Trabajo Final de Grado en Ingeniería Informática

DESARROLLO DE UN VIDEOJUEGO MULTIPLATAFORMA EN JAVA

Autor Natalio Elías Sacerdote
Tutor Javier Sánchez Pérez
Curso 2014/2015
Convocatoria Extraordinaria – Diciembre 2014
Grado en Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Agradecimientos

Antes de empezar me gustaría agradecer a todas las personas que me han ayudado durante la realización de este trabajo de fin de grado.

A mi tutor, Javier, que me ha guiado, orientado y supervisado a lo largo de todo el proyecto ofreciéndome su ayuda en todo momento.

A Cristina, que a pesar de lo largo que ha sido este proyecto, no ha parado de animarme y de darme ideas para mejorarlo.

A mis familiares y amigos, que siempre me han alentado a seguir adelante.

A todos ellos, muchas gracias.

Índice

1. Introducción

El videojuego que vamos a desarrollar será un juego de acción, concretamente un videojuego de plataformas, con una vista de desplazamiento horizontal.

Un videojuego de acción es un videojuego en el que el jugador debe usar su velocidad, destreza y tiempo de reacción para superar los obstáculos que se le presentan, el género de acción es el más amplio del mundo de los videojuegos, englobando muchos subgéneros como videojuegos de lucha, videojuegos de disparos en primera persona, beat 'em ups y videojuegos de plataformas.

Los videojuegos de plataformas o, simplemente, plataformas, son un género de videojuegos que se caracterizan por tener que caminar, correr, saltar o escalar sobre una serie de plataformas y acantilados, con enemigos, mientras se recogen objetos para poder completar el juego. Este tipo de videojuegos suelen usar vistas de desplazamiento horizontal hacia la izquierda o hacia la derecha y representan un género muy popular de videojuegos, surgido a comienzo de la década de 1980 y que sigue manteniendo bastante popularidad en la actualidad.

El juego contará con una serie de niveles agrupados en mundos, en cada nivel el jugador controlará a un personaje humanoide con el que deberá superar diversos obstáculos, recolectar objetos que incrementen su puntuación y derrotar a los enemigos con los que se encuentre. En el último nivel de cada mundo el jugador se enfrentará a un enemigo especial (Jefe Final) al que deberá derrotar para poder acceder al siguiente Mundo. Un mundo no es más que un conjunto de diez niveles ordenados y con una temática en común y un “Jefe Final” en el último de los niveles.

Los obstáculos que nos podemos encontrar en un nivel consistirán en caídas al vacío, trampas, etc., para incrementar nuestra puntuación podremos recolectar monedas y también eliminar enemigos que tendrán distintas características: velocidad, capacidad de disparo, capacidad de salto, etc. Así, la puntuación de un jugador al final de cada nivel se calcula a partir del número de objetos recolectados y la cantidad de enemigos eliminados.

Para el proceso de desarrollo de este proyecto se ha elegido el Proceso Unificado de Desarrollo (PUD).

1.2 Motivación y objetivos

1.2.1 Motivación profesional

A través del tiempo, los videojuegos han sido vistos como un tipo entretenimiento exclusivo para personas jóvenes. Pero esta concepción ha ido cambiando y hoy en día es cada vez más habitual encontrar personas de cualquier rango de edades disfrutando de este entretenimiento.

A nivel educativo, existen diversos estudios que demuestran que los videojuegos pueden incidir positivamente en el desarrollo de ciertas habilidades y destrezas. Por ejemplo, un estudio realizado en la Universidad de California con un grupo de 200 universitarios de Estados Unidos e Italia demostró que jugar con videojuegos contribuye a la formación informática en aspectos científico-técnicos. También existe otro estudio que hace una valoración positiva de los

videojuegos en lo que respecta al desarrollo de destrezas y capacidades de tipo multisensorial, tanto auditivo como visual, y kinestésico; y un tercer trabajo estudió las capacidades espaciales de los videojuegos y alcanzó resultados muy positivos. En consonancia con ellos, otros estudios consideran que los videojuegos permiten un alto de desarrollo de habilidades ligadas a la lateralidad y la coordinación óculo-manual.

A nivel económico, a pesar de su juventud, la industria de los videojuegos ha experimentado un crecimiento extraordinario y los ingresos totales de la industria a nivel mundial se estiman en 93.282.000.000$ para 2013.

Por otra parte, recientemente, las plataformas móviles han ganado importancia, en mayo de 2013, el 56% de los usuarios de telefonía utilizaba ya un smartphone. Este dato representaba el 35% hace sólo dos años, lo que da una idea de la rápida penetración del smartphone entre usuarios de todo el mundo.

Como ingeniero informático el desarrollo de un videojuego resulta muy interesante, abarca muchas de las facetas que un ingeniero informático tiene que dominar. Es necesario aplicar una gran dosis de ingeniería del software, ser capaz de crear de gráficos por ordenador, y llevar a cabo el diseño de las interfaces necesarias.

Es principalmente por todos estos motivos que surge la idea de hacer este proyecto en el que crearemos un videojuego, con la capacidad de ejecutarse en plataformas móviles.

1.2.2 Motivación personal

Desde que tengo memoria siempre he estado conectado con el mundo de los videojuegos, mis andaduras empezaron en un Intel Pentium 133 MHz con juegos clásicos como el “Doom” ejecutándose en MS-DOS, desde entonces no he parado de disfrutar de una afición que poco a poco va ganando más adeptos. Cuando me planteé qué quería hacer como trabajo de fin de grado tenía claro que sería un videojuego.

1.2.3 Objetivos

El objetivo principal de este trabajo será la creación de un videojuego, como objetivos secundarios tenemos:

- Crear un prototipo que cubra el ciclo de vida completo de un juego con las siguientes características como mínimo:
  - Niveles
  - Mundos
  - Objetos recolectables
  - Obstáculos
  - Enemigos
  - Enemigo Final
  - Menús
- Debe ser capaz de ejecutarse en distintas plataformas incluyendo como mínimo:
  - Dispositivos móviles Android
  - PC Windows
  - PC Mac
- Utilizar una metodología de desarrollo de software (Proceso Unificado de Desarrollo)

Para conseguir estos objetivos elegimos Java, debido a su capacidad multiplataforma, y la librería libGDX, porque encaja perfectamente con nuestros objetivos.

1.3 Aportaciones

Tradicionalmente cuando queremos desarrollar un videojuego capaz de ejecutarse en distintas plataformas no teníamos más remedio que desarrollar una versión del videojuego para cada una de ellas.

Con este trabajo se pretende desarrollar un videojuego que sea capaz de ejecutarse en dispositivos móviles y ordenadores personales, minimizando la cantidad necesaria de código específico requerido por plataforma.

1.4 Organización del documento

En este documento explicaremos el proceso que se ha seguido para desarrollar la aplicación resultante. Contará con seis capítulos en los que nos centraremos en el trabajo realizado y un conjunto de anexos en los que se detallarán las competencias aplicadas y se proporcionará un manual de usuario.

En el primer capítulo nos hemos centrado en dar una introducción a los conceptos que se utilizarán. Además hemos explicado el problema a resolver y detallado los objetivos y las aportaciones del trabajo.

El segundo capítulo explica el estado del arte, en nuestro caso, hablaremos de cómo se encuentra la industria de los videojuegos actualmente, repasaremos la historia de los videojuegos de plataforma y mencionaremos los principales competidores dentro de este género.

En el tercer capítulo detallaremos los recursos utilizados para llevar a cabo este proyecto.

El cuarto capítulo hablaremos sobre la planificación del trabajo. Para ello, explicaremos nuestro plan de trabajo y realizaremos el presupuesto necesario para desarrollar el software. También hablaremos sobre la metodología de desarrollo empleada y haremos una estimación del tiempo necesario para cada una de las fases del proyecto.

En el quinto capítulo nos centraremos en detallar las etapas del desarrollo y en cómo se han llevado a cabo estas etapas utilizando la metodología descrita con anterioridad.

Finalmente, en el sexto capítulo expondremos las conclusiones obtenidas y las posibles ampliaciones futuras de este proyecto.

Cerraremos el documento con un anexo en el que se detallan las competencias aplicadas, otro anexo en el que se proporcionará un manual de usuario y un apartado para la bibliografía utilizada en este proyecto.

2. Estado del arte

Dentro del ambiente tecnológico industrial, se entiende como "estado del arte", "estado de la técnica" o "estado de la cuestión", todos aquellos desarrollos de última tecnología realizados a un producto, que han sido probados en la industria y han sido acogidos y aceptados por diferentes fabricantes.

También aprovecharemos para comentar algunos videojuegos de plataformas populares, así como entornos de desarrollo de videojuegos y frameworks conocidos.

2.1 Videojuegos

La industria de los videojuegos es una industria relativamente joven, tiene su origen en la década de 1940 cuando tras finalizar la segunda guerra mundial, las potencias vencedoras construyeron los primeros supercomputadores programables. Los primeros programas de carácter lúdico no tardaron en aparecer (inicialmente programas de ajedrez), pero no es hasta la década de los 60 cuando nos encontramos con los primeros videojuegos modernos. Desde entonces el mundo de los videojuegos no ha cesado de crecer y desarrollarse teniendo como único límite el impuesto por la creatividad de los desarrolladores y la evolución de la tecnología.

Nacido como un experimento en el ámbito académico, los videojuegos lograron establecerse como un producto de consumo de masas en tan sólo diez años, ejerciendo un formidable impacto en las nuevas generaciones que veían los videojuegos con un novedoso medio audiovisual que les permitiría protagonizar en adelante sus propias historias.

A pesar de su juventud, ha experimentado un crecimiento extraordinario y los ingresos totales de la industria a nivel mundial se estiman en 93.282.000.000$ para 2013 (Fuente: Gartner, Octubre 2013).

Las plataformas en las que nos podemos encontrar videojuegos actualmente son muy variadas, tenemos máquinas arcade, consolas, consolas portátiles, ordenadores, ordenadores portátiles, tabletas y móviles.

2.2 Videojuego de plataformas

Los videojuegos de plataformas o, simplemente, plataformas, son un género de videojuegos que se caracterizan por tener que caminar, correr, saltar o escalar sobre una serie de plataformas y acantilados, con enemigos, mientras se recogen objetos para poder completar el juego.

El género nace en la década de los 80 y como pionero del género nos encontramos con el mítico Donkey Kong (Nintendo, 1981) la mecánica del juego se centra en controlar al personaje sobre una serie de plataformas mientras se evitan obstáculos. También en esta misma década nos encontramos con el extraordinario Super Mario Bros (NES, Nintendo, 1985), que introdujo en el género de plataformas una serie de elementos que sentarían precedente: la recolección de 100 ítems para conseguir vidas extra (en este caso, monedas), y la división en subniveles de los mundos.

El género ha seguido evolucionando a lo largo de los años pasando por la migración hacia las tres dimensiones, sin embargo, recientemente se ha recuperado la vista en dos dimensiones debido a; menores costes de desarrollo, la posibilidad de proporcionar un apartado artístico más elaborado y los problemas de cámara que sufren muchos plataformas en tres dimensiones. Ejemplos de esta vuelta de las dos dimensiones son Super Meat Boy (2010, Team Meat), Sonic Generations (2011, Sonic Team) y FEZ (2012, Polytron).

2.3 Ejemplos de videojuegos de plataformas

En el mercado actual nos encontramos con muchísima competencia en los videojuegos de plataformas, nos limitaremos a mencionar los más reseñables:

- *Super Mario Bros.:* el rey indiscutible de los videojuegos de plataforma. La última versión del videojuego se llama New Super Mario Bros. U y fue lanzada en Europa a finales 2012 y es exclusiva de la consola Wii U. Una de las novedades introducidas en las últimas versiones del videojuego ha sido la capacidad de que múltiples jugadores participen en la partida a la vez.

Ilustración 1: New Super Mario Bros. U

- *Rayman Origins:* es un juego de plataformas con elementos de resolución de puzzles. Fue lanzado a finales de 2011 por Ubisoft. El juego contiene modos para un solo jugador y multijugador, con soporte para 4 jugadores. El objetivo es completar los niveles y salvar el Claro de los sueños. En general, hay más de 60 niveles en el juego, que tendrán lugar en 12 entornos diferentes.

Ilustración 2: Rayman Origins

- *Dustforce:* desarrollado por Hitbox Team, el juego se lanzó a principios de 2012. En este juego el jugador controla a uno de los cuatro barrenderos que tienen como misión limpiar un mundo lleno de suciedad. Los personajes tienen habilidades más propias de un ninja que les permitirán cumplir su cometido, limpiando distintos rincones y liberando de la suciedad a las criaturas que ahí viven.

Ilustración 3: Dustforce

- *Super Meat Boy:* desarrollado por Team Meat y lanzado a finales de 2010. el jugador controla a un personaje llamado "Meat Boy", cuya misión es rescatar a su novia "Bandage Girl" del científico malvado "Dr. Fetus". El desarrollo de la historia se divide en varios capítulos y en conjunto tiene más de 300 niveles de juego.

Ilustración 4: Super Meat Boy

2.4 Herramientas de desarrollo de videojuegos

2.4.1 Plataformas de desarrollo

Es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica, entre otras características. Son herramientas que proporcionan un nivel de abstracción mayor que un framework, a continuación listaremos algunos de los más populares para el desarrollo de videojuegos:

- Unity: es un ecosistema de desarrollo de juegos: un poderoso motor de renderizado totalmente integrado con un conjunto completo de herramientas intuitivas y flujos de trabajo rápido para crear contenido 3D interactivo; publicación multiplataforma sencilla y una Comunidad donde se intercambian conocimientos.

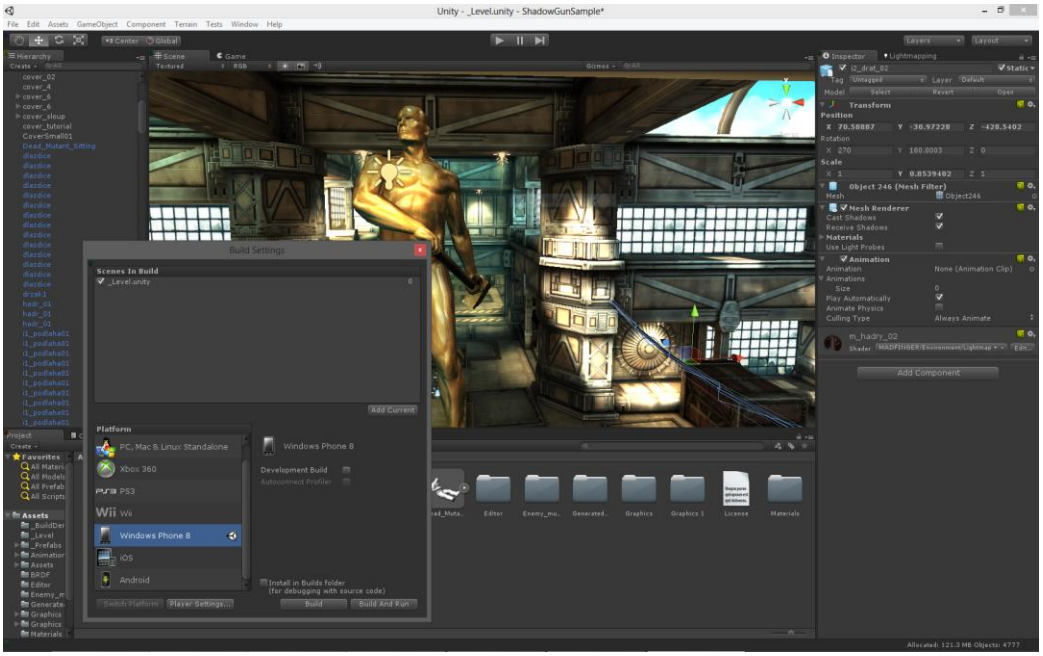


Ilustración 5: Interfaz Unity

- UDK: es la versión gratuita de Unreal Engine 3, proporciona acceso a un conjunto de herramientas profesionales que se utilizan en grandes producciones de videojuegos. También permite desarrollo para móviles, renderizados 3D, creación de películas de animación y más.

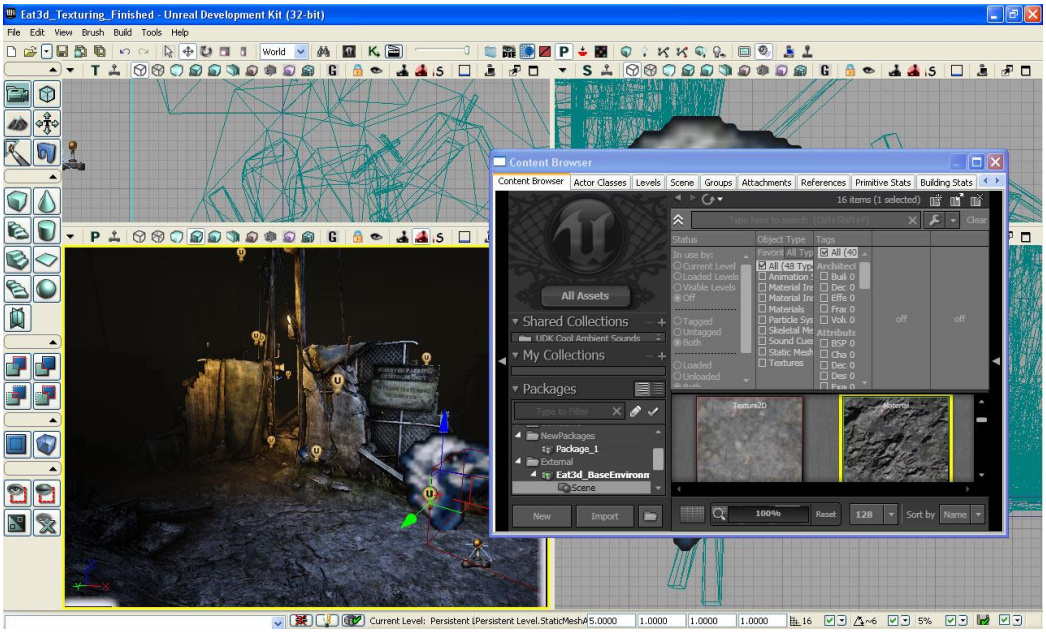


Ilustración 6: Interfaz UDK

- GameMaker: permite a los desarrolladores crear videojuegos multiplataforma. Permite exportar los videojuegos a: Android, iOS, HTML5, Mac, Windows, Windows Phone, entre otras.

