar algunas funciones tienes que servirte de la propia documentación oficial que carece muchos ejemplos.

En si el desarrollo del proyecto me ha sido muy emocionante, he aprendido muchas cosas y sobre todo si se tiene pensado realizar un proyecto en red, debes pensar de manera muy distinta a como realizarías un proyecto sin funciones en red.

Para realizar alguna acción especifica que de manera tradicional no requeriría de mucho esfuerzo, el tratar de realizarlo en red es todo un reto.

UNET todavía está en fase de desarrollo y le quedan muchos aspectos por pulir, pero es una herramienta que está ahí y que nos ofrece un gran potencial en lo que se refiere al desarrollo en Networking.

Respecto al tema del MatchMaking, dependiendo de los días o las horas que lo utilices puede darnos problemas, en algunos casos las conexiones no se efectúan o las partidas no se registran, en otros casos simplemente la latencia que nos ofrece es horrible.

12. BIBLIOGRAFÍA

- https://docs.unity3d.com/Manual/UNet.html
- http://www.gamasutra.com/blogs/ChristianArellano/20150922/254218/UNET_Unity_5_Networking_Tutorial_Part_1_of_3__Introducing_the_HLAPI.php
- http://forum.unity3d.com/threads/unity-5-unet-multiplayer-tutorials-making-a-basic-survival-co-op.325692/
- http://opengameart.org/
- https://www.assetstore.unity3d.com/
- https://www.freesound.org/
- http://stackoverflow.com/
- https://www.reddit.com/
- https://www.youtube.com

13. ANEXOS ILUSTRACIONES