PROYECTO DE GRADO

Presentado ante la Ilustre UNIVERSIDAD DE LOS ANDES como requisito final para obtener el Título de INGENIERO DE SISTEMAS

DISEÑO E IMPLEMENTACIÓN DE LA CAPACIDAD MULTIJUGADOR EN LÍNEA PARA EL VIDEOJUEGO K.O. MANAGER.

Por:

Br. Jesús Francisco Rosales Izarra

Tutores:

Prof. Gérard Páez M.

Prof. Alejandro Mujica

Diseño e implementación de la capacidad multijugador en línea para el videojuego K.O. Manager

Br. Jesús Francisco Rosales Izarra

Proyecto de Grado — Sistemas Computacionales, 63 páginas

Escuela de Ingeniería de Sistemas, Universidad de Los Andes, 2017.

Resumen: Este Trabajo de Grado presenta el desarrollo e implementación de la capacidad multijugador en un videojuego, donde los usuarios que en este caso son llamados jugadores se conectan a través de sus dispositivos móviles a una partida simultánea a través de una red local o internet con el objetivo de generar un canal de comunicación para poder ejecutar la dinámica del juego, la cual consiste en conectar dos jugadores a un servidor y sincronizar características como son: audio, animaciones, posiciones, cantidad de vida, cantidad de estamina (indicador de energía) y mensajes en general. Este proyecto ha sido realizado mediante el uso del motor gráfico Unity junto al API de red UNET (Unity Networking)

Palabras claves: Videojuego, Multijugador, Dispositivo móvil, Unity Networking.

Índice general

Índice de figuras

Capítulo 1

Introducción

Se define como Videojuego a toda aplicación o software que ha sido creado con el fin del entretenimiento, siendo basado principalmente en la interacción de uno o más jugadores, ejecutados en cualquier dispositivo electrónico como lo son las consolas de videojuegos, computadoras, dispositivos móviles, entre otros.

En general un videojuego posee las siguientes características:

Software: el juego propiamente dicho que ha sido desarrollado para la diversión.

Dispositivo Electrónico: puede ser tanto un ordenador, máquinas arcade, consola de videojuegos o inclusive un dispositivo portátil (como ha sido en los últimos años, el avance de los juegos para teléfonos móviles).

Controlador: un periférico de entrada que nos permite realizar las distintas acciones dentro del juego. Este controlador varía de diseño y funciones dependiendo de qué dispositivo se está utilizando (generalmente son teclados o controles, aunque un dispositivo móvil también es usado como controlador a través de su interfaz gráfica).

Los videojuegos que encontramos hoy en día son variados, teniendo desde los más sencillos, hasta aquellos que inclusive tienen un argumento, verdaderos

guiones cinematográficos y son capaces inclusive de ir narrando historias que pueden continuar en futuras ediciones del mismo juego.

Entre los géneros de videojuegos más populares están los de acción, estrategia, rol, aventura, rompecabezas, simulación, deportes o carreras, cada uno de ellos con varios subgéneros. Por otro lado, hoy en día son habituales los videojuegos que toman elementos de más de un género, lo que ha dado lugar a géneros mixtos (por ejemplo rol-acción, aventura-acción, entre otros.).

Por otra parte, también se distingue a unos juegos de otros, incluso dentro de un mismo género, por la perspectiva visual que adoptan o dicho de otra manera, la posición de la cámara. Así, hay juegos con perspectiva 2D (ya sea con proyección paralela, vista lateral o vista cenital), 2.5D (mediante proyección isométrica, oblicua, o parallax scrolling, entre otras), y 3D (en perspectiva fija, en primera persona, o en tercera persona).

En muchos juegos se puede encontrar la opción de multijugador, es decir, que varias personas puedan participar simultáneamente en la misma partida, ya sea empleando todos la misma máquina (como suele ocurrir con las videoconsolas) o bien usando cada uno su propio dispositivo (el caso habitual en los PC o dispositivos portátiles, en lo que se conoce como videojuegos en línea). Existen juegos en que un jugador humano se enfrenta contra otros jugadores controlados por la máquina, mediante inteligencia artificial, pero en este caso no se considera que sea un videojuego multijugador. Por último, hay videojuegos que están pensados para congregar a un gran número de jugadores de todo el mundo conectados a través de Internet, son los conocidos como videojuegos MMO (de massively multiplayer online). Muchos juegos móviles admiten múltiples jugadores, de forma remota a través de una red o localmente a través de wifi, Bluetooth o tecnología similar.

Con el objetivo de entender al máximo el estado de las tecnologías, es extremadamente importante introducir al lector una breve historia de los juegos en los dispositivos móviles:

En la década de los noventa, empresas como Nokia o Philips decidieron integrar a los dispositivos móviles una funcionalidad más completa ofreciendo un tipo de entretenimiento a los usuarios. Este concepto se basaba en los beneficios más que notables que estaban generando las primeras arcade y consolas comercializadas en 1980. Estas compañías nunca pensaron que revolucionarían el sector de las videoconsolas portátiles, estos primitivos juegos fueron mejorando, desbloqueando niveles e incluso posteriormente con la conexión a Internet evolucionaron aún más. No existía la descarga como se realiza al día de hoy y los juegos venían integrados en el dispositivo programados a bajo nivel en lenguaje máquina y grabados en la memoria ROM.

El primer juego conocido en un teléfono móvil era una variante de Tetris en el dispositivo Hagenuk MT-2000 de 1994. En 1997, Nokia lanzó la serpiente muy exitosa (junto a sus variantes), que fue preinstalado en la mayoría de los dispositivos móviles fabricados, ya que se ha convertido en uno de los videojuegos más jugados y se encuentra en más de 350 millones de dispositivos en todo el mundo. Una variante del juego de la serpiente para el Nokia 6110, utilizando el puerto de infrarrojos, también fue el primer juego de dos jugadores para los teléfonos móviles. [3]

Hoy en día, los juegos móviles generalmente se descargan en las tiendas de aplicaciones, así como de los portales del operador de telefonía móvil, pero en algunos casos también se cargan previamente en los dispositivos de mano por el fabricante o por el operador de telefonía móvil cuando se adquiere, a través de conexión bluetooth, tarjeta de memoria, infrarrojos o desde la computadora con un cable conectado al dispositivo.

Los primeros juegos móviles descargables se comercializaron por primera vez en Japón en 1999 y por la década de 2000 estaban disponibles a través de una variedad de plataformas. Sin embargo, los juegos móviles distribuidos por los operadores móviles y portales de terceros (canales inicialmente desarrollados para rentabilizar los tonos de llamada descargables, fondos de pantalla y otras pequeñas piezas de contenido utilizando SMS premium o cargos del proveedor directos como mecanismo de facturación) mantuvieron una forma marginal de juego hasta que iOS App de Apple tienda que se puso en marcha en 2008, como la primera plataforma de contenido móvil operado directamente por el titular de la plataforma móvil, la App Store [10] ha cambiado de manera significativa el comportamiento de los consumidores y rápidamente amplió el mercado de los juegos móviles, ya que casi todos los propietarios de teléfonos inteligentes comenzaron a descargar aplicaciones móviles sobre esta plataforma. Poco después apareció Android Market (actual Play Store [9] de Google) la cual es una plataforma de distribución digital de aplicaciones móviles para los dispositivos con sistema operativo Android [4].

Actualmente pequeños grupos de desarrolladores enfrentan el mercado de los videojuegos mediante el emprendimiento, a esto le llamamos Startup.

Una startup es una empresa joven que está empezando a desarrollarse, está diseñada para crecer (o escalar) rápidamente. Las startups suelen ser pequeñas e inicialmente financiadas y operadas por un puñado de fundadores o un individuo [1].

Es importante recordar otros Trabajos de Grado previos a este, donde los videos juegos son la esencia principal, y han servido de guía en general para trabajos posteriores a estos, entre ellos tenemos:

Desarrollo de un entorno para la creación de los escenarios de juegos para niños por Boscan Luis (2015).

Diseño de entorno para la creación de juegos para niños incorporando avatares virtual por Parra Alonso (2015).

Creación de una Startup con un videojuego aplicando el método Lean Startup por Helderth Henríquez (2016).

1.1 El Problema

Actualmente la interacción entre dispositivos móviles juega un papel importante para la comunicación en los videos juegos, es por ello que surge el siguiente problema: ¿Cómo implementar la comunicación entre ellos para compartir información simultáneamente?

K.O. Manager es un videojuego desarrollado por una empresa llamada Kylum Games, está inspirado en el deporte de combate de las artes marciales mixtas (sus siglas en ingles MMA) y está diseñado para smartphones. Este videojuego consiste en 2 personajes que combaten entre ellos con el objetivo de encontrar un ganador. Actualmente solo posee la característica monousuario es por ello que surge la necesidad de crear un modo multijugador, es decir, permitir la conectividad entre dos dispositivos para jugar en una misma partida a través de la red.

Como Proyecto de Grado se diseñara e implementara la capacidad multijugador en el videojuego K.O. Manager.

Cabe destacar que con el constante aumento del uso de las tecnologías y del porcentaje de personas que hacen uso del internet, en Venezuela, por ejemplo 62 de cada 100 habitantes acceden al mismo según cifras de Conatel (Estimación primer trimestre año 2017) [5]. Por otro lado, el uso de Internet para actividades de entretenimiento, entre ellas el consumo de videojuegos en red, también se ha incrementado en los últimos años.

1.2 Objetivos

En esta sección se presenta de manera macro cuál es el propósito de este trabajo de grado, para dar introducción a los objetivos específicos que ayudarán al cumplimiento del proyecto.

1.2.1 Objetivo general

Diseñar e implementar la capacidad multijugador para el videojuego K.O Manager.

1.2.2 Objetivos específicos

- Aprender sobre la creación de videojuegos, específicamente usando el motor gráfico de Unity.
- Utilizar el API de Networking de Unity (UNET) para diseñar y crear la funcionalidad multijugador online.
- Sincronizar elementos en la dinámica del videojuego a través de la red.
- Realizar una versión inicial del videojuego con las funcionalidades básicas requeridas.
- Analizar las opiniones de los usuarios y clientes para retroalimentar el videojuego y así conseguir un producto viable durante el desarrollo.

1.3 Alcance

Se realizara el diseño e implementación de la capacidad multijugador en el videojuego K.O. Manager de la empresa Kylum Games, donde dos dispositivos

móviles se conectaran a través de la red (local o internet) e interactuaran en un servidor. Durante el tiempo de juego 2 jugadores accederán a través de sus smarthphones a una misma partida ejecutando su dinámica de juego a través de la red y sincronizando datos como lo son barras de vida, estamina, animaciones, sonidos y mensajes.

1.4 Metodología

Durante el desarrollo de este proyecto se usara un método de trabajo, que consiste en avances iterativos e incrementales, donde se desarrollaran las actividades propuestas en los objetivos [7].

El producto se desarrolla por incrementos en el que cada iteración obtiene una versión funcional del producto, de esta forma el sistema se desarrolla paso a paso y obtiene retroalimentación continua por parte del usuario.

Este método consta de tres etapas: inicialización, iteración y lista de control de proyecto [8].

- Inicialización: Se crea una versión del sistema. La meta de esta etapa es crear un producto con el que el usuario pueda interactuar y por ende retroalimentar el proceso.

- Iteración: Esta etapa involucra el rediseño e implementación de una tarea de la lista de control de proyecto y el análisis de la versión más reciente del sistema. La meta del diseño e implementación de cualquier iteración es hacerla simple, directa y modular, para poder soportar el rediseño de la etapa o como una tarea añadida a la lista de control de proyecto.

- Lista de control del proyecto: contiene un historial de todas las tareas que necesitan ser realizadas. Incluye información como nuevas funcionalidades para ser implementadas y áreas de rediseño de la solución ya existente. Esta lista de control se revisa periódica y constantemente como resultado de la fase de análisis.

