Trabajo de fin de grado

Grado en ingeniería informática

Especialidad en ingeniería de computadores

Universidad Carlos III de Madrid

Diseño de un juego multijugador en Android

Autor: Hugo Alberto Huerta García

Tutor: Daniel Higuero Alonso-Mardones

Co-director: Juan Manuel Tirado Martin

Curso 2011-2012

Índice de contenido

Índice de tablas

Índice de figuras

1. Introducción

Los videojuegos están cada vez más a la orden del día, actualmente tienen una gran influencia en el mercado, por lo que es posible realizar negocios o proyectos en torno a ellos.

Este trabajo de fin de grado se trata de un proyecto de investigación en el que se quiere conseguir diseñar un juego multijugador para dispositivos móviles Android capaz de albergar al mayor número posible de jugadores.

Para ello primero se analizarán artículos, trabajos o juegos que tratan de este tema, lo que dará resultado a un diseño del cual se desarrollará una versión base del juego. Una vez que se tenga suficiente información se procederá a la implementación de optimizaciones para aumentar el número de jugadores soportado por los terminales.

También se trata tanto el tema de sistemas operativos para móviles como el tema de entornos de desarrollo o librerías ya que su uso reduce el tiempo de desarrollo significativamente.

1.2 Motivación

Hoy en día los dispositivos móviles son cada vez más comunes, cada vez más gente los utiliza a diario, tanto como herramienta de trabajo como para su ocio. Al igual que los ordenadores, son capaces de ejecutar aplicaciones y juegos, pero en cambio, los dispositivos móviles tienen menor capacidad.

La motivación de este trabajo de fin de grado es el de diseñar una arquitectura para un juego multijugador para móviles capaz de albergar al mayor número posible de jugadores al mismo tiempo, el problema está en que los recursos tanto de la máquina que proporcionará el servicio como de los dispositivos móviles son limitados. Para ello se partirá de algoritmos y arquitecturas ya existentes, se analizarán describiendo que objetivos buscan o cómo funcionan y después se propondrán optimizaciones que mejorarán estas arquitecturas.

1.3 Objetivos

El objetivo de este proyecto es el de llegar a conocer los límites de los dispositivos móviles en cuanto a la ejecución de juegos multijugador y una vez alcanzados, descubrir si es posible superar esos límites utilizando distintas técnicas, como por ejemplo utilizando una buena arquitectura.

Para lograr este objetivo se diseñarán optimizaciones, las cuales reducirán los recursos utilizados por cada jugador aumentando así el número de posibles jugadores que podrá tener.

Otro objetivo que surge a partir del primero es el de diseñar e implementar un prototipo de juego con las optimizaciones previamente diseñadas para comprobar si es posible utilizarlas y determinar cuál es su impacto.

1.4 Estructura del documento

El documento se divide en 5 partes, la primera es el estado del arte, en ella se empieza describiendo el entorno, define lo que es Android y lo que es un juego multijugador, para luego describir cómo están diseñadas las arquitecturas actuales.

En la segunda parte se realiza un análisis exhaustivo de los posibles entornos o librerías que se pueden utilizar para implementar un juego multijugador en Android.

En la tercera, se explica cómo está diseñado el juego propuesto, se exponen imágenes y esquemas de cómo funcionan el cliente, el servidor y la conexión que los une.

En la cuarta parte se realiza una evaluación del diseño propuesto, explicando los recursos que tienen los dispositivos móviles indicando cuáles son los límites de cada uno a partir de una serie de pruebas realizadas con el juego desarrollado.

Por último están las conclusiones, en las que se describe qué se ha conseguido con el proyecto y que es lo que se ha aprendido.

1.5 Acrónimos

AAC – Advanced Audio Coding

API – Aplication programming interface

ARM – Advanced RISC machine

CPU – Central processing unit

DNS – Domain name system

GNU – General public license

GPS – Global positioning system

IDE – Integrated development environment

IP – Internet protocol

IVA – Impuesto sobre el valor añadido

LGPL – GNU lesser public license

LWJGL – Lightweight java game library

MIDI – Musical instrument digital interface

MMO – Massive multiplayer online

MP3 – MPEG audio layer III

MPEG – Moving picture experts group

P2P – Peer to peer

RAM – Random access memory

RISC – Reduced instrucion set computer

SDK – Software development kit

SSL – Secure sockets layer

TCP – Transmission control protocol

TMX – Tile map xml

UDP – User datagram protocol

WAV – Waveform audio file format

XML – Extensible markup language

1.6 Definiciones

Ancho de banda – Es la capacidad máxima de transferencia de datos a través de la red.

Bot – Es un jugador controlado por software.

Cliente – Programa que activamente se conecta al servidor

Dirección IP – Es una serie de números que identifica a cada dispositivo en la red.

Google Play – Es la tienda de google donde se pueden adquirir las aplicaciones y juegos para Android.

Librería o entorno – Se llama librería a un grupo de recursos utilizados para desarrollo software [1].

Servidor – Programa que proporciona servicio al cliente.

Script – Es un conjunto de instrucciones que posteriormente será interpretado por un programa.

Sprite – Es un objeto visible en pantalla que tiene varios atributos como posición, velocidad, estado.

Smartphone – Es un teléfono móvil con capacidad para realizar cómputos avanzados y por consiguiente permite el uso de sistemas operativos como Android [2].

Software – Es la parte lógica de un sistema informático [3].

Switch – Es un dispositivo hardware de interconexión de redes de computadores cuya función es interconectar dos o más segmentos de red.

Tablet – Es un computador portátil más grande que un teléfono móvil y que incluye una pantalla táctil [4].

Tasa de transferencia de datos – La cantidad de información que se transmite por unidad de tiempo.

2. Estado del arte

Es importante definir la plataforma donde se va a desarrollar el juego por lo que primero se van a explicar dos de los sistemas operativos más utilizados en los que se puede desarrollar un juego multijugador.

Después se expondrán las arquitecturas de varios juegos multijugador ya existentes en el mercado.

En la última parte del estado del arte se va a hablar de las tecnologías actuales, de cómo están diseñadas, que objetivos se quieren conseguir, así como de los dos tipos más comunes, comparándolos y exponiendo sus ventajas e inconvenientes.

2.1 Sistemas operativos

Tanto en Android como en iOS se definirán qué partes los componen, sus características, cómo funcionan y en el caso de Android las versiones actuales indicando el porcentaje de uso de cada una.

2.1.1 Android

Android es un sistema operativo basado en Linux creado para ser usado en dispositivos móviles. Permite el uso de gráficos 2D y 3D usando una variante para sistemas empotrados de la librería OpenGL llamada OpenGL ES. La plataforma hardware habitual que usa Android es ARM.

Es la plataforma Smartphone más vendida, con más de 190 millones de dispositivos activados por todo el mundo [5]. También tiene una gran comunidad de desarrolladores que aportan aplicaciones y juegos, estas aplicaciones se pueden descargar de Google Play market.

Para facilitar el desarrollo de aplicaciones y juegos, Android pone a disposición de los programadores una serie de herramientas, que son un compilador, un depurador y un emulador.

Dalvik es una máquina virtual incluida en Android que permite la ejecución de programas escritos en código byte de java. Para generar código byte de java es necesario utilizar el compilador proporcionado por Android.

Antes de comenzar con el diseño o la programación hay que decidir a qué versiones de Android irá dirigido el juego. Teniendo en cuenta que cuanta más alta sea la versión mayores van a ser las prestaciones, pero a su vez será menor el número de dispositivos compatibles lo que reducirá significativamente el número de posibles clientes.

En la siguiente tabla se puede observar la distribución de los dispositivos que accedieron a Google Play durante 15 días hasta el día 1 de junio de 2012.

Versión	Nivel API	Distribución
4.0.x *Ice Cream Sandwich*	14-15	7.1%
3.x.x *Honeycomb*	11-13	2.7%
2.3.x *Gingerbread*	9-10	65.0%
2.2 *Froyo*	8	19.1%
2.0, 2.1 *Eclair*	7	5.2%
1.6 *Donut*	4	0.6%

Versión	Nivel API	Distribución
1.5 *Cupcake*	3	0.3%

Tabla 1 – Porcentaje de dispositivos Android

Esta imagen se ha generado con los datos de la tabla anterior, aquí se ve como las versiones que están desfasadas o son han salido al mercado recientemente son las menos usadas.

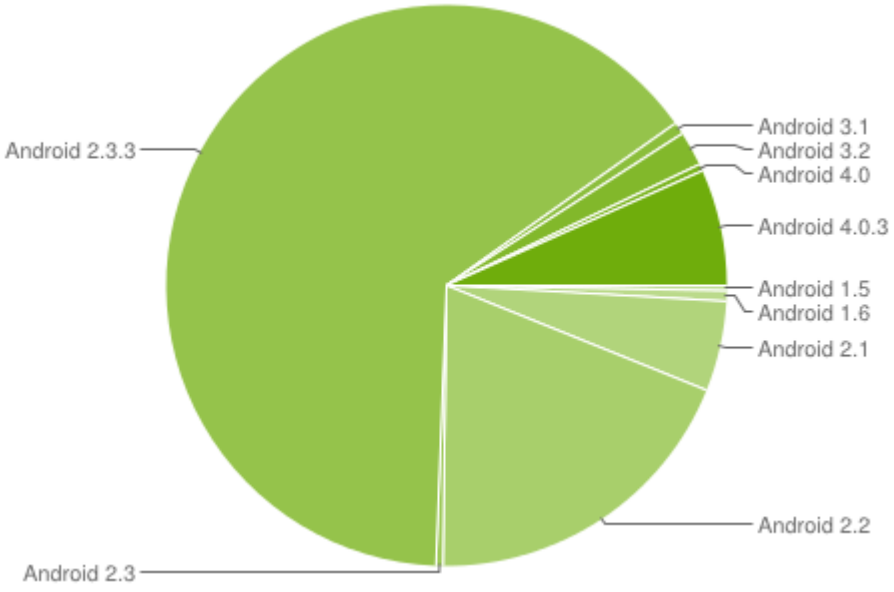

Figura 1 – Distribución de versiones de Android

En esta otra imagen se incluyen los datos de versiones de 6 meses, aquí se ve la tendencia de los dispositivos a aumentar de versión progresivamente.

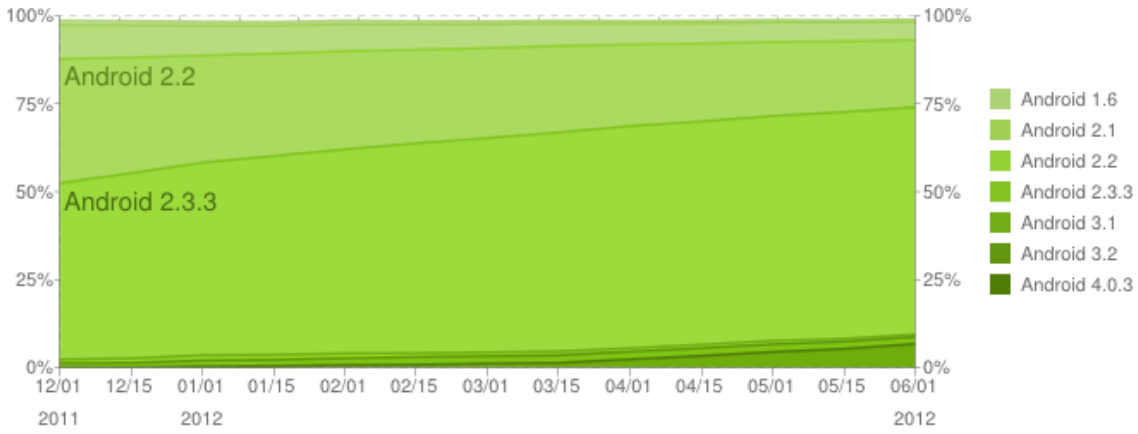

Figura 2 – Distribución de Android a lo largo de 6 meses [6]

El juego se desarrollará utilizando la versión 2.2 de Android, lo que le permitirá funcionar en la mayoría de de los dispositivos con versión 2.2 o superior, que representa un 93.9% de la población Android.

2.1.2 iOS

iOS es el sistema operativo utilizado en los móviles iPhone, a diferencia de otros sistemas operativos para smartphones, iOS no permite su integración en otro hardware que no pertenezca a Apple. Las aplicaciones de iOS se descargan a través de la tienda de Apple llamada App store, la cual dispone tanto de aplicaciones gratuitas como de pago [7].

La siguiente imagen corresponde con un móvil iPhone que tiene como sistema operativo a iOS.

Figura 3 – iPhone [8]

2.1.3 Comparación de sistemas operativos

La gran ventaja de Android frente a iOS es que desarrollar en Android es gratis mientras que inscribirse como desarrollador en iOS tiene un coste de 99$ y además Apple se lleva un 30% de cada venta en la tienda.

Otro punto a favor de Android es que tiene más cantidad de usuarios que iOS y además tiene un mayor crecimiento como se puede observar en el siguiente gráfico.

