Trabajo de Fin de Grado

GRADO DE INGENÍERIA INFORMÁTICA

Facultad de Matemáticas e Informática

Universidad de Barcelona

Desarrollo de videojuegos con LIBGDX

Iñaki Fernández Palomo

Director: Inmaculada Cristina Rodríguez Santiago

Realizado en: Departamento de Matemáticas e Informática

Barcelona, 22 de junio de 2017

Agradecimientos

Antes de comenzar con el proyecto quería dar las gracias a todas las personas que han colaborado y me han ayudado en este proyecto.

Primero de todo dar las gracias a mi tutora Inma Rodríguez Santiago, quien me ha apoyado en todo momento y se ha interesado por todo el desarrollo del proyecto aportando nuevas ideas para mejorar el resultado final.

También quiero dar las gracias a todos los profesores de la universidad por su implicación y por todo el apoyo que me han dado durante mi trayecto como estudiante universitario.

Por último, agradecer el apoyo de las personas que me rodean y me han estado apoyando para poder conseguir mi objetivo.

Índice de contenido

1. Introducción 7
2. Objetivos 8
3. Antecedentes 9
3.1 Tecnologías utilizadas 9
3.2 Análisis de la competencia: Frameworks y Engines 12
3.3 Trabajos relacionados: Juegos LIBGDX 14
4. Conceptos básicos de LIBGDX 16
5. Análisis 18
5.1 Mecánica del juego *AirHockey* 18
5.2 Casos de uso 18
5.2.1 Diagrama de casos de uso 19
5.3 Requisitos 23
6. Diseño 25
6.1 Diagramas de secuencia 25
6.2 Diagramas de clase 29
7. Implementación 32
7.1 Arquitectura de LIBGDX 32
7.1.2 Arquitectura del juego 32
7.2 Detalles generales de implementación 36
7.2.1 Configuración proyecto Android (AndroidManifest.xml) 36
7.2.2 Mantener el dispositivo activo 37
7.2.3 Cámara 37
7.2.4 Splash Screen y Animaciones 38
7.2.5 Escena 38
7.2.6 Texto 41
7.2.7 Multijugador 42
7.2.8 IA del mazo 47
7.2.9 Audio 47
7.3 Detalles de implementación de GUI 48
7.3.1 Adaptación de la GUI a diferentes pantallas 48
7.3.2 HUD 49
7.3.3 Sombras 49
8. Resultados 50

8.1 Resultados del videojuego *AirHockey* 50
8.2 Benchmark LIBGDX VS Unity VS AndEngine 56
9. Conclusiones y trabajo futuro 61
10. Referencias 63
11. Apéndice manual de desarrollador 65

Índice de tablas y de figuras

Figura 3.1 The Spartan War. 14
Figura 3.2 Village and Farm. 14
Figura 3.3 Keep IT Alive. 15
Figura 3.4 Glow Hockey. 15
Figura 3.5 Partida multijugador. 15
Figura 5.1 Diagrama de casos de uso. 19
Figura 6.1 Secuencia comenzar nueva partida. 25
Figura 6.2 Secuencia crear partida como servidor. 26
Figura 6.3 Secuencia buscar partida como cliente. 27
Figura 6.4 Secuencia visualizar instrucciones de juego. 28
Figura 6.5 Diagrama de clases. 29
Figura 7.1 Estructura proyecto LIBGDX. 32
Figura 7.2 Diagrama de arquitectura del juego AirHockey. 33
Figura 7.3 Ejemplo de un widget tipo label. 33
Figura 7.4 Ejemplo de un widget tipo botón. 33
Figura 7.5 Ejemplo de un widget de tipo tabla. 34
Figura 7.6 Ejemplo de skin. 34
Figura 7.7 Ejemplo de imagen. 34
Figura 7.8 Ejemplo de scrollpane. 35
Figura 7.9 Sistema coordenadas LIBGDX. 39
Figura 7.10 Escena mono jugador. 40
Figura 7.11 Escena multijugador. 40
Figura 7.12 Imagen de fondo del juego AirHockey. 41
Figura 7.13 Imagen con los caracteres. 41
Figura 7.14 Fichero de mapeo. 42
Figura 7.15 El cliente abre el socket para la conexión. 44
Figura 7.16 Cliente hace solicitud de conexión. 45
Figura 7.17 El servidor acepta la conexión del cliente. 45
Figura 7.18 Conexión establecida. 46
Figura 7.19 Envío de datos del cliente al servidor 46
Figura 7.20 “FSM – Finite State Machine” que define el comportamiento de la IA 47

Figura 7.21 HUD. 49
Figura 7.22 Sombras de los objetos. 49
Figura 8.1 Splash Screen..... 50
Figura 8.2 Menú principal. 50
Figura 8.3 Partida un jugador. 51
Figura 8.4 Pausa partida un jugador. 51
Figura 8.5 Menú pantalla multijugador..... 52
Figura 8.6 Esperando conexión de un cliente. 52
Figura 8.7 Menú partida multijugador (cliente). 53
Figura 8.8 Multiplayer. 53
Figura 8.9 Perdida de conexión del contrincante. 54
Figura 8.10 Imagen menú ayuda. 54
Figura 8.11. Escenario de pruebas. 58
Figura 8.12 Resultado benchmark LIBGDX. 59
Figura 8.14 Resultado benchmark Unity. 59
Figura 8.13 Resultado benchmark AndEngine. 59
Figura 11.1 Página web LIBGDX. 65
Figura 11.2 Project Setup. 65
Figura 11.3 Proyecto generados. 66
Figura 11.4 Importando proyectos..... 67
Figura 11.5 Importar proyecto gradle. 68
Figura 11.6 Selección de proyecto a importar. 68
Figura 11.7 Construyendo el modelo para gradle. 69
Figura 11.8 Proyecto preparado para importar. 69
Figura 11.9 Proyecto importado a eclipse 69
Figura 11.10 Ejecutar aplicación 70

1. Introducción

En la última década los juegos han aumentado los ingresos de las empresas que se dedican a su desarrollo, incluso más que el cine y la música. Se estima que la industria de los videojuegos generó una recaudación mundial de 57.600 millones de euros durante 2009 y 91.000 millones de dólares en 2016. Si nos enfocamos en la entrada al mercado de los smartphones y tablets y el desarrollo de los juegos online, podemos ver que el mercado de consolas ha disminuido y el de los juegos online y móviles (smartphones) ha aumentado. Como conclusión, podemos ver los videojuegos en smartphones tienen un gran futuro [9].

Para tener una buena introducción al mundo de la programación de videojuegos es importante ayudarnos con un *framework*. Este dará herramientas al desarrollador para implementar funciones más complejas como pueden ser la física. Un *framework* es una librería que nos ayudará en nuestro trabajo con funcionalidades ya implementadas, estandarizadas y con su documentación pertinente. No tendremos una interfaz gráfica donde posicionar los objetos en pantalla, ni nada por el estilo, pero tendremos funciones que nos permitirán implementar físicas, luces, dibujado de gráficos, etc. Hay que tener en cuenta que, a la hora de desarrollar un videojuego, existen características, como el dibujado de imágenes o la implementación de la física que pueden llegar a ser demasiado complejas para el desarrollador. En este punto es donde entran los *frameworks* o librerías, ya que facilitan dichas faenas más complejas.

En este proyecto se va a mostrar el *framework* LIBGDX para el desarrollo de videojuegos y se comparará con otro de los *frameworks* más punteros actualmente, AndEngine. También se hará una comparación con el motor gráfico Unity 3D, para comparar el rendimiento entre un *framework* y un motor gráfico. Para mostrar la herramienta LIBGDX se desarrollará un pequeño videojuego (*AirHockey*).

Voy a centrar este proyecto en el sistema operativo Android (móvil) y en algunos puntos haré referencia a PC (en todos los casos por defecto LIBGDX crea el proyecto para este). Otros sistemas como iOS o MAC necesitan licencias de desarrollador entre otras. Concretamente me centraré en el desarrollo de juegos en perspectiva 2D.

LIBGDX 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 o videojuego. En LIBGDX se puede trabajar o modificar las diferentes capas de la programación, pudiendo acceder al sistema de ficheros, dispositivos de entrada de audio, e incluso a las interfaces de OpenGL 2.0 y 3.0.

Me gustaría remarcar que este proyecto va dirigido a estudiar un *framework* de desarrollo que ayuda a la programación de videojuegos, no un motor gráfico. Un motor gráfico contiene muchas funcionalidades ya implementadas que el desarrollador no llega a ver ni a entender, ya que estas se suelen utilizar desde una interfaz gráfica. A continuación, se muestran los objetivos generales y personales de este trabajo de final de curso.

2. Objetivos

El objetivo de este trabajo es mostrar el *framework* LIBGDX, compararlo con otros frameworks y desarrollar un videojuego para ver el resultado que ofrece LIBGDX. El videojuego tendrá un modo para jugar contra una Inteligencia Artificial (IA) en el modo "mono jugador" y otro modo Multijugador para poder jugar contra nuestros amigos, compañeros o quién queramos.

Los objetivos específicos de este proyecto son:

- Realizar un estudio del *framework* LIBGDX, donde se compara con otros *frameworks* y se realizará un *benchmark* para ver los resultados de rendimiento.
- Crear un videojuego, será una simulación del típico *Airhockey*, en el cual dos personas compiten sobre una mesa de hockey de aire, utilizando mazos para impulsar un disco, con la finalidad de anotar puntos en la portería contraria.
- Crear un modo de un solo jugador contra la IA, se implementará una IA básica para el funcionamiento del contrincante.
- Crear un modo multijugador para dos usuarios, en este caso será necesario dos móviles, y la conexión se hará mediante Bluetooth.
- Realizar un diseño visual atractivo que proporcione una buena experiencia de juego.
- Ajustar el diseño para que se adapte a todos los tipos de pantalla, ya sea pantalla FULL HD, HD etc.
- Documentar todos los pasos realizados en cada una de las etapas del proyecto.

Seguidamente se mostrarán los objetivos personales, antes de citarlos quiero remarcar que ya tenía conocimiento previo de desarrollo de videojuegos, tanto en *Unity* como en *AndEngine* y *LIBGDX*.

Objetivos personales:

- Utilización y aplicación de conocimientos adquiridos en la carrera.
- Aumentar el conocimiento sobre el desarrollo de videojuegos.
- Desarrollar un producto de calidad y que sea innovador (en el multijugador).
- Compartir mis conocimientos de LIBGDX con personas interesadas en la introducción al mundo de los videojuegos.

3. Antecedentes

A continuación, se van a mostrar las tecnologías utilizadas en el proyecto, tanto para desarrollar el videojuego *AirHockey*, como para realizar el estudio y la comparación entre *frameworks*, y algunos procesos relacionados.

LIBGDX es un *framework* de desarrollo de juegos, con lo cual no ofrece herramientas para hacer otros tipos de tareas, como puede ser el diseño de la interfaz. Para realizar estos otros procesos se han utilizado otras herramientas, Photoshop y Hiero han sido algunas de ellas.

3.1 Tecnologías utilizadas

Algunas de las herramientas utilizadas ya las conocía previamente, pero algunas de ellas las desconocía completamente y para su uso he tenido que estudiarlas durante este proyecto. Entre las que desconocía se encuentran Hiero, ObjectAidUML y Photoshop, solo lo había utilizado en contadas ocasiones para modificar alguna imagen.

LIBGDX

LIBGDX es un *framework* de código abierto para desarrollo de videojuegos, propuesto por Mario Zechner a mediados de 2009 [1]. En marzo de 2010, Zechner decidió abrir el código y actualmente tiene la licencia apache 2.0. La última versión disponible y estable de LIBGDX es la 1.9.6 (a día 31/05/2017) y ésta es la utilizada para la implementación del videojuego. La potencia, flexibilidad, constante evolución y soporte de la comunidad de este software, son unos de sus puntos fuertes. El hecho de que haya una comunidad tan fuerte detrás del *framework* ayuda mucho a la hora de encontrar soporte, puesto que otros *frameworks* no cuentan con este factor tan importante.

En este proyecto, LIBGDX se ha utilizado para desarrollar el videojuego *AirHockey*. También se ha utilizado para crear un escenario de pruebas y comparar su rendimiento con otros frameworks y motores gráficos.

AndEngine

AndEngine es un *framework* para desarrollo de videojuegos para Android, escrito en JAVA y desarrollado por Nicolas Gramlich [2]. El motor utiliza OpenGL para proporcionar una salida de gráficos acelerados.

Este *framework* al igual que *LIBGDX* tiene una gran cantidad de videojuegos desarrollados a sus espaldas. *AndEngine* se ha utilizado en este proyecto para crear un escenario de pruebas y comparar su rendimiento con *LIBGDX*.

Unity

Unity es un *motor gráfico*, el cual tiene todas las funciones desarrolladas por Unity Technologies [3]. Actualmente es uno de los *motores gráficos* más importantes y más utilizados para desarrollar videojuegos para dispositivos móviles. Unity en este proyecto, al igual que AndEngine, se ha utilizado para crear un escenario de pruebas y comprobar su rendimiento con LIBGDX.

Photoshop

Photoshop es un editor de gráficos desarrollado por Adobe Systems Incorporated. Usado principalmente para retocar imágenes y fotografías, es el líder mundial en el mercado de edición fotográfica. Este programa se lanzó el 10 de febrero de 1990 y actualmente su última versión estable es la 18.1.1, la versión utilizada para el proyecto es la 13.0.

La razón por la que se ha seleccionado este software en este proyecto, es por su simplicidad de uso y su liderazgo en el mercado, ya que es el que tiene más soporte por parte de la comunidad y el que mayor proyección y utilidad tiene de cara a un futuro. En este proyecto se ha utilizado para creación, modificación y recorte de imágenes.

Hiero

Hiero es una herramienta para crear mapa de bits para las fuentes de texto. Puede ser utilizado por *BitmapFont* en aplicaciones LIBGDX [6]. En este proyecto, este programa se ha utilizado para crear las fuentes para los textos del videojuego. Se puede descargar directamente desde Github como soporte directo para LIBGDX.

Modelio

Modelio es una herramienta de código abierto, para crear diagramas UML, desarrollado por Modeliosoft. Es compatible con UML2 [4]. El software fue liberado bajo la licencia GPLv 3 el 5 de octubre de 2011 [19].

En este proyecto, este software se ha utilizado para crear documentación, básicamente para crear todos los diagramas UML del documento. La razón por la que se seleccionó este software fue porque es libre, y ya tenía conocimientos previos, ya que se utilizó en la asignatura de diseño de software.

A diferencia de Modelio, para exportar automáticamente el proyecto de LIBGDX a diagramas de clases, se ha utilizado un plugin de Eclipse llamado *ObjectAidUML*

ObjectAidUML [5] a diferencia de Modelio, se ha utilizado exclusivamente para exportar automáticamente el proyecto de LIBGDX a diagramas de clases, todos los demás diagramas se han realizado a mano con Modelio.

Eclipse

Eclipse es un IDE compuesto por un conjunto de herramientas de programación de código abierto y multiplataforma[7]. Eclipse se desarrolló inicialmente por IBM, pero actualmente está siendo desarrollado por la Fundación Eclipse, esta fundación es independiente y sin ánimo de lucro. Eclipse actualmente tiene la Licencia Publica de Eclipse [18].

El motivo por el que se ha seleccionado este entorno de desarrollo fue porque es software libre, tiene una gran proyección en el mundo laboral y existe una versión para desarrolladores de Android, la cual viene ya preparada con todas las librerías integradas, esto nos facilitará mucho el desarrollo en LIBGDX.

Lenguajes de programación utilizados:

JAVA

El lenguaje de programación utilizado para el desarrollo del videojuego *AirHockey* ha sido JAVA, ya que LIBGDX está basado en este lenguaje. Java es un lenguaje de programación orientado a objetos. La intención con este lenguaje es permitir a los desarrolladores que escriban el código una vez y lo puedan ejecutar en cualquier dispositivo. Java actualmente es uno de los lenguajes de programación más populares [15]. JAVA fue desarrollado originalmente por James Gosling de Sun Microsystems. Oracle adquirió Sun Microsystems, por tanto, JAVA actualmente es propiedad de Oracle. [30]

C#

Para realizar los *benchmarks* en Unity se ha desarrollado en C#, aunque Unity también permite desarrollar en Javascript. La elección de C# ha sido porque es un lenguaje muy parecido a java y esto ayuda a su desarrollo.

Android

Cuando se desarrolla en LIBGDX no es necesario tener conocimientos de Android, pero en este desarrollo se ha tenido que trabajar con la API del bluetooth de Android para poder crear el modo multijugador. El proyecto se ha desarrollado en eclipse, concretamente en la versión para desarrolladores de Android. En la sección [7] se explican detalles concretos sobre la implementación en Android.

En el proyecto de Android, para la configuración del proyecto nativo se utiliza XML. XML es un meta-lenguaje que permite definir lenguajes de marcas. Desarrollado por World Wide Web Consortium (W3C), utilizado para almacenar datos en forma legible.

3.2 Análisis de la competencia: Frameworks y Engines

En el siguiente apartado se muestran varios *frameworks* y *motores gráficos* que hoy en día son los más utilizados y valorados por la comunidad de desarrollo de videojuegos.

LIBGDX:

LIBGDX es un *framework* de código abierto, creado por BadLogicGames y apoyado por una gran comunidad. Uno de los objetivos principales del *framework* es mantener la simplicidad, sin renunciar al amplio abanico de plataformas finales. Está basado en Java, esto ayuda a los desarrolladores que tienen conocimiento en este lenguaje a iniciarse en la creación de juegos.

LIBGDX es una herramienta multiplataforma con la que podemos crear juegos para Windows, Linux, OS X, HTML, Android y iOS (Para esto necesitaremos el soporte de RoboVM, con el cual podremos ejecutar código Java en iOS). Hay que tener en cuenta las licencias necesarias para el desarrollo en MAC etc. Nos encontramos también ante una herramienta donde sólo tendremos que escribir el código una sola vez y exportarlo a la tecnología que nos interese sin modificar nada. Exceptuando cuando sea necesario el trabajo con las APIs de los dispositivos, en este caso será necesaria la implementación del código para el acceso y gestión de las APIs. Por ejemplo, en los casos de acceso a cámara, bluetooth etc.

En LIBGDX nos encontramos con una total libertad ya que nos permite bajar el nivel de abstracción tanto como se quiera, dando acceso directo al sistema de archivos, dispositivos de entrada y audio e incluso a las interfaces de Open GL. Sobre estos niveles de abstracción se encuentra un potente conjunto de APIs que permiten mostrar imágenes y texto, construir interfaces de usuario, reproducir sonidos, música o realizar cálculos matemáticos entre otras, de forma mucho más sencilla.

Otro punto fuerte que encontramos en LIBGDX es que podemos combinar el código de la librería con código nativo Android o de otro dispositivo soportado, con esto conseguiríamos trabajar con la API del dispositivo fácilmente. Concretamente en este proyecto he trabajado con la API de Android para acceder y utilizar el bluetooth.

AndEngine:

AndEngine igual que LIBGDX es compatible con librerías externas como puede ser Box2D para implementar físicas (gravedad, rozamiento…). Hay que tener en cuenta que este *framework* es solo para desarrollar en Android. Para desarrollar con este *framework* podemos utilizar Eclipse o Android Studio.

Podemos ver que para desarrollar un videojuego es más interesante utilizar LIBGDX, ya que nos permite exportar a varias plataformas y de esta manera no tendremos que programar más de una vez el código, y el coste sería menor. LIBGDX es compatible con la última tecnología de gráficos OpenGL y AndEngine no se ha actualizado desde hace bastante tiempo. Además, LIBGDX es más fácil de aprender, ya que está mejor documentado que AndEngine y tiene una comunidad que lo rodea, así como una gran wiki que enseña los fundamentos de manera muy fácil de entender.

Torque2D:

Toque2D es un motor gráfico de código abierto muy potente, flexible y rápido dedicado al desarrollo de videojuegos 2D. Torque2D al igual que LIBGDX utiliza la librería Box2D como su sistema de física.

Para desarrollar en Torque2D han creado el lenguaje *TorqueScript*, es similar a C++ y a C#, por esta razón vamos a descartarlo de este estudio, ya que el objetivo es comparar frameworks basados en Java.

Unity 3D / 2D:

A diferencia de LIBGDX, Unity es un proyecto de código cerrado, con lo cual tiene actualizaciones constantemente. Esto implica que para un estudio pequeño o independiente puede suponer un costo demasiado grande, aunque Unity tiene una versión gratuita donde entran varias funcionalidades.

La primera diferencia que se notará cuando inicie Unity es que su principal característica es un editor gráfico, en cambio en LIBGDX el desarrollo es principalmente basado en código. Una gran cantidad del desarrollo de juegos con Unity pasa por arrastrar cosas del editor y colocarlas en el mapeado del juego, pero para darle funcionalidad se debe programar en C# o en Javascript.