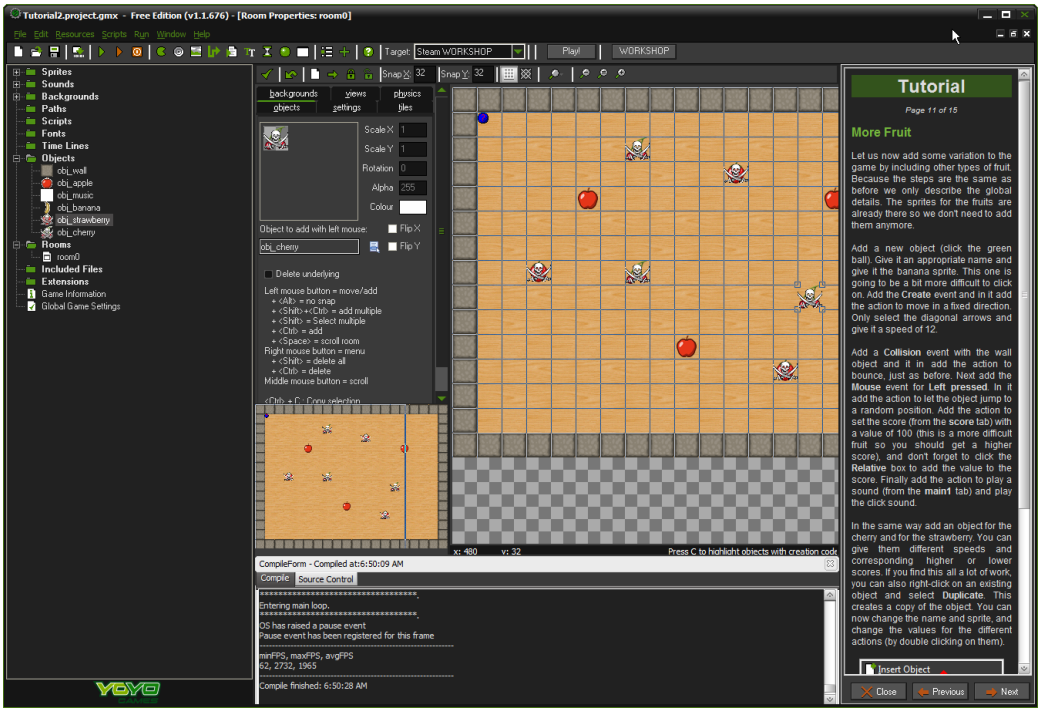


Ilustración 7: Interfaz Game Maker

2.4.2 Frameworks

En el desarrollo de software, un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. Normalmente, un framework nos permite desarrollar un videojuego con un nivel de profundidad mayor que un entorno de desarrollo.

- AndEngine: es un motor pensado para videojuegos en 2D, permite a los desarrolladores, experimentados o novatos, desarrollar juegos para Android con facilidad. A pesar de esta simplicidad AndEngine incluye suficientes funcionalidades para crear cualquier tipo de juego en dos dimensiones.
- LimeJS: es un framework de desarrollo de videjuegos en HTML5. Nos permite crear rápidamente videojuegos para navegadores.
- cocos2D for Iphone: es un framework de código abierto para crear videojuegos multiplataforma utilizando Xcode y Objective-C. Las plataformas que soporta son: Android, iOS and OSX.

3. Recursos Utilizados

En esta capítulo detallaremos todos los recursos utilizados, tanto software como hardware, para llevar a cabo este proyecto.

3.1 Recursos Software

3.1.1 Gliffy

Gliffy es una alternativa popular y más económica a Microsoft Visio. Se usa principalmente en desarrollo de software, planificación y documentación, procesos empresariales y diagramas organizativos, con énfasis en gráficos funcionales para empresas como diagramas, organigramas y diagramas funcionales.

3.1.2 Eclipse Android Developer Tools Bundle v22.6.2-1085508

Eclipse ADT Bundle incluye todo lo necesario para desarrollar aplicaciones para Android:

- Eclipse + ADT plugin
- Herramientas Android SDK
- Herramientas para la platforma Android.

Eclipse

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU (GNU GPL).

SDK Android

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

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

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

3.1.3 Gimp 2.8.14

GIMP (GNU Image Manipulation Program) es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la Licencia pública general de GNU y GNU Lesser General Public License.

GIMP tiene herramientas que se utilizan para el retoque y edición de imágenes, dibujo de formas libres, cambiar el tamaño, recortar, hacer fotomontajes, convertir a diferentes formatos de imagen, y las tareas más especializadas. Se pueden crear imágenes animadas en formato GIF. En formato MPEG se pueden crear también usando un plugin de animación.

Los desarrolladores y encargados de mantener GIMP se esfuerzan por crear una aplicación gráfica de software libre, de alta calidad para la edición y creación imágenes originales, de fotografías, y también de iconos, elementos gráficos de las páginas web y otros elementos artísticos de interfaz de usuario.

3.1.4 Microsoft Word 2013

Microsoft Word es un software destinado al procesamiento de textos, creado por la empresa Microsoft. Actualmente viene integrado en la suite ofimática Microsoft Office. Originalmente fue desarrollado por Richard Brodie para el computador de IBM bajo sistema operativo DOS en 1983.Versiones subsecuentes fueron programadas para muchas otras plataformas, incluyendo, las computadoras IBM que corrían en MS-DOS (1983). Es un componente de la suite ofimática Microsoft Office; también es vendido de forma independiente e incluido en la Suite de Microsoft Works.

Las versiones actuales son Microsoft Office Word 2013 para Windows y Microsoft Office Word 2011 para Mac. Ha llegado a ser el procesador de texto más popular del mundo.

3.1.5 Windows 8.1 Pro N

Windows 8 es la versión actual del sistema operativo de Microsoft Windows, producido por Microsoft para su uso en computadoras personales, incluidas computadoras de escritorio en casa y de negocios, computadoras portátiles, netbooks, tabletas, servidores y centros multimedia. El principal cambio es la polémica decisión de eliminar Menú Inicio, existente desde Windows 95 como estándar de facto en cómo presentar aplicaciones en interfaces gráficas. El 2

de abril de 2014, Microsoft reconoció el error de la eliminación del menú de inicio y anunció que lo volverían a implementar en la siguiente versión de Windows. Aunque no llegará hasta 2015.

Añade soporte para microprocesadores ARM, además de los microprocesadores tradicionales x86 de Intel y AMD. Su interfaz de usuario ha sido modificada para hacerla más adecuada para su uso con pantallas táctiles, además de los tradicionales ratón y teclado. El efecto Aero Glass no está presente en este sistema operativo, poniendo nuevos efectos planos para ventanas (no App) y botones con un simple color.

El 30 de septiembre de 2014, Microsoft presentó su sucesor, Windows 10, orientado a solucionar el fracaso de Windows 8, recuperando, entre otros, el desaparecido Menú Inicio.

3.1.6 GIT

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

El mantenimiento del software Git está actualmente supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores.

3.1.7 SourceTree 1.6.5

SourceTree es un poderoso cliente de Git y Mercurial para desarrolladores que utilizan Mac o WIndows. Fue desarrollado por Atlassian y lanzado el 26 de octubre de 2010. Cuenta con una interfaz sencilla que nos facilita el manejo del sistema de control de versiones.

3.1.8 Tiled Map Editor 0.9.1

Tiled Map Editor es un editor de mapas tile o cuadriculados, es una herramienta gratuita, flexible y sencilla de utilizar que nos permite generar mapas con formato TMX.

El formato de mapas TMX (Tile Map XML) es una manera flexible de describir un mapa basado en cuadricula. Puede describir mapas con cualquier tamaño de cuadrículas y capas, también permite añadir propiedades personalizadas en la mayoría de elementos.

3.1.9 OS X Mavericks (versión 10.9)

OS X Mavericks (versión 10.9) es la décima versión principal de OS X para ordenadores, portátiles y servidores Mac. OS X Mavericks se anunció el 10 de junio 2013 en la WWDC 2013, y se puede descargar desde el Mac App Store. Esta nueva versión de OS X marca el comienzo de un cambio en el esquema de nombres de OS X, dejando la utilización de los grandes felinos y pasando a nombres basados en lugares en California. Así que esta versión del sistema operativo se ha llamado Mavericks, una localidad de California donde el “surf” es muy popular, lo que hace que el logotipo sea una ola del mar.

3.1.10 Android 4.4.4

Android es un sistema operativo basado en el kernel de Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, y también para relojes inteligentes, televisores y automóviles, inicialmente desarrollado por Android Inc., que Google respaldó económicamente y más tarde compró esta empresa en 2005. Android fue presentado en 2007 junto la fundación del Open Handset Alliance: un consorcio de compañías de hardware, software y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008.

El éxito del sistema operativo se ha convertido en objeto de litigios sobre patentes en el marco de las llamadas «Guerras por patentes de teléfonos inteligentes» entre las empresas de tecnología.

3.2 Recursos Hardware

Durante la realización de este proyecto se utilizaron los siguientes recursos hardware:

- Ordenador sobremesa (Sistema Operativo Windows).
- Ordenador portátil Mac (Sistema Operativo OS X).
- Dispositivo móvil LG Nexus 4 (Sistema Operativo Android).

4. Planificación del trabajo

En este apartado describiremos la metodología de desarrollo que emplearemos durante el proyecto, continuaremos especificando las tareas a realizar, asignándoles un tiempo estimado de ejecución a cada tarea, y terminaremos realizando un presupuesto de lo que supondría llevar a cabo el proyecto.

4.1 Metodología de desarrollo

La metodología de desarrollo que utilizaremos en este proyecto es la metodología PUD. El Proceso Unificado de Desarrollo Software o simplemente Proceso Unificado es un marco de desarrollo de software que se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental. El refinamiento más conocido y documentado del Proceso Unificado es el Proceso Unificado de Rational o simplemente RUP.

El Proceso Unificado no es simplemente un proceso, sino un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos. De la misma forma, el Proceso Unificado de Rational, también es un marco de trabajo extensible, por lo que muchas veces resulta imposible decir si un refinamiento particular del proceso ha sido derivado del Proceso Unificado o del RUP. Por dicho motivo, los dos nombres suelen utilizarse para referirse a un mismo concepto.

El nombre Proceso Unificado se usa para describir el proceso genérico que incluye aquellos elementos que son comunes a la mayoría de los refinamientos existentes. También permite evitar problemas legales ya que Proceso Unificado de Rational o RUP son marcas registradas por IBM (desde su compra de Rational Software Corporation en 2003). El primer libro sobre el tema se denominó, en su versión española, El Proceso Unificado de Desarrollo de Software (ISBN 84-7829-036-2) y fue publicado en 1999 por Ivar Jacobson, Grady Booch y James Rumbaugh, conocidos también por ser los desarrolladores del UML, el Lenguaje Unificado de Modelado. Desde entonces los autores que publican libros sobre el tema y que no están afiliados a Rational utilizan el término Proceso Unificado, mientras que los autores que pertenecen a Rational favorecen el nombre de Proceso Unificado de Rational.

Las principales características que nos encontramos cuando hablamos del Proceso Unificado de Desarrollo son:

Iterativo e Incremental

El Proceso Unificado es un marco de desarrollo iterativo e incremental compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones (la de inicio sólo consta de varias iteraciones en proyectos grandes). Estas iteraciones ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

Cada una de estas iteraciones se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada: Análisis de requisitos, Diseño, Implementación y Prueba. Aunque todas las iteraciones suelen incluir trabajo en casi todas las disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto.

Dirigido por los casos de uso

En el Proceso Unificado los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc.

Centrado en la arquitectura

El Proceso Unificado asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc.

Enfocado en los riesgos

El Proceso Unificado requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida. Los resultados de cada iteración, en especial los de la fase de Elaboración deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero.

4.1.1 Fases del Proceso Unificado de Desarrollo

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Al final de cada uno de ellos se obtiene una versión final del producto, que no sólo satisface ciertos casos de uso, sino que está lista para ser entregada y puesta en producción. En caso de que fuese necesario publicar otra versión, deberían repetirse los mismos pasos a lo largo de otro ciclo.

Cada ciclo se compone de varias fases, y dentro de cada una de ellas, los directores o desarrolladores pueden descomponer adicionalmente el trabajo en iteraciones, con sus incrementos resultantes. Cada fase termina con un hito, determinado por la disponibilidad de un conjunto de artefactos, modelos o documentos.

Las iteraciones de cada fase se desarrollan a través de las actividades de identificación de requisitos, análisis, diseño, implementación, pruebas e integración.

4.1.1.1 Fase de inicio

Suele ser la fase más corta del desarrollo, y no debería alargarse demasiado en el tiempo. En caso contrario, podríamos encontrarnos en una situación de excesiva especificación inicial, yendo en contra del enfoque relativamente ágil del Proceso Unificado.

Durante la fase de inicio se desarrolla una descripción del producto final, y se presenta el análisis del negocio. En esta fase también se identifican y priorizan los riesgos más importantes. Además se establecen las principales funciones del sistema para los usuarios más importantes, la arquitectura a grandes rasgos y un plan de proyecto.

El objetivo de esta fase es ayudar al equipo de proyecto a decidir cuáles son los verdaderos objetivos del proyecto. Las iteraciones exploran diferentes soluciones posibles, y diferentes arquitecturas posibles. Puede que todo el trabajo físico realizado en esta fase sea descartado.

Lo único que normalmente sobrevive a la fase de inicio es el incremento del conocimiento del equipo.

Los artefactos que típicamente sobreviven a esta fase son:

- Un enunciado de los mayores requerimientos planteados generalmente como casos de uso.
- Un boceto inicial de la arquitectura.
- Una descripción de los objetivos del proyecto.
- Una versión muy preliminar del plan de proyecto.
- Un modelo del negocio.

La fase de inicio termina con el hito de los objetivos del desarrollo.

4.1.1.2 Fase de elaboración

Durante esta fase deberían capturarse la mayoría de requisitos del sistema, aunque los objetivos principales son tratar los riesgos ya identificados y establecer y validar la base de la arquitectura del sistema. Esta base se llevará a cabo a través de varias iteraciones, y servirá de punto de partida para la fase de construcción.

Durante la fase de elaboración se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura. Las iteraciones de la fase de elaboración:

- Establecen una firme comprensión del problema a solucionar.
- Establece la fundación arquitectural para el software.
- Establece un plan detallado para las siguientes iteraciones.
- Elimina los mayores riesgos.

El resultado de esta fase es la línea base de la arquitectura. En esta fase se construyen típicamente los siguientes artefactos:

- El cuerpo básico del software en la forma de un prototipo arquitectural.
- Casos de prueba.
- La mayoría de los casos de uso (80%) que describen la funcionalidad del sistema.
- Un plan detallado para las siguientes iteraciones.

La fase de elaboración finaliza con el hito de la Arquitectura del sistema.

4.1.1.3 Fase de construcción

Es la fase más larga del proyecto, y completa la implementación del sistema tomando como base la arquitectura obtenida durante la fase de elaboración. A partir de ella, las distintas funcionalidades son incluidas en distintas iteraciones, al final de cada una de las cuales se obtendrá una nueva versión ejecutable del producto.

Durante la fase de construcción se crea el producto. La línea base de la arquitectura crece hasta convertirse en el sistema completo. Al final de esta fase el producto contiene todos los casos de uso implementados, sin embargo puede que no esté libre de defectos.

Los artefactos producidos en esta fase son:

- El sistema software.
- Los casos de prueba.
- Los manuales de usuario.

Por tanto, esta fase concluye con el hito de obtención de una funcionalidad completa, que capacite al producto para funcionar en un entorno de producción.

4.1.1.4 Fase de transición

En la fase final del proyecto se lleva a cabo el despliegue del producto en el entorno de los usuarios, lo que incluye la formación de éstos.

La fase de transición cubre el período durante el cual el producto se convierte en la versión beta. Las iteraciones en esta fase continúan agregando características al software. Sin embargo, las características se agregan a un sistema que el usuario se encuentra utilizando activamente.

Los artefactos construidos en esta fase son los mismos que en la fase de construcción. El equipo se encuentra ocupado fundamentalmente en corregir y extender la funcionalidad del sistema desarrollado en la fase anterior.

La fase de transición finaliza con el hito de Lanzamiento del Producto.

4.2 Planificación y temporización

Para la ejecución de este proyecto hemos planificado diez actividades, a las cuales les hemos asignado un tiempo estimado para llevarlas a cabo. El total de horas son las trescientas horas asignadas al Trabajo de Fin de Grado.

- Estudio de herramientas y librerías (15h). Estudiaremos las herramientas y librerías que se podrían utilizar para el desarrollo del proyecto.
- Análisis de alternativas (5h). En base al estudio anterior elegiremos que herramientas y librerías vamos a utilizar.
- Elaboración de un prototipo (Prueba de concepto) (25h). Demostración rápida de que es posible realizar el proyecto utilizando las herramientas elegidas, con el objetivo de detectar posibles problemas.
- Elaboración Lista de Características (15h). Crearemos un listado con las características que nos gustaría tener en el producto final. Además las priorizaremos.
- Elaboración de una arquitectura base (20h). Diseñaremos una arquitectura base teniendo en cuenta el conjunto de características obtenidas en la actividad anterior. Se podrá modificar a medida que el desarrollo avance.
- Especificación Casos de Uso (20h). Una vez elegida la característica más prioritaria del listado, especificamos los casos de uso necesarios para desarrollarla.
- Desarrollo de Casos de Uso (130h). Implementación de los casos de uso especificados.
- Probar el software (10h). Pruebas a realizar para comprobar que los casos de uso han sido implementados correctamente.
- Análisis de los resultados (15h). Una vez concluido el proyecto analizaremos el resultado obtenido y evaluaremos el grado de satisfacción con el mismo.

- Memoria final (45h). Redacción de la memoria del proyecto.

4.3 Presupuesto

En este aparatado crearemos un presupuesto para el desarrollo de este proyecto. En el presupuesto incluiremos los gastos derivados del personal que participara en él y también los gastos de material hardware y software necesarios para el desarrollo.

Para poder hacer el cálculo necesitamos realizar algunas suposiciones y estimaciones. Considerando que este año (2014) tenemos 10 días festivos que caen entre lunes y viernes, y que el año cuenta con aproximadamente 52 semanas:

$$\left((52 * \ 5\ días) - 10\ días\ \right) * 8\ horas = 2000\ horas$$

Gracias a este cálculo estimamos que cada año tiene 2000 horas laborables. Los salarios que asignaremos a nuestro personal son los siguientes:

- Alumno: 17.500 € Brutos Anuales
- Profesor: 30.000 € Brutos Anuales

Con estos dos datos podemos calcular el precio por hora de nuestro personal:

- Alumno: 8,75 €/h
- Profesor: 15 €/h

Para realizar el proyecto es necesaria una dedicación de 300 horas por parte del alumno. Por otro lado el tutor se reunirá con el alumno periódicamente y ayudará al alumno cuando lo necesite, debido a esto se estima que el profesor dedicara un 20 % del tiempo total del proyecto, resultando así en unas 60 horas.