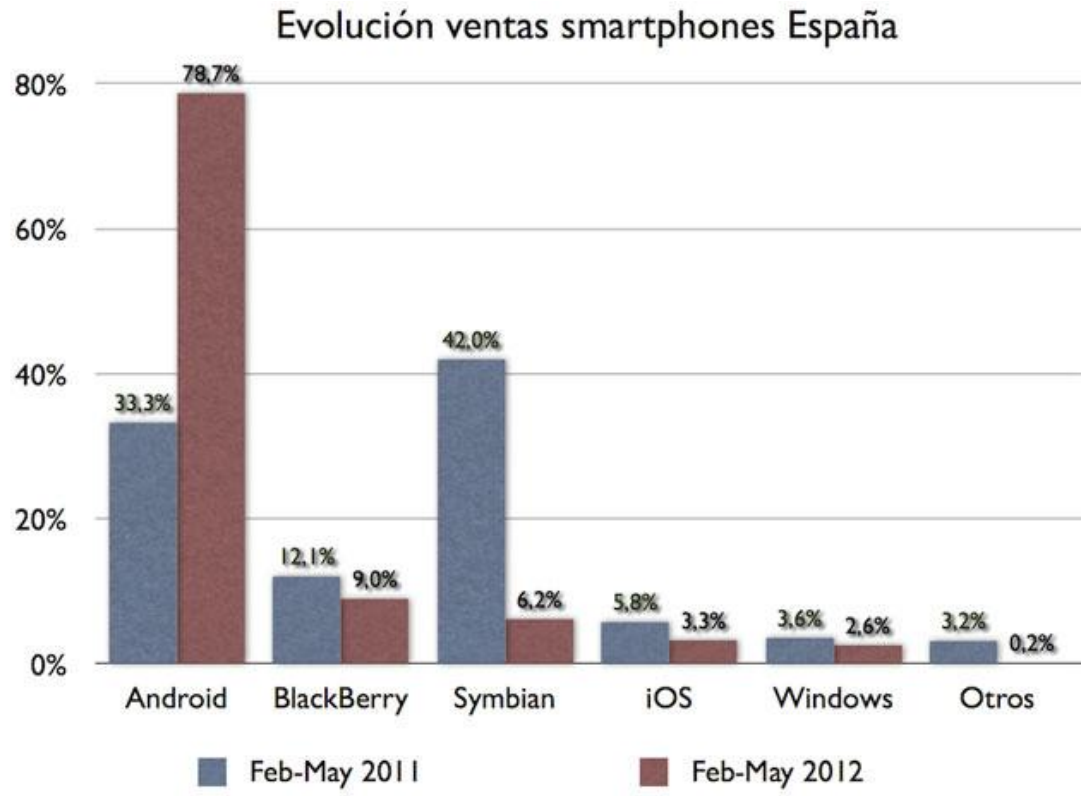

Figura 4 – Comparación de sistemas operativos [9]

Lo bueno del iPhone es que tiene unas buenas características que pueden ser interesantes a la hora de hacer pruebas de rendimiento.

Por último, ya tenía conocimiento de programación en Android por lo que no habría apenas fase de aprendizaje mientras que con iOS se necesitaría más tiempo para acostumbrarme al lenguaje.

A partir de esta comparación se ha decidido elegir Android como objetivo del proyecto.

2.2 Juegos multijugador

Un juego MMO es un tipo de juego en el que pueden participar cientos de jugadores compartiendo un mundo virtual. El juego tiene una serie de normas que son comunes para todos, como son restricciones o leyes físicas, además cada jugador tiene su propio punto de vista desde el cual puede realizar acciones que influyen en el mundo virtual.

Los juegos multijugador se pueden dividir en dos grandes categorías, juegos en primera persona y juegos con la vista alejada. Donde los juegos en primera persona requieren tener un retardo bajo pero tienen la ventaja de necesitar sincronizar pocos elementos, mientras que en los juegos con la vista alejada no es tan importante el retardo pero sí que requieren sincronizar un mayor número de elementos, la mayoría de los juegos para Android son de este último tipo [10].

A continuación se citan otros juegos ya existentes de tipo MMO para Android, todos tienen en común que son de rol, permiten interactuar entre jugadores mediante chat, comercio e incluso combates.

The world of magic

The world of magic se trata de un juego multijugador de rol en el cual se puede elegir ser parte de una de las 3 distintas clases que hay, guerrero, arquero o mago.

El juego dispone de varios servidores distribuidos por el mundo para evitar el retraso que es provocado por los enlaces de red, esto también hace que se distribuya la carga y permita un mayor número de jugadores en total aunque en realidad están separados y no existe ningún tipo de interacción entre jugadores de distintos servidores [11].

En la siguiente imagen se aprecia que admite bastante cantidad de jugadores a la vez.

Figura 5 – The world of magic

Warspear

Al igual que el anterior juego, Warspear es un juego multijugador de rol. El número de jugadores que admite es algo menor que el del juego anterior.

Este juego es gratuito aunque permite la compra de objetos especiales dentro del juego [12].

Figura 6 – Warspear

Pocket legends

Este juego en cambio utiliza gráficos en 3D por lo que el número de jugadores que admite es menor, de 25 jugadores en las zonas más grandes [13].

Figura 7 – Pocket legends

Después de analizar varios juegos se llega a la conclusión de que en general, cuanto mejores sean los gráficos, menor es la cantidad de jugadores que pueden estar conectados al mismo tiempo.

2.3 Arquitectura

Actualmente se utilizan distintos tipos de arquitecturas para la implementación de juegos multijugador online, cliente-servidor, p2p o una mezcla de ambas.

A fin de obtener un buen rendimiento hay que definir qué partes del juego van a correr en el lado del servidor y qué partes en el cliente. Si el cliente procesa más cantidad del juego, necesitará usar más cantidad de recursos del terminal y será menos seguro ya que los jugadores podrían modificar el estado de la información que se encuentra en sus móviles permitiéndoles dañar el sistema o hacer trampas.

Una arquitectura cliente-servidor o también llamada arquitectura centralizada se caracteriza en que el procesamiento y reparto de datos se realiza en un servidor o grupo de servidores. El siguiente esquema es un ejemplo de conexión entre varios clientes y un servidor los clientes son muy heterogéneos, tienen distintas características.

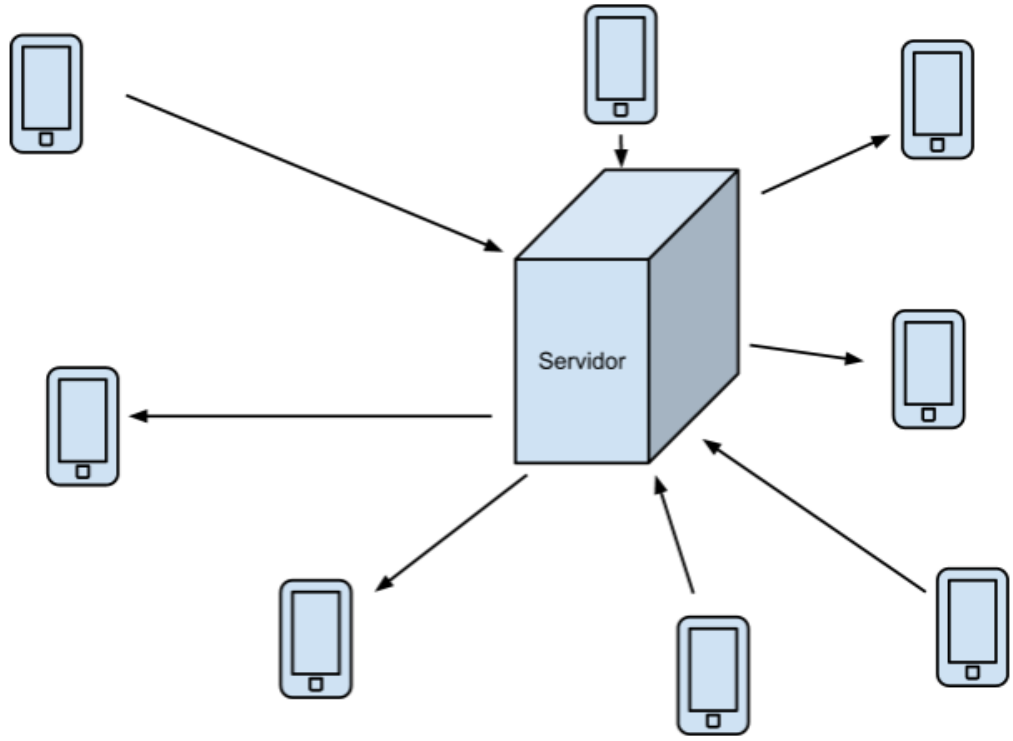

Figura 8 - Arquitectura cliente-servidor

Una arquitectura P2P o no centralizada, a diferencia de las arquitecturas centralizadas, distribuye el peso de procesamiento y control entre los distintos clientes. Al igual que lo que se puede ver en el esquema, los clientes se pueden conectar entre ellos de manera no uniforme.

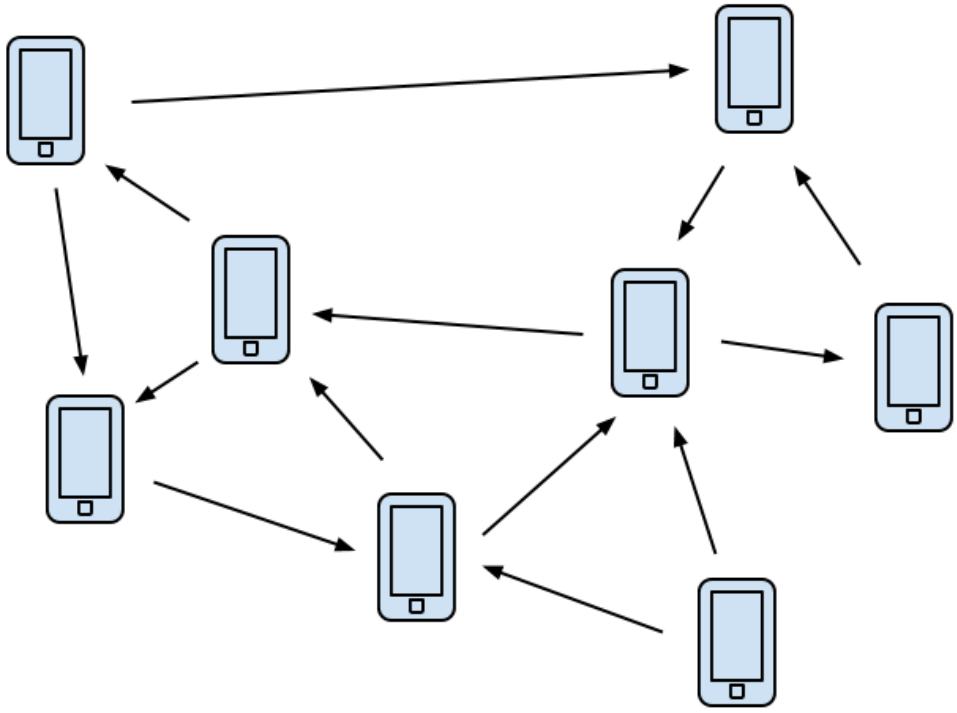

Figura 9 - Arquitectura P2P

La desventaja de una arquitectura centralizada frente a una no centralizada es que no es tan escalable y además depende de un único servidor que es más probable que falle en comparación con la modalidad P2P que tiene más redundancia [14].

Para que sea un buen juego MMO tiene que cumplir principalmente los siguientes objetivos: minimizar el tráfico de datos, proporcionar una opción para equilibrar la carga y recursos, proporcionar un entorno seguro de juego, ser escalable y de fácil mantenimiento y maximizar el rendimiento de gráficos del cliente en tiempo real [15].

- Minimizar el tráfico de datos.

El transporte de datos a través de la red es un recurso bastante limitado, sobre todo a la hora de incrementar el número de jugadores. Una buena arquitectura transmitirá solo la información que le es relevante a cada jugador, por ejemplo, en lugar de enviar el estado de todo el mapa a cada jugador se envía únicamente la zona más cercana, que es con la que va a interactuar el usuario.

Para implementar esta mejora se utiliza un sistema de subscripción, el cliente se subscribirá a los cambios que sean importantes y el servidor solo enviará los cambios a los clientes que se los pidan.

Por ejemplo, pongamos el caso de que hay un jugador conectado al servidor, y cerca de él hay varios elementos del juego (objetos, enemigos, otros jugadores…) con los que está interactuando, el servidor le estará enviando la información de esos elementos, pero no tiene por qué estar enviándole la información de otros que sea improbable que interactúe con ellos porque no estén a su alcance.

