



YERKO ÁLVAREZ PINTO

BOMBERMAN JAVA EDITION

Índice

1. Resumen.....	3
1.1. Sumary	3
2. Introducción	5
2.1. Videojuego.....	5
2.2. LibGDX	7
2.3. ECS	7
3. Objetivos y características del proyecto.....	9
4. Finalidad.....	11
5. Medios materiales usados.....	11
5.1. Software	11
5.1.1. IDE: Visual Studio Code	11
5.1.2. Gradle.....	13
5.1.3. Tiled.....	14
5.1.4. GDX texture packer	16
5.1.5. Photoshop.....	17
5.1.6. Github	19
5.1.7. Mysql / phpMyAdmin	20
5.1.8. LibGDX	21
5.1.9. Ashley	23
5.1.10. Java	25
5.2. Hardware.....	27

5.3.	Humanos	28
6.	Planificación del proyecto	29
6.2.	Ajustes del entorno	31
6.3.	Ajustes iniciales del proyecto	32
6.4.	Creación de Screens	38
6.5.	ECS	42
6.6.	Managers.....	44
6.7.	Data base	46
7.	Fases de prueba	47
8.	Conclusiones y trabajos futuros	49
8.1.	Organización del código	49
8.2.	Tests	49
8.3.	Corrección de errores conocidos.....	49
8.4.	Mayor uso de Git y Gradle	50
9.	Referencias Bibliográficas	50
9.1.	Libros	50
9.2.	Artículos y páginas	50
9.3.	Videos	51

1. Resumen

Se pretende realizar una Alpha del juego *Super Bombermam*, juego desarrollado por *Hudson Soft*, para la consola de videojuegos *SNES (Super Nintendo)*, en el año de 1993.

Hecho completamente en *Java*, gracias a la librería de código abierto *LibGDX*, así como otros frameworks como lo son: *BOX2D*, *BOX2DLigth*, *Ashley* y *FreeType*. Que vienen incluidos en la librería anteriormente mencionada, adaptados para su correcto funcionamiento. Para esta versión de este juego mundialmente reconocido, se pretende hacer uso de la arquitectura *ECS (Entity Component System)*, gracias al framework de *Ashley*, el cual pretende cambiar la forma de trabajar en los videojuegos, pasando de una arquitectura de Objeto (POO) a una orientada a entidades, componentes y sistemas (ECS), para dar a cualquier juego mayor adaptabilidad, y hacerlo más mantenible.

Por lo cual el proyecto se basa en crear una copia del juego de *Super Bombermam* usando, la arquitectura de *ECS* en *Java*.

1.1. Summary

We pretend to make an Alpha of the game *Super Bombermam*, a game developed by *Hudson Soft*, for the *SNES (Super Nintendo)* video game console, in 1993.

Made in *Java*, with the *LibGDX* open source library, as well as other frameworks such as: *BOX2D*, *BOX2DLigth*, *Ashley* and *FreeType*. That are included in the that library, adapted for its correct operation. For this version of this world-renowned game, it is intended to make use of the *ECS (Entity Component System)* architecture, thanks to the *Ashley* framework, which aims to change the way of working in video games, moving from an Object architecture (POO) to one oriented to entities, components and systems (ECS), to give any game greater adaptability, and make it more maintainable.

Therefore, the project is based on creating a copy of the *Super Bombermam* game using the *ECS* architecture in *Java*.

2. Introducción

Para la realización de este proyecto se ha planificado primeramente una investigación del de las distintas librerías y frameworks a utilizar, para poder conseguir un mayor grado de acercamiento al título original del que se quiere inspirar este proyecto.

Por ende, en los siguientes apartados se realizará unas breves explicaciones de algunos conceptos necesarios para entender los principios de la base de nuestro videojuego.

2.1. Videojuego

Con relación al videojuego: *Super Bomberman* es el primer videojuego en la serie de *Bomberman* en aparecer en la consola *Super Nintendo*. También es el primer juego con opción de jugar 4 jugadores simultáneamente.



Este videojuego cuenta con 6 Stages con 7 niveles cada uno en los cuales se posee un TileMap para cada uno de los Stages

BOMBEMAN JAVA EDITION



Y un Boss final por cada uno de los Stages, así como enemigos individuales



Como se puede ver en las imágenes, nos encontramos con un juego muy complejo.

Luego la dinámica del mismo se basa en la colocación de bombas para abrirte paso y eliminar a los enemigos, consiguiendo puntos y avanzando entre los distintos niveles hasta acabar el juego.

Se eligió este juego debido a su poca complejidad a nivel lógico, y al ser antiguo, se pudo conseguir varios de los elementos que forman al juego, los Sprites, música, etc.

2.2. *LibGDX*

Como se ha mencionado anteriormente LibGDX es una librería de Java de código abierto basada en el desarrollo de los videojuegos. La cual posee distintos frameworks que se explicaran mas adelante.

Se opto por esta opción debido al lenguaje con el cual esta creado, ya que *Java* es el lenguaje de programación que se ha visto estos dos años con mas detenimiento y es por eso que se tiene una mayor afinidad.

Además se quería usar la mayor parte de los conceptos vistos en clase los cuales son : El modelo MVC, los hilos, POO, desarrollo con base de datos, etc.

2.3. *ECS*

Como se ha mencionado anteriormente, en este proyecto se hace uso de una arquitectura pensada para el desarrollo de videojuegos, esta es ECS, la cual se explicará posteriormente.

Se quiso aprender mas de esta arquitectura en particular porque, aunque se podía hacer con POO perfectamente, al investigar sobre la resolución del proyecto, se vio la posibilidad de añadir ECS gracias a su facilidad de manejo con el framework de *Ashley* y la buena implementación de esta con LibGDX. Aunque en futuras actualizaciones, este framework paso a ser descartado por otro framework, con más potencia pero más complejo en su ejecución.

Es por ello mismo que se eligió una versión de LibGDX que poseyera este framework para este proyecto. Pero no se descarta la idea de hacer la implementación del nuevo framework de LibGDX dado su gran potencial demostrado en distintos proyectos estudiados para la resolución de este.

3. Objetivos y características del proyecto

Como se ha mencionado en un inicio, este proyecto no deja de ser nada mas que una Alpha que intenta emular al juego original. Y dado el tiempo planteado para la resolución del mismo, y aun con todo el esfuerzo implantado, se hace muy difícil conseguir que se asemeje al juego.

Aun así se aspira a tener un proyecto bien encaminado, el cual posea los aspectos básicos del juego original asi como su esencia.

Por otra parte, se inició este proyecto para poder ver a ciencia cierta que era enfrentarse a la resolución de un juego desde cero y es que aun quitando una de las partes más desafiantes como lo es la planificación de la idea y el diseño de los niveles y los distintos elementos decorativos e interactivos de un juego. No deja atrás la dificultad luego de al darle vida al mismo gracias a la programación. Es con esa idea que se inició este proyecto buscando aprender las distintas implementaciones, así como una forma de trabajar, dado que cada motor de juego, así como cada lenguaje posee sus distintos distintivos que lo hacen diferentes a los demás. En esencia, podemos decir que al aprender cómo funciona uno de ello podemos transpolar lo aprendido a otros. Es por ello que se intentó hacer el trabajo en el marco de *Java*.

Por otra parte, se intentó aprender más sobre el lenguaje de programación que se uso en el proyecto debido a su robustez y a sus millones de implementaciones. *Java* se ha convertido en un estándar en más de una empresa a nivel mundial, y es por ello que se quería ver algunos de los aspectos de ella orientado a los videojuegos, aun sabiendo que no es la mejor opción en el mercado a día de hoy. Pero si teniendo en cuenta que se ha usado para crear uno de los juegos mundialmente reconocidos como lo es *Minecraft*. Un juego de mundo abierto creado inicialmente en Java y que a día de hoy se ha convertido en un juego de culto. Y aun, es más, sabiendo que las primeras versiones de ese gran juego se han creado en el seno de las primeras versiones de LibGDX. Aunque posteriormente se cambiaría de motor por uno desarrollado por los creadores del juego.

También cabe destacar que se ha buscado saber más acerca de la arquitectura de ECS, el cual tiene una cierta peculiaridad que dota a los juegos con gran disponibilidad al cambio y que en sus inicios se lo llamo un estándar para los juegos de RPG o MMORPG, así como los JRPG. Debido a que estos juegos poseen una gran extensión a nivel de personajes, niveles, jerarquías, mapas, armas, etc. Que en un modelo de POO se hace muy difícil de mantener llegados a cierto punto. Es por ello que se crea ECS el cual intenta generalizar los distintos elementos de un videojuego para poder lograr una mayor adaptabilidad y optimización a la hora de ejecutar el código.

Por tanto, podemos decir que el proyecto nos ha ayudado a ver el desarrollo de videojuegos desde un punto de vista diferente y se ha intentado aprender tanto del lenguaje de programación utilizado como de las distintas funcionalidades que puede dar las librerías open source de java.

4. Finalidad

Como se ha mencionado anteriormente, se quiere realizar un Alpha del clásico juego de Super Bomberman creado inicialmente para la SNES.

Como tal no se espera hacer el juego completo ni llegar a implementar todas las funcionalidades que posee el juego original, dado el tiempo así como los recursos utilizados que no son los mas apropiados para el desarrollo de videojuego.

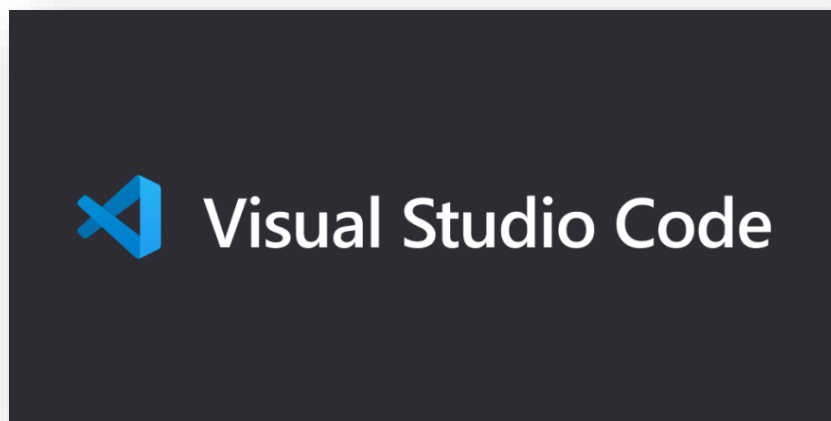
Por ende, se pretende crear un juego funcional en el cual se pueda mover al personaje y destruir algunos elementos del mapa, además de poder eliminar a los enemigos del mapa para encontrar un portal que lleve al nivel siguiente y finalizar con una pantalla de puntuación que se guarde en una base de datos y que se pueda visualizar directamente desde el juego teniendo así un recuento de las puntuaciones de las distintas personas que jueguen a nuestro videojuego, haciendo todas las validaciones pertinentes y buscando principalmente el entretenimiento y la diversión de del jugador.