Multiplicando los valores obtenidos conseguimos el coste total de personal para el proyecto:

- Alumno: $300\ horas * 8{,}75\ €/hora\ = 2625€$
- Profesor: $60\ horas * 15\ €/hora\ = 900€$
- **Coste total personal: 3525 €**

El coste del hardware utilizado se detalla a continuación:

- Ordenador sobremesa + Pantalla: 1300 €
- Ordenador portátil Mac: 1200 €
- Dispositivo móvil LG Nexus 4: 300€
- **Coste total hardware: 2800 €**

El coste de las licencias del software utilizado sería el siguiente:

- Windows 8.1 Pro: 279 €
- Office Hogar y Empresas 2013: 269 €
- **Coste total software: 548 €**

Sumando todos los totales obtenemos el coste total del proyecto: **6873 €**

5. Desarrollo del trabajo

En este capítulo describiremos las distintas fases del desarrollo del proyecto e incluiremos los artefactos obtenidos en ellas. Abarcaremos desde el análisis y la recopilación de requisitos hasta la implementación y pruebas del sistema, dejando para el último capítulo el análisis de los resultados y las conclusiones.

Empezaremos identificando los requisitos del sistema y del software, a partir de los requisitos podremos definir y delimitar el ámbito del proyecto. Para conseguir esto utilizaremos el modelo del dominio, la lista de características y definiremos los casos de uso.

Con el modelo del dominio pretendemos capturar y expresar el entendimiento del sistema en general. La lista de características nos servirá para recopilar todas las funcionalidades que consideremos interesantes y que podrían ser incluidas en el sistema. Finalmente, la definición de los casos de uso nos ayudará a perfilar de forma más concreta las funciones que se esperan del software.

5.1 Requisitos del sistema

Empezaremos estableciendo el ámbito en el que se moverá la aplicación, para ello utilizaremos dos artefactos muy importantes: la lista de características y el modelo del dominio. Además estos artefactos servirán para que los desarrolladores obtengan una mejor comprensión del contexto del sistema y para recopilar requisitos funcionales y no funcionales.

5.1.1 Modelo del dominio

El modelo del dominio muestra clases conceptuales significativas en un problema; es el artefacto más importante que se crea durante el análisis orientado a objetos y se utiliza con frecuencia como fuente de inspiración para el diseño de los objetos software.

Un modelo del dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. Utilizando la notación UML, un modelo del dominio se representa con un conjunto de diagramas. Pueden mostrar:

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

Diagramas

Tras analizar el contexto de la aplicación hemos conseguido capturar el dominio de interés en dos diagramas. A continuación les presentamos el primer diagrama en el que mostramos una visión global de la aplicación.

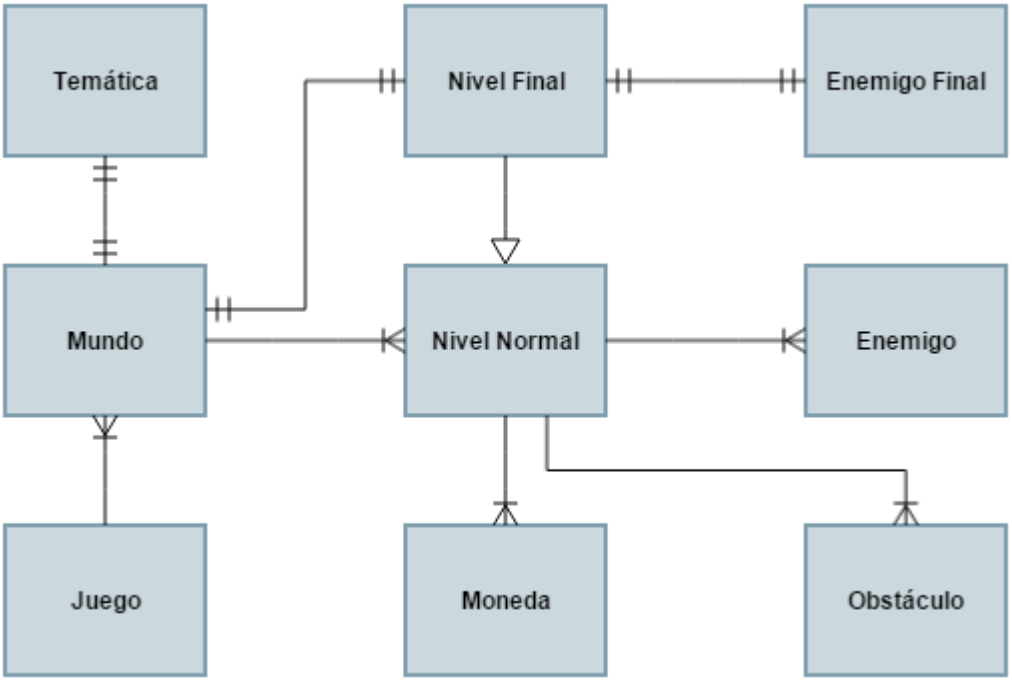


Ilustración 8: Modelo del dominio - Vista general

Definiciones

- Juego: entendemos por juego al software que pretendemos desarrollar.
- Temática: la temática es, básicamente, el estilo artístico de un mundo, sin embargo, también entendemos por temática a las características especiales del mismo. Por ejemplo, un mundo podría tener un estilo artístico invernal, en el que predominase el hielo y en el que el jugador resbalase más de lo habitual debido a ello.
- Mundo: definimos un mundo como un conjunto de niveles con una temática en común. En concreto tendremos diez niveles por mundo, los primeros nueve niveles de ese mundo serán niveles normales y el décimo nivel será un nivel final.
- Nivel Normal: un nivel normal es una de las fases o pantallas que debe superar el jugador en un mundo. En él el jugador se encontrará con enemigos, obstáculos y monedas.
- Moneda: una moneda es un objeto recolectable que nos proporciona puntos. El jugador al pasar por encima de las monedas con el personaje las recolecta y se le suman los puntos.
- Obstáculo: definimos un obstáculo como cualquier entidad inanimada con capacidad para matar al personaje que controla el jugador. Por ejemplo: un pozo de magma, al tocarlo el personaje muere y tiene que volver a empezar.
- Enemigo: un enemigo es una entidad animada con capacidad para matar al personaje del jugador. Puede tener capacidad de disparo, movimiento, salto, etc. Al derrotarlos el jugador obtiene puntos.
- Nivel Final: un nivel final es básicamente lo mismo que un nivel normal pero con la peculiaridad de tener un enemigo final al que hay que derrotar.
- Enemigo Final: un enemigo final es básicamente lo mismo que un enemigo pero con una mayor dificultad para derrotarlo.

El segundo diagrama que hemos creado se centra en la interacción del jugador con los niveles:

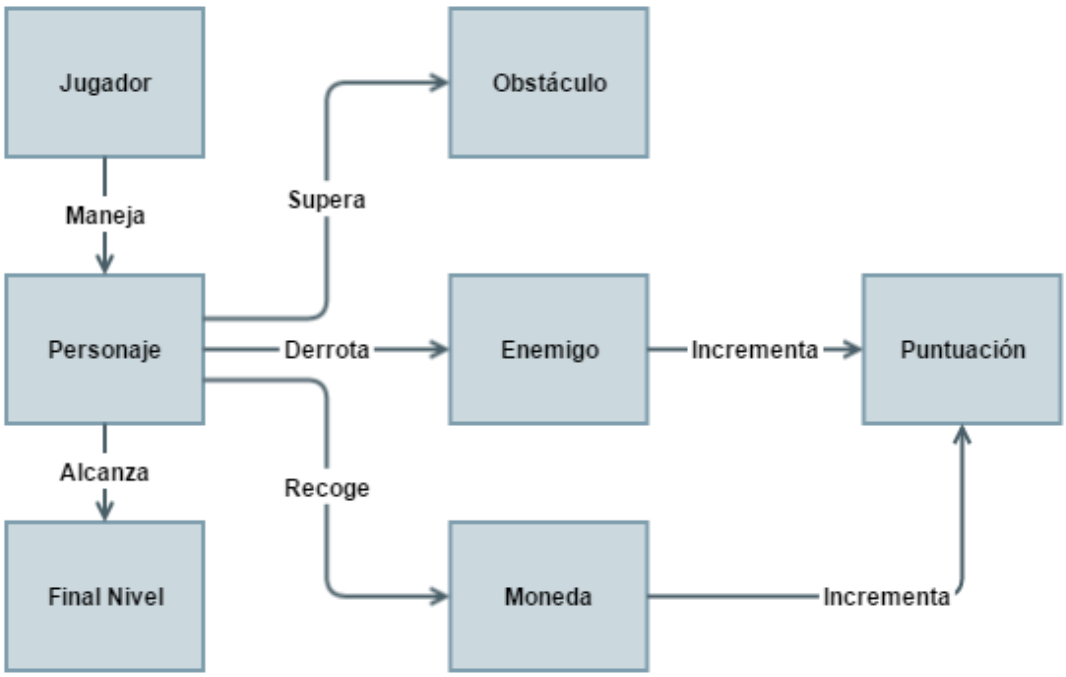


Ilustración 9: Modelo del dominio - Nivel

Definiciones

- Jugador: es la persona que interactúa con nuestro software con la finalidad de Jugar.
- Personaje: llamamos personaje al ente humanoide que es controlado por el jugador.
- Final Nivel: El final del nivel es la ubicación dentro del nivel que da por finalizado el nivel actual, al llegar a esta ubicación pasamos al siguiente nivel.
- Puntuación: es el número de puntos que ha obtenido el Jugador en el Nivel actual.

5.1.2 Lista de características

En la lista de las características incluimos todas las ideas deseables desde una perspectiva de las necesidades del cliente. Se pueden desarrollar para la versión actual del sistema o postergarse para versiones futuras.

Utilizamos la lista de características para gestionar el proyecto y tener una idea general de la planificación del mismo. A medida que el desarrollo avance esta lista puede ser modificada o ampliada.

Para nuestro proyecto hemos agrupado las características y funcionalidades en seis categorías a las que les hemos asignado una letra:

- I -- Interfaz de Usuario
- J -- Jugabilidad
- T – Tienda
- G -- Gráficos
- S -- Sonido
- C -- Conectividad

A continuación les presentamos la lista de características:

Código	Nombre	Descripción	Prioridad
I.1	Menú principal	Es el primer elemento interactivo que se encuentra el usuario, en él se debe poder acceder a los distintos modos de juego, rankings, tienda, opciones y salir	Alta
I.2	Ranking	Listado de las mejores puntuaciones para un nivel determinado	Media
I.3	Opciones	Parámetros configurables del juego, tales como, sonido y música	Baja
I.4	Menú Pausa	Salir al menú principal, continuar, etc.	Alta
J.1	Cronómetro	Tiempo que llevamos en el nivel actual	Alta
J.2	Puntuación	Puntuación acumulada en el nivel actual, calculada en base al número de enemigos derrotados y los objetos recogidos.	Alta
J.3	Registro de Usuarios	Cada usuario queda registrado en el sistema para llevar un control de sus puntuaciones, dinero virtual y objetos adquiridos	Baja
J.4	Modo Un Jugador	Modalidad del juego en la que podemos jugar solos	Alta
J.5	Niveles	Tenemos una serie de niveles, agrupados en mundos, en los que iremos avanzando. Cada modo de juego tendrá asociado su conjunto de mundos	Alta
J.6	Mundos	Conjunto de niveles con una temática determinada. Al finalizar uno se desbloquea el siguiente.	Alta
J.7	Sistema de salud	El personaje controlado por el usuario tendrá asociada una salud si la salud llega a cero el personaje muere.	Media
J.8	Mejoras	A lo largo del nivel encontraremos objetos que modificarán las características de nuestro personaje, por ejemplo, mayor velocidad, invulnerabilidad, etc.	Media
J.9	Enemigos	En los niveles nos encontraremos con distintos tipos de enemigos los cuales contarán con características diferentes.	Alta
J.10	Jefes Finales	En el nivel final de cada mundo nos enfrentaremos a un enemigo con una dificultad superior al resto de enemigos de ese mundo.	Alta
J.11	Obstáculos	A lo largo de los niveles nos encontraremos con pequeñas dificultades para avanzar, por ejemplo, caídas al vacío, trampas, etc.	Alta
J.12	Modo Cooperativo	Modalidad del juego en la que podemos jugar con un compañero.	Baja
J.13	Modo Jugador contra Jugador	Modalidad de juego en la que podremos enfrentarnos con otro jugador.	Baja
G.1	Presentación del Juego	Primera pantalla que visualiza el usuario, resume visualmente el estilo del juego	Media
G.2	Introducción Historia	Cinemática en la que se explica la historia de nuestro personaje	Baja

Código	Nombre	Descripción	Prioridad
G.3	Presentación Jefes Finales	Cinemática en la que se presenta a cada Jefe Final	Baja
G.4	Final Historia	Cinemática final de la historia	Baja
G.5	Estilo	Gráficos del juego con estilo Pixel Art	Alta
G.6	Créditos	Créditos al final del modo historia	Baja
S.1	Efectos	Sonidos de disparos, explosiones, saltos, recogida de objetos, etc.	Alta
S.2	Música	Música por nivel	Alta
T.1	Moneda virtual	Se obtiene jugando al juego en cualquiera de los modos	Media
T.2	Personajes desbloqueables	Disponibles en la tienda	Baja
T.3	Microtransacciones	Se pueden comprar monedas virtuales por dinero real	Baja
T.4	Mundos extras	Disponibles en la tienda	Baja
C.1	Servidor de puntuaciones	Servidor en el que se almacenan las puntuaciones de los jugadores para poder mostrarse en el Ranking	Baja
C.2	Coop. Bluetooth	La conexión entre los jugadores en el modo cooperativo se lleva a cabo mediante Bluetooth	Baja
C.3	Coop. Wifi LAN	La conexión entre los jugadores en el modo cooperativo se lleva a cabo mediante Wifi en Red Local	Baja
C.4	Coop. Internet	La conexión entre los jugadores en el modo cooperativo se lleva a cabo mediante a través de Internet	Baja
C.5	Servidor de mundos	Los mundos desbloqueables se almacenan en un servidor para ser enviados a los clientes.	Baja
C.6	Servidor Tienda		Baja

5.2 Requisitos del software

Hasta ahora hemos presentado y analizado la aplicación, utilizando el modelo del dominio y además hemos recopilado las funcionalidades que nos gustaría incluir en el listado de características. Es el momento de pasar a identificar los actores que utilizarán la aplicación y definir los casos de uso.

5.2.1 Actores

Se le llama actor a toda entidad externa al sistema que guarda una relación con éste y que le demanda una funcionalidad. Esto incluye a los operadores humanos pero también incluye a todos los sistemas externos. Para nuestra aplicación hemos identificado los siguientes actores:

- Jugador: llamamos jugador al usuario que utiliza el software con el objetivo principal de jugar.
- Administrador: el administrador del sistema es aquel que se encargará de administrar y controlar el sistema.

5.2.2 Modelo de Casos de Uso

En el contexto de ingeniería del software, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor

principal sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema.

Para los casos de uso comunes hemos creado un actor abstracto del que heredan los demás con el fin de evitar repetir información:

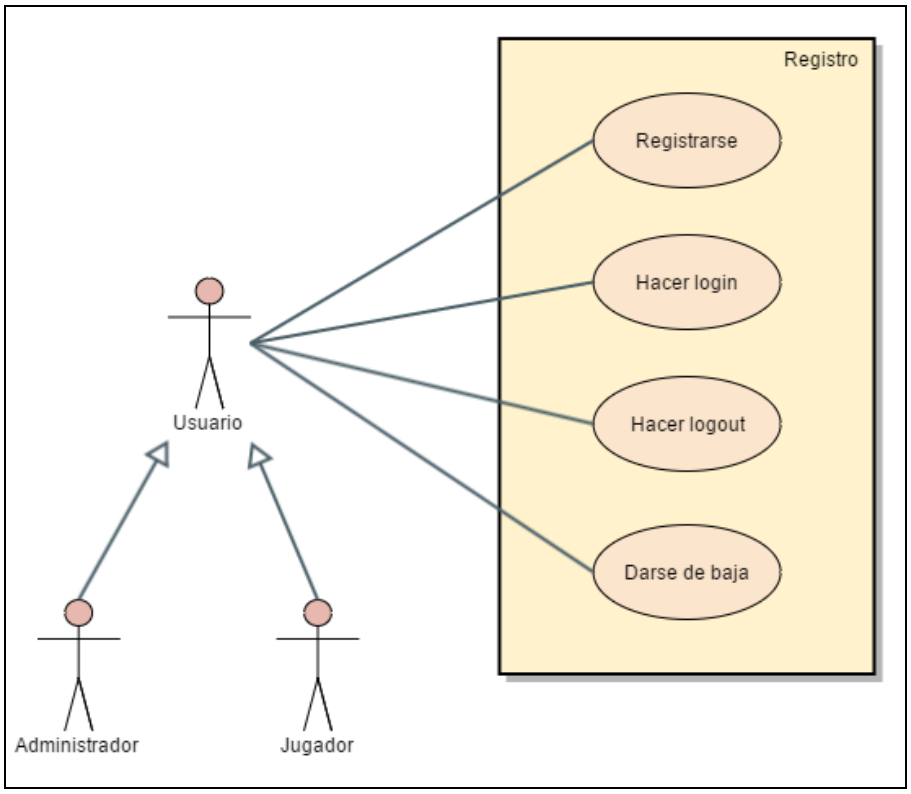


Ilustración 10: Casos de Uso - Usuario Abstracto

Los casos de uso del actor Jugador a nivel general son:

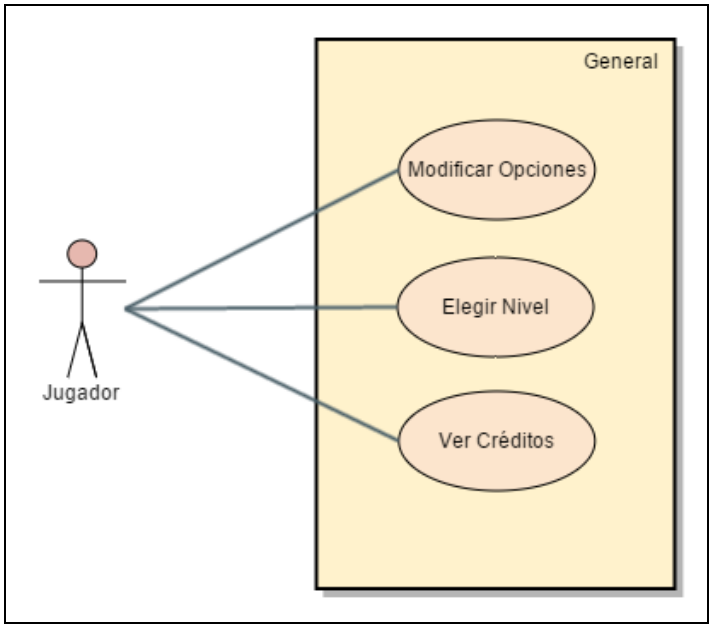


Ilustración 11: Casos de Uso - Jugador (General)

Por otra parte los casos de uso del jugador relacionados con la acción de Jugar y las partidas son:

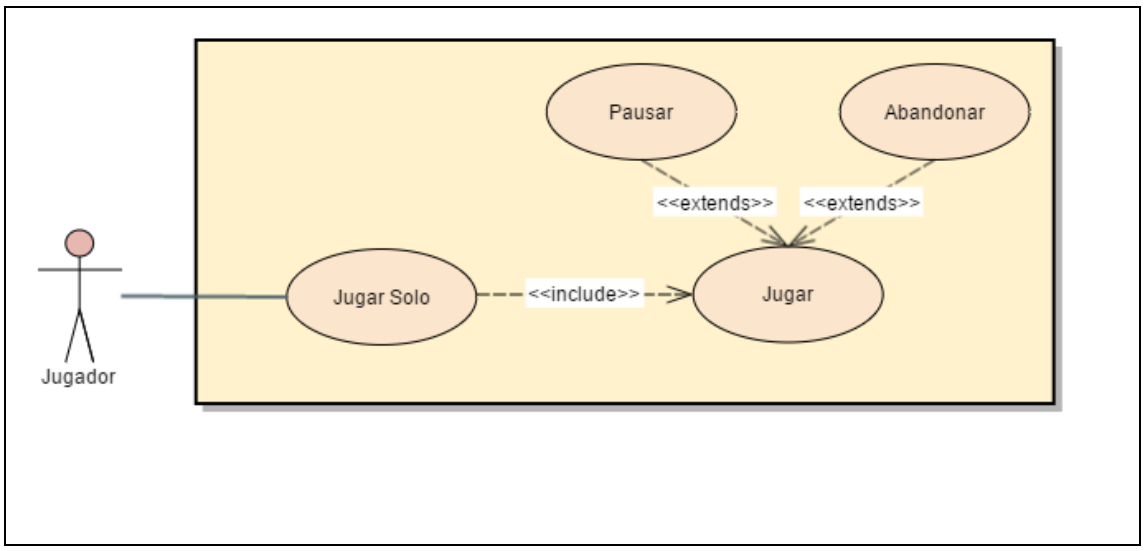


Ilustración 12: Casos de Uso - Jugador (Juego)

A partir de este listado seleccionamos los casos de uso más prioritarios, nos servirán para asentar las bases del sistema, los elegidos son:

- Jugar Solo
- Elegir Nivel
- Pausar
- Abandonar

5.2.3 Especificación de casos de uso

Caso de Uso: Elegir Mundo

Actores	Jugador
Precondición	-
Postcondición	Mundo Seleccionado
Camino Normal	**Camino Alternativo**
1. Se le presenta al usuario la lista de mundos	
2. El usuario elige el mundo que desea seleccionar	2.1.a El usuario decide abandonar la lista de mundos 2.1.b Volver al menú principal 2.2.a El usuario elige un mundo bloqueado 2.2.b El sistema ignora el evento de selección
3. El sistema inicia el caso de uso “Elegir Nivel”	
4. Fin	

Caso de Uso: Elegir Nivel

Actores	Jugador
Precondición	Mundo Seleccionado
Postcondición	Nivel Seleccionado
Camino Normal	**Camino Alternativo**
1. Se le presenta al usuario la lista de Niveles para el Mundo Seleccionado	
2. El usuario elige el nivel que quiere jugar	2.1 El usuario elige un nivel bloqueado 2.2 El sistema ignora el evento de selección
3. Se ejecuta el caso de uso “Jugar Solo”	
4. Fin	

Caso de Uso: Jugar Solo

Actores	Jugador
Precondición	Nivel Seleccionado
Postcondición	
Camino Normal	**Camino Alternativo**

1. Se le presenta al usuario la pantalla de Juego	
2. El usuario juega al nivel seleccionado	2.1.a El usuario decide pausar la partida 2.1.b Pasamos al caso de uso “Pausar Juego” 2.2.a El personaje controlado por el jugador muere 2.2.b El personaje reaparece al inicio del nivel 2.2.c Continuamos en el paso 2
3. El usuario consigue superar el nivel	
4. El siguiente nivel se desbloquea	4.1.a En caso de no haber siguiente nivel, se desbloquea el primer nivel del siguiente mundo 4.2.a En caso de no haber siguiente nivel, ni tampoco siguiente mundo, se vuelve al menú principal 4.2.b Fin
5. El sistema muestra el resumen de la puntuación obtenida en el nivel	
6. El usuario elige jugar el siguiente nivel	6.1.a El usuario elige volver a Jugar el mismo nivel para mejorar su puntuación 6.1.b El personaje vuelve a su posición inicial 6.1.c Continuamos en el paso 2 6.2.a El usuario elige dejar de jugar 6.2.b Volver al menú principal 6.2.c El usuario sale de la aplicación 6.2.d Antes de cerrar la aplicación se guarda el avance del Jugador (Niveles y Mundos desbloqueados) 6.2.e Fin
7. Volvemos al paso 2	

Caso de Uso: Pausar Juego

Actores	Jugador
Precondición	Caso de uso Jugar iniciado
Postcondición	Menú de pausa en pantalla
Camino Normal	**Camino Alternativo**

1. El sistema pausa el juego y muestra el menú de pausa	
2. El usuario elige seguir jugando	2.1 El usuario elige abandonar la partida 2.2 Se ejecuta el caso de uso “Abandonar Juego”
3. Se vuelve al caso de uso “Jugar”	

Caso de Uso: Abandonar Juego

Actores	Jugador
Precondición	Menú de pausa en pantalla
Postcondición	-
Camino Normal	**Camino Alternativo**
1. El sistema abandona la partida y se muestra el menú principal 2. Fin	

Caso de Uso: Ver Opciones

Actores	Jugador
Precondición	-
Postcondición	-
Camino Normal	**Camino Alternativo**
1. El usuario elige “Opciones” en el menú	
2. Se le presentan al usuario las opciones disponibles	
3. El usuario modifica los parámetros	
4. El usuario pulsa salir	
5. Fin	

Caso de Uso: Ver Créditos

Actores	Jugador
Precondición	Menú de pausa en pantalla
Postcondición	
Camino Normal	**Camino Alternativo**

1. El usuario elige “Créditos” en el menú	
2. Se le presentan al usuario los créditos	
3. El usuario pulsa salir	
4. Fin	

5.2.4 Prototipo de Interfaz de Usuario

Una vez definidos los casos de uso necesitamos hacer un prototipo de las vistas que presentaremos al usuario. Los prototipos se muestran a continuación:

Ilustración 13: Prototipo menú

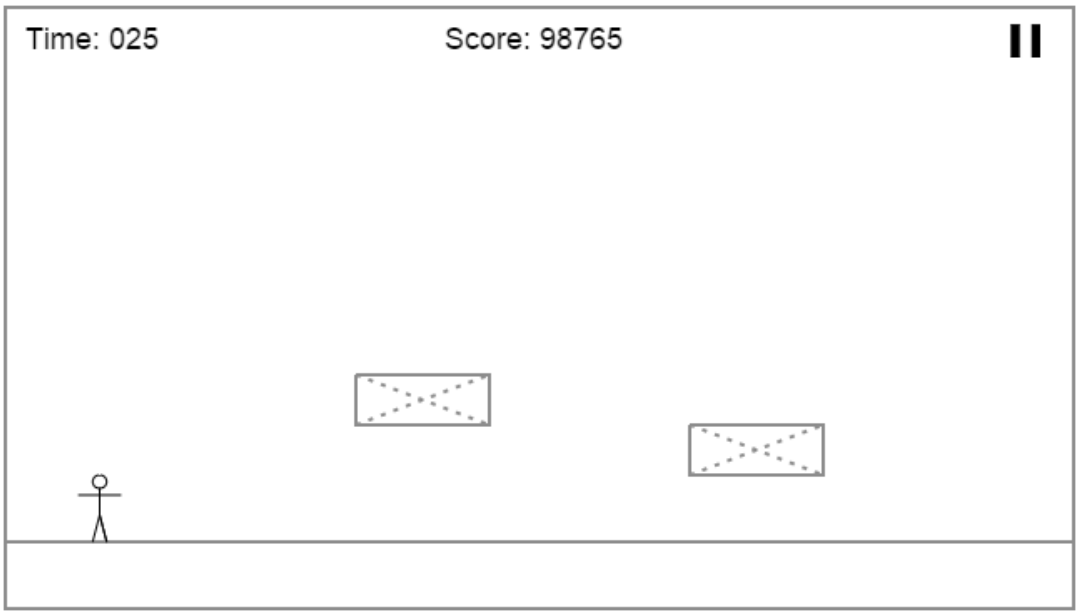


Ilustración 14: Prototipo interfaz PC

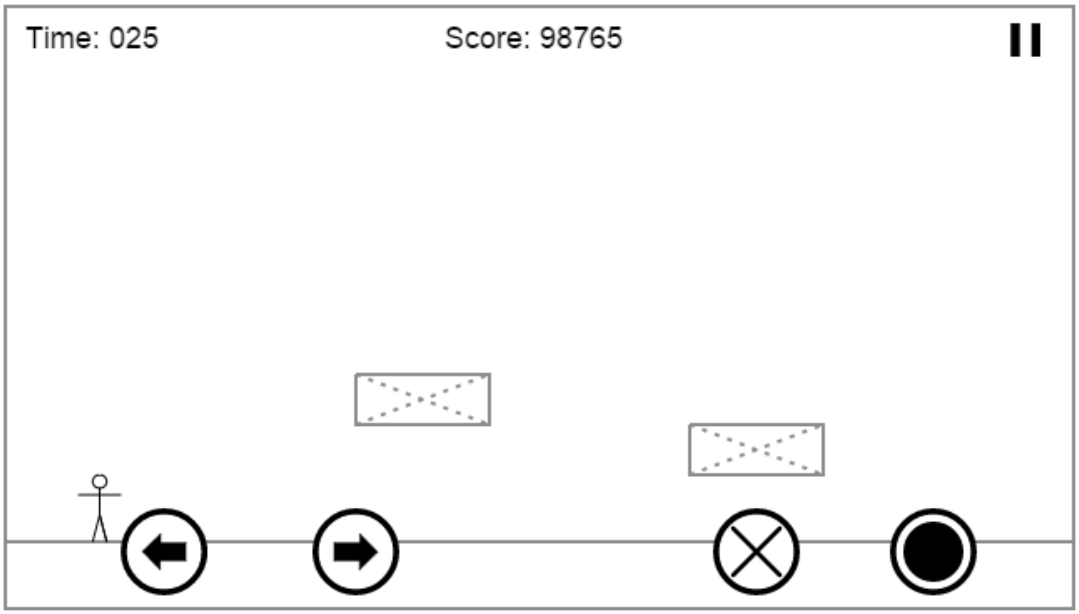


Ilustración 15: Prototipo interfaz móvil

En el primer prototipo mostramos cómo será el menú principal del programa, Tenemos el título del juego en grande seguido de una serie de botones para acceder a las distintas acciones:

- Jugar Solo
- Jugar Cooperativo
- Opciones
- Créditos

En la segunda y tercera imagen esbozamos cómo será la interfaz que verá el usuario al jugar al juego. La segunda imagen nos muestra la interfaz al ejecutarse en un PC, mientras que la tercera imagen nos muestra la interfaz al ejecutarse en un dispositivo móvil.

En ambas imágenes tenemos en la parte superior dos contadores, el primero, a la derecha, nos indica el tiempo que llevamos en el nivel actual, y el segundo, centrado, nos indica la puntuación acumulada en el nivel actual. También podemos ver un botón, arriba a la derecha, que nos permite pausar el juego.

La interfaz móvil se diferencia de la interfaz para PC en los controles mostrados en pantalla, al no contar con teclado físico tenemos que mostrar botones en la propia pantalla que el jugador pueda pulsarlos e interactuar con el juego.

5.3 Modelo de análisis

En este apartado presentaremos los resultados del análisis, para llevar a cabo el análisis nos hemos basado en los casos de uso especificados. Primero organizaremos la aplicación en distintos paquetes y obtendremos una visión general de la misma. Una vez hecho esto, crearemos diagramas de clases para cada caso de uso especificado y así veremos los elementos estructurales que intervienen en ellos.

5.3.1 Organización en Paquetes

Hemos decidido organizar el modelo de análisis en tres paquetes que presentamos a continuación:

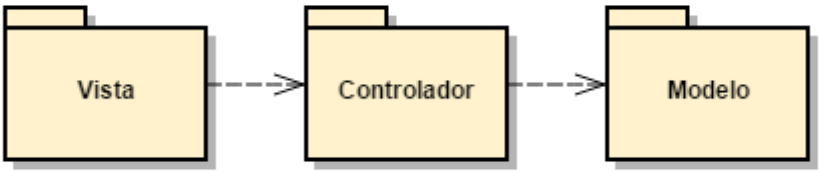


Ilustración 16: Paquetes del sistema

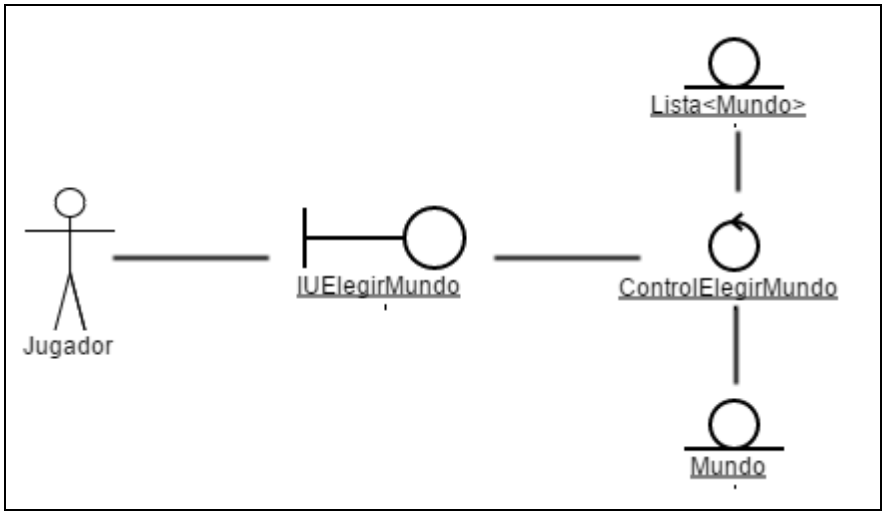
En el paquete de "Vista" se agruparan todas las clases que se encarguen de mostrar información al usuario y de interactuar con él, en el paquete "Controlador" tendremos toda la lógica de la aplicación y se encargará de mediar entre el "Modelo" y la "Vista", y por último en el paquete "Modelo" tendremos las clases que gestionan el contenido, la información del sistema.

5.3.2 Diagramas de clases

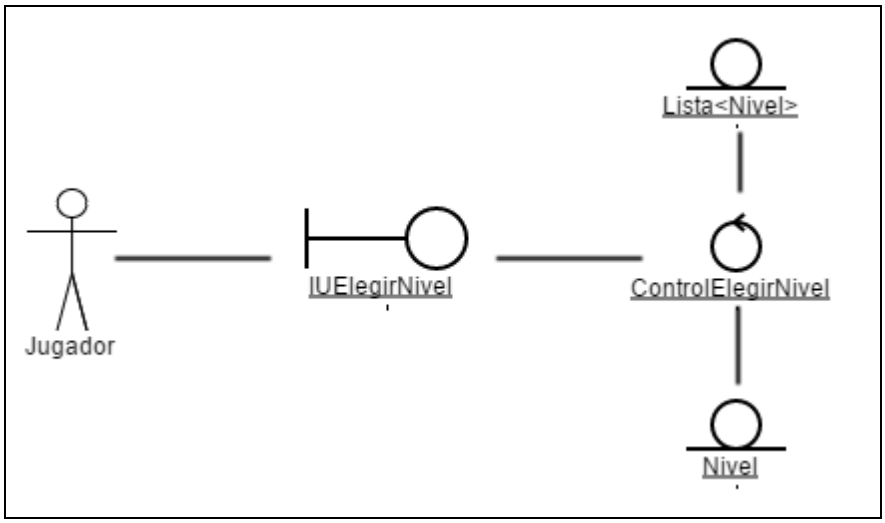
Para modelar las clases de análisis utilizaremos diagramas de clases estáticos. Utilizaremos las tres clases típicas que se utilizan en estos tipos de diagrama:

- Interfaz: Las clases de interfaz se utilizan para modelar la interacción entre el sistema y los actores. Las clases de interfaz modelan las partes del sistema que dependen de sus actores, lo cual implica que clasifican y reúnen los requisitos en los límites del sistema. Las clases de interfaz representan a menudo abstracciones de ventanas, formularios, paneles, interfaces de comunicaciones.
- Control: Las clases de control representan coordinación, secuencia, transacciones y control de otros objetos y se usan con frecuencia para encapsular el control de un caso de uso en concreto. Los aspectos dinámicos del sistema se modelan con clases de control, debido a que ellas manejan y coordinan las acciones y los flujos de control principales, y delegan trabajo a otros objetos.
- Entidad: Las clases de entidad modelan información y el comportamiento asociado de algún fenómeno o concepto, como persona o un objeto. Las clases de entidad reflejan la información de un modo que beneficia a los desarrolladores al diseñar e implementar el sistema, incluyendo su soporte de persistencia. Las clases de entidad suelen mostrar una estructura de datos lógica y contribuyen a comprender de qué información depende el sistema.