- Proporcionar una opción para equilibrar la carga y recursos

Para lograr minimizar el tráfico de datos y evitar la mala gestión de recursos, algunas partes del juego tienen que procesarse en el cliente y otras en el servidor, en el cliente irán recursos con ninguna dependencia como son los recursos multimedia, los sonidos e imágenes ya vendrán precargados en el cliente. Por otra parte, el servidor se encargará de la parte de manejo de datos persistentes y conexión

- Proporcionar un entorno seguro de juego

La confidencialidad se consigue cifrando y descifrando los mensajes que se envían entre el cliente y el servidor.

La integridad se preserva evitando que mensajes que sigan un esquema incorrecto sean aceptados, para evitar que se modifique el estado del servidor de forma no autorizada.

Lo que no tiene en cuenta Caltagirone, el autor del artículo, en su modelo de seguridad es la disponibilidad, no dice cómo se podría evitar que un atacante dañase la disponibilidad evitando que los demás usuarios puedan jugar.

- Ser escalable y de fácil mantenimiento

La escalabilidad es un punto importante, el nivel de escalabilidad es la habilidad para aumentar el número de interacciones que se manejan evitando que la calidad de juego se vea afectada. La escalabilidad depende de las especificaciones de cada sistema por lo que es imposible ofrecer una solución universal. También depende de lo que queramos conseguir, distintos tipos de juego tienen distintos requisitos. Para lograr este objetivo hay que eliminar o controlar cada uno de los cuellos de botella que aparezcan.

Un ejemplo claro de buena escalabilidad es el sistema de nombres de dominio (DNS), cuya función es la de traducir nombres a direcciones IP, a pesar de estar en constante crecimiento la complejidad y costes están bajo control.

En cambio, un ejemplo de un sistema mal escalable es el protocolo IPv4, que a pesar de que inicialmente tenía suficientes direcciones IP para cada dispositivo, el rápido crecimiento y la demanda masiva rompió este esquema que inicialmente parecía suficiente y fue necesario diseñar un nuevo protocolo.

Hay que tener en cuenta que para que un sistema sea escalable, todos sus componentes deben ser escalables, principalmente ancho de banda y CPU [16].

- Maximizar el rendimiento de los gráficos del cliente

Un juego que es más atractivo visualmente que otro se considera mejor que otro que no lo es tanto, pero esto ya no es tan claro cuando las mejoras visuales influyen en la fluidez del juego, por lo que es tan importante la apariencia como obtener un buen rendimiento.

Por ejemplo la librería AndEngine, de la que se va a hablar más adelante, da la opción de crear partículas, las partículas son grupos de pequeñas imágenes en movimiento que tienen como objetivo lograr efectos tales como fuego, humo o chispas entre otros, estas partículas le dan luminosidad y colorido a los juegos pero a costa de perder fluidez. Si el número de imágenes por segundo no es suficiente para dar la sensación de un movimiento fluido se deberá suprimir este tipo de efectos para mejorar el rendimiento.

2.4. Análisis de entornos para el desarrollo

Aquí se listan los distintos entornos o librerías que se han analizado para el desarrollo del juego, en la lista se nombran las características o funciones que incluyen más relevantes de cada una.

Con este análisis se quiere conseguir elegir la librería que más se adapte a las necesidades del proyecto en la mayor medida posible.

Una vez que se han expuesto las características de todas las librerías analizadas, se hará una comparación y se elegirá una para su posterior implementación.

2.4.1 AndEngine

La mejor característica de AndEngine es el manejo de los recursos multimedia, en concreto de los sprites, a los cuales se les pueden aplicar diversas funciones como por ejemplo para darles movimiento. Aunque también incluye la reproducción reproducir y manejo de sonidos del tipo AAC, MP3, Ogg, MIDI y WAV. También tiene un módulo para la física basado en box2D que permite crear un mundo con leyes físicas tales como aceleración, gravedad y colisión.

En cuanto a la parte de conexión, incluye una capa multijugador que utiliza TCP y sirve tanto para conexiones por internet como para bluetooth.

Otra característica que cabe destacar es que AndEngine utiliza la licencia LGPL, la cual es totalmente libre, incluso permite comerciar con el producto generado a partir de la librería [17].

La siguiente imagen es un ejemplo de un juego ya terminado que ha utilizado AndEngine durante su desarrollo, este juego en particular utiliza el módulo de física para los proyectiles y la detección de colisión.

Figura 10 – Ejemplo de juego que utiliza AndEngine [18]

2.4.2 LibGDX

LibGDX no incluye una capa dedicada a multijugador, aun así, se pueden utilizar otras librerías para completar el multijugador. Tiene la característica de que simplifica los métodos básicos de Android como son resume, pause o destroy. También permite la conexión con aplicaciones de escritorio, significa que se puede exportar a aplicación de escritorio de forma sencilla para que pueda ser ejecutado también en otras plataformas.

La librería antes usaba licencia LGPL, pero ahora utiliza apache 2.0 que es aún más libre.

Cabe destacar que libGDX es una librería muy rápida a la hora de dibujar gráficos en la pantalla, lo cual es útil para juegos 3D [19].

En cuanto al manejo de mapas, es capaz de generar mapas tanto isométricos como en dos dimensiones.

La siguiente imagen corresponde a un juego que se ha creado utilizando libGDX, se puede ver que se ha utilizado la librería para generar el mapa y para la animación del personaje.

Figura 11 – Ejemplo de juego que utiliza libGDX [20]

2.4.3 Slick

Es una librería la cual en un principio estaba hecha para ordenador pero se portó a Android, por lo que la librería utiliza un emulador, reduciendo su rendimiento. Utiliza como base la librería de java para juegos 2D llamada LWJGL. También es capaz de procesar y transformar imágenes de tipo PNG, GIF, JPG y TGA, al igual utilizar sonidos WAC, Ogg y XM. Tiene un controlador de eventos para facilitar la creación de reglas para el juego [21].

Esta imagen corresponde a un juego sencillo donde se utiliza Slick.

Figura 12 – Ejemplo de juego que utiliza Slick [22]

2.4.4 Skiller

La función principal de Skiller es la de añadir una capa social a los juegos, a su vez es capaz de integrarse en el IDE eclipse.

La gran desventaja de esta librería es que en la capa multijugador, solo admite como máximo dos jugadores y por turnos. Pero ofrece otras funcionalidades como la creación de tablas de puntuaciones, creación de perfiles personales y el comercio de objetos.

Es posible integrar las notificaciones que salgan del juego con Facebook y Twitter. También se puede tener una lista de amigos y avatares personalizados para cada jugador.

Es escalable, lo que implica que la cantidad de jugadores puede aumentar sin que el rendimiento se vea afectado [23].

Como se puede ver en la imagen, Skiller solo sirve para juegos por turnos.

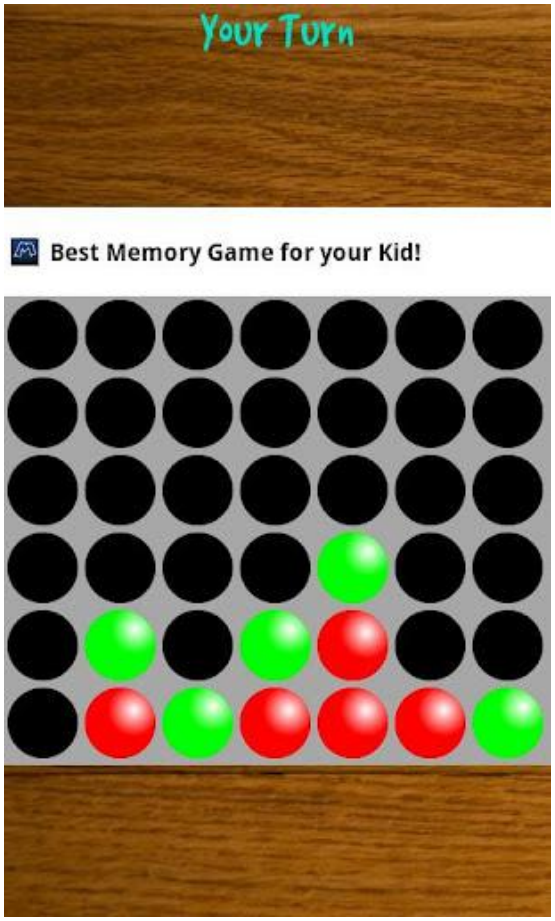

Figura 13 – Ejemplo de juego que utiliza Skiller [24]

2.4.5 Roarengine

Roarengine es una librería que solo implementa la capa social, la programación es visual, por lo que no hay que escribir código salvo para llamar a la librería desde el juego.

Para ver un estado en tiempo real del juego es capaz de realizar reportes, métricas y gráficas [25].

La imagen corresponde con una captura de pantalla del IDE de Roarengine, se observa que la lista de virtual goods, son los objetos que se han creado y a la derecha de la interfaz se pueden cambiar sus propiedades.

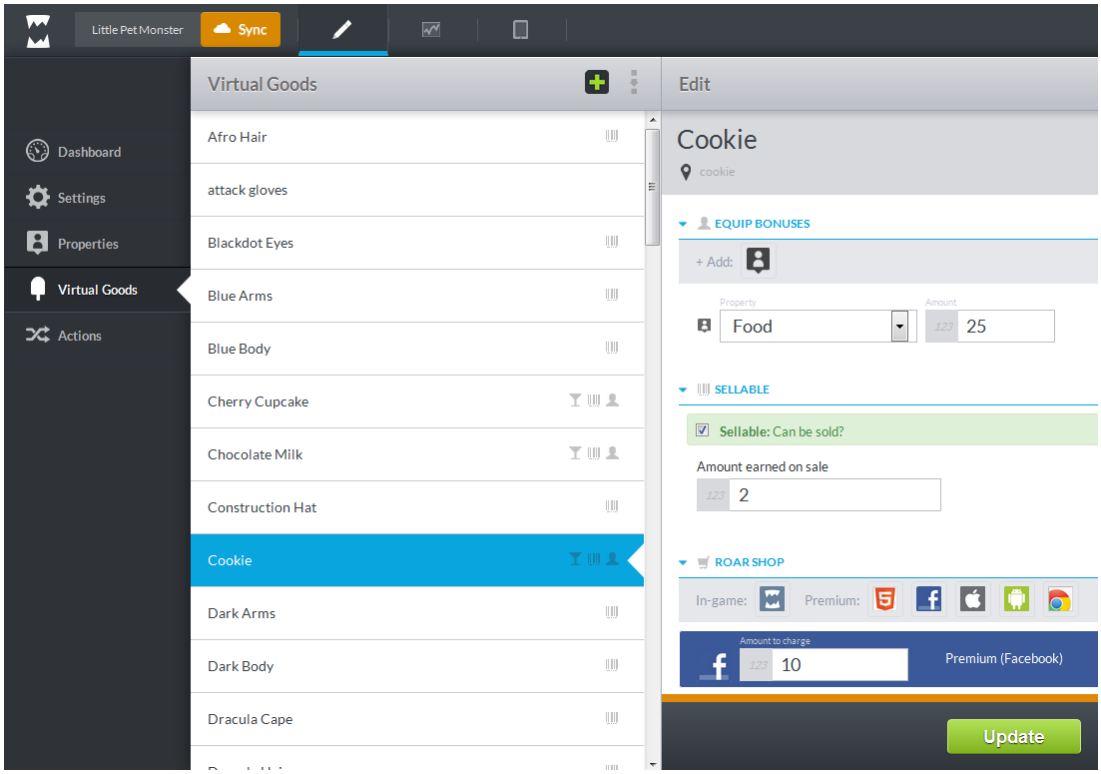

Figura 14 – Entorno de desarrollo de RoarEngine [26]

2.4.6 Corona

Corona, al igual que la mayoría de librerías ofrece ayuda a la hora de la utilización de sprites. También facilita el uso de las características de Android como la cámara, el GPS, el acelerómetro y teclados [27].

La imagen corresponde a un juego hecho con Corona, en ella se ve como el juego trata de pulsar en la pantalla para evitar que los enemigos lleguen hasta

las ovejas. El evento de pulsado de pantalla se ha manejado utilizando el módulo del sensor de pantalla que incluye la librería.

Figura 15 – Ejemplo de juego que utiliza Corona [28]

2.4.7 Marmalade

El desarrollo en Marmalade se hace en C++, esta librería tiene la característica de ser compatible con muchas plataformas. Permite utilizar gráficos y animaciones 3D.

Al igual que la librería AndEngine incluye un módulo para la simulación de la física [29].

En esta imagen se ve un juego que se ha portado a otras plataformas gracias a que ha utilizado Marmalade.

Figura 16 – Ejemplo de juego que utiliza Marmalade [30]

2.4.8 Unity

Unity es un motor 3D cuyo desarrollo se hace en tiempo real con el IDE que viene integrado, se puede ver el producto final mientras se va editando. Es un buen entorno en cuanto a gráficos porque incorpora mejoras y optimizaciones gráficas como por ejemplo la ocultación de texturas no visibles.

La desventaja de esta herramienta es que su precio es de 400$ y su versión gratuita no tiene todas las funcionalidades [31].

En la imagen se ve como a pesar de ser un juego para dispositivos móviles incluye unos gráficos detallados.

