HeartStone M.E.

AUTOR: ENRIQUE MARTÍN ARENAL
Tutor: José María González Ríos

Tabla de contenido

Índice de tablas

Índice de ilustraciones

Agradecimientos

Me gustaría agradecer a todas las personas que de una manera u otra me han apoyado durante la realización de este proyecto. Por compartir mis dudas, preocupaciones, alegrías y sobre todo el día a día.

A mis padres, por aguantarme en algunos días difíciles, por estar siempre ahí y por creer en mí.

A mi hermano por echarme una mano en lo que pudo y no darme demasiado la vara mientras realizaba el proyecto.

A mis compañeros de clase, con los que he compartido tantos momentos, por escucharme con los problemas del proyecto.

Y por último y de manera especial a José Luis y Gonzalo que son los que más me han ayudado tanto durante el curso como en la realización del proyecto. Sin ellos no hubiera sido capaz de realizar las modificaciones necesarias para la correcta visualización de las cartas.

Muchas gracias a todos.

1. Estudio del problema y análisis del sistema

1.1 Introducción

Con el auge de las nuevas tecnologías, la masificación del uso de los conocidos como teléfonos inteligentes y la generalización de tarifas planas de datos (acceso a internet) para móviles, es bastante obvio que el mercado potencial de una aplicación para teléfonos móviles es enorme.

Sí a esto le unimos que el uso de los teléfonos móviles ha pasado de ser tan solo para comunicarse con otras personas a ser usados en gran parte como medio de entretenimiento, se puede concluir que enfocar el proyecto final del grado superior de desarrollo de aplicaciones multiplataforma en el desarrollo de un juego para dispositivos Android es una oportunidad para adquirir conocimientos que pueden servir para buscar un futuro profesional dentro de esta rama de la programación.

En este proyecto se va a intentar precisamente crear un juego para Android, además se va a tratar de un juego exclusivamente multijugador lo que va a suponer tener que enfrentarse, además de al complicado mundo del desarrollo de videojuegos, a las no menos complicadas comunicaciones en red.

El nombre elegido para el proyecto es “HeartStone M.E. (Mobile Edition)”.Se va a tratar de un juego de cartas de fantasía en el que cada jugador tiene un turno para elegir qué hacer con sus cartas. Al empezar la partida a cada jugador se le asignan unas cartas del mazo y una cantidad determinada de ‘puntos de vida’ y al empezar su turno recibirá una carta más del mazo.

Cada carta tiene unos atributos (coste, ataque, defensa) y una vez puestas en la zona de juego tendremos que esperar al siguiente turno para poder usarlas ya sea contra las cartas del oponente o el propio oponente. La partida terminará cuando se le agote la vida de uno de los jugadores.

1.2 Motivación y finalidad

La intención de este proyecto es crear una versión simple pero entretenida y con una alta jugabilidad para dispositivos Android del conocido juego HearthStone, de la compañía Blizzard.

La principal motivación para llevar a cabo el proyecto es realizar una introducción lo más amplia que el tiempo y los recursos permitan en el mundo del desarrollo de juegos para dispositivos móviles. Además, es un proyecto en el que se va a tener que echar mano de muchas de las materias impartidas durante el curso.

1.3 Objetivos

El objetivo principal del proyecto es crear y desarrollar para dispositivos Android la parte gráfica y lógica de un juego parecido al HearthStone de Blizzard. Para llevar a cabo este objetivo serán necesarios otra serie de objetivos más concretos, que son los siguientes:

-Objetivos principales:

- Diseñar y crear las cartas y los escenarios del juego.
- Crear una interfaz gráfica que contenga toda la información necesaria para el desarrollo de las partidas, pero que a la vez no esté demasiado sobrecargada.
- Dicha interfaz ha de poderse usar en los distintos tamaños de pantalla que nos podemos encontrar en los teléfonos móviles de hoy en día.
- Desarrollo de la lógica de las partidas de cartas (esta lógica se explicará detalladamente más adelante).
- Desarrollar un servidor que permita que dos jugadores, cada uno desde su propio dispositivo Android, puedan enfrentarse el uno al otro.

2. Material necesario

2.1 Recursos hardware

Para el desarrollo del proyecto se han usado los siguientes recursos hardware:

- Ordenador portátil Acer Aspire 5738-Z cuyas especificaciones son:
  - Sistema operativo Windows 7.
  - 4Gb de memoria ram.
  - 500 Gb de disco duro.
  - Procesador Intel 2.1Ghz dual core.
  - Tarjeta gráfica de 512Mb de memoria.
- Dispositivo Android: Sony Xperia U con versión 4.1 de Android instalada.

Como referencia se expone a continuación los requisitos recomendados para poder ejecutar los programas software necesarios para la ejecución del proyecto:

- Requisitos recomendados para usar NetBeans IDE 7.4 en Windows 7:
  - Procesador: Intel Core i5 o equivalente.
  - Memoria RAM: 2 GB (32-bit), 4 GB (64-bit).
  - Espacio en disco: 1.5 GB de espacio libre.

- Requisitos recomendados para usar Eclipse
  - Procesador: Intel Core i5 o equivalente.
  - Memoria RAM: 2 GB (32-bit), 2 GB (64-bit).
  - Espacio en disco: 300 MB de espacio libre.

- Requisitos recomendados para usar el emulador de Android:
  - Procesador: Intel Core i3 o equivalente.
  - Memoria RAM: 4 GB.
  - Espacio en disco: 250 MB de espacio libre.

- Requisitos recomendados para usar Photoshop CS6:
  - Procesador Intel® Pentium® 4 o AMD Athlon® 64
  - 1 GB de RAM
  - 1 GB de espacio libre en disco duro para la instalación; se necesitará espacio libre adicional durante la instalación (no se puede instalar en dispositivos extraíbles de almacenamiento basados en memoria flash).
  - Resolución de pantalla de 1024 x 768 (se recomienda 1280 x 800) con color de 16 bits y 512 MB de VRAM (se recomienda 1 GB)
  - Sistema con OpenGL 2.0

2.2 Recursos software

Los recursos software que han sido necesarios para poder desarrollar el proyecto son:

- Lenguaje de programación Java.
- LibGdx.
- Android Sdk.
- Emulador de android.
- Eclipse con el plugin para desarrollar en Android.
- Photoshop CS6.
- Netbeans para el desarrollo del servidor.
- Editor de partículas.
- Editor de sonidos.
- Librería Kryonet para la comunicación en red.

A continuación se explica un poco en qué consisten algunos de ellos.

Lenguaje de programación Java

Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.

Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como *WORA*, o "*write once, run anywhere*"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados.

Eclipse IDE

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

NetBeans IDE

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Android Sdk

El SDK (Software Development Kit) de Android, incluye un conjunto de herramientas de desarrollo.5 Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales. Las plataformas de desarrollo soportadas incluyen Linux (cualquier distribución moderna), Mac OS X 10.4.9 o posterior, y Windows XP o posterior.

La plataforma integral de desarrollo (IDE, Integrated Development Environment) soportada oficialmente es Eclipse junto con el complemento ADT (Android Development Tools plugin), aunque también puede utilizarse un editor de texto para escribir ficheros Java y Xml y utilizar comandos en un terminal (se necesitan los paquetes JDK, Java Development Kit y Apache Ant) para crear y depurar aplicaciones. Además, pueden controlarse dispositivos Android que estén conectados (e.g. reiniciarlos, instalar aplicaciones en remoto).

Las Actualizaciones del SDK están coordinadas con el desarrollo general de Android. El SDK soporta también versiones antiguas de Android, por si los programadores necesitan instalar aplicaciones en dispositivos ya obsoletos o más antiguos. Las herramientas de desarrollo son componentes descargables, de modo que una vez instalada la última versión, pueden instalarse versiones anteriores y hacer pruebas de compatibilidad.

LibGdx

LibGdx es un framework para el desarrollo de videojuegos multiplataforma, soportando actualmente Windows, Linux, Mac OS X, Android, iOS y HTML5.

Una de los objetivos principales de la librería es mantener la simplicidad, sin renunciar al amplio abanico de plataformas finales. Para ello, te permite únicamente escribir tu código en un único proyecto y exportarlo a las tecnologías mencionadas anteriormente sin modificar nada. Pudiendo utilizar la versión de escritorio como entorno de pruebas para el resto, siguiendo así una iteración de desarrollo rápida e integrable con el resto de herramientas de Java.

Además, Libgdx permite bajar el nivel de abstracción tanto como quieras, dándote acceso directo al sistema de archivos, dispositivos de entrada y audio, e incluso a las interfaces de OpenGL 2.0 y 3.0.

Encima de esta capa de interacción, se establece un potente conjunto de APIs que te permite mostrar imágenes y texto, construir interfaces de usuario, reproducir sonidos o música, realizar cálculos matemáticos, parsear ficheros XML o JSON, etc.

Ligdx pretende ser un framework más que un motor de desarrollo, admitiendo que no es una solución todo en uno, sino que provee al programador de un potente conjunto de abstracciones y le dan total libertad para desarrollar su aplicación.

Photoshop

Adobe Photoshop es un editor de gráficos rasterizados desarrollado por Adobe Systems principalmente usado para el retoque de fotografías y gráficos. Su nombre en español significa literalmente "taller de fotos".

Es líder mundial del mercado de las aplicaciones de edición de imágenes y domina este sector de tal manera que su nombre es ampliamente empleado como sinónimo para la edición de imágenes en general.

Kryonet

Kryonet es una librería de Java que provee de una limpia y eficaz API para realizar una comunicación eficiente TCP y UDP entre cliente y servidor. KryoNet utiliza la biblioteca de serialización Kryo para transferir de forma automática y de manera eficiente los gráficos de objetos a través de la red.

KryoNet se ejecuta tanto en aplicaciones de escritorio como de Android.

KryoNet es ideal para cualquier aplicación de cliente / servidor. Es muy eficiente, por lo que es especialmente bueno para los juegos. También puede ser útil para la comunicación entre procesos.

3. Planificación

3.1 Secuencia del desarrollo del proyecto

Para planificar este proyecto se ha decidido dividirlo en las siguientes tareas:

- Especificación de los requisitos.
- Descarga e instalación de los recursos software necesarios.
- Diseño del sistema.
  - Diagrama de clases de la aplicación cliente.
  - Lógica del juego
  - Forma de interacción entre cliente y servidor.
- Adquisición de recursos usados en la aplicación :
  - Imágenes.
  - Sonidos.
  - Efectos especiales.
- Programación del sistema:
  - Desarrollo del motor del juego.
  - Implementación de recursos.
  - Desarrollo de las animaciones.
  - Comunicaciones servidor/cliente y viceversa.
  - Opciones de configuración del juego.
- Ejecución de las pruebas.
- Realización de un tutorial del juego.