En este proyecto, se ha utilizado Unity para realizar un *benchmark* y ver rendimientos comparados con los de LIBGDX. Es interesante hacer una comparación ya que posiblemente es uno de los *motores gráficos* más conocidos por los desarrolladores de videojuegos.

JMonkeyEngine

Es un motor gráfico de código abierto, hecho especialmente para los desarrolladores en JAVA, creado para desarrollar juegos en 3D. El software está programado en JAVA. En JMonkeyEngine podemos exportar nuestros proyectos a Windows, Mac y Linux, así como Android y iOS. Además, se apoya las nuevas tecnologías como puede ser VR (Realidad virtual).

A diferencia de Unity no voy a estudiarlo ni a entrar en comparaciones, ya que no es un framework, sino un *motor gráfico*, pero en este caso no es tan conocido como Unity y he decidido solo introducir Unity en las comparaciones. Este motor gráfico es para desarrollar en 3D y en este proyecto me he focalizado en 2D.

Conclusión

Para desarrollar videojuegos 2D es una buena opción utilizar LIBGDX, ya que no tendremos problemas de licencias, tenemos una buena gestión de recursos implementada, y un buen soporte de la comunidad. También podemos acceder a todo el código y modificarlo según nuestra necesidad incluso llegando a bajar a bajo nivel (Open GL). En cambio, para desarrollar videojuegos 3D sería mejor utilizar un motor gráfico tipo Unity 3d ya que su editor gráfico puede ayudar mucho a la hora de trabajar en 3 dimensiones.

Utilizar LIBGDX es más eficiente que utilizar AndEngine, ya que con LIBGDX se programará el código una sola vez (exceptuando el código de acceso a las APIs de las diferentes plataformas) y se podrá exportar a varias plataformas. Si se desarrolla con AndEngine y se quiere importar a otra plataforma se deberá buscar otro *framework* compatible con la nueva plataforma y desarrollar todo el proyecto desde cero.

Ya que éste proyecto va enfocado a mostrar las capacidades de LIBGDX y se dirige a los que se inician en la programación de videojuegos, el *framework* mejor posicionado es LIBGDX, porque el desarrollo es en Java y esto facilitará mucho el trabajo ya que es uno de los lenguajes más utilizados.

3.3 Trabajos relacionados: Juegos LIBGDX

Para tener una idea más sólida de lo que se puede llegar a conseguir con este *framework* se ha buscado información sobre videojuegos desarrollados con él. A continuación, en este apartado se muestran varios videojuegos que actualmente están en producción, y han sido desarrollados con LIBGDX, podemos encontrarlos en Google Play.

Figura 3.1 The Spartan War.

Spartania: TheSpartanWar

Videojuego de estrategia, actualmente tiene entre 1.000.000 y 5.000.000 de descargas.

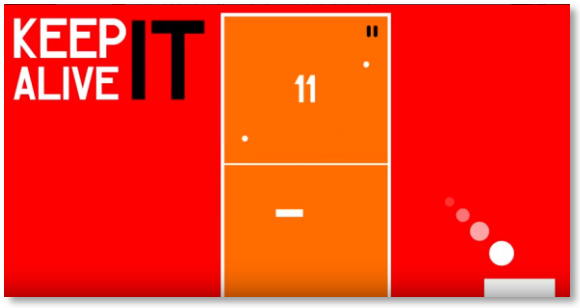
Perspectiva: 2.5 D

Figura 3.2 Village and Farm.

Village and Farm

Videojuego Arcade, actualmente tiene entre 5.000.000 y 10.000.000 de descargas.

Perspectiva: 2.5 D



Keep IT Alive

Videojuego Arcade, actualmente tiene entre 50 descargas.

Perspectiva: 2 D

Figura 3.3 Keep IT Alive.

Como se puede comprobar en las imágenes anteriores los resultados pueden ser excelentes, podemos obtener prácticamente el mismo resultado que con un motor gráfico.

Actualmente en el mercado existen varios videojuegos de simulación de *AirHockey*. A continuación, se muestra una imagen de uno de ellos escogido al azar, similar al que se ha desarrollado en este proyecto.

Figura 3.4 Glow Hockey.

En el videojuego *AirHockey* desarrollado en este proyecto, para innovar, se ha hecho que, en el modo multijugador, cada usuario tenga en pantalla solo la parte de su zona del campo y que el disco vaya pasando de un móvil a otro como se puede ver en la siguiente imagen.

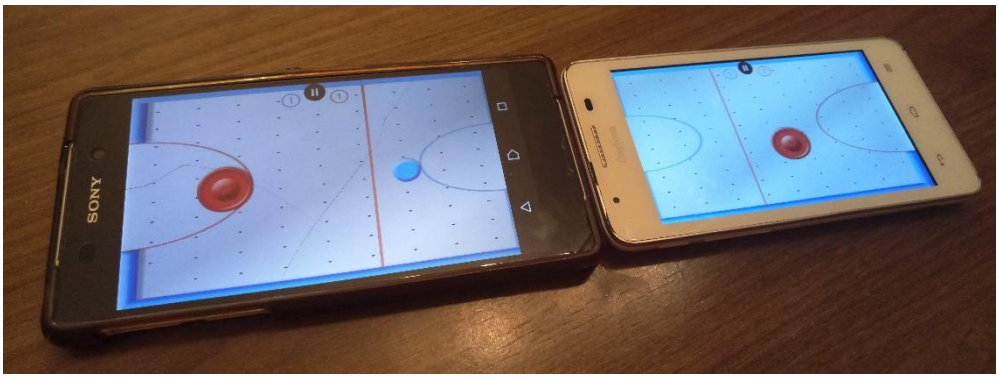


Figura 3.5 Partida multijugador.

4. Conceptos básicos de LIBGDX

A continuación, se muestran unos conceptos básicos de LIBGDX para facilitar la lectura y entendimiento de los apartados posteriores de este documento.

- Actor: Es un objeto del escenario. Un actor tiene una posición, un tamaño, rotación y color. En el juego *AirHockey* serán actores el mazo y el disco.
- Viewport: Este no es un concepto de LIBGDX, pero es un concepto general necesario entender para poder utilizar la cámara correctamente. El viewport define la cantidad que desea ver del mundo del juego. Es como una ventana por la que se mira al mundo del juego. Puede ser que el escenario ocupe toda la pantalla del dispositivo, pero lo que el usuario visualizará, es lo que se vea a través del viewport, es decir, si el escenario tiene un tamaño de 800x800 y el viewport un tamaño de 200x200, el usuario solo verá este trozo del escenario [13].
- Stage: Maneja el viewport y distribuye los eventos de entrada a los actores, como pueden ser los "clics" en pantalla. Para detectar los eventos de entrada existe:
    - *InputProcessor:* Sirve para detectar los eventos de entrada del teclado, pantalla táctil, acelerómetro etc.
- SpriteBatch: Para dibujar texturas se utiliza el objeto *SpriteBatch*. Enviar texturas a la tarjeta gráfica es un proceso muy lento, aquí es donde entra en juego el *SpriteBatch*. Este se encarga de enviar todas las imágenes juntas a la tarjeta gráfica de una vez, y con esto se consigue optimizar el proceso.
- Scene2D: Sirve para crear un escenario gráfico 2D con una jerarquía de actores, es importante remarcar que a diferencia del objeto Stage, este es para la interfaz de usuario. Proporciona las siguientes características:
    - Rotación y escala.
    - Simplificación del dibujado 2D a través del *SpriteBatch*.
- Body: Para crear cuerpos rígidos, a los cuales se les asignará una física y una forma de cuerpo.
    - FixtureDef: Se utiliza para definir cómo será un cuerpo. Contiene la información de cómo puede ser el tipo de cuerpo que va a formar el objeto como puede ser la densidad o la fricción entre otros. También contiene la forma que tendrá el objeto, LIBGDX ofrece varias formas de cuerpos predefinidos como son circulo, triangulo y rectángulo.
- Shape: Define una forma, se usa para dar una forma a un cuerpo. Con un shape se pueden detectar colisiones.
    - CircleShape: Para crear una forma de circulo.
- TextureRegion: Define un área rectangular de una textura.
- Skin: Guarda los recursos de texturas para los widgets de la interfaz del usuario. Se accederá a cada recurso mediante un JSON que mapea regiones de la textura.
- World: Esta clase gestiona todas las entidades de la física y simulación de la dinámica.

- Vector2: Es un vector de dos dimensiones, este se utiliza para trabajar con las velocidades del mazo y del disco, ya que sus velocidades son en dos dimensiones, se desplazan solo en los ejes de X y Y cuando se trabaja en dos dimensiones.
- BitmapFont: Representa las fuentes de mapa de bits. La fuente consta de 2 archivos, un archivo de imagen, que contiene todos los símbolos del abecedario y un archivo en el formato de texto donde se indica la posición de cada símbolo en la imagen.
- Box2D: Es una librería de físicas 2D desarrollada en C++. Es una de las librerías más populares de física para juegos 2D y ha sido portado a muchos idiomas y motores diferentes, incluyendo LIBGDX [14].
  La librería Box2D en LIBGDX es una capa Java alrededor del motor C++.

A la hora de desarrollar es importante tener claros los siguientes métodos que ofrece LIBGDX:

Método *create*: Es un método que se ejecuta cuando se abre la ventana de juego por primera vez o la escena de nuestro proyecto. Este método debe contener el código que creará la escena.

Método *render*: Es un método que se ejecuta para renderizar la escena. Se llama dentro del ciclo de vida del videojuego, por tanto, se llamará aproximadamente 60 veces por segundo, depende del dispositivo y del estado de la escena. Es importante no realizar grandes cálculos ni procesos complejos en esta función, ya que esta es la que se encarga del refresco de la escena. Si se crean procesos que sean muy lentos puede ralentizar la escena. En este caso, sería necesario crear un nuevo thread para lanzar este proceso.

Método *dispose*: Es el método que se llamará al cerrar la escena, aquí debemos eliminar todos los objetos creados en la escena para liberar toda la memoria, utilizando *dispose()*. En el caso de un objeto *Textura*, si no se hiciera el *dispose()*, el objeto textura desaparecerá porque lo eliminará el recolector de basura de java, pero la textura que está cargada en GPU no la eliminará. Por tanto, hay que eliminar todos los objetos creados en la escena dentro de este método, si no se acumularán texturas en la GPU que no se podrán usar y estarán ocupando espacio.

5. Análisis

En este apartado se va a mostrar un análisis sobre el videojuego desarrollado.

5.1 Mecánica del juego *AirHockey*

El videojuego desarrollado está basado en el género Arcade, enfocado a deportes. Arcade se utiliza para referirse a máquinas recreativas de videojuegos situadas en lugares públicos de ocio, centros comerciales, restaurantes, bares o salones recreativos especializados. Estos videojuegos son similares a pinball, air hockey etc. Los juegos de género Arcade se suelen basar en la destreza del jugador.

El videojuego creado es un simulador del juego air hockey, de los salones recreativos. Se puede jugar como un solo jugador o con modo multijugador, con un máximo de dos personas. En este caso será necesaria la utilización de dos teléfonos móviles. En PC no se podrá utilizar el modo multijugador, ya que la comunicación se hace vía bluetooth.

En el modo de un solo jugador, al iniciar la partida encontraremos el tablero de juego con nuestro mazo y el de la IA. En este modo, el juego consistirá en intentar marcar nueve goles en la portería contraria, antes de que la IA nos marque los nueve goles a nosotros.

En el modo multijugador, al iniciar la partida nos encontraremos con que el tablero de juego se divide en dos mitades, cada mitad en un móvil distinto, es decir, en un móvil tendremos la mitad del tablero correspondiente a nuestra zona de juego y en el otro dispositivo móvil el contrincante, verá su otra mitad del campo. Seguidamente veremos que cuando el disco se aproxima al borde de la pantalla del dispositivo móvil del jugador 1, el disco pasa automáticamente a la pantalla del dispositivo móvil del jugador 2 y viceversa. También cabe decir, que, en cada pantalla de móvil, el jugador sólo verá su mazo y en ningún caso el del contrincante.

5.2 Casos de uso

A continuación, se muestra una descripción de los pasos o las actividades que he realizado para llevar a cabo el desarrollo de software del proyecto.

5.2.1 Diagrama de casos de uso

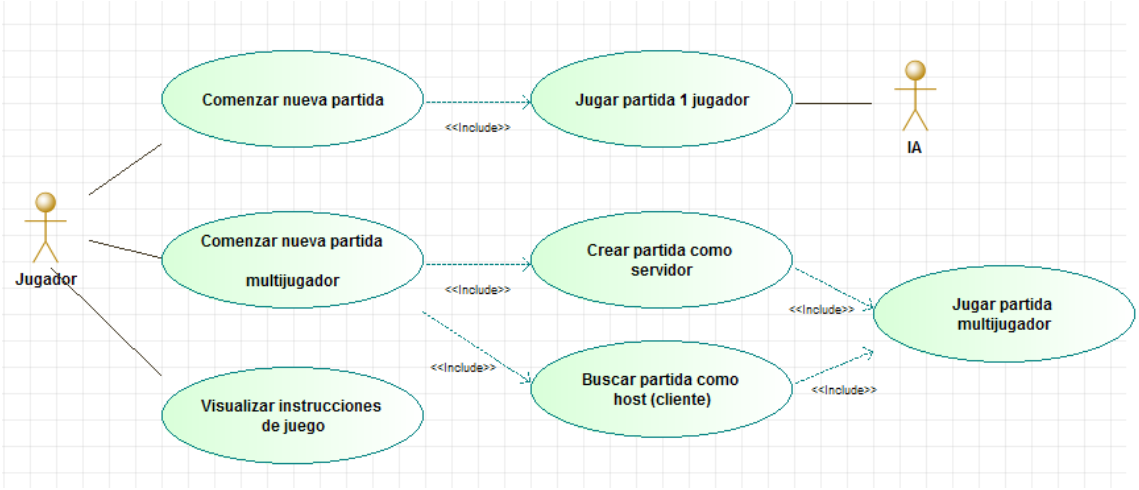


Figura 5.1 Diagrama de casos de uso.

Descripción de casos de uso

En este apartado se describen los casos del diagrama de casos de uso [33]. Figura 5.1.

Nombre:	**UC1 – Comenzar nueva partida**
Actor:	Usuario
Descripción:	Proceso de inicio de la partida para un jugador.
Precondiciones:	El usuario se encuentra en el menú principal.
Flujo Básico: 1. El usuario selecciona en el botón de *Un Jugador* 2. El sistema inicia la partida para un jugador 3. El sistema inicia el primer nivel de juego, carga las texturas del fondo, la textura del mazo, del disco, el escenario y los marcadores.	
Flujo Alternativo: *No tiene flujo alternativo*	
Postcondiciones:	El usuario se encuentra en la nueva partida.

Nombre:	**UC2 – Jugar partida de 1 jugador**
Actor:	Usuario y IA
Descripción:	Ejecución de la partida de 1 jugador

Precondiciones:	El usuario se encuentra al inicio de la partida

Flujo Básico:

1. El sistema muestra un mensaje para iniciar la partida.
2. El usuario hace "clic" en la pantalla para comenzar la partida.
3. El usuario mueve el mazo y golpea el disco.
4. El sistema evalúa la posición del disco y lo golpea para retornarlo al campo contrario
5. El disco entra en la portería del usuario y el sistema detecta gol, y lo muestra en el marcador
6. El sistema muestra un mensaje avisando al usuario para que posicione la bola para continuar la partida y se retorna al paso 2.

Flujo Alternativo:

3a. El disco entra en la portería de la IA y el sistema detecta gol, lo muestra en el marcador, si el gol es el número 9 gana la partida el usuario y el sistema muestra un mensaje indicando *"WIN!"*

3b. El sistema posiciona el disco y además lo golpea para continuar la partida y vuelve al punto 3

5a. Si el gol es el número 9 gana la partida el sistema y muestra un mensaje indicando *"WIN!"*

Postcondiciones:	El usuario ha finalizado la partida.

Nombre:	**UC3 – Comenzar nueva partida multijugador**
Actor:	Usuario1, Usuario2
Descripción:	Proceso de inicio de la partida para el multijugador.
Precondiciones:	El usuario se encuentra en el menú principal.

Flujo Básico:

1. El usuario selecciona en el botón de *Multijugador*
2. El sistema carga el menú de configuración de la partida
3. El usuario hace "clic" en el botón *Servidor*, va al caso UC4.

Flujo Alternativo:

3. El usuario hace "clic" en el botón *Cliente* va al caso UC5.

Postcondiciones:	El usuario se encuentra iniciando una partida

	multijugador

Nombre:	**UC4 – Crear partida como servidor**
Actor:	Usuario
Descripción:	Proceso de inicio de la partida para el multijugador como servidor.
Precondiciones:	El usuario se encuentra creando la partida multijugador como servidor.
Flujo Básico: 1. El sistema pide al usuario activar y poner visible el bluetooth. 2. El usuario acepta la condición. 3. El sistema pone visible el bluetooth y espera a recibir una conexión de un cliente.	
Flujo Alternativo: 2a. El usuario deniega la condición de poner visible el bluetooth. (Vuelve a UC3). 3a. Si hay algún fallo en la conexión se vuelve a UC3.	
Postcondiciones:	El usuario inicia la partida multijugador como servidor.

Nombre:	**UC5 – Buscar partida como cliente**
Actor:	Usuario
Descripción:	Proceso de inicio de la partida para el multijugador como cliente.
Precondiciones:	El usuario se encuentra creando la partida multijugador como cliente.
Flujo Básico: 1. El sistema pide al usuario activar y poner visible el bluetooth. 2. El usuario acepta la condición. 3. El sistema pone visible el bluetooth y muestra el menú de búsqueda y selección de dispositivos para conectarse. 4. El usuario selecciona buscar contrincante. 5. El sistema realiza una búsqueda de dispositivos que tengan el bluetooth activo y los muestra en una lista.	

6. El usuario selecciona el dispositivo al que quiere conectarse para iniciar la partida y hace "clic" en el botón *Conectar.*
7. El sistema abre la conexión contra el dispositivo seleccionado y se incorpora a la partida creada por el servidor.

Flujo Alternativo:

2a. El usuario deniega la condición de poner visible el bluetooth. (Vuelve a UC3).

5a. Si hay un fallo al realizar la búsqueda o el sistema no encuentra ningún dispositivo, no se cargará nada en la lista, el usuario debería volver a realizar la búsqueda. (Desde el punto 4).

7a. Si hay algún fallo en la conexión se vuelve a UC3.

Postcondiciones:	El usuario inicia la partida multijugador como host (cliente).

Nombre:	**UC6 – Jugar partida de multijugador**
Actor:	Usuario
Descripción:	Ejecución de la partida multijugador
Precondiciones:	El usuario se encuentra al inicio de la partida

Flujo Básico:

1. El sistema inicia el primer nivel de juego, carga las texturas, del fondo, la textura del mazo, del disco, el escenario y los marcadores, seguidamente muestra un mensaje para iniciar la partida.
2. El usuario hace "clic" en la pantalla para comenzar la partida.
3. El usuario mueve el mazo y golpea el disco.
4. El enemigo (usuario contrario) evalúa la posición del disco y lo golpea para retornarlo al campo contrario
5. El disco entra en la portería del usuario, el sistema detecta gol, y lo muestra en el marcador
6. El sistema muestra un mensaje avisando al usuario para que posicione la bola para continuar la partida y se retorna al paso 2.

Flujo Alternativo:

3a. El disco entra en la portería del usuario (enemigo) y el sistema detecta gol, lo muestra en el marcador, si el gol es el número 10 gana la partida el usuario y el sistema muestra un mensaje indicando *"WIN!"* para el usuario y *"LOSE"* para el usuario (enemigo).

3b. El usuario (enemigo) posiciona el disco y lo golpea con el mazo para continuar la

partida y vuelve al punto 3 5a. Si el gol es el número 10 gana la partida el usuario (enemigo) y muestra un mensaje indicando *"WIN!"* al usuario enemigo y *"LOSE"* al usuario.	
Postcondiciones:	El usuario ha finalizado la partida.

Nombre:	**UC7 – Visualizar instrucciones de juego**
Actor:	Usuario
Descripción:	Visualización de las instrucciones de juego
Precondiciones:	El usuario se encuentra en el menú principal
Flujo Básico: 1. El usuario hace "clic" en el botón de ayuda. 2. El sistema muestra el menú de ayuda.	
Flujo Alternativo: 2a. El usuario hace "clic" en el botón volver 2b. El sistema carga el menú principal.	
Postcondiciones:	El usuario se encuentra en el menú de ayuda

5.3 Requisitos

Los requisitos de hardware mínimos para poder utilizar el videojuego en dispositivos móviles son los siguientes:

Procesador	RAM	Tarjeta Gráfica	Bluetooth	Espacio en memoria
800 MHz	512 MB	128 MHz	SI (Para modo multijugador).	21 MB

Como se puede ver en la tabla anterior el juego podrá jugarse en prácticamente todos los Android disponibles en el mercado.