El proceso de desarrollo consta de:

- Planificación: extraer los requisitos [6] de un producto software es la primera etapa para crearlo. Durante esta fase, el cliente plantea las necesidades que se presentan e intenta explicar lo que debería hacer el software o producto final para satisfacer dicha necesidad mientras que el desarrollador actúa como interrogador, es decir, como la persona que resuelve el problema. Se plantea la planificación como un permanente diálogo entre las partes empresarial y técnica (Deseable-Posible). Se planifica a nivel de negocio (Ámbito, prioridad, versiones, fechas) y técnico (Estimaciones de tiempo, consecuencias, procesos, detalles).

- Diseño: se generan pequeñas versiones (prototipos). La arquitectura de software consiste en el diseño de componentes de una aplicación (entidades del negocio), generalmente utilizando patrones de arquitectura. El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del negocio y además poder ser validado, por ejemplo por medio de diagramas de secuencia. Un diseño arquitectónico describe en general el cómo se construirá una aplicación de software, para ello se documenta utilizando diagramas, por ejemplo: diagramas de clases, diagramas de base de datos, diagramas de despliegue, diagramas de secuencia, entre otros.

- Codificación: en esta fase el objetivo principal es plasmar el diseño elaborado para la tarea, es decir, en el lenguaje de codificación que será implementado en el proyecto.

- Pruebas: consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación del problema. Una técnica es probar por separado cada módulo del software, y luego probarlo de manera integral, para así llegar al objetivo.

Capítulo 2

Estado del Arte

2.1 Tecnologías

2.1.1 El lenguaje de programación C# o C Sharp

Es un lenguaje orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, está diseñado para la infraestructura de lenguaje común la cual permite que aplicaciones escritas en distintos lenguajes alto nivel puedan ejecutarse en múltiples plataformas tanto de hardware como de software sin necesidad de escribir o recompilar nuevamente su código fuente o de recompilar [11].

2.1.2 El motor de videojuego Unity

Es un motor de videojuegos multiplataforma creado por Unity Technolgies disponible para sistemas operativos como Microsoft Windows, Linux y OS X. Principalmente es usado para el desarrollo de videojuegos y simulaciones (ambas para computadoras, consolas y dispositivos móviles) soportando el desarrollo grafico en dos y tres dimensiones. Los programadores pueden utilizar los lenguajes UnityScript que está inspirado en la sintaxis ECMAScript (basado en el popular lenguaje Javascript), C# o Boo (posee una sintaxis similar a Python pero se encuentra obsoleto desde Agosto 2017 con el lanzamiento de Unity 5). Este motor utiliza como APIs graficas OpenGL, Direct3D (Windows y Xbos One), OpenGL ES

en Android e iOS, WebGL en la web y APIs propietarias de consolas de videojuegos.

Unity ofrece servicios a los desarrolladores como lo son: Anuncios, Análisis, Certificación, Integración continua, Everyplay (graba y comparte tus gameplays), IAP (compras dentro de la aplicación), Multiplayer(conectividad entre dispositivos), Performance Reporting (recopila errores en la aplicación) y Collaborate (guardar , compartir y sincronizar proyectos).

Entre las plataformas actuales que soporta se encuentran Android, Facebook Gameroom, Fire OS. Gear VR, Goofle Cardboard, Goodl Daydream, HTC Vive, iOs, Linux, macOs, Microsoft HoloLens, Nintendo 3DS family, Nintendo Switch, Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOs, WebGL, Wii U, Windows, Windows Phone, Windows Store y Xbox One [12].

2.1.3 Conceptos básicos en Unity

2.1.3.1 El Inspector

El Inspector es usado para ver y editar propiedades de objetos de juego, también preferencias y otros ajustes dentro de Unity.

2.1.3.2 La Escena

Las escenas contienen los objetos del juego. Pueden ser usadas para crear un menú principal, niveles individuales y cualquier otra cosa. Se piensa en cada archivo de escena como un nivel único.

2.1.3.3 Componentes

Los Componentes son dependencias instanciadas de otras clases, que añaden ciertas funcionalidades a nuestro objeto.

2.1.3.4 Game Objetcs (Objetos de juego)

Son contenedores, que permiten guardar los diferentes componentes que son requeridos para hacer un personaje, o cualquier objeto.

2.1.3.5 Prefabs (Objetos prefabricados)

Son una colección de gameobjects y componentes predefinidos, re-utilizables a lo largo del juego.

2.1.3.6 Transform

El Transform es usado para almacenar y manipular la posición, rotación, escala. Cada transform puede tener un padre, que le permite aplicar la posición, la rotación y la escala jerárquicamente.

2.1.3.7 API

API viene del inglés "Application programming interface" que significa interfaz para programación de aplicaciones. Es la parte de una biblioteca a la que accede un programa que usa la biblioteca, haciendo así el uso de la biblioteca independiente de los detalles de implementación. Una API puede ser implementada por distintas bibliotecas.

2.1.3.8 Assets

Un asset es una representación de cualquier ítem que puede ser utilizado en un proyecto. Un asset puede venir de un archivo creado fuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos que Unity soporta.

2.1.3.9 Serialización

Consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato más legible como XML, JSON, entre otros.

2.1.4 Unity Multijugador (Networking)

Existen dos tipos de usuarios para la característica networking:

- Los que realizan un juego multijugador y empiezan con el Administrador de red (NetworkManager) o el API de Alto Nivel, que es el que se utilizará en este trabajo de grado[13].
- Los que construyen una infraestructura avanzada de juegos multijugador y usan el API de Network Transport que es de bajo nivel, siendo usada para juegos con requerimientos específicos que el API de Alto Nivel no provee.

2.1.4.1 API de bajo nivel (Network Transport o Transport Layer)

En adición al API de alto nivel que Unity provee, se da acceso a un API de bajo nivel llamada Transport Layer, la cual nos da la habilidad de crear nuestro

propio sistema de red a bajo nivel, el cual puede ser útil si se quiere tener requerimientos específicos más avanzados en la red de un juego. La capa de transporte es una capa delgada que funciona sobre la red basada en sockets del sistema operativo. Es capaz de enviar y recibir mensajes representados como arrays de bytes, y ofrece una serie de opciones diferentes de calidad de servicio para adaptarse a diferentes escenarios. Se centra en la flexibilidad y el rendimiento, y expone una API dentro de la clase UnityEngine.Networking.NetworkTransport.
La capa de transporte admite servicios básicos para la comunicación en red. Estos servicios básicos incluyen:

- Establecer conexiones
- Comunicarse con una variedad de calidad de servicios
- Control de flujo
- Estadísticas base
- Servicios adicionales como comunicación a través de servidor de retransmisión o descubrimiento local. La capa de transporte puede utilizar dos protocolos: UDP para comunicaciones genéricas y WebSockets para WebGL.

2.1.4.2 API de Alto Nivel (High Level API)

El API de Alto Nivel (HLAPI) es un sistema para construir capacidades multijugador para juegos de Unity. Está construido encima de la capa del nivel menor del transporte de la comunicación en tiempo real, y maneja las tareas comunes que son requeridas para juegos multijugador. Mientras que la capa de transporte soporte cualquier tipo de topología de red, el HLAPI es un sistema servidor autoritario, aunque solo permite que uno de los participantes sean un

cliente y el servidor al mismo tiempo, por lo que no hay un proceso dedicado al servidor requerido. Trabajando en conjunto con los servicios de internet, esto permite que los juegos multijugador sean jugados a través de internet con el menor trabajo posible de los desarrolladores.

El HLAPI es un nuevo conjunto de comandos de red construidos en Unity, dentro de un nuevo namespace: UnityEngine.Networking**.** Está enfocado en su fácil uso y desarrollo iterativo y proporciona servicios útiles, tales como:

- Operar juegos alojados al cliente, donde el host es también un cliente jugador.
- Controlar el estado de red del juego utilizando un Administrador de Red.
- Manejadores de mensajes.
- Serializadores de alto rendimiento de propósito general.
- Una administración de objeto distribuida.
- Sincronización de estados.
- Clases de Network (red): Servidor, cliente, conexión, etc.
- Enviar comandos de red de clientes a servidores.
- Realizar llamados de procedimientos remotos de servidores a clientes (calls-RPCs).
- Enviar eventos de red de servidores a clientes.

En la figura 2.1 se pueden observar las capas que construyen al HLAPI y agregan funcionalidades.

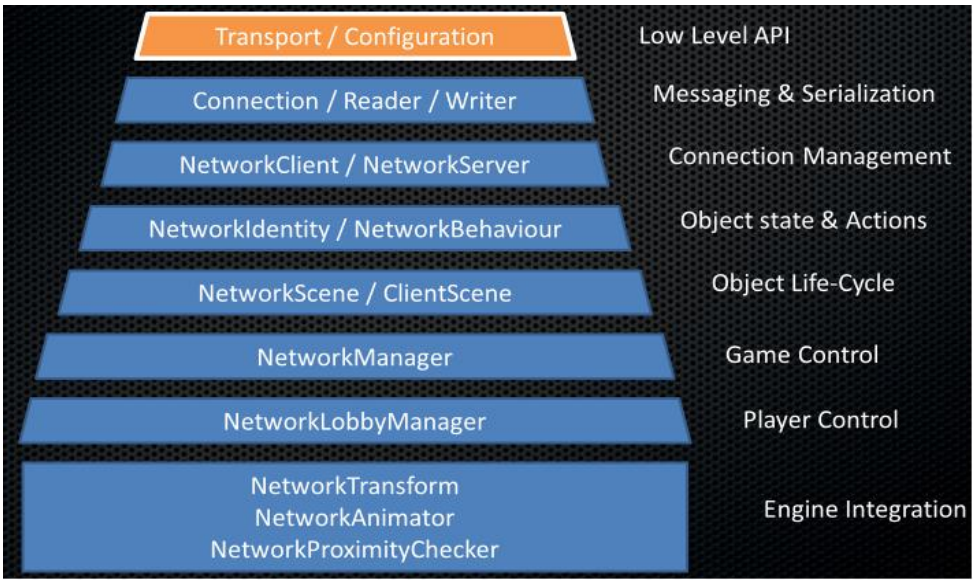

Figura 2. 1: Capas de construcción del HLAPI.

2.1.4.3 Servidor y Anfitrión

En el sistema de red de Unity, los juegos tienen un servidor y múltiples clientes donde no hay un servidor dedicado, uno de los clientes juega el rol del servidor y es llamado cliente anfitrión (Host).

El anfitrión es un servidor y un cliente en el mismo proceso, el cual utiliza un tipo especial de cliente llamado LocalClient (cliente local), mientras otros clientes son RemoteClientes (clientes remotos). El LocalClient (cliente local) se comunica al servidor (local) a través de llamadas de funciones 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.

Una meta del sistema de red es que el código para LocalClients y RemoteClientes sea el mismo, entonces los desarrolladores solo tienen que pensar acerca de un tipo de cliente la mayoría de veces.

La figura 2.2 representa al host (servidor y cliente en el mismo dispositivo) y los clientes conectados a el.