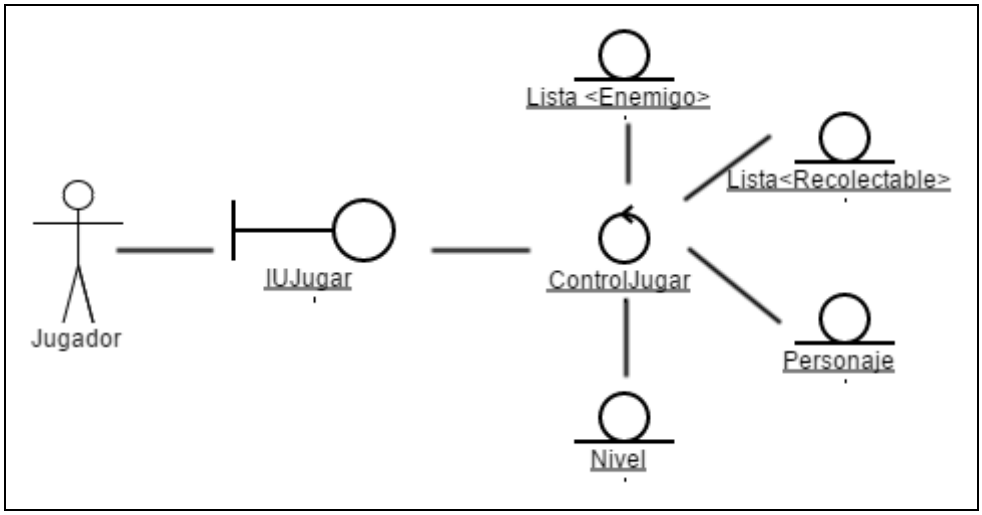
5.3.2.1 Elegir Mundo



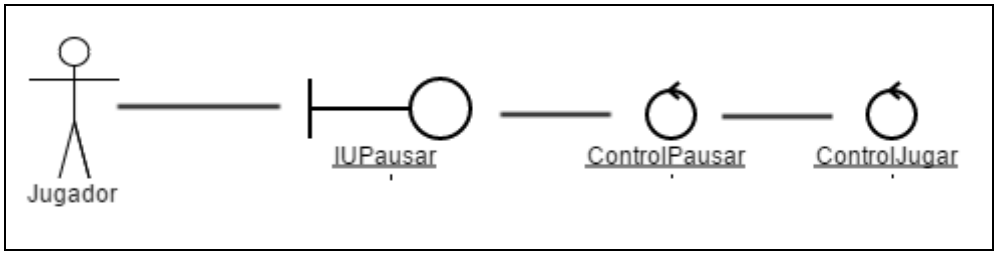
5.3.2.2 Elegir Nivel



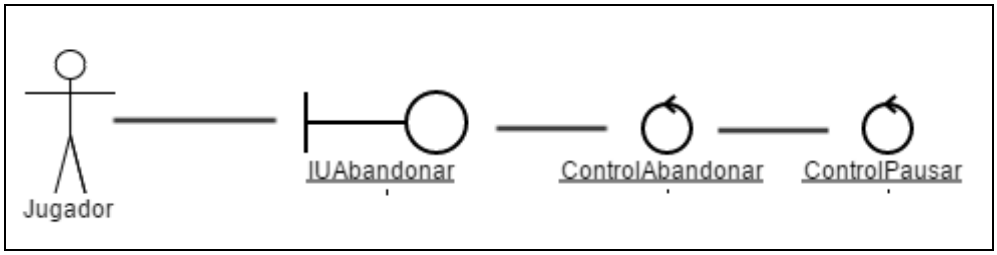
5.3.2.3 Jugar Solo



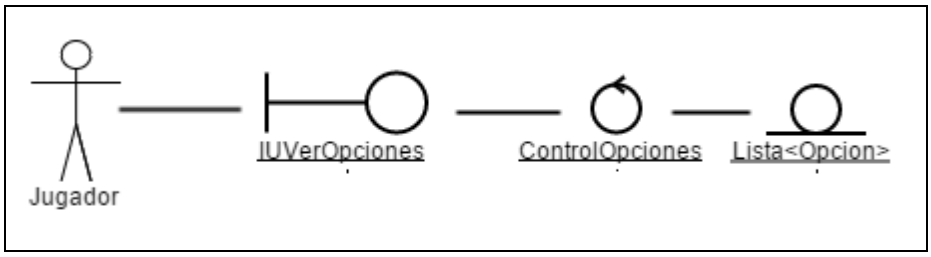
5.3.2.4 Pausar Juego



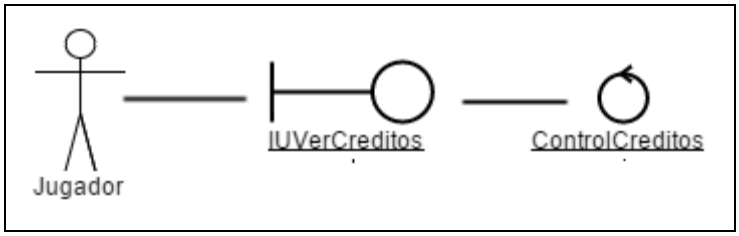
5.3.2.5 Abandonar Juego



5.3.2.4 Ver Opciones



5.3.2.5 Ver Créditos



5.4 Modelo de diseño

5.4.1 Arquitectura

La arquitectura que hemos elegido para desarrollar este proyecto es la arquitectura modelo-vista-controlador (MVC).

El modelo–vista–controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

- El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la "vista" aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al "modelo" a través del "controlador".
- El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al "modelo" cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su "vista" asociada si se solicita un cambio en la forma en que se presenta de "modelo", por tanto se podría decir que el "controlador" hace de intermediario entre la "vista" y el "modelo".
- La Vista: Presenta el "modelo" (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho "modelo" la información que debe representar como salida.

Una vez generado el software utilizando la arquitectura elegida, crearemos un ejecutable por plataforma, es decir, un ejecutable para PC y Mac (.jar), y otro ejecutable para dispositivos Android (.apk). Estos ejecutables consistirán en un envoltorio para nuestra aplicación, como se puede observar en el siguiente diagrama:

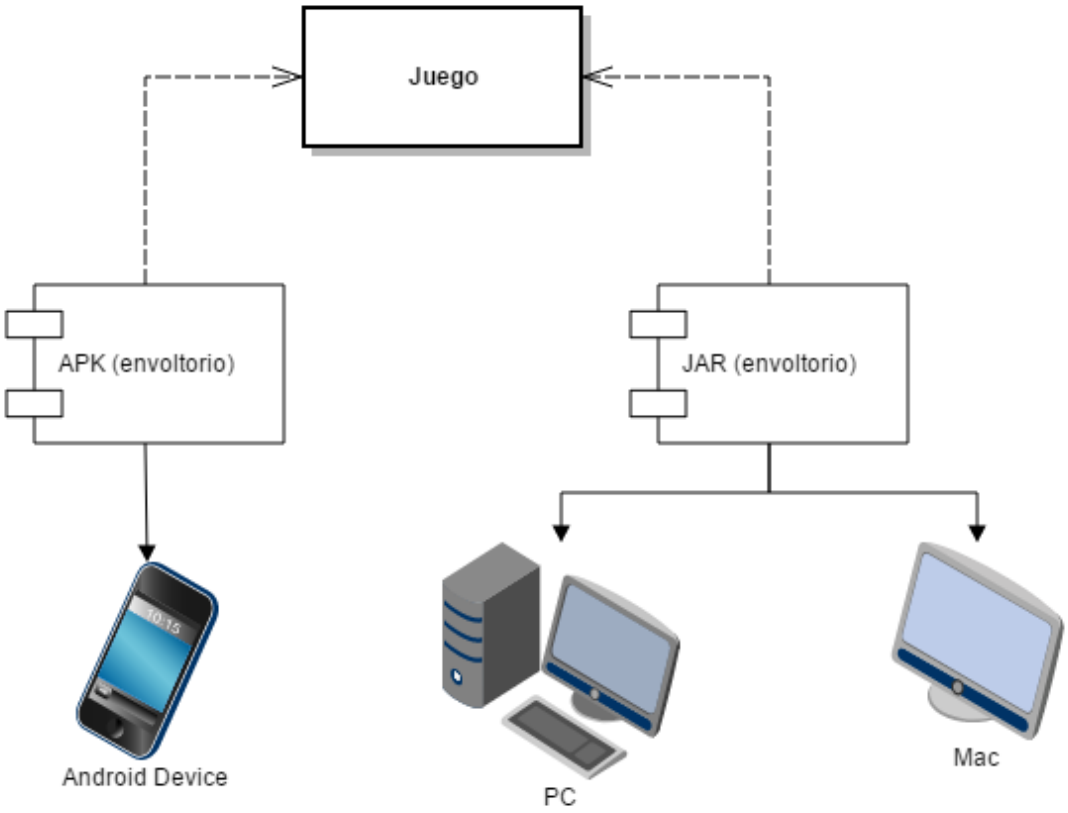


Ilustración 17: Despliegue

Gracias a esto conseguiremos, como se ha comentado al inicio del documento, uno de los objetivos de este proyecto que es conseguir un software capaz de **ejecutarse en múltiples plataformas**.

Para conseguir desarrollar esto, hemos elegido como **lenguaje de programación Java**, debido a que es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

Para facilitar el proceso de desarrollo del videojuego, utilizaremos la **librería especializada libGDX**. libGDX es un framework para el desarrollo de videojuegos multiplataforma, soportando actualmente Windows, Linux, Mac OS X, Android, iOS y HTML5. Uno de los objetivos principales de la librería es mantener la simplicidad, sin renunciar al amplio abanico de plataformas finales. Para ello, permite escribir código en un único proyecto y exportarlo a las tecnologías mencionadas anteriormente sin modificar nada. Pudiendo utilizar la versión de escritorio como entorno de pruebas para el resto, siguiendo así una iteración de desarrollo rápida e integrable con el resto de herramientas de Java.

5.4.2 Diagramas de clases

En este apartado repasaremos las clases que conforman nuestra aplicación, no entraremos en demasiado detalle ya que en el apartado de implementación nos centraremos más en los detalles técnicos. Pretendemos obtener una visión general de la organización del software.

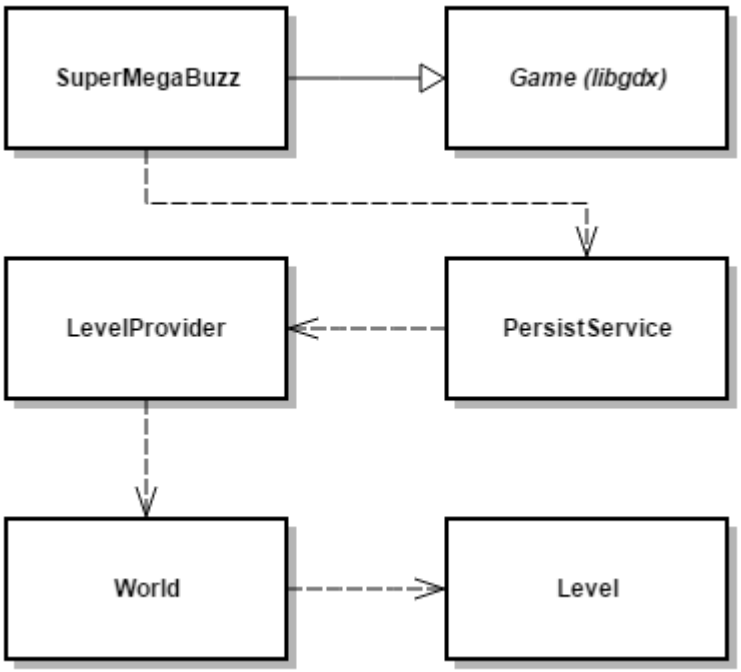


Ilustración 18: Clases Aplicación

En el diagrama superior podemos observar las siguientes clases:

- SuperMegaBuzz: la clase principal de nuestro videojuego, extiende la clase Game (proporcionada por la librería libGDX) y se encarga de proporcionar la base para que toda la aplicación funcione, es nuestro controlador principal. Es la responsable del manejo de las pantallas y de la navegación entre las mismas. Además se encarga de realizar las llamadas necesarias para que el avance del jugador se guarde cuando la aplicación se cierra.
- Game (libGDX): Implementa la interfaz ApplicationListener y a su vez delega las responsabilidades del ApplicationListener a un objeto Screen, permite que una aplicación tenga múltiples pantallas. Una clase que implementa la interfaz ApplicationListener es invocada cuando la aplicación se crea, se resume, se renderiza, se pausa o se destruye. La interfaz ApplicationListener sigue el ciclo de vida estándar de Android y este ciclo de vida se emula en el escritorio apropiadamente.
- PersistService: es la clase encargada de utilizar el sistema de ficheros para recuperar y guardar el estado de la aplicación al iniciar y cerrar el juego respectivamente. El estado se almacena en un fichero como una representación en texto de la clase LevelProvider.
- LevelProvider: nos proporciona la lista de mundos (Clase World) disponibles.
- World: entidad que representa un mundo, contiene una lista de diez niveles.

- Level: entidad que representa un nivel, en ella se almacena el mapa del nivel, la gravedad del mismo, la lista de enemigos y de objetos recolectables que contiene, entre otras propiedades.

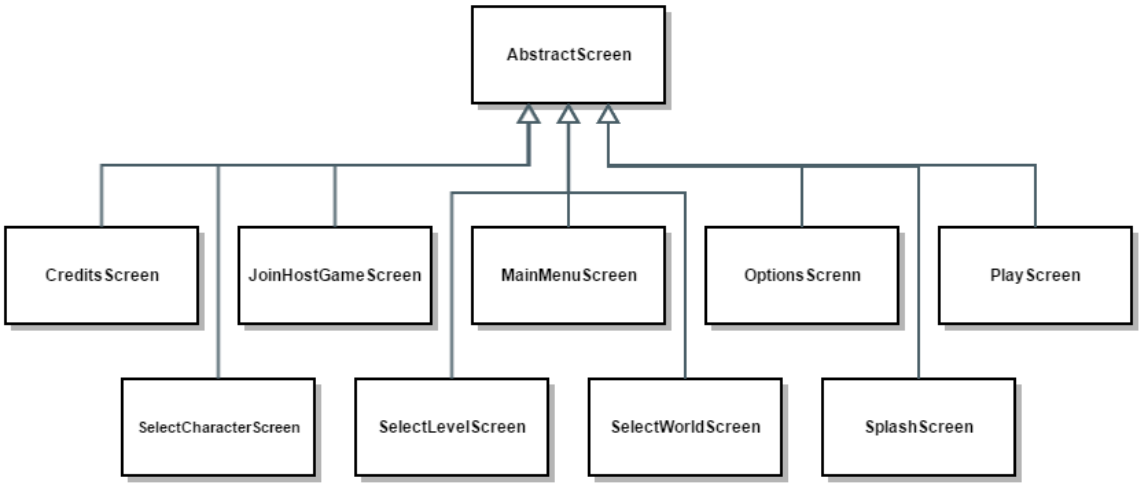


Ilustración 19: Pantallas

Ahora pasamos a ver la organización de las distintas pantallas de la aplicación:

- AbstractScreen: clase de la que heredan todas las demás pantallas de nuestra aplicación, implementa la interfaz Screen y proporciona una base para que las demás pantallas puedan funcionar.
- PlayScreen: en ella se desarrolla el caso de uso Jugar, por tanto es la más compleja de todas las pantallas, utiliza varias tablas para representar los elementos de la interfaz de usuario.
- SplashScreen: primera pantalla que se encuentra el usuario en la que vemos una representación artística del estilo del videojuego, es decir, una pantalla de presentación.
- Resto de clases Screen: representa uno de los menús de nuestra aplicación, utilizan una interfaz organizada utilizando la clase Table (libGDX).

Para poder entender un poco mejor la aplicación, veamos con mayor detenimiento la pantalla de Juego (PlayScreen):

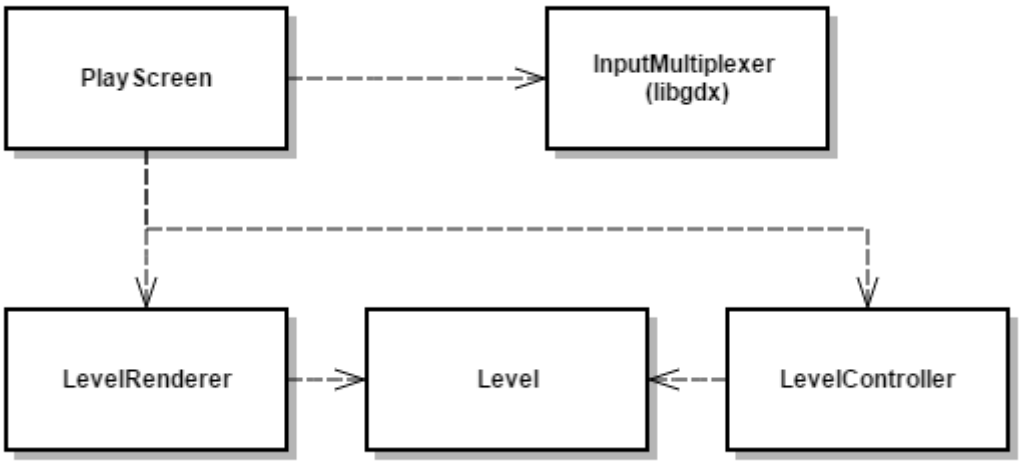


Ilustración 20: Pantalla de Juego

Las clases en las que se apoya son las siguientes:

- InputMultiplexer (libGDX): Un InputMultiplexer se encarga de procesar la entrada del usuario (teclado, pantalla táctil, etc.), para procesar esta entrada delega el proceso a una lista ordenada de InputProcessor. En el apartado de implementación veremos con mayor detalle cómo funciona esta delegación.
- LevelRenderer: es la clase encargada de dibujar el nivel, esto incluye a todos los enemigos, los objetos recolectables, los obstáculos, al jugador y el mapa. También se ocupa de manejar la cámara y de animar ciertos elementos.
- LevelController: su principal labor es dar vida a todos los elementos del nivel, proporciona la inteligencia artificial de los enemigos, mueve al jugador, comprueba las colisiones, reproduce sonidos con las acciones del jugador y maneja las pulsaciones (teclado / botones táctiles) del jugador que mueven al personaje.

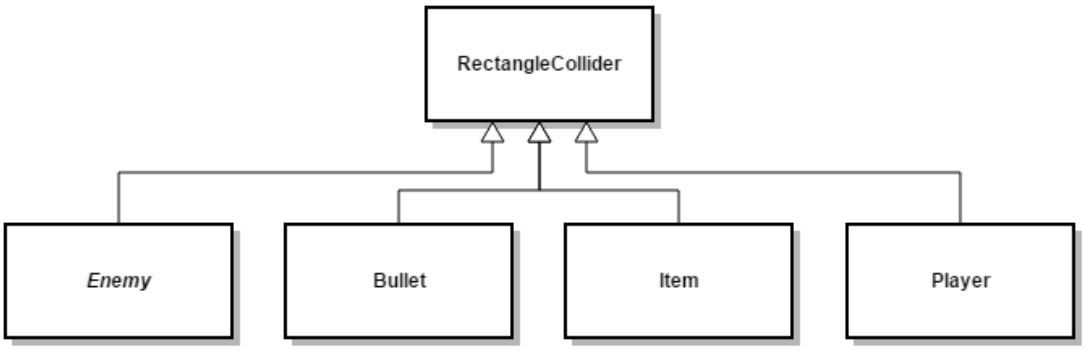


Ilustración 21: Detección de colisiones

Para detectar las colisiones entre las distintas entidades que se encuentran en el mapa hemos diseñado la clase RectangleCollider de la que heredaran todas las clases para las que queramos calcular colisiones:

- RectangleCollider: aparte de proporcionar la capacidad de detectar colisiones con otros objetos RectangleCollider, proporciona una posición en el espacio, un ancho y un alto.

- Enemy: clase abstracta que representa un enemigo, contiene propiedades que todos los enemigos tendrán, tales como puntos y vida.
- Bullet: utilizamos esta clase para representar los proyectiles que dispara el jugador y los disparados por los enemigos.
- Item: representa un objeto recolectable, por ejemplo, una moneda que al recogerla otorga puntos al jugador.