En el apartado de PC no se podrá utilizar el modo multijugador, ya que no se ha desarrollado la funcionalidad de conexión por bluetooth para este, pero se podrá jugar al modo de un jugador, para ello los requisitos serán los siguientes.

Procesador	RAM	Tarjeta Gráfica	Disco duro
1 GHz	512 MB	Integrada / Dedicada	21 MB

El videojuego en PC tiene un uso de memoria RAM aproximado de 134.000 KB, funciona con tarjetas de video integradas, para comprobar esto se ha mirado desde el panel de administración de Windows, y se ha comprobado el uso de memoria del proceso del videojuego *AirHockey*. En disco duro necesita tener como mínimo 21 MB espacio libre.

Podemos ver que dados los requisitos anteriores prácticamente va a funcionar en cualquier dispositivo móvil u ordenador, siempre que tengamos en cuenta que en PC no funcionará el multijugador y en móviles necesitamos tener bluetooth para poder jugarlo.

6. Diseño

A continuación, se muestran los diagramas de secuencia de algunos casos de uso. Cabe decir que todos ellos se han intentado simplificar, ya que eran demasiado grandes para representarlos. Por ende, las clases de las que extienden, las que se han representado en el diagrama, se han omitido.

6.1 Diagramas de secuencia

UC1 - Comenzar nueva partida

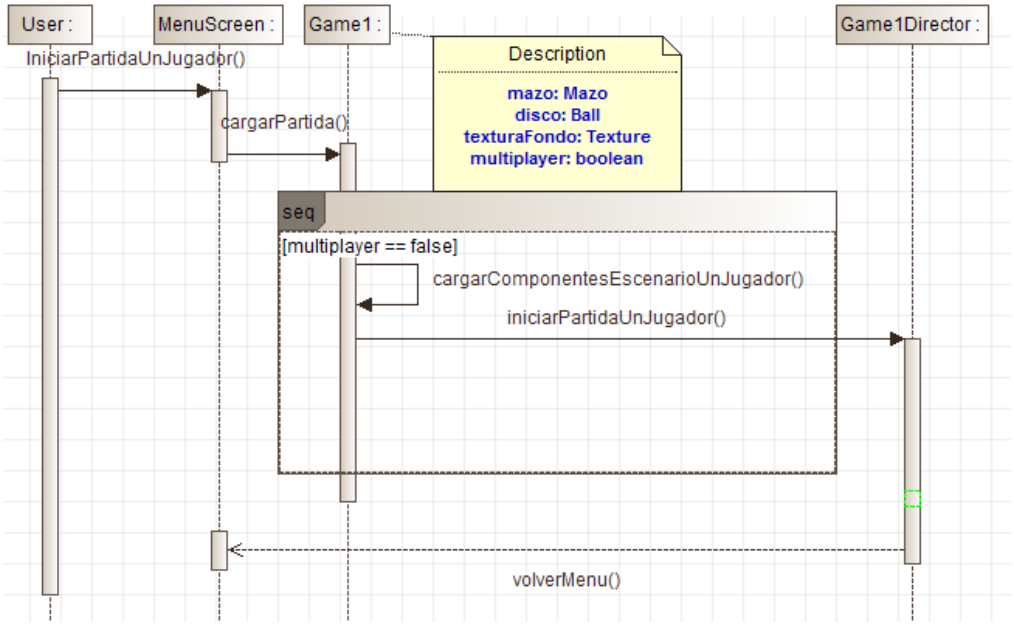


Figura 6.1 Secuencia comenzar nueva partida.

El usuario selecciona la opción de partida para un jugador desde el menú principal. Se llama a la función *cargarPartida* de Game1, esta clase es la encargada de crear la partida. Esta función comprueba si está configurado como multijugador, en caso de que no lo esté, carga los componentes para la escena de un jugador y a continuación llama a la clase *Game1Director* encargada de gestionar la partida de 1 jugador.

UC3 - Comenzar nueva partida multijugador y **UC4 -** Crear partida como servidor

Los casos UC3 y UC4 he decidido integrarlos en un solo diagrama, ya que el caso UC3 era demasiado corto para representarlo en un solo diagrama.

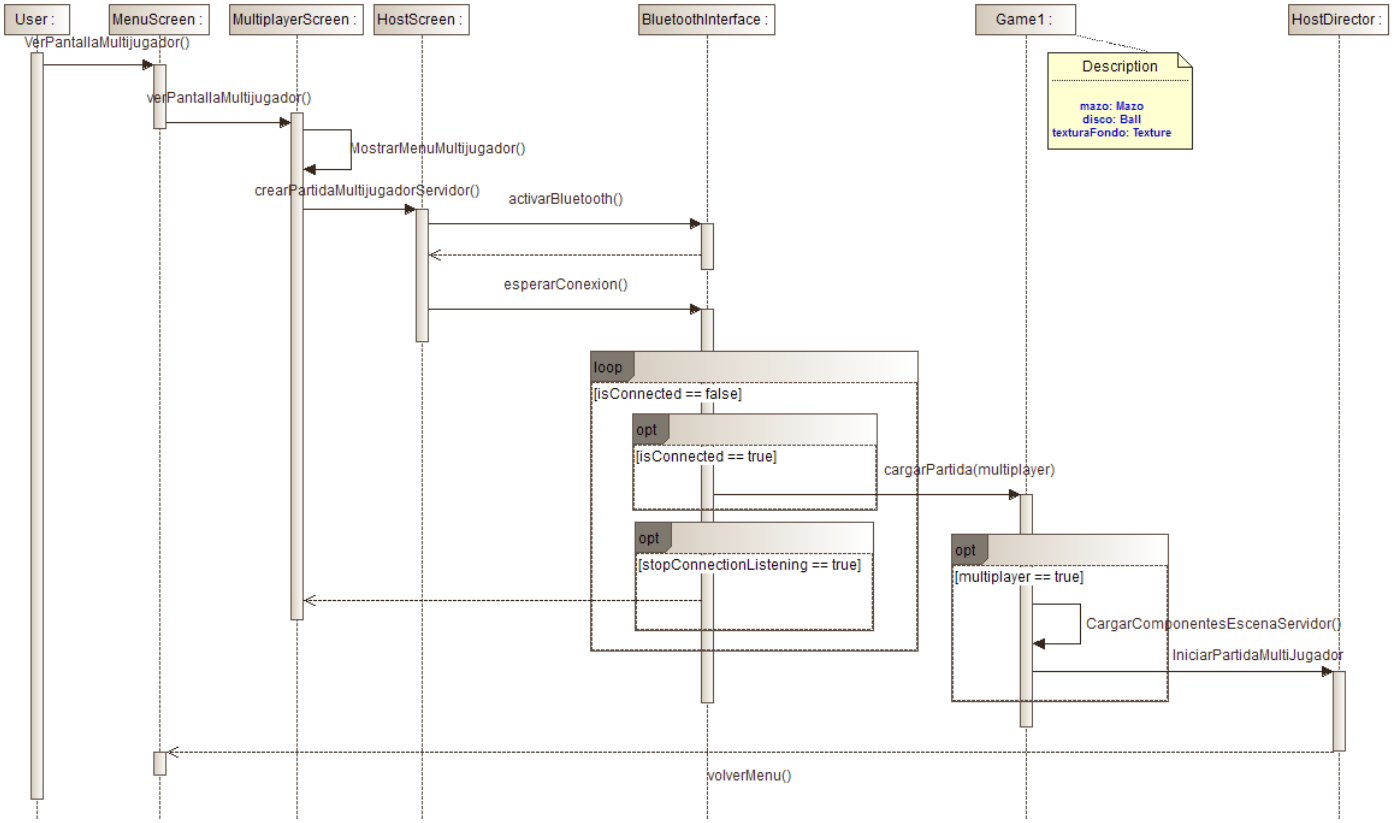


Figura 6.2 Secuencia crear partida como servidor.

Cuando el usuario selecciona la opción multijugador desde el menú principal se llamará a la clase MutiplayerScreen, la cual mostrará dicho menú multijugador con tres botones "Servidor", "Cliente" y "Atrás".

El usuario selecciona la opción de "Servidor", se llamará a la clase *HostScreen*, esta automáticamente llamará a bluetoothInterface. La clase bluetoothInterface es la encargada de conectar con la parte de Android y acceder a la API del bluetooth. Se activa el bluetooth a través de la interface y seguidamente se espera una conexión de un cliente.

Mientras no haya conexión el dispositivo móvil queda en espera. Si hay un fallo se vuelve a la clase *MultiplayerScreen*. Si no hay fallo y se conecta un cliente se llama a la función *cargarPartida* de *Game1*. Si está configurado como multiplayer se cargan los componentes de la escena para el servidor. A continuación, se inicia la partida multijugador llamando a la clase *HostDirector*, esta clase es la encargada de gestionar la partida para el servidor.

UC5 - Buscar partida como cliente

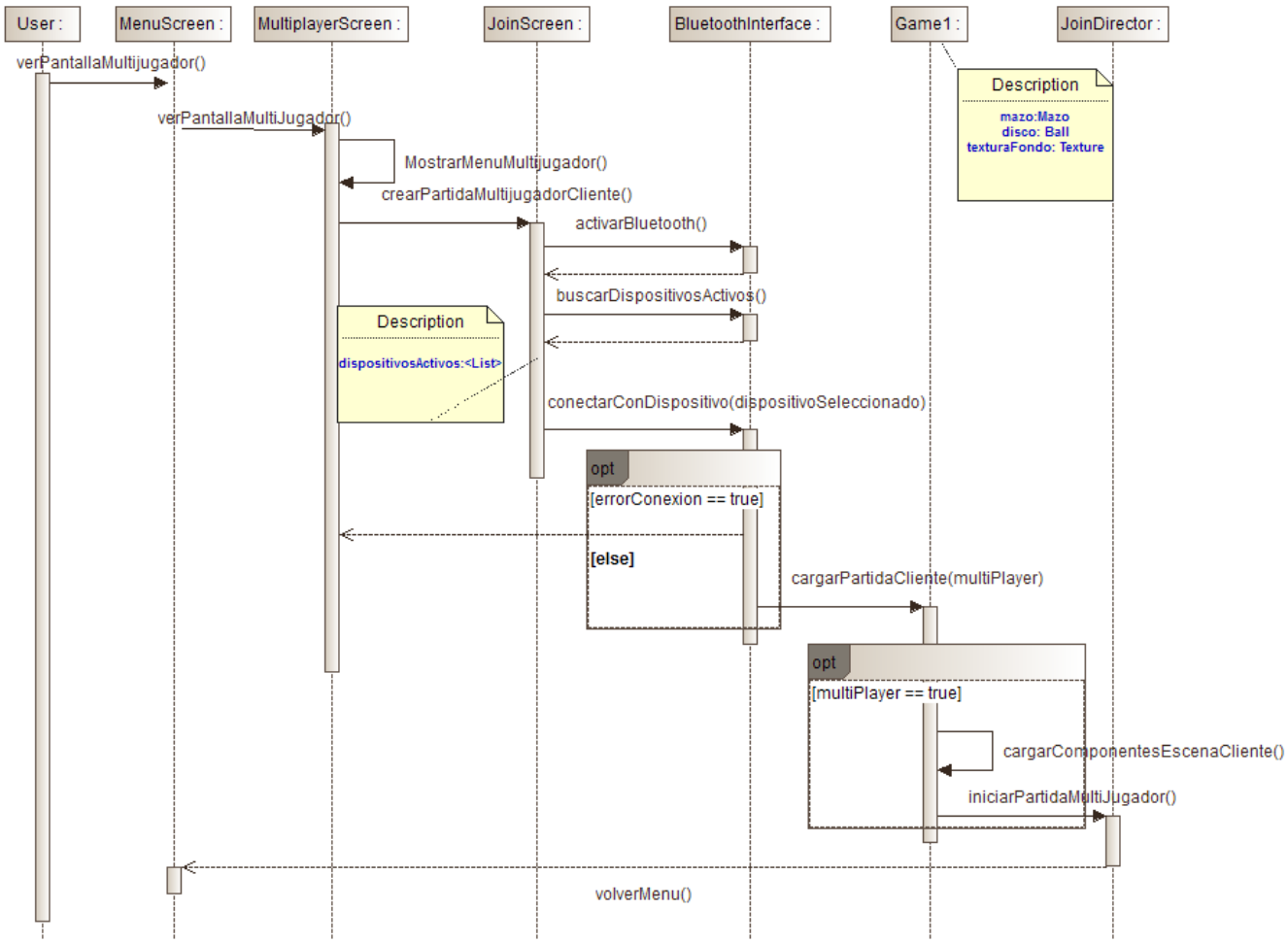


Figura 6.3 Secuencia buscar partida como cliente.

El usuario selecciona la opción multijugador desde el menú principal, se llama a la clase *Multiplayerscreen*, la cual mostrará el menú multijugador. El usuario selecciona la opción cliente y se llamará a la clase *JoinScreen*, esta clase es la encargada de mostrar los dispositivos disponibles para conectarse o buscar nuevos dispositivos. En esta clase se activa el bluetooth y se buscan nuevos dispositivos para conectar.

El usuario selecciona un dispositivo para conectar, si hay fallo en la conexión se vuelve a la clase *Multiplayerscreen*, si todo va bien se conecta con el dispositivo seleccionado mediante la clase *BluetoothInterface*.

Seguidamente se carga la partida para el cliente llamando a la clase *Game1*, si está configurado como modo multijugador se cargan los componentes de la escena para el cliente. Por último, se inicia la partida multijugador llamando a la clase *JoinScreen*, esta clase es la encargada de gestionar la partida para el cliente.

UC7 – Visualizar instrucciones de juego

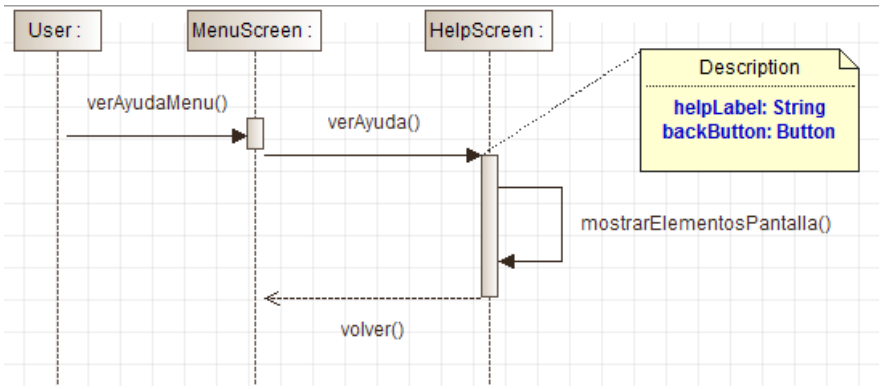


Figura 6.4 Secuencia visualizar instrucciones de juego.

El usuario selecciona la opción Ayuda del menú principal. Al seleccionar esta opción se llama a la función *verAyuda* de la clase *HelpScreen*. Esta mostrará los elementos del menú de ayuda.

6.2 Diagramas de clase

En este apartado se muestra la representación de clases. No se muestran sus variables y métodos con el objetivo de hacerlo más leíble y más compacto.

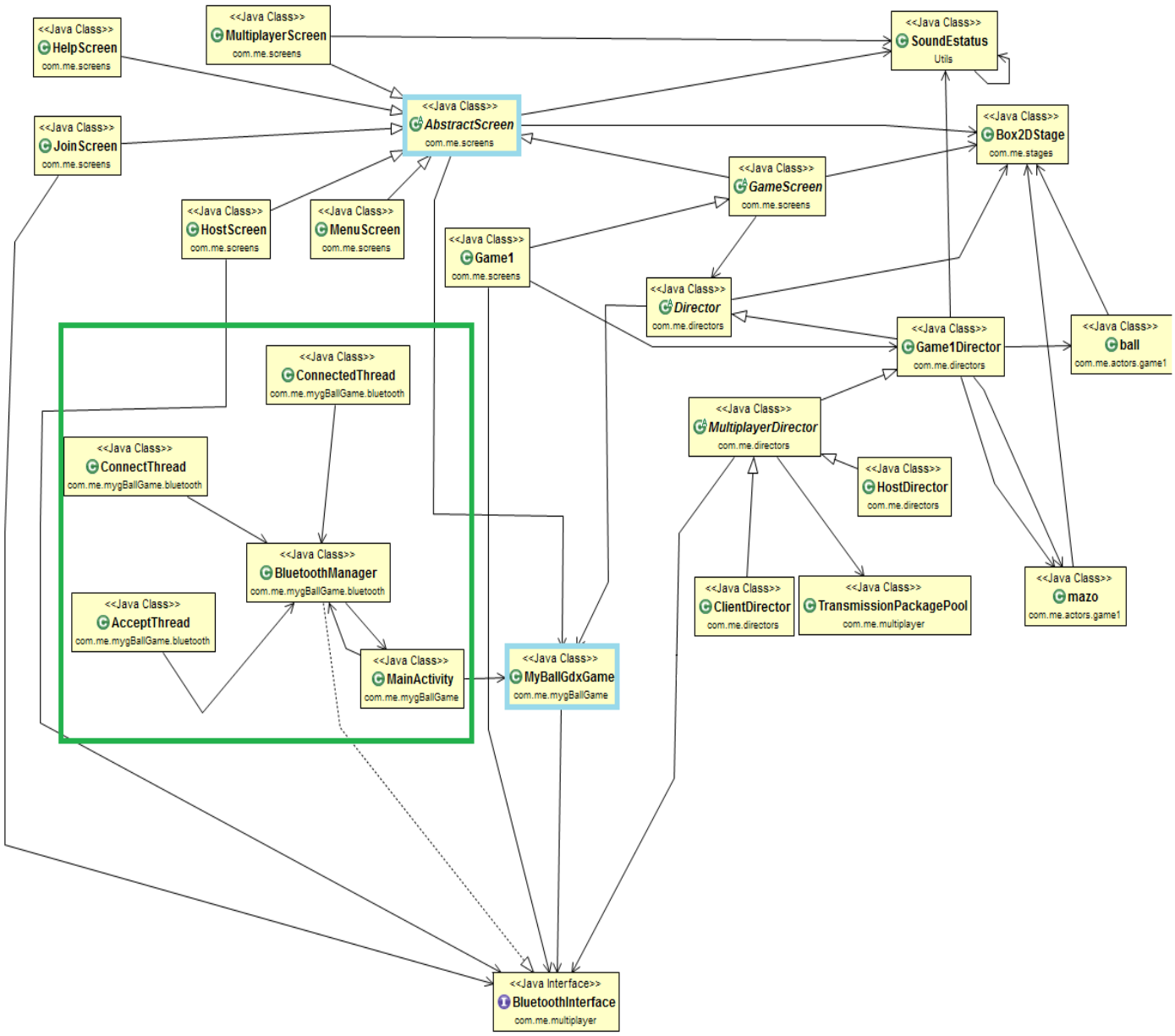


Figura 6.5 Diagrama de clases.

A continuación, se hace una breve explicación de las clases más importantes.

Las clases principales se muestran marcadas en azul (gris claro en blanco y negro).

- **AbstractScreen**: Esta clase contiene la estructura de todos los menús, las demás pantallas extienden de esta para poder mostrar su contenido.
- **MyBallGdxGame**: Esta es la clase principal, esta se encarga de cargar las texturas e iniciar la pantalla del menú principal (*MenuScreen*).

El apartado marcado en verde (en blanco y negro se verá gris oscuro), indica que se trata de las clases del proyecto de Android. Aquí se puede ver la clase ***BluetoothManager***. Esta clase es la que se encarga de acceder a la API del bluetooth de Android. Las clases ***AcceptThread, ConnectThread, ConnectedThread*** son las que se encargaran de abrir un nuevo thread y gestión la comunicación entre los dispositivos, apoyándose en la clase *BluetoothManager* para acceder a la API.

Game1: Encargada de crear el escenario y la interfaz de juego de la partida, tanto para un jugador como para multijugador, y renderizar la escena.

GameScreen: Encargada de activar el ContactListener (listener que detecta colisiones en el mundo del juego), ya que en LIBGDX para que se detecten colisiones en el mundo se necesita activar este listener. Esta clase también es la responsable de crear el *viewport* del juego *AirHockey*.