5. Medios materiales usados

En este apartado se hará un recuento junto con una breve explicación de los distintos elementos para la creación del nuestro videojuego.

5.1. Software

5.1.1. IDE: Visual Studio Code



Uno de los principales elementos a la hora de desarrollar software es un IDE que se ajuste a las necesidades del proyecto, así como uno que sea cómodo para los trabajadores, ya sea por su menor curva de aprendizaje o por su mayor comodidad visual que el usuario en cuestión pueda encontrar en este. Podemos afirmar que un buen IDE ayuda mucho a la ejecución del trabajo y es por eso que nos hemos decantado por *VS Code*.

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

Visual Studio Code se basa en Electron, un framework que se utiliza para implementar Chromium y Node.js como aplicaciones para escritorio, que se ejecuta en el motor de diseño Blink. Aunque utiliza el framework Electron, el software no usa Atom y en su lugar emplea el mismo componente editor (Monaco) utilizado en Visual Studio Team Services (anteriormente llamado Visual Studio Online).

Por estas razones y gracias a su plugin de Java, es que se ha podido realizar la totalidad del trabajo en este IDE, configurando el proyecto desde su inicio con Gradle y haciendo un seguimiento con Git, tecnologías que veremos a continuación

5.1.2. *Gradle*



Gradle es un sistema de automatización de construcción de código de software que construye sobre los conceptos de Apache Ant y Apache Maven e introduce un lenguaje específico del dominio (DSL) basado en Groovy en vez de la forma XML utilizada por Apache Maven para declarar la configuración de proyecto. Gradle utiliza un grafo acíclico dirigido ("DAG") para determinar el orden en el que las tareas pueden ser ejecutadas.

Gradle fue diseñado para construcciones multi-proyecto las cuales pueden crecer para ser bastante grandes, y da apoyo a construcciones incrementales determinando inteligentemente qué partes del árbol de construcción están actualizadas, de modo que cualquier tarea dependiente a aquellas partes no necesitarán ser reejecutada.

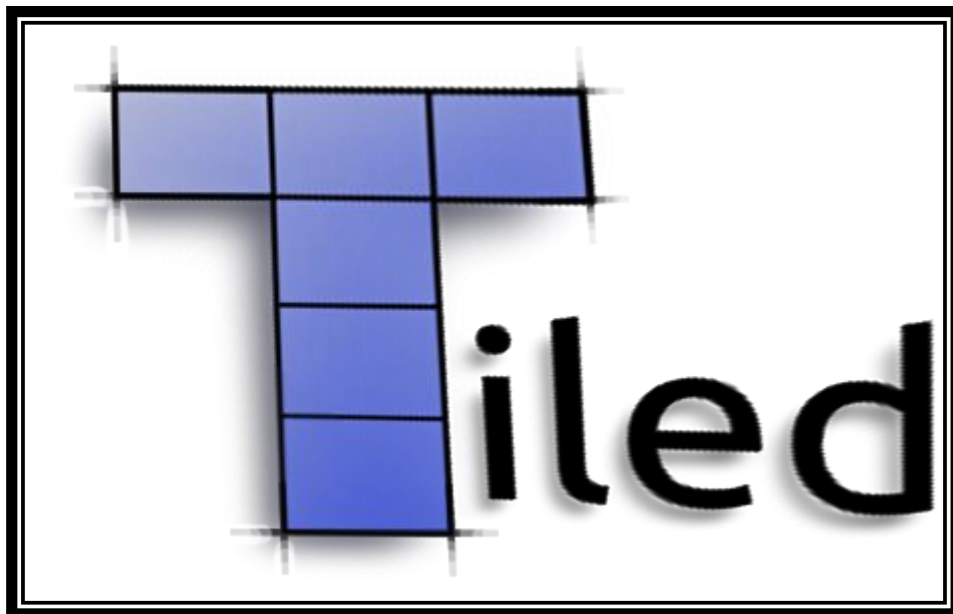
Los plugins iniciales están principalmente centrados en el desarrollo y despliegue en Java, Groovy y Scala, pero existen más lenguajes y workflows de proyecto en el roadmap.

Esta tecnología ha sido fundamental para el proyecto, ya no tanto por su sistema de automatización que, por causas de tiempo, no se ha podido expresar lo que se hubiera querido. Sino por su gran capacidad de gestión de multiproyectos y es que LibGdx al ser una librería que otorga la posibilidad de, ya no solo hacer nuestro juego multiplataforma a nivel de sistemas operativos como Linux, MacOS o Windows. Sino que también permite portarlos para dispositivos móviles, web, e incluso IOS. Es por esta gran gama de dispositivos que se requiere separarlos para poder gestionar de manera más eficiente el

comportamiento que tendrá. Y es ahí en donde Gradle resulta muy útil para gestionar estos proyectos por separado.

Si es verdad que cabe recalcar que, en nuestro juego, solo hemos acogido la posibilidad de ejecutarlos en varios sistemas operativos y no tanto en dispositivos móviles o de web debido al tiempo y a la complejidad a la hora de configurar cada uno de estos dispositivos para que puedan funcionar de manera correcta.

5.1.3. Tiled



Tiled es un editor de niveles 2D que ayuda a desarrollar el contenido del juego. Su característica principal es editar mapas de mosaicos de varias formas, pero también admite la colocación de imágenes gratuitas, así como formas poderosas de anotar su nivel con información adicional utilizada por el juego. Tiled se enfoca en la flexibilidad general mientras intenta mantenerse intuitivo.

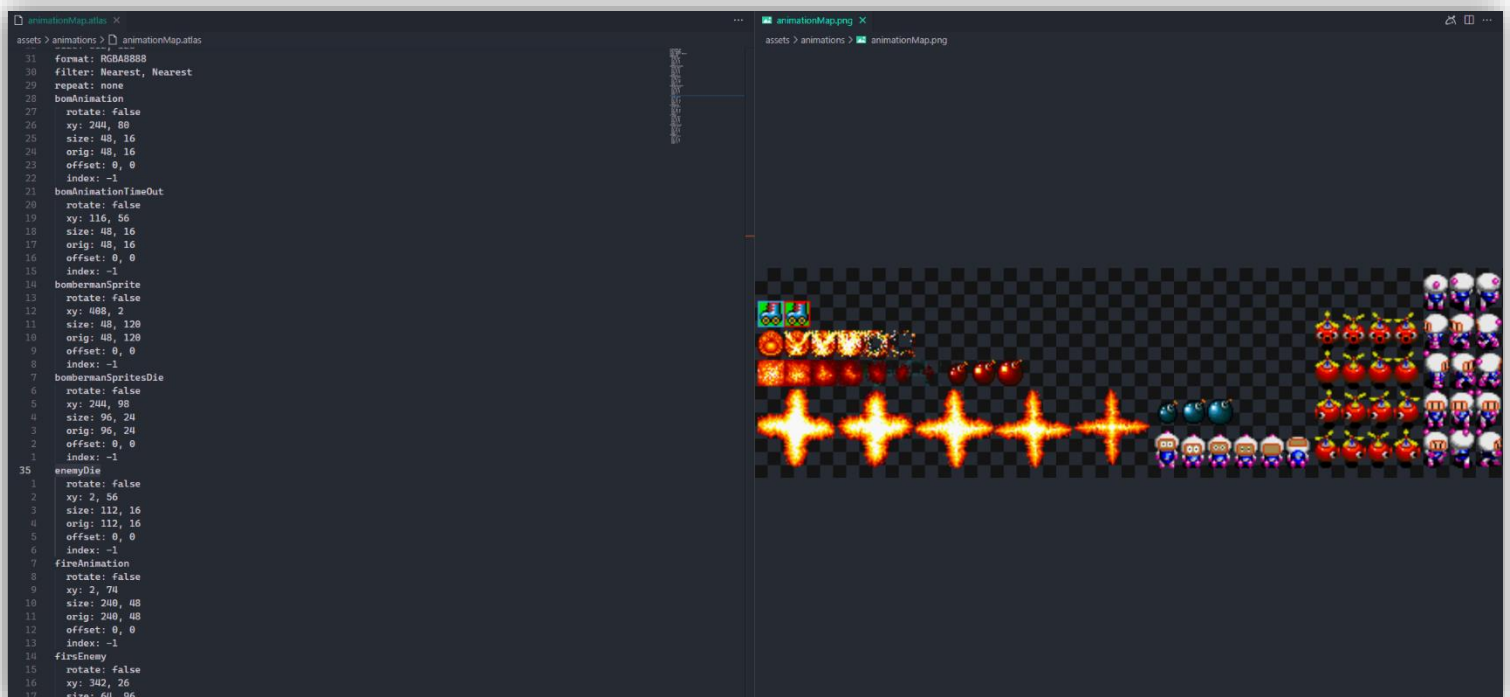
En términos de mapas de mosaicos, admite capas de mosaicos rectangulares rectos, pero también capas isométricas proyectadas, isométricas escalonadas y hexagonales

escalonadas. Un mosaico puede ser una sola imagen que contenga muchos mosaicos o puede ser una colección de imágenes individuales. Para admitir ciertas técnicas de falsificación de profundidad, los mosaicos y las capas se pueden compensar con una distancia personalizada y se puede configurar su orden de representación.

Es por esto que esta herramienta ha sido fundamental para el desarrollo de nuestro videojuego. Aunque si es verdad que no es un editor tan complejo como lo es el de Unity u otros, este editor de niveles no tiene nada que envidiarles y su forma de trabajar es muy intuitiva y fácil de aprender.