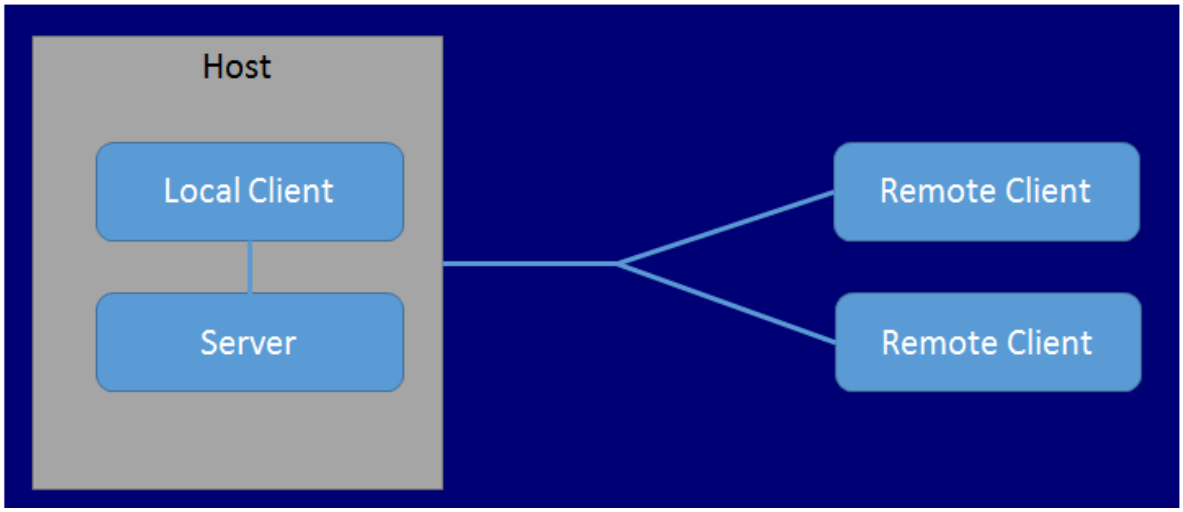

Figura 2. 2: Representación general del host junto a los clientes.

2.1.4.4 Instanciar (Instantiate) y Generar (Spawn)

En Unity, GameObject.Instantiate crea nuevos game objects, pero con el sistema de red, los objetos también deben ser generados (spawned) para estar activos en la red. Esto solamente se puede hacer en el servidor y hace que los objetos sean creados en clientes conectados. Una vez los objetos estén generados, el sistema de generación (spawning system) hace la administración del ciclo de vida del objeto distribuido y la sincronización de estados.

2.1.4.5 Jugadores, Jugadores Locales 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, 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 sobre el cliente.

Se puede observar en la figura2.3 encerrados en círculos a los objetos jugador asociados a cada persona jugando (clientes).

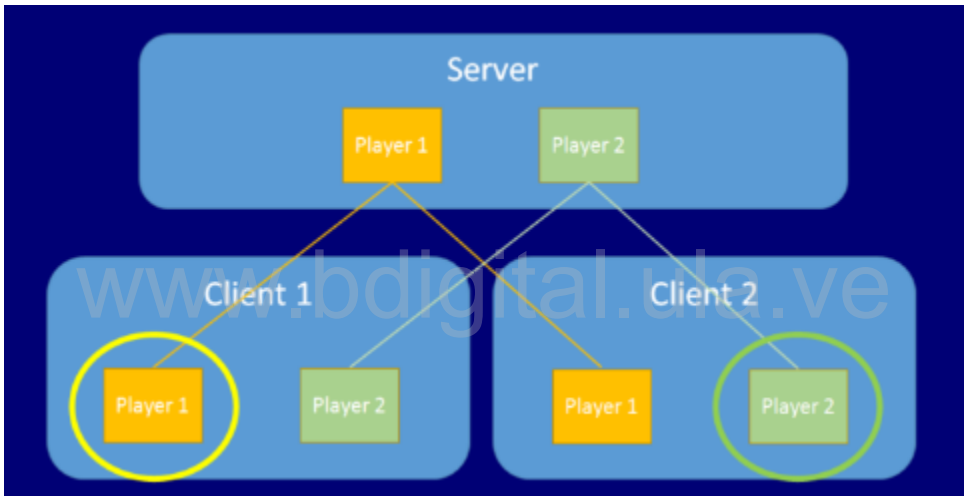

Figura 2. 3: Diagrama de dos clientes junto a sus jugadores locales.

Solamente el objeto jugador que es propio tendrá la bandera isLocalPlayer configurada. Esto puede ser utilizado para filtrar 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 comúnmente 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.

2.1.4.6 Propiedades del contexto de la red

Hay propiedades en la clase NetworkBehaviour que le permite al 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 anfritrión 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.

2.1.4.7 Administrador de Red (Network Manager)

Es un componente para manejar el estado de red de un juego multijugador. En realidad esta implementado completamente utilizando el HLAPI, entonces cualquier cosa que haga, está disponible para desarrolladores en otras formas. Sin embargo, el NetworkManager envuelve muchas funcionalidades útiles en un solo lugar y hace que crear, correr, y depurar juegos multijugador sean lo más simple posible.

El Administrador de Red puede ser utilizado completamente sin scripting. Tiene controles del inspector en el editor que permite que la configuración de todas sus características.

El NetworkManagerHUD proporciona una simple y predeterminada interfaz de usuario durante el tiempo de ejecución que permite que los juegos de red sean controlados por los usuarios. Para usos avanzados, los desarrolladores pueden derivar una clase del NetworkManager y personalizar su comportamiento al anular cualquiera de las funcionales virtuales que proporciona.

Las características incluyen:

- Manejo del estado de juego.
- Manejo de generaciones (Spawning).
- Manejo de escenas.
- Información de depuración.
- Emparejamiento de partidas.
- Personalización.

2.1.4.8 Manejo del Estado del Juego

Un juego de red multijugador puede correr en tres modos, como un cliente, como un servidor dedicado, o como un host (anfitrión) que es cliente y servidor a la misma vez. El networking (red) está diseñado para hacer que el mismo código del juego y los assets funcionen en todos estos casos. Desarrollar para la versión de un solo jugador y la versión multijugador debería ser la misma cosa.

El Network Manager tiene métodos para ingresar en cada uno de estos modos. NetworkManager.StartClient(), NetworkManager.StartServer() y NetworkManager.StartHost() están todos disponibles para código script, por lo que pueden ser invocados de los manejadores de entrada del teclado o de interfaces de usuario personalizadas. Los controles por defecto en tiempo de ejecución que pueden ser mostradas opcionalmente también invocan estas mismas funciones. También hay botones en el inspector NetworkManagerHUD

disponibles cuando se está en modo de reproducción que llama las mismas funciones. Cualquiera que sea el método utilizado para cambiar el estado del juego, las propiedades networkAddress y networkPort son utilizadas. Cuando un servidor o host (anfitrión) ha iniciado, el networkPort se vuelve el puerto de escucha. Cuando un cliente comienza, la networkAddress es la dirección para conectarse, y el networkPort es el puerto para conectarse.

2.1.4.9 Manejo de generaciones (Spawning)

El NetworkManager puede ser utilizado para manejar la generación de objetos en red (prefabs). La mayoría de juegos tienen un prefab utilizado como el objeto principal del jugador, entonces el NetworkManager tiene una ranura para arrastrar el prefab del jugador. Cuando un prefab del jugador ha sido configurado, un objeto jugador automáticamente será generado de ese prefab para cada usuario en el juego. Esto aplica al jugador local en el servidor (anfitrión) y a los jugadores remotos en clientes remotos. Hay que tomar en cuenta cuenta que el prefab del jugador debe tener un componente NetworkIdentity.

Adicionalmente al prefab del jugador, los prefabs de otros objetos que serán generados dinámicamente deben ser registrados con la ClientScene (Escena del cliente). Esto se puede realizar con las funciones ClientScene.RegisterPrefab(), o puede ser hecho por el NetworkManager automáticamente. Agregar prefabs a la lista de generaciones hará que sean auto-registradas.

Una vez el prefab de un jugador esté configurado, se debe poder comenzar el juego como anfitrión y ver el objeto del jugador generado. Parar el juego debería hacer que el objeto jugador sea destruido. Correr otra copia del juego y conectándose como un cliente a localhost debería hacer que otro objeto jugador

aparezca y al parar ese cliente debería hacer que el objeto jugador del cliente sea destruido.

El objeto jugador es generado por la implementación por defecto del NetworkManager.OnServerAddPlayer. Si se desea personalizar la manera de que los objetos jugadores son creados, se puede anular la función virtual.

2.1.4.10 Posiciones de Inicio (Start Positions)

Para controlar dónde los jugadores son generados (spawned), se puede utilizar el componente NetworkStartPosition. Este componente busca objetos NetworkStartPosition en la escena y si encuentra alguno entonces va a generar el jugador en la posición y orientación de uno de ellos. Un código personalizado puede acceder el NetworkStartPositions disponible por la lista NetworkManager.startPositions y también hay una función de ayuda GetStartPosition() en el NetworkManager (Administrador de red) que puede ser utilizado en la implementación de OnServerAddPlayer para encontrar una posición inicial.

Para utilizar posiciones iniciales, se adjunta un componente NetworkStartPosition a un objeto en la escena de juego. Puede haber varias posiciones dentro de una escena. El NetworkManager luego registrará la posición y orientación del objeto como una posición inicial. Cuando un cliente se une en el juego y un jugador es agregado, el objeto jugador será creado en una de las posiciones iniciales, con la misma posición y orientación.

El NetworkManager tiene una propiedad PlayerSpawnMethod que permite una configuración acerca de cómo las startPositions son escogidas.

- Random para generar jugadores en opciones startPosition aleatorias.
- Round Robin para recorrer las opciones startPosition en una lista.

2.1.4.11 Manejo de escena

La mayoría de los juegos tienen más de una escena. Al menos siempre hay una pantalla de título o una escena de menú al inicio adicionalmente a la escena dónde el juego en realidad es jugado. El NetworkManager está configurado para automáticamente manejar el estado de la escena y las transiciones de la escena en una manera que funcione para un juego multijugador. Hay dos ranuras en el inspector NetworkManager, la offlineScene (escena no en linea) y la onlineScene (escena en línea).

Cuando un servidor o anfitrión es iniciado, la escena en línea será cargada. Esto luego se volverá la escena de red actual. Cualquier cliente que se conecte al servidor será instruido para también cargar la escena. El nombre de esta escena es almacenado en la propiedad networkSceneName.

Cuando la red, el anfitrión o un cliente se detienen, la escena fuera de línea será cargada. Esto permite que el juego vuelva automáticamente a la escena del menú cuando se desconectada de un juego multijugador.

También se pueden cambiar escenas mientras el juego está activado al llamar NetworkManager.ServerChangeScene(). Esto hará que todos los clientes conectados actuales cambien de escena también, y va actualizar el networkSceneName (nombre de la escena en red) para que nuevos clientes también carguen la escena nueva.

Mientras el manejo de escena en red está activo, cualquier llamado a las funciones del manejo de estados tal como NetworkManager.StartHost() o NetworkManager.StopClient() puede causar cambios en la escena. Esto aplica al control UI en tiempo de ejecución. Por lo que al ajustar las escenas y llamar estas funciones, es fácil controlar el flujo de un juego multijugador.

Hay que tener en cuenta que los cambios de escena causan que los objetos en la escena sean destruidos. Esto puede incluir el NetworkManager. Si se quiere que el NetworkManager persista entre escenas, entonces hay que asegurar que la casilla de verificación Dont Destroy On Load sea configurada a true. En los casos más simples, esta es la mejor configuración. Pero, es posible tener un NetworkManager en cada escena con diferentes ajustes para controlar la carga incremental de prefab, o diferentes transiciones de escena.

2.1.4.12 Información De Depuración

El panel del inspector NetworkManagerHUD muestra información adicional acerca el estado de la red en tiempo de ejecución. Esto incluye:

- Conexiones en red.
- Objetos del servidor NetworkIdentity activos.
- Objetos cliente NetworkIdentity activos.
- Compañeros clientes.

También, los manejadores de mensajes de clientes registrados son mostrados en la ventana de pre-visualización.