La ejecución de estas tareas intentará seguir la siguiente línea temporal:

Nombre de la tarea	Fecha de inicio	Duración aproximada
Especificación de los requisitos	3-03-2014	4 días
Descarga e instalación de los recursos software necesarios	7-03-2014	1 día
Diseño del sistema	8-03-2014	5 días
Desarrollo del motor de juego	13-03-2014	3 días
Adquisición de imágenes para usar en el juego	16-03-2014	3 días
Implementación de las imágenes	19-03-2014	5 días
Desarrollo de las animaciones	24-03-2014	5 días
Pruebas de las animaciones	29-03-2014	2 días
Adquisición de los efectos especiales	31-03-2014	1 día

Implementación de los efectos especiales	1-04-2014	1 día
Comunicaciones cliente/servidor	2-04-2014	16 días
Pruebas de la comunicación cliente/servidor	18-04-2014	10 días
Adquisición de los sonidos	28-04-2014	1 día
Implementación de los sonidos en el juego	29-04-2014	3 días
Desarrollo de las opciones de configuración del juego	1-05-2014	4 días
Crear un tutorial del juego	5-05-2014	3 días
Ejecución de pruebas y perfeccionamiento del juego	8-05-2014	Hasta el día de la entrega

Tabla 1 Planificación del proyecto

4. Seguimiento

4.1 Secuencia real del desarrollo del proyecto

A la hora de realizar el desarrollo del proyecto se ha intentado seguir en la medida de lo posible la planificación vista anteriormente. Evidentemente los tiempos no han coincidido en la mayoría de los casos con los expuestos, en algunos casos se ha tardado más de lo esperado y en otros menos, pero en general se ha seguido el mismo orden.

A continuación se explicará qué es lo que se ha hecho en cada tarea y se expondrán las dificultades encontradas a la hora de desarrollarlas.

4.1.1 Especificación de los requisitos

Durante esta tarea se han identificado los requisitos necesarios para poder llevar a cabo el proyecto. Dichos requisitos son:

Conocimientos:

- Conocer el lenguaje de programación Java.
- Adquirir conocimientos sobre programación de hilos.
- Adquirir conocimientos sobre programación en red.
- Aprender a editar imágenes y sonidos.
- Aprender a usar la librería LibGdx para desarrollar juegos en Android.

Herramientas software:

- Librería LibGdx y sus utilidades (Hiero, ParticleEditor).
- Java JDK.
- Eclipse IDE con plugin para Android.
- Photoshop CS6.
- SDK de Android.
- Dos dispositivos móviles con sistema operativo Android o en su defecto el emulador para ordenador.
- Polderbits para capturar y editar sonidos.
- Librería KryoNet para la comunicación en red.

Se tardó aproximadamente un día en realizar esta tarea

4.1.2 Descarga e instalación de los recursos software necesarios

Los recursos software necesarios son todas las herramientas software mencionadas en el apartado anterior. La gran mayoría ya estaban instaladas

antes de empezar este proyecto por lo que no se ha perdido apenas tiempo en esta tarea.

Aquellos recursos que se necesitó instalar no dieron ningún problema a la hora de hacerlo. Aproximadamente se tardó un día como mucho en realizar esta tarea

4.1.3 Diseño del sistema

El diseño del sistema se repartió en tres tareas, el desarrollo de un diagrama de clases, la forma en la que los clientes y el servidor van a interactuar y el desarrollo de la lógica del juego.

Las tareas que componen éste apartado se completaron en un total de 5 días de trabajo.

Diagrama de clases

La siguiente imagen muestra el diagrama de clases del proyecto llamado “heartstone”, que es el proyecto desde el que se ejecuta el juego:

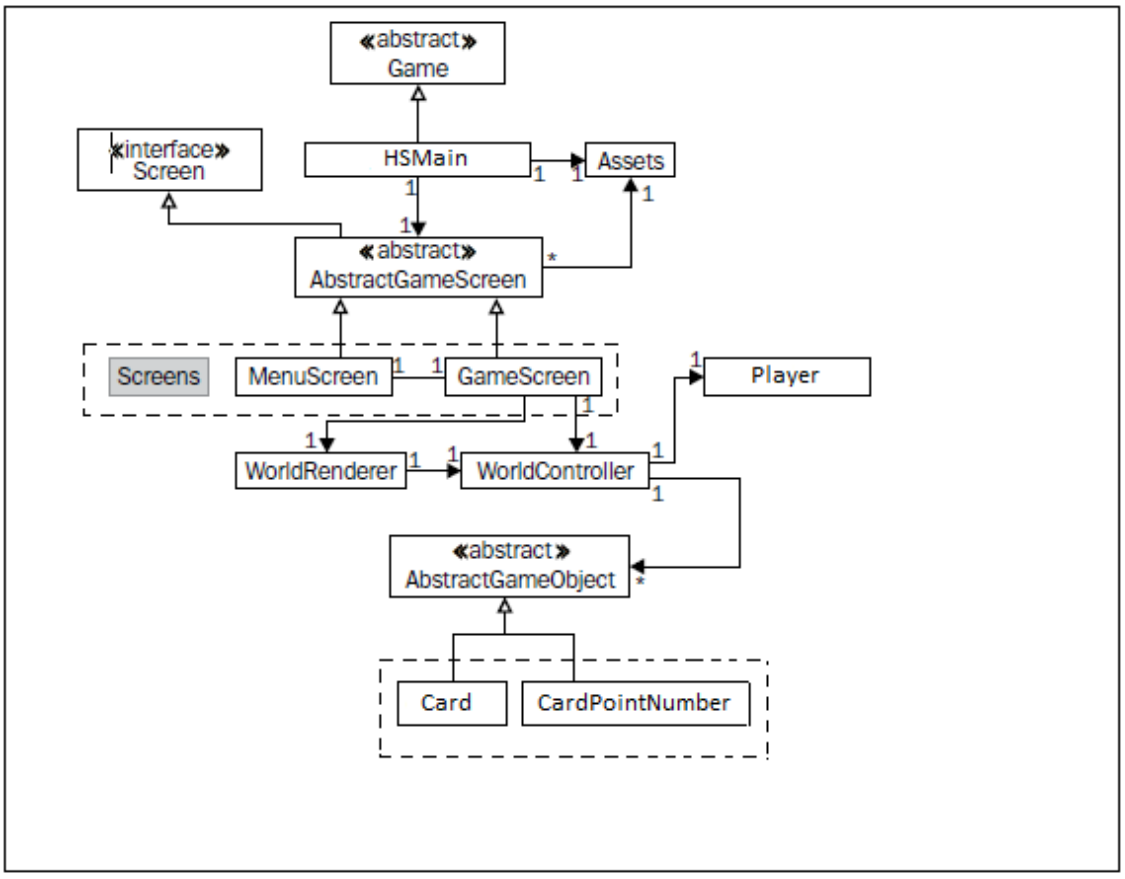

Ilustración 1 Diagrama de clases

Además de lo que se puede ver en el diagrama hay otras clases dentro de un paquete llamado “util”. Entre las clases dentro de este paquete se encuentra una clase llamada “Constants” en la que se han escrito todas las constantes que se usan en el proyecto y a la cual se puede acceder desde cualquier clase del proyecto.

Cabe destacar también las clases Network y Stats, también dentro del paquete “útil”, que se usan para facilitar el traspaso de información entre cliente y servidor.

Interacción cliente/servidor

Se decidió que la interacción entre el cliente y el servidor se realice siguiendo el siguiente modelo:

-Cada cliente que quiere iniciar una partida conecta con el servidor y se queda esperando hasta que se le encuentre un enemigo contra el que enfrentarse.

-Cada vez que un cliente conecta con el servidor, éste se encarga de asignarle otro cliente contra el que se jugará la partida. Para hacer esto el servidor almacena una lista con los clientes que aún no tienen enemigo asignado y según va recibiendo conexiones de clientes va emparejándolos con los clientes de esa lista.

-Una vez dos clientes están emparejados se les manda la señal de que la partida puede comenzar, junto con los datos necesarios para empezar la partida.

-Estos datos están encapsulados en dos clases diferentes, una que representa los datos de las cartas y otra que representa los datos generales de la partida. Además se usan otras clases para mandarse mensajes entre el cliente y el servidor e identificar ciertas acciones realizadas por los clientes.

-El jugador que tiene el turno le envía al servidor la acción que acaba de realizar, cuando el servidor la recibe la identifica y le pasa los datos necesarios al otro jugador para que la acción que había realizado el primer jugador se refleje.

-Además de todo lo anterior se crearán métodos para manejar las desconexiones de los clientes y enviarles un aviso de que se ha perdido la conexión con el servidor o que el enemigo se ha desconectado.

Lógica del juego

La lógica del juego se basa en las cartas y los atributos ligados a cada carta y se desarrolla en un sistema basado en turnos en los que cada jugador solamente puede realizar acciones cuando es su turno.

Cada carta tiene los siguientes atributos:

- Puntos de ataque, que determinan el daño que hace esa carta al enfrentarse a otra carta o al jugador enemigo.
- Puntos de vida, es la vida restante de cada carta, determina cuánto daño puede recibir esa carta antes de “morir”.
- Coste de cristales, representa el coste para poder jugar esa carta.

Cada partida se desarrolla de la siguiente manera:

-Al comenzar a cada jugador se le asigna su mazo propio de cartas y se elige quién tiene el turno. Los mazos están compuestos por 24 cartas diferentes.

-Se le dan a cada jugador dos cartas aleatorias de su mazo, y un máximo de un cristal.

-El jugador que tiene el turno puede realizar las siguientes acciones:

- Sacar una carta.
    - Para poder sacar una carta a la mesa el coste de cristales de la carta ha de ser menor o igual que los cristales que actualmente tiene el jugador. Una vez sacada la carta se le resta este coste a los cristales del jugador. Hay que tener en cuenta que un jugador sólo puede tener como máximo cinco cartas en la mesa a la vez y por lo tanto si esto sucede no se le dejará sacar carta hasta que tenga menos cartas.
- Atacar con una carta de la mesa a otra carta.
    - Al realizar esta acción las dos cartas se enfrentan y se les resta de sus puntos de vida los puntos de ataque que tenga la carta enemiga. Si los puntos de vida después de enfrentarse son superiores de cero entonces la carta permanece en la mesa en caso contrario desaparece.
- Atacar con una carta de la mesa al jugador enemigo.
    - Al realizar esta acción a la vida del jugador enemigo se le restan los puntos de ataque que tenga la carta. Si después de la acción los puntos de vida del enemigo son superiores a cero el juego continua sino el juego termina y el jugador atacante es proclamado ganador de la partida.
- Pasar de turno, esto también sucede automáticamente pasados 2 minutos desde que el jugador recibió el turno para evitar la total interrupción de una partida si un jugador decide no hacer nada durante su turno.

-Sólo se puede realizar una acción con cada carta en cada turno.

-Cuando se pasa de turno el jugador que lo recibe coge una carta aleatoria de las que queden en su mazo y a la vez sus cristales máximos se reestablecen. Hay un tope de 5 cartas en la mano del jugador, si al recibir el turno el jugador tiene ese tope entonces la carta se perderá.