Su implementación, también cabe destacar, ha sido sencilla gracias a que posee gran compatibilidad con distintos lenguajes entre los cuales no solo se encuentra Java, sino que la librería de LibGDX posee varios métodos para leer los mapas generados por esta herramienta y es que la implementación llega a tal nivel que poseen la misma manera de incorporar SpriteMaps. El cual se basa en el uso de atlas.

Un atlas no es mas que un archivo de información que contiene todos los estados y posiciones de una imagen de un grupo de imágenes. Esto se puede ver en la siguiente imagen:



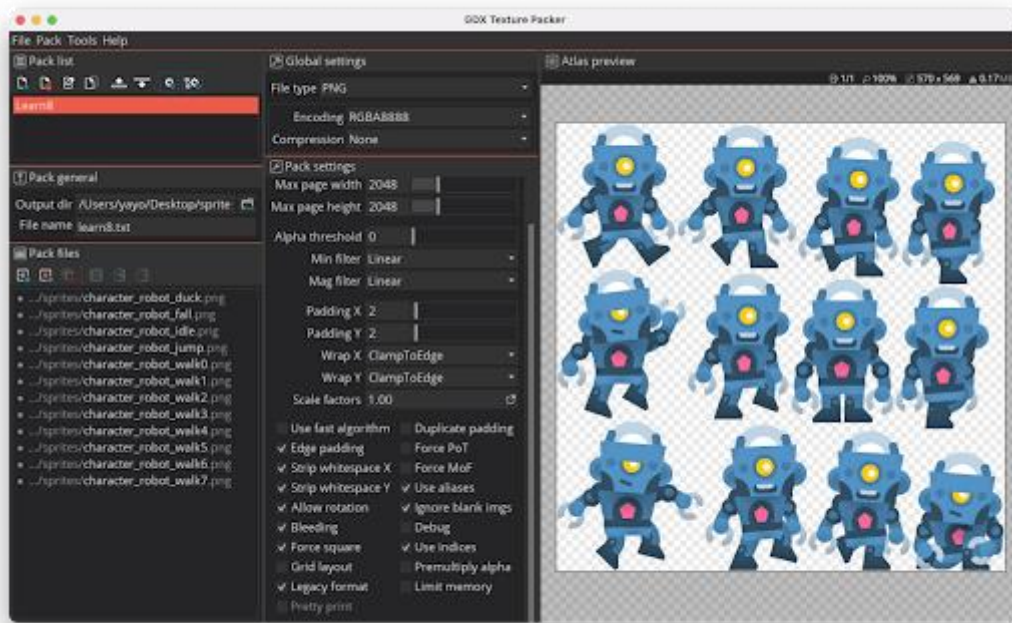
Por ejemplo en este atlas que hace referencia a las distintas animaciones usadas en el juego, se puede ver como resguarda la información que tiene el conjunto de imágenes unificado en un *.png* , y de esta manera se puede crear tanto mapas enteros de *png* que luego tengan información para el juego.

5.1.4. *GDX texture packer*



Y así como hemos hablado en el apartado anterior en este apartado hablaremos de cómo se pueden conseguir los atlas. Estos son autogenerados por una herramienta creada por los desarrolladores de la librería de LibGDX la cual es *GDX texture packer* esta herramienta unifica un conjunto de imágenes que pueden o no ser pertenecer a una secuencia. Ya que está diseñada para realizar el empaquetado de la manera más optima posible, dando en estas muchas opciones como se puede ver a continuación:

BOMBEMAN JAVA EDITION



Como se puede ver en esta imagen, ya no solo se puede crear el package propiamente dicho, sino que se puede dar distintas opciones para que se guarden de maneras diferentes. Para el desarrollo de nuestro proyecto se ha optado por una configuración de 1024x1024 en casi todos los mapas de imágenes, esto debido a que algunos poseen más información en imágenes y requieren una resolución diferente. Por parte del espaciado, siempre se ha dejado en cero y buscando la división en grid para dar más información a la hora de crear los mapas en Tiled.

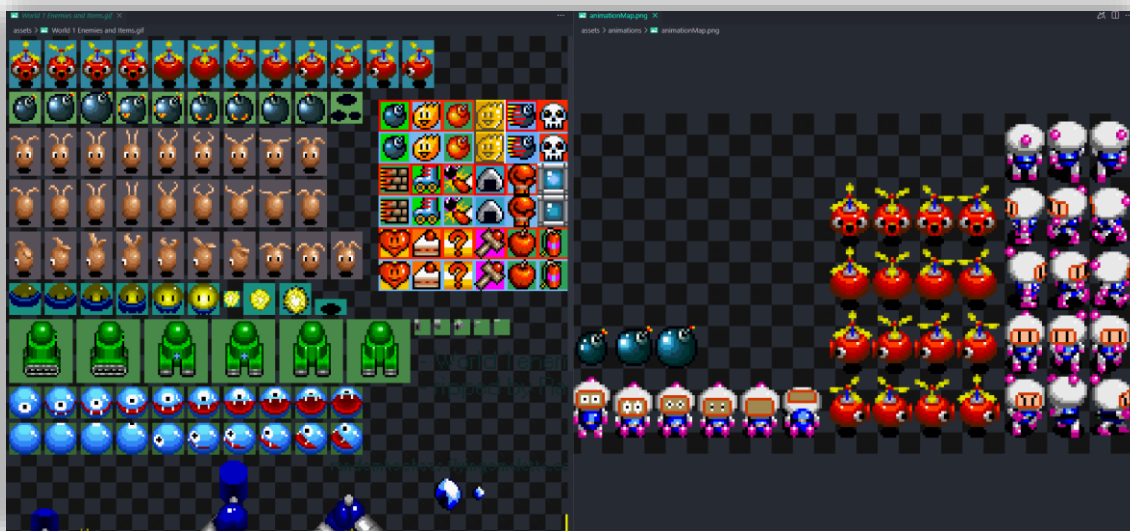
5.1.5. Photoshop



Adobe Photoshop es un editor de fotografías desarrollado por Adobe Systems Incorporated. Usado principalmente para el retoque de fotografías y gráficos, su nombre en español significa "taller de fotos". Es conocido mundialmente. Fue creado en 1986 por los hermanos Thomas Knoll y John Knoll, desde entonces se ha convertido en una marca de uso común, lo que lleva a su uso como un verbo, aunque Adobe desaconseja su uso.

Photoshop puede editar y componer imágenes rasterizadas y soporta varios modelos de colores: RGB, CMYK, CIELAB, colores sólidos y semitonos. Photoshop usa sus propios formatos de archivo PSD y PSB para soportar estas características. Desde junio de 2013, con la presentación de Creative Cloud, el esquema de licencia de Photoshop se cambió al modelo de software como servicio.

Y es que para poder apaliar algunas de las herramientas faltantes que ya te vienen por defecto en Unity, hemos utilizado a esta gran herramienta la cual da nos ha ayudado a la modificación de las imágenes del juego original. Esto es debido a que, en un primer momento las imágenes estaban planificadas para usarse en un sistema de Grid con relación al mapa que está dividido en una cuadrícula en la que cada espacio tiene un tamaño de 16x16. Pero en esta versión. Para dar más dinamismo al juego y aun usando las texturas del juego original. Este sistema de Grid no ha sido tan bien implementado. Resultando en un cambio de la mayoría de imágenes para quitarles el fondo y en algunos casos cambiando el orden de las imágenes para poder usarlas de manera más eficiente en nuestro juego, he aquí un ejemplo:



En esta imagen en el lado izquierdo se puede ver como la imagen original le da un fondo a los enemigos para identificarlos, además de poner solo algunos de sus movimientos y en un orden lineal. Es por eso que para este proyecto se le ha dado un orden diferente quitándole el fondo como se puede apreciar en el lado derecho de la imagen.

Esto se ha hecho principalmente por comodidad y optimización, ya que evitamos que el código haga los cambios en tiempo de ejecución y solo hacemos que siga la secuencia que queramos en el momento en el que lo deseemos.

5.1.6. Github



GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador. El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. Anteriormente era conocida como Logical Awesome LLC. El código de los proyectos alojados en GitHub se almacena generalmente de forma pública.

Y es que el control de versiones más usado en el mundo de la informática y es por ello que se ha implementado para poder tener un mejor resguardo de todos los cambios que se han tenido a lo largo de la creación de este proyecto y es que al ser un videojuego. El código resultante del mismo es enorme y difícil de mantener si no se usan herramientas como esta.

5.1.7. *Mysql / phpMyAdmin*



MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, todo para entornos de desarrollo web.

MySQL fue inicialmente desarrollado por MySQL AB (empresa fundada por David Axmark, Allan Larsson y Michael Widenius). MySQL AB fue adquirida por Sun Microsystems en 2008, y ésta a su vez fue comprada por Oracle Corporation en 2010, la cual ya era dueña desde 2005 de Innobase Oy, empresa finlandesa desarrolladora del motor InnoDB para MySQL.

Al contrario de proyectos como Apache, donde el software es desarrollado por una comunidad pública y los derechos de autor del código están en poder del autor individual,

MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código. Esto es lo que posibilita el esquema de doble licenciamiento anteriormente mencionado. La base de datos se distribuye en varias versiones, una Community, distribuida bajo la Licencia pública general de GNU, versión 2, y varias versiones Enterprise, para aquellas empresas que quieran incorporarlo en productos privativos. Las versiones Enterprise incluyen productos o servicios adicionales tales como herramientas de monitorización y asistencia técnica oficial. En 2009 se creó un fork denominado MariaDB por algunos desarrolladores (incluido algunos desarrolladores originales de MySQL) descontentos con el modelo de desarrollo y el hecho de que una misma empresa controle a la vez los productos MySQL y Oracle Database.

Está desarrollado en su mayor parte en ANSI C y C++. Tradicionalmente se considera uno de los cuatro componentes de la pila de desarrollo LAMP y WAMP.

MySQL es usado por muchos sitios web grandes y populares, como Wikipedia, Google (aunque no para búsquedas), Facebook, Twitter, Flickr, y YouTube.