Figura 17 – Ejemplo de juego que utiliza Unity [32]

2.4.9 PubNub

PubNub es una buena librería para la comunicación entre muchos clientes debido a que incorpora un módulo API para una comunicación masiva, que además utiliza mensajes asegurados con SSL. Al igual que Marmalade es también una librería multiplataforma.

Esta librería tiene la ventaja de ser escalable pero esta escalabilidad tiene un precio que depende de la cantidad de paquetes de información que se envíen entre sí los jugadores [33].

En esta imagen se ve como la librería permite la creación de juegos multijugador para varias plataformas como Android y Windows.

Figura 18 – Ejemplo de juego que utiliza PubNub [34]

2.4.10 Comparación de librerías

Para una mejor comparación de las librerías se han reunido en una tabla con las siguientes columnas:

Entorno

Es la primera columna de la tabla e indica que librería se está analizando.

Capas

Indica en que partes de la estructura del juego está presente la librería. Las librerías que se han analizado están presentes en 3 niveles, recursos, conexión multijugador y capa social.

En el siguiente esquema se observa con mayor detalle la organización de las capas:

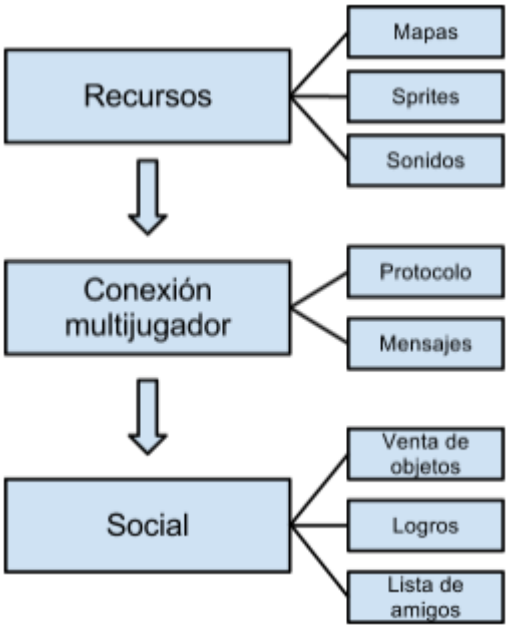

Figura 19 – Capas

La capa más baja, la de recursos, proporciona ayuda en el momento de manejar recursos multimedia.

La siguiente capa, conexión multijugador, ofrece protocolos de fácil manejo para la comunicación entre clientes, puede darse el caso de que tenga limitaciones como en el caso de la librería Skiller que solo permite juegos multijugador por turnos.

La capa social abre las puertas a las redes sociales incluyendo utilidades como logros, tablones de puntuación o listas de amigos.

Lenguaje:

Son los lenguajes de programación soportados por la librería. Algunas librerías disponen de su propio lenguaje de programación.

Mapas:

Indica si la librería facilita el creado y uso de mapas.

Multijugador:

Indica si la librería incluye soporte multijugador.

Coste:

Es el precio por usar la librería, o indica qué limitaciones tiene en el caso de que disponga de versión gratuita.

Entorno	Capas	Lenguaje	Mapas	Multijugador	Coste
Andengine	Todas	Java	Sí	Sí	Gratis
libGDX	Todas	Java	Sí	No	Gratis
Slick	Todas	Java	Sí	No	Gratis
Skiller	Multijugador y social	Java	No	Por turnos	Gratis
Roarengine	Social	Propio	No	Sí	Gratis hasta 100 logins/dia
Corona	Multijugador	Lua	No	Sí	349$
Marmalade	Todas	Html5, C++	Sí	Sí	149$ la versión completa
Unity	Todas	Propio, C#	Sí	Sí	Funcionalidades básicas gratuitas 400$ la versión completa
PubNub	Multijugador	C#, Python, .NET, PHP	No	Sí	15$ al mes y 1$ por cada millón de mensajes

Tabla 2 – Comparación de librerías

Se ha empezado descartando la librería Skiller cuya capa multijugador no sirve para juegos en tiempo real, sino que solo permite juegos por turnos.

Después se han descartado las librerías que son de pago, que son: Roarengine, Corona, Marmalade, Unity y PubNub.

Tras buscar manuales o referencias de cómo usar cada una, se ha descartado la librería Slick, porque carece de una comunidad donde buscar información y ayuda, afectando negativamente a la velocidad de desarrollo.

Por último, con el objetivo de elegir una de las dos librerías restantes, AndEngine y libGDX, se ha decidido comparar el rendimiento entre ambas.

Para las pruebas de comparación de rendimiento se utilizó un móvil modelo Huawei u8230 con las siguientes características:

- 528 MHz de velocidad de procesador
- 256 MB de memoria RAM
- 180 MB de memoria interna
- Android V2.1

Figura 20 - Huawei u8230 [35]

En la siguiente tabla están expuestos los resultados, la primera fila, la de Android SDK, corresponde con el método por defecto que Android ofrece para manejar imágenes. La segunda fila son las pruebas que se han hecho utilizando AndEngine y la última fila son las pruebas con libGDX.

Se puede ver como se pierden bastantes imágenes por segundo en Android SDK y Andengine a partir de 300 sprites.

En esta tabla no se ha tenido en cuenta la optimización que incluye AndEngine llamada spriteBatch, según el autor con esta optimización se dobla el rendimiento a costa de perder la capacidad de poder cambiarle el nivel de transparencia a cada imagen de forma individual.

Librería	50 Sprites	150 Sprites	300 Sprites	500 Sprites
Android SDK	29	25	11	4
AndEngine	29	27	16	10

libGDX	30	29	29	24

Tabla 3 – Android SDK, AndEngine y libGDX [36]

A pesar de que libGDX tiene un mejor rendimiento a la hora de dibujar sprites, se ha decidido utilizar AndEngine porque viene incluida la capa multijugador, tiene una comunidad mayor, recibe actualizaciones muy a menudo y tiene un ejemplo de juego para cada funcionalidad.

Profundizando en AndEngine se encontró que también dispone de un módulo para el desarrollo de un servidor en java que puede ser ejecutado en cualquier ordenador, no tiene por qué estar en un móvil.

Además AndEngine usa licencia LGPL, lo cual permite su uso, modificación y distribución de cualquier tipo.

La librería AndEngine permite importar mapas definidos mediante XML, los cuales se pueden crear utilizando la herramienta Tiled Map Editor [37].

Esta herramienta facilita la creación de mapas, se pueden crear los mapas de forma gráfica, basta con ir rellenando cada celda con los objetos que hemos cargado en la paleta. Además admite la implantación de scripts para conseguir generar un mapa de forma automática

La siguiente imagen corresponde a la ventana principal de la herramienta, donde se realizarán la mayoría de las tareas de creación de mapas.

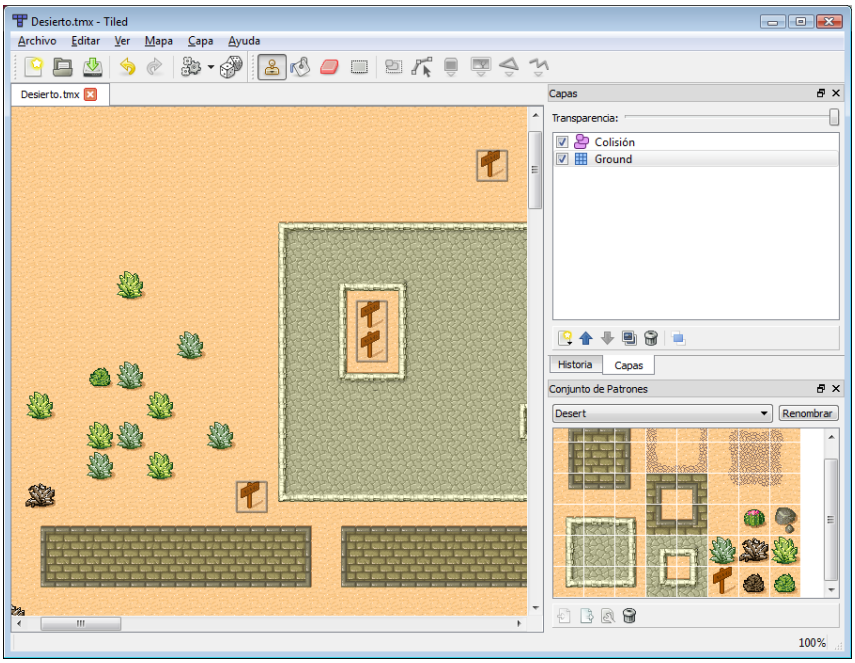

Figura 21 – Tiled Map Editor

Otra característica de la herramienta es que el máximo número de celdas que permite crear la herramienta es de 9999 x 9999, por lo tanto, el tamaño máximo que tendrá un mapa es de 99980001 celdas.

3. Diseño y arquitectura

Se ha desarrollado una base para el juego utilizando el motor previamente elegido, AndEngine. Con esta base se pretende resolver ciertas dudas acerca de los límites de los dispositivos móviles.

El desarrollo del juego se divide en dos partes, cliente y servidor. La parte del cliente va instalado en la aplicación móvil de cada usuario mientras que la parte del servidor se encuentra un único ordenador, ambas partes utilizan AndEngine y también deben tener acceso a internet para funcionar.

El servidor estará esperando a que cualquier cliente se conecte para comenzar el envío de datos, en el apartado de mensajes se definirán qué tipo de datos se pueden intercambiar entre ellos.

El jugador es representado mediante un personaje, y a su vez cada personaje tiene sus propios atributos como son, nombre, posición, dirección, vida y un número de identificación.

A cada personaje se le asigna un número de identificación o ID al establecer conexión, este número sirve para que el cliente y el servidor puedan sincronizarse, por ejemplo, el cliente con ID 5 le dice al servidor que su personaje se está moviendo, entonces el servidor le dice a los demás jugadores que muevan el personaje con ID 5, cada personaje tiene una id que es la misma en el servidor y cada cliente.

3.1 Cliente

El siguiente esquema es un resumen de las partes que componen el cliente, los recuadros de color azul son extensiones o módulos que incluye AndEngine y la parte de color verde son las que se han definido para el desarrollo del juego.

De la librería AndEngine se utilizan básicamente 3 partes, la extensión multijugador, que sirve para definir los mensajes que se utilizarán en la comunicación, la extensión de mapas, que sirve para generar el mapa TMX y el paquete básico de AndEngine que incluye el manejo de recursos tales como sprites.

Por otro lado, en la parte que se ha desarrollado se incluye la comunicación, aquí se han definido qué mensajes son necesarios, incluyendo la plantilla de mensaje del servidor para que pueda entender mensajes entrantes.

El cliente tiene definido lo que es un personaje, dice qué atributos tiene, como por ejemplo posición y nombre, a partir de ahí se definen lo que es un jugador y lo que es un bot

La parte central del cliente es la actividad, desde ahí se hacen las llamadas a Android para reunir todos los elementos y poder mostrárselos al usuario.

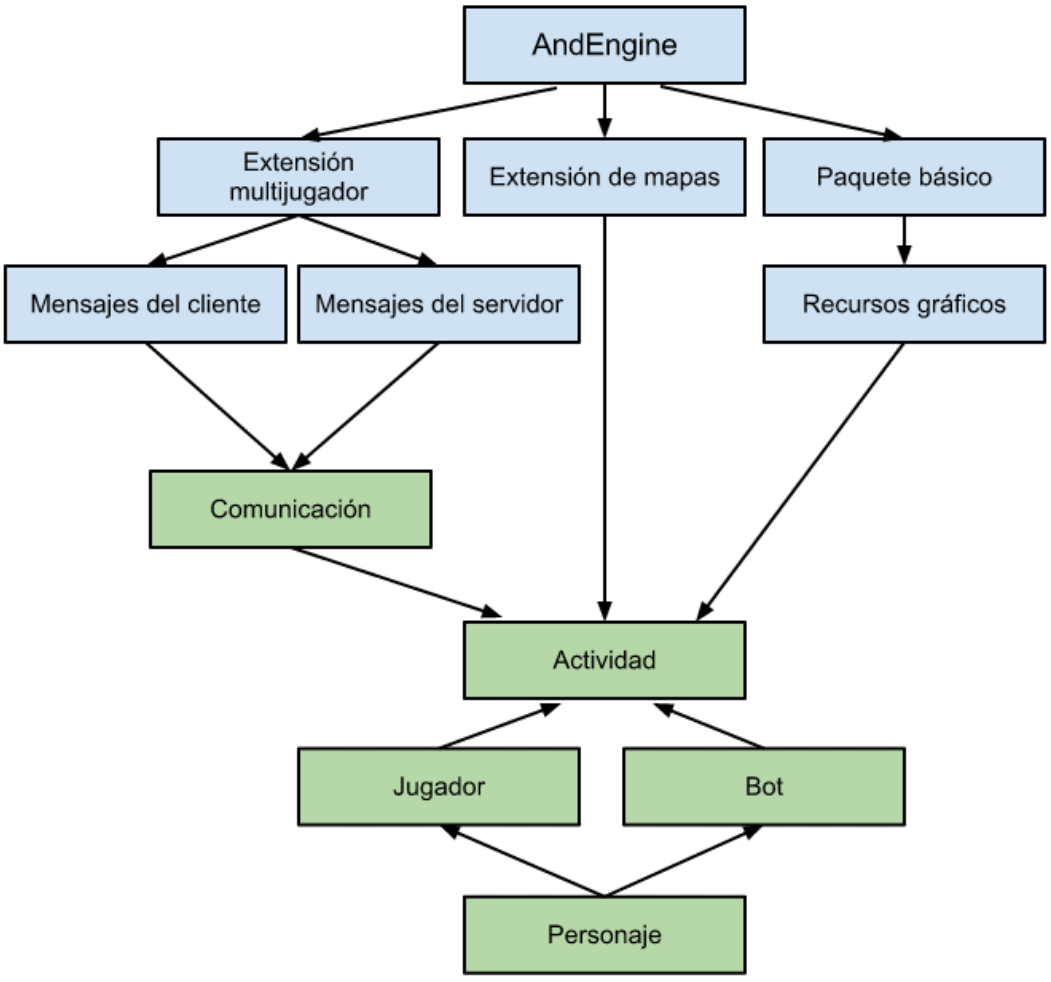

Figura 22 – Estructura del cliente

Lo primero que hace el cliente es cargar el mapa y las texturas que va a utilizar, para ello crea un hilo que carga los recursos en segundo plano mientras se muestra una pantalla de carga. El hecho de mostrar una pantalla de carga produce la sensación de que el juego tarda menos en cargar.