Box2DStage: Encargada de crear el objeto mundo y asignarle la gravedad. Hay que tener en cuenta que esta clase extiende de la clase *Stage* de LIBGDX (ver en el apartado [4]). Al extender de la clase *Stage* otra de sus funciones es gestionar el *viewport* o utilizar el *SpriteBatch* (ver en el apartado [4]) para dibujar. Por ese motivo clases como *AbstractScreen* o *GameScreen* tienen acceso a ella. En el diagrama de clases de la figura 6.5 se puede ver que clases como *Mazo* y *Disco* están enlazados con *Box2DStage*, esto es porque para crear el cuerpo de esto objeto es necesario tener acceso al mundo.

Director: Esta clase se encarga de almacenar todos los datos de la partida, como por ejemplo los goles recibidos y goles marcados. En caso de jugar partida multijugador, también guarda si es *Servidor* o *Cliente*. Esta clase solo tiene esta funcionalidad.

Game1Director: Esta clase se encarga de iniciar la partida, tanto para multijugador como para mono jugador. También se encarga de detectar las colisiones que hay en el mundo, y con esto detectar los goles. Otra tarea que tiene es detectar los inputs del usuario en caso de jugar una partida mono jugador, es decir detectará cuando el usuario hace “clic” o “drag & drop” del mazo.

MultiplayerDirector: Recibe las notificaciones de Bluetooth en las partidas multijugador y notificar a las clases *HostDirector* o *ClientDirector*, dependiendo de si es cliente o servidor.

HostDirector: Recibe datos de la clase *multiplayerDirector* cuando el usuario sea servidor. Los datos que se notificarán a esta clase son entre otros, el usuario que juega como servidor ha marcado gol, si el disco le ha llegado a su dispositivo para ser mostrado, la posición en el eje de las Y y la velocidad de origen, para mostrarlo siguiendo la trayectoria que llevaba en el dispositivo del contrincante. Gestionará estos datos y actualizará la partida. Es decir, por ejemplo, si el servidor marca un gol esta clase recibirá una notificación a través del Bluetooth y se encargará de cambiar el marcador etc. Otra funcionalidad que tiene esta clase es detectar los inputs del usuario que está jugando como servidor.

ClientDirector: Es lo mismo que la clase *HostDirector* pero para gestionar la partida del cliente.

TransmisionPackagePool: Es la que se usa para encapsular los datos que se transmitirán por bluetooth. Los datos que se transmiten son entre otros:

- Velocidad del disco, en los ejes X y Y.
- Posición Y del disco, para saber a qué altura dibujarlo en el otro dispositivo al pasar el disco de un móvil a otro.
- Una variable para saber si la bola ha cambiado de dispositivo.

SoundStatus: Es la que se encarga de gestionar el sonido del juego.

7. Implementación

En este apartado se describen algunos aspectos importantes de la implementación del videojuego.

7.1 Arquitectura de LIBGDX

Antes de comenzar con la explicación es importante remarcar la estructura que usa LIBGDX cuando crea los proyectos. A continuación, se muestra una imagen con los proyectos que se crean para un juego de ejemplo llamado *ball*:

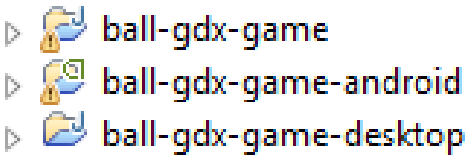


Figura 7.1 Estructura proyecto LIBGDX.

Como se puede ver se genera un "core" (ball-gdx-game), este proyecto será para desarrollar toda la aplicación de LIBGDX. También se genera un proyecto para desktop y otro para Android. Estos dos se generan por defecto, ya que el desarrollo base se hace para la versión de desktop. Por último, se ha generado el proyecto de Android, aquí hay que desarrollar todo lo que sea referente a las APIs de Android, por ejemplo, Bluetooth.

Si se configura el proyecto de LIBGDX para poder exportarlo a otras plataformas se crearán sus respectivos proyectos.

Hay que tener en cuenta que si se quiere trabajar con las APIs de las diferentes plataformas se tendrá que desarrollar por separado, en cada uno de los correspondientes proyectos, el acceso y la gestión a las APIs. Por defecto si no se van a utilizar las APIs solo hará falta desarrollar en el "core".

7.1.2 Arquitectura del juego

A continuación, en la figura 7.2 se puede ver un diagrama donde se muestran diferentes elementos que forman el menú y la escena de juego, con esto también se da a conocer un poco más los objetos disponibles en LIBGDX.

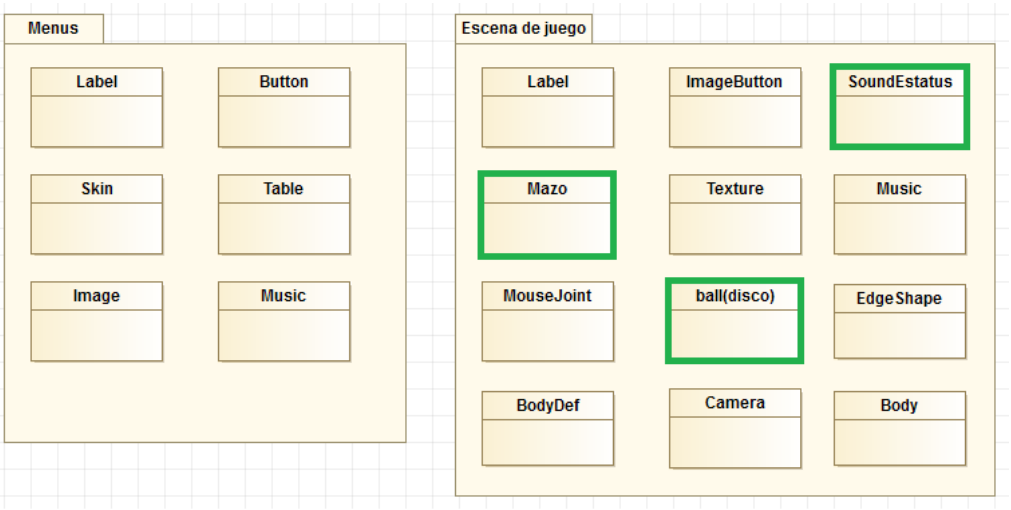


Figura 7.2 Diagrama de arquitectura del juego *AirHockey*.

A continuación, se explica la funcionalidad de cada objeto mencionado en el diagrama, los marcados con un recuadro verde (en blanco y negro se verá gris oscuro) son los creados por mí para el proyecto.

Elementos de los menús

Label: Es un objeto que se utiliza para mostrar texto. A continuación, se muestra una imagen de como se muestra un texto.

Figura 7.3 Ejemplo de un widget tipo label.

Button: Es un objeto para crear botones. A continuación, se muestra una imagen de ejemplo de un botón.

Figura 7.4 Ejemplo de un widget tipo botón.

Table: Es un objeto para crear tablas, los menús están formados por tablas que contienen botones. A continuación, se muestra una imagen de ejemplo de cómo se vería una tabla.

Figura 7.5 Ejemplo de un widget de tipo tabla.

Skin: Almacena las imágenes y los recursos para los widgets de la interfaz, como puede ser el fondo de los botones, la fuente de la letra etc. A continuación, se muestra una imagen de ejemplo de cómo estaría formado un fichero de una *Skin.* Una *Skin* va acompañada de un fichero JSON con las coordenadas de cada carácter o botón para poder mapearlo y recogerlo cuando se necesite.

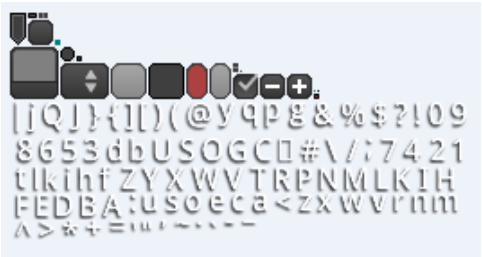


Figura 7.6 Ejemplo de skin.

Image: Es un objeto para dibujar imágenes. A continuación, se muestra un ejemplo de cómo se vería una imagen.

Figura 7.7 Ejemplo de imagen.

ScrollPane: Como el nombre indica sirve para crear un widget tipo scroll, como puede ser una lista, en mi caso lo he utilizado para mostrar al usuario los dispositivos visibles cuando se realiza la búsqueda por bluetooth. En la siguiente imagen se puede ver un scrollpane, pero como la lista que contiene no está llena el scroll está desactivado.

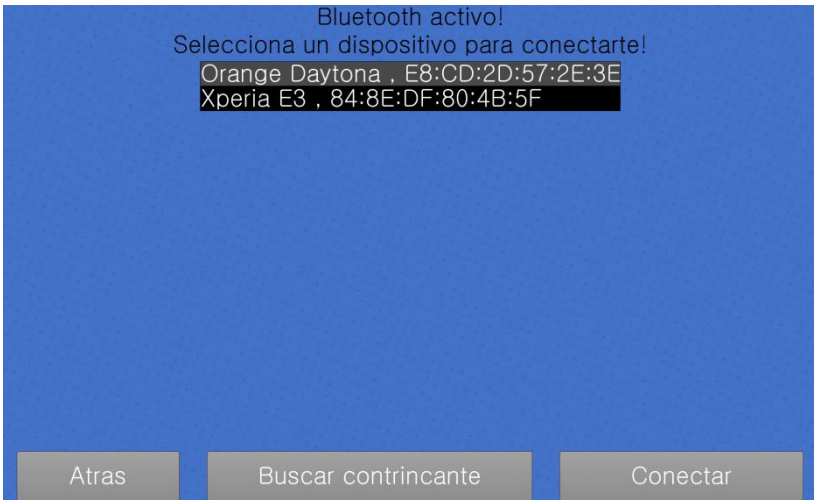


Figura 7.8 Ejemplo de scrollpane.

Elementos de la escena de juego

ImageButton: Es un botón al cual se le añade una imagen de fondo, este puede tener una imagen para los diferentes eventos que se pueden dar, ya puede ser al hacer clic etc. Se ha usado para los botones de pausa, reanudar partida etc.

Texture: Sirve para cargar una textura, luego esta textura la podemos mostrar en la escena como forma de imagen, la diferencia entre imagen y textura es que la textura se puede tratar de diferente manera. Se ha usado para cargar las texturas de los objetos de la escena, como puede ser el mazo, el disco o el fondo que simula el tablero.

Mazo: Este objeto esta creado en este proyecto, extiende de la clase actor y forma el objeto que nosotros vemos representado como un mazo. Se pueden comprobar las propiedades físicas asignadas en el apartado de la explicación de la escena. Ver apartado [7.2.5].

Body: Es el objeto que crea el cuerpo dado un *BodyDef*.

BodyDef: Es el objeto que contiene toda la información que especifica un *Body* y se usará para construirlo.

EdgeShape: Crea una línea (Borde), estos pueden conectarse en cadenas o bucles creando otras nuevas formas o cuerpos más complejos, en nuestro caso se ha utilizado para dibujar las paredes del campo.

Ball (Disco): Este objeto, como el de mazo, se ha creado para este proyecto. El funcionamiento es el mismo, lo único que cambia son las propiedades del cuerpo que lo forma. Se pueden comprobar las propiedades físicas asignadas en el apartado de la explicación de la escena. Ver apartado [7.2.5].

Camera: Este objeto se usa para, representar la cámara de la escena, este objeto puede ser de dos tipos, OrthographicCamera o PerspectiveCamera, explicado en el apartado [7.2.3].

MouseJoint: Crea un enlace entre un punto dado y un objeto, el punto dado en este caso sería el cursor del mouse o en caso de dispositivos móviles la posición donde se toca la pantalla. Se ha utilizado para poder coger los objetos de la escena. Al hacer clic sobre un objeto, se crea un enlace entre la posición del mouse o del dedo y el objeto, esto los une y hace que el objeto se mueva a la posición donde vaya el cursor o el dedo.

Importante, en este proyecto se ha tenido que trasladar las coordenadas de pantalla (donde se ha hecho el clic) a las coordenadas de mundo del videojuego. Para ello se ha utilizado el comando *unproject* sobre la cámara. A la función *unproject* hay que pasarle un *vector3*, es decir un vector de 3 dimensiones, el cual contiene la posición del clic sobre la pantalla y automáticamente las transforma en coordenadas de mundo.

Music: Es la clase para guardar, reproducir y controlar el sonido.

SoundEstatus: Es el objeto que controla todo el sonido del videojuego. Este objeto es un singleton.

7.2 Detalles generales de implementación

En este apartado se explican algunos detalles importantes utilizados para la creación del videojuego, tanto del aspecto visual como funciones propias del framework.

7.2.1 Configuración proyecto Android (AndroidManifest.xml)

En el proyecto de Android hay un fichero llamado AndroidManifest.xml [17], todas las aplicaciones Android, tanto nativas como de LIBGDX deberían tenerlo. Este archivo proporciona información esencial sobre la aplicación, información que el sistema debe tener para poder ejecutar el código de la aplicación.

En este fichero se tienen que declarar los permisos para poder acceder a la API de Android.

Para poder utilizar el bluetooth en el juego del *AirHockey* se han utilizado los siguientes permisos:

- BLUETOOTH_ADMIN: Permite a las aplicaciones descubrir y emparejar dispositivos bluetooth.
- BLUETOOTH: Permite que las aplicaciones se conecten a dispositivos bluetooth emparejados.

Para conseguir que el dispositivo no entre en suspensión durante el juego se ha utilizado el siguiente permiso:

- WAKE_LOCK: Permite el uso de *PowerManager* [12] para mantener el procesador activo o para mantener la pantalla siempre activa.

El fichero AndroidManifest.xml también contiene la información referente a la orientación que tendrá la aplicación en pantalla, hay dos tipos de orientación [16]:

- Landscape: Modo horizontal, la pantalla siempre se mostrará en modo apaisado.
- Portrait: Modo Vertical, la pantalla siempre se mostrará en modo vertical.

En el juego del AirHockey se ha utilizado el modo Landscape, para ello se debe indicar para la *Activity* dentro del AndroidManifest.xml, utilizando la propiedad *android:screenOrientation.* En el fichero Android Manifest también se puede cambiar la versión del sistema operativo Android que se va a aceptar para poder ejecutar la aplicación. Con la versión que he usado de LIBGDX, la versión mínima de Android aceptada es Android 2.3 con nivel de API 9 [10].

7.2.2 Mantener el dispositivo activo

Los dispositivos Android, suelen estar configurados para entrar en suspensión pasado un tiempo si no se usan. Por tanto si iniciamos una aplicación y no la utilizamos, el móvil procederá a ponerse en suspensión.

Para evitar esto primero se debe de activar el permiso WAKE_LOCK explicado en el apartado 7.2.1. Una vez activado el permiso, se debe utilizar la clase *PowerManager,* esta clase es propia de Android y sirve para controlar el estado de la alimentación del dispositivo.

Lo que se debe hacer es indicarle a la clase *PowerManager* que no deje entrar en suspensión al dispositivo, para ello se utiliza la propiedad *WakeLock,* lo que hará es mantener activo aquello que se le indique. A la propiedad *WakeLock* hay que indicarle aquello que quieres mantener encendido, en mi caso es la pantalla, para ello he utilizado *PowerManager.SCREEN_BRIGHT_WAKE_LOCK.*

Es importante remarcar que la utilización de esta clase afectará a la duración de la batería del dispositivo, ya que en nuestro caso siempre mantendrá la pantalla encendida. Por tanto, una vez cerramos el videojuego se debe desactivar.

7.2.3 Cámara

Como se ha comentado anteriormente, existen dos tipos de cámara, OrthographicCamera y PerspectiveCamera. En mi caso, solo he utilizado el OrthographicCamera, Puesto que este se utiliza para entornos 2D. Esta cámara implementa una proyección paralela (ortográfica) y no habrá ningún factor de escala final sin importar donde los objetos se colocan en el mundo. Es decir, los objetos se verán siempre a la misma distancia y no habrá profundidad en el escenario. En cambio, la cámara en perspectiva se suele utilizar para proyecto en 3D, dado que se tendría en cuenta la profundidad del campo y los objetos se verían más alejados o más cercanos en función de su posición.

Si no se aplica una orden especifica en el tamaño del viewport, éste ocupara el 100% de la ventana de juego. En caso de que la relación de aspecto de la ventana no coincida con la del viewport aparecerán barras negras. En mi caso he creado un viewport con un tamaño de 48x32. Puede parecer muy pequeño, pero con este viewport no habrá problemas con las diferentes pantallas, ya que al hacer el viewport más pequeño hará que la cámara solo visualice la parte de la escena que está dentro del viewport. Es decir, la cámara hará un “efecto zoom” sobre la zona del viewport. Si el viewport fuera demasiado grande para las imágenes, estas perderían resolución, a causa de que se tendrían que estirar para adaptarse al viewport, en este caso como es la cámara la que “hace zoom” las imágenes no se ven afectadas.

7.2.4 Splash Screen y Animaciones

Al iniciar la aplicación se ha creado una pantalla de bienvenida, un "splash screen". Un "splash screen", normalmente se utiliza para dar la introducción a la aplicación y se suele mostrar el logo del desarrollador o patrocinadores.

En mi caso he decidido mostrar solo el logo del framework de desarrollo LIBGDX. Para hacer la pantalla un poco más dinámica le he aplicado animaciones.

Las animaciones aplicadas son:

- Al iniciar el "splash screen", la imagen esta invisible por la opacidad, con la animación va disminuyendo la opacidad del logo hasta que se muestra completamente.
- La segunda animación aplicada es cuando se ha terminado de mostrar el logo completamente, este se mueve hacia la parte inferior de la pantalla hasta que desaparece.

Para aplicar animaciones en LIBGDX se debe añadir un *Action* al widget que se quiera animar. Un *Action* de LIBGDX sirve para asignárselo a un objeto y que haga una o varias acciones a través del tiempo. Para ello simplemente se hará con la función *addAction* sobre el widget. A esta función hay que pasarle todas las acciones que queremos que se ejecuten. Las acciones que se asignen al widget se ejecutarán una detrás de otra en el orden de asignación.

Para conseguir el efecto de la opacidad LIBGDX ofrece la acción *fadeIn*, a la cual hay que indicarle el tiempo de animación. Automáticamente LIBDX se encarga de disminuir la opacidad progresivamente en ese tiempo hasta mostrar la imagen.

Para conseguir el efecto de movimiento LIBGDX ofrece la acción *moveBY*, a esta función hay que indicarle la posición de X e Y hacia donde se quiere mover el widget y la duración de la animación.

Por último, si se quiere conseguir que al finalizar una animación se haga algo, existe la acción *run*. A esta función hay que pasarle un Runnable. Un Runnable sirve para definir un subproceso, en mi caso servirá para indicar lo que se quiera hacer una vez mostradas todas las animaciones. Lo que hará es llamar al menú principal y mostrarlo por pantalla.

7.2.5 Escena

En este apartado se va a explicar la composición de la escena de juego del AirHockey, tanto para el escenario de un jugador como para el escenario del multijugador, ya que son diferentes.

Primero de todo hace falta mencionar que en LIBGDX las coordenadas funcionan de la siguiente manera, situando el punto 0.0 en la parte inferior izquierda de la pantalla, es importante remarcar este punto ya que en otros *frameworks* las coordenadas están situadas de diferente forma.

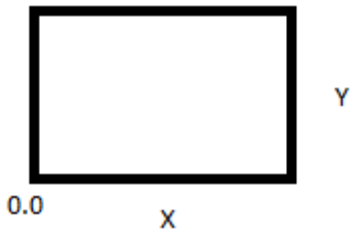


Figura 7.9 Sistema coordenadas LIBGDX.

Escenario de un jugador

Para las físicas se utiliza la librería Box2D que viene integrada en el *framework*. La gravedad tanto en el eje de las X como de las Y es 0.0f, ya que se simula un escenario visto desde arriba, por tanto, donde se aplica la física es en los rebotes de los objetos.

Físicas aplicadas a los cuerpos:

- **Mazo:** El cuerpo de este objeto es circular. Por tanto, se ha utilizado el objeto *CircleShape* para crearlo. Para definir un cuerpo necesitamos declarar un *FixtureDef* al cual se le ha asignado la densidad, fricción, restitución y el tipo de shape (en este caso será el *CircleShape*). Una vez tenemos el *FixtureDef* se lo asignamos al *Body*.
    - Densidad: (Restitution), restitución del objeto (Elasticidad), usualmente situado en un rango de [0, 1].
    - Fricción: (Friction) es el coeficiente de fricción del objeto, usualmente situado en un rango de [0, 1].
    - Restitución: (Density) densidad del objeto, usualmente en kg/m^2.

  Los valores del *FixtureDef* son: densidad 0f, fricción 0f, restitución: 0.1f, shape de tipo *CircleShape*.

- **Disco:** Los valores del *FixtureDef* son, densidad 0f, fricción 0f, restitución 0.7f, shape de típo *CircleShape*.

  Como vemos en este caso, la restitución, es decir la elasticidad o reacción a los rebotes es mucho más alta que en el caso del mazo. Esto es porque, interesa que el disco tenga mucha reacción a los rebotes, es decir rebote mucho en las colisiones.

