DISEÑO Y DESARROLLO DE VIDEOJUEGO DE LUCHA CON EL MOTOR UNITY

Trabajo de Fin de Grado

Universidad de Las Palmas de Gran Canaria
Escuela de Ingeniería Informática
Grado en Ingeniería Informática

Autor: Jose David Santana Rocha
Tutor: Agustín Trujillo Pino

Las Palmas de Gran Canaria
Diciembre de 2017

Contenido

1 OBJETIVOS INICIALES Y MOTIVACIÓN

El objetivo principal de este Trabajo de Fin de Grado es desarrollar un videojuego en 2D del género de lucha aplicando técnicas de Ingeniería del Software, utilizando para ello un motor de desarrollo. Como principales características, se pretende desarrollar el sistema de combate entre dos jugadores, la inteligencia artificial del oponente controlado por la máquina, así como todas las interfaces de usuario correspondientes a indicadores o menús.

Por otro lado, también se pretende que todo el proceso sea de apoyo a aprender sobre el desarrollo de videojuegos y para familiarizarse con un entorno de trabajo con el que nunca se había trabajado.

Lo que me motivó personalmente a tomar esta temática para el Trabajo de Fin de Grado fue la posibilidad de desarrollar mi propio videojuego con herramientas modernas y poner en práctica y afianzar los conocimientos y técnicas adquiridas a lo largo del periodo de aprendizaje en Ingeniería Informática, y que todo el proceso resultara en un producto de calidad.

2 ESTADO ACTUAL

Hoy en día, la industria de los videojuegos está en pleno crecimiento. Muchas son las compañías que desarrollan este tipo de software de entretenimiento. Y es que incluso pequeños estudios se atreven a dar el paso para hacerse un hueco en este sector, ya que cada vez existen más facilidades para que sus productos lleguen a mucha más gente.

Existen plataformas como **Kickstarter** o **Indiegogo** que permiten a los desarrolladores publicar una idea o trabajo prematuro, y así posibles interesados en ver esa idea florecer pueden dar recursos y apoyo necesario para que el proyecto siga adelante.

En el ámbito de los ordenadores, de la mano de *Valve Corporation* tenemos la plataforma **Steam** donde los desarrolladores pueden publicar fácilmente su producto terminado o en desarrollo para su presentación a los usuarios finales. Por otro lado, en dispositivos móviles es posible publicar este tipo de aplicaciones en **Play Store** de *Google* para sistemas Android, o **App Store** de *Apple* para sistemas iOS.

2.1 Historia de los videojuegos

El origen de los videojuegos se remonta a la década de 1940 tras la creación de los primeros superordenadores programables como el **ENIAC** de 1946. Primero aparecieron diversos intentos de videojuegos de ajedrez, hasta que en la década de los 60 empezaron a surgir los primeros videojuegos modernos. Desde entonces, el crecimiento de las tecnologías ha permitido evolucionar al mundo de los videojuegos hasta el día de hoy.

En el año 1962, apareció el videojuego **Spacewar!** de mano de *Steve Russell* y otros estudiantes del Instituto de Tecnología de Massachusetts (MIT) en un computador **PDP-1**. Este videojuego se considera el primer juego interactivo de ordenador. Cosechó mucho éxito entre los miembros del MIT, que distribuyeron numerosas copias a través de *ARPAnet* y otros medios con el fin de demostrar las capacidades del nuevo **PDP-1**. Pero, aun así, **Spacewar!** no llegó a patentarse ni comercializarse, ya que para su uso se necesitaba un equipo de unos 120.000 dólares. Posteriormente, el juego ha servido de inspiración para futuras versiones, como las incluidas de serie en las posteriores consolas domésticas de **Atari** y **Magnavox**.

Imagen de Spacewar!