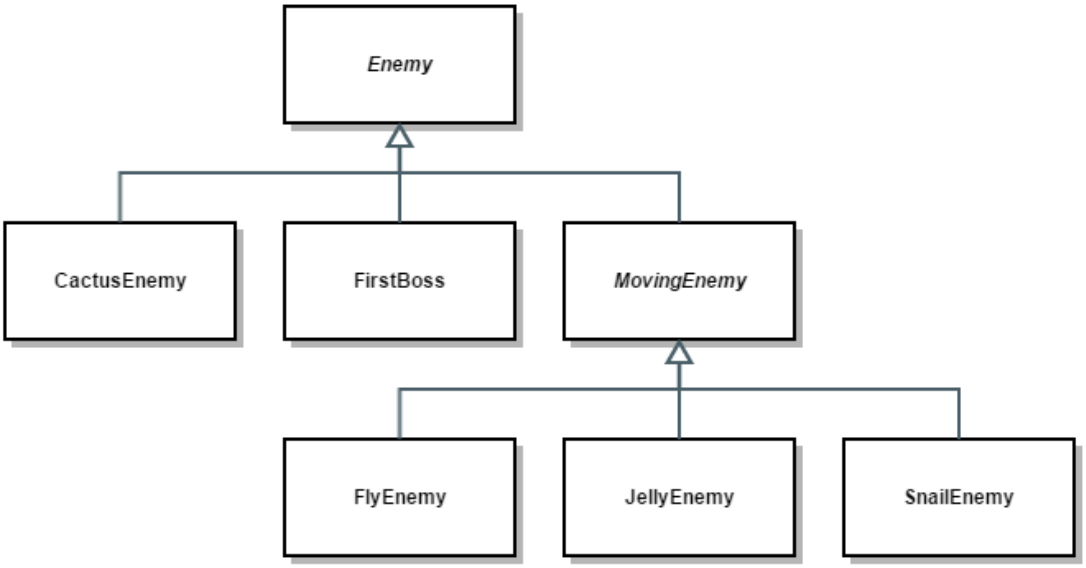


Ilustración 22: Enemigos

De la clase Enemy heredan todas las clases de enemigos que hemos creado:

- CactusEnemy: representa un enemigo con forma de cactus, es un enemigo estático con capacidad de disparo.
- FirstBoss: representa un enemigo con forma humanoide, es un enemigo con capacidad de salto y de disparo múltiple. Es el primer Jefe Final que hemos desarrollado (Ver característica J.10 del listado de características).
- MovingEnemy: clase abstracta que representa un enemigo con capacidad de moverse.
- FlyEnemy: enemigo con forma de mosca, vuela en una dirección cambiado el sentido del vuelo cada 0-7 segundos (aleatorio), también cambia el sentido cuando se encuentra con un obstáculo.
- JellyEnemy: enemigo con forma de gelatina rosa, camina en un sentido hasta que se encuentra con un obstáculo, entonces cambia de sentido.
- SnailEnemy: enemigo con forma de caracol, se comporta igual que la clase JellyEnemy en cuanto al movimiento, para eliminar a este enemigo hay que dispararle dos veces, en caso de no conseguir darle la segunda vez, al cabo de un tiempo, recupera la vida.

5.5 Implementación

En esta sección nos centraremos en explicar los detalles técnicos que han permitido llevar a cabo este proyecto, empezaremos hablando de la librería libGDX.

5.5.1 libGDX

libGDX es un framework de desarrollo de videojuegos escrito en Java y con algunos componentes escritos en C y C++ para componentes que necesitan un mayor rendimiento. Permite el desarrollo de videojuegos para dispositivos móviles y para ordenadores de escritorio utilizando el mismo código. Es multiplataforma y soporta Windows, Linux, Mac OS X, Android, iOS, y navegadores web con soporte WebGL.

Permite al desarrollador escribir, probar y depurar su aplicación en el propio ordenador y utilizar el mismo código en Android. Nos proporciona la capacidad de abstraernos de las diferencias entre las distintas plataformas. El ciclo normal de desarrollo con esta librería consiste en mantenerse desarrollando en el PC tanto como nos sea posible, y periódicamente ir probando en Android que el proyecto todavía funciona. Su principal objetivo es proporcionar compatibilidad total entre ordenadores de escritorio y dispositivos móviles.

Módulos

Simplificando un poco, libGDX consta de seis interfaces que proporcionan una manera de interactuar con el sistema operativo. Cada uno de los motores implementan estás interfaces:

- Application: ejecuta la aplicación e informa al usuario del API acerca de eventos relacionados con la aplicación, por ejemplo, el redimensionamiento de la ventana. Proporciona facilidades para realizar logging y métodos para consultar distintos parámetros como el uso de memoria.
- Files: Expone el sistema de ficheros de la plataforma. Proporciona una abstracción sobre distintos tipos de ubicación de ficheros utilizando un sistema de manejo de ficheros personalizado.
- Input: proporciona al usuario del API información acerca de la entrada del usuario, por ejemplo ratón, teclado, pantalla táctil, etc. Se permite la consulta constante (polling) y el proceso dirigido por eventos.
- Net: proporciona una forma de acceder a recursos http/https de una manera multiplataforma, también permite la creación de servidores TCP y sockets de cliente.
- Audio: permite reproducir efectos de sonido y música.
- Graphics: expone OpenGL ES 2.0 y permite establecer modos de videos, entre otras cosas.

Clases envoltorio

El único código específico de la plataforma que tenemos que escribir, es el de las clases envoltorio, o clases de arranque. Para cada una de las plataformas, estas clases se encargan de instanciar una implementación concreta de la interfaz Application, proporcionada por el motor de la plataforma. Veamos cómo se escriben estas clases de arranque, la versión de escritorio:

```
public class Main {
	public static void main(String[] args) {
		LwjglApplicationConfiguration cfg = new
LwjglApplicationConfiguration();
		cfg.title = "super-mega-buzz";
		cfg.useGL20 = true;
		cfg.width = 1280;
		cfg.height = 720;
```

```
            cfg.fullscreen = false;
            new LwjglApplication(new SuperMegaBuzz(), cfg);
        }
}
```

Para Android, la clase de arranque correspondiente sería:

```
public class MainActivity extends AndroidApplication {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        AndroidApplicationConfiguration cfg = new
AndroidApplicationConfiguration();
        cfg.useGL20 = true;

        initialize(new SuperMegaBuzz(), cfg);

getWindow().getDecorView().setSystemUiVisibility(View.SYSTEM_UI_FLAG_L
OW_PROFILE);
    }
}
```

Estas dos clases las guardamos en proyectos separados.

El verdadero código de la aplicación se encuentra en la clase que implementa la interfaz ApplicationListener (en nuestro caso la clase SuperMegaBuzz), una instancia de esta clase se le pasa a los métodos de inicialización de cada uno de los motores que implementan la clase Application. La clase Application se encargará de llamar a los métodos de la interfaz ApplicationListener en el momento adecuado.

5.5.2 Pantallas

En este apartado explicaremos los mecanismos utilizados para crear los menús de la aplicación, también explicaremos el funcionamiento de los botones de estos menús y finalmente hablaremos de la pantalla de juego y de cómo dibujamos todos los elementos de ella.

La clase principal de nuestra aplicación, “SuperMegaBuzz”, extiende la clase abstracta Game que define una propiedad de tipo Screen. Esta propiedad representa la pantalla que se está mostrando actualmente al usuario y a ella se le delegan todas las responsabilidades definidas en la interfaz ApplicationListener (implementada por la clase abstracta Game). Para crear las interfaces de nuestra aplicación utilizaremos Scene2d.

Scene2d es un módulo de libGDX que nos facilita el manejo y el renderizado de componentes 2d, a los que denomina “Actores” (clase Actor). Estos actores se añadirán a un contenedor denominado “Stage” (Escenario). Los actores almacenan su posición, color, visibilidad, tamaño, entre otras propiedades. También son los responsables de detectar pulsaciones (clics de ratón o pantalla táctil). Ejemplos de actores serían: botones, campos de texto, imágenes, tablas, etc.

Para poder usar la propiedad Screen (de la clase Game), hemos definido una clase abstracta (AbstractScreen) que implementa la interfaz Screen. En esta implementación de la clase Screen, hemos añadido una propiedad del tipo Stage. Usando esta propiedad crearemos nuestras interfaces. A continuación les mostramos un diagrama en el que se resume esta primera parte de la explicación:

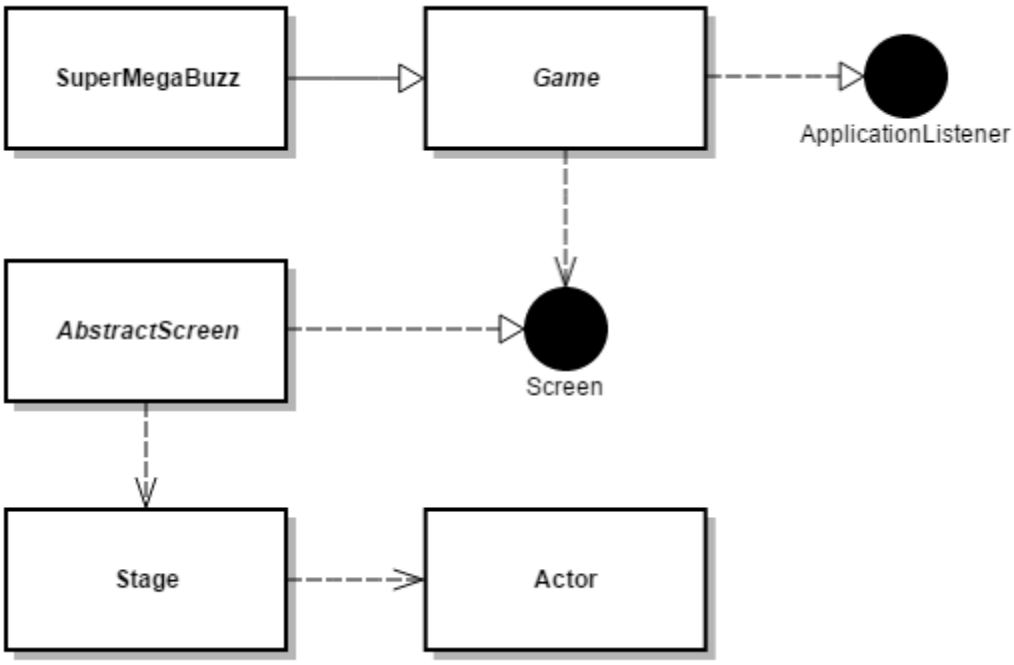


Ilustración 23: Pantallas

Una vez tenemos esta implementación, crear una pantalla para la aplicación resulta sumamente simple: basta con crear una clase que herede de AbstractScreen que añada objetos de tipo "Actor" al objeto "Stage" definido en la clase AbstractScreen.

El "Actor" que añadiremos en la mayoría de las pantallas será un objeto de tipo "Table" (extiende de la clase "Actor"). La clase Table nos permite organizar la pantalla utilizando una estructura tabular y añadir actores a las celdas de esa tabla, normalmente botones (clase TextButton) y etiquetas (clase Label). Para los botones se nos permite definir una acción al pulsarlos, que en la mayoría de casos consistirá en cambiar la pantalla actual de la aplicación (navegación entre pantallas).

5.5.3 Renderizado Pantalla de Juego

La pantalla de Juego (PlayScreen) es distinta al resto de pantallas, en ella combinamos varios métodos de dibujo para conseguir el resultado deseado. En los siguientes diagramas podemos ver los distintos elementos que intervienen en el renderizado:

Ilustración 24: Pantalla PlayScreen

Para dibujar la interfaz de usuario (botones e indicadores de puntos y tiempo, marcados en rojo) se utiliza la clase Stage como se ha indicado en el apartado "Pantallas". Por otra parte, para dibujar el resto de elementos, se utiliza la clase LevelRenderer.

La clase LevelRenderer es la clase encargada de dibujar el nivel. Para dibujar el mapa se apoya en la clase OrthogonalTiledMapRenderer y para dibujar el resto de elementos (personaje, objetos, enemigos y disparos, marcados en azul), utilizamos la clase SpriteBatch:

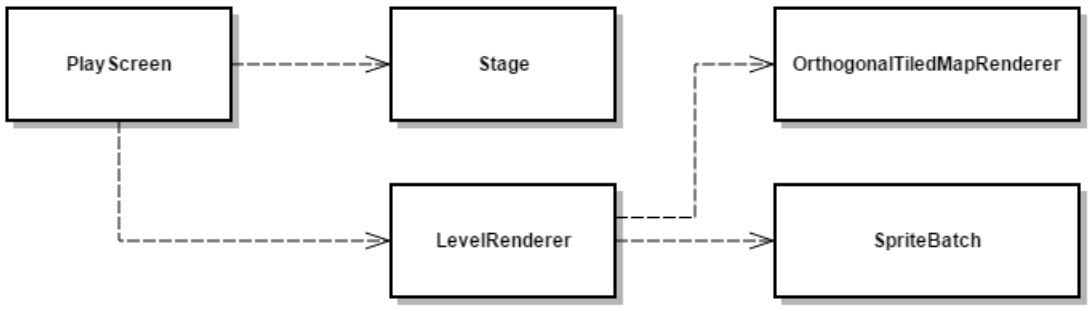


Ilustración 25: Renderizado de PlayScreen

Además de estas clases se utiliza la clase OrthographicCamera para manejar la cámara:

- OrthogonalTiledMapRenderer: clase proporcionado por libGDX que nos permite dibujar en pantalla un objeto de tipo TiledMap.
- TiledMap: consiste en una matriz de dos dimensiones que contiene referencias a un objeto de tipo "Tile". Este objeto "Tile" contiene información como el tipo de terreno, si hace daño al jugador, etc. También contiene información acerca de la representación gráfica que se debe utilizar al dibujar el propio objeto Tile.

- SpriteBatch: se utiliza para dibujar rectángulos 2D con una textura aplicada en ellos. La clase se encargara de agrupar los comandos y optimizarlos para facilitar el procesamiento por parte de la GPU. Es una clase proporcionada por libGDX.
- OrthographicCamera: nos proporciona una cámara con proyección ortográfica. La cámara irá siguiendo al personaje a lo largo del mapa.

Para hacernos una idea de lo que estamos hablando cuando nos referimos a un TiledMap, a continuación podemos ver una captura del programa de edición de mapas:

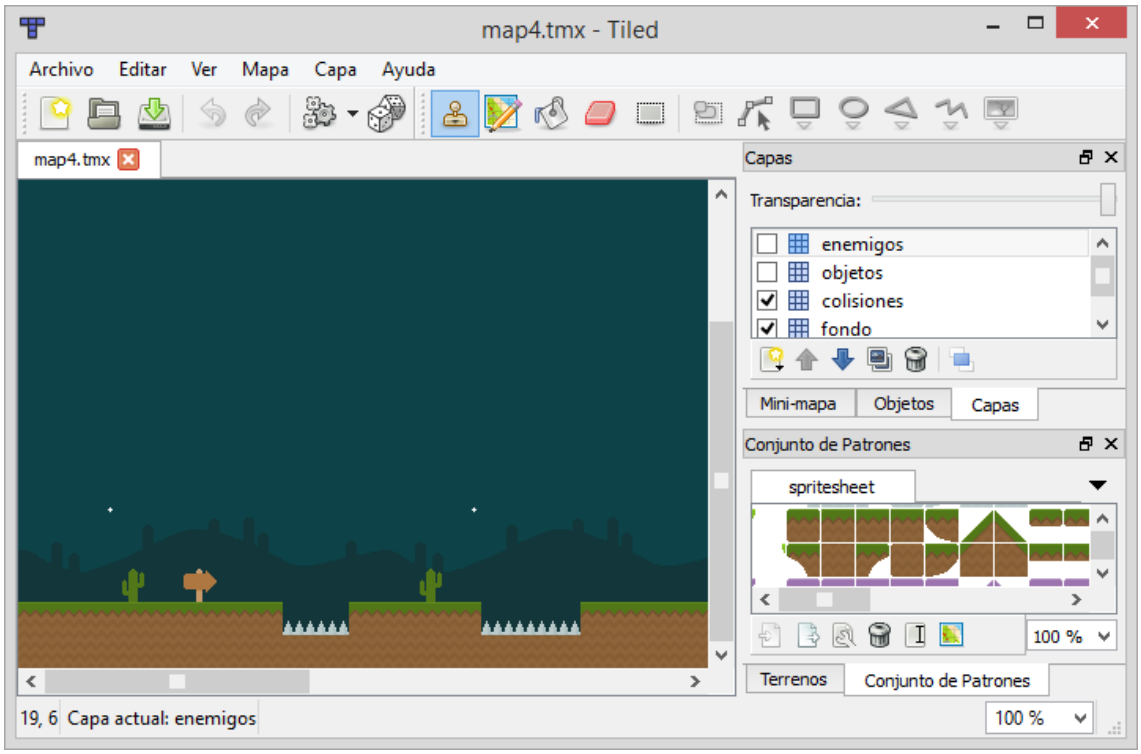


Ilustración 26: Tiled Map Editor

En la captura de pantalla se aprecia en la parte izquierda el mapa que estamos editando.

En la parte superior derecha tenemos el listado de capas que hemos creado. Una capa es un conjunto de objetos "Tile", por ejemplo, tenemos una capa para el "fondo" que consiste en elementos decorativos del mapa y otra capa "colisiones" para los elementos con los que el jugador puede colisionar.

En la parte inferior derecha tenemos cada uno de los objetos "Tile" que podemos utilizar para crear nuestro mapa.

5.5.4 Controles Pantalla de Juego

Otro de los retos que nos encontramos en la pantalla de juego (PlayScreen) fue la implementación de los controles para permitir al usuario controlar al personaje y a la vez mantener la interfaz proporcionada por la clase Stage.

Afortunadamente, libGDX nos proporciona la clase InputMultiplexer que implementa la interfaz InputProcessor y delega sus responsabilidades a una lista ordenada de InputProcessor. Un

InputProcessor se utiliza para recibir eventos de entrada del teclado y de la pantalla táctil (la pantalla táctil se emula con el ratón en PC).

La clase Stage extiende la clase InputAdapter que a su vez implementa la interfaz InputProcessor, por lo que podemos añadirlo a un InputMultiplexer. La clase Stage detecta cuando un Usuario pulsa los elementos de la interfaz, por ejemplo, el botón de pausa.