El tablero de juego se muestra en la siguiente imagen.

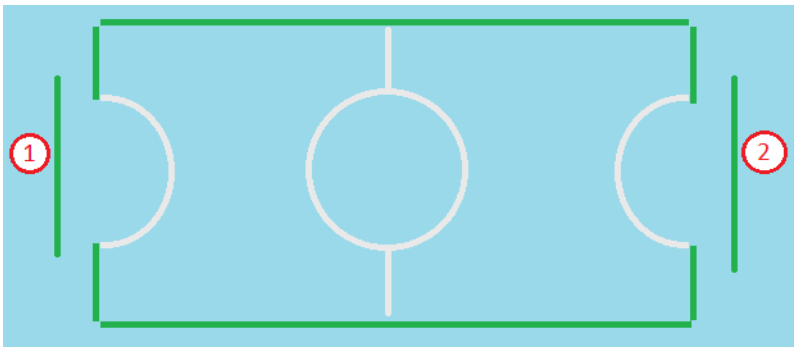


Figura 7.10 Escena mono jugador.

Lo marcado en verde (en blanco y negro se verá en gris oscuro) son los *EdgeShapes* que forman la escena, estos *EdgeShapes* son los que forman la pared del escenario. El que está marcado con un "1" es el que se encarga de detectar el gol en contra del usuario. El que está marcado con un "2" es el que se encarga de detectar el gol en contra de la IA.

Escenario de multijugador

La gravedad en el escenario multijugador es igual que el escenario de un jugador. El tablero del multijugador se explica en la siguiente imagen:

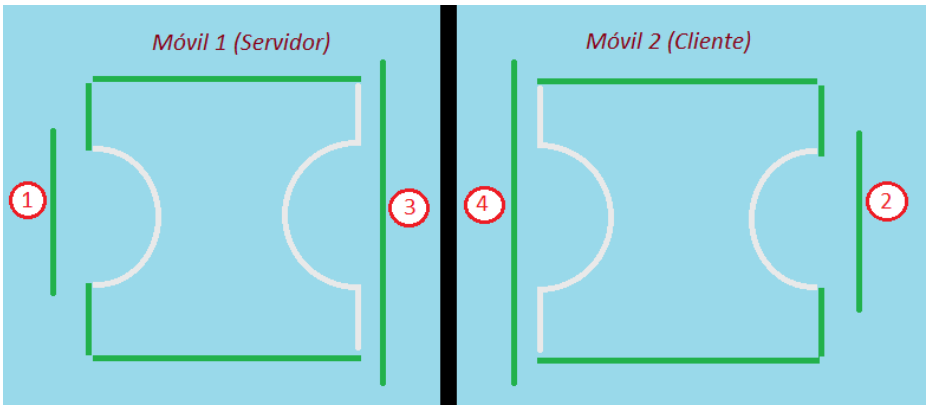


Figura 7.11 Escena multijugador.

La imagen está dividida en dos, por una franja negra. A la izquierda se muestra la estructura de la escena del móvil que hace de servidor y a la derecha la estructura de la escena del móvil cliente.

Como en la imagen anterior los *EdgeShapes* marcados en verde (en blanco y negro gris oscuro) forman las paredes.

- El *EdgeShape* marcado con un 1 es para detectar los goles en contra del usuario que está jugando con el campo de la izquierda.
- El *EdgeShape* marcado con un 2 es para detectar los goles en contra del usuario que está jugando en la derecha.

- El *EdgeShape* marcado con un 3 es para detectar cuando el disco llega al borde de la pantalla del servidor. En este caso el disco se elimina del escenario del servidor y se posicionará en la pantalla del cliente, con la misma velocidad en los ejes de las X y Y, también con la misma posición en el eje de las Y. Con esto se consigue un efecto visual que dará la sensación de que el disco pasa de un móvil al otro.
- El *EdgeShape* marcado con un 4 funciona igual que el 3, pero detecta cuando el disco tiene que pasar del móvil del cliente al del servidor.

La textura de fondo, que simula el tablero se ajusta al tamaño de pantalla, cogiendo la altura y la anchura de la pantalla del dispositivo. Funciona con dispositivos de pantallas con menor resolución que HD, con HD y FULL HD.

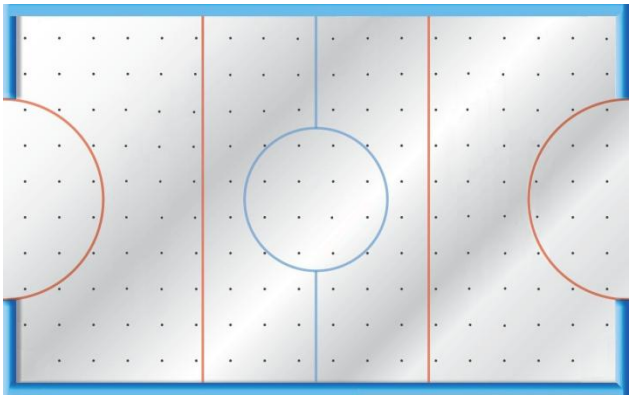


Figura 7.12 Imagen de fondo del juego AirHockey.

7.2.6 Texto

Para utilizar texto en LIBGDX se utilizan los *BitmapFont*, es decir un mapa de bits. Para ello es necesario un archivo de imagen que contenga los caracteres dibujados como una imagen (png) y un archivo en formato de texto (fnt), que describe donde se encuentra cada carácter en la imagen.

- Ejemplo de imagen con los caracteres dibujados

Figura 7.13 Imagen con los caracteres.

- Ejemplo de fichero .fnt

```
info face="GulimChe" size=52 bold=0 italic=0 charset="" unicode=0 stretchH=100 smooth=1 aa=1 padding=1,1,1,1 spacing=-2,
common lineHeight=61 base=49 scaleW=512 scaleH=512 pages=1 packed=0
page id=0 file="prueba48.png"
chars count=97
char id=10      x=0    y=0    width=0    height=0    xoffset=0    yoffset=0    xadvance=0    page=0    chnl=0
char id=32      x=0    y=0    width=0    height=0    xoffset=-1   yoffset=44   xadvance=17   page=0    chnl=0
char id=33      x=290  y=93   width=7    height=41   xoffset=5    yoffset=6    xadvance=17   page=0    chnl=0
char id=34      x=499  y=134  width=11   height=17   xoffset=4    yoffset=3    xadvance=19   page=0    chnl=0
char id=35      x=120  y=51   width=35   height=42   xoffset=2    yoffset=5    xadvance=39   page=0    chnl=0
char id=36      x=119  y=0    width=27   height=49   xoffset=2    yoffset=2    xadvance=31   page=0    chnl=0
char id=37      x=331  y=0    width=41   height=43   xoffset=3    yoffset=5    xadvance=46   page=0    chnl=0
char id=38      x=155  y=51   width=33   height=42   xoffset=2    yoffset=6    xadvance=35   page=0    chnl=0
char id=39      x=506  y=0    width=5    height=17   xoffset=5    yoffset=3    xadvance=14   page=0    chnl=0
char id=40      x=0    y=0    width=13   height=51   xoffset=4    yoffset=1    xadvance=19   page=0    chnl=0
char id=41      x=13   y=0    width=14   height=51   xoffset=2    yoffset=1    xadvance=20   page=0    chnl=0
char id=42      x=126  y=173  width=22   height=21   xoffset=2    yoffset=16   xadvance=26   page=0    chnl=0
char id=43      x=98   y=173  width=28   height=28   xoffset=2    yoffset=13   xadvance=32   page=0    chnl=0
char id=44      x=216  y=173  width=8    height=14   xoffset=4    yoffset=38   xadvance=17   page=0    chnl=0
char id=45      x=272  y=173  width=28   height=5    xoffset=2    yoffset=24   xadvance=32   page=0    chnl=0
char id=46      x=265  y=173  width=7    height=8    xoffset=5    yoffset=39   xadvance=17   page=0    chnl=0
char id=47      x=99   y=0    width=20   height=49   xoffset=1    yoffset=2    xadvance=22   page=0    chnl=0
char id=48      x=264  y=93   width=26   height=41   xoffset=2    yoffset=6    xadvance=29   page=0    chnl=0
char id=49      x=90   y=93   width=13   height=41   xoffset=6    yoffset=6    xadvance=30   page=0    chnl=0
char id=50      x=373  y=93   width=26   height=40   xoffset=2    yoffset=6    xadvance=30   page=0    chnl=0
char id=51      x=103  y=93   width=26   height=41   xoffset=2    yoffset=6    xadvance=30   page=0    chnl=0
char id=52      x=129  y=93   width=30   height=41   xoffset=0    yoffset=6    xadvance=30   page=0    chnl=0
char id=53      x=399  y=93   width=27   height=40   xoffset=2    yoffset=7    xadvance=30   page=0    chnl=0
```

Figura 7.14 Fichero de mapeo.

Como se puede ver en las imágenes anteriores hay una imagen PNG con el dibujo de los caracteres de la fuente, y un fichero de mapeo (fnt) el cual hará posible acceder a cada carácter.

Una vez generado el fichero hay que ponerlo en el proyecto de LIBGDX. Para utilizarlo se tiene que crear un *LabelStyle*, este definirá el estilo de la *label* que contendrá el texto. El *LabelStyle* tiene dos parámetros:

- font: Define la fuente que se utilizará para el estilo. Se le añade un *BitmapFont* que hay que cargarlo con el fichero ".fnt" generado por *Hiero.* Este *BitmapFont* será la fuente dentro del *LabelStyle.*
- fontColor: Aquí se define el color.

Una vez definido el *LabelStyle* sólo hace falta crear la *label* donde pondremos el texto, creándola con el *LabelStyle.* Para generar este tipo de ficheros hay una herramienta llamada Hiero v5, se puede encontrar en el apartado de referencias [6]. Esta herramienta es totalmente gratuita, y la podemos descargar directamente desde Github. Esta ofrecida directamente por la página de LIBGDX como herramienta de soporte para creación de textos.

7.2.7 Multijugador

Para poder implementar la funcionalidad en modo multijugador se ha utilizado la API del bluetooth de Android. Se ha tenido que crear una clase para que controle todas las funciones que nos interesan, se puede ver en el diagrama de clases de la figura 6.5 llamada *BluetoothManager.*

Para implementar el bluetooth de un dispositivo en LIBGDX hay que tener en cuenta que este código será no multiplataforma, ya que será propio de la API del dispositivo. Para implementarlo hay que crear una interfaz en el proyecto de LIBGDX. La interfaz contendrá los métodos para acceder a las propiedades del bluetooth como puede ser para comprobar si el bluetooth está activo, si esta visible, para obtener los dispositivos conectados etc. Básicamente hay que desarrollar los métodos que sean necesarios para nuestro proyecto.

Esta interfaz se debe implementar en el proyecto nativo, para cada plataforma. Es decir, en caso de querer desarrollar el videojuego para Android, se deberá de crear en el proyecto de

Android, en caso de querer desarrollar para iOS se deberá implementar en el proyecto para iOS.

Antes de empezar, es importante explicar el *BluetoothAdapter* de la API de Android [24]. El *BluetoothAdapter* representa el adaptador de bluetooth del dispositivo local. Este adaptador se controla con la clase *BluetoothManager,* pero esta clase se encuentra en el proyecto de Android, aquí es donde entra en juego la clase *BluetoothInterface* situada en el proyecto de LIBGDX. *BluetoothInterface* comunicará el código de LIBGDX con la parte nativa de Android.

BluetoothAdapter permite realizar tareas fundamentales, como iniciar detección de dispositivos, consultar una lista de dispositivos enlazados, crear un *BluethoothServerSocket* para escuchar las solicitudes de conexión de otros dispositivos etc.

A continuación, se describen las utilidades que he necesitado de la API del *BluetoothAdapter*:

- *startDiscovery*: Inicia el proceso de descubrimiento del dispositivo remoto.
- *isDiscovering*: Devuelve verdadero si el adaptador Bluetooth local se encuentra actualmente en proceso de detección de dispositivos.
- *cancelDiscovery*: Hace que el dispositivo local deje de estar visible, otros dispositivos no podrán detectarlo ni verlo por medio del bluetooth.
- *listenUsingRfcommWithServiceRecord*: Sirve para crear una escucha, usa la seguridad de bluetooth RFCOMM con Registro de servicio. El protocolo de Bluetooth RFCOMM es un simple conjunto de protocolos de transporte, hechos en la parte superior del protocolo L2CAP. RFCOMM ofrece transporte de datos binarios y permite hasta sesenta conexiones simultáneas a un dispositivo de Bluetooth a la vez. El protocolo se basa en el estándar ETSI TS 07.10. En este proyecto no se va a entrar en el funcionamiento interno del bluetooth, pero para saber más del bluetooth se puede consultar el siguiente documento [11].
- *createRfcommSocketToServiceRecord*: Crea un socket para la comunicación entre el servidor que está ofreciendo el servicio y el cliente que intenta conectarse, se necesita utilizar el mismo UUID, este UUID se genera una vez, al desarrollar la aplicación y se añade por código.

  El UUID es un identificador único universal inmutable. Un UUID representa un valor de 128 bits [22]. El UUID está formado por una cadena que se compone por 32 dígitos hexadecimales (0-9 y a-z), estos dígitos están en grupos y separados por guiones, por ejemplo:

  560a8451-a29c-41d4-a716-544676554400

  Para crear este código hay páginas web y programas que generan este código automáticamente y de manera aleatoria [23].

Para el intercambio de datos entre dispositivos vía bluetooth existe el *BluetoothSocket* [20]. La interfaz de los sockets de bluetooth es similar a la de sockets TCP: *Socket* y *ServerSocket*. En el lado del servidor hay que crear un *BluetoothServerSocket* para crear un socket de escucha. Cuando una conexión es aceptada por el *BluetoothServerSocket*, devolverá un *BluetoothSocket*

para administrar la conexión. En el lado del cliente se debe utilizar el *BluetoothSocket* para iniciar y administrar una conexión.

Para recibir y leer el flujo de datos que envíen al dispositivo se utiliza *InputStream*. Para enviar los datos a un destinatario se utiliza *OutputStream*. El intercambio de datos se hace en bytes, para convertir a bytes los datos a intercambiar, y a la inversa, se ha utilizado la clase *SerializationUtils* propia de java, esta clase serializa automáticamente y deserializa.

Antes de comenzar con la explicación de cliente-servidor es importante remarcar que el protocolo Bluetooth es muy parecido a otros como el TCP. A continuación, se describe más a fondo el esquema de cliente-servidor [32].

- El servidor crea un socket (*BluetoothServerSocket*) y lo pone a la escucha de peticiones de conexión

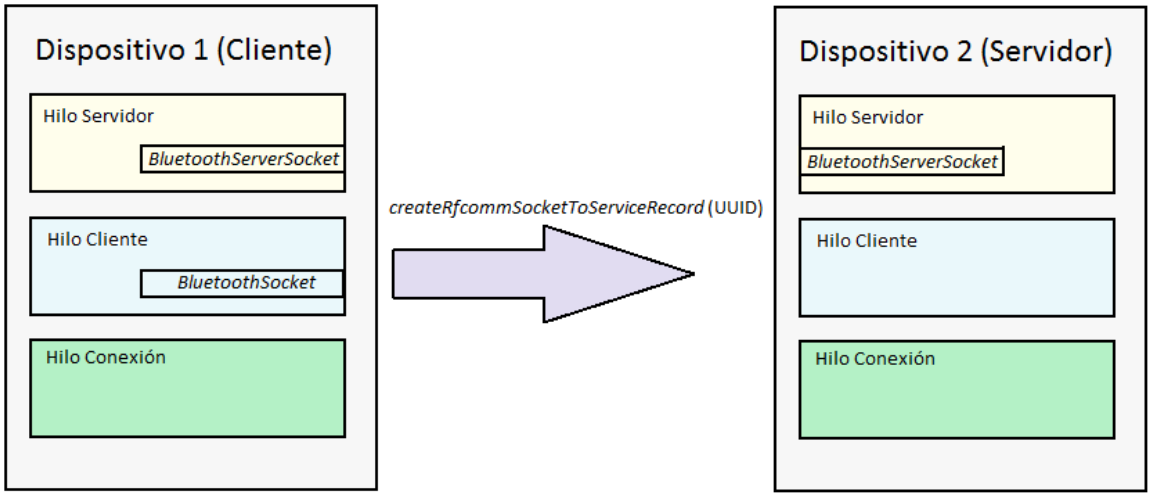


Figura 7.15 El cliente abre el socket para la conexión.

- Una vez abierto el socket, el cliente crea una solicitud de conexión (connect). Esta llamada es bloqueante, por tanto, el resultado de esta llamada solo podrá ser éxito o fallo.

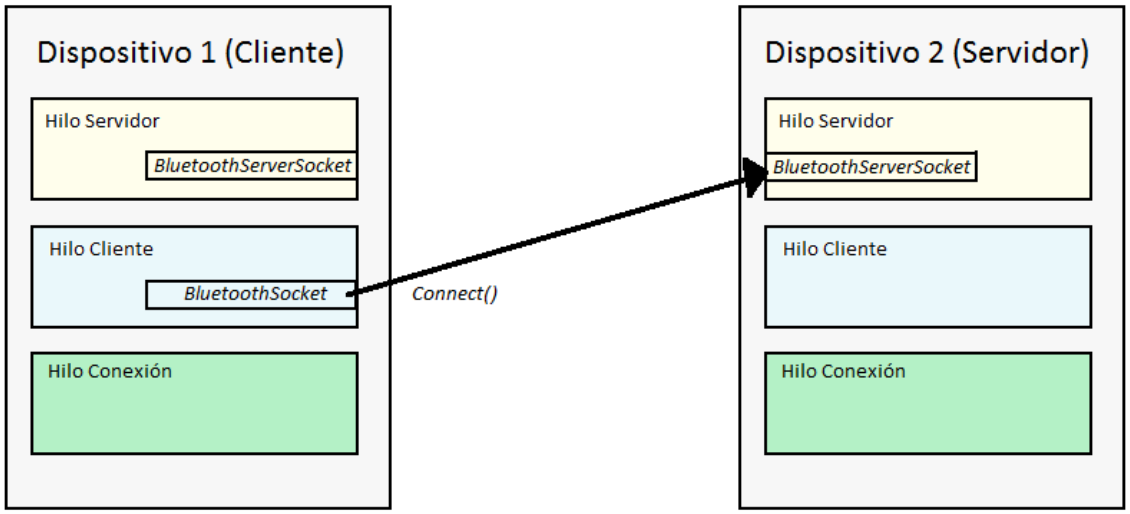


Figura 7.16 Cliente hace solicitud de conexión.

- El servidor acepta la conexión (accept) y lo notifica al cliente. A continuación, abre un socket de tipo *BluetoothSocket*.

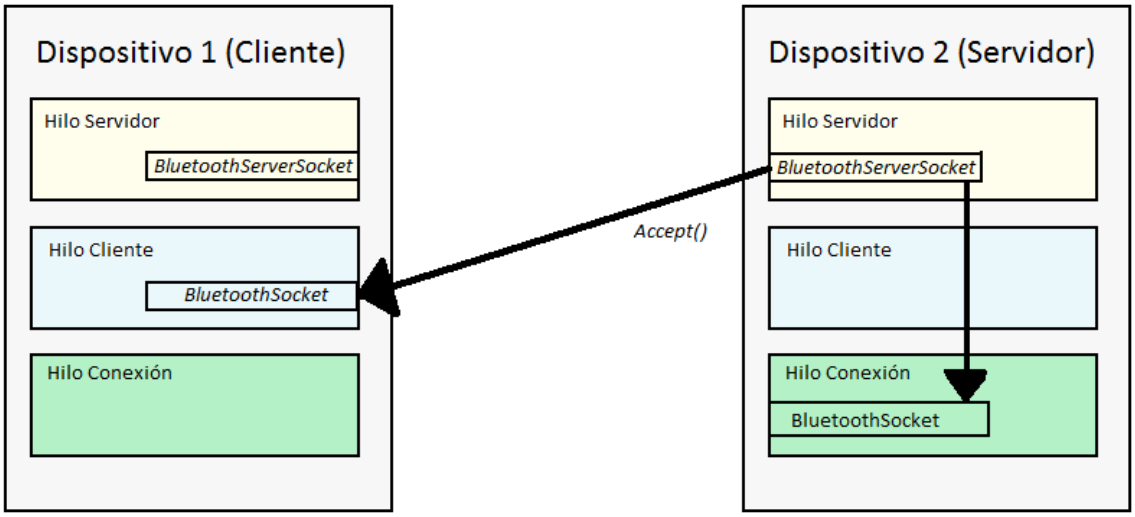


Figura 7.17 El servidor acepta la conexión del cliente.