Por estas razones es por lo que se ha decidido implementar esta base de datos en el proyecto ya que para los datos a guardar en la misma solo es requerido un cliente y la propia base de datos.

5.1.8. *LibGDX*



LibGDX es una librería para el desarrollo de videojuegos multiplataforma, soportando actualmente Windows, Linux, Mac OS, Android, IOS y HTML5

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

Además, LibGDX permite bajar el nivel de abstracción tanto como se quiera, dando acceso directo al sistema de archivos, dispositivos de entrada y audio, e incluso a las interfaces de OpenGL ES 2.0 y 3.0.

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

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

Dados estos atributos y dada su gran mantenimiento con respecto a otras librerías al igual que su documentación, hicieron de esta la mejor de las opciones para este proyecto, y es que aun siendo una Alpha, es gracias a esta librería que se ha podido llegar a una aproximación muy elaborada al que es el juego original.

5.1.9. Ashley



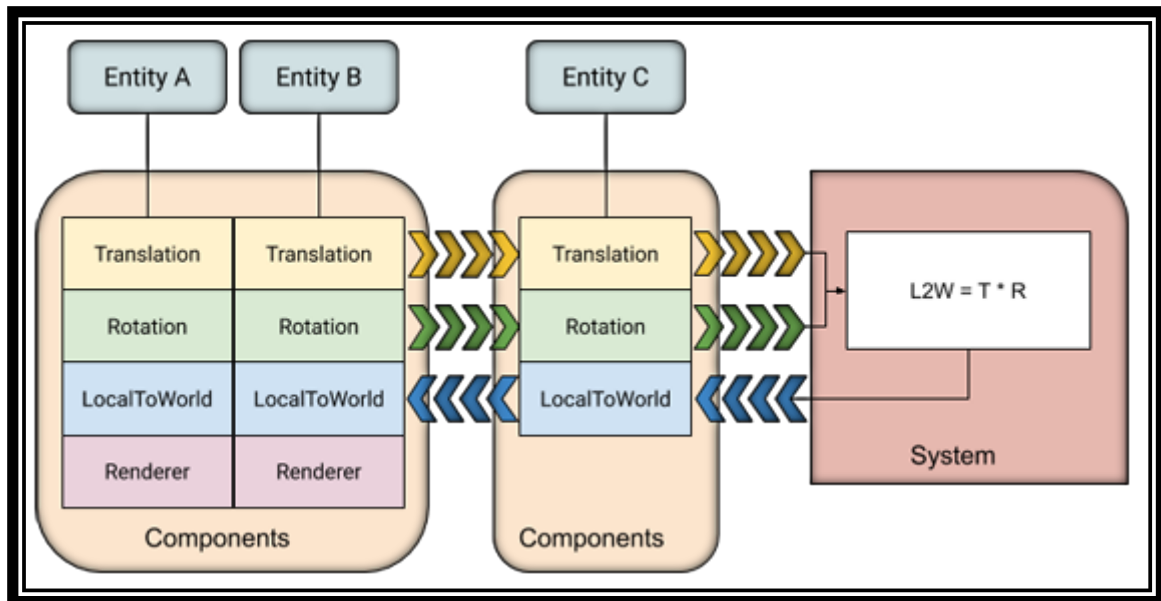
Ashley es uno de los frameworks que incluye LibGDX en su repertorio de herramientas. Este en específico se utiliza para la implementación de la arquitectura de ECS esta, explicándola más detalladamente:

Es una estructura que intenta dividir la carga de trabajo del equipo, formando grupos de Entidades, estas pueden ser cualquier elemento del juego ya que solo son marcos que ni siquiera poseen información por ellas mismas. De esto se encargan los componentes, que no son mas que contenedores de datos, generalmente generales, a esto nos referimos que pueden ser reutilizados, puede ser por ejemplo un *MoveComponent* el cual solo tenga la información que se requiera para moverse, en este caso las coordenadas de x e y.

Como se puede ver este componente luego puede ser usada por nuestro player o incluso nuestro enemigo que, aunque su sistema de movimiento sea diferente, si va a necesitar de los mismos atributos para moverse.

Posteriormente los sistemas serán los encargados de manipular esta información según nosotros lo programemos. De esta manera al finalizar, poseemos componentes reutilizables que, al ponerlo en una entidad en concreto, reaccionará a un sistema del juego para tener un comportamiento distinto en el mismo.

Esto se puede ver en el siguiente diagrama:



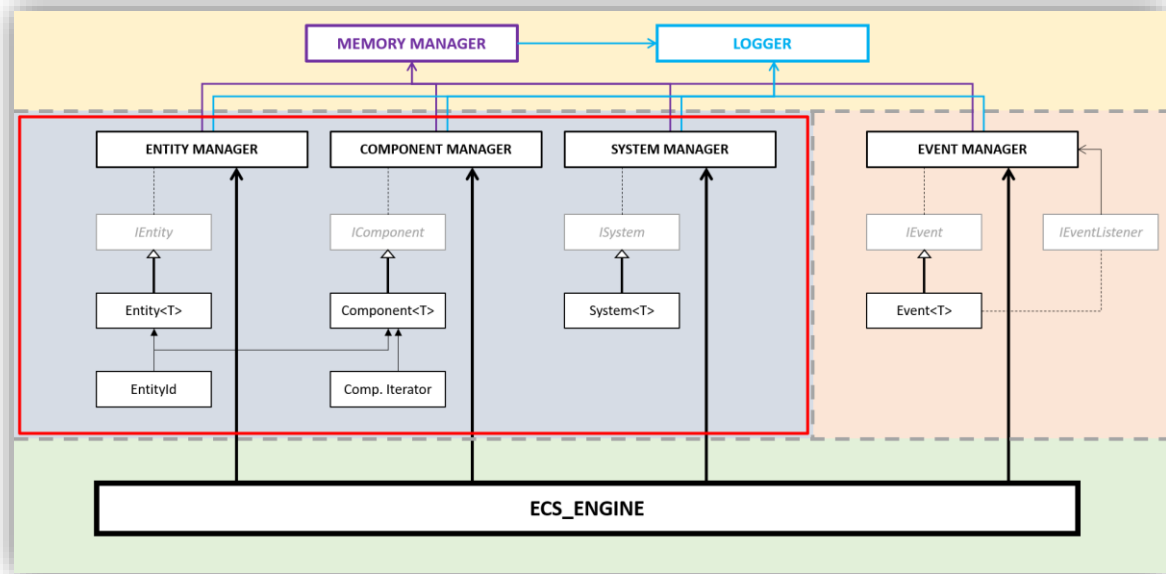
Esta arquitectura se puede desarrollar de muchas maneras diferentes y es que Ashley tiene una manera específica de hacerlo, pero hay muchas otras, por ejemplo, el que se usa en las últimas versiones de LibGDX *Artemis-ods* es un framework de ECS pero con muchas más opciones y más completo que Ashley.

Pero en este proyecto se eligió esta última debido a su gran facilidad de manejo y gran adaptabilidad para distintos tipos de proyectos.

En este caso Ashley, apuesta por un Engine que maneje todo el marco de ECS y se encargue de gestionar tanto las entidades como los sistemas y los componentes usando el Deltatime del juego.

Así mismo deja la opción de crear un Engine o un PoolEngine, los cuales se diferencian en eficiencia a la hora de lidiar con varias entidades al mismo tiempo, cosa la cual puede servir más en un juego shotter que se crean y destruyen entidades continuamente.

Esto se puede ver en el siguiente diagrama:



En el se puede ver mejor la gestión de Ashley con las entidades.

Para el proyecto que nos acontece en esta memoria, se opto por el uso del poolEngine, debido a la facilidad de implementación y además la mejora de rendimiento que esto conlleva a la máquina que este ejecutando el juego.

5.1.10. Java



Y en último lugar tenemos al lenguaje de programación elegido para este proyecto, el cual no es ninguno más que *Java* el cual es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán, probablemente, a menos que tengan Java instalado, y cada día se crean más. Java es rápido, seguro y fiable. Desde ordenadores portátiles hasta centros de datos, desde consolas para juegos hasta computadoras

avanzadas, desde teléfonos móviles hasta Internet, Java está en todas partes. Si es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos diez millones de usuarios reportados.

El lenguaje de programación Java fue desarrollado originalmente por James Gosling, de Sun Microsystems (constituida en 1983 y posteriormente adquirida el 27 de enero de 2010 por la compañía Oracle), y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales y librerías de clases en 1991, y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento de las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros han desarrollado también implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Ahora, si es verdad que, aunque posible, Java no es un lenguaje que se haya creado con la finalidad de hacer videojuegos, así como Lua por ejemplo. Pero su capacidad da la posibilidad de realizar estos proyectos, que, aunque un poco ineficientes, son capaces de igualar o superar a juegos hechos en otros lenguajes de programación.

Además de ser el primer lenguaje de programación que podíamos conocer de manera tan clara, para llevarlo al mundo de los videojuegos de forma, más o menos eficiente.

5.2.Hardware

En el apartado de Hardware podemos mencionar dos máquinas cruciales en las cuales fue ejecutado este proyecto y que lograron correrlo de manera efectiva.

El primero con las siguientes características:

- So: Windows 10 pro
- Arquitectura: x64
- Procesador: Intel(R) Xeon(R) CPU E5-2689 0 @ 2.60GHz 2.60 GHz
- Ram: 32GB

Y el segundo con:

- So: Ubuntu 20.1
- Arquitectura: x64
- Procesador: Intel Core i5 de 5ª generación
- Ram: 8GB

Como se puede apreciar, se ha intentado probar el juego en dos ordenadores diferentes para ver su desempeño y sus distintos fallos. Y es que como se esperaba, la compatibilidad es casi perfecta en todos los aspectos, quedando entre medio algunos errores visuales como pequeños artifacts en pantalla en momentos puntuales o mas rendimiento en algunos que en otros. Pero no deja de ser bastante impresionante que , gracias a Java y a la librería de LibGDX, podamos llegar a tener tal disponibilidad para tantos tipos distintos de Hardware. Resultando en casi la misma experiencia de usuario y solo desarrollándolo una vez.