Nuestra clase LevelController también implementa la interfaz InputProcessor para detectar las pulsaciones en el teclado:

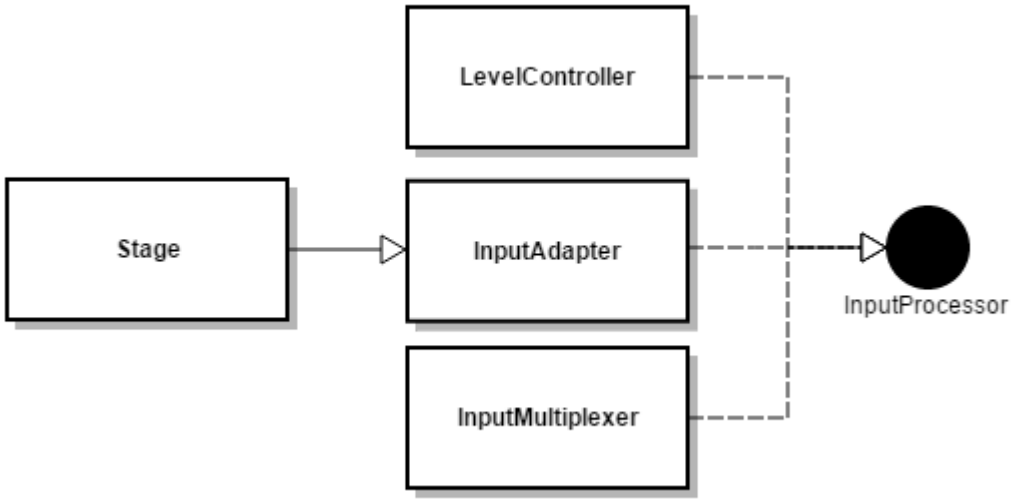


Ilustración 27: Manejo controles en PlayScreen

Una vez creados estos objetos le indicamos a libGDX que queremos utilizar nuestro InputMultiplexer utilizando el módulo "Input" (Gdx.input) concretamente el método "setInputProcessor".

El código que resultante es el mostrado a continuación:

```
multiplexer = new InputMultiplexer();
multiplexer.addProcessor(stage);
multiplexer.addProcessor(levelController);
Gdx.input.setInputProcessor(multiplexer);
```

5.5.5 Controlador Pantalla de Juego

En este apartado nos centraremos en explicar el controlador de la pantalla de juego, la clase LevelController. Esta clase se encarga de aplicar físicas (movimiento del personaje), calcular colisiones y dar inteligencia a los enemigos entre otras cosas.

Físicas

Para simular la gravedad en el nivel, en la clase LevelController aplicamos constantemente una aceleración negativa en el eje vertical sobre el personaje que controla el jugador. Para conseguir que el jugador se mueva horizontalmente, aplicamos, según las teclas que pulse el jugador, una aceleración negativa o positiva en el eje horizontal.

El código que se encarga de esto es el siguiente:

```
player.getAcceleration().y = gravity;
player.getAcceleration().scl(delta);
player.getVelocity().add(player.getAcceleration().x,
            player.getAcceleration().y);
```

Para permitir al jugador saltar, cuando pulsamos la tecla asignada a esta función aplicamos una fuerza positiva en el eje vertical:

```
player.getVelocity().y = JUMP_SPEED;
```

Para frenar al jugador cuando deja de caminar, aplicamos una reducción en la velocidad para conseguir un efecto de frenado suave (damp = 0,90):

```
player.getVelocity().x *= damp;
```

Colisiones

Para detectar colisiones se utiliza la clase RectangleCollider, cada objeto de tipo RectangleCollider posee una posición en el espacio, un ancho y una altura, en el siguiente diagrama definimos visualmente las colisiones:

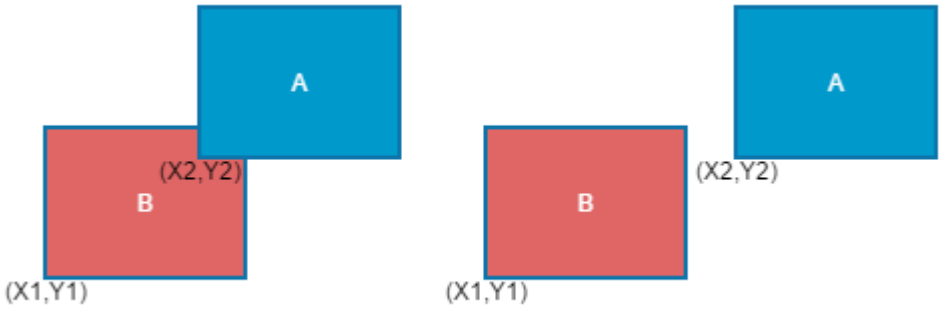


Ilustración 28: Colisiones

En la figura de la izquierda los objetos RectangleCollider han colisionado mientras que en la figura de la derecha no lo han hecho. El código que se encarga de hacer estas comprobaciones es el que se muestra a continuación:

```
float left1 = this.getX();
float right1 = this.getX() + this.getWidth();
float left2 = rectangleCollider2.getX();
float right2 = rectangleCollider2.getX() +
rectangleCollider2.getWidth();

float bottom1 = this.getY();
float top1 = this.getY() + this.getHeight();
float bottom2 = rectangleCollider2.getY();
float top2 = rectangleCollider2.getY() +
rectangleCollider2.getHeight();

if (bottom1 > top2) return false;
if (top1 < bottom2) return false;
if (right1 < left2) return false;
if (left1 > right2) return false;

return true;
```

Cuando devolvemos el valor false significa que no hemos detectado colisión, al devolver true significa que hemos detectado colisión.

Cada vez que movemos al personaje, comprobamos que no haya colisionado con ningún obstáculo o enemigo. En caso de colisionar con un obstáculo impedimos que continúe avanzando y si colisiona con un enemigo, matamos al personaje y lo devolvemos al inicio del Nivel.

El mismo proceso de comprobación se repite para los disparos: si son disparos enemigos que colisionan con el personaje, lo devolvemos al inicio del nivel y si son disparos del personaje que colisionan con los enemigos, reducimos la vida de los enemigos.

Los objetos recolectables también son susceptibles a colisionar con el Jugador, en caso de colisión los objetos desaparecen y su puntuación se suma al marcador de puntos.

5.5.6 Audio Pantalla de Juego

Para reproducir sonidos durante la partida hemos utilizado el módulo de libGDX denominado audio. Para los sonidos puntuales, como los sonidos que se reproducen al saltar, disparar, recoger una moneda, avanzar de nivel o morir, hemos utilizado la clase Sound. Por otra parte para los sonidos que se reproducen continuamente, como la música de fondo, hemos empleado la clase Music:

- Sound: Un objeto de tipo Sound es un clip de audio corto que puede ser reproducido múltiples veces en paralelo. Se carga completamente en memoria, por lo que es recomendable cargar solamente ficheros pequeños.
- Music: Una instancia de la clase Music representa un fichero de audio cargado utilizando streaming. Se utiliza para ficheros grandes y cuando queremos reproducir continuamente un sonido ya que tiene capacidad de looping (reproducción continua).

5.5.7 Persistencia

Con el fin de guardar los avances del jugador, hemos creado un sistema que escribe el estado de la partida, utilizando el sistema de ficheros, al cerrar la aplicación. Para conseguir esto hemos creado una clase llamada PersistService que utilizara las clases Json y Base64Coder del paquete Utils proporcionado por libGDX.

La clase que persistiremos es la clase LevelProvider, que a su vez contiene un listado de los mundos disponibles, cada mundo contiene un valor booleano que indica si el mundo ha sido superado (todos los niveles superados) y el listado de niveles de ese mundo, es en la clase Level (Nivel) en donde guardaremos si el nivel ha sido superado, la puntuación obtenida y el mejor tiempo para el nivel, entre otros valores. Un ejemplo de la clase LevelProvider en formato JSON:

```
{
   availableWorlds:[
      {
         class:com.me.supermegabuzz.domain.World,
         levels:[
            {
               levelNumber:0,
               completed:true,
               highScore:1100,
               bestTime:25.354853,
```

```
                mapFile:maps/map1.tmx,
                damp:0.9,
                gravity:-600
            },
            ...
            {
                levelNumber:9,
                completed:false,
                highScore:0,
                bestTime:0,
                mapFile:maps/map10.tmx,
                damp:0.9,
                gravity:-600
            }
        ],
        completed:false,
        worldName:"World 1"
    },
      ...
    {
        class:com.me.supermegabuzz.domain.World,
        levels:[
            {
                levelNumber:0,
                completed:false,
                highScore:0,
                bestTime:0,
                mapFile:maps/map11.tmx,
                damp:0.9,
                gravity:-600
            },
            ...
            {
                levelNumber:9,
                completed:false,
                highScore:0,
                bestTime:0,
                mapFile:maps/map20.tmx,
                damp:0.9,
                gravity:-600
            }
        ],
        completed:false,
        worldName:"World N"
    }
  ]
}
```

Para evitar que sea tan sencillo modificar si un nivel ha sido completado o no, codificaremos la información utilizando Base64. Un ejemplo de cómo quedaría la información codificada:

```
e2F2YWlsYWJsZVdvcmxkczpbe2NsYXNzOmNvbS5tZS5zdXBlcm1lZ2FidXp6LmRvbWFpbi
5Xb3JsZCxsZXZlbHM6W3tsZXZlbE51bWJlcjowLGNvbXBsZXRlZDp0cnVlLGhpZ2hTY29y
ZToxMTAwLGJlc3RUaW1lOjE4LjE4NDMsbWFwRmlsZTptYXBzL21hcDEudG14LGRhbXA6MC
45LGdyYXZpdHk6LTY
…
OdW1iZXI6OSxjb21wbGV0ZWQ6ZmFsc2UsaGlnaFNjb3JlOjAsYmVzdFRpbWU6MCxtYXBGa
WxlOm1hcHMvbWFwMjAudG14LGRhbXA6MC45LGdyYXZpdHk6LTYwMH1dLGNvbXBsZXRlZDp
mYWxzZSx3b3JsZE5hbWU6IldvcmxkIDIifV19
```

5.5.8 Resultados

Ilustración 29: Pantalla de presentación (SplashScreen)

Ilustración 30: Menú principal (MainMenuScreen)

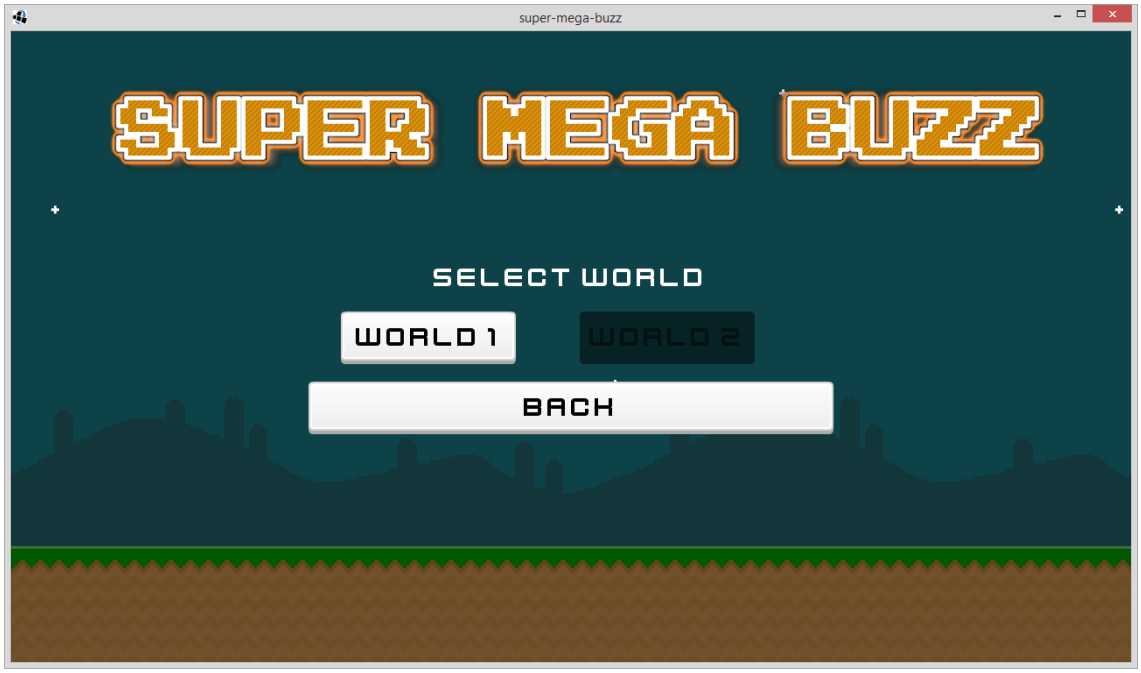


Ilustración 31: Selección de Mundo (SelectWorldScreen)

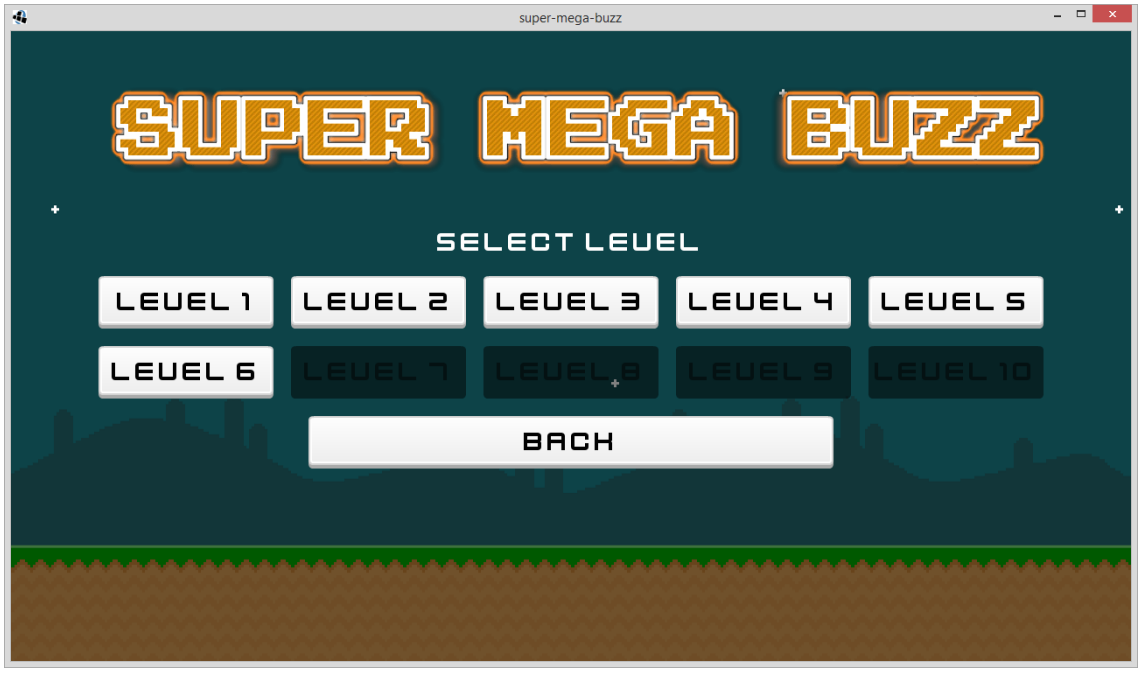


Ilustración 32: Selección de Nivel (SelectLevelScreen)

Ilustración 33: Pantalla de Juego (PlayScreen)

Ilustración 34: Pantalla de Juego (PlayScreen)

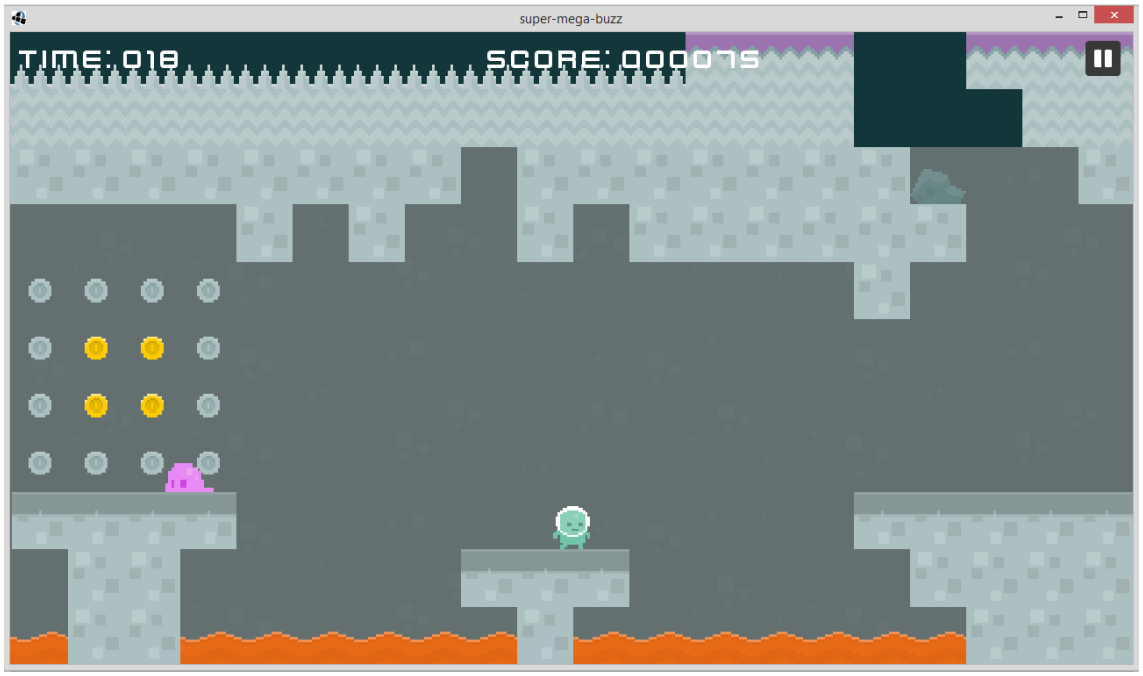


Ilustración 35: Pantalla de Juego (PlayScreen)

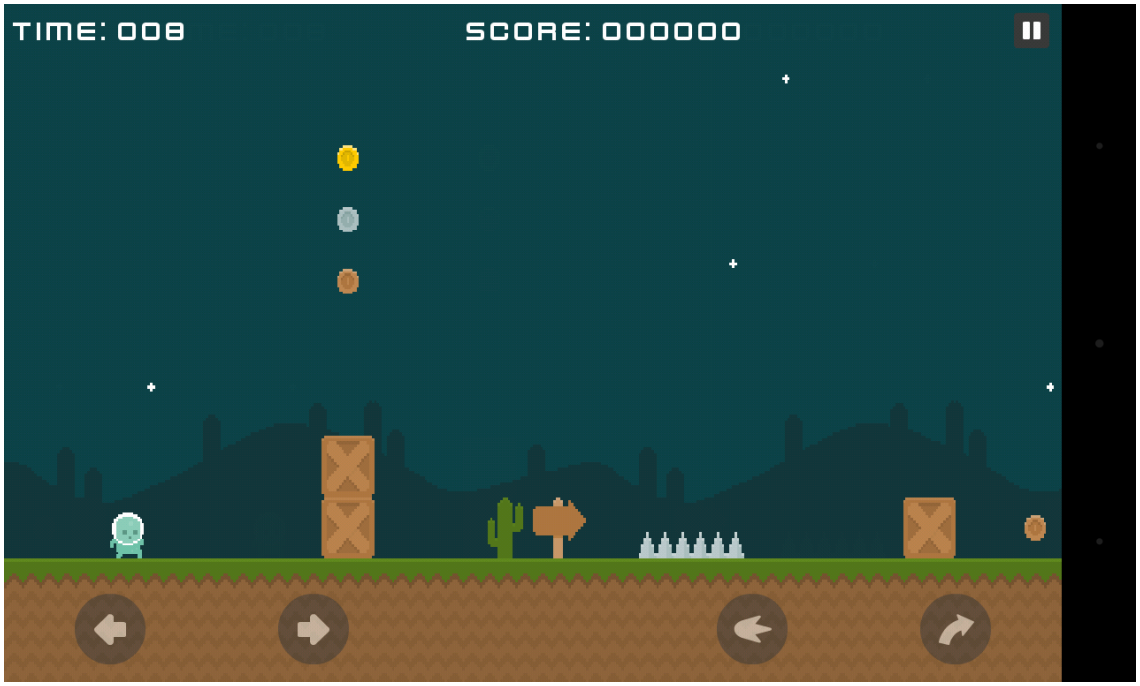


Ilustración 36: Pantalla de Juego (Móvil)

Ilustración 37: Menú de Pausa

Ilustración 38: Resumen nivel finalizado

6. Conclusiones y trabajo futuro

Para finalizar, en este capítulo recogeremos las conclusiones extraídas del trabajo realizado y también añadiremos algunas características y mejoras que se podrían implementar como trabajo futuro.