- El cliente recibe la notificación del servidor, en este caso la conexión será correcta y saldrá de la espera ocupada.
- El cliente y el servidor obtienen los flujos de entrada y salida de sus respectivos sockets.

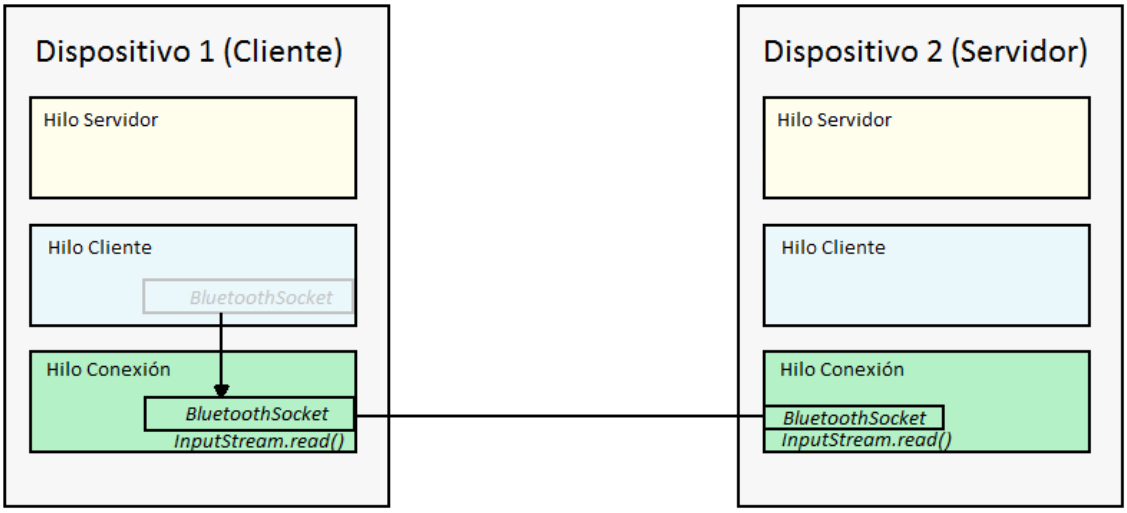


Figura 7.18 Conexión establecida.

- El hilo de la conexión comienza a observar el flujo de entrada esperando obtener datos. Si llegan datos, se envían a las funciones que lo procesarán para informar a la partida de juego a través de un *Handler*.
- Para enviar datos se realiza escribiendo en el flujo de salida, utilizando un *OutputStream*.

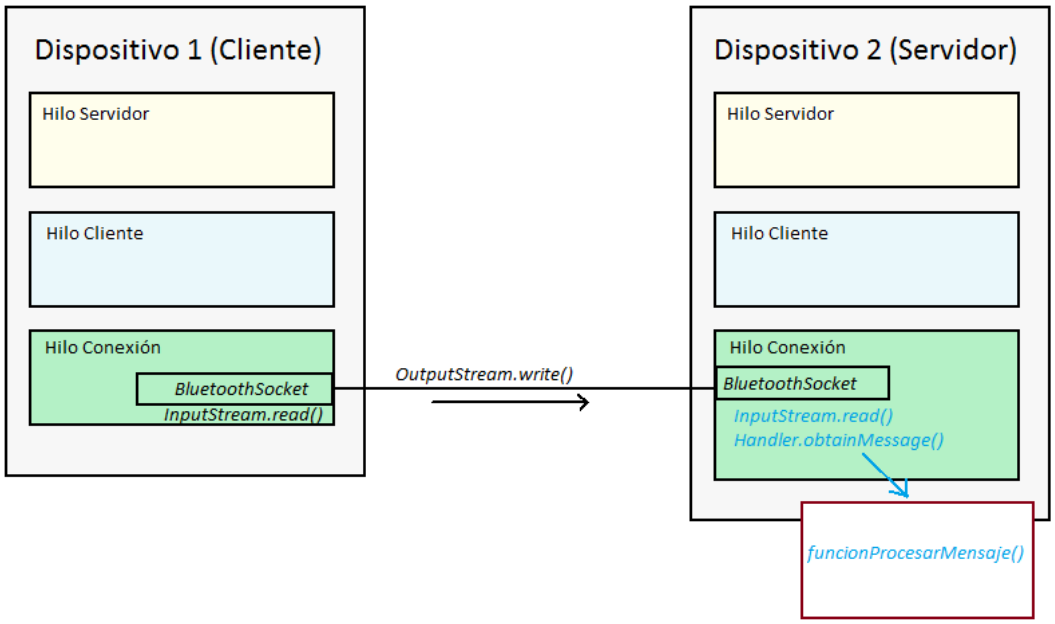


Figura 7.19 Envío de datos del cliente al servidor.

7.2.8 IA del mazo

En el apartado de un jugador se ha implementado una IA básica para poder mover el *mazo enemigo*, a continuación, se detalla brevemente la implementación.

- Si el disco en el eje de las X está en una posición cercana al *mazo enemigo*, este golpeará el disco.
- Si el mazo enemigo se aleja mucho de su portería y no está atacando volverá a su posición original.
- Si el mazo enemigo ya ha golpeado el disco vuelve a su posición original.
- Si la posición del mazo enemigo es su posición original, la velocidad en los ejes X y Y será 0, por tanto, dejará de moverse.

A continuación, se muestra un diagrama de estados para dejar más claro el funcionamiento de la IA.

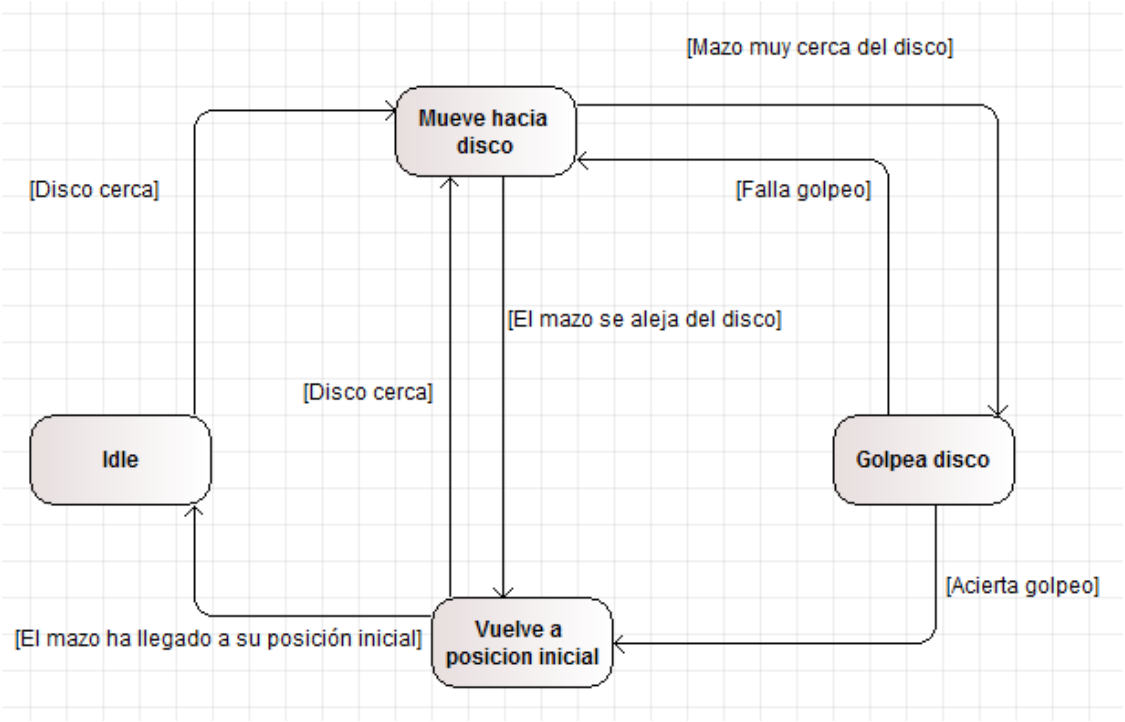


Figura 7.20 “FSM – Finite State Machine” que define el comportamiento de la IA

7.2.9 Audio

En este apartado se va a explicar el proceso para añadir sonido al videojuego *AirHockey*.

Para poder tener un control en todo momento del sonido, se ha creado una nueva clase llamada *SoundEstatus*. Esta clase se ha creado como un singleton. Es decir, sólo se podrá instanciar una vez y proporcionará un punto de acceso global a ella. Con esta clase se controla si algún sonido está reproduciéndose en cada momento. Otra tarea de esta clase es guardar los sonidos a reproducir.

LIBGDX proporciona el objeto *Audio.* Esta clase encapsula la creación y administración de audio. El objeto *Audio* proporciona la interfaz *Music* que permite la reproducción de sonido y música. La interfaz *Music* representa un archivo de audio streaming. Esta interfaz permite pausar, reanudar y reproducir el archivo de audio. Para reproducir el sonido se usa la función *play.* Para pausar el sonido se usa la función *pause*. Para detener la reproducción se usa la función *stop*. A parte de estas funciones, LIBGDX nos ofrece algunas más como puede ser comprobar si el archivo de sonido se está reproduciendo, o cambiar el volumen. Es importante, cuando se acaba con el uso del sonido deshacerse de él para liberar espacio en memoria con el comando *dispose* de LIBGDX.

En el videojuego se ha añadido una música para el menú principal. Esta música al iniciar la partida se pausa y prepara los sonidos de la escena de juego. En la escena de juego lo que encontraremos, es sonido al golpear o chocar el disco con algún objeto de la escena y cuando haya un gol sonará el típico sonido de cuando el disco cae dentro de la portería.

7.3 Detalles de implementación de GUI

En este apartado se explican los temas relacionados con el diseño y la interfaz del usuario.

Antes de comenzar con la explicación, se mostrarán algunos detalles que han sido necesarios para crear los elementos de la interfaz de usuario.

- El software utilizado para la creación y modificación de las imágenes ha sido Photoshop.
- Las imágenes tienen una resolución alta, para que puedan verse correctamente en pantallas de menor y mayor calidad.
- Se ha dado un estilo minimalista a los botones y a la interfaz, para no sobrecargar la pantalla y conseguir un diseño atractivo.

7.3.1 Adaptación de la GUI a diferentes pantallas

Existen dispositivos móviles con diferentes tipos de pantalla. A la hora de desarrollar la aplicación es importante tener esto en cuenta y adaptar la aplicación para todas ellas. Hay que tener en cuenta que los widgets de LIBGDX (texto o botones), no se adaptan en función del viewport, sino que lo hacen mediante los pixeles de la pantalla.

Para gestionar esto desde código, simplemente hace falta comprobar el tamaño que tiene la pantalla y asignarle un tamaño diferente a la fuente de texto o botones. Para gestionar el tamaño de botones se puede usar la siguiente formula, la cual los adaptará con la misma proporción en las diferentes pantallas:

```
Gdx.graphics.getWidth()*TamañoX/100, Gdx.graphics.getHeight()*TamañoY/100
```

- Esta fórmula hay que introducirla en la propiedad *size* del botón.
- Donde *Gdx.graphics.getWidth* es una función propia de LIBGDX la cual recoge la anchura de la pantalla del dispositivo y *Gdx.graphics.getHeight.*

Para adaptar el texto no hay una formula genérica, se debe comprobar la dimensión de la pantalla y asignarle un tamaño a la fuente según el resultado que se quiera obtener.

Por ejemplo, en este proyecto para el título del menú principal (AirHockey), en pantallas de resolución 854 x 480 píxeles, se ha utilizado una fuente de 52 y para pantallas con resolución 1080 x 1920 píxeles se ha utilizado una fuente de 72.

7.3.2 HUD

Es la parte de la interfaz de usuario que se utiliza para mostrar el estado del juego. En nuestro caso para el marcador de la partida y el botón de pausa.

Al inicio del desarrollo se diseñó para que los datos del marcador fueran simples *Labels* que mostraran el resultado, pero una vez se creó el escenario no quedaban integradas en el entorno. Finalmente se decidió cambiar las *Labels* por imágenes.

A continuación, se muestra el resultado final.

Figura 7.21 HUD.

Como se puede ver en la imagen, en el centro del HUD tenemos el botón para pausar la escena. Cuando se pausa la escena todos los objetos de la escena dejaran de moverse y aparecerá un menú de pausa. Al reanudar la partida todos los objetos volverán a recobrar su velocidad original. A la Izquierda del botón tenemos el marcador con los goles del "player" que juega en el campo de la izquierda, y a la derecha del botón tenemos el marcador del que juega en la derecha.

7.3.3 Sombras

En la escena de juego se decidió crear sombras para todos los objetos, con el objetivo de dar un mayor realismo y de profundidad al juego. La sombra se crea con una imagen gris transparente que se posiciona debajo del objeto simulando el sombreado, con esto se consigue el efecto oscuro de la sombra y se transparenta lo que haya debajo. A continuación, se muestra el resultado.

Figura 7.22 Sombras de los objetos.

8. Resultados

En este apartado se muestran los resultados del videojuego *AirHockey* desarrollado para el proyecto. Se muestra la interfaz y se hace una breve explicación del funcionamiento, también se estudia el rendimiento del videojuego en diferentes dispositivos con un *benchmark*, y por último se compara el *framework* LIBGDX con Unity y AndEngine con otro *benchmark*.

8.1 Resultados del videojuego *AirHockey*

En este apartado se mostrará y analizará el resultado final del videojuego.

Más adelante, se muestran imágenes del videojuego y se explicarán los detalles más relevantes.

Al iniciar la aplicación he creado una pantalla de introducción (Splash Screen) que muestra el logo del framework. Esta pantalla tiene animaciones aplicadas, véase el apartado 7.2.4.

Figura 8.1 Splash Screen.

Una vez iniciado el videojuego aparecerá el menú principal. Seguidamente, se muestra la imagen del menú principal.

Figura 8.2 Menú principal.

Si el usuario selecciona una partida en modo “un jugador”, se iniciará una partida donde el mazo izquierdo de la pantalla, estará controlado por el propio usuario. Así mismo, el mazo derecho de la pantalla, será controlado por la IA.

El usuario deberá arrastrar su mazo y golpear el disco para introducirlo en la portería contraria.

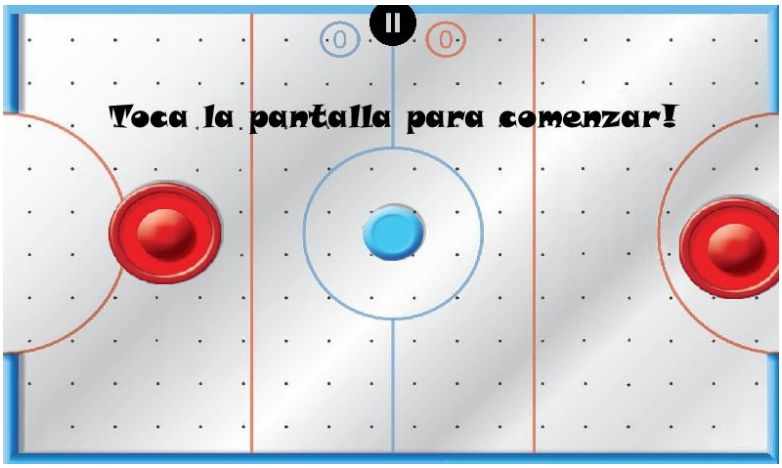


Figura 8.3 Partida un jugador.

Si el usuario pulsa el botón de pausa, todos los objetos en pantalla se quedarán quietos y se abrirá el menú. Aquí el usuario tiene un botón para reanudar la partida u otro botón para volver al menú principal.

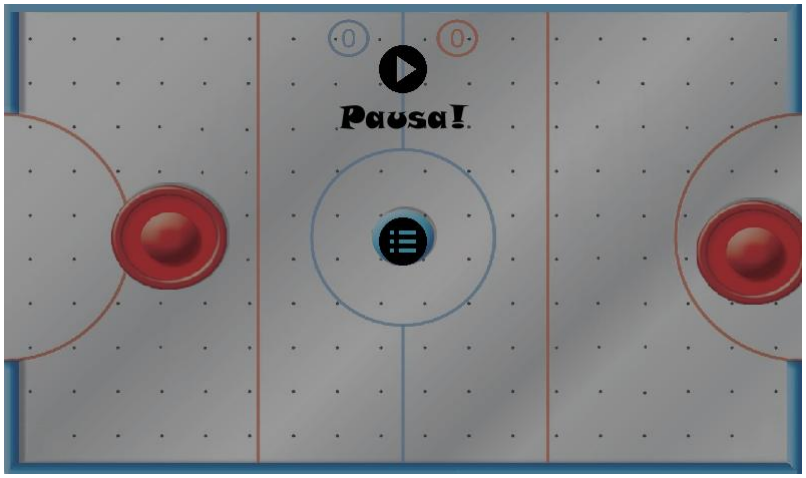


Figura 8.4 Pausa partida un jugador.

En cambio, cuando el usuario desde el menú principal selecciona la opción multijugador, se abrirá un nuevo menú donde podrá seleccionar entre tres opciones: Servidor, Cliente y Atrás. Es importante remarcar que a esta pantalla no se podrá acceder si el dispositivo no es Android y no tiene bluetooth, en este caso se mostrará un mensaje en pantalla "*No se puede jugar en modo multijugador, Bluetooth no soportado en este dispositivo*" para avisar al usuario.

Figura 8.5 Menú pantalla multijugador.

Si el usuario selecciona el botón de "Servidor" se abrirá una pantalla de la API de bluetooth de Android pidiendo permiso al usuario para activar el bluetooth y ponerlo visible. Al aceptar esta opción se abrirá una nueva ventana donde se nos indicará que se está esperando la conexión de un contrincante.

Figura 8.6 Esperando conexión de un cliente.

Si el usuario selecciona la opción de "Cliente" en la pantalla de multijugador, se abrirá una nueva pantalla que nos mostrará una lista con los dispositivos enlazados. En caso de querer buscar uno nuevo, hay un botón para realizar la búsqueda.

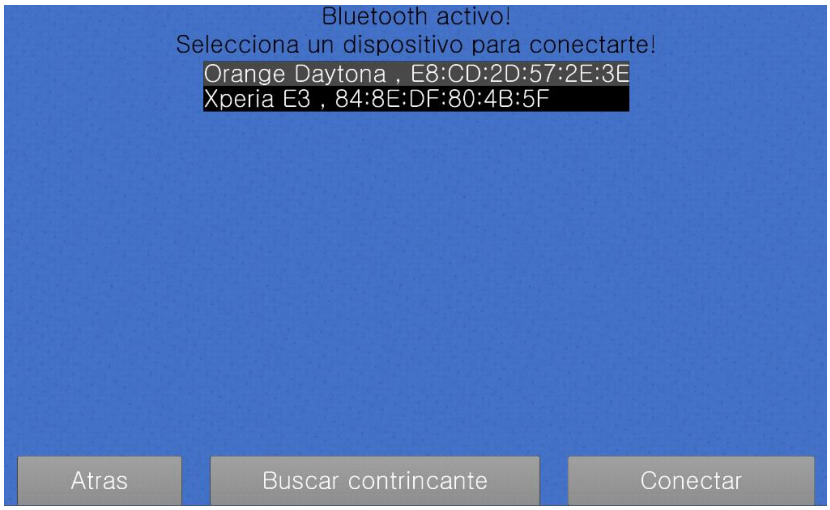


Figura 8.7 Menú partida multijugador (cliente).

Una vez se iniciado la partida, ya sea como servidor o cliente, se mostrará el tablero dividido en los dos móviles como se puede ver en la siguiente imagen.

Figura 8.8 Multiplayer.

En cuanto el disco se aproxime al borde de la pantalla, desaparecerá de nuestro campo y pasará al otro dispositivo móvil, manteniendo su velocidad y su trayectoria, creándose así un buen efecto visual.

En tanto a la conexión bluetooth del contrincante, si durante la partida multijugador falla, se mostrará el mensaje “Problema en la conexión, vuelva al menú principal” como se muestra en la siguiente figura.

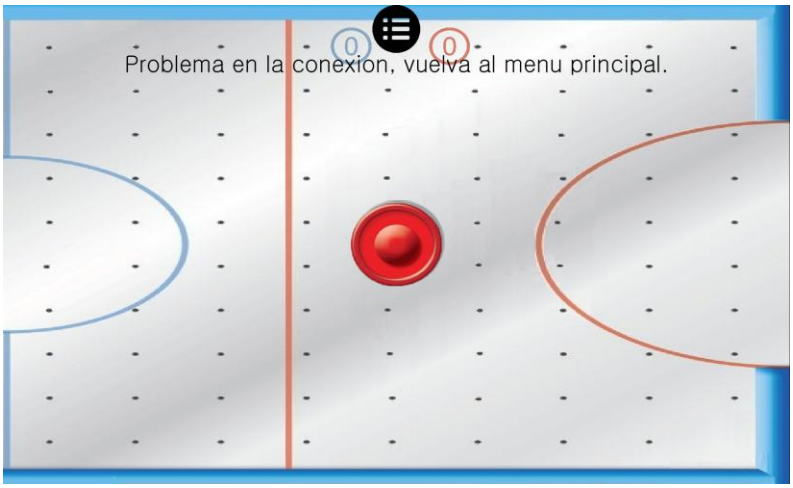


Figura 8.9 Perdida de conexión del contrincante.