5.3. *Humanos*

En relación al apartado humano utilizado, caben destacar dos fuentes de información muy importantes para el desarrollo de este proyecto, que si bien no fueron llevados a la práctica al cien por ciento de sus aptitudes, sí que se pudo saber más de la librería que se está utilizando, gracias a los siguientes:

- “*Mastering LibGDX Game Development*” de *Patrick Hoey*
- “*Java Game Development with LibGDX*” de *Lee Stemkoski*

Ambos libros cubren el desarrollo de videojuegos con la librería de LibGDX, pero lo hacen de distintas maneras de implementación para conseguir juegos completamente diferentes. Por ejemplo *Patrick* opta por el desarrollo de ECS mientras que *Lee* no lo hace, si es verdad que ambos tienen un nivel medio para el desarrollo de un juego a media escala, pero esto era más que suficiente para comprender los medios básicos para empezar con el buen desarrollo de nuestro proyecto.

Por otra parte, cabe destacar a aquellas personas que han probado el juego y que han podido apreciar tanto lo fallos como los aciertos del juego. Se ha optado por usar “Testers” dado el proyecto, que, si bien es un videojuego en Alpha, no hay que olvidar que esta destinado a uso de personas externas y su adaptabilidad al mismo.

En resumen, podemos decir, que, si bien el alcance del juego no ha sido excesivamente alto, se ha conseguido crear una Alpha que logra asemejarse al juego original y con mas tiempo y dedicación. Podrá ser igual que su antecesor.

6. Planificación del proyecto

Este apartado lo dividiremos de la siguiente manera:

- Búsqueda de la información
- Ajustes del entorno
- Ajustes iniciales del proyecto
- Creación de Screens
- ECS
- Managers
- Data Base

6.1. Búsqueda de la información

Para este proyecto se hizo un gran apartado de búsqueda de información para entender y comprender de manera clara lo que era, ya no solo crear un videojuego en Java, sino hacerlo casi desde cero con la ayuda de la librería de LibGDX. La cual, aunque soporta gran parte de las cosas que se quieren hacer en ella, si es verdad que se tienen que implementar gran parte de ellas para poder hacer que funcionen de la manera en la que tu necesitas en ese momento en concreto y aunque si es verdad que gracias a esto, la herramienta en si mismo llega a tener gran versatilidad. También logra hacer que su curva de aprendizaje sea algo elevada, pero no con ello imposible de entender.

Y es que gracias a los libros mencionados anteriormente y distintos videos asi como consejos por parte de otros desarrolladores de esta librería, se opto por usar la siguiente configuración de la misma:



En ella se puede ver como la versión de LibGDX (`gdxVersion`), no es otra que la 1.10.0 una de las últimas en salir, pero no la última, esto es debido a que, como ya se ha explicado anteriormente, las librerías posteriores a esta usan `Artemis_odb`, otro framework de ECS completamente diferente a `Ashley`, que, aunque posee más opción y optimización, debido a que es una implementación de una librería de C++. Es mucho más compleja que `Ashley` y que para una Alpha, no valía la pena alargar más el tiempo de versionado y busca de información de la aplicación.

Otros del aspecto que se barajó, fue el de ver como implementar la mayoría de las opciones que tiene el juego de forma nativa, lo cual fue descartando distintas opciones para procurar terminar a tiempo el proyecto. Y es que como se ha podido ver en el principio de esta memoria. El juego de `Bomberman`, aun siendo sencillo en su ejecución es largo en su extensión, lo cual nos ha dejado con la posibilidad de poner o no varias características. Las cuales, la mayoría fueron descartadas a causa del tiempo necesario para realizar una base para la misma.


Y una de las ultimas cosas que más dio que planificar fue, la fuente de información utilizada para la realización de este proyecto, el cual era muy variado y es que, aunque la documentación de la librería de `LibGDX` tiene una buena cantidad de información acerca del uso de las herramientas. Se hace muy complicada adaptarte a ella cuando

inicias en el desarrollo de videojuegos. Es por eso que se decidió optar por libros u otros medios, los cuales también escanciaban. Aun así, se pudo conseguir la información necesaria para poder concebir un producto que se asemeja a los objetivos propuestos.

6.2. Ajustes del entorno

Posteriormente se inició con el ajuste de entorno elegido. El cual tuvo sus complicaciones por tener dos formas de iniciación. En primer lugar, se tuvo que planificar, cual sería el alcance del proyecto dado que LibGDX permite la opción de desarrollar, ya no solo en distintas plataformas sino también con algún que otro lenguaje más, como lo es Kotlin.

Y aunque se pensó en un primer momento en la idea de usar todo el potencial que podía y puede dar esta librería, se optó por la opción de solo crear el juego para sistemas operativos de escritorio, debido a su facilidad y a su comodidad a la hora de trabajar, además de ser más rápido configurar las opciones de estos sistemas operativos que en otros. Al final este fue la configuración que se ha elegido:



```
1 public static void main (String[] arg) {  
2     Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();  
3     config.setForegroundFPS(60);  
4     config.setWindowIcon("Bomberman_icon.png");  
5     new Lwjgl3Application(new Bomberman(), config);  
6 }
```

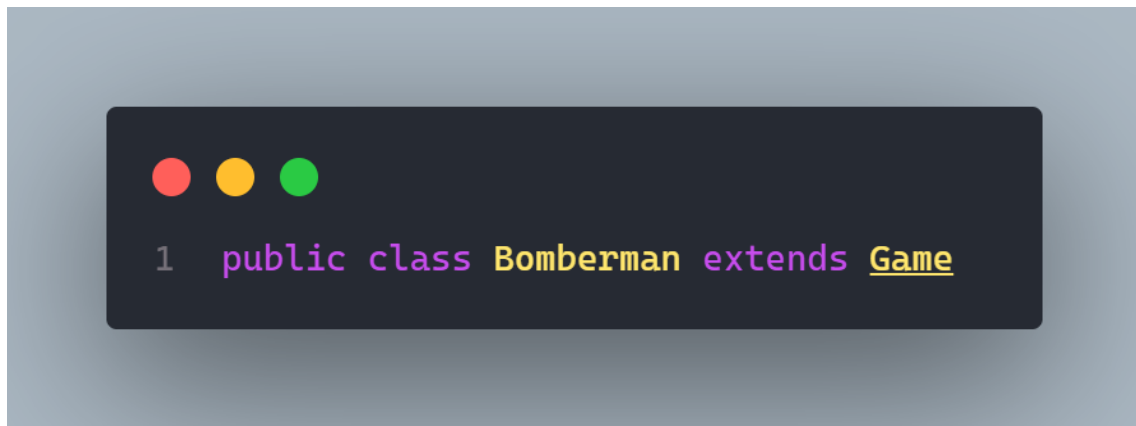
En esta se puede ver como se ha elegido una tasa de refresco de, máximo 60 FPS o fotogramas por segundo y un icono personalizado para la aplicación.

6.3. *Ajustes iniciales del proyecto*

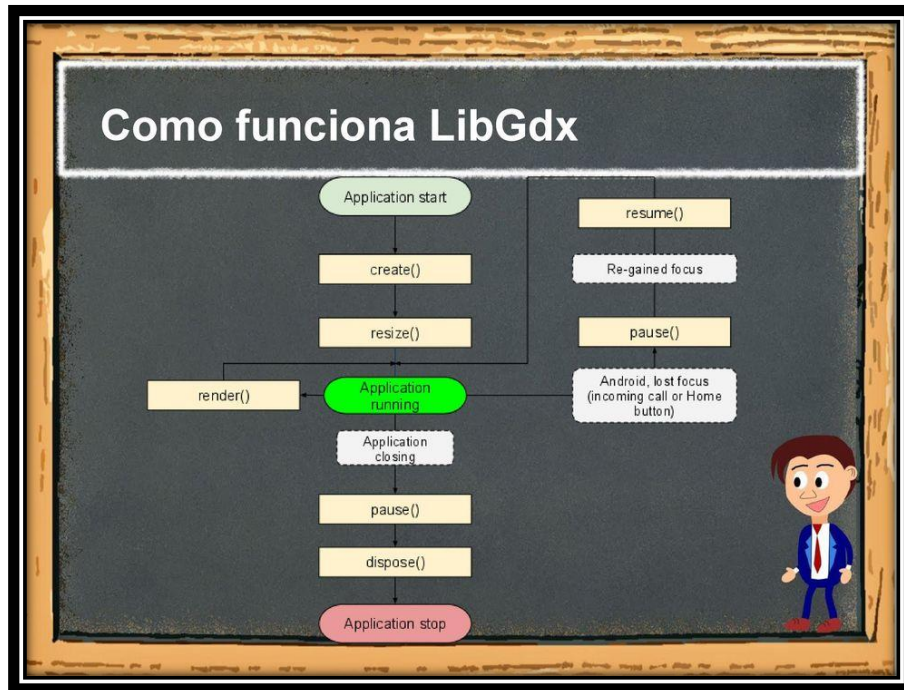
Para la configuración inicial del proyecto se tuvo en cuenta los siguientes factores:

- Extensión
- Igualdad a su antepasado
- Funcionalidad

Y una vez tenida clara estos factores se inició con el apartado central de la aplicación .



Esta clase principal es usada como monolito central por el cual todos los componentes se pueden llegar a crear y llamar desde distintas clases. Como se puede ver extiende de *Game* el cual es una clase principal de LibGDX que da las primeras configuración es y como tal debe de tener métodos asociados muy específicos estos son:



Como se puede ver en la imagen son varios métodos que pasa, aunque los principales son:

- create()
- render()
- dispose()

Estos métodos sirven para crear objetos, renderizarlos continuamente y eliminarlos respectivamente. Otros métodos como resume() pause() son métodos opcionales o usados en momentos concretos como minimizar una pantalla o girarla para dispositivos móviles.