2.1.4.13 Emparejamiento de partidas

El UI en tiempo de ejecución del NetworkManager y el inspector UI del NetworkManager permite por interacciones con el servicio de emparejamiento de partidas, y emerge la propiedad NetworkManager.matchmaker con un objeto NetworkMatch. Una vez esto esté activo, los UIs predeterminados lo utilizan y hacen callbacks a las funciones en el NetworkManager para permitirle realizar un emparejamiento de partidas simple.

Hay funciones virtuales en el NetworkManager que las clases derivadas pueden utilizar para personalizar comportamiento de callbacks de emparejamiento de partidas.

2.1.4.14 Personalización

Hay funciones virtuales en el NetworkManager que las clases derivadas pueden utilizar para personalizar comportamiento. Cuando se implementan estas funciones, hay que asegurar la funcionalidad que las implementaciones por defecto proporcionan. Por ejemplo, en OnServerAddPlayer(), la función NetworkServer.AddPlayer debe ser llamada para activar el objeto jugador para esa conexión.

2.1.4.15 Sincronización de Estados

La sincronización de estados (state Synchronization) está hecha del servidor a los clientes remotos. El cliente local no tiene datos serializados a este, ya que comparte la escena con el servidor. Cualquier dato serializado a un cliente local sería redundante. Los ganchos (hooks) SyncVar son llamados en clientes locales.

Los datos no son sincronizados de clientes remotos al servidor. Esto es un trabajo para los comandos.

2.1.4.15.1 SyncVars

Son variables miembro de scripts NetworkBehaviour que son sincronizadas desde el servidor a los clientes. Cuando un objeto es generado o un nuevo jugador se une a un juego en progreso, son enviados los últimos estados de todos los SyncVars en los objetos en red que están visibles para ellos. Las variables miembro

están hechas por SyncVars al utilizar el atributo personalizado [SyncVar]. El estado de SyncVars es aplicado a objetos en clientes antes de que OnStartClient() sea llamado, para que el estado del objeto sea garantizado en estar actualizado dentro de OnStartClient(). Los SyncVars pueden ser tipos básicos tal como enteros, strings y floats. Estos también pueden ser tipos de Unity tal como Vector3 y structs definidas por el usuario, pero las actualizaciones para struct SyncVars son enviadas como actualizaciones monolíticas, sin cambios incrementales si los campos dentro de una struct cambian. Puede haber hasta 32 SyncVars en un solo script de NetworkBehaviour, esto incluye SyncLists.

Las actualizaciones de SyncVar son enviadas automáticamente por el servidor cuando el valor de una SyncVar cambia. No hay necesidad de realizar cualquier cambio manual de los campos para SyncVars.

2.1.4.15.2 SyncLists

Los SyncLists son como SyncVars pero estas son listas de valores en vez de valores individuales. Los contenidos de las SyncList están incluidos en unas actualizaciones de estado iniciales con el estado SyncVar. SyncLists no requieren de los atributos de SyncVar, estas son especificas a clases. Hay tipos de SyncList integrados para tipos básicos:

- SyncListString
- SyncListFloat
- SyncListInt
- SyncListUInt
- SyncListBool

También hay SyncListStruct que pueden ser utilizados para listas de structs definidas por el usuario. La clase struct derivada utilizada en SyncListStruct puede contener miembros de tipos básicos, arreglos, y tipos comunes de Unity. No pueden contener clases complejas o contenedores genéricos.

2.1.4.16 Acciones Remotas

El sistema de red tiene maneras de realizar acciones a través de la red. Este tipo de acciones a veces se llama Remote Procedure Calls (llamadas a procedimientos remotos). Hay dos tipos de RPCs en el sistema de red, Commands (comandos) que son llamados del cliente y corren en el servidor y ClientRpc que son llamados en el servidor y corren en los clientes.

En la figura 2.4 podemos observar los tipos y direcciones que toman estas acciones.

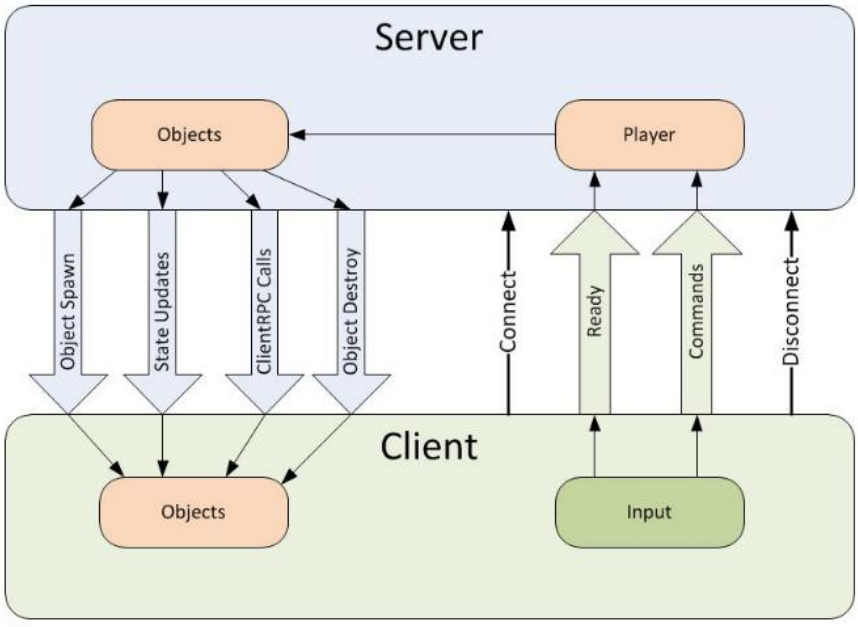

Figura 2. 4: Diagrama de direcciones que las acciones remotas toman.

2.1.4.17 Commands (Comandos)

Los comandos son enviados de objetos jugadores en el cliente a objetos jugadores en el servidor. Por seguridad los commands solamente pueden enviarse desde su objeto jugador, por lo que no pueden controlar los objetos de otros jugadores. Para hacer una función un comando, se agrega el atributo personalizado [Command] a este, además del prefijo Cmd. Esta función va a correr en el servidor cuando sea llamada en el cliente. Cualquier argumento será automáticamente pasado al servidor con el comando.

Las funciones de comandos deben tener prefijos con Cmd. Esta es una pista para leer código que llame el comando, esta función es especial y no es invocada localmente como una función normal.

2.1.4.18 Llamados ClientRpc

Las llamadas del ClientRpc son enviadas de objetos en el servidor a objetos en clientes. Estos pueden ser enviados de cualquier objeto servidor con un NetworkIdentity que fue generado, ya que el servidor tiene autoridad, entonces no hay problemas de seguridad con los objetos del servidor siendo capaces de enviar estas llamadas. Para hacer una función a una llamada ClientRpc, se agrega el atributo personalizado [ClientRpc] a este, y el prefijo Rpc. Esta función ahora va a ejecutarse en clientes cuando sea llamado en el servidor. Cualquier argumento va a automáticamente ser pasado a los clientes con un llamado ClientRpc.

Las funciones ClientRpc deben tener un prefijo Rpc. Esta es una pista cuando se lee el código que llame al método, esta función es especial y no es invocada localmente como una función normal.

Cuando ejecute un juego como un host (anfitrión) con un LocalClient, las llamadas ClientRpc serán invocadas en el LocalClient incluso si está en el mismo proceso que el del servidor. Por lo que el comportamiento de LocalClients (clientes locales) y RemoteClients (Clientes remotos) es el mismo para llamadas ClientRpc.

2.1.4.19 Argumentos para Acciones Remotas (Remote Actions)

Los argumentos pasados a los comandos y llamados ClientRpc son serializados y enviados sobre la red. Estos argumentos pueden ser:

- Tipos básicos (byte, int, float, string, UInt64, etc).
- Arreglos de tipos básicos.
- Structs conteniendo tipos permitidos.
- Tipos de matemáticas integrados en Unity (Vector3, Quaternion, etc).
- NetworkIdentity.
- NetworkInstanceId.
- NetworkHash128.
- GameObject con un componente NetworkIdentity adjunto.

2.1.4.20 Local Discovery

El componente NetworkDiscovery permite que los juegos de Unity se encuentren en una red local. Puede transmitir presencia, escuchar transmisiones y opcionalmente unir juegos coincidentes usando el NetworkManager entre los dispositivos. Esto no funciona a través de internet, sólo en las redes locales. Este componente utiliza la característica de difusión UDP de la capa de transporte de red.

Al igual que el NetworkManagerHUD, este componente tiene una GUI predeterminada para controlarla. El componente puede ejecutarse en modo

servidor o en modo cliente. Cuando está en modo servidor, envía mensajes de difusión a través de la red en el puerto especificado. Estos mensajes contienen la clave y la versión del juego, estos identifican este tipo particular de juego. Para evitar conflictos como el juego que intenta unirse a un juego de otro tipo, debe personalizar el valor del campo clave. El componente se debe ejecutar en modo servidor si se está hospedando un juego en esa máquina. Cuando no utiliza la GUI predeterminada, la función StartAsServer () hace que el componente se ejecute en modo servidor.

Cuando está en modo cliente, el componente escucha los mensajes de difusión en el puerto especificado. Cuando se recibe un mensaje y la clave del mensaje coincide con la clave en el componente NetworkDiscovery, esto significa que un juego está disponible para unirse a la red local. Cuando no se utiliza la GUI predeterminada, la función StartAsClient () hace que el componente se ejecute en modo cliente.

Cuando se utiliza la GUI por defecto, aparecerá un botón que hará que el cliente se una al juego (si hay un NetworkManager disponible).

Hay una función virtual en el componente NetworkDiscovery que se puede implementar para ser notificada cuando se reciban mensajes de difusión.

2.1.4.21 Network Clients y Servers

Muchos juegos multijugador podrán utilizar el NetworkManager para administrar las conexiones, pero es posible utilizar las clases NetworkServer y NetworkClient de nivel inferior directamente.

Cuando se utiliza el HLAPI, cada juego debe tener un servidor que aloje el juego. Así, cada participante en un juego multijugador puede ser un cliente, un servidor dedicado o una combinación de servidor y cliente al mismo tiempo. Este

rol de combinación es el caso común de un juego multijugador sin servidor dedicado.

Para juegos multijugador sin servidor dedicado, uno de los jugadores que ejecuta el juego actúa como el servidor para ese juego. En ese caso particular del jugador del juego estaría ejecutando un cliente local en lugar de un cliente remoto normal. El cliente local utiliza las mismas escenas y objetos Unity como el servidor y se comunica internamente utilizando colas de mensajes en lugar de enviar mensajes a través de la red. Pero, para el código HLAPI y los sistemas, el cliente local es sólo otro cliente, por lo que casi todo el código de usuario es el mismo si un cliente es local o remoto. Esto hace que sea fácil de hacer un juego que funciona en modo multijugador y stand-alone con el mismo código.

Un patrón común para los juegos multijugador es tener un objeto que gestiona el estado de la red del juego.

2.1.4.22 Networking en Dispositivos Móviles

El motor de Networking en Unity iOS/Android es completamente compatible con el networking para dispositivos móviles, por lo que el código para networking en el juego funciona en dispositivos iOS/Android. Sin embargo, se puede hacer reingeniería al código si será empleado principalmente para conectarse a redes WiFi o celulares. Además, dependiendo del dispositivo, el chip de networking puede también ser el cuello de botella puesto que el tiempo ida y vuelta de los pings entre dispositivos móviles (o entre un móvil y uno de escritorio) está entre los 40 y 60 ms (milisengundos), incluso en redes wifi de alto desempeño.

Al usar networking se puede crear un juego que pueda ser jugado en simultáneo desde escritorio y desde móviles conectados a wifi o a redes celulares.

En el último caso, el servidor del juego debe tener una dirección IP pública (accesible desde internet).