Por último, si se selecciona en la pantalla principal la opción de ayuda, se mostrará una pantalla con una pequeña explicación del videojuego.

Figura 8.10 Imagen menú ayuda.

En el siguiente enlace se puede ver en funcionamiento el videojuego *AirHockey*:

- Video partida mono jugador.
- Video partida multijugador.

Una vez finalizado el desarrollo del videojuego, se ha probado en diferentes dispositivos, para verificar su rendimiento, y para revisar el aspecto visual en diferentes pantallas.

Dispositivos donde se ha probado el videojuego:

Dispositivo	**Procesador**	**RAM**	**Gráfica**	**Plataforma**	**Resolución de Pantalla**
Meizu pro 6	MediaTek Helio X25	4GB	ARM Mali-T880	Android	1080 x 1920 px
Sony xperia E3	Qualcomm Snapdragon 400	1GB	Adreno 305	Android	854 x 480 px
LG L5	Qualcomm MSM7225A Snapdragon	512MB	Adreno 200	Android	320 x 480px
Sony xperia Z2	Qualcomm Snapdragon 801	3GB	Adreno 330	Android	1080 x 1920 px
Samsung S8500 Wave	Procesador ARM Cortex A8 1GHz	384 MB	PowerVRSGX540	Android	320 x 480 px

- En esta tabla es importante remarcar que el Samsung wave originalmente monta un sistema operativo Bada OS, pero se ha modificado mediante una ROM para cambiarlo por Android. Esta prueba se hizo con el objetivo de probar el videojuego en un dispositivo móvil mucho más ajustado, ya que el hardware que monta es muy limitado y además el sistema operativo no corre de manera nativa.

BenchMark del juego *AirHockey*

Para determinar el rendimiento del videojuego en los diferentes móviles mostrados en la tabla anterior, se ha realizado un pequeño benchmark para ver los FPS que alcanza el videojuego en cada uno de los dispositivos.

Dispositivo	**FPS**
Meizu pro 6	62 FPS
Sony xperia E3	40 FPS
LG L5	30 FPS
Sony xperia Z2	55 FPS
Samsung S8500 Wave	25 FPS

Para poder ver este resultado se ha creado un log en el cual se iban mostrando los FPS del juego. Se ha cogido aproximadamente la media de los resultados. Se puede ver que los

resultados han sido muy buenos obteniendo en casi todos los casos unos FPS por encima de los 30 FPS.

Se puede ver que el móvil con menos capacidad (LG L5) ha dado una media de unos 30 FPS, en el mercado actualmente hay pocos móviles que tengan un hardware inferior a este. El teléfono móvil Samsung S8500 Wave ha dado 25 FPS con algunas caídas por debajo de estos, se ha mostrado un poco más irregular, pero hay que tener en cuenta que este dispositivo no es Android nativo y es el que tiene el hardware inferior, por tanto tendrá siempre un rendimiento menor que los demás.

8.2 Benchmark LIBGDX VS Unity VS AndEngine

A la hora de programar un videojuego es importante tener en cuenta los fotogramas por segundo que mostrará para conseguir un buen rendimiento. Los FPS (fotogramas por segundo), es la velocidad a la cual un dispositivo muestra imágenes, estas imágenes son llamadas fotogramas.

El sistema perceptivo humano procesa entre 10 y 12 imágenes separadas por segundo. Aunque las capta individualmente, si se supera este número de imágenes por segundo se percibirán fusionadas como movimiento [8].

A pesar de que 10 FPS sería un mínimo de frames para percibir un movimiento animado para conseguir una ilusión de movimiento fluido, se deben proyectar como mínimo 24 imágenes por segundo. Este fenómeno se llama persistencia retiniana. Si se superan las 40 imágenes por segundo conseguimos el máximo realismo de movimiento, por encima de estos FPS, nuestro sistema perceptivo no percibe un cambio significativo.

Para comprobar el rendimiento del *framework* LIBGDX se han realizado un *benchmark* en diferentes móviles. Para ello, se ha desarrollado un escenario simple en las plataformas LIBGDX, Unity 3D y AndEngine. Con esto se los hace trabajar de manera intensiva y se comprueban los FPS que se obtienen en un momento dado, el objetivo es realizar una comparación de rendimiento entre ellos.

Estas pruebas se realizarán en LIBGDX, AndEngine y Unity3D, creando una aplicación para cada una de ellas.

El escenario utilizado es el siguiente (Para todas las pruebas):

Gravedad:

- Gravedad en el eje de las X: 0
- Gravedad en el eje de las Y: -9.8

Bola:

- La bola tiene aplicados los siguientes parámetros físicos:
    - Restitución: 0.5f
    - Fricción: 0.5f
    - Densidad: 1.0f
    - Radio: 1.2f

Para realizar un *benchmark* más real, los proyectos generados se probarán en 2 móviles diferentes, gama alta y gama media.

Móviles utilizados para los benchmarks:

Gama alta: Meizu pro 6[26]

Especificaciones:

- Procesador MediaTek Helio X25

  ARM® Cortex®-A53™1.4GHz x4 + ARM® Cortex®-A53™ 2.0GHz x4 + ARM® Cortex®-A72™ 2.5GHz x2

- Procesador de imagen ARM Mali-T880
- 4 GB LPDDR3 memoria RAM
- Sistema operativo: Android v.6.0 marshmallow

Gama media: Sony xperia Z2 [27]

Especificaciones:

- Procesador Qualcomm MSM8974AB Snapdragon 801 quad-core 2.3GHz Adreno 330
- 3 GB de RAM
- Sistema operativo: Android v6.0 marshmallow

Pruebas Benchmark

Una vez generados los ejecutables de las aplicaciones (fichero APK, para instalar en Android), podemos observar el tamaño de los ejecutables.

LIBGDX: El APK generado tiene un tamaño de 2.3MB

AndEngine: El APK generado tiene un tamaño de 0.98MB

Unity3D/2D: El APK generado tiene un tamaño de 18.5MB

Podemos ver que la herramienta que genera un APK menos pesado es AndEngine. La razón es que solo genera un proyecto para desarrollar en Android.

Seguidamente encontramos el APK generado por LIBGDX. Este tiene un peso de 2.3MB, un poco más pesado que el de AndEngine, esto es debido a que LIBGDX genera tantos proyectos como plataformas seleccionadas para el desarrollo, ya que LIBGDX es multiplataforma. Al generar estos proyectos crea código auxiliar para poder trabajar con los diferentes lenguajes, por tanto, es más pesado por la capa de programación que tiene para trabajar con varios lenguajes.

Por último, tenemos el APK más pesado, el de Unity 3D/2D, tenemos que tener en cuenta que LIBGDX y AndEngine son *frameworks* y Unity es un motor gráfico el cual tiene muchas más librerías y funcionalidades integradas, esto hace que el APK generado sea más pesado.

El tamaño de nuestros ejecutables es importante, ya que a la hora de subir nuestras aplicaciones a los "stores" correspondientes, en el momento de la descarga consumirán menos datos móviles, y en nuestro dispositivo una vez instalados ocuparan menos espacio y eso lo agradecen los usuarios.

Juego de pruebas

Para realizar los *benchmarks* se ha creado un log que muestra los FPS cuando en pantalla hay 50, 100, 150, 200 y 250 objetos. Para ello se han conectado los dispositivos al entorno de desarrollo y se ha debugado consiguiendo mostrar por pantalla los resultados del log.

El escenario que nos encontramos es un fondo de color azul y bolas que van cayendo desde la parte superior de la pantalla hacia abajo con la gravedad aplicada. En la parte superior de la derecha podemos ver un indicador que nos muestra los Frames por segundo (FPS) y los objetos en pantalla en cada momento, esto nos servirá para determinar el resultado.

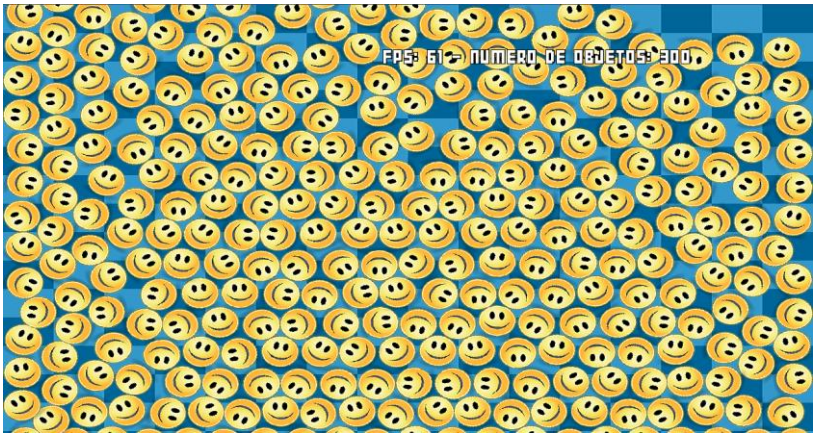


Figura 8.11. Escenario de pruebas.

A continuación, se muestran unos gráficos con el rendimiento detectado con cada teléfono móvil, indicado anteriormente en cada herramienta.

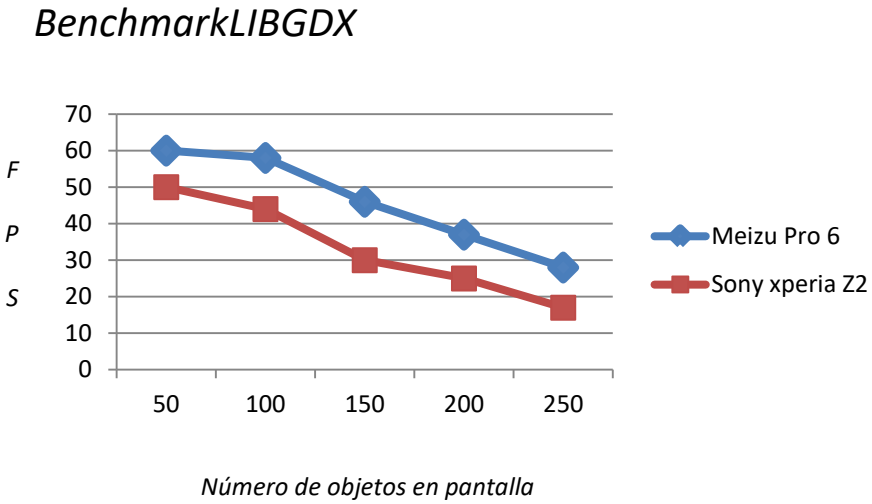


Figura 8.12 Resultado benchmark LIBGDX.

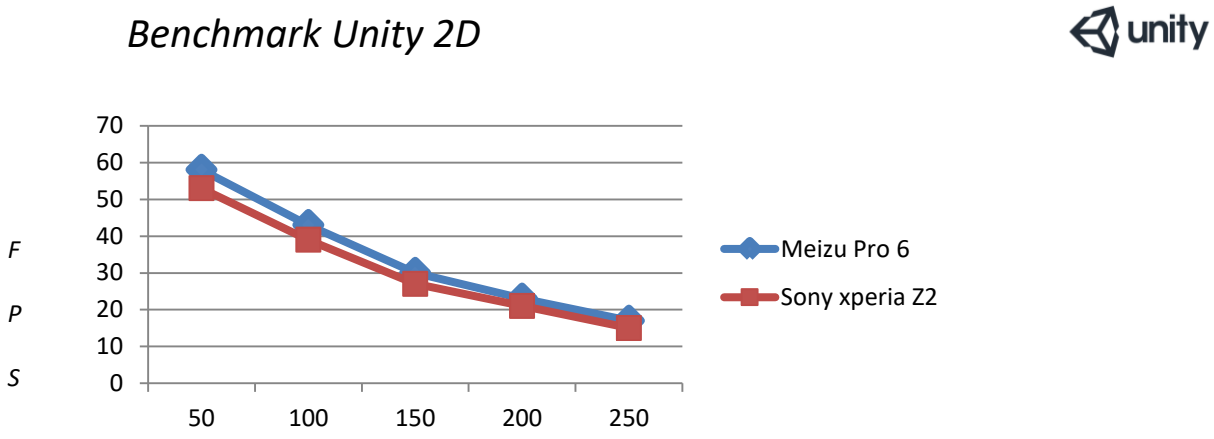


Figura 8.14 Resultado benchmark Unity.

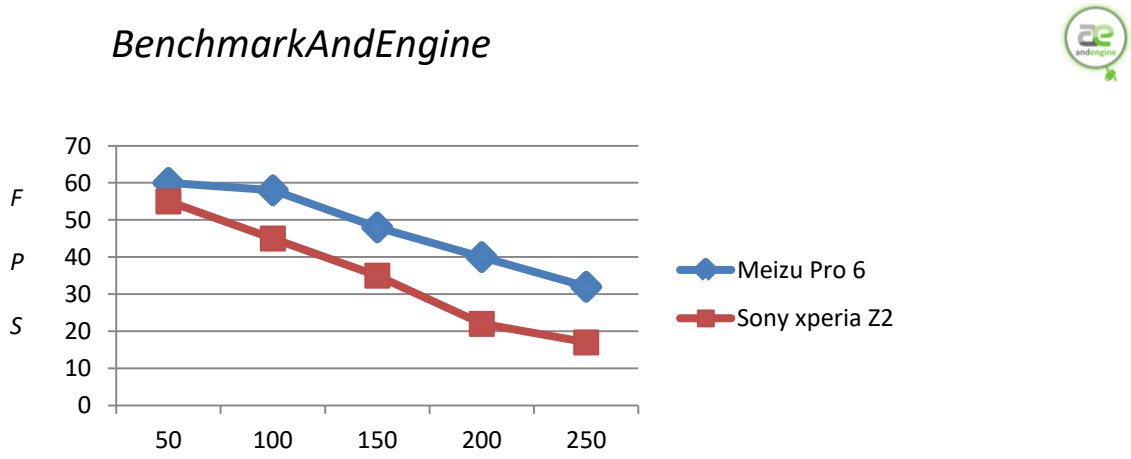


Figura 8.13 Resultado benchmark AndEngine.

A continuación, se muestra un enlace para poder ver el video del benchmark.

- Benchmark LIBGDX vs AndEngine vs Unity.

Los resultados demuestran que los *frameworks* LIBGDX y AndEngine consiguen un mejor rendimiento que Unity 2D. En los gráficos se puede ver que el motor gráfico Unity 2D es más estable en diferentes dispositivos móviles, ya que los resultados del *benchmark* han salido muy parecidos entre un móvil de gama alta y uno de gama media.

En los *frameworks* AndEngine y LIBGDX el resultado no muestra mucha diferencia. Los dos tratan de forma muy parecida la prueba. AndEngine tiene una tasa de frames menos estable que LIBGDX, en ocasiones consiguiendo picos altos de FPS y en otras ocasiones picos bajos que podrían dañar la experiencia de juego del usuario.

Conclusión del benchmark

Como conclusión, mientras los FPS se mantengan por encima de los 24 FPS, tendremos un movimiento fluido, si bajamos de este número tendremos una mala experiencia de juego, que puede llegar a frustrar al jugador.

Este *benchmark* no lo he realizado con el videojuego *Airhockey* porque hubiera sido mucho más costoso, ya que se tendría que haber desarrollado para Unity 3d y AndEngine a parte de LIBGDX. El escenario desarrollado para este *benchmark* busca colocar muchos objetos colisionando en pantalla, para hacer trabajar a los *frameworks*, esto no podríamos conseguirlo con el videojuego de *AirHockey*, ya que el número de objetos en pantalla es fijo, se mantendrían los FPS estables y no se incrementaría el trabajo de procesamiento.

9. Conclusiones y trabajo futuro

Una vez finalizado el proyecto, analizando el trabajo realizado y considerando los objetivos iniciales, podemos extraer las siguientes conclusiones:

Conceptos generales:

- Son muchas las herramientas y tecnologías que hay que aprender para el desarrollo de videojuegos.
- Es necesario tener un equipo de personas especialidades en diferentes campos para desarrollar videojuegos, ya que son muchas las tareas a realizar y muy variadas (Diseñador gráfico, Desarrollador de IA, lógica, creativo).
- Las tareas de diseño y creación de imágenes para la interfaz o para los modelos del escenario, requieren muchos recursos y tiempo de dedicación.
- LIBGDX es un framework muy potente, y que tiene un gran soporte de la comunidad, pero en ocasiones es difícil encontrar documentación para poder trabajar con las APIs de los diferentes dispositivos.

Implementación y código:

- Se ha intentado crear una buena estructura de código. Por ejemplo, creado una clase global para los menús llamada *AbstractScreen* de aquí extienden todas las clases que crean menús. La clase *AbstractScreen* tiene definida la estructura donde se colocan los botones u otros elementos de los menús.
- La implementación del código se ha realizado de forma modular, separando en diferentes clases todo aquello que podría ser modificable en posibles ampliaciones o cambios posteriores.
- Durante el desarrollo del proyecto, han ido surgiendo varias dificultades y problemas, tanto con las tecnologías conocidas como con las nuevas. Por ejemplo, adaptar los textos a los diferentes tipos de pantallas. Con el paso del tiempo y estudiando las herramientas más a fondo se han ido solucionando todos los problemas.

Análisis de objetivos generales:

- El videojuego del *AirHockey* desarrollado en este proyecto, se ha probado en varios dispositivos para valorar su rendimiento. Los resultados han sido muy positivos, consiguiendo una optimización muy buena tanto en dispositivos de gama alta como en los de gama baja.
- Se ha realizado una comparación de LIBGDX con Unity 3D y AndEngine, donde se ponen a prueba y se comprueba el rendimiento. Como conclusión, viendo los resultados de rendimiento las tres herramientas analizadas están muy igualadas. Como LIBGDX es multiplataforma se recomendaría utilizar este framework antes que AndEngine para realizar menos faenas a la hora de exportar el proyecto a otros sistemas.

Análisis de objetivos personales:

- Una vez finalizado el desarrollo, puedo decir que he adquirido muchos conocimientos tanto teóricos como prácticos, sobre el desarrollo y diseño de videojuegos.
- Varias asignaturas de la carrera me han ayudado a completar tareas del proyecto, como puede ser creación de diagramas, o entender cómo funciona el framework LIBGDX internamente.
- Una vez finalizado el proyecto, se puede decir que no ha quedado ninguna tarea sin realizar o acabar, y que el objetivo inicial se ha conseguido. Aun así, conforme se iba desarrollando han ido surgiendo nuevas ideas que se podrían incluir en un trabajo futuro.

A continuación, se nombran estas ideas para el trabajo futuro.

Modo Campaña: En el videojuego no se han creado niveles simplemente se ha creado un modo mono jugador en el cual jugamos contra la IA. Se podría mejorar el videojuego añadiendo un modo campaña, donde haya varios niveles y podamos jugar contra diferentes tipos de IA, donde en cada nivel se vaya incrementado el nivel de dificultad y vaya cambiando el escenario.

Diseño: Para este proyecto se han utilizado muchas texturas. Con más tiempo se podría mejorar el diseño creando nuevas texturas para los escenarios, o incluso mejorar la GUI con nuevas opciones.

Plataformas: El videojuego *AirHockey* se ha desarrollado solo para PC y Android, pero como LIBGDX nos permite importar también para varias plataformas, se podría implementar el acceso a la API del bluetooth de estas plataformas y generar el videojuego para ellas.

Inteligencia artificial (IA): El videojuego incluye una IA muy básica, la cual no integra ningún algoritmo complejo. En un futuro se podría implementar una IA que decidirá la estrategia como sería más eficiente para que en el golpeo del disco sea más probable marcar gol.

Mecánica de juego: Para hacer el juego más dinámico y más divertido se podría implementar una funcionalidad para que cuando un jugador marque X goles seguidos el mazo aumente de tamaño y así conseguir ventaja a la hora de defender la portería.