Una vez que se ha cargado el juego se muestra el escenario con el personaje del jugador, al ser una versión base del juego todos los jugadores son representados con el mismo sprite.

El juego se puede iniciar sin conexión porque no intenta realizarla hasta que no se le indique, está configurado así para facilitar las pruebas. Una vez que se pulsa el botón de menú del móvil se cambia la pantalla a la de configuración.

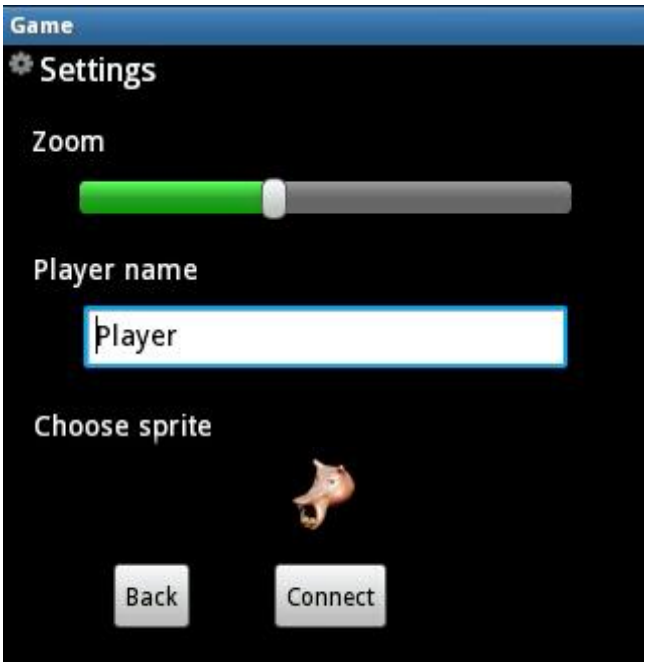

Figura 23 – Menú de configuración del juego

Actualmente el juego permite cambiar el nombre del jugador y el tamaño de la pantalla mediante la barra deslizable de zoom, el nombre del personaje será mostrado a los demás jugadores.

Al pulsar el botón de Connect en el menú de configuración el cliente intentará establecer una conexión con el servidor mandándole su información, entre la información enviada se encuentra el nombre del personaje y su posición. El servidor le responde con la lista de jugadores que ya estaban conectados, en esa lista va incluido el nombre de cada jugador y su posición para que el usuario pueda dibujarlos en el escenario.

En la imagen se puede ver cómo queda la vista después de conectarse con el servidor y recibir la lista de jugadores, en este caso los jugadores son bots simulados por el servidor.

Figura 24 – Escenario del juego

En la imagen de arriba también se puede ver cuánto mide cada celda del mapa en relación con el tamaño del jugador, lo que está resaltado en rojo en la parte central de la pantalla corresponde a una única celda.

El personaje tiene la capacidad para moverse, para ello el jugador tiene que pulsar en la pantalla y su personaje caminará hasta esa posición. Nada más pulsar en el escenario, el programa cliente le mandará un mensaje al servidor indicándole que se está moviendo a hacia esa posición.

Con AndEngine la implementación del movimiento del personaje ha sido más sencilla que si se hubiese hecho sin ayuda, para conseguirlo, se carga el sprite indicándole el tamaño de cada postura del personaje, para que cuando la librería lo anime pueda diferenciar entre distintas posturas. En la imagen se ve como es el fichero del sprite utilizado en el juego, este contiene todas las posturas que es capaz de hacer el personaje.

Figura 25 – Sprite del jugador

La implementación del nombre y vida del jugador se ha conseguido adjuntando dos textos encima del sprite del personaje, uno para el nombre y otro para la vida, AndEngine automáticamente los moverá a la vez que el sprite ya que se han ligado.

3.2 Servidor

La estructura del servidor es parecida a la del cliente salvo que no tiene parte gráfica ni de manejo de mapa y que en lugar de utilizar una actividad el módulo principal es una clase normal de Java que crea a los personajes y reserva memoria para los mensajes de red.

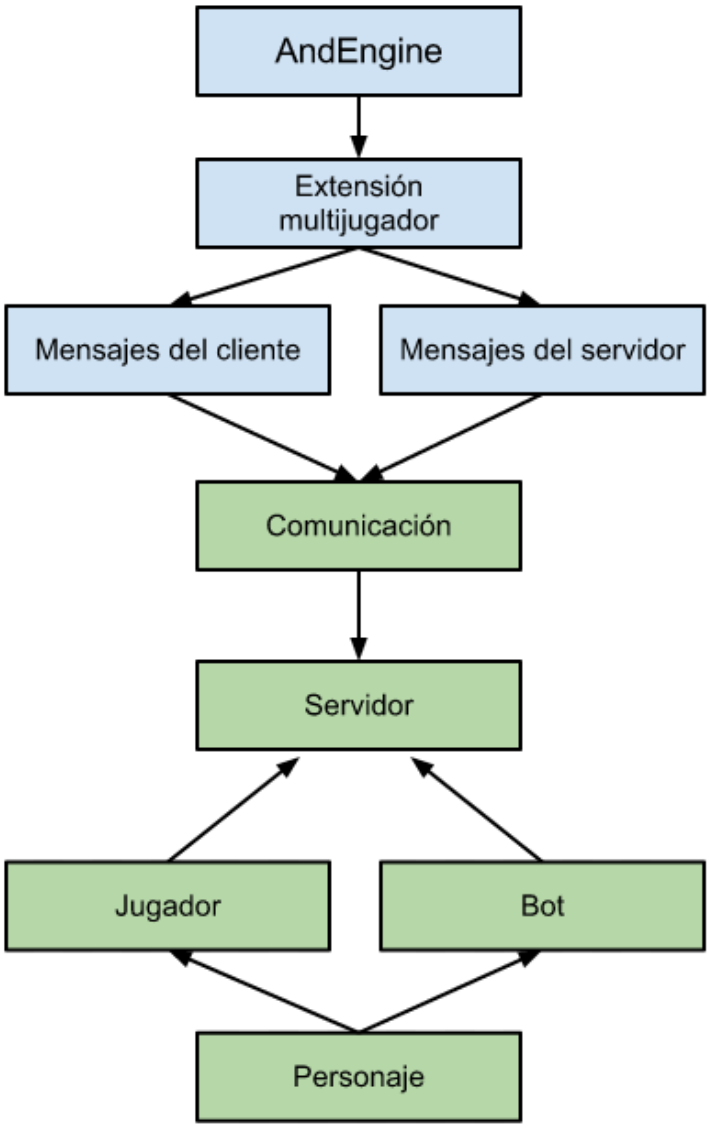

Figura 26 – Estructura del servidor

El servidor actúa de forma pasiva, significa que está en ejecución esperando a que un cliente se conecte, cuando esto ocurre el servidor informa a todos los demás jugadores conectados de que se ha conectado un jugador nuevo.

Cada vez que ocurre un evento se muestra por consola indicando la hora y los minutos a la que ha ocurrido con el objetivo de encontrar fallos o tener constancia de lo que está ocurriendo ya que el servidor no tiene por qué tener una representación gráfica de lo que está ocurriendo.

El servidor también tiene definido que es un personaje, sabe que atributos tiene que tener, deben ser los mismos que los que están definidos en la parte del cliente para una correcta sincronización.

La siguiente imagen es una captura de pantalla del servidor en marcha, la primera línea indica en qué puerto se ha iniciado la escucha, la segunda y tercera línea indican que se han reservado los mensajes en el almacén, las últimas dos líneas indican que se ha conectado un jugador desde esa dirección IP, con el nombre “Player” y utilizando la id número 5.

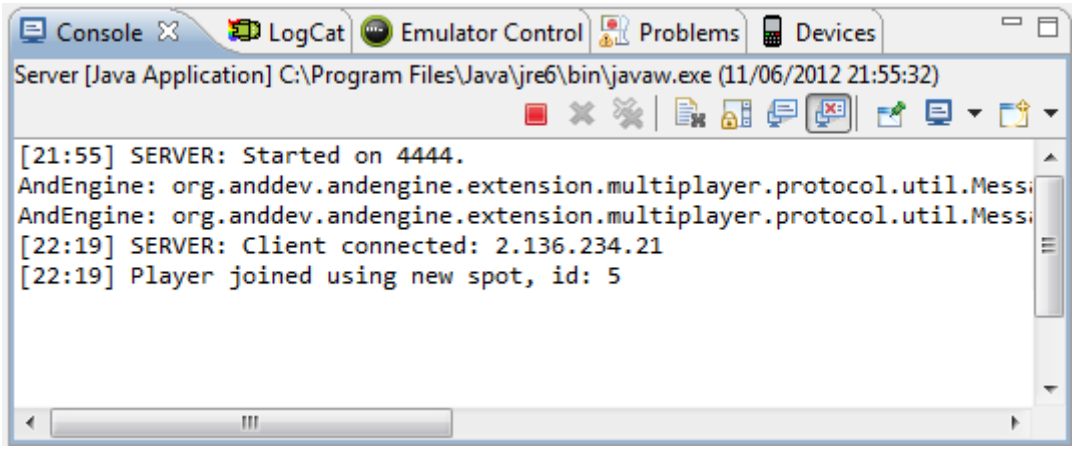

Figura 27 – Consola del servidor

3.3 Conexión

El diseño propuesto consiste en un modelo cliente-servidor en el que el servidor está compuesto por un ordenador con conexión a internet

La capa multijugador de AndEngine utiliza el protocolo de red TCP, esto puede no que no sea la mejor opción para mandar muchos mensajes ya que en cada conexión se mandan más cantidad de mensajes que si se utilizase UDP, pero TCP otorga fiabilidad. En el caso de que se utilizase UDP y el cliente al moverse por alguna razón no logra mandar el mensaje de movimiento, el

cliente que ha intentado mandar el mensaje verá que su personaje se ha movido pero los demás clientes no porque no les ha llegado el mensaje, en cambio TCP reenviaría el mensaje hasta conseguirlo.

En este esquema se describe cómo se realiza la conexión, AndEngine utiliza sockets de java utilizando el protocolo de transporte TCP sobre IP, en el caso del dispositivo móvil puede conectarse a internet o bien mediante WiFi o bien utilizando una conexión 3G.

En el esquema aunque solo se represente un cliente, puede haber más conectados a internet.