2.1.4.23 Referencias al Networking (Componentes de Red)

1. **NetworkAnimator**: es utilizado para sincronizar animaciones a través de la red.

2. **NetworkBehaviour:** son secuencias de comandos especiales que funcionan con objetos junto con el componente NetworkIdentity. Estos scripts son capaces de realizar funciones HLAPI como Comandos, ClientRPCs, SyncEvents y SyncVars.

3. **NetworkClient:** es una clase HLAPI que administra una conexión de red a un servidor, en este caso un NetworkServer. Se puede utilizar para enviar mensajes al servidor y recibir mensajes desde el servidor. NetworkClient también ayuda a administrar los objetos de red generados en el juego, enrutamiento de mensajes RPC y eventos de red.

4. **NetworkConnection:** es una clase HLAPI que encapsula una conexión de red. Los objetos NetworkClient tienen NetworkConnections y NetworkServers tienen varias conexiones, una de cada cliente. NetworkConnections tiene la capacidad de enviar arrays de bytes u objetos serializados como mensajes de red.

5. **NetworkDiscovery:** es una clase que permite que la aplicación Unity que usa el sistema de red se encuentre en una red local.

6. **NetworkIdentity:** es un componente de Unity que está en el corazón del nuevo sistema de red. Este componente controla la identidad de red de un objeto y hace que el sistema de red sea consciente de ello. Con el sistema autoritario del servidor del sistema de red Unity, los objetos en red con NetworkIdentities deben ser generados por el servidor mediante NetworkServer.Spawn (). Esto hace que se les asigne un NetworkInstanceId y que se creen en los clientes que están conectados al servidor. Los objetos de escena se tratan un poco de forma diferente a los objetos dinámicamente instanciados. Estos objetos están presentes en la escena tanto en el cliente como en el servidor. Sin embargo, al crear el juego todos los objetos de escena con identidades de red están deshabilitados. Cuando el cliente se conecta al servidor, este le dice al cliente qué objetos de escena deben estar habilitados y cuál es su información de estado más actualizada a través de mensajes generados. Esto garantiza que el cliente no contendrá objetos colocados en ubicaciones incorrectas en ese momento al iniciar la reproducción o incluso objetos que se eliminarán inmediatamente al conectarse porque algún evento los ha eliminado antes de que el cliente se haya conectado. La casilla Server Only (Solo servidor) garantizará que un objeto determinado no se generará o habilitará en los clientes.

El Local player authority (autoridad del jugador local) permite que el objeto sea controlado por el cliente que lo posee. Esto es utilizado por otros componentes como NetworkTransform.

Este componente contiene información de seguimiento, como ID (Identificador) de escena, ID de red e ID de los assets que se han asignado al objeto. El ID de escena es válido en todos los objetos de escena con un componente NetworkIdentity. El identificador de red es el ID de esta instancia

de objeto particular, puede haber varios objetos instanciados de un tipo de objeto particular y el ID de red se utiliza para identificar qué objeto es. El identificador de asset se refiere a qué recurso de origen creó el objeto. Esto se utiliza internamente cuando un objeto prefabricado particular se genera sobre la red. Esta información se expone en el panel de vista previa en la parte inferior del inspector.

7. **NetworkManager:** es una clase de nivel superior que le permite controlar el estado de un juego en red. Proporciona una interfaz en el editor para controlar la configuración de la red, los prefabs utilizados para aparecer y las escenas que se utilizan para diferentes estados de juego en red.

8. **Network Manager HUD:** El Administrador de red HUD proporciona una interfaz de usuario predeterminada para controlar el estado de la red del juego. También muestra información sobre el estado actual del NetworkManager en el editor. Esto proporciona ayuda en el desarrollo y prueba.

9. **NetworkServer:** es una clase de HLAPI (High-Level-API) que gestiona las conexiones de varios clientes.

10. **NetworkStartPosition:** es utilizado por NetworkManager al crear objetos del jugador. La posición y la rotación de NetworkStartPosition se utilizan para colocar el objeto de jugador recién creado.

11. **NetworkTransform:** sincroniza el movimiento de objetos de juego a través de la red. Este componente tiene en cuenta la autoridad, por lo que los objetos LocalPlayer (que tienen autoridad local) sincronizan su posición del cliente al

servidor y luego a otros clientes. Otros objetos (con autoridad de servidor) sincronizan su posición del servidor a los clientes. Para sincronizar el movimiento, se debe agregar al prefab de juego este componente. El componente requiere que los objetos de juego tengan un NetworkIdentity. Hay que tener en cuenta que los objetos en red deben generarse para sincronizarse.

12. **NetworkTransformChild:** sincroniza la posición y la rotación de un objeto secundario de un objeto con un componente de NetworkTransform. El componente NetworkTransformChild debe colocarse en el mismo objeto raíz que el NetworkTransform. Esta clase no utiliza la física, simplemente sincroniza la posición y la rotación del hijo hacia valores actualizados a una velocidad personalizable.

2.1.5 Servicio multijugador de Unity

Es la forma más sencilla de crear juegos multijugador en unity, sencillo de implementar y altamente personalizable. Unity provee servidores y un servicio matchmaking que asegura que los jugadores puedan fácilmente conectarse unos con otros, en internet.

Es servicio en la nube (cloud), que consta de una infraestructura global distribuida, utilizando centros de datos en Estados Unidos, Europa y el sur este de Asia

Utiliza una topología Peer to Peer (P2P), donde un cliente host (anfitrión) juega el papel de server. Las bibliotecas de redes principales son flexibles y pueden utilizarse para soportar cliente-servidor. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.

Un centro de datos o centro de Proceso de datos (Data Center) es un espacio que alberga los recursos tecnológicos que permiten procesar una gran cantidad de información. Estos lugares también se denominan centro de cálculo o centro de cómputo. Esta definición engloba las dependencias y los sistemas asociados gracias a los cuales

- Los datos son almacenados, tratados y distribuidos al personal o procesos autorizados para consultarlos y/o modificarlos.
- Los servidores en los que se albergan estos datos se mantienen en un entorno de funcionamiento óptimo.

Un servidor cloud se sustenta sobre una infraestructura especial que le permite abstraerse totalmente del hardware, donde no sólo la máquina o el servidor están virtualizados, sino que otros componentes como la red y el almacenamiento también lo están. Está abstracción respecto al hardware significa que un servidor cloud no está atado a un ordenador físico concreto, no está ubicado en un único ordenador.

Capítulo 3

3.1 Diseño del Videojuego

3.1.1 Descripción

Inspirado en las artes marciales mixtas (también llamado Mixed Martial Arts o MMA por la población de habla inglesa) el cual es un deporte de combate surge este proyecto de videojuego como parte del trabajo de grado para dispositivos móviles.

Llamado K.O. Manager, sus siglas K.O. provienen del inglés knock out y significa fuera de combate, sucede cuando el peleador no se encuentra en condiciones físicas de continuar su pelea. Pertenece al género de los deportes, es un videojuego de dirección o managment (gestión), donde los jugadores hacen el papel de jugador y al mismo tiempo de entrenador.

Su dinámica consiste en dos peleadores los cuales se enfrentan con el objetivo de vencerse uno del otro, usando habilidades individuales o combos los cuales son una sucesión de golpes. Durante la partida ambos usuarios jugaran 3 rounds los cuales serán decisivos para obtener la victoria de la pelea.

El videojuego ha sido desarrollado desde cero, por un equipo de tres programadores los cuales abarcan áreas específicas e incluso se inmiscuyen en otras para aportar sus ideas. K.O Manager consta de una sección de entrenamiento del jugador, una sección para jugar individual y una sección multijugador.

Este trabajo, se enfoca en la parte multijugador, ya que su esencia es la de conectar dispositivos para generar comunicación entre ellos, y así permitir el flujo de datos para ejecutar la dinámica de juego.

3.1.2 Análisis y diseño

3.1.2.1 Requisitos

Tanto el estado de producción del producto como el de desarrollo requieren del uso de hardware esencial para su prueba, específicamente se usan teléfonos inteligentes (smartphones) con la característica de poseer conexión wifi e internet.

Además es necesario para pruebas un router, con la finalidad de proporcionar conectividad a nivel de red entre los dispositivos.

Tomando en cuenta la perspectiva del usuario, se debe mantener la simplicidad visual de las funcionalidades para así ser más amistoso ante el jugador y poder generar una experiencia de juego agradable.

3.1.2.2 Diseño general

Como el objetivo principal es conectar dispositivos para establecer comunicación se plantea un caso de uso general el cual engloba las funcionalidades básicas de esta aplicación donde el actor es el jugador humano.

En la figura 3.1 podemos observar el caso de uso general de la capacidad multijugador.

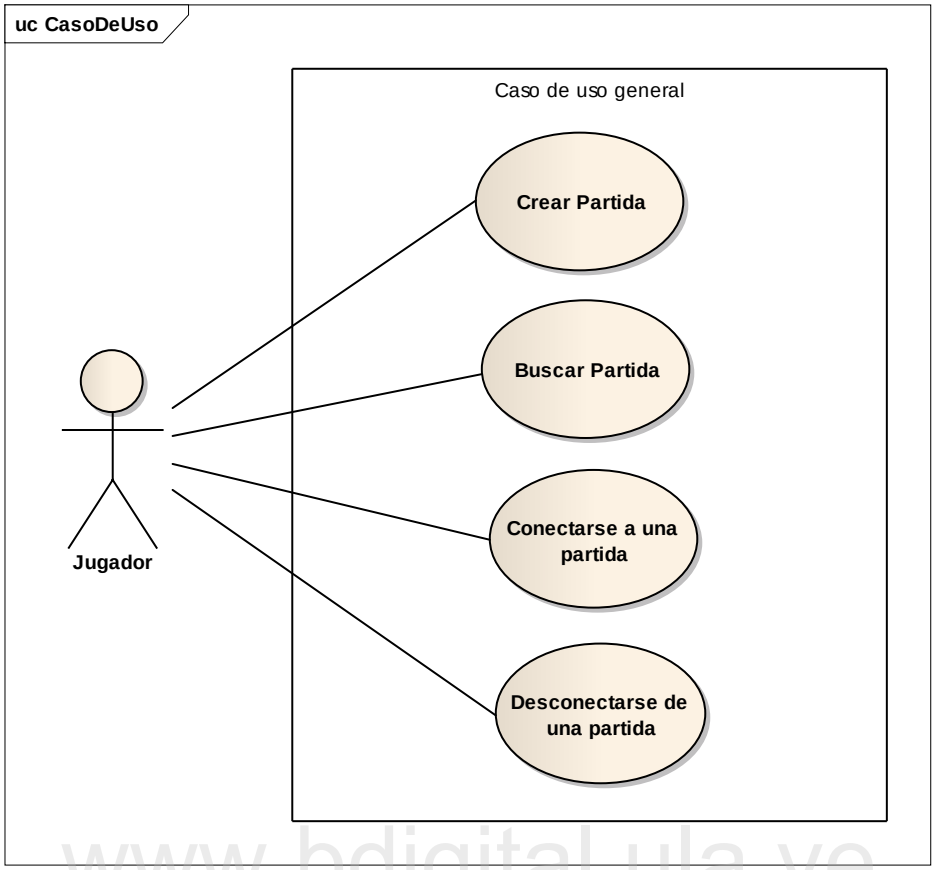

Figura 3. 1: Caso de uso general.

3.1.2.3 Diagrama de clases

A continuación se presentan los diagramas de clases donde las figuras 3.2 y 3.3 muestran las clases y relaciones implementadas en el diseño del videojuego. Cabe destacar que la mayoría de las clases surgieron como herencia de las que provee el motor gráfico, debido a que necesitaban ser personalizadas.