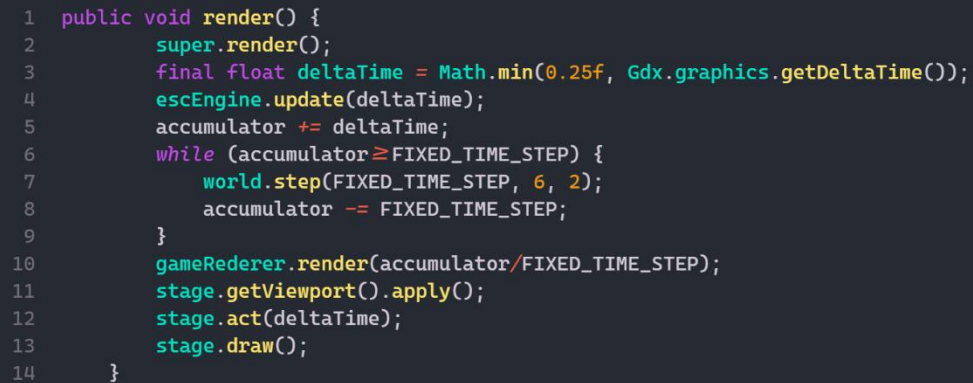
Una vez tenido clara esta diferenciación se crearon los distintos Objetos:

```
1  @Override
2  public void create() {
3      //set Debug Mode
4      Gdx.app.setLogLevel(Application.LOG_DEBUG);
5      spriteBatch = new SpriteBatch();
6      accumulator = 0;
7      //Box2d stuff
8      Box2D.init();
9      worldContactListener = new WorldContactManager();
10     world = new World(new Vector2(0,0), true);
11     world.setContactListener(worldContactListener);
12     //Initialize AssetManager
13     assetManager = new AssetManager();
14     assetManager.setLoader(TiledMap.class, new TmxMapLoader(assetManager.getFileHandleResolver()));
15     initializeSkin();
16     stage = new Stage(new FitViewport(Gdx.graphics.getWidth(), Gdx.graphics.getHeight(), spriteBatch));
17     //audio
18     audioManager = new AudioManager(this);
19     //Input creation
20     inputManager = new InputManager();
21     Gdx.input.setInputProcessor(new InputMultiplexer(inputManager, stage));
22     //Hud Manager
23     hudManager = new HudManager();
24     //ECS
25     escEngine = new ESCEngine(this);
26     //map manager
27     mapManager = new MapManager(this);
28     //Set first Screen
29     orthographicCamera = new OrthographicCamera();
30     screenViewport = new FitViewport(17, 16, orthographicCamera);
31     screenCache = new EnumMap<>(ScreenType.class);
32     setScreen(ScreenType.MENU);
33     //Game Render
34     gameRender = new GameRender(this);
35 }
```

Varios de estos Objetos son usados para la creación de del juego en cuestión, como lo son:

- Box2d: creación de cuerpos para colisiones
- World: creación de un mundo para mover los cuerpos
- AssertManager: Forma general de LibGDX para la obtención de recursos
- SpriteBath: Para el dibujado de los sprites
- Los distintos Managers: realizados para manipular distintos elementos dentro del juego
- OrthographicCamera: la cual es la camera del juego de forma Ortográfica (no tiene 3º dimensión)
- GameRendered: El cual es el que se encarga de renderizar la mayoría de animaciones e interacciones en el juego

Una vez creado los objetos, toca renderizarlos con el deltaTime del juego, el cual no es más que el tiempo que hay entre cada Frame.



```
1  public void render() {
2      super.render();
3      final float deltaTime = Math.min(0.25f, Gdx.graphics.getDeltaTime());
4      escEngine.update(deltaTime);
5      accumulator += deltaTime;
6      while (accumulator ≥ FIXED_TIME_STEP) {
7          world.step(FIXED_TIME_STEP, 6, 2);
8          accumulator -= FIXED_TIME_STEP;
9      }
10     gameRederer.render(accumulator/FIXED_TIME_STEP);
11     stage.getViewport().apply();
12     stage.act(deltaTime);
13     stage.draw();
14 }
```

En este caso se optó por realizar un pequeño arreglo en el deltaTime, y es que este era muy rápido en muchas ocasiones y muy lento en algunas, lo cual se debía, principalmente en la potencia de la máquina que se estaba probando, es por ello que se optó por realizar un pequeño acumulador que unificase ese tiempo perdido o ganado y de esta forma el juego fuese más fluido, independiente de la máquina en la que este corriendo.

Por último, se le dio a la clase principal la potestas de poder cambiar de Screen, cada vez que sea necesario y se pusieron algunos métodos generales comúnmente usados para la creación de Box2d.

BOMBEMAN JAVA EDITION

```
1 public void setScreen(final ScreenType screenType) {
2     final Screen screen = screenCache.get(screenType);
3     if (screen == null) {
4         try {
5             Gdx.app.debug(TAG, "Creando una nueva escena llamada: " + screenType);
6             final Screen newScreen = (Screen) ClassReflection
7                 .getConstructor(screenType.getScreenClass(), Bomberman.class).newInstance(this);
8             screenCache.put(screenType, newScreen);
9             setScreen(newScreen);
10        } catch (ReflectionException e) {
11            throw new GdxRuntimeException("La escena " + screenType + " no se ha creado por : ", e);
12        }
13    } else {
14        Gdx.app.debug(TAG, "Cambiendo a la scenea llamada: " + screenType);
15        setScreen(screen);
16    }
17 }
```

```
1 public static void resetBodieAndFixture() {
2     BODY_DEF.position.set(0, 0);
3     BODY_DEF.gravityScale = 1;
4     BODY_DEF.type = BodyType.StaticBody;
5     BODY_DEF.fixedRotation = false;
6
7     FIXTURE_DEF.density = 0;
8     FIXTURE_DEF.isSensor = false;
9     FIXTURE_DEF.restitution = 0;
10    FIXTURE_DEF.friction = 0.2f;
11    FIXTURE_DEF.filter.categoryBits = 0x0001;
12    FIXTURE_DEF.filter.maskBits = -1;
13    FIXTURE_DEF.shape = null;
14 }
```

Así como la configuración Básica para los distintos textos usados en el juego:

```

1 private void initializeSkin() {
2     // setup marhut color
3     Colors.put("orange", Color.ORANGE);
4     Colors.put("black", Color.BLACK);
5     Colors.put("white", Color.WHITE);
6     // converter a = x/255;
7     Colors.put("orangeGame", new Color(1f, 0.615f, 0.2f, 1));
8     Colors.put("blueGame", new Color(0.196f, 0.274f, 0.843f, 1));
9     // generate ttf bitmaps
10    ObjectMap<String, Object> resources = new ObjectMap<>();
11    FreeTypeFontGenerator fontGenerator = new FreeTypeFontGenerator(Gdx.files.internal("interface/font/font.ttf"));
12    FreeTypeFontParameter fontParameter = new FreeTypeFontParameter();
13    fontParameter.borderWidth = 2.5f;
14    fontParameter.borderColor = Color.BLACK;
15    fontParameter.minFilter = Texture.TextureFilter.Linear;
16    fontParameter.magFilter = Texture.TextureFilter.Linear;
17    final int[] sizesToCreate = { 12, 20, 26, 32 };
18    for (int size : sizesToCreate) {
19        fontParameter.size = size;
20        BitmapFont bitmapFont = fontGenerator.generateFont(fontParameter);
21        bitmapFont.getData().markupEnabled = true;
22        resources.put("font_" + size, bitmapFont);
23    }
24    fontGenerator.dispose();
25    // load skin
26    SkinLoader.SkinParameter skinParameter = new SkinParameter("interface/hud/hud.atlas", resources);
27    assetManager.load("interface/hud/hud.json", Skin.class, skinParameter);
28    assetManager.load("properties/Interface", I18NBundle.class);
29    assetManager.finishLoading();
30    skin = assetManager.get("interface/hud/hud.json", Skin.class);
31    i18nBundle = assetManager.get("properties/Interface", I18NBundle.class);
32 }

```

Cabe destacar que la información del Skin que no deja de ser la forma genérica de LibGDX de configurar los labels, buttons, tables, etc. Es en base a un Json:

```

1 {
2     "color": {
3         "white": {
4             "a": 1,
5             "b": 1,
6             "g": 1,
7             "r": 1
8         }
9     },
10    "com.badlogic.gdx.scenes.scene2d.ui.TextButton$TextButtonStyle": {
11        "smile": {
12            "font": "font_12",
13            "down": "buttonNormalPress",
14            "up": "buttonNormal",
15            "disabled": "buttonNormal"
16        },
17        "normal": {
18            "font": "font_20",
19            "down": "buttonNormalPress",
20            "up": "buttonNormal",
21            "disabled": "buttonNormal"
22        },
23        "big": {
24            "font": "font_26",
25            "down": "buttonNormalPress",
26            "up": "buttonNormal",
27            "disabled": "buttonNormal"
28        },
29        "huge": {
30            "font": "font_32",
31            "down": "buttonNormalPress",
32            "up": "buttonNormal",
33            "disabled": "buttonNormal"
34        },
35        "default": {
36            "font": "font_26",
37            "down": "buttonNormalPress",
38            "up": "buttonNormal",
39            "disabled": "buttonNormal"
40        },
41        "notBackgroundNormal": {
42            "font": "font_20"
43        },
44        "notBackgroundBig": {
45            "font": "font_26"
46        },
47        "label": {
48            "font": "font_20",
49            "up": "buttonNormalPress"
50        },
51        "hudBackground": {
52            "font": "font_12",
53            "up": "backgroundHudInformation"
54        }
55    },
56    "com.badlogic.gdx.scenes.scene2d.ui.ProgressBar$ProgressBarStyle": {
57        "default": {
58            "background": "knobAfter",
59            "knob": "knob",
60            "knobBefore": "knobBefore"
61        }
62    },
63    "com.badlogic.gdx.scenes.scene2d.ui.TextField$TextFieldStyle": {
64        "normal": {
65            "font": "font_20",
66            "fontColor": "white",
67            "background": "buttonNormalPress"
68        }
69    }
70 }

```

En el cual hemos podido especificar los valores por defecto de distintas clases, como, por ejemplo: Color, TextButton, ProgressBar y TextField.

Y estos a su vez son imágenes que se pueden conseguir desde el siguiente Png:



Cada una de estas imágenes tiene un valor que es recogido por un atlas y dado al Json, de esta manera luego el Skin puede sacar los datos y realizar el cambio visual en la aplicación.