-Cuando se pasa una ronda (esto es un turno de cada jugador) el máximo de cristales para jugador se aumenta en 1 hasta un tope de 10.

4.1.4 Desarrollo del motor del juego

Dado que LibGdx es un marco para desarrollo y no un motor de juego en sí mismo hay que desarrollar nuestro propio motor de juego. Para ellos se han seguido los consejos de los creadores de LibGdx. La estructura del motor del juego puede verse en el diagrama de clases anteriormente expuesto

En la parte superior del diagrama se ve a la clase HSMain. Es la clase inicial del juego y hereda de la clase abstracta de LibGdx Game que implementa la interfaz ApplicationListener lo que nos permitirá detectar la entrada de datos del usuario. Esta clase contiene una referencia a la clase Assets (usada para organizar y simplificar el acceso a los recursos del juego) y hereda el método setScreen() que nos permitirá cambiar entre las distintas pantallas del juego.

Las pantallas que se van a usar en el juego están encapsuladas en diferentes clases. Todas las pantallas heredan de la clase AbstractGameScreen lo que permite definir el comportamiento común de todas las pantallas. Además AbstractGameScreen() implementa la interfaz Screen que introduce los métodos hide() y show().

En total hay dos pantallas en el juego la del menú inicial y la pantalla en la que, por decirlo así, se "juega", que se llama GameScreen. Ésta última clase contiene una referencia a otras dos clases, WorldController y WordlRenderer.

La clase WorldController contiene toda la lógica del juego y es la encargada de inicializar y modificar el mundo virtual que vamos a crear dependiendo de la interacción del usuario y la lógica del juego. Contiene una lista de referencias a varios objetos que han heredado de la clase AbstractGameObject como son las cartas que se muestran en el juego y que contienen varios atributos que nos ayudarán a manejarlas. Y otra referencia a la clase Player que sirve para ir almacenando el estado de una partida.

Por último nos encontramos con la clase WorldRenderer que como su propio nombre indica será la clase en la que se realizará el "render" ("dibujo" dicho de otra forma) de nuestro mundo. Dentro de esta clase necesitaremos una referencia a WorldController y a los objetos del mundo que vamos a dibujar.

En total se tardaron 3 días en realizar esta tarea.

4.1.5 Adquisición de imágenes para usar en el juego

En un principio las imágenes (cartas, fondos y demás) iban a ser imágenes sacadas del juego original y escaladas para mostrarse en dispositivos Android, pero estas imágenes no resultaban adecuadas para su uso en Android ya que la información que debían transmitir al jugador (coste de cristales, puntos de ataque etc.) no era lo suficientemente visible desde un teléfono móvil y por lo tanto se tuvieron que modificar.

La modificación más importante que se tuvo que realizar fue la de las cartas que iban a formar parte del juego (24 en total) ya que se necesitaban realizar los siguientes ajustes:

- Aumentar el tamaño de los números que representaban el coste de cristales y los puntos de ataque de la carta.
- Aumentar el nombre de la carta para que fuera visible.
- Eliminar el número que representaba los puntos de vida de la carta, esto es debido a que este número no es fijo durante el desarrollo de una partida, sino que va disminuyendo según se le ataca a la carta.

Las dos imágenes siguientes muestran el antes y el después de una carta:

Ilustración 2 Carta original

Ilustración 3 Carta Modificada

Aunque no lo parezca ambas imágenes tienen el mismo tamaño pero la original tiene alrededor zonas transparentes que ocupan mayor espacio. Además a simple vista se puede apreciar que tanto el tamaño de las letras como el de los número es de, al menos, el doble en la carta modificada.

Para realizar todas estas modificaciones y alguna más relativa al fondo del juego y a diversos botones se utilizó Photoshop CS6 y el tiempo aproximado que se invirtió fue de 4 días de trabajo.

4.1.6 Implementación de las imágenes

Una vez obtenidas las imágenes que se iban a utilizar en el juego para poder usarlas en dispositivos Android hay que tener en cuenta que aquellos dispositivos que no dispongan de OpenGL 2.0 o superior no van a permitir texturizar imágenes cuya resolución no sea una potencia de 2, además cada vez que se renderiza una textura distinta (imágenes desde distintos archivos) se necesita enviar información a la memoria de video y el procesador gráfico tiene que realizar muchas operaciones por lo que el rendimiento disminuye.

Para evitar estos problemas LibGdx incluye la clase TextureAtlas que nos permite manejar distintas imágenes que estén incluidas en la misma textura, y de esta manera quitarnos los problemas anteriormente comentados de un plumazo.

LibGdx contiene una utilidad con la que se pueden crear y actualizar de forma automática los texture atlas que se han usado en el desarrollo del proyecto. Tan solo hay que darle una ruta a la carpeta que contenga las imágenes que van a ser incluidas en el atlas y generará en la ruta que se le diga dos archivos, un png con las imágenes y un archivo con extensión .pack que contiene toda la información necesaria para acceder a cada imagen que contiene el atlas.

A la hora de usar las imágenes almacenadas en el texture atlas lo único que hay que hacer es buscar la textura que queremos mostrar por el nombre de la región que la contiene, este nombre coincide con el nombre de la imagen sin la extensión, y a partir de ahí dibujarla con el batch.draw().

Como ya se ha visto anteriormente el proyecto contiene una clase llamada Assets cuya misión es manejar de forma un poco más simple los recursos de la aplicación. Por lo tanto dentro de ésta clase se ha creado otra clase que es la encargada de manejar la utilización de los texture atlas.

Aproximadamente se tardó 3 días en realizar esta tarea.

4.1.7 Desarrollo de las animaciones

Después de aprender a mostrar las texturas se prosiguió el proyecto desarrollando todas las animaciones que se iban a producir durante una partida entre dos jugadores.

Por animaciones uno se refiere a un conjunto de movimientos de alguno de los elementos que se muestran en el juego cuando se produce alguna acción. En el caso de este proyecto las animaciones son bastante simples ya que en esencia sólo se necesita que las cartas se muevan de un lugar a otro.

El siguiente fragmento de código muestra cómo desplazar en línea recta un objeto de un punto a otro:

```
// Animation to move a card to the table
    private void moveCardToTableAnimation(float deltaTime) {
        if (moveToTable) {
            // The .set() is setting the distance from the starting position to
            // end position
            v2Velocity.set(
                    Constants.CARD_POSITION_TABLE[cardsTable.size() - 1].x
                    - animatedCard.position.x,
                    Constants.CARD_POSITION_TABLE[cardsTable.size() - 1].y
                    - animatedCard.position.y);

            v2Velocity.x *= 0.05f; // Set speed of the object
            v2Velocity.y *= 0.05f;
            animatedCard.position.add(v2Velocity);
            if (animatedCard.position.y > Constants.CARD_POSITION_TABLE[cardsTable
                    .size()].y - 0.02) {
                animatedCard.position.y = Constants.CARD_POSITION_TABLE[cardsTable
                        .size()].y;
                moveToTable = false;
                animatedCard.setSelected(false);
            }

        }
    }
```

Éste es el método que se utiliza para mover una carta de la mano del jugador a la mesa y es el más simple de todos los métodos que ejecutan animaciones. El método se llama cada vez que se llama al método update de la clase que controla la lógica del juego (esto suele suceder unas cuantas veces por segundo). Lo que sucede en el método es que, una vez realizadas las comprobaciones de que una carta se puede y se quiere mover a la zona de la mesa de juego, mediante la ayuda de la clase Vector2 de LibGdx se mueve el vector position de la carta hasta el punto de destino representado por el vector v2Velocity mediante el uso de los métodos set() y add(). Cada vez que se va acercando la carta al punto final la velocidad a la que se produce la animación va disminuyendo, tanto que en muchas ocasiones ni llega al punto de destino, por lo que una vez alcanzado un punto lo suficientemente cercano al destino se coloca la carta manualmente en él y se detiene la animación.

En total se tuvieron que realizar 9 métodos que son los que se encargan de ejecutar cada animación siguiendo más o menos el esquema que se acaba de mencionar.

Para evitar posibles conflictos en la ejecución de las animaciones se decidió que el usuario no pudiera ejecutar acciones mientras se está reproduciendo alguna animación. Esto se decidió tanto para evitar conflictos entre las animaciones como para evitar que el uso de memoria de la aplicación se dispare.

Esta tarea se llevó a cabo en 7 días de trabajo debido a la adquisición de los conocimientos necesarios para aprender a ejecutar animaciones y a que la ejecución de las animaciones ha de seguir adecuadamente la lógica del juego.

4.1.8 Adquisición e implementación de los efectos especiales

Durante la planificación de este proyecto se decidió añadir dos efectos especiales para representar por una parte, que una carta ataque a otra, y por otra parte para representar la “muerte” de una carta cuyos puntos de vida bajan de 1.

Estos efectos especiales se diseñaron con una herramienta que viene incluida en LibGdx, el ParticleEditor. Para ejecutar esta herramienta es necesario abrir la línea de comandos y colocarnos en la carpeta en la que tengamos descomprimido LibGdx y ejecutar la siguiente línea:

```
Java -cp gdx.jar;gdx-natives.jar;gdx-backend-lwjgl.jar;gdx-backend-lwjgl-natives.jar;extensions\gdx-tools\gdx-tools.jar com.badlogic.gdx.tools.particleeditor.ParticleEditor
```

Una vez hecho esto se nos mostrará la siguiente pantalla:

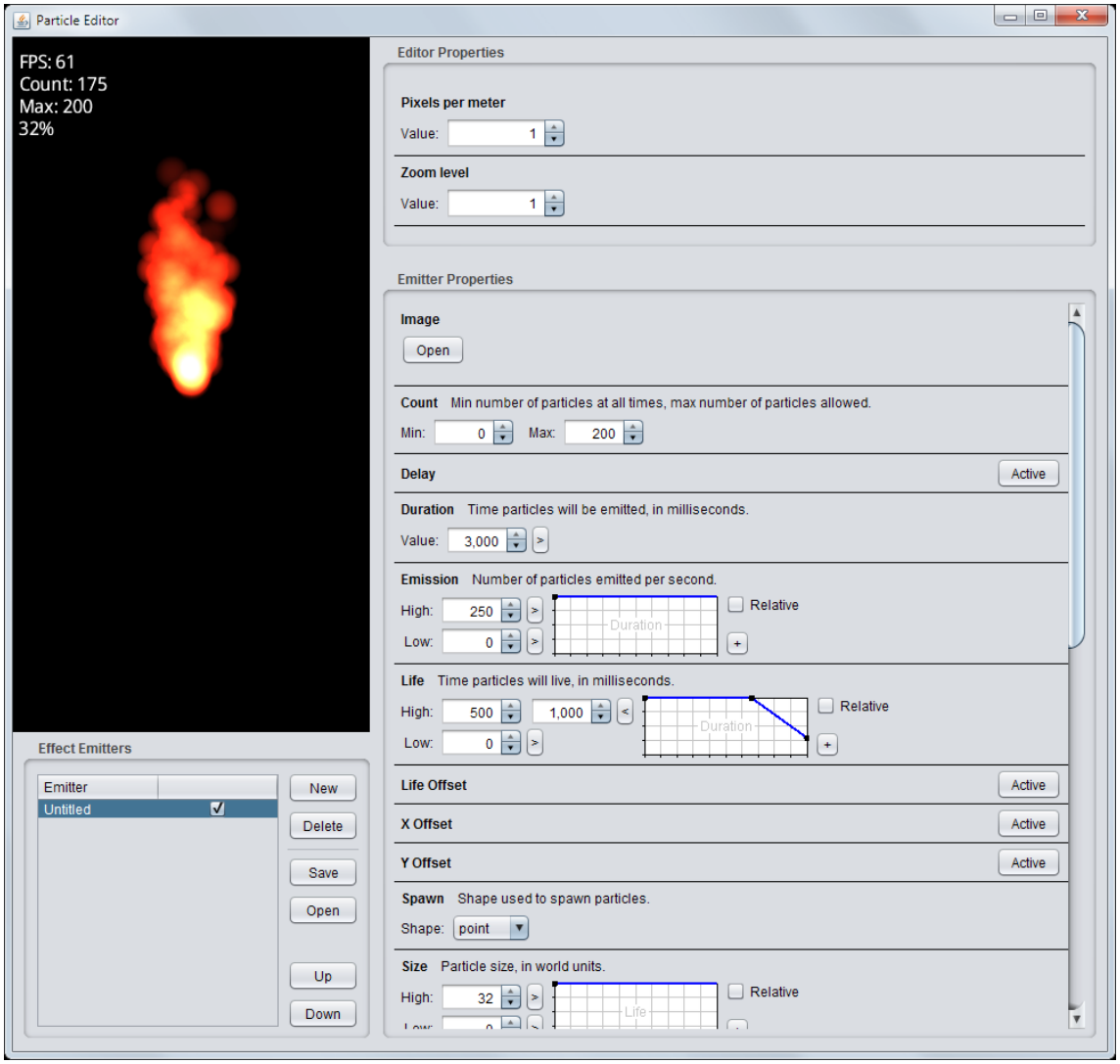

Ilustración 4 Pantalla inicial del ParticleEditor

En la zona superior izquierda podemos ver cómo quedará nuestro efecto, justo debajo se ve una lista con los emisores de partículas que va a tener nuestro efecto especial. La zona derecha se divide en dos partes, la de arriba controla la forma de representar nuestro efecto en el cuadro de la izquierda mediante varias variables. La zona que queda es la que controla todo lo relacionado con cómo las partículas del emisor se emiten y la forma en la que se van a comportar a lo largo del tiempo.

Son muchas las propiedades que se pueden cambiar para lograr el efecto deseado y por lo tanto no se va a entrar en qué significa cada una. Una vez conseguido crear el efecto especial que se desee, para poder implementarlo en el proyecto hay que guardarlo (tener en cuenta que no tienen una extensión de archivo determinada) y meterlo junto a la imagen de la partícula utilizada, en este caso una imagen que viene incluida en LibGdx. Después de esto se utiliza a través de una clase de LibGdx tal y cómo se ve a continuación:

```
ParticleEffect dyingParticles = new ParticleEffect();
dyingParticles.load(Gdx.files.internal("particles/desaparecer.pfx"),Gdx.files.internal("
particles"));
dyingParticles.setPosition(animatedCard.position.x + 1.3f,animatedCard.position.y+2.1f);
dyingParticles.start();
dyingParticles.allowCompletion();
```

El método load() carga el efecto especial y la partícula usada para realizar la animación, luego lo colocamos en el lugar que queremos que se vea y a continuación se usa el método start() para que comienza la animación del efecto. Podemos forzar que el efecto deje de mostrarse mediante el método allowCompletion().

Los dos efectos especiales que se han incluido en el juego están ligados a las animaciones que se producen durante una partida. Debido a esto se tuvo que tener especial cuidado tanto a la hora de iniciarlos como a la de terminarlos para que ambas cosas sucedieran en el momento adecuado.

Se tardó un poco más de lo esperado en la realización de esta tarea, en total se tardó 4 días. La principal razón de este retraso fue la ausencia de conocimientos sobre el ParticleEditor lo que retrasó el conseguir unos efectos especiales que se ajustaran a lo que se pretendía crear.

4.1.9 Comunicaciones cliente/servidor

A la hora de realizar esta tarea han surgido diversos problemas, entre ellos el tener que usar una nueva librería, y por lo tanto su realización ha llevado más tiempo del esperado.

Uno de los problemas ha sido encontrar la tecnología adecuada para realizar la comunicación. En principio se decidió usar los Socket estándar de Java dada la familiaridad que se tenía con ellos pero a la hora de ejecutar la aplicación en Android surgían errores y problemas (errores que en la versión de escritorio de la aplicación no aparecían), debido a estos problemas se decidió empezar a usar los Socket que vienen incorporados en LibGdx.

Se supuso que los Socket de LibGdx estarían preparados para poderse usar por igual en todas las plataformas que ofrece LibGdx. Tras la recopilación de datos sobre cómo manejarlos y varios intentos de ponerlos en funcionamiento se descubrió que los creadores de LibGdx no recomiendan su uso ya que no están implementados adecuadamente.

Después de todo esto se buscó información sobre cómo poder llevar a cabo esta tarea y se decidió usar la librería KryoNet, que es con la que finalmente se ha llevado a cabo la comunicación cliente/servidor intentando seguir en la medida de lo posible el esquema visto previamente de la interacción entre cliente y servidor.

Kryonet es una librería creada en Java enfocada especialmente para el desarrollo de comunicaciones cliente/servidor en dispositivos Android. Además

se trata de una librería que intenta simplificar en la medida de lo posible el envío y recepción de los datos (que normalmente vienen encapsulados en objetos).

Todas las clases de las que se van a transmitir objetos necesitan ser registradas tanto por el cliente como por el servidor, y la misma librería se encarga ella misma de serializar los objetos de esas clases.

Para realizar este paso se recurrió a una clase que contiene además un par de clases static que se usarán para pasarse información y el número del puerto por el que el servidor va a escuchar. Esta clase es la misma tanto para el servidor como para el cliente para así evitar problemas.

Una de las cosas que hay que tener en cuenta a la hora de usar Kryonet es que en el punto hacia el que se van a enviar objecto (ya sea el servidor o un cliente) ha de implementar un Listener. Lo que hace este Listener es crear un hilo en el que se esperan los envíos de datos. Cada vez que se recibe algo el Listener crea un hilo en el que se ejecutan los métodos que se hayan definido en el Listener.

Otro de los problemas a la hora de realizar la comunicación cliente/servidor ha sido configurar el router para que aceptara conexiones desde fuera de la red local al puerto en el que el servidor estaba funcionando. Tras informarse de los pasos necesarios para hacer esto posible y ejecutarlos, se debió cometer algún error y al no poder conectar con el servidor se buscaron otras opciones.

Como primera opción para solucionar este problema se intentó buscar algún host gratuito de aplicaciones Java, pero después de un par de días de búsqueda no se encontró nada que se adecuara a las necesidades del proyecto.

Al final se intentó configurar de nuevo el servidor y se logró que aceptara las conexiones y las redirigiera al ordenador y puerto en el que se ha situado finalmente el servidor de la aplicación.

Para la realización de esta tarea se consumió un total de 20 días de trabajo.

4.1.10 Desarrollo de las opciones de configuración del juego

En la planificación del desarrollo del proyecto aparece antes el tema de adquisición y adición de los sonidos al juego, pero al final se decidió realizar la parte de la configuración del juego y la ventana de inicio en primer lugar ya que dependiendo de la configuración los sonidos se reproducirán o no y si se reproducen el volumen dependerá de la configuración del volumen que se seleccione en las opciones del juego. Para hacerse una idea de cómo será la pantalla principal y de cuáles son las opciones incluidas en el juego se incluyen las siguientes imágenes:

Ilustración 5 Menú inicial

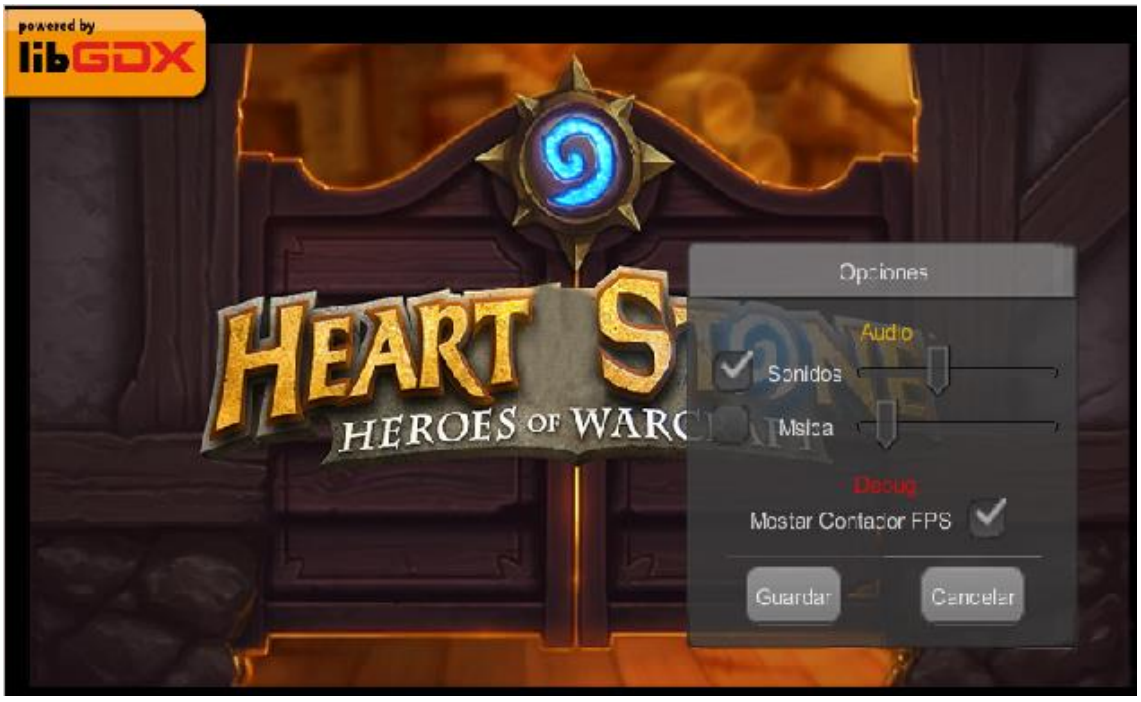

Ilustración 6 Opciones en el menú inicial

Tal y como se puede ver la pantalla principal contiene un fondo y un par de botones, el botón de play, que inicia una partida ,y el de options, que cuando se pulsa oculta los botones de play y options y muestra un cuadro de dialogo con las opciones de configuración del juego. La configuración del juego permite elegir entre reproducir los sonidos y la música y elegir el volumen en el que se reproducirá cada uno y una última opción que sirve para mostrar o no el contador

de fps del juego. Todas estas opciones se guardan en un archivo al que más tarde se accederá para tenerlas en cuenta durante la ejecución de la aplicación.