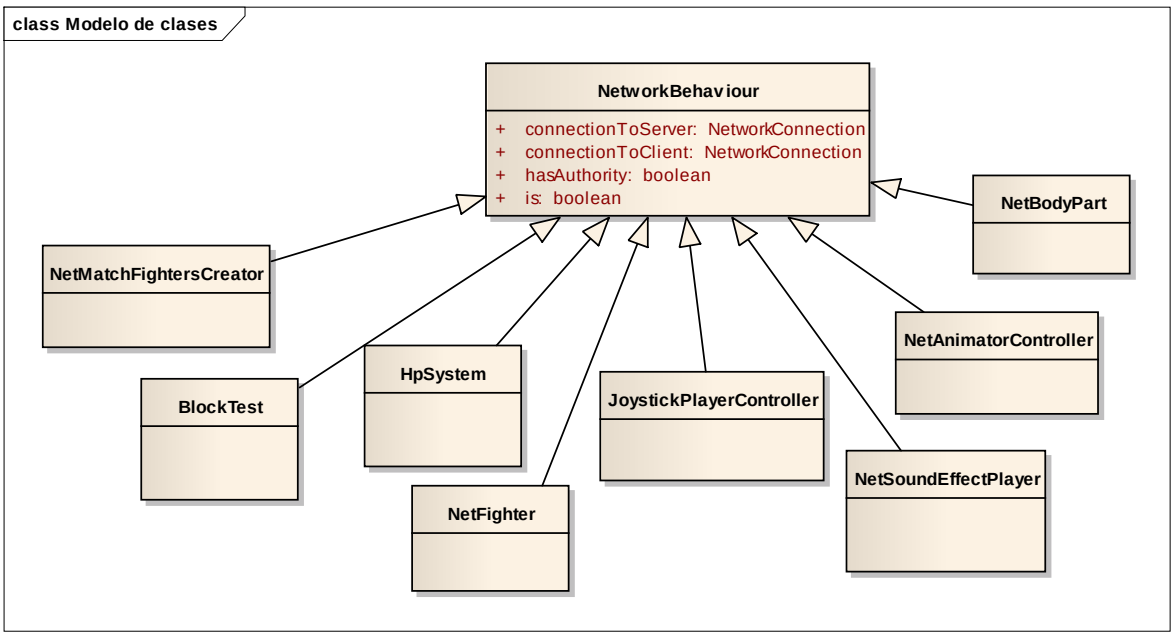

Figura 3. 2: Diagrama de clases 1.

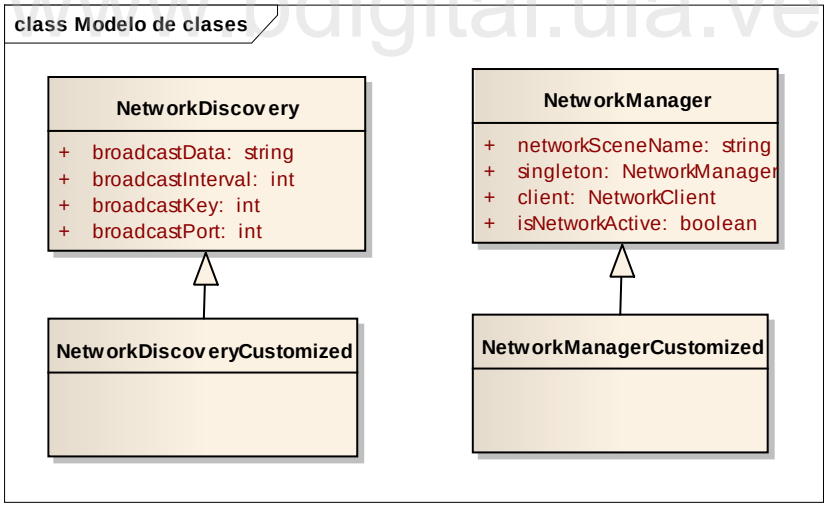

Figura 3. 3: Diagrama de clases 2.

3.1.2.4 Diagrama de componentes

Describen los elementos físicos del sistema y sus relaciones. Estos elementos físicos tales como clientes y servidor poseen componentes que hacen que la funcionalidad multiplayer (red) se ejecute (ver figura 3.4).

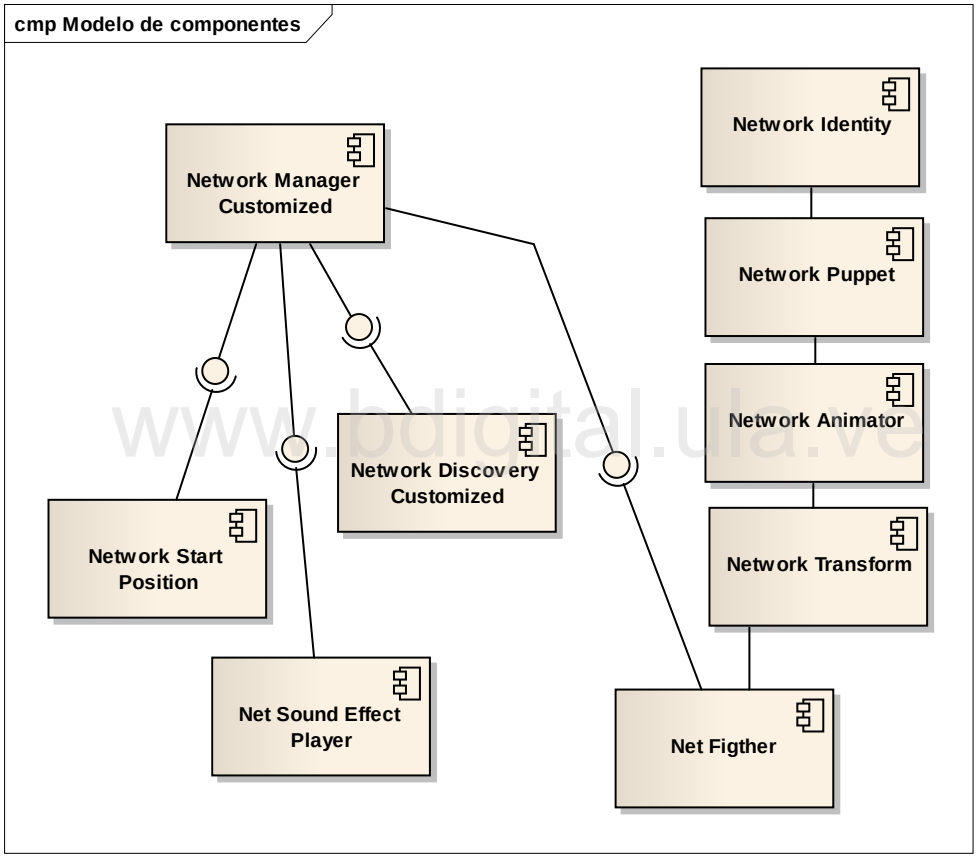

Figura 3. 4: Diagrama de componentes.

3.1.2.5 Diagrama de despliegue

El siguiente diagrama se usa para modelar la disposición física de los artefactos de software en la plataforma de hardware, es decir la perspectiva de la red. (ver figura 3.5).

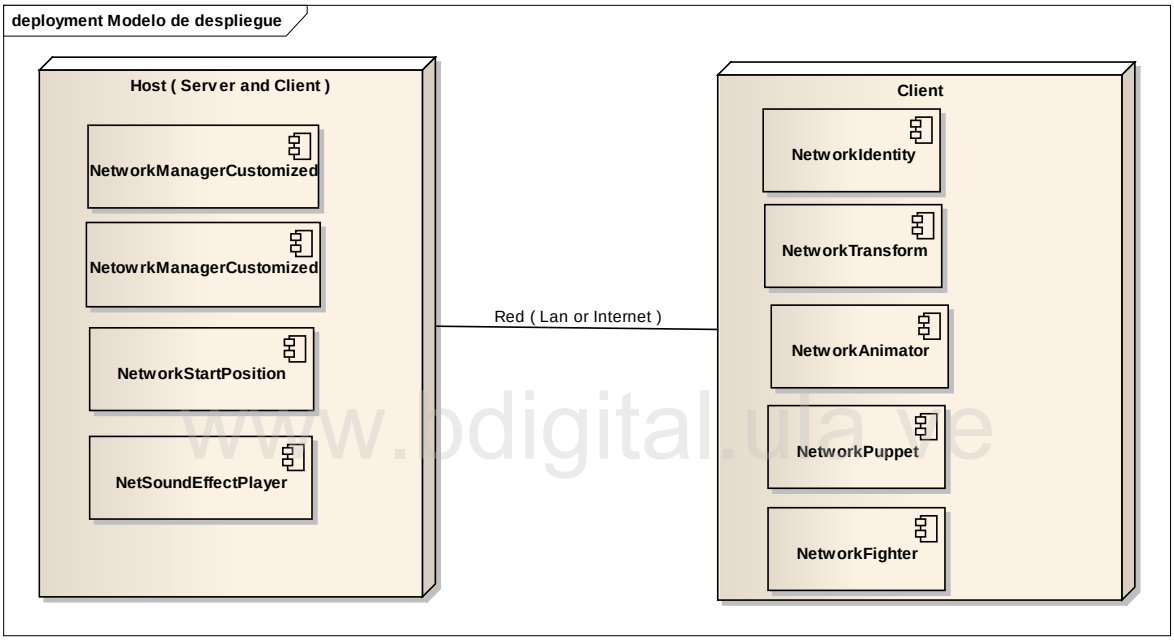

Figura 3. 5: Diagrama de despliegue.

3.2 Desarrollo

Debido a que todo el videojuego ha sido desarrollado desde lo más bajo, poco a poco se fue viendo el progreso, hasta obtener lo mínimo requerido para empezar a implementar la funcionalidad multijugador. Entre estas características se encuentran:

- Un escenario.
- Un modelo 3d del jugador, el cual puede desplazarse sobre una superficie.
- Controles.
- Reacciones en el jugador, animaciones, vida, estamina y sonidos.
- Interfaz gráfica (menú).

Principalmente se introdujo el Administrador de Red, quien será el encargado de controlar la parte multijugador, así como también los personajes usados para ejecutar la partida. El personaje tiene la característica principal de poseer un identificador de red el cual será controlado mientras esté el juego ejecutándose; dicho identificador está vinculado con su ubicación en el espacio, lo que nos permite saber quién es y su permisología en el juego.

3.2.1 Crear Partida

Esta área, comprende todo lo relacionado a la inicialización del juego, en donde se crea un servidor que permitirá auspiciar la partida. En el servicio LAN (Red de área local) el host cumple el papel de cliente servidor al mismo tiempo, es decir, quien crea el juego será el anfitrión (server). En el servicio de internet, el anfitrión será un servidor en la nube, pero el cliente que lo crea tendrá el mismo control sobre él como si lo hiciera localmente. Todo esto se logra a través del administrador de red, personalizando su comportamiento a través de una clase heredada.

Luego de iniciar la partida, se procede a instanciar procesos básicos como es el de espera, el cual congela temporalmente el juego hasta que el jugador contrario se conecte. Durante este tiempo se empieza a emitir sobre la red información indicando que el servidor está disponible para un usuario en la red (para lan se usa

el network discovery y para el internet el matchmaking lo provee automaticamente). Luego al estar ambos jugadores se procede a seleccionar las habilidades de pelea tales como patadas altas, ganchos o combos. Estando ambos jugadores listos, se procede a instanciar las funcionalidades en la partida, como lo son la vida, animaciones, sonidos, cámara, ubicación de los personales y rotaciones de los mismos. Una vez iniciada la partida durante el tiempo de juego se manejan aspectos internos para que todos estos procesos se lleven a cabo, tales como:

- Sincronización de posiciones: Las posiciones juegan un papel importante ya que refleja un punto en el espacio en común entre dispositivos, y es de suma importancia que sea el mismo en ambos.