6.1 Conclusiones

Mediante la realización de este proyecto hemos conseguido crear un videojuego capaz de ejecutarse en múltiples plataformas. Para ello hemos empleado el lenguaje de programación Java y hemos utilizado la librería libGDX.

Considerando los objetivos planteados al principio de este documento y comparándolos con los resultados obtenidos, se puede concluir que hemos alcanzado los objetivos de este trabajo.

Además hemos extraído las siguientes conclusiones durante el desarrollo del proyecto:

- El framework de desarrollo de videojuegos libGDX proporciona un conjunto de herramientas que simplifica en gran medida el proceso de desarrollo de un videojuego, sin embargo, sigue siendo un proceso complicado. No es tan sencillo de utilizar como otras herramientas de creación de videojuegos, como GameMaker, por ejemplo. Además la documentación sobre el framework no es demasiado completa.
- Utilizando Java como lenguaje de programación y libGDX, hemos conseguido abstraernos de la plataforma en la que se ejecuta nuestro videojuego. Gracias a esto el código específico por plataforma consiste en una simple clase envoltorio.
- La planificación ha resultado de vital importancia y ha permitido que el proyecto salga adelante.
- La metodología empleada ha resultado sumamente útil, desde la etapa de análisis de requisitos hasta la realización del diseño nos ha facilitado sumamente la posterior implementación del sistema.

Como valoración personal del proyecto, la experiencia ha sido muy gratificante y he conseguido obtener una visión de todo lo que conlleva la creación de un videojuego. Considero que los conocimientos adquiridos a lo largo del proyecto me resultarán sumamente útiles en mi futuro profesional.

6.2 Trabajo Futuro

El producto final, fuera del alcance de este proyecto, debería también incluir las siguientes características.

El jugador estará representado en los niveles por un personaje que será elegido al inicio de cada nivel del conjunto de personajes que posea ése jugador. Cada personaje tendrá una representación gráfica y capacidades características.

Al finalizar un nivel se nos sumará dependiendo de la puntuación obtenida una cantidad de "estrellas" (0-3) a nuestra cuenta, las "estrellas" funcionarán como una moneda de cambio con las que podremos desbloquear personajes y mundos en la tienda del juego.

Como modelo de negocio podríamos utilizar las microtransacciones también denominadas compras integradas en la aplicación. Mediante este sistema los jugadores podrán comprar "estrellas" para luego usarlas en la tienda.

Resumiendo, las mejoras y características que se proponen para aumentar en gran medida el valor de este proyecto:

- Tienda: La inclusión de una tienda dentro del juego resultaría sumamente interesante, en ella se podrían comprar personajes y mundos. Los distintos personajes tendrían una representación gráfica característica y habilidades distintivas. Los mundos proporcionarían diez niveles extra para disfrutar.
- Moneda: Una vez incluida la tienda dentro del juego, resulta evidente la necesidad de crear una moneda con la que los jugadores puedan comprar los mundos y personajes. Desde mi punto de vista sería interesante que esta moneda se pueda conseguir jugando al juego pero también pagando dinero real.
- Microtransacciones: Permitirían comprar Monedas para gastar en la Tienda utilizando dinero real.
- Servidor: Para poder utilizar el concepto de Tienda, Monedas y Microtrasacciones será necesario rediseñar un poco la arquitectura para añadir un Servidor que será el encargado de proporcionar los Mundos desbloqueados y los Personajes comprados.
- Multijugador: Para conseguir diferenciarse de la competencia resultaría muy atractivo la implementación la capacidad de Jugar con un amigo a los niveles del videojuego.
- Publicidad: como alternativa para obtener ingresos se podría considerar la posibilidad de añadir publicidad al juego, como recomendación, evitaría añadir publicidad a la pantalla de Juego ya que esto podría perjudicar gravemente la experiencia. Incluso se podría ofrecer una versión sin publicidad pagando un pequeño precio.

También, a nivel general, sería interesante refinar la arquitectura para que los cambios a realizar en un futuro no sean muy costosos.

Anexo I: Competencias

G1. Poseer y comprender conocimientos en un área de estudio (Ingeniería Informática) que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados, incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio.

G2. Aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.

G3. Reunir e interpretar datos relevantes (normalmente dentro de su área de estudio) para emitir juicios que incluyan una reflexión sobre temas relevantes de índole social, científica o ética.

G4. Transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado.

G5. Desarrollar aquellas habilidades de aprendizaje necesarias para emprender estudios posteriores con un alto grado de autonomía.

N1. Comunicarse de forma adecuada y respetuosa con diferentes audiencias (clientes, colaboradores, promotores, agentes sociales, etc.), utilizando los soportes y vías de comunicación más apropiados (especialmente las nuevas tecnologías de la información y la comunicación) de modo que pueda llegar a comprender los intereses, necesidades y preocupaciones de las personas y organizaciones, así como expresar claramente el sentido de la misión que tiene encomendada y la forma en que puede contribuir, con sus competencias y conocimientos profesionales, a la satisfacción de esos intereses, necesidades y preocupaciones.

N2. Cooperar con otras personas y organizaciones en la realización eficaz de funciones y tareas propias de su perfil profesional, desarrollando una actitud reflexiva sobre sus propias competencias y conocimientos profesionales y una actitud comprensiva y empática hacia las competencias y conocimientos de otros profesionales.

N3. Contribuir a la mejora continua de su profesión así como de las organizaciones en las que desarrolla sus prácticas a través de la participación activa en procesos de investigación, desarrollo e innovación.

N4. Comprometerse activamente en el desarrollo de prácticas profesionales respetuosas con los derechos humanos así como con las normas éticas propias de su ámbito profesional para generar confianza en los beneficiarios de su profesión y obtener la legitimidad y la autoridad que la sociedad le reconoce.

T1. Capacidad para concebir, redactar, organizar, planificar, desarrollar y firmar proyectos en el ámbito de la ingeniería en informática que tengan por objeto, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada, la concepción, el desarrollo o la explotación de sistemas, servicios y aplicaciones informáticas. (G1, G2)

T2. Capacidad para dirigir las actividades objeto de los proyectos del ámbito de la informática, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada. (G1, G2)

T3. Capacidad para diseñar, desarrollar, evaluar y asegurar la accesibilidad, ergonomía, usabilidad y seguridad de los sistemas, servicios y aplicaciones informáticas, así como de la información que gestionan. (G1, G2)

T5. Capacidad para concebir, desarrollar y mantener sistemas, servicios y aplicaciones informáticas empleando los métodos de la ingeniería del software como instrumento para el aseguramiento de su calidad, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada. (G1, G2)

T6. Capacidad para concebir y desarrollar sistemas o arquitecturas informáticas centralizadas o distribuidas integrando hardware, software y redes, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada. (G1, G2)

T7. Capacidad para conocer, comprender y aplicar la legislación necesaria durante el desarrollo de la profesión de Ingeniero Técnico en Informática y manejar especificaciones, reglamentos y normas de obligado cumplimiento. (N4)

T8. Conocimiento de las materias básicas y tecnologías, que capaciten para el aprendizaje y desarrollo de nuevos métodos y tecnologías, así como las que les doten de una gran versatilidad para adaptarse a nuevas situaciones.

T11. Capacidad para analizar y valorar el impacto social y medioambiental de las soluciones técnicas, comprendiendo la responsabilidad ética y profesional de la actividad del Ingeniero Técnico en Informática.

CII01. Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

CII02. Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

CII04. Capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.

Anexo II: Manual de Usuario

La utilización de la aplicación resulta sumamente sencilla, al iniciar la aplicación nos encontraremos con el menú principal:

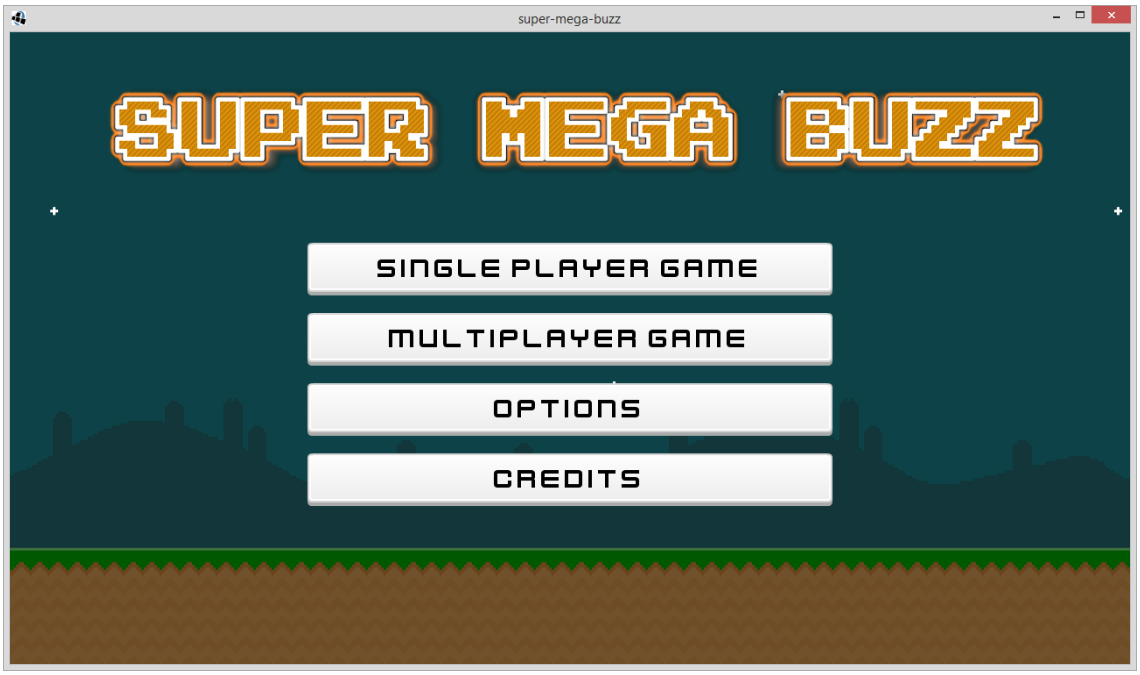


Ilustración 39: Menú Principal

Para elegir una opción utilizaremos el ratón en ordenadores de escritorio y la pantalla táctil en dispositivos móviles. Las opciones que nos encontramos son las siguientes:

- Single Player Game: Nos permite jugar en solitario al videojuego.
- Multiplayer Game: Nos permitiría crear o unirnos a una partida (No implementado).
- Options: Nos permite acceder a las opciones de nuestra aplicación, hemos implementado la visualización de las opciones, pero no las guardamos ni aplicamos.
- Credits: Permite acceder a los créditos. Hemos listado las personas que nos han proporcionado material gráfico, sonidos y/o música, también incluimos enlaces a sus páginas web personales o perfiles sociales.

Si elegimos la primera de las opciones (Single Player Game) se nos pedirá que elijamos un Mundo (World) y posteriormente que elijamos un Nivel (Level), nótese que los niveles y mundos que tienen un color más oscuro representan niveles o mundos bloqueados. Una vez seleccionado el Nivel se nos presentará la pantalla de juego:

Ilustración 40: Pantalla de Juego (PC)

Para manejar al personaje utilizaremos las siguientes teclas:

- Flecha izquierda: mover al personaje hacia la izquierda
- Flecha derecha: mover al personaje hacia la derecha
- Tecla Control: hace que el personaje dispare.
- Tecla Espacio: hace que el personaje salte.

En esta pantalla nos encontramos los siguientes elementos:

- Indicador de tiempo: arriba a la izquierda, representa el número de segundos que llevamos en el nivel actual.
- Indicador de puntuación: arriba al centro, representa el total de puntos acumulados para el nivel actual.
- Botón de pausa: arriba a la derecha, nos permite pausar el juego.

En caso de ejecutarse en un dispositivo móvil, nos encontraremos con la siguiente pantalla:

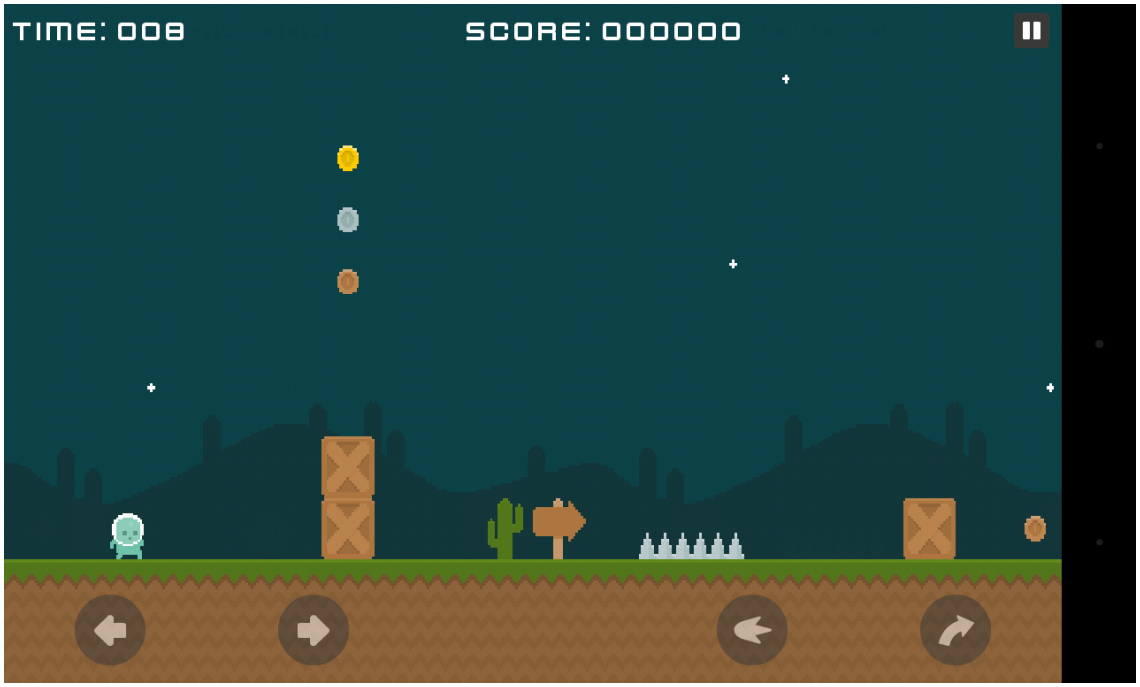


Ilustración 41: Pantalla de Juego (Android)

Podemos ver que nos encontramos con algunos elementos nuevos en la pantalla:

- Botón izquierda: situado en la esquina inferior izquierda, mueve al personaje hacia la izquierda.
- Botón derecha: situado a la derecha del "Botón izquierda", mueve al personaje hacia la derecha.
- Botón salto: se encuentra en la esquina inferior derecha, hace que el personaje salte.
- Botón disparo: se encuentra a la izquierda del "Botón salto", hace que el personaje dispare.

Cuando pulsamos el "Botón de pausa" se nos presenta el menú de pausa, que nos ofrece las siguientes opciones:

- Resume Game: Volver a la partida.
- Restart Level: Reiniciar el nivel.
- Exit Game: Salir del nivel.

Ilustración 42: Menú de Pausa

Cuando finalizamos un nivel, se nos presenta un resumen del nivel con las siguientes posibles acciones:

- Botón “X”: salir al menú principal.
- Botón “Flecha hacia atrás”: Repetir el nivel.
- Botón “Play”: Jugar siguiente nivel.

Ilustración 43: Resumen del nivel

Bibliografía

[1] Ivaar Jacobson, Grady Booch, James Rumbaugh. *El Proceso Unificado de Desarrollo de Software.* Madrid: Pearson Educación S.A., 2000.

[2] *Wikipedia: The Free Encyclopedia* [Wiki en Internet]. Wikimedia Foundation, Inc. 2014 [Consulta: noviembre 2014]. Disponible en http://es.wikipedia.org/

[3] *Información sobre UDK* [en línea]. Epic Games, INC. [Consulta: noviembre 2014]. Disponible en: https://www.unrealengine.com/products/udk/

[4] *Información sobre Unity* [en línea]. Unity Technologies. [Consulta: noviembre 2014]. Disponible en: https://unity3d.com/es/unity

[5] Tõnis Tiigi. *Información sobre LimeJS* [en línea]. DigitalFruit. [Consulta: noviembre 2014]. Disponible en: http://www.limejs.com/

[6] *Información sobre cocos2D* [en línea]. Cocos2D. [Consulta: noviembre 2014]. Disponible en: http://www.cocos2d-swift.org/

[7] *Información sobre GameMaker* [en línea]. Yoyo Games. [Consulta: noviembre 2014]. Disponible en: https://www.yoyogames.com/studio

[8] Mario Zechner. *Información sobre libGDX* [en línea]. Badlogic Games. [Consulta: noviembre 2014]. Disponible en: http://libgdx.badlogicgames.com/

[9] Nicolas Gramlich. *AndEngine* [blog]. [Consulta: noviembre 2014]. Disponible en: http://www.andengine.org/blog/

[10] Gustavo Steigert. "Tutoriales sobre Screens, Stage y Actor (libGDX)". *steigert | Android Development* [blog]. São Paulo: 2012. [Consulta: noviembre 2014]. Disponible en: http://steigert.blogspot.com.es/search/label/libgdx

[11] Tamas Jano. "Tutoriales sobre físicas de salto y colisiones (libGDX)". a*gainst the grain* [blog]. Londres: 2013. [Consulta: noviembre 2014]. Disponible en: http://obviam.net/index.php/getting-started-in-android-game-development-with-libgdx-create-a-working-prototype-in-a-day-tutorial-part-1/

[12] Robin S. (dermetfan). "Videotutoriales sobre Tiled Map (libGDX)". Youtube, 2013. [Consulta: noviembre 2014]. Disponible en: https://www.youtube.com/playlist?list=PLXY8okVWvwZ0qmqSBhOtqYRjzWtUCWylb