El 27 de enero de 1972, la empresa **Magnavox** comenzó la producción de la que es la primera videoconsola de la historia, la **Magnavox Odyssey**. Se trataba de un dispositivo capaz de generar señales simples en la pantalla de una televisión. Los veintiocho juegos diferentes que se lanzaron eran de una sencillez extrema: ping-pong, tenis de mesa, voleibol, etc. Dado el reducido hardware de la consola, no disponían de sonido. No contenía ninguna **unidad central de procesamiento** o **memoria de acceso aleatorio**, estaba compuesta únicamente de transistores, resistencias y condensadores.

Magnavox Odyssey

Siguiendo sus pasos, en 1975 la compañía **Atari** lanzó al mercado la consola **Atari Pong**, la cual no llevaba cartuchos, tan solo tenía el juego **Pong**, que imitaba el juego de tenis de mesa de **Magnavox Odyssey**. Atari vendió los derechos a **Sears**, que quedó sorprendida con la consola y decidió comprarla, conociéndose posteriormente la consola como **Tele-Games** bajo la marca Sears.

Pese a la poca innovación que presentaba, Atari vendió más de 55.000 unidades de la consola.

Versión de Pong de Sears, llamada Tele-Games

Con la aparición de los microprocesadores de propósito general, los videojuegos fueron ganando cada vez más complejidad. A principios de 1976, Atari estaba trabajando en un nuevo proyecto, la nueva consola **Atari 2600** que iba a ser la primera en utilizar tecnología de microprocesador, pero debido a los problemas económicos que atravesaba la empresa, fue adelantada por **Fairchild Channel F** en agosto de 1976.

Mientras tanto, en Japón nacía **Space Invaders** de la compañía **Taito** para máquinas recreativas en el año 1978, un juego que consistía en disparar contra oleadas de naves alienígenas y obtener la mayor puntuación posible. Éste es uno de los primeros juegos del género **shoot 'em up** y uno de los más importantes de la historia de los videojuegos.

Space Invaders

Durante 1982, la popularidad de los videojuegos creció considerablemente gracias a **Space Invaders**, y los ingresos producidos por la industria habían pasado de los 454 millones de dólares en 1978 hasta los 5313 millones en 1982, es decir, que estaba incrementando sus beneficios un 5% mensualmente. Las máquinas recreativas estaban dispersas por toda clase de establecimientos, como restaurantes, hoteles o supermercados. Este periodo fue conocido como la edad de oro de los videojuegos.

Posteriormente, siguieron apareciendo sistemas en el mercado, como **ZX Spectrum**, **Commodore 64** y **Yamaha MSX**, los cuales sentaron las bases en los sistemas de **8 bits**, que ya permitían videojuegos que presentaban una mayor complejidad y características gráficas.

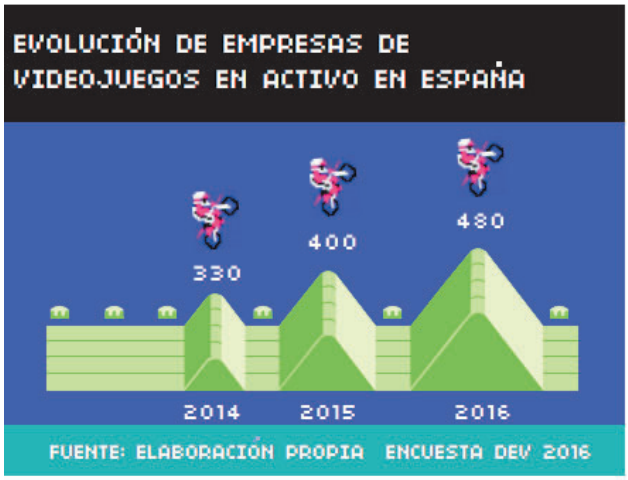
Imágenes de Metal Gear, videojuego de MSX