- Sincronización de animaciones: Las animaciones son movimientos en los personajes, producto de una reacción o acción, estas se deben transmitir a través de la red, para que dichos efectos se reproduzcan al mismo tiempo en ambos juego.

- Sonido: Generado por la acción de algún evento en el juego, como por ejemplo un golpe, reproduciéndose este sonido de animación en ambos dispositivos.

- Sincronización de vida y estamina: estas variables las cual controla el servidor, son sincronizadas, permitiendo así estar presente para tomar decisiones como son las de morir, cansancio (la estamina maneja cuanta energía posee el jugador y esta se desgasta al momento de usar habilidades pero también se regenera con el tiempo), efectos visuales como lo son animaciones en las barras de vida, entre otros.

En la figura 3.6 se presenta el caso de uso llamado crear partida.

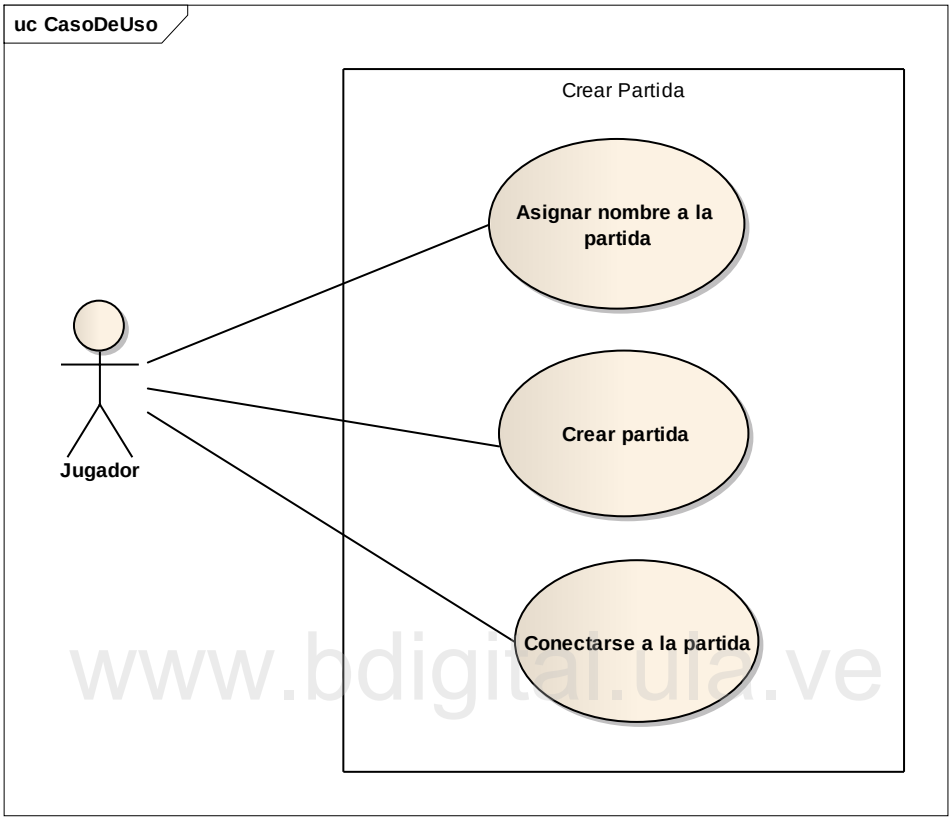

Figura 3. 6: Caso de uso crear partida.

Descripción textual del diagrama de caso de uso crear partida

Actores	Jugador
Propósito	Funcionalidad que permite al usuario crear y unirse a una partida de manera simultanea
Precondición	El jugador no debe estar en ninguna partida, es decir debe estar disponible para ejecutar esa acción.

Postcondición	El usuario habrá iniciado una partida, y estará esperando por una conexión entrante que será el jugador contrario.		
Curso Normal de Eventos			
Paso	**Acción del actor**	**Paso**	**Respuesta del sistema**
1	El usuario selecciona play with other en el menu, luego local o internet dependiendo de la opción que escoja	2	El sistema retorna una vista en la que se visualiza un cuadro de texto para el nombre del servidor y algunos botones entre los que se encuentra crear partida (create game).
3	El usuario llena la información requerida en el cuadro de texto y procede a seleccionar el botón crear partida. Si no llena el cuadro de texto se le asignara un nombre por defecto.	4	El sistema iniciara el servidor, pondrá su disponibilidad sobre la red y unirá automáticamente al jugador en la partida. Luego el sistema espera por el otro jugador.

3.2.2 Buscar partida

Todo cliente que quiera conectarse a una partida debe buscar, para obtener la dirección del juego y poder conectarse, llamamos dirección a un puerto y a un ip.

Lo que se espera es obtener una lista de todos los servidores disponibles, llamamos disponible al servidor que posee solo un jugador y está esperando una

conexión entrante. Cabe destacar que los servidores con 2 jugadores o servidores no disponibles no serán mostrados al cliente para evitar incongruencias. En la figura 4.7 se presenta el caso de uso llamado buscar partida.

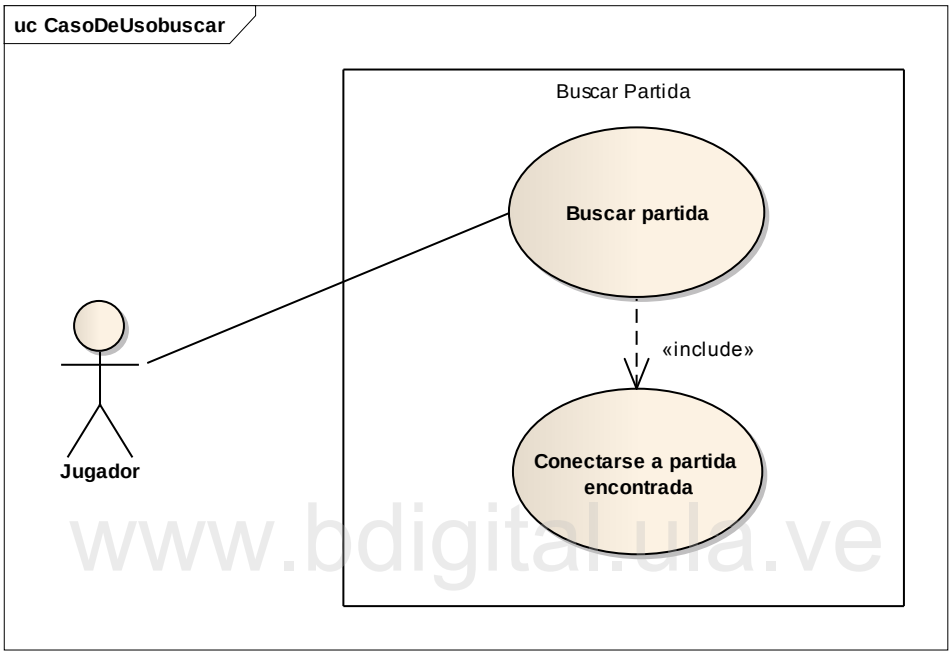

Figura 3. 7: Caso de uso buscar partida.

Descripción textual del diagrama de caso de uso buscar partida

Actores	Jugador
Propósito	Funcionalidad que permite al usuario buscar y encontrar una partida para poder unirse posteriormente.
Precondición	El jugador no debe estar en ninguna partida, es decir debe estar disponible para ejecutar esa acción. Para tener éxito algún jugador en otro dispositivo debe haber creado una partida previamente.

Postcondición	El jugador se habrá conectado exitosamente a una partida.		
Curso Normal de Eventos			
Paso	**Acción del actor**	**Paso**	**Respuesta del sistema**
1	El usuario selecciona play with other en el menu“, luego local o internet dependiendo de la opción que escoja	2	El sistema retorna una vista en la que se visualiza un cuadro de texto para el nombre del servidor y algunos botones entre los que se encuentra buscar servidores (find servers).
3	El usuario selecciona buscar servidores	4	El sistema iniciara la búsqueda desplegando una lista de botones con los servidores disponibles para unirse en caso de que existan

3.2.3 Conectarse a partida

Localmente existen dos maneras de conectarse a una partida, a través de la búsqueda o conociendo el ip local. En internet solo existe una y es buscando servidores. Al establecer una conexión exitosa el anfitrión (server) deja de emitir información a la red, es decir el servidor pasa a un estado no disponible para nuevos usuarios, esto es necesario ya que es un juego solo de 2 participantes.

3.2.4 Desconectarse de una partida

Cuando por fallas o por deseo de abandonar la partida el jugador desaparece, el servidor notara su ausencia notificando a los clientes para

posteriormente apagar el servicio. Cabe destacar que si desea jugar nuevamente, puede crear otra vez la partida, sin permitir reconexiones.

Capítulo 4

4.1 Resultados

En general los resultados obtenidos cumplen con los aspectos fundamentales multijugador, que es la sincronización. Pero además existen otros detalles que complementan este diseño, ya que es un sistema que está integrado por componentes complejos para poder llevar a cabo este objetivo.

4.1.1 Pruebas y validación del sistema

El conjunto de pruebas para la detección de fallas fue implementado a través de las iteraciones para posteriormente corregirlos.

4.1.2 Objetivos de las pruebas

Validar que el sistema cumpla con los objetivos multijugador, es decir sincronización y comunicación entre dispositivos.

4.1.3 Técnicas a implementar

Realizada a través de pruebas manuales mediante la interfaz de usuario, donde cada caso de uso fue utilizado como guía.

4.1.4 Criterios de las pruebas

Se llamara caso exitoso cuando se obtenga un resultado esperado en la prueba y así validar el óptimo desempeño del sistema.

4.1.5 Pruebas realizadas

A continuación se presentan las pruebas realizadas para cada caso de uso.

4.1.5.1 Crear partida

Flujo de la prueba

Se inicia el videojuego, se accede a la sección de jugar, luego a la sección de jugar con otro (play with other), se escoge internet o local, se asigna un nombre del servidor y se crea el juego (create game). En las figuras 4.1, 4.2 y 4.3 se observa los modos multijugador en lan y en internet.

Figura 4. 1: Menú Multijugador.

Figura 4. 2: Menú multijugador local.

Figura 4. 3: Menú multijugador internet.

Postcondiciones

Luego de haber sido creada la partida, el jugador entrara en un tiempo de espera, mientras el otro jugador se une a la partida para posteriormente escoger habilidades, combos y empezar la partida (ver figura 4.4).

Figura 4. 4: Selección de habilidades antes de empezar la partida.

4.1.5.2 Buscar partida

Flujo de prueba

Se inicia el videojuego, se accede la sección de jugar, luego a la sección de jugar con otro (play with other), se escoge internet o local y se procede a seleccionar la opción buscar servidores (find servers). En la figura 4.5 observamos la interfaz gráfica para el modo multijugador desplegando la opción buscar servidores.

Figura 4. 5: Menu multiplayer local, buscando servidores disponibles para conectarse.

Postcondiciones

De haber servidores disponibles, aparecerá una serie de botones lista para poder ser accedida y empezar la partida. Cabe destacar cuando ambos participantes están en la partida se procede a la selección de habilidades o combos que serán usados al momento del combate. Posteriormente los personajes comienzan a combatir hasta obtener un vencedor como se observa en las figuras 4.6 y 4.7.