10. Referencias

Programas y herramientas:

[1] **LIBGDX:** Enlace de descarga para el framework LIBGDX.
[2] **AndEngine:** Enlace de descarga para el framework AndEngine.
[3] **Unity:** Enlace de descarga para el motor gráfico Unity.
[4] **Modelio:** Enlace para descargarModelio.
[5] **ObjectAIDUML:** Enlace para descargar la herramienta ObjectAIDUML.
[6] **Hiero v5**: Enlace para descargar la herramienta Hiero v5.
[7] **Eclipse:** Enlace para descargar el entorno de desarrollo Eclipse

Libros:

[8] **Read, Paul; Meyer, Mark-Paul; Gamma Group (2000). Restoration of motion picture film. Conservation and Museology. Butterworth-Heinemann. pp. 24-26.**

Articulos:

[9] **Takahashi, Dean. Worldwidegameindustry hits $91 billion in revenues in 2016 (2016).**
[33] **Gutierrez, Demián. Casos de Uso, diagramas de Casos de Uso (2011).**
[11] **Francisco Martín Archundia Papacetzi. El estándar Bluetooth (2003).**
[32] **García Daniel. El esquema cliente-servidor (2013).**

Sitios Web: (Consultado en junio de 2017)

[10] **Versiones LIBGDX:** Descripción de los cambios que se han ido introduciendo en todas las versiones de LIBGDX.
[12] **PowerManager:** Documentación sobre la API para gestionar la alimentación del dispositivo Android.
[13] **ViewPorts:** Documentación sobre viewPorts en LIBGDX.
[14] **Box2D:** Documentación y descarga.
[15] **JAVA:** Programming Language Popularity.
[16] **Orientación pantalla Android:** En esta página se muestran los tipos de orientación de pantalla en Android.
[17] **AndroidManifest:** En este enlace se muestra el funcionamiento del fichero de configuración AndroidManifest.
[18] **Licencia publica de eclipse:** Enlace a la página donde aparece la licencia de eclipse.
[19] **Licencia modelio:** Enlace a la página donde aparece la licencia de modelio.
[20] **Bluetooth socket:** En este enlace se muestra información del funcionamiento de los sockets de Android.
[22] **UUID**: En este enlace se ha visto y aprendido lo que es un UUID y cómo funciona.
[23] **Generacion de UUID**: Esta página sirve para crear UUID online y gratuitamente.
[24] **Bluetooth adapter:** En esta página se muestra información sobre la API del bluetooth de Android.

[25] **Potencia grafica adreno 200:** En esta página se ha encontrado la potencia de la gráfica Adreno 200.
[26] **Meizu Pro 6:** Especificaciones del dispositivo móvil Meizu Pro 6.
[27] **Sony Xperia Z2:** Especificaciones del dispositivo móvil Sony Xperia Z2.
[30] **Historia de JAVA**: TheHistory of Java Technology.
[31] **Licencia Unity 3D:** Licencias de Unity Personal.

Otros sitios web consultados:

LIBGDX. Documentación y ayuda.
Youtube: Vídeos para el aprendizaje de LIBGDX.

11. Apéndice manual de desarrollador

A continuación, se muestra un manual para la descarga y configuración de LIBGDX.

Podemos descargar el *framework* desde la página oficial, para ello ir a la página [1].

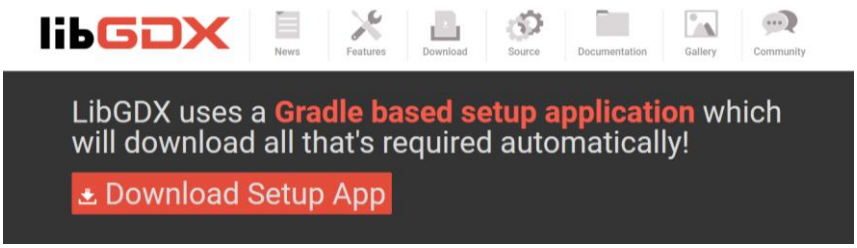


Figura 11.1 Página web LIBGDX.

Se descargará un .jar. Este jar es para la configuración, preparación y generación del proyecto donde vamos a trabajar.

Al ejecutar el jar aparecerá la siguiente pantalla.

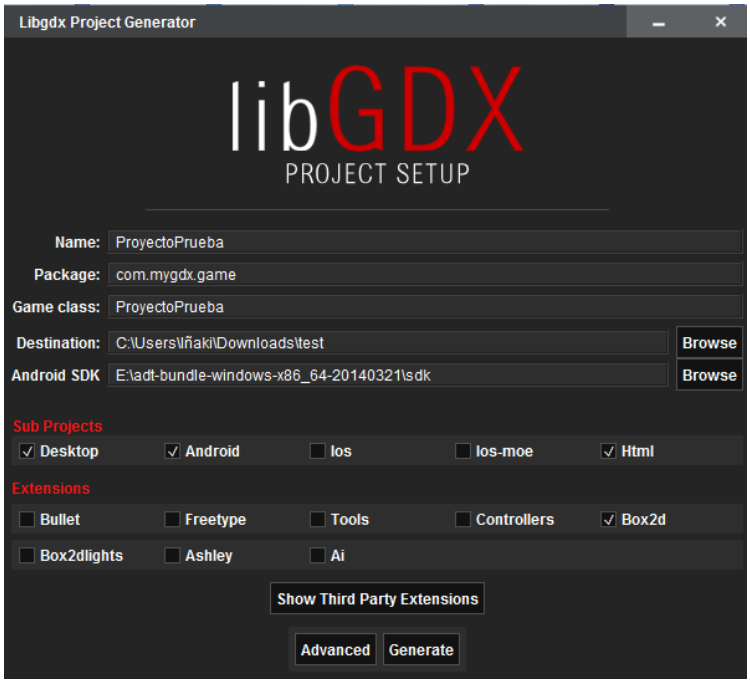


Figura 11.2 Project Setup.

En esta pantalla debemos indicar el Nombre de nuestro proyecto, nombre del paquete del proyecto y nombre de la clase principal,

En el apartado Android SDK debemos indicar el lugar donde tenemos instalado el SDK de Android.

Seguidamente en el apartado “Sub Proyects” debemos seleccionar los sistemas para los que queremos generar el proyecto, estos más tarde serán los sistemas a los que podremos exportar nuestro proyecto.

Por último, tenemos el apartado “Extensions”, esto son librerías externas que se han integrado en LIBGDX y aquí podemos seleccionarlas para incluirlas en nuestro proyecto. Las más importantes son las siguientes:

- Box2d: Esta librería nos proporcionará funcionalidades para poder implementar físicas en nuestro proyecto.
- Bos2dLights: Esta librería nos proporciona funcionalidades para poder añadir luces a nuestra escena.

Una vez tengamos configurado nuestro proyecto hacemos clic en el botón “Generate” y comenzará a generarse el proyecto, esta operación puede tardar un par de minutos.

Cuando haya finalizado podremos ver que se nos han generado varios proyectos y que son de tipo GRADLE, esto es porque se generan con una herramienta de automatización de construcción de código que ayuda en los builds de código en diferentes IDEs.

En nuestro caso los proyectos generados son los siguientes:

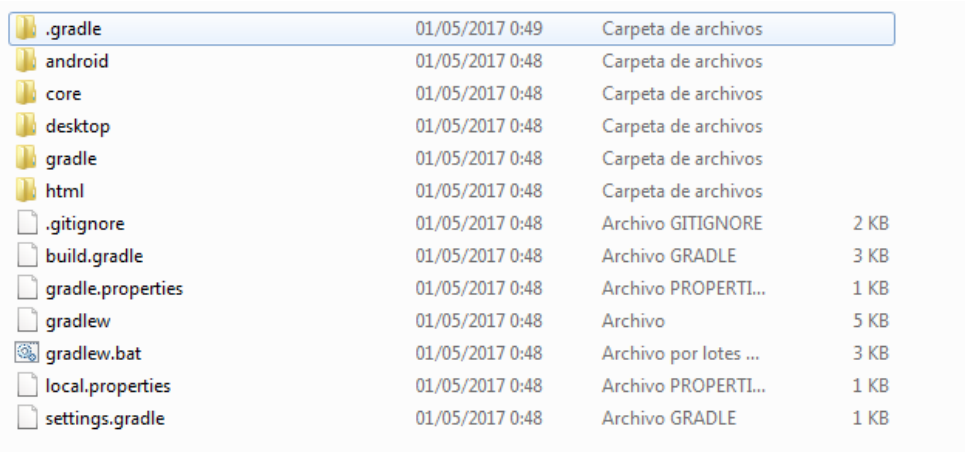


Figura 11.3 Proyecto generados.

- Android: En este proyecto se alojará la parte para android
- Desktop: En este proyecto se alojará la parte para PC.
- Html: En este proyecto se alojará la parte para Web.
- Core: Este es el proyecto principal, aquí será donde desarrollaremos todo nuestro proyecto y desde los otros solo llamaremos a este o implementaremos funcionalidades propias de cada sistema.
- Gradle: este contiene todas las librerías relacionadas con gradle, este proyecto no debemos tocarlo.

Como vemos sólo se nos han generado los proyectos seleccionados desde el JAR, si hubiéramos seleccionado más proyectos, tendríamos más.

Para comenzar a trabajar debemos importar los proyectos a algún entorno de desarrollo, los IDEs más utilizados son Eclipse y Intellij IDEA, pero en caso de desarrollar para Android recomiendo utilizar Ecplise, ya que hay una versión preparada para ello y de esta manera no tendremos problemas de librerías.

En este informe se utilizará eclipse.

Para importar los proyectos abrimos eclipse, vamos a file y seleccionamos importar.

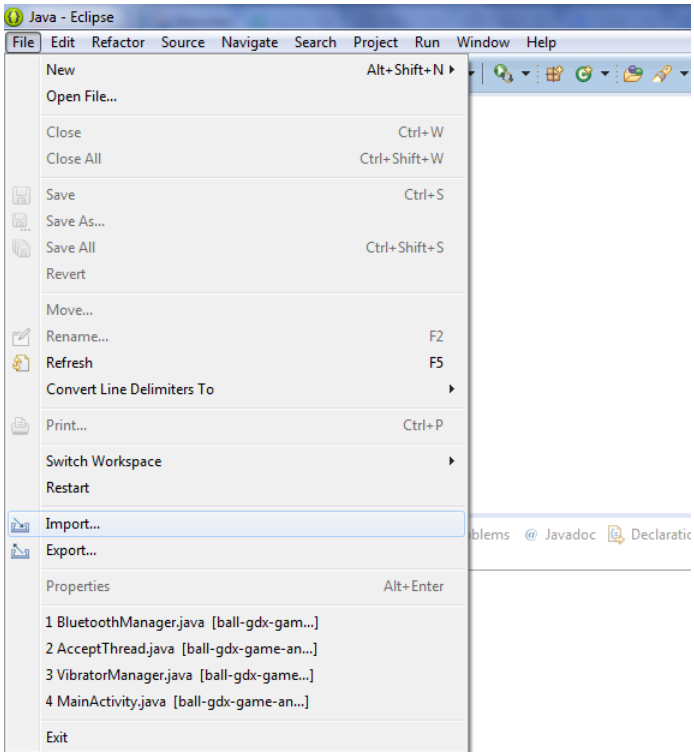


Figura 11.4 Importando proyectos.

Seguidamente seleccionamos Gradle/GradleProyect, si no tenemos instalado Gradle en nuestro Ecplise deberemos descargar la extensión, en caso contrario no nos aparecerá esta opción.

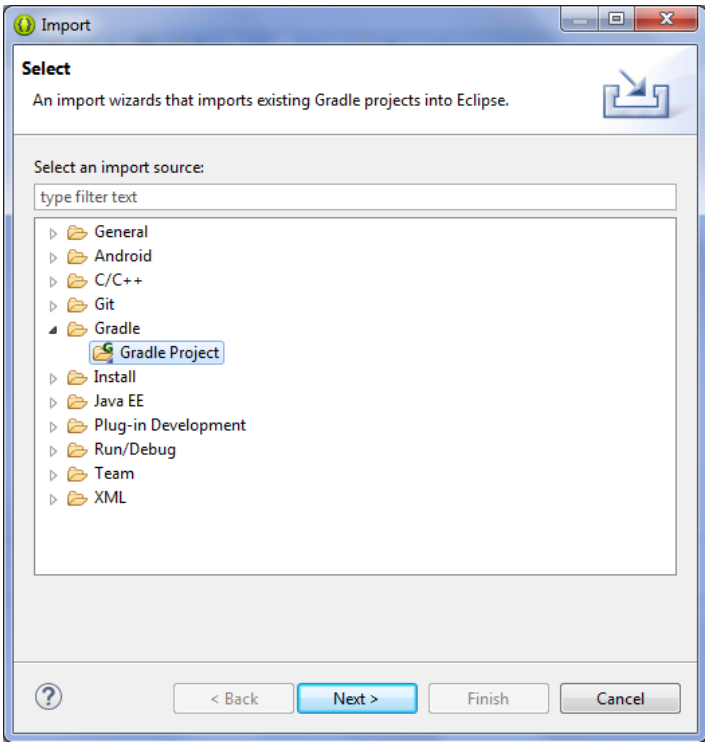


Figura 11.5 Importar proyecto gradle.

Seleccionamos la carpeta que contiene los proyectos. Atención: El proyecto que aparece en la imagen es un proyecto de prueba que se ha creado para mostrar este manual, no es el juego de *AirHockey*.

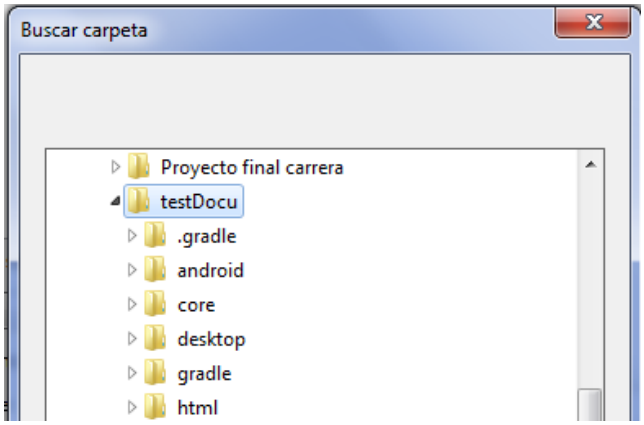


Figura 11.6 Selección de proyecto a importar.

Una vez seleccionado seleccionamos el botón de BuildModel, que se encargara de descompilar el código de gradle.

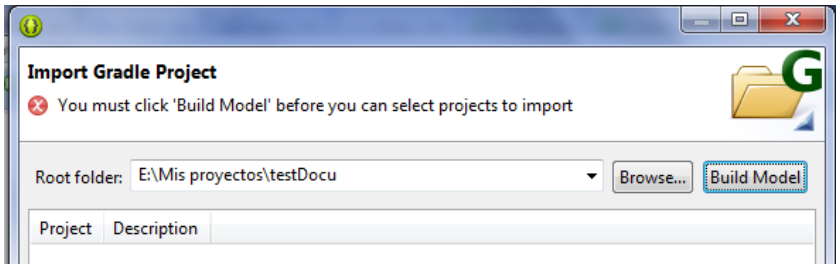


Figura 11.7 Construyendo el modelo para gradle.

Al terminar de descompilar seleccionamos todos los proyectos y clicamos en “finish”.

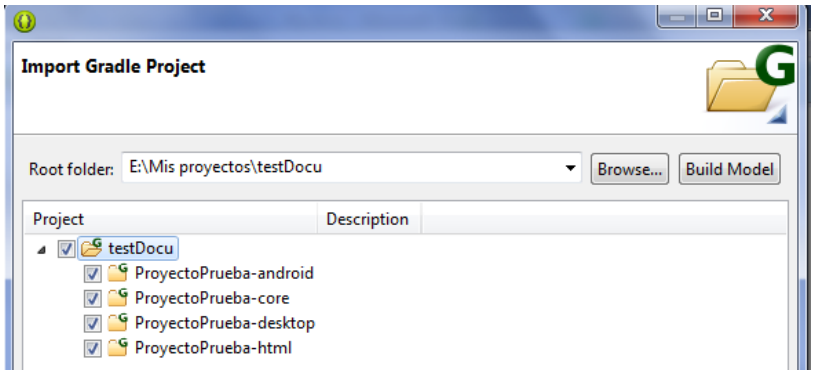


Figura 11.8 Proyecto preparado para importar.

En este punto el IDE comenzará a realizar la importación de los proyectos.

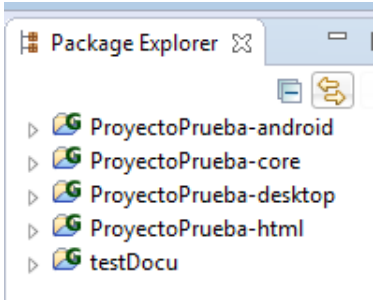


Figura 11.9 Proyecto importado a eclipse.

Una vez importado si queremos ejecutar el código para probarlos, seleccionamos el proyecto que queramos ejecutar, en mi caso seleccionaré el desktop, clicamos con el botón derecho y seleccionamos “Run As/Java aplication”.

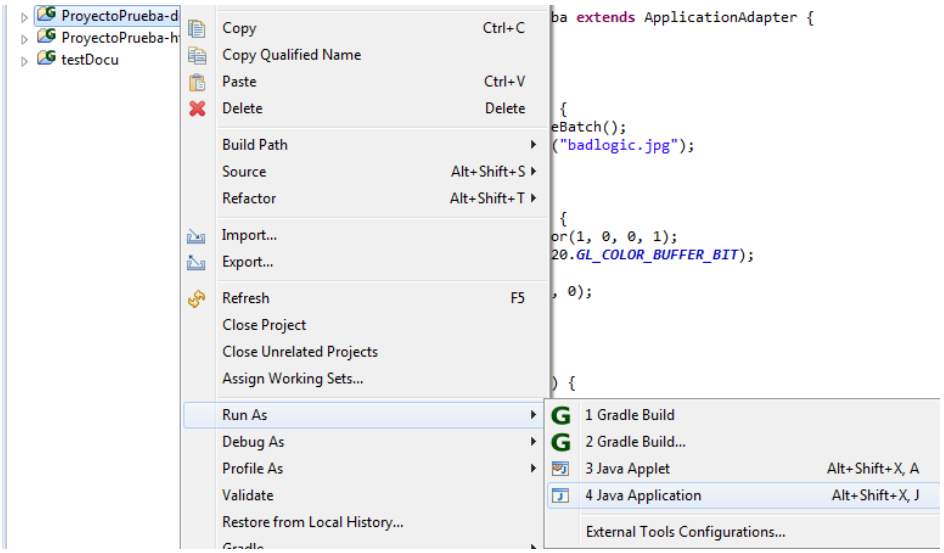


Figura 11.10 Ejecutar aplicación

En la pantalla que se abrirá seleccionamos la opción “Java application”.

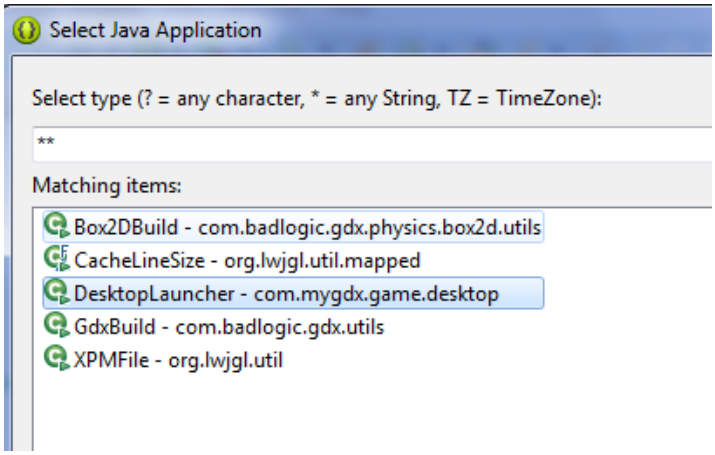


Figura 11.11 Seleccionar tipo de ejecución.

Se ejecutará el proyecto para PC y mostrará un fondo y una imagen.

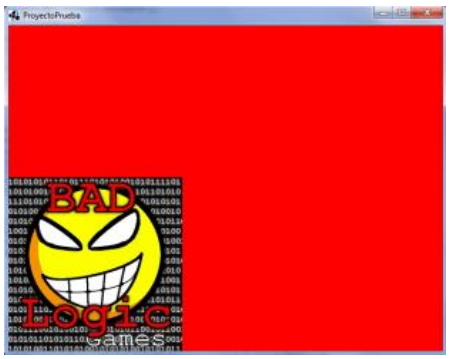


Figura 11.12 Ejecución de la aplicación.

Para desarrollar videojuegos no es recomendable utilizar el emulador de Android que ofrecen los diferentes IDEs, ya que no ofrecen un buen rendimiento. Se recomienda utilizar directamente un dispositivo Android.

Para ejecutar el código en un dispositivo Android primero se deben instalar los drivers del dispositivo que se vaya a utilizar. Una vez instalados Eclipse detectará el dispositivo. Para ejecutar el código como una aplicación Android en un dispositivo hay que hacer “clic” con el botón derecho en el proyecto de Android y seleccionar la opción *Run As* y seguidamente en *Android Application*.