Todo lo relacionado con esta ventana se maneja desde la clase MenuScreen. Esta clase está diseñada a partir de una utilidad incluida dentro de LibGdx, se trata de Scene2D (UI).

Scene2D (UI) permite la creación de pantallas y el manejo de sus elementos por capas. Además incluye los típicos controles de usuario listos para usar como botones, sliders, checkbox, etc. También permite la creación y uso de nuevos controles.

Los controles están vinculados a skins que son una colección de recursos usados para dar estilo y para mostrar los controles en pantalla. Normalmente estas skins usan un texture atlas del que sacan las imágenes(recordar que un texture atlas está formado por un archivo png que contiene las imágenes empaquetadas y un archivo pack que contiene la información para acceder a las imágenes) y un archivo en formato JSON que contiene la información del estilo del control.

Afortunadamente LibGdx tiene ya una skin por defecto, lo malo es que el tamaño de los controles era demasiado pequeño. Lo que se hizo fue doblar el tamaño de la imagen png y cambiar toda la definición del archivo pack para que concordaran las zonas y tamaños en los que se encuentra cada elemento.

A partir de ahí el resto ya sólo fue ir colocando las cosas que se quería que aparecieran en pantalla.

La realización de esta tarea supuso 2 días de trabajo

4.1.11 Adquisición e implementación de los sonidos

Los sonidos que aparecen en la aplicación provienen de dos fuentes, sonidos creados con el programa Sfxr generator y sonidos grabados de diversos programas y editados con el programa Polderbits. La música que suena en el juego es el tema principal del juego original en el que está basada la aplicación, que se eligió debido a que era la que más encajaba.

Una vez conseguidos todos los sonidos y música que van a usarse en el juego a la hora de implementarlos se siguieron las recomendaciones de la gente de LibGdx y se incluyeron dos clases internas más a la clase Assets (que es la que se encarga de manejar los recursos de la aplicación sin tener que preocuparnos de irlos destruyendo y creando cada vez que queramos acceder a ellos) una para la música y otra para los sonidos, éstas clases incluyen un campo static por cada archivo al que se quiere acceder.

También se creó una clase especializada en manejar los sonidos y la música teniendo en cuenta las opciones de configuración del juego.

Para reproducir un sonido o música en LibGdx basta con crear una instancia de su respectiva clase (Sound o Music) y usar el método .play(). Este método lo usaremos a través de la clase que ha sido creada para manejar los sonidos.

Una cosa que ha dado problemas a la hora de reproducir los sonidos es que la mayoría de ellos se ejecutaban varias veces por segundo debido a que se reproducían en medio de las animaciones para que empezaran en el momento adecuado y se tuvieron que crear medios especiales para que solamente se reprodujeran una sola vez.

Todo este trabajo se hizo en 2 días

4.1.12 Creación de un tutorial del juego

Como parte final del desarrollo de la aplicación se decidió incluir un tutorial del juego para que cuando un usuario use la aplicación por primera tenga una noción básica de cuál es la mecánica del juego.

Este tutorial (incluido en el anexo III) consta de una serie de imágenes en las que se explica todo lo que se necesita saber para usar la aplicación, esto incluye una explicación de las reglas del juego.

Se decidió no incluir el tutorial dentro del juego debido a que su tamaño era demasiado elevado y podría ralentizar el juego.

Se tardó 1 día en llevar a cabo esta tarea

4.2 Casos de uso

En esta sección se explicará todo lo relacionado con la interacción con el usuario y la descripción de las acciones que puede llevar a cabo el usuario al usar la aplicación.

4.2.1 Diseño de la interfaz

Durante el desarrollo de la interfaz de la aplicación se ha tenido siempre en cuenta que su uso está enfocado a teléfonos móviles y por lo tanto las dimensiones de las pantallas en las que la aplicación se mostrará variarán tanto en tamaño como en la relación ancho/alto.

También se ha tenido especial cuidado en que la información que se le proporciona al usuario durante el desarrollo de una partida sea visible en las pantallas más pequeñas y no se pixele demasiado en las más grandes.

Además hubo que tener especial cuidado con las zonas en las que el usuario puede interaccionar (botones, imágenes, etc.) mediante toques de la pantalla, ya que necesitan tener un tamaño adecuado y estar separados lo suficiente para que no se produjeran problemas al tocar.

Todas estas cosas son las que se han tenido en cuenta a la hora de realizar los apartados, anteriormente explicados, de adquisición e implementación de las imágenes del juego.

A continuación se pueden ver dos capturas para hacerse una idea de la apariencia de la pantalla del juego:

Ilustración 7 El jugador recibe una carta mientras el enemigo saca una a la mesa

Ilustración 8 El jugador acaba de atacar a una carta del enemigo

4.2.2 Historias de usuario

Descripción de cada una de las acciones que se pueden llevar a cabo en la aplicación.

Número de historia: 1	
Pantalla	Menú inicial.
Situación	El usuario acaba de iniciar la aplicación.
Acción	Pulsar botón de opciones.
Efecto	Desaparecen los botones de jugar y opciones, aparece en su lugar un menú con las opciones del juego. Se cargan las opciones que el usuario haya cambiado anteriormente de un archivo y si todavía no se ha hecho ningún cambio en ellas se cargan las de por defecto.

Tabla 2 Historia de usuario 1

Número de historia: 2	
Pantalla	Menú inicial.
Situación	El usuario acaba de pulsar el botón de opciones.
Acción	Pulsar checkbox del sonido.
Efecto	Activa o desactiva los sonidos que se reproducen durante el juego.

Tabla 3 Historia de usuario 2

Número de historia: 3	
Pantalla	Menú inicial.
Situación	El usuario acaba de pulsar el botón de opciones.
Acción	Pulsar checkbox de la música.
Efecto	Activa o desactiva la música que se reproduce durante el juego.

Tabla 4 Historia de usuario 3

Número de historia: 4	
Pantalla	Menú inicial.
Situación	El usuario acaba de pulsar el botón de opciones.
Acción	Pulsar en la barra del slider del sonido o arrastrar el slider de la barra de sonido.
Efecto	Regula el volumen con el que se van a reproducir los sonidos del juego.

Tabla 5 Historia de usuario 4

Número de historia: 5	
Pantalla	Menú inicial.
Situación	El usuario acaba de pulsar el botón de opciones.
Acción	Pulsar en la barra del slider de la música o arrastrar el slider de la barra de la música.
Efecto	Regula el volumen con el que se va a reproducir la música del juego.

Tabla 6 Historia de usuario 5

Número de historia: 6	
Pantalla	Menú inicial.
Situación	El usuario acaba de pulsar el botón de opciones.
Acción	Pulsar en el checkbox “show fps counter”.
Efecto	Hace que se muestre/oculte un contador de frames por segundo en la esquina inferior derecha cuando se está jugando.

Tabla 7 Historia de usuario 6

Número de historia: 7	
Pantalla	Menú inicial.
Situación	El usuario acaba de pulsar el botón de opciones.
Acción	Pulsar en el botón de guardar.
Efecto	Se guarda en un archivo las opciones que el usuario ha escogido y se actualiza lo relacionado con el volumen de la música. El resto de opciones se tendrán en cuenta a la hora de reproducir los sonidos o mostrar el indicador de fps. Se cierra el menú de opciones y vuelven a aparecer los botones de jugar y opciones.

Tabla 8 Historia de usuario 7

Número de historia: 8	
Pantalla	Menú inicial.
Situación	El usuario acaba de pulsar el botón de opciones.
Acción	Pulsar en el botón de cancelar.
Efecto	Simplemente se cierra el menú de opciones y vuelven a aparecer los botones de jugar y opciones. Sin tener en cuenta las modificaciones que el usuario haya hecho en las mismas.

Tabla 9 Historia de usuario 8

Número de historia: 9	
Pantalla	Menú inicial.
Situación	Se muestra la pantalla del menú inicial.
Acción	Pulsar en el botón de jugar.
Efecto	Se cierra el menú inicial y se abre la pantalla de juego.

Tabla 10 Historia de usuario 9

Número de historia: 10	
Pantalla	Juego.
Situación	En cualquier momento durante la pantalla del juego.
Acción	Pulsar tecla atrás.
Efecto	La aplicación se desconecta del servidor y se vuelve a la pantalla del menú inicial.

Tabla 11 Historia de usuario 10

Número de historia: 11	
Pantalla	Juego.
Situación	El usuario acaba de iniciar la pantalla de juego.
Acción	Ninguna.
Efecto	La aplicación se conecta con el servidor, si no se puede conectar (el dispositivo no tiene acceso a internet o el servidor no está arrancado) se avisa al usuario. Si se consigue conectar se avisa al usuario de que se está buscando un enemigo contra el que enfrentarse.

Tabla 12 Historia de usuario 11

Número de historia: 12	
Pantalla	Juego.
Situación	No se ha podido conectar con el servidor.
Acción	Pulsar en la pantalla.
Efecto	Vuelve a la pantalla del menú inicial.

Tabla 13 Historia de usuario 12

Número de historia: 13	
Pantalla	Juego.
Situación	El servidor acaba de encontrar un enemigo.
Acción	Pulsar una carta.
Efecto	Si el jugador no tiene el turno no ocurrirá nada. En caso contrario la aparecerá una especie de halo alrededor de la carta para indicar que está seleccionada.

Tabla 14 Historia de usuario 13

Número de historia: 14	
Pantalla	Juego.
Situación	Es el turno del jugador, acaba de seleccionar una carta de la mano.
Acción	Pulsar en la zona de las cartas de la mesa (medio de la pantalla).
Efecto	Si se pulsa en una zona que no esté ocupada y el jugador tiene los suficientes cristales para jugar esa carta, la carta seleccionada se moverá a la zona de las cartas de la mesa. En caso contrario no ocurrirá nada.

Tabla 15 Historia de usuario 14

Número de historia: 15	
Pantalla	Juego.
Situación	Es el turno del jugador, acaba de seleccionar una carta de la mesa.
Acción	Pulsar en una carta del enemigo o en el enemigo.
Efecto	Si la carta no ha sido usada durante el turno, atacará a lo que esté en la zona pulsada (si es que hay algo en esa zona).

Tabla 16 Historia de usuario 15

Número de historia: 16	
Pantalla	Juego.
Situación	Es el turno del jugador.
Acción	Pulsar en el botón de pasar turno.
Efecto	El botón pasar turno pasará a ser de color gris y pondrá en él “turno del enemigo”. Desde este instante el enemigo será el que puede realizar las historias de usuario 13,14 y 15 hasta que el enemigo pase de nuevo el turno al jugador.