6.4.Creación de Screens

En este apartado nos vamos a centrar en la preparación de uno de los elementos mas comunes en todo videojuego, las Screens, y es que, en todo Loading, pause, carga, guardar, opciones, cada una de las stages del juego. Se resuelve como una Screen que luego se puede modificar dependiendo de cuales sean las necesidades que se presenten.

Si es verdad que no se va a explicar a detalle como se han creado todas las screens de nuestro juego, mas, no obstante, podemos decir que todas de ellas siguen un esquema general para que el trabajo sea más rápido.

Por ende, la primero que presentaremos aquí es la clase general del que luego cuelgan las demás Screens:


```
1 public abstract class AbstractScreen<T extends Table> implements Screen , InputListener{
2     protected final Bomberman context;
3     protected final AssetManager assetManager;
4     protected final FitViewport viewport;
5     protected final World world;
6     protected final Stage stage;
7     protected final T screenUI;
8     protected final InputManager inputManager;
9     protected final AudioManager audioManager;
10
11     public AbstractScreen(final Bomberman context){
12         this.context=context;
13         this.assetManager = context.getAssetManager();
14         this.viewport = context.getScreenViewport();
15         this.world = context.getWorld();
16         this.stage = context.getStage();
17         this.screenUI = getScreenUI(context);
18         this.inputManager = context.getInputManager();
19         this.audioManager = context.getAudioManager();
20     }
21     protected abstract T getScreenUI(Bomberman context);
22     @Override
23     public void resize(int width, int height) {
24         viewport.update(width, height);
25         this.stage.getViewport().update(width, height,true);
26     }
27     @Override
28     public void show() {
29         inputManager.addInputListener(this);
30         stage.addActor(screenUI);
31     }
32     @Override
33     public void hide() {
34         inputManager.removeInputListener(this);
35         stage.getRoot().removeActor(screenUI);
36         audioManager.stopAudio();
37     }
38 }
```

Como se puede ver en esta Clase se implementan cosas básicas como un InputManager, el cual da la posibilidad de escuchar acciones del teclado, implementa Screen del propio LibGDX, y por ultimo consigue la mayoría de Objetos ya creados en la Clase principal.

A partir de aquí podremos crear distintas Clases hijas que tendrán por un lado un Ui y por otro la crase Screen que se encargara de realizar los cambios necesarios para su correcto funcionamiento.


```
1 public class GameScreen extends AbstractScreen<GameUi> implements MapListener{
2
3     private final MapManager mapManager;
4     private boolean isFirstTime = true;
5     public GameScreen(final Bomberman contextBomberman) {
6         super(contextBomberman);
7
8         this.mapManager = context.getMapManager();
9         mapManager.setMap(MapType.MAP_1);
10
11         context.getHudManager().setHud(screenUI);
12     };
13
14
15     @Override
16     public void render(float delta) {
17         if (isFirstTime) {
18             audioManager.playAudio(AudioType.START_PLAY);
19             Timer.schedule(new Task() {
20
21                 @Override
22                 public void run() {
23                     audioManager.playAudio(AudioType.STAGE1);
24                 }
25
26             }, 3f);
27             isFirstTime = false;
28         }
29
30         if (Gdx.input.isKeyJustPressed(Input.Keys.NUM_1)) {
31             mapManager.setMap(MapType.MAP_1);
32         } else if (Gdx.input.isKeyJustPressed(Input.Keys.NUM_2)) {
33             mapManager.setMap(MapType.MAP_2);
34         }
35     }
```

Por ejemplo, en la propia Screen del juego se puede ver como se recogen las acciones de los botones de teclado, y se inicia la música del juego a través de un hilo, lo cual es posible gracias a la clase *Timer* que implementa LibGDX.

Por otro lado, podemos explicar su Ui, que en este caso no va a ser otra cosa que el Hud, la información que se recoge de la partida y que se actualiza a tiempo real, para darle información al usuario.

```
1 public GameUi(Bomberman context) {  
2     super(context.getSkin());  
3  
4     i18nBundle = context.getI18nBundle();  
5     setPosition(64, 390);  
6     background("backgroundHud");  
7     setWidth(widthOfTheHud);  
8     setHeight(heightOfTheHud);  
9  
10    playerIcon = new Image(context.getSkin(), "playerIcon");  
11    lives = new TextButton("5", context.getSkin(), "hudBackground");  
12    scoreittle = new TextButton(i18nBundle.format("interface.score"), context.getSkin(), "notBackgroundBig");  
13    score = new TextButton("0", context.getSkin(), "hudBackground");  
14    timeIcon = new Image(context.getSkin(), "timerIcon");  
15    time = new TextButton("2:00", context.getSkin(), "hudBackground");  
16  
17    add(playerIcon).center().width(widthOfCells-ajustOfImages).height(heightOfCells).padRight(padding);  
18    add(lives).center().width(widthOfCells).height(heightOfCells);  
19    add(scoreittle).padLeft(padding);  
20    add(score).center().width(widthOfCells).height(heightOfCells);  
21    add(timeIcon).center().width(widthOfCells-ajustOfImages).height(heightOfCells).padLeft(padding);  
22    add(time).center().width(widthOfCells).height(heightOfCells);  
23 }  
24  
25 }
```

Y así como todas las demás pantallas, la forma de estructurar la información que se usa en esta ocasión es la de tablas. Gracias a las librerías de LibGDX de Ui, hay un gran abanico de opciones para crear las distintas vistas de nuestro juego. Y nosotras nos hemos decantado por la creación de una tabla que organice todos los componentes que queramos ponerle. En este caso no son más que componentes meramente informativos. Pero se pueden crear muchos más.

Cabe resaltar, que cada uno de los textos esta internacionalizado, de tal manera que el juego es tanto jugable en inglés como en español. Y esto es gracias a la librería de I18nBundle, La cual, gracias a distintos properties, logra esta acción.

6.5.ECS

En este apartado se explicará, de forma muy general, la integración de ECS en este proyecto.

Para empezar, hemos creado un Engine que, como se explicó anteriormente, este será el que gestione de manera interna toda la información que tendremos en nuestro juego. El resultante en este proyecto es algo así:



```
1 public ESCEngine(Bomberman context) {  
2     super();  
3     world = context.getWorld();  
4     this.addSystem(new TimeSystem(context, this));  
5     this.addSystem(new PortalSystem(context));  
6     this.addSystem(new PlayerMovementSystem(context));  
7     this.addSystem(new AnimationSystem(context));  
8     this.addSystem(new PlayerAnimationSystem(context));  
9     this.addSystem(new EnemyMovementSystem(context));  
10    this.addSystem(new AnimationMoveEnemySystem(context));  
11    this.addSystem(new PlayerAttackSystem(context, this));  
12    this.addSystem(new ExplotionSystem(context, this));  
13    this.addSystem(new FireSystem(context, this));  
14    this.addSystem(new BomDestructionSystem(context, this));  
15    this.addSystem(new DestructionOfGameObject(this));  
16    this.addSystem(new EnemyDieSystem(context, this));  
17    this.addSystem(new EnemyDieAnimationSystem( this));  
18    this.addSystem(new PlayerDieSystem(context, this));  
19    this.addSystem(new PlayerDieAnimationSystem(context, this));  
20    this.addSystem(new RemoveSystem());  
21 }
```

En esta primera parte se puede ver el constructor del Engine, el cual crea todos los sistemas que se van a usar el juego, entre estos existen algunos muy generales como el *RemoveSystem()*, y otros más específicos, esto es debido a que no todas las acciones se pueden generalizar para que funcionen correctamente.

Posteriormente existen métodos para la creación de entidades, de los cuales solo se explicara uno :

```
1 public void createPlayer(final Vector2 startSpawnLocation, final float width,final float height) {
2     final Entity player = this.createEntity();
3     //Add Components
4     final PlayerComponent playerComponent =this.createComponent(PlayerComponent.class);
5     playerComponent.timeToRecharge = 1000;
6     playerComponent.speed.set(3, 3);
7     playerComponent.lives = 5;
8     player.add(playerComponent);
9     //box2d component
10    resetBodieAndFixture();
11    final B2DComponent b2dComponent = this.createComponent(B2DComponent.class);
12    BODY_DEF.position.set(startSpawnLocation.x,startSpawnLocation.y);
13    BODY_DEF.fixedRotation = true;
14    BODY_DEF.type = BodyType.DynamicBody;
15    b2dComponent.body = world.createBody(BODY_DEF);
16    b2dComponent.body.setUserData(player);
17    b2dComponent.width = width;
18    b2dComponent.height = height;
19    b2dComponent.renderPosition.set(b2dComponent.body.getPosition());
20
21    FIXTURE_DEF.filter.categoryBits = BIT_PLAYER;
22    FIXTURE_DEF.filter.maskBits = BIT_GROUND | BIT_BOM | BIT_ENEMY | BIT_GAMEOBJECT|BIT_FIRE;
23    final PolygonShape pShape = new PolygonShape();
24    pShape.setAsBox(width,height,b2dComponent.body.getLocalCenter(), 0);
25    FIXTURE_DEF.shape = pShape;
26    b2dComponent.body.createFixture(FIXTURE_DEF);
27    pShape.dispose();
28
29    player.add(b2dComponent);
30    //animation
31    final AnimationComponent animationComponent= this.createComponent(AnimationComponent.class);
32    animationComponent.aniType = AnimationType.BOMBERMAN_DOWN;
33    animationComponent.width = 16 *UNIT_SCALE;
34    animationComponent.height = 16 *UNIT_SCALE;
35    player.add(animationComponent);
36
37    this.addEntity(player);
38 }
```

Este es el creador de la entidad del personaje principal, el jugador, como se puede ver se crea la entidad y se le dan disantos componentes para su correcto funcionamiento, como el B2dComponent, el cual resguarda toda la información de del cuerpo, y el AnimationComponent, que resguarda toda la información de las animaciones usadas para esa entidad en concreto.