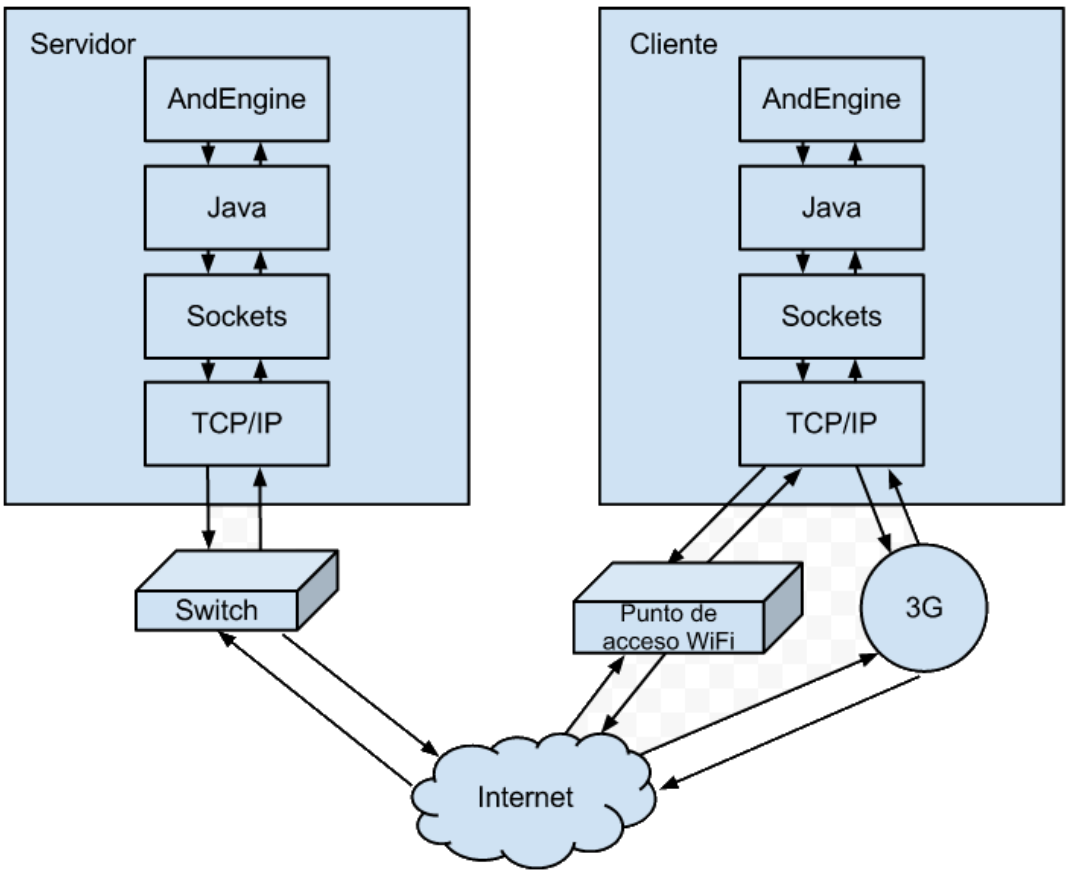

Figura 28 – Esquema cliente-servidor

3.3.1 Mensajes

Se diferencias dos tipos de mensajes, los de tipo cliente, que son los que el cliente envía y los de tipo servidor, que como su nombre indica son los que envía el servidor.

Los mensajes que hay implementados para ambos tipos son de unión, de movimiento o de abandono.

En el siguiente esquema se pone un ejemplo en el que se utilizan los 3 mensajes, en él, un jugador se conecta, se mueve y luego se desconecta.

Después de establecerse la conexión mediante TCP, el cliente le manda al servidor un mensaje de que se ha unido incluyendo su nombre de jugador, en este caso “Player”, el servidor mira en la lista de jugadores y le asigna una ID al nuevo jugador, le responde al cliente diciéndole cual es su ID y a su vez le avisa a los demás clientes de que se ha unido un nuevo jugador, en el momento que los demás clientes reciban el mensaje de que se ha unido un nuevo jugador, lo dibujarán en sus pantallas.

Una vez realizada la conexión y mandados los mensajes de unión, cada vez que se mueva el jugador el servidor replicará el mensaje a los demás clientes para que cada uno represente al jugador moviéndose, hasta que el cliente decida irse en cuyo caso enviará un mensaje de tipo abandono para que el servidor lo reenvíe y los demás clientes puedan borrarlo de sus pantallas, en el caso de que el cliente pierda la conexión, el servidor se dará cuenta porque el cliente no le responde y por consiguiente le dirá a los demás clientes que se ha desconectado.

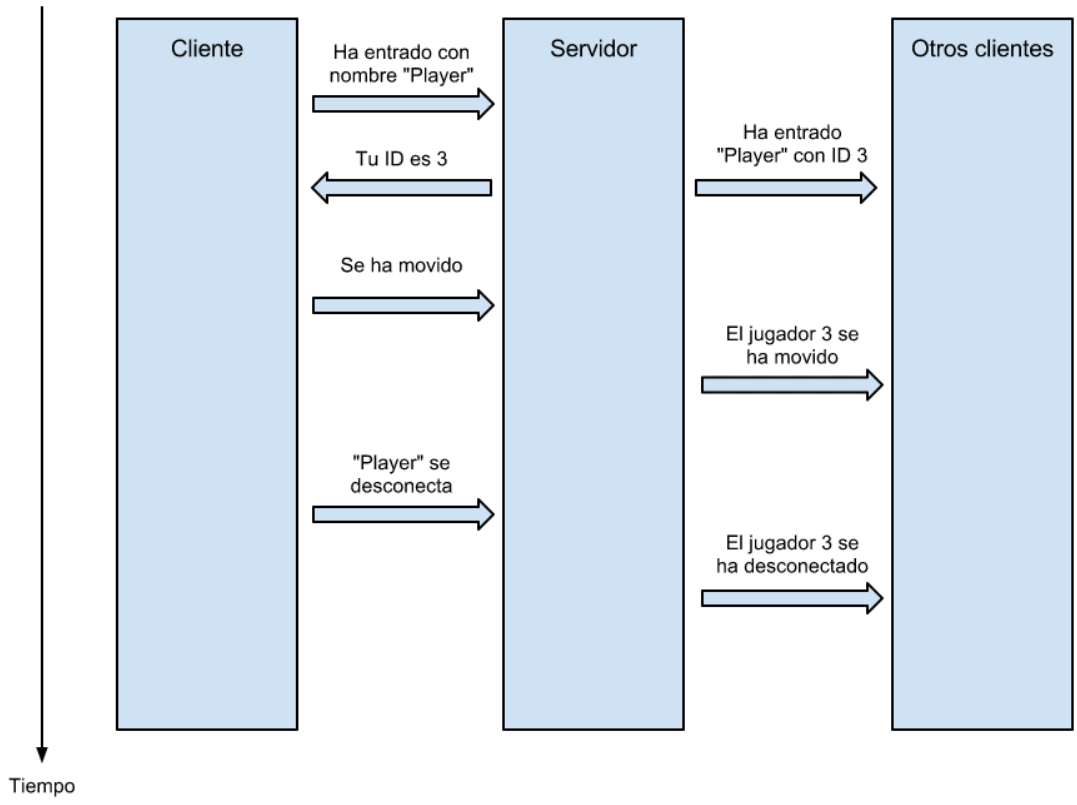

Figura 29 – Esquema de mensajes

3.4 Optimizaciones

A partir de las pruebas de rendimiento de las que se hablará más adelante se concluye que el límite de jugadores esta en entre 200 y 275 jugadores para un dispositivo móvil de gama baja-media,

Dado que el recurso que antes se agota es la capacidad del dispositivo para dibujar imágenes se han propuesto una serie de optimizaciones que evitarán que se dibujen otros jugadores que no le son relevantes al cliente, por ejemplo los que están fuera de su alcanze. Esto se puede conseguir de dos formas que se explican a continuacion.

La primera solución consiste en que el cliente aunque reciba mensajes de que otros jugadores se están uniendo o de que se están moviendo ignora la mayoría, haciendo caso solo a los que estén a la vista. Para saber que jugadores están a la vista, se calcula la distancia a la que están, para ello se calcúla el módulo del vector distancia que es la diferencia entre la posición del otro jugador y la del jugador actual.

Con esto se consigue evitar que se reduzcan las imágenes por segundo cuando hay muchos jugadores distribuidos por el mapa, en cambio no será efectivo cuando todos estén muy cerca del propio jugador. Pero la optimización solo reduce el impacto en las imágenes por segundo, aún así se alcanza el límite del ancho de banda al aumentar el número de jugadores por lo que se propona otra versión de esta optimización.

Por otro lado, se propone una mejora de la anterior optimización en la que el cliente si que dibuja todos los jugadores que recibe, pero el servidor solo le envía los que están más cerca, funciona de la siguiente manera.

El servidor tiene una lista por cada jugador en la que apunta que otros jugadores le son relevantes, cuando un jugador se acerca lo suficiente, es agregado a la lista del otro jugador y cuando se aleja, se borra. A partir de esta lista cuando un jugador le manda al servidor un mensaje de tipo movimiento, el servidor comprueba que jugadores le tienen agregado, y en consecuencia, les replica ese mensaje solo a los clientes a los que les es relevante ese mensaje, reduciendo notablemente la carga en la red.

Este método tiene la desventaja de que en comparación con la primera optimización, el servidor tiene que realizar comprobaciones y calculos de distancias constantemente y esto aumenta el porcentaje de CPU que utiliza, aunque dependiendo de la precisión que se requiera se puede reducir este efecto.

4. Evaluación

Con el fin de conocer los límites de rendimiento a los que puede llegar un juego multijugador en Android se han realizado una serie de pruebas analizándose los recursos del dispositivo.

Las pruebas de rendimiento han sido realizadas en un Samsung Galaxy Mini S5570 con las siguientes características:

- 600 MHz de velocidad de procesador
- 280 MB de memoria RAM
- 180 MB de memoria interna
- Android V2.2.1
- Pantalla multitactil

Figura 30 - Samsung Galaxy Mini [38]

Al diseñar un juego se tiene que tener en cuenta tanto las características que influyen en el rendimiento como las que no. Entre las características que si influyen en el rendimiento se encuentran, el tiempo de carga, el número de imágenes por segundo y el ancho de banda utilizado.

El tiempo de carga es el tiempo que tarda en iniciarse el juego, dependerá de la velocidad a la que se extraen los datos de la memoria secundaria para meterlos en la memoria principal. Lo que más tarda en cargar es el mapa, por lo que el tiempo de carga está influido directamente por su tamaño.

El número de imágenes por segundo que el dispositivo es capaz de mostrar limitado por la frecuencia de la pantalla, a mayor número de imágenes por segundo mayor es la fluidez del juego. Siempre que se van añadiendo nuevas

características al juego hay que tener en cuenta el impacto que tendrán en las FPS, si es un impacto notable la característica no podrá ser incluida en el juego. En general las FPS limitan la cantidad de elementos gráficos que se pueden dibujar en la pantalla de forma concurrente.

El ancho de banda que utiliza un dispositivo es la cantidad máxima de bytes que es capaz de enviar, dependiendo del ancho de banda disponible se puede enviar más cantidad de información o menos en el mismo periodo de tiempo. Hay que

Por otro lado, entre las características que no influyen en el rendimiento como son el espacio físico utilizado y el consumo de batería.

El espacio físico es lo que ocupa el juego dentro del móvil en MegaBytes, el espacio es importante ya que puede influir en si una persona podrá instalar nuestro juego o no, sobre todo si sus terminales disponen de poca memoria.

El consumo de batería mide cuanto porcentaje de batería se ha gastado mientras se ejecutaba la aplicación, en general, el consumo es directamente proporcional a la cantidad de recursos utilizados. Por lo que si aumenta la cantidad de CPU o ancho de banda que utiliza el juego se gastará la batería más rápidamente.

4.1.1 Tiempo de carga

El dispositivo con el que se han realizado las pruebas no permite el uso de mapas con más de un millón de celdas, al intentar cargar un mapa de mayor tamaño da error porque se sobrepasan los recursos disponibles y no se completará la carga.

Tamaño del mapa	Número de celdas	Tiempo de carga en segundos
100x100	10 000	2,1
200x200	40 000	7,1
400x400	160 000	27,4
800x800	640 000	99,1
1000x1000	1 000 000	156,6
1200x1200	1 440 000	Error
1500x1500	2 250 000	Error
2000x2000	4 000 000	Error

Tabla 4 – Tiempo de carga

Esta imagen muestra lo que tarda en cargar en función del tamaño, se puede observar que la función que define el tiempo de carga es lineal, en consecuencia se podría afirmar que el mapa es escalable.

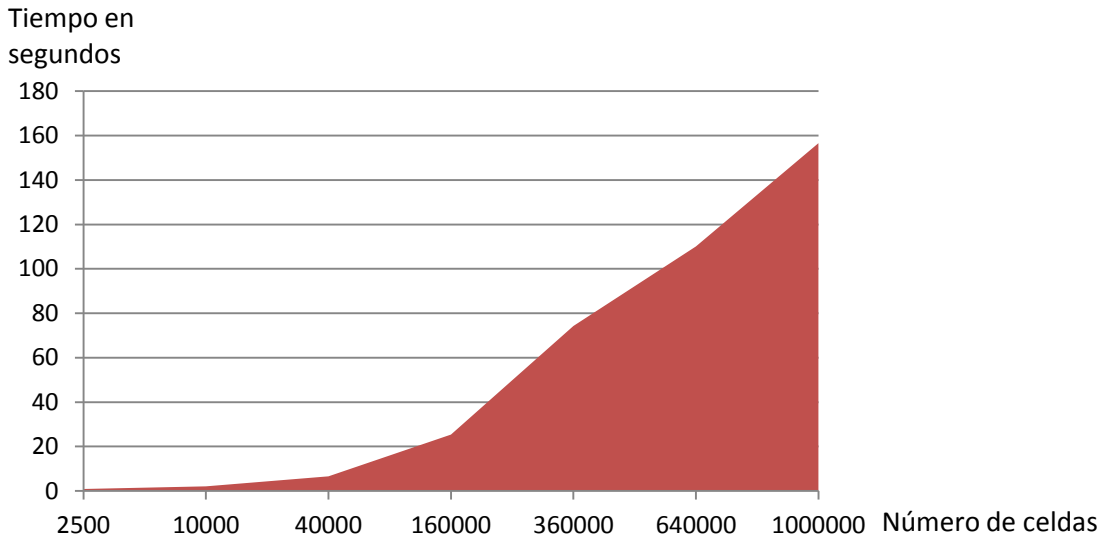

Figura 31 – Tiempo de carga

Cabe destacar que la fluidez del juego, lo que se va a explicar en el próximo punto, no se ve afectado por el tamaño del mapa, pero sí por el número de celdas a la vista. Si se aleja la cámara hacia el cielo, aumenta el número de celdas dibujadas en la pantalla y en consecuencia se reduce la fluidez del juego.