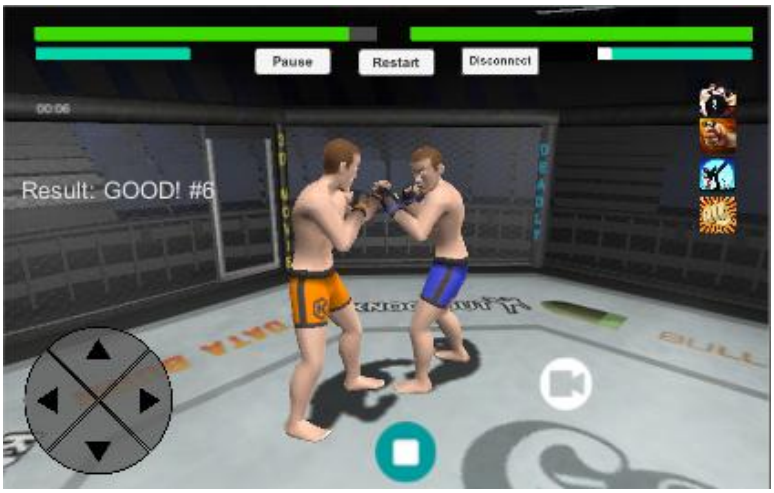

Figura 4. 6: Partida de K.O Manager.

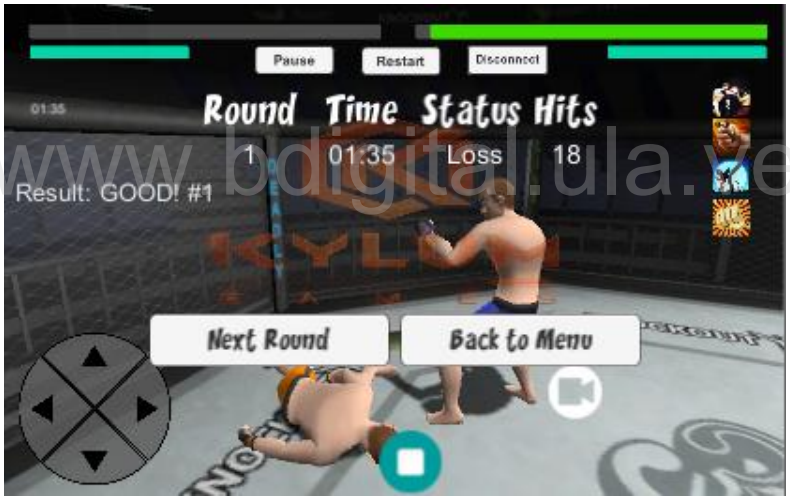

Figura 4. 7: Estadísticas al final de la partida.

4.1.5.3 Extras Validaciones

Pausar: esta detiene el juego de ambos clientes hasta que alguno de los dos decida continuar. En la figura 4.8 se observa esta funcionalidad activada.

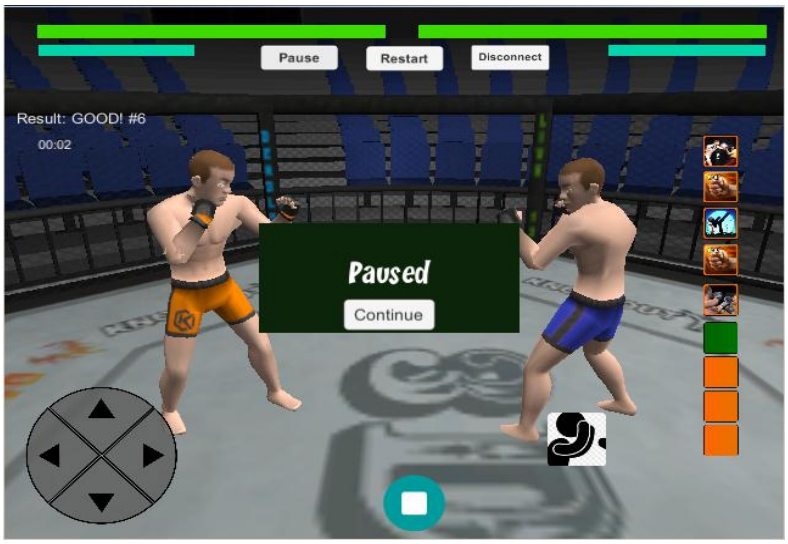

Figura 4. 8: Pausa activada.

Detectar abandono: será desplegado cuando un jugador se desconecta de la partida como se puede ver en la figura 4.9.

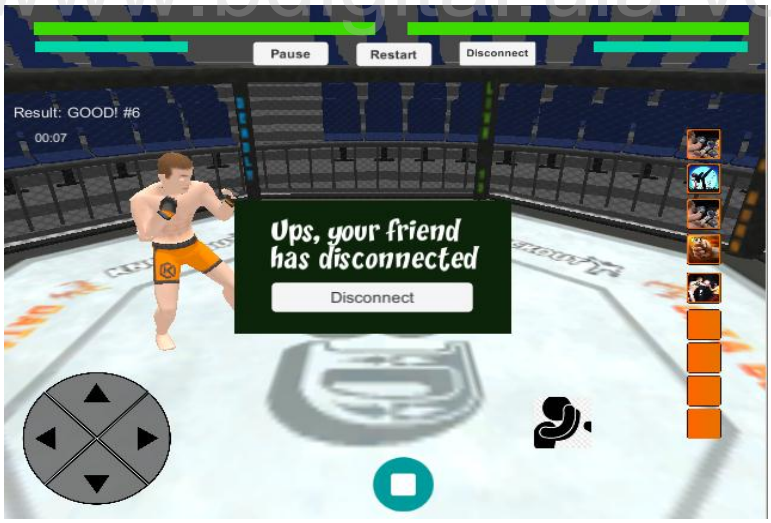

Figura 4. 9: Abandono detectado.

Validación al crear un server: un mismo dispositivo no puede crear dos servidores. En la figura 4.10 podemos ver cuando un dispositivo intenta crear dos partidas al mismo tiempo y la validación aparece.

Figura 4. 10: Validación al crear servidor en el mismo dispositivo.

Dinámica de rounds y sincronización: Cuando un round concluye aparecerá un dialogo que indica sus estadísticas (tiempo, round, ganador y golpes), además de poseer el botón para permitir jugar el siguiente. En la figura 4.11 podemos observar dos dispositivos en los cuales resalta la sincronización de vida, estamina, posiciones y animaciones.

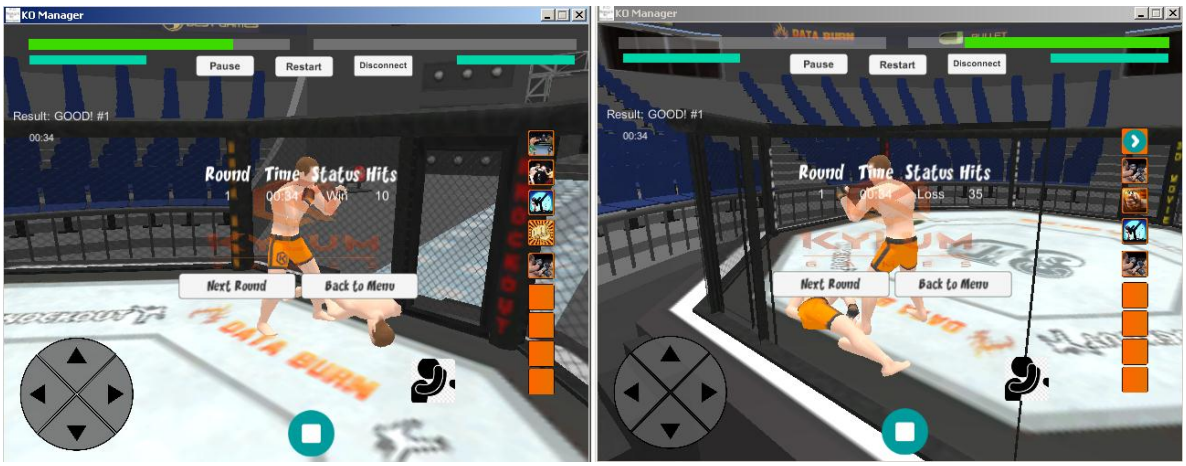

Figura 4. 11: Dinámica de round y sincronización.

Capítulo 5

Conclusiones y recomendaciones

5.1 Conclusiones

Cada vez más, las tecnologías móviles llegan a las personas, envolviéndolas como parte del entretenimiento, usualmente videojuegos. Siendo más atractivo interconectar estos videojuegos para disfrutarlos en grupos; es decir, donde participen desconocidos, conocidos o amigos, estableciendo comunicación entre dispositivos con un objetivo en común: entretenimiento y diversión.

Se demostró que se pueden conectar dos o más dispositivos a través de la red local o internet, permitiendo así la transmisión de información entre ellos con diferentes fines, en este caso un videojuego multijugador. Usando como base los conocimientos aprendidos durante la carrera, especialmente los de programación e Ingeniería del Software, fue posible el desarrollo de este producto. Además del nuevo conocimiento adquirido sobre los videos juegos y su desarrollo el cual ayudara a los amantes del entretenimiento digital y especialmente a las personas que les gusta los videojuegos, expandiendo esto más allá a través de las tecnologías multiusuario. Cabe destacar que durante el desarrollo, se recibió retroalimentación de parte de usuarios (amigos y familia), los cuales fueron de utilidad para que todas las funcionalidades fuesen mejoradas y así generar un mejor modo de juego.

5.2 Perspectivas a futuro

K.O Manager está siendo desarrollado con el objetivo de ser un Startup, donde un grupo de jóvenes incluyendo al autor, integren todas las ideas necesarias para hacerlo realidad. Está planificado para ser un juego de acceso libre para todos, disponibles en plataformas tanto Apple (App Store) como Android (Play Store). Se planea seguir agregándole mejores dinámicas de juego, con el fin de hacerlo más atractivo y divertido para los futuros usuarios.

5.3 Recomendaciones

- Generar más Startups orientadas al mundo del entretenimiento como son los videos juegos. Fomentar el emprendimiento

- Aprovechar la interconexión entre dispositivos para hacer grandes proyectos.

- No perder de vista la evolución tecnológica al momento de desarrollar las ideas

- Trabajar en equipos (grandes proyectos surgen entre varias personas).

- Seguir metodologías de desarrollo.

- Apostar a las ideas de los estudiantes, donde se exista el apoyo y críticas constructivas.

- Discutir entre varias personas las nuevas ideas, con el fin de mejorarlas. Siempre es bueno discutir propuestas, esto hará que el producto sea mejor.

Bibliografía

[1] What exactly is a startup?

http://www.investopedia.com/ask/answers/12/what-is-a-startup.asp?ad=dirN&qo=investopediaSiteSearch&qsrc=0&o=40186

[3] Historia de los videojuegos para teléfonos móviles

http://people.uta.fi/~tlilma/Mobile_Games.pdf

[4] Videojuegos para teléfonos móviles

https://en.wikipedia.org/wiki/Mobile_game

[5] Resumen Cifras del Primer trimestre 2017

http://www.conatel.gob.ve/informe-cifras-del-sector-tercer-trimestre-2016/

[6] Ingeniería del software

https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software#Obtenci.C3.B3n_de_los_requisitos

[7] Desarrollo iterativo y creciente

https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente#Etapa_de_inicializaci.C3.B3n

[8] Modelo iterativo e incremental

https://danelly1236.files.wordpress.com/2013/12/modelo-iterativo-incremental-presentacion2.pdf

[9] Play Store (Plataforma de distribución digital)

https://es.wikipedia.org/wiki/Google_Play

[10] App Store (Plataforma de distribución digital)

https://es.wikipedia.org/wiki/App_Store

[11] Lenguaje de programación C Sharp

https://es.wikipedia.org/wiki/C_Sharp

[12] Documentación de Unity

https://docs.unity3d.com/Manual/

[13] Documentación de Unity Networking

https://docs.unity3d.com/Manual/UNet.html

[14] Videojuego de Deportes

https://en.wikipedia.org/wiki/Sports_game