6.6. Managers

Otro de los elementos más importantes del videojuego, es el uso de Managers. Estos realizan diversas acciones de las cuales solo se explicara una para su entendimiento general, ya que la mayoría sigue un mismo esquema de acción.

```
1 public class AudioManager {
2     private AudioType currentMusicType;
3     private Music currentMusic;
4     private AssetManager assetManager;
5
6     public AudioManager(Bomberman context){
7         this.assetManager = context.getAssetManager();
8         currentMusic = null;
9         currentMusicType = null;
10    }
11    public void playAudio(AudioType type) {
12        if (type.isMusic()) {
13            if (currentMusicType == type) {
14                return;
15            } else if (currentMusic != null) {
16                currentMusic.stop();
17            }
18            currentMusicType = type;
19            currentMusic = assetManager.get(type.getFilePath(), Music.class);
20            currentMusic.setLooping(true);
21            currentMusic.setVolume(type.getVolume());
22            currentMusic.play();
23        } else {
24            assetManager.get(type.getFilePath(), Sound.class).play(type.getVolume());
25        }
26    }
27    public void stopAudio() {
28        if (currentMusic != null) {
29            currentMusic.stop();
30            currentMusic = null;
31            currentMusicType = null;
32        }
33    }
34 }
```

En la mayoría de Managers de la aplicación siempre se intenta usar la misma estructura, la cual es, realizar una verificación del tipo de acción que se esta intentando cambiar, ya sea una acción por teclado, un Screen o un audio como es el de este caso.

Posteriormente se ve hace el cambio, buscándolo, en su mayoría, en un Enum que posee la información de lo que se este buscando. En este caso se hace una llamada al AudioType el cual conlleva toda la información de los audios del juego:

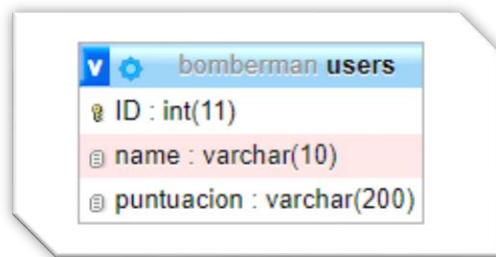
```
1  public enum AudioType {
2      INTRO("sounds/Tittle.mp3",true,0.3f),
3      SELECT("sounds/choseOption.mp3",false,0.3f),
4      STAGE1("sounds/Level_1.mp3",true,0.3f),
5      START_PLAY("sounds/Start.mp3",false,0.3f),
6      GAME_OVER("sounds/Game_Over.mp3",true,0.3f),
7      DIE_PLAYER("sounds/DiePlayer.ogg",false,0.3f),
8      DIE_ENEMY("sounds/EnemyDie.ogg",false,0.3f),
9      EXPLOSION("sounds/Explosion.ogg",false,0.3f),
10     PLACEBOM("sounds/PlaceBomb.ogg",false,0.3f),
11     TELEPORT("sounds/Teleport.ogg",false,0.3f);
12
13     private final String filePath;
14     private final boolean isMusic;
15     private final float volume;
16     private AudioType(String filePath, boolean isMusic, float volume) {
17         this.filePath = filePath;
18         this.isMusic = isMusic;
19         this.volume = volume;
20     }
21     public String getFilePath() {
22         return filePath;
23     }
24     public boolean isMusic() {
25         return isMusic;
26     }
27     public float getVolume() {
28         return volume;
29     }
30 }
```

El cual en este caso posee esta forma, pero, evidentemente, se ajustará mas o menos dependiendo de cuál sea la información que se quiera meter en el juego.

6.7.Data base

El último punto que me gustaría recalcar es el tema de la base de datos que tiene el juego que, si bien no es muy grande, es una de las partes que le dan mas variedad al juego ya que el hecho mismo de ver la puntuación de otras personas da ese animo a querer mejorar en nuestro juego.

El diagrama del mismo sería:



Y es que como se ve, solo se compone de una tabla con dos registros, esto es debido a que no necesitamos más. Ya que, gracias a esta tabla, podemos realizar la siguiente vista:



Como se puede ver en ella solo se muestran los valores que se han ido registrando en la base de datos.

Pero logra dar a los usuarios esa sensación de competitividad que en algunos juegos logra mantener más a

los jugadores y aunque esto es algo que a día de hoy es mas de los juegos shooters o más competitivos. Pero también es un recurso que se usaban en los juegos de maquinas recreativas en las cuales se tenia este tipo de interfaces.

7. Fases de prueba

Una de las cuestiones que más se ha podido probar y al la vez que menos se ha podido solucionar, debido al tiempo el cual se le ha propuesto al proyecto, esto se debe a que, al ser un videojuego, este se tiene que estar continuamente viendo que los componentes funciones de acuerdo a lo planificado y esto se intenta comprobando constantemente la aplicación.

Pero, así como se ha probado tanto, también se han tenido que dejar varias cosas atrás, debido al tiempo que tomaría implantarlas en el videojuego, una de ella, y la mas visual, es el tema de las bombas. Y es que, aunque el juego coste con un sistema de ataque basado en bombas para el jugador. Su implementación o al menos como era en el juego real, no es del todo exacto. Como se puede ver a continuación:



En esta imagen se puede ver como la bomba no es un objeto variable como el juego original, sino que tiene un valor de tamaño fijo para que sea más fácil su ejecución. A su vez se puede ver como los B2box de los objetos, son empujados por la bomba, esto es debido a que, al momento de crear la bomba en el juego, esta ocupa un espacio, un espacio el cual para otros objetos debería de ser imposible tocar y esto crea este efecto de rebote en los objetos e incluso en la misma bomba, que se mueve al estallar.

Estos y más defectos como, fallos en las colisiones, algunos eventos que no se dan o se dan múltiples veces. Son ejemplos de errores, que, si bien están bien identificados, su resolución tomaría mas que el desarrollo del juego en su fase Alpha. Por lo que se ha pensado que lo mejor es para y dar una solución más “sencilla” pero funcional.

Estos errores son lo que, gracias a las continuas pruebas de usabilidad y gracias a algunas personas que han probado el juego, hemos podido identificar. Pero si es verdad que, como casi todos los juegos, este sería el mejor punto para hacer una fase de testeo por un grupo de betatesters, para tener más información de los fallos y tenerlos mas identificados he incluso clasificarlos dependiendo de su gravedad para su futura reparación.

Aunque como bien se ha dicho desde un principio, este no es mas un Alpha de lo que se quiere conseguir. Una representación del juego original multiplataforma.

8. Conclusiones y trabajos futuros

Como es evidente, uno de los trabajos que mas se quiere concretar es la finalización del este proyecto, pudiendo sacar la versión completa. Pero es bien sabido que eso consta de tiempo y mucho esfuerzo a su vez hay que planificar bien los pasos a seguir para conseguir el mayor acercamiento posible.

8.1. Organización del código

Una de las primeras cosas que se quiere realizar es la organización del código, y es que no es que el código no tenga un orden ahora en su Alpha, pero si es verdad que , dado al desconocimiento. Este trabajo requiere de más organización para que se pueda encontrar las cosas más rápidamente.

Por esta razón se requiere de una estructura parecida pero más versátil para tener mas claridad de las clases que estas posee.

8.2. Tests

Otras de las cosas que se quiere implementar con más fuerza es el uso de test tanto unitarios como generales que nos den más información de los distintos pasos por los cuales este pasando la aplicación y de esta manera poder identificar más rápidamente los fallos u arreglar más detalles para tener un acabado más profesional en el proyecto final.

8.3. Corrección de errores conocidos

Como es evidente, se quiere hacer una corrección exhaustiva de los distintos errores ya conocidos y gestionarlos de tal manera que sean más parecidos al del juego original.

8.4. Mayor uso de Git y Gradle

Y es que es verdad que no se ha podido explotar estas tecnologías tanto como se ha querido en este proyecto.

Primeramente, Gradle así como Maven, otro gestor de dependencias más conocido, tiene un ciclo de vida que permite automatizar, ya no solo las dependencias generales del proyecto, sino dependencias específicas que se quiera tener en cada uno de los ciclos del software, he incluso incluir algunos únicos para un proyecto en específico. Y es que esa versatilidad hace de Gradle una herramienta muy potente que, si bien no se esta desperdiciando del todo ya que resguarda las dependencias por proyecto. Si es verdad que se puede ampliar mas uso y es eso mismo lo que se quiere lograr con esta herramienta.

Por otra parte, Git, una herramienta de control de versiones que en este caso solo sirve como almacén de información de los distintos primeros pasos del proyecto y, aunque útil, al igual que Gradle, se le puede sacar aun mas jugo a esta herramienta para poder gestionar mejor lo que se sube al repositorio y cuando, he incluso, como se sube a este.

9. Referencias Bibliográficas

9.1. Libros

- “*Mastering LibGDX Game Development*” de *Patrick Hoey*
- “*Java Game Development with LibGDX*” de *Lee Stemkoski*

9.2. Artículos y páginas

- [Wiki - libGDX](#)
- [group com.badlogicgames.gdx has published 15 artifact\(s\) with total 588 version\(s\) \(javadoc.io\)](#)
- [Set Up a Dev Env - libGDX](#)
- [Create something awesome! | HyperLap2D Wiki \(rednblackgames.github.io\)](#)
- [libgdx/ashley: A Java entity system inspired by Ash & Artemis. \(github.com\)](#)

- [Home · junkdog/artemis-odb Wiki \(github.com\)](#)
- [Tutorial - Programación de juegos para principiantes \(edu4java.com\)](#)
- [memoria.pdf \(ub.edu\)](#)
- [Desarrollo de juegos con Libgdx y Android Studio | jc-Mouse.net](#)
- [Tiled Documentation — Tiled 1.8.4 documentation \(mapeditor.org\)](#)
- [Welcome \(adobe.com\)](#)
- [ECS concepts | Package Manager UI website \(unity3d.com\)](#)
- [Entity component system - Wikipedia](#)

9.3. Videos

- [\(224\) 1 - Tutorial LibGDX: Comenzando con LibGDX - YouTube](#)
- [\(224\) Tutorial de libGDX – 1. Instalación y creación de proyectos - YouTube](#)
- [\(224\) How to make your own 2D game *01* Introduction \[Java|LibGDX|Box2D/-light|Ashley|Tiled\] - YouTube](#)
- [\(224\) Introducción a la arquitectura Entity-Component-System \(ECS\) - YouTube](#)
- [\(224\) Entidad-Componente-Sistemas: arquitectura para juegos en HTML5 con Phaser | JSDay Canarias 2018 - YouTube](#)