Tabla 17 Historia de usuario 16

Número de historia: 17	
Pantalla	Juego.
Situación	Es el turno del jugador.
Acción	Estar más de dos minutos sin pasar de turno.
Efecto	El mismo que la historia de usuario anterior. Esta acción es automática para evitar que un jugador se quede con el turno indefinidamente, paralizando de esta forma la partida.

Tabla 18 Historia de usuario 17

Número de historia: 18	
Pantalla	Juego.
Situación	La vida de uno de los jugadores ha llegado a cero.
Acción	Pulsar en la pantalla.
Efecto	En cuanto aparece el cartel indicando que el usuario ha ganado o perdido la partida, al pulsar en cualquier sitio de la pantalla se vuelve a la pantalla del menú inicial.

Tabla 19 Historia de usuario 18

4.2.3 Código fuente

Aunque ya ha aparecido algún fragmento de código durante la explicación del desarrollo de la aplicación en este apartado se mostraran aquellos fragmentos de código más representativos de la aplicación.

4.2.3.1 AbstractGameScreen

Se trata de una clase abstracta de la que heredarán las diferentes pantallas que se desarrollen en la aplicación y que define las acciones comunes entre ellas.

```
import game.Assets;
import com.badlogic.gdx.Game;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.assets.AssetManager;



public abstract class AbstractGameScreen implements Screen {

	protected Game game;

	public AbstractGameScreen (Game game) {
		this.game = game;
```

```
        }

        public abstract void render (float deltaTime);

        public abstract void resize (int width, int height);

        public abstract void show ();

        public abstract void hide ();

        public abstract void pause ();

        public void resume () {
                Assets.instance.init(new AssetManager());
        }

        public void dispose () {
                Assets.instance.dispose();
        }

}
```

Como se puede ver implementa la interfaz Screen de LibGdx que añade los métodos show() y hide() los cuales son llamados por Game y ocupan el lugar de los métodos create() y dispose(). Otro método importante es el render() que se encarga de actualizar y dibujar lo que sale en pantalla.

Cada pantalla tendrá que tener una instancia de Game para poder llamar al método setScreen() de Game y así podernos mover entre las distintas pantallas que conformen la aplicación. Además se han añadido dos líneas para asegurarnos de que los recursos de la aplicación sean correctamente cargados y eliminados.

4.2.3.2 El método render()

Sea cual sea la pantalla que se esté desarrollando el método render() va a ser el encargado de hacer dos cosas:

-Actualizar el estado del mundo del juego.

-Dibujar el mundo del juego ya actualizado.

Para lograr esto LibGdx se encarga de ejecutar este método constantemente varias veces por segundo cuando la pantalla se muestra. A continuación se muestra la implementación de este método en la clase GameScreen:

```
@Override
public void render(float deltaTime) {
        // Do not update game world when paused.
        if (!paused) {
                // Update game world by the time that has passed
                // since last rendered frame.
                worldController.update(Gdx.graphics.getDeltaTime());
        }
        // Sets the clear screen color to: Cornflower Blue
        Gdx.gl.glClearColor(0, 0, 0, 0xff / 255.0f);
        // Clears the screen
        Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
        // Render game world to screen
        worldRenderer.render();

}
```

Se puede ver que no es algo muy complicado pero hay que tener en cuenta que el orden en el que se ejecutan los métodos ha de ser siempre el mismo, primero actualizar, luego borrar toda la pantalla y por último dibujar el mundo actualizado.

Veamos más en concreto lo que sucede en el método renderGameObjects de la clase WorldRenderer que se encarga de dibujar lo que aparece en pantalla durante el desarrollo de una partida:

```
private void renderGameObjects() {
        worldController.cameraHelper.applyTo(camera);
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
        for(Sprite sprite : worldController.testSprites) {
                if(sprite!=null){
                        if(sprite.getTexture()!=null)
                        sprite.draw(batch);
                }
        }
        if(worldController.player.isYourTurn()){
                worldController.buttonEnemyTurn.draw(batch);
                worldController.buttonPassTurn.draw(batch);
```

```
        }else{
                worldController.buttonPassTurn.draw(batch);
                worldController.buttonEnemyTurn.draw(batch);

        }
        for(Card carta:worldController.cardsHand)
                carta.render(batch);
        for(Card carta:worldController.cardsEnemy)
                carta.render(batch);
        for(Card carta:worldController.cardsTable)
                carta.render(batch);
        if(worldController.dyingCard){
                worldController.dyingParticles.draw(batch);
                worldController.dyingParticles2.draw(batch);
        }
        if(worldController.attackSpecialEffect){
                worldController.attackParticles.draw(batch);
                worldController.attackParticles2.draw(batch);
        }
        batch.end();


}
```

Las dos primeras líneas se encargan de hacer saber al batch que todos los draw() que se van a hacer a continuación seguirán las reglas de la proyección ortográfica, dicho de otra de manera, se dibujará en 2D usando la posición y tamaño de la cámara.

Entre los método batch.begin() y batch.end() es dónde han de ir dibujándose todos los elementos que se ven. Siempre hay que llamar a begin() antes de dibujar nada y a end() cuando se termina de dibujar y además no se pueden anidar porque si no puede dar problemas. Además es conveniente que siempre que se pueda se dibujen todos los objetos que usen la misma cámara dentro del mismo bloque de begin() end() para que el rendimiento de la aplicación sea mayor.

4.2.3.3 El servidor

Ya se ha explicado anteriormente cómo funciona un servidor que utiliza la librería Kryonet, ahora se va a ver en concreto cómo se ha implementado:

```
server.addListener(new Listener() {
            public void received(Connection c, Object object) {
                // We know all connections for this server are actually GameConnections.
                GameConnection connection = (GameConnection) c;

                if (object instanceof RegisterName) {
                    // Ignore the object if the name is invalid.
                    String name = ((RegisterName) object).name;
                    if (!name.equals("Start")) {
                        return;
                    }
                    // Store the name on the connection.
                    connection.name = name;

                    //Bind the connection to an enemy if its possible
                    if (checkFreeConnections()) {
                        connection.firstTurn = false;
                        connection.enemy = getFreeConnection();
                        removeFreeConnections();
                        Connection[] connections = server.getConnections();
                        for (int i = 0; i < connections.length; i++) {
                            if (connections[i].equals(connection.enemy)) {
                                GameConnection enemy = (GameConnection) connections[i];
                                //the enemy of this player is this player himself
                                enemy.enemy = c;
                                break;
                            }
                        }
                        Network.ActionMessage sendAction = new Network.ActionMessage();
                        sendAction.action = Network.ActionMessage.START;
                        connection.enemy.sendTCP(sendAction);
                        connection.sendTCP(sendAction);

                    } else {
                        connection.firstTurn = true;
                        addFreeConnections(c);
                    }
```

```
            return;
        }

        if (object instanceof ActionMessage) {
            ActionMessage msg = (ActionMessage) object;
            if (msg.action.equals(ActionMessage.START)) {
                Player player = new Player(!connection.firstTurn);
                sendCard((GameConnection) connection.enemy);
                sendCard((GameConnection) connection.enemy);

                if (connection.firstTurn) {
                    player.setCrystalsLeft(0);
                } else {
                    player.setCrystalsLeft(1);
                }

                connection.enemy.sendTCP(player);
            } else if (msg.action.equals(ActionMessage.PASS_TURN)) {
                sendCard((GameConnection) connection.enemy);
                connection.enemy.sendTCP(msg);
            }
        }

        if (object instanceof Stats) {
            Stats card = (Stats) object;
            //if (card.getCardAction().equals(Stats.CARD_ACTION_NEW_ENEMY_CARD)) {
            connection.enemy.sendTCP(card);
            //} else if (card.getCardAction().equals(Stats.CARD_ACTION_ATTACKED_CARD)) {
            //} else if (card.getCardAction().equals(Stats.CARD_ACTION_ATTACKING_CARD)) {
            //} else if (card.getCardAction().equals(Stats.CARD_ACTION_ATTACK_PLAYER)) {
            // }

            return;
        }

        if (object instanceof Player) {
            connection.enemy.sendTCP(object);
        }
    }

    public void disconnected(Connection connection) {
```

```
                GameConnection c = (GameConnection) connection;
                if(c.enemy!=null){
                    Network.ActionMessage sendAction = new Network.ActionMessage();
                    sendAction.action=ActionMessage.DISCONNECT;
                    c.enemy.sendTCP(sendAction);
                }else{
                    notYetInGame.remove(c);
                }
            }
        });
```

Lo que se ve en este fragmento de código es prácticamente el funcionamiento completo del servidor. Básicamente se le crea un listener al servidor que es el que se encarga de recibir las conexiones.

El método received produce un evento cada vez que un cliente envía algún objeto al servidor, luego se identifica la clase al que pertenece el objeto y a partir de ahí ya se codifican las acciones que ha de realizar el servidor con él. Algunos objetos simplemente se reenvían al enemigo, otros se procesan antes de que el servidor realice ninguna acción.

El método disconnected es en el que nos vamos a encargar de manejar las desconexiones del servidor para su correcto funcionamiento. En esencia lo que hacemos es que cuando alguien se desconecta si todavía no tenía a nadie con el que enfrentarse se le quita de la lista de personas disponibles para enfrentarse. Si ya estaba emparejado con otra persona se le manda a esa persona una señal de que su enemigo se ha desconectado.

4.2.3.4 Método moveCardToTable()

Para ver un poco la forma en la que se actualiza el juego se verá el método moverCardToTable() que es el encargado de determinar si un jugador quiere mover una de las cartas que tiene en la mano a la mesa:

```
// Drag a selected card on the hand to the table
    private void moveCardToTable(float deltaTime) {
        if (player.isYourTurn()) {
            if (Gdx.input.justTouched() && cardsTable.size() < 5) {
                for (int i = 0; i < cardsHand.size(); i++) {
                    if (cardsTable.size() > 0) {
                        // if there is another card on the table set the
                        // table size according to the number of free space
                        table.set(
                                Constants.CARD_POSITION_TABLE[cardsTable.size()].x,
```

```
                Constants.CARD_POSITION_TABLE[cardsTable.size() - 1].y,
                Constants.CARD_WIDTH * (5 - cardsTable.size()),
                Constants.CARD_HEIGHT);
        } else {
            table.set(-9, -1.9f, Constants.CARD_WIDTH * 5,
                    Constants.CARD_HEIGHT);
        }
        if (cardsHand.get(i).isSelected()
                && (cardsHand.get(i).getCrystalCost() <= player
                .getCrystalsLeft())) {

            if (table.contains(
                    (float) (Gdx.input.getX() - (Gdx.graphics
                    .getWidth() / 2))
                    / (Gdx.graphics.getWidth() / 2) * (9),
                    (float) (Gdx.input.getY() - (Gdx.graphics
                    .getHeight() / 2))
                    / (Gdx.graphics.getHeight() / 2)
                    * (6)
                    * -1)) {
                cardsHand.get(i).bounds.x = cardsHand.get(i).position.x;
                cardsHand.get(i).bounds.y = cardsHand.get(i).position.y;
                animatedCard = cardsHand.get(i);
                cardsHand.get(i).setUsed(true);
                cardsHand.get(i).setSelected(false);
                player.setCrystalsLeft(player.getCrystalsLeft()
                        - cardsHand.get(i).getCrystalCost());
                player.setCardsOntable(player.getCardsOntable() + 1);
                player.setCardsOnHand(player.getCardsOnHand() - 1);
                setGuiNumbers(
                        testSprites[Constants.GUI_POSITION_CRISTALES_TENS],
                        testSprites[Constants.GUI_POSITION_CRISTALES_UNITS],
                        player.getCrystalsLeft());
                testSprites[1].setPosition(11, 11);
                cardsTable.add(cardsHand.get(i));
                cardsHand.remove(i);
                moveToTable = true;
                reorderHandCards = true;
                AudioManager.instance
```

```
                                .play(Assets.instance.sounds.new_card);
                        // send data to server
                        Stats sendCard = getStatsFromCard(animatedCard);
                        sendCard.setCardAction(Stats.CARD_ACTION_NEW_ENEMY_CARD);
                        clientSocket.sendTCP(sendCard);
                        return;

                    }
                }
            }
        }
    }
```

En esencia lo que hace el método es ir comprobando que se cumplen todas las condiciones necesarias para saber que el jugador puede mover una carta y además que quiere moverla.