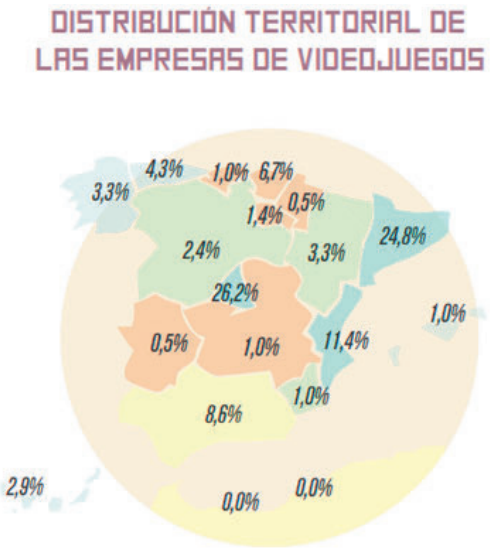
En Japón surgieron dos grandes consolas que competían por el público aficionado, **Nintendo Entertainment System** (1983) y **Sega Master System** (1985) de las compañías Nintendo y Sega respectivamente. La calidad de los juegos de ambas plataformas hizo propulsarse a la industria y en prácticamente todos los hogares disponían ya de una videoconsola. Este es considerado un punto de inflexión por muchos expertos y aficionados, ya que supuso un antes y un después en el mundo de las videoconsolas domésticas. A partir de aquí siguieron apareciendo modelos en el mercado, las consolas de **16 bits**, **Sega Megadrive** y **Super Nintendo Entertainment System**, consolas portátiles como **Game Boy** de Nintendo, consolas de **32 bits** y gráficos en **3D** de **Nintendo 64, Sega Saturn** o **PlayStation** de **Sony**, y así hasta donde las conocemos el día de hoy.

Game Boy de Nintendo (1989), que popularizó el concepto de videoconsola portátil

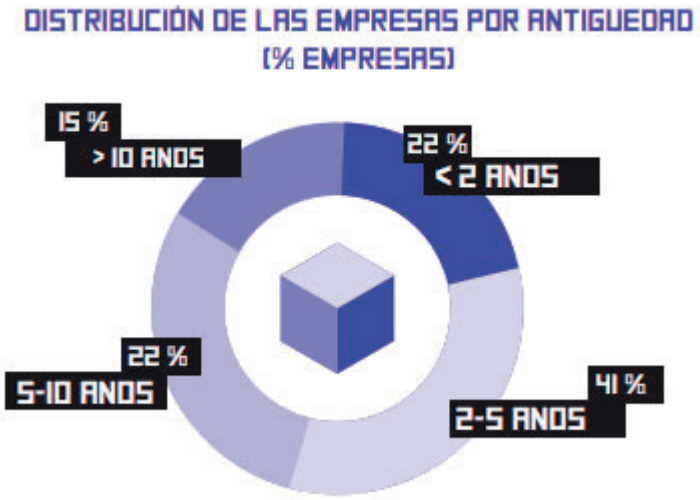
2.2 Situación de la industria en España

Datos del *Libro Blanco del Desarrollo Español de Videojuegos 2016* nos indican que hay censadas **480 empresas** de videojuegos en activo en España, un 20% más que 2015. También afirman que hay cerca de **125 iniciativas y proyectos empresariales** que próximamente se consolidarán como empresas.