4.1.2 Imágenes por segundo

Es necesario hacer pruebas con el número de imágenes por segundo para llegar a conocer cuál es el número máximo de elementos que se pueden dibujar de forma concurrente.

El juego se considera fluido si el número de imágenes por segundo se mantiene entre 20 y 30 por lo que actualmente el móvil con el que se han hecho las pruebas admitiría entre 200 y 275 jugadores sin perder fluidez.

Se ha realizado una serie de pruebas en las que se va aumentando el número de bots mientras se miden las FPS. Se han hecho dos tipos de pruebas para que además de medir la fluidez del juego, ver si se ve influido por el hecho de ser mostrados o no en pantalla, en el primer tipo de pruebas todos los bots estaban dentro de la pantalla (línea azul en la gráfica), mientras que en

las otra pruebas todos los bots estaban fuera de la pantalla (línea roja en la gráfica).

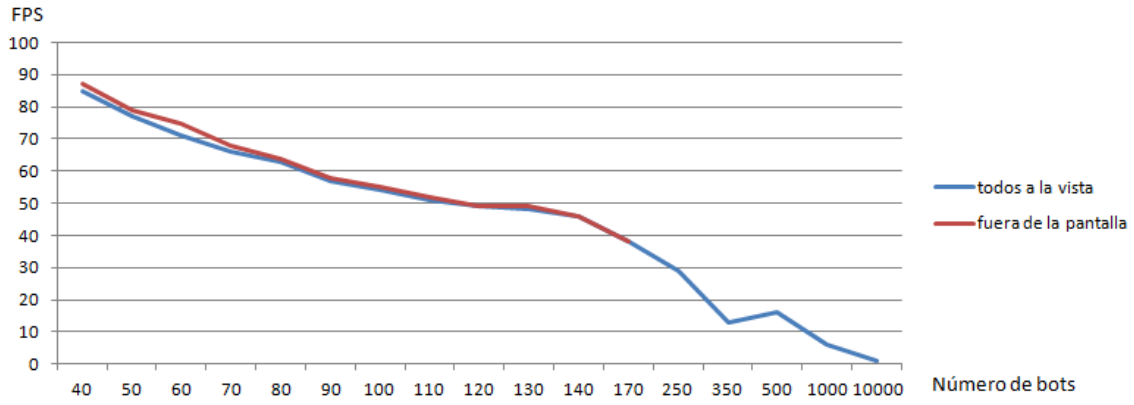

Figura 32 – Imágenes por segundo

A partir del gráfico se llega a la conclusión de que no importa si las imágenes estan a la vista o no, siguen siendo procesadas bajando el número de imágenes por segundo.

Estos resultados se pueden mejorar implementando la optimización explicada en la parte de diseño y arquitectura. Se consigue una mejora en las imágenes por segundo pero que no es constante, ya que los otros jugadores no están siempre en la misma posición y el número de jugadores visibles varía.

Aún con esta optimización las imágenes por segundo caen al alcanzar un gran número de jugadores, esto es porque se aumenta la carga que tiene que realizar la CPU calculando las posiciones y distancias de los jugadores.

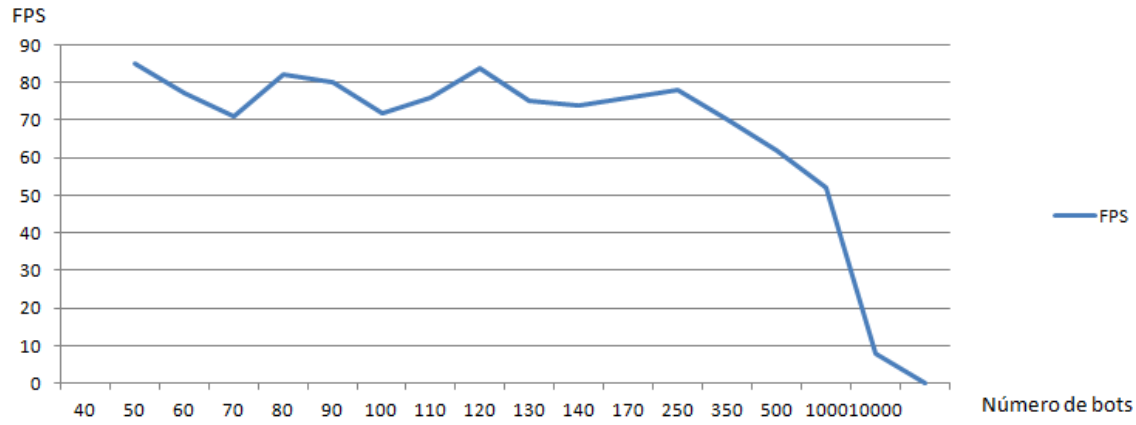

Figura 33 – Imágenes por segundo con optimización

En el caso de que se utilize la optimización por parte del servidor, los resultados son muy parecidos, aumentan un poco las imágenes por segundo con muchos bots pero sigue cayendo a partir de los 500, parece que es porque entran y salen muchos jugadores de la pantalla y esto causa que se tengan que volver a dibujar.

4.1.3 Ancho de banda

Se realizaron pruebas de estrés a fin de conocer el límite de ancho de banda, para ello el servidor le envía al cliente muchos mensajes de movimiento de bots.

Todas las pruebas se han centrado en medir la capacidad del cliente (el dispositivo móvil), debido a que el servidor no llega al 1% de CPU cuando el cliente ya se ha saturado, principalmente porque el servidor solo maneja números, los procesa y los envía, mientras que el cliente además tiene que dibujar los gráficos.

En el caso de que el cliente dibuje cada movimiento la aplicación deja de funcionar si se le envían muchas ordenes seguidas de mover bots, para medir la capacidad de la red, se ha configurado que aunque el servidor le diga que mueva los bots, el cliente no los dibuja moviéndose, en consecuencia recibe los mensajes y los procesa sin dar error.

Las pruebas se han realizado con el cliente y el servidor en la misma red, por lo que el tiempo mínimo que tardan en llegar los mensajes es de pocos milisegundos.

Para medir cuándo se ha alcanzado el límite se ha hecho que el servidor le mande al cliente una petición de tiempo de retardo cada 2 segundos para ver cuánto tarda en responder.

En el siguiente gráfico el eje Y es lo que tarda el cliente en responder en milisegundos. El eje X es la frecuencia con la que se envían mensajes de movimiento en milisegundos.

Como se puede observar cuando hay poca carga el retardo disminuye al aumentar el número de mensajes que se envían, cuando se espera lo contrario, esto es porque el cliente guardará los mensajes en un almacén temporal hasta tener suficientes y luego mandarlos todos a la vez para realizar una menor cantidad de conexiones. Esto ocurre entre los 2 mensajes por segundo y los 10 mensajes por segundo para 50 y 100 bots

A partir de 500 bots se empiezan a perder mensajes y por eso la latencia es más inestable.

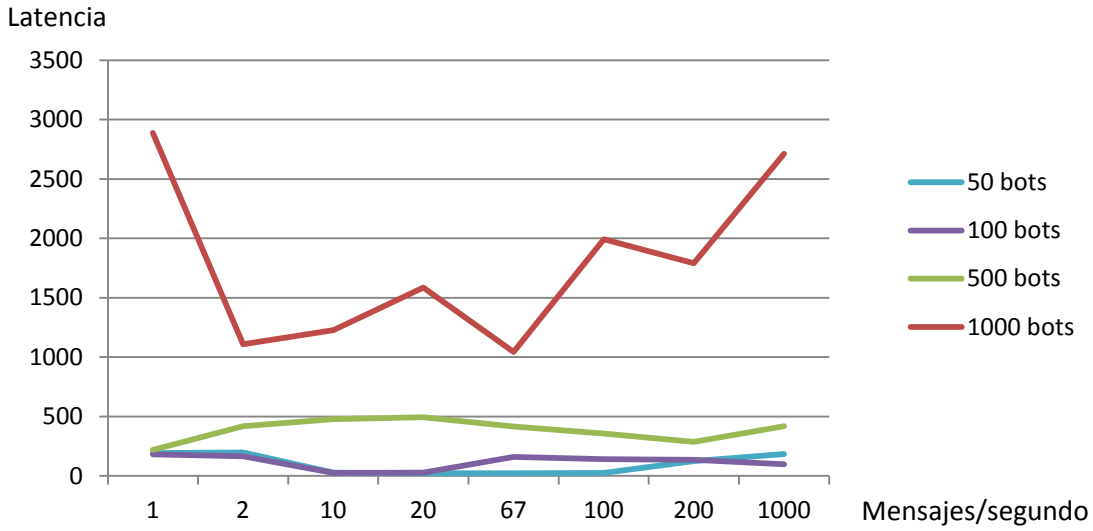

Figura 34 – Latencia

El siguiente gráfico mide en kiloBytes por segundo, la cantidad de información que el servidor le envía al cliente.

Se puede ver que hay un momento en que al aumentar la cantidad de mensajes que se envían no aumenta el ancho de banda, esto es porque el dispositivo móvil puede descargar a más velocidad, pero la gran cantidad de mensajes pequeños lo satura antes.

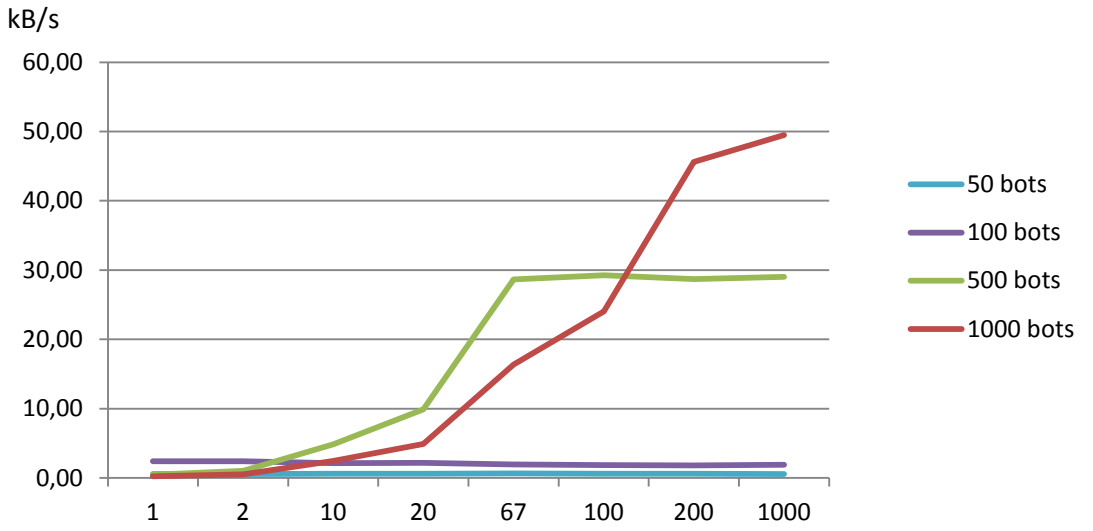

Figura 35 – Tasa de transferencia

La optimización de mandar solo la parte relevante para cada jugador hace que se envíen muchos menos mensajes pero a costa de que el servidor tenga que realizar más cálculos provocando que pierda velocidad y no consiga mandar mensajes con tanta frecuencia por lo que se pierden o llegan con mucho retraso.

En la gráfica se ve cómo se reduce el ancho de banda en comparación con la versión sin optimizar pero al enviar muchos mensajes con muchos bots hay un momento en el que al servidor no le da tiempo a terminar de calcular las distancias necesarias para la optimización y baja el ancho de banda utilizado.