Primero hay que asegurarse de que el jugador tenga el turno (no se puede hacer nada cuando no se tiene el turno), luego se comprueba que el jugador ha tocado la pantalla y que hay espacio en la zona de la mesa para otra carta, si es así se prosigue definiendo el tamaño de un objeto de la clase Rectangle el cual se usa para comprobar que se ha pulsado en una zona de la mesa que no esté ocupada.

Por último se comprueba que haya alguna carta de la mano seleccionada y que dicha carta cueste moverla a la mesa como mucho los cristales que le quedan al jugador.

Comprobado todo lo anterior se actualiza el estado del juego se cambia la variable moveToTable a true para que se empiece a ejecutar un método que haga la animación de mover la carta elegida a la mesa y se envían los datos necesarios al servidor para que la acción se refleje en el jugador enemigo.

5. Fase de pruebas

Las pruebas que se han llevado a cabo para comprobar el correcto funcionamiento de la aplicación han ido realizándose a la par que se iban desarrollando las diversas partes de la aplicación.

La principal razón de que esto se haya hecho de ésta manera es que así se puede evitar que se olvide de probar alguna parte. De todas formas también se han vuelto a realizar pruebas de cada parte una vez terminada la aplicación para comprobar el correcto funcionamiento de la misma.

5.1 Pruebas realizadas

La mayoría de las pruebas han consistido en simples comprobaciones de que el código recién implementado funcionara cómo se esperaba. Las pruebas que se van a comentar a continuación fueron un poco más especiales.

5.1.1 Visualización de las imágenes

Estas pruebas se realizaron en cuanto se consiguió que se mostraran las imágenes. Se comprobó, mediante el emulador, que las imágenes fueran lo suficientemente nítidas en los distintos tamaños de pantalla, que la colocación fuera también la correcta y que los números y nombres de las cartas se vieran bien.

De la realización de estas pruebas apareció uno de los mayores problemas que han surgido durante el desarrollo del proyecto, la modificación de las imágenes para conseguir una correcta visualización de las mismas.

5.1.2 Lógica del juego y ejecución de las animaciones/efectos especiales

Para que se ejecuten las animaciones creadas se han tenido que crear una serie de métodos que comprueben tanto la entrada de datos del usuario como que los datos introducidos por el usuario siguen la lógica del juego y por lo tanto la acción se puede realizar.

Estos métodos tienen que realizar bastantes comprobaciones por lo tanto a la hora de codificarlos era bastante probable que hubiera errores en algunos de ellos y por ello se decidió realizar por cada animación un conjunto de pruebas que determinaran si las animaciones y los efectos especiales seguían adecuadamente la lógica del juego.

Se fue probando la entrada del usuario y verificando que la respuesta de la aplicación era la esperada en diversos escenarios. Los escenarios son básicamente cada una de las historias de usuario que están situadas en la pantalla del juego y como entrada de usuario se intentó cubrir el máximo de posibilidades, que, al tratarse de teléfonos móviles, en esencia es pulsar en diversos lugares de la pantalla.

Además se hicieron pruebas para verificar que el comportamiento de las animaciones y los efectos especiales (velocidad a la que se mueven los objetos, puntos de retorno o de parada) fuera el deseado.

5.1.3 Comunicaciones cliente/servidor

Las pruebas que se hicieron para comprobar el correcto funcionamiento de las comunicaciones entre cliente y servidor se hicieron a la vez que las pruebas de la lógica del juego y las animaciones ya que la mayoría de las acciones que puede realizar un jugador tienen que reflejarse en la pantalla del enemigo.

Se siguió el mismo esquema que a la hora de realizar las pruebas del apartado anterior pero en ésta ocasión no se comprobaba lo que pasaba en el cliente que realizaba la entrada de datos a la aplicación. Se comprobaba que los datos que se enviaban al servidor y los que recibía el enemigo eran los mismos y que la acción se reflejaba correctamente.

También se realizaron pruebas para comprobar que el servidor funcionaba correctamente a la hora de manejar las desconexiones de los clientes.

Hubiera sido interesante realizar una prueba de esfuerzo del server para ver cómo se comportaba cuando había muchos clientes conectados al mismo tiempo, pero no se pudo hacer precisamente por la falta de clientes.

5.1.4 Sonidos, música y las preferencias del juego

Se hicieron pruebas tanto para comprobar que los sonidos y la música se reproducían como para comprobar que se reproducían en el momento correcto.

Además se hicieron pruebas para comprobar que las opciones de configuración que tiene el juego se tenían en cuenta a la hora de reproducir sonidos y música y a la hora de mostrar el contador de fps.

Las primeras pruebas que se hicieron de los sonidos revelaron que era necesario incluir código extra para evitar que se reprodujeran varias veces seguidas.

Las siguientes pruebas consistieron en comprobar que los sonidos se reproducían en el momento adecuado y consecuentemente ir ajustando poco a poco el momento en el que cada sonido se tenía que reproducir.

5.1.5 Pruebas de la aplicación en distintos dispositivos

Para finalizar las pruebas se ha intentado comprobar que la aplicación funciona correctamente en el máximo de dispositivos posibles.

Se ha probado en un total de 14 dispositivos Android que incluyen desde la versión 2.3 hasta la 4.1 de Android.

De los 14 dispositivos ha funcionada correctamente en 12 de ellos. La aplicación dio problemas a la hora de cargar las imágenes de la pantalla del juego en los otros dos dispositivos. Se cree que la razón de estos problemas está ligada a la poca memoria RAM de esos dispositivos.

Durante la realización de estas pruebas se descubrió que algunos dispositivos cierran las conexiones 3g cuando se lleva cierto tiempo (alrededor de 1 minuto) sin recibir o enviar datos de la conexión y se incluyó un método para evitar que esto sucediera.

6. Conclusiones finales

6.1 Reflexión sobre los resultados obtenidos

El desarrollo de esta aplicación ha sido complejo ya que se han tenido que usar librerías, tecnologías y programas con los que no se estaba familiarizado. Además como ya se ha comentado en apartados anteriores los teléfonos móviles poseen un rango muy amplio de tamaños de pantalla lo que supone un problema a la hora de diseñar las pantallas del juego para que la visibilidad de los elementos que la componen sea la adecuada en una pantalla pequeña y la calidad no se resintiera en las pantallas grandes.

Pero a pesar de los problemas que han ido apareciendo y a la falta de conocimientos específicos del tema a tratar se ha conseguido llevar a cabo el proyecto que se tenía en mente al completo.

He mejorado mis conocimientos sobre el uso de photoshop y sobre el desarrollo de juegos utilizando LibGdx, además me he topado con la librería Kryonet que podría ser útil en futuros proyectos. Por lo tanto el desarrollo de este proyecto se puede calificar como una experiencia muy enriquecedora tanto en el ámbito profesional como el personal.

6.2 Grado de cumplimiento de los objetivos fijados

Los objetivos principales que se establecieron al comienzo de este proyecto se han cumplido por completo. Básicamente se trataba de crear un juego en el que dos personas se enfrentasen cada uno desde su teléfono móvil.

En mi opinión los objetivos que aparecen en la sección 1.3 y que eran lo que se pretendía hacer al empezar este proyecto se han cumplido al 100%.

6.3 Propuesta de ampliaciones futuras del sistema implementado

A pesar de que el juego está completo, incluso puede que sea entretenido, es posible que el jugador necesite de una motivación extra más allá de simplemente vencer a un oponente desconocido. Por esta razón se piensa que las posibles ampliaciones futuras deberían ir encaminadas a llenar ese vacío de motivación que se acaba de comentar.

Entre las posibles mejoras al juego que siguen esta línea de pensamiento cabe destacar las siguientes:

- Dejar una opción para que los usuarios puedan retarse a duelos entre ellos.
- Eliminar parte del anonimato de los usuarios permitiendo que puedan ser identificados por un “nickname”.
- Crear un ranking con los jugadores que tenga en cuenta de forma equilibrada el número de victorias y el porcentaje de victorias.
- Añadir más variedad de cartas.
- Permitir a los usuarios personalizar el mazo de cartas que va a utilizar.
- Ofrecer puntos por cada victoria, con esos puntos se podrían adquirir más cartas.

Además de todo esto, también se podría mejorar el sistema de juego añadiendo al juego cartas que tengan efectos especiales, de esta forma las partidas serían mucho más variadas y estratégicas.

Aunque todas estas ideas de mejora serían muy interesantes de implementar, hay que tener en cuenta que probablemente cambiarían prácticamente la forma en la que el proyecto funciona, ya que supondrían cambios en la lógica, en las imágenes, en los sonidos, incluso puede que hiciera falta implementar una forma para acceder a una base de datos con las estadísticas de cada carta.

Todo ello supondría un tiempo del que no se disponía a la hora de desarrollar este proyecto, pero si se podría intentar hacer como proyecto personal.

7. Documentación del sistema desarrollado

7.1 Manual de instalación

A continuación se procederá a explicar cómo instalar y ejecutar el servidor del juego y el cliente.

7.1.1 El cliente

Dicho sea, instalar el cliente no es complicado. Una vez generado el fichero .apk del proyecto heartstone-android con eclipse (basta con ejecutar el proyecto hearstone-android en un teléfono o en el emulador). Es tan fácil como pasar ese fichero al teléfono móvil en el que se desee instalar y ya desde el teléfono pulsar en el fichero y seguir las instrucciones.

Hay que tener en cuenta que es posible que la configuración de seguridad que se tenga en el teléfono puede bloquear la instalación de aplicaciones que no provengan de la Play Store de Google. Si esto ocurre tenemos que cambiar dicha configuración para que nos permita instalar la aplicación.