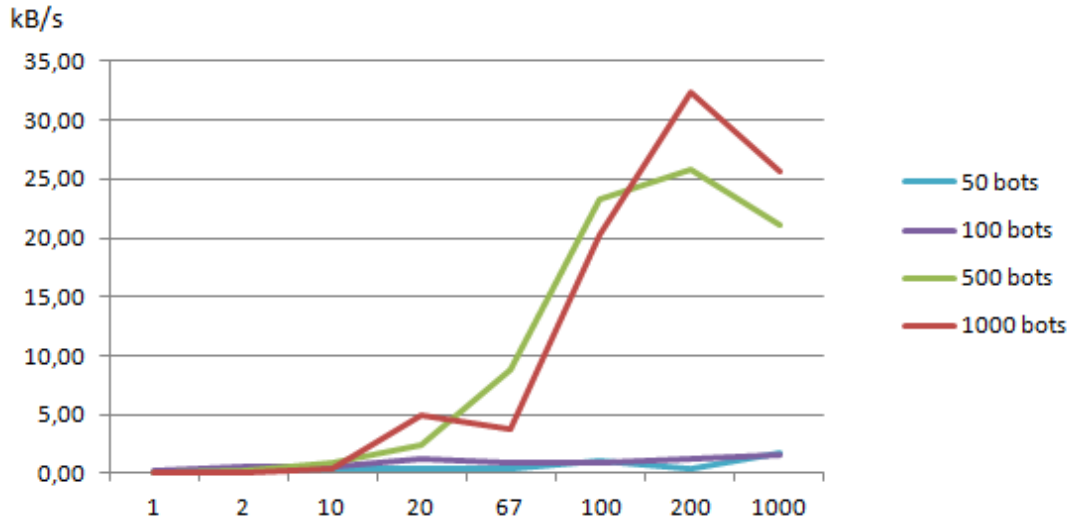

Figura 36 – Tasa de transferencia con optimización

El algoritmo que calcula las distancias de los jugadores es algo pesado, tiene una complejidad de N^2 ya que tiene que ir comparando cada jugador con los demás jugadores.

4.1.4 Espacio físico

El juego actualmente ocupa 5.59 MegaBytes, lo que más pesa son la librería y las imágenes de los sprites.

Con la configuración por defecto Android nos instala el juego en la memoria interna.

- 5.59 MB de memoria interna.

Cambiando el fichero AndroidManifest.xml, se consigue configurar la instalación por defecto a almacenamiento externo.

- 1.47 MB de memoria interna.

- 4.12 MB tarjeta SD.

Así se consigue que se utilice menor cantidad de memoria interna, que es la que comúnmente suele ser más pequeña.

4.1.5 Uso de batería

Los dispositivos móviles, ya sean smartphones o tablets, disponen de una batería interna recargable que alimenta a todos los componentes del dispositivo y es necesario que se utilice de forma eficiente y sin derrochar para obtener la mayor autonomía posible.

Es importante destacar que los usuarios tienen en cuenta la cantidad de batería que utilizan los juegos, un usuario puede decidir no jugar si piensa que prefiere no gastar la batería.

Se han realizado varias pruebas con el fin de determinar si es posible reducir el consumo de batería haciendo el juego más eficiente.

Se realizaron dos pruebas:

- La primera prueba se realizó con 50 bots en pantalla moviéndose constantemente.
- En la segunda, no se añadió ningún bot, únicamente se dibujaba al jugador propio y al escenario.

En la tabla se aprecia que al aumentar el número de bots, aumenta el consumo de batería, esto es debido a que se utiliza más la CPU y el dispositivo de red.

	Con 50 bots	Sin bots
Nivel de batería inicial	100%	100%
Nivel de batería después de 1 h de uso	81%	83%
Batería consumida	19%	17%

Tabla 5 – Uso de batería

4.2 Resumen de resultados

A partir de los resultados obtenidos se llega a la conclusión de que aunque se mejore una parte reduciendo la carga para un recurso, hay otras limitaciones que también hay que tener en cuenta. Por ejemplo, la optimización de dibujar solo los gráficos relevantes hace que se aumenten las imágenes por segundo, pero solo hasta cierto punto, porque el hecho de realizar las operaciones necesarias para la optimización, hace que la CPU esté más ocupada e incluso en algunos casos de peores resultados. Si ponemos que todos los jugadores están al alcance, se dibujarán todos y además se seguirá calculando su posición, por lo que la optimización empeoraría el rendimiento del juego.

Otra conclusión que se ha sacado es que en la mayoría de los móviles se llega antes al límite gráfico que al límite de la red, y en el caso en que no se requiera dibujar muchos sprites, lo que lo limita es el dispositivo de red, el cual se satura por un alto número de mensajes en poco tiempo no por el tamaño de estos.

5. Conclusiones

Con este proyecto he conseguido consolidar mis conocimientos sobre Android. También he logrado ver de cerca lo que es llevar a cabo un proyecto de principio a fin con todas sus fases.

Ya se sabía desde un principio que el proyecto iba a requerir más tiempo de lo que le corresponde a un cuatrimestre, por lo que se empezó un poco antes, aún así se ha ido con el tiempo justo.

Es posible mejorar algunas características de un juego multijugador, pero teniendo en cuenta que si se mejoran unas se empeoran ya es más difícil decantarse por un diseño u otro.

6. Líneas futuras

Algunos autores afirman que una arquitectura del tipo cliente-servidor no es escalable, pero esto no es cierto, una arquitectura de este tipo puede ser escalable en cierto modo, por ejemplo si se va dividiendo el mapa en distintos servidores en el que cada uno controla una parte, es posible aumentar el rendimiento más de lo que se puede conseguir con uno solo. Incluso es posible que varios jugadores interactúen entre ellos estando cada uno en un servidor distinto, aunque esto ya es más complejo de implementar porque requiere

comunicación extra entre los servidores provocando problemas de sincronización y de concurrencia.

Es posible completar el juego que se ha propuesto, añadiéndole funcionalidades como son una base de datos para los usuarios, un sistema de identificación y registro, objetivos y reglas de juego y un módulo para el chat entre usuarios. Con estas mejoras el juego estaría listo para enfrentarse a la competencia del mercado, sería incluso posible publicarlo en Google Play para su distribución y venta.

Acerca de las optimizaciones, su implementación es posible no solo en juegos de tipo MMO, sino también en otros que requieran una gran cantidad de dibujos en la pantalla, e incluso para aplicaciones también tengan muchos dibujos.

7. Planificación y presupuesto

7.1 Planificación

Para una mejor organización de la planificación se han distribuido las tareas en un calendario coloreado, dependiendo del tiempo que se le haya asignado a cada tarea, que a un día se le haya asignado una tarea significa que la mayoría del tiempo se le ha dedicado a esa tarea, por ejemplo, mientras se redactaba la memoria se han ido completando otras tareas.

La leyenda correspondiente al calendario de planificación es la siguiente.

En el calendario se puede ver cómo se ha ido realizando el proyecto.

Tabla 6 – Calendario

7.2 Presupuesto

Teniendo en cuenta que la duración del proyecto ha sido de 6 meses el presupuesto final necesario a lo largo del proyecto ha sido el siguiente.

7.2.1 Personal

En la siguiente tabla se muestra coste total por mano de obra.

Nombre	Sueldo/hora	Horas	Total (€)
Hugo Alberto Huerta García	10 €	223	2.230,00 €
Total (€)			**2.230,00 €**

Tabla 7 – Mano de obra

7.2.2 Hardware

En la siguiente tabla se muestra el cálculo del coste del material de Hardware.

Nombre	Precio/ud	Unidades	Coste ud/mes	Total (€)
Medion desktop core 2 duo	300 €	1	5,00 €/mes	30,00 €
Periféricos	5 €	3	0,20 €/mes	3,60 €
Galaxy Mini	50 €	1	1,00 €/mes	6,00 €
Router Zyxel	20 €	1	0,50 €/mes	3,00 €
Total (€)				**42,60 €**

Tabla 8 – Coste del hardware utilizado

7.2.3 Software

En la siguiente tabla se muestra el software adquirido.

Nombre	Precio/ud	Unidades	Coste ud/mes	Total (€)
Office 2003	65 €	1	0,90 €/mes	5,40 €
Total (€)				**5,40 €**

Tabla 9 – Coste del software utilizado

7.2.4 Consumibles

En la siguiente tabla se muestra el cálculo del coste del material consumible.

Nombre	Precio/ud	Unidades	Coste ud/mes	Total (€)
Tinta negra para impresora	25 €	1	2,00 €/mes	12,00 €
Total (€)				**12,00 €**

Tabla 10 – Coste de los consumibles utilizados

7.2.5 Viajes

En la siguiente tabla se muestran los costes asociados a los viajes necesarios para las reuniones.

Nombre	Precio/ud	Unidades	Coste ud/mes	Total (€)
Abono mensual de metro	28,50 €	6	28,50 €/mes	171,00 €
Total (€)				**171,00 €**

Tabla 11 – Coste de los viajes

7.2.6 Cálculo final de costes

Concepto	Coste (€)
Mano de obra	2.230,00 €
Hardware	42,60 €
Software	5,40 €
Consumibles	12,00 €
Viajes	171,00 €
Total sin IVA (€)	2.461,00 €
IVA (18%)	442,98 €
Total (€)	**2.903,98 €**

Tabla 12 – Cálculo final de costes

8. Bibliografía

1. Wikipedia. [En línea] http://en.wikipedia.org/wiki/Library_(computing).

2. Wikipedia. [En línea] http://en.wikipedia.org/wiki/Smartphone.

3. Wikipedia. [En línea] http://es.wikipedia.org/wiki/Software.

4. Wikipedia. [En línea] http://en.wikipedia.org/wiki/Tablet_computer.

5. Wikipedia. Comparison of Android devices. [En línea] Noviembre de 2011. http://en.wikipedia.org/wiki/Comparison_of_Android_devices.

6. Android. Android developers. [En línea] http://developer.android.com/resources/dashboard/platform-versions.html.

7. iOS. [En línea] http://es.wikipedia.org/wiki/IOS_(sistema_operativo).

8. TalkAppleIphoneOnline. [En línea] http://www.talkappleiphoneonline.com/.

9. AndroidAyuda. [En línea] http://androidayuda.com/2012/06/13/el-80-de-los-smartphones-espanoles-son-dispositivos-android/.

10. *A communication ARchitecture for Massive Multiplayer Games.* Fiedler, Stefan.

11. The World Of Magic. [En línea] com2us. http://global.com2us.com/game/worldofmagic/android.

12. Warspear Online. [En línea] Aigrind. http://warspear-online.com/en/home.

13. Pocket legends. [En línea] Spacetime Studios. http://www.pocketlegends.com/.

14. *Challenges in Peer-to-Peer Gaming.* Neumann, Christoph, y otros.

15. *Architecture for a massively multiplayer online role playing game engine.* Caltagirone, Sergio.

16. *Architecting scalability for massively multiplayer online gaming experiences.* Gil, Rui, Tavares, José Pedro y Roque, Licinio. Coimbra : s.n., 2005.

17. AndEngine. [En línea] http://www.andengine.org/.

18. Bunny shooter. [En línea] https://play.google.com/store/apps/details?id=com.bestcoolfungamesfreegameappcreation.bunnyshooter.

19. libGDX. [En línea] http://code.google.com/p/libgdx/.

20. MiniFarm. [En línea] https://play.google.com/store/apps/details?id=de.netzweh.minifarm.

21. Slick. [En línea] http://slick.cokeandcode.com/.

22. Foro de Slick. [En línea] http://slick.javaunlimited.net/viewtopic.php?f=21&t=2834&start=30.

23. Skiller. [En línea] http://www.skiller-games.com/.

24. Kunligu Kvar Online. [En línea] https://play.google.com/store/apps/details?id=de.goddchen.android.x.fourinarow.

25. RoarEngine. [En línea] http://roarengine.com/.

26. RoarEngine blog. [En línea] http://blog.roarengine.com/.

27. Corona. [En línea] http://www.anscamobile.com/corona/.

28. Sheep Guardian. [En línea] http://es.androidzoom.com/android_games/arcade_and_action/sheep-guardian_bkutr.html.

29. Marmalade. [En línea] http://www.madewithmarmalade.com/.

30. Plant vs Zombies. [En línea] http://www.madewithmarmalade.com/marmalade/benefits/number-one-for-mobile-games-on-ios-android-and-beyond.

31. Unity. [En línea] http://unity3d.com/.

32. Dino Picker. [En línea] https://play.google.com/store/apps/details?id=com.Voicegateindia.DinoPicker.

33. PubNub. [En línea] http://www.pubnub.com/.

34. Epoch Go. [En línea] http://www.twylah.com/PubNub/tweets/192092324522770432.

35. huawei u8230. [En línea] http://www.gizmonder.com/2010/09/huawei-u8230.html.

36. longinusboy. Android sdk vs AndEngine vs libGDX. [En línea] http://www.youtube.com/watch?v=RBiq2SGs93U.

37. Lindeijer, Thorbjørn. Tiled Map Editor. [En línea] http://www.mapeditor.org/.

38. Imagen Galaxy Mini. [En línea] http://www.three-clearance.co.uk/three-clearance-samsung-galaxy-mini-silver-refurbished.html.

39. Smus, Boris. Developing multiplayer HTML5 games with node.js. [En línea] http://smus.com/multiplayer-html5-games-with-node/osmus-architecture.png.