7.1.2 El servidor

Como requisitos para instalar el servidor se necesita tener instalada la máquina virtual de Java 1.7 en el ordenador y poner en la variable de entorno PATH la ruta en la que se tiene instalado Java. También se tiene que generar desde Netbeans el fichero ejecutable .jar del proyecto

Para instalar el servidor en sí no hay que hacer nada. Se reparte en una carpeta que contiene un fichero ejecutable .jar y una subcarpeta con las librerías que se necesiten.

7.2 Manual de uso

Para ejecutar el servidor basta con ejecutar el fichero .jar del que se ha hablado en la sección anterior desde la línea de comandos. Para ello nos posicionamos en la carpeta en la que esté situado el fichero e introducimos la siguiente línea:

```
java –jar HeartStone_Server.jar
```

Esto arrancará el servidor, que no requiere de nada más para que realice su tarea. Si se quiere parar el servidor se puede pulsar Ctlr+C en la línea de comandos.

En el caso del cliente ya se ha explicado en otro apartado que se creó un tutorial sobre cómo jugar las partidas que va incluido en uno de los anexos. Dicho tutorial contiene una serie de imágenes que explican paso a paso la lógica del juego, la información que aparece en la pantalla y las posibles acciones que el usuario puede realizar.

8. Problemas y dificultades en la realización del proyecto

Durante el desarrollo de cualquier proyecto surgen problemas y dificultades, en el caso de este proyecto no ha sido distinto y, aunque ya se ha explicado en apartados anteriores algún que otro problema, en este apartado de la memoria se explicarán cuáles han sido los mayores problemas y dificultades que han aparecido.

8.1 Las imágenes

Durante toda la realización del proyecto las imágenes que se han usado en él han sido un quebradero de cabeza.

Tal y cómo se ha explicado en otros apartados la intención a la hora de realizar este proyecto es que la aplicación pudiera visualizarse en la mayoría de los teléfonos móviles con sistema operativo Android.

Esto supone un gran esfuerzo ya que abarca una cantidad bastante elevada de resoluciones, formatos y tamaños de pantalla. Además hay que tener en cuenta que cuanto mayor es el tamaño de una imagen, mayor es la cantidad de memoria que ocupa y por lo tanto el rendimiento de la aplicación disminuirá.

Para solventar estos problemas han tenido que modificar el tamaño de las imágenes y también se han tenido que retocar los contenidos de muchas de las imágenes para adaptarlas a lo que se pretendía mostrar al usuario.

La misma resolución del problema ha supuesto una dificultad añadida, ya que no se poseían conocimientos sobre tratamiento y edición de imágenes.

8.2 El texto en pantalla

Aunque se ha pasado este tema un poco de puntillas a la hora de explicar el desarrollo del proyecto, ya que al final solamente se usa texto en pantalla para un par de cosas poco importantes, el mostrar una cadena de texto en pantalla ha supuesto grandes dificultades.

Para poder representar cadenas de texto en pantalla se necesita acceder a dos archivos un .fnt y una imagen que contenga todos los caracteres que se van a representar.

Esta parte supone hacer uso de un nuevo programa, el Hiero que permite crear los dos archivos necesarios indicándole el tipo de letra y el tamaño de la misma.

Donde surge el mayor problema es a la hora de representar el texto en la pantalla. Para que el texto sea visible en pantalla se necesita crear una nueva

cámara de distintas dimensiones a la cámara que se estaba usando para dibujar el resto de imágenes.

En un principio todo lo referente a los datos de cada jugador (vida de los jugadores, cristales restantes, etc.) iba a ser mostrado en pantalla usando texto ya que era la forma más simple de hacerlo.

Una vez posicionados todos los textos que se iban a mostrar en pantalla se descubrió que dependiendo del tamaño de pantalla los textos aparecían en una zona u otra.

Se intentó hacer una relación entre las dos cámaras que se estaban usando para que los textos aparecieran siempre en una posición independientemente del tamaño de la pantalla, pero tras varios intentos y pruebas se decidió abandonar los textos para mostrar en pantalla los datos de los jugadores (los textos se siguen usando para mostrar varios mensajes al jugador, pero éstos mensajes no importa si están un poco más a la izquierda o a la derecha de donde deben estar).

Cómo solución final para mostrar los datos de los jugadores se recurrió a imágenes con el texto que se quería mostrar ya que de esta manera se podía seguir usando la misma cámara que para el resto de los objetos que aparecían en pantalla y la posición no cambiaba al cambiar el tamaño de la pantalla. Se tuvo que crear un sistema un poco más rebuscado para poder mostrar las partes que no eran estáticas, los números.

Para representar los números, aunque es más simple, es evidente que no se iban a crear imágenes de todos los números del 0 al 30 (vida máxima de un jugador) ya que eso supone un gasto extra de recursos para el dispositivo. Por lo tanto se recurrió a un sistema en el que se colocaban dos imágenes usando los números que ya se tenían para la vida de las cartas.

Este sistema consiste en colocar las dos imágenes una al lado de la otra representado la primera las decenas y la segunda las unidades del dato a mostrar. Se crearon métodos específicos para realizar este proceso para cada dato que se quería mostrar.

8.3 Entrada de datos del usuario

Ya se ha explicado en otras ocasiones que las dimensiones del mundo del juego dependen de las dimensiones que se le dé a la cámara que se va a usar para representar ese mundo.

En el caso de este proyecto estas dimensiones se decidieron que fueran de 16 de ancho por 12 de alto. Debido a esto todos los tamaños de los objetos que aparecen en el mundo del juego dependen del tamaño del mundo en el que se encuentran.

El problema surgió a la hora de detectar los toques de pantalla del usuario ya que las coordenadas que LibGdx proporciona cuando se detecta un toque en pantalla dependen de la resolución de la pantalla.

La verdad es que la fuente exacta de este problema se tardó en identificar, era evidente que las coordenadas de lectura no se correspondían con las del juego pero se tuvieron que hacer varias consultas para poder determinar cuáles eran en concreto las coordenadas que se nos estaban proporcionando cuando el usuario toca la pantalla.

Para resolver este problema se realizó una conversión entre la resolución del dispositivo y el tamaño del mundo del juego.

8.4 Comunicaciones cliente/servidor

Las comunicaciones entre cliente y servidor han sido una de las mayores dificultades a la hora de desarrollar este proyecto, tanto a la hora de elegir la tecnología adecuada para realizar la comunicación como a la hora de implementarla.

Descubrimiento de Kryonet

El primer problema con el que tuvo que lidiar fue el cómo realizar la comunicación. En un principio se creó un sistema de comunicación que funcionaba con Sockets de Java.

Aunque estaba por depurar y no estaba completo se descubrió que a pesar de que funcionaba en la versión de escritorio (recordar que LibGdx es multiplataforma) no se consiguió hacerlo funcionar en la versión Android (recordar también que este proyecto está enfocado a crear un juego para Android).

Se desechó todo el código relacionado con los Sockets de Java y se creó un nuevo sistema de comunicación usando los Sockets que incluye LibGdx. Se pensó que estarían preparados para realizar la tarea que se necesitaba pero tras informarse un poco más sobre ellos se descubrió que todavía estaban por implementar correctamente.

Al final se encontró una librería especializada en la comunicación en red especializada en Android pero que a la vez podía ser utilizada en escritorio. El nombre de esta librería es Kryonet.

Para usar esta librería basta con importar los .jar que la componen a nuestro proyecto, o en su defecto un .jar que incluye toda la librería al completo. Una vez hecho esto se crearon tanto el cliente como el servidor usando esta librería.

Aunque Kryonet intenta simplificar en todo lo posible las comunicaciones en red, requiere de un tiempo de aprendizaje ya que la forma en la que se hacen las

cosas difiere bastante de cómo lo hacen por ejemplo los Sockets de Java (que era con lo que se estaba familiarizado). Una vez entendido cómo usar Kryonet no supuso mayores complicaciones el crear las comunicaciones entre cliente y servidor.

Acceso al servidor

Otro de los grandes problemas fue el realizar la configuración adecuada para que los clientes pudieran conectar con el servidor desde internet.

Como es lógico las primeras pruebas de la comunicación en red se hicieron en local para evitar quebraderos de cabeza y posibles errores que estuvieran vinculados con cosas que no tuvieran que ver con lo que se estaba desarrollando en ese momento.

Una vez se comprobó que todo funcionaba adecuadamente se intentó configurar el router para poder conectar los clientes con el servidor a través de internet.

En principio basta con entrar en la configuración del router y crear un re-direccionamiento de los datos que se reciben desde el exterior de la red en el número de puerto con el que va a intentar conectar los clientes y la dirección ip y puerto en el que está funcionando el servidor dentro de la red local.

Se intentó hacer esto con el router del que se disponía pero, la primera vez que se intentó, el router no dejaba hacerlo. Más tarde se descubrió que había que “engañar” al router para poder configurarlo de la manera que se pretendía. Se tuvo que indicar al router que se pretendía abrir el puerto para una aplicación http y luego simplemente se introdujeron los datos del servidor en lugar de los datos de una aplicación de ese tipo.

Debido a que la primera vez que se intentó configurar el router no se pudo realizar la configuración adecuadamente, se buscaron otras alternativas para que se pudiera acceder desde fuera de la red local al servidor.

Se invirtieron un par de días en buscar algún alojamiento gratuito de aplicaciones Java o algún servidor dedicado gratuito. Al final no se encontró nada que se adecuara a la forma en la que se había desarrollado el servidor ya que aunque se encontraron lugares en los que alojar aplicaciones Java se restringía el uso a aplicaciones de la rama Web de Java.

Otra de las alternativas fue intentar colocar el servidor en el ordenador de otra persona y aunque se conseguía conectar con el servidor, no era la situación más adecuada ya que se dependía de otra persona para poder usarse ese servidor.

Se investigó un poco más con el router que se posee y se consiguió finalmente configurarlo adecuadamente tal y como se ha comentado anteriormente.

9. Bibliografía

Durante la realización de este proyecto se han consultado diversas fuentes de información que han ayudado tanto a la realización del mismo como a la realización de esta memoria.

A continuación se expone una lista con las más utilizadas:

- https://github.com/EsotericSoftware/kryonet
- http://libgdx.badlogicgames.com/documentation.html
- http://stackoverflow.com/questions/tagged/libgdx
- http://es.wikipedia.org/wiki/Wikipedia:Portada
- http://gafferongames.com/networking-for-game-programmers/

También se han realizado diversas búsquedas en https://www.google.es/ para realizar consultas o intentar resolver alguna duda.

10. Anexos

Además del presente documento se adjuntan a él los siguientes anexos.

I. Proyecto heartstone completo.
II. Proyecto heartstone-server completo.
III. Tutorial del juego.
IV. Imágenes usadas en el proyecto.
V. Sonidos y música usados en el proyecto.
VI. Presentación del proyecto.
VII. Javadoc del cliente y el servidor.
VIII. Librería Kryonet.
IX. Librería LibGdx.