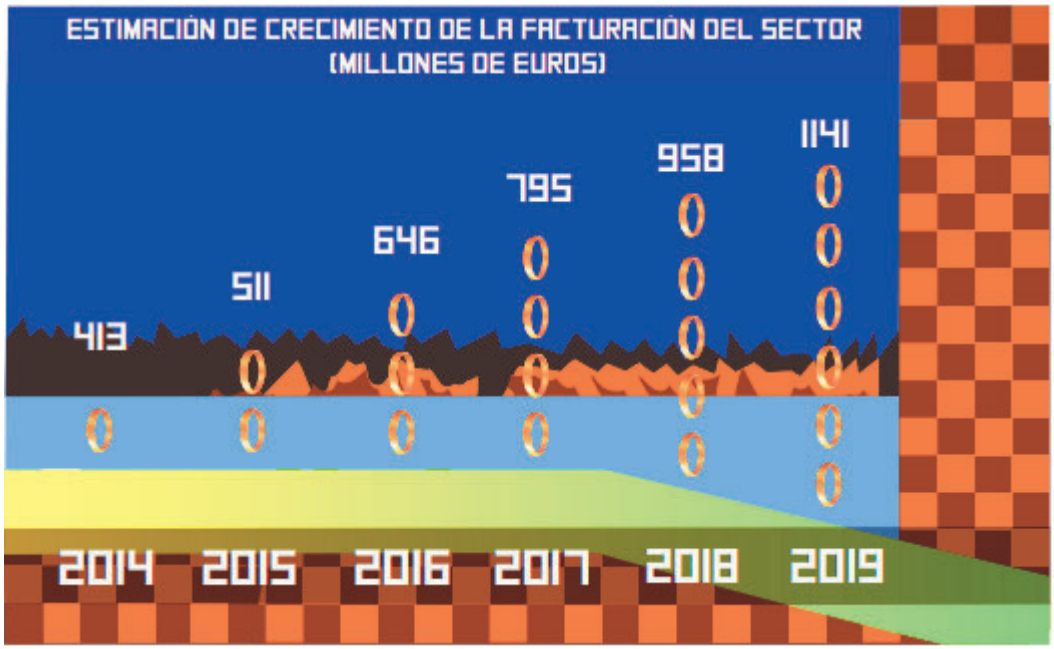




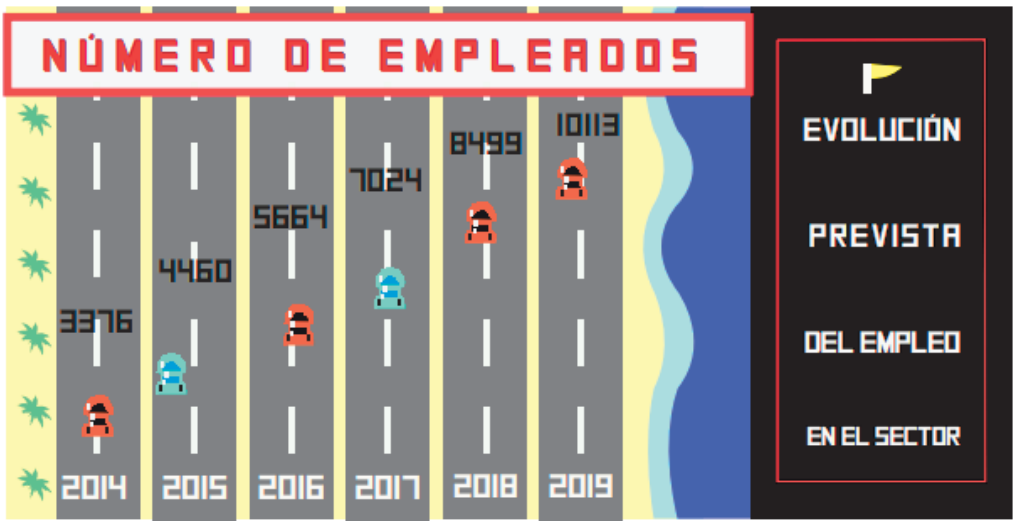
Y es que la industria del videojuego es joven y está en pleno crecimiento, como nos confirma la creación del **63% de las empresas actuales** en los últimos **5 años**.



Teniendo en cuenta los datos obtenidos en el año 2015, la industria de desarrollo de videojuegos facturó **510,7 millones de euros**, lo que supone un 24% más que en el año anterior. Además, según las previsiones habrá un crecimiento anual del 22,4%, lo que supondría llegar a los **1140 millones de euros** de facturación en el año 2019.



Como consecuencia de este enorme crecimiento, en 2015 cerca de **4460 profesionales** se incorporaron a la industria española del videojuego, entre empleados y colaboradores directos. En total, unos **7849 profesionales vinculados al sector** si incluimos las estimaciones de empleos generados de forma indirecta. Si las previsiones de crecimiento del empleo son correctas, se alcanzará hasta más de **10000 empleos en 2019**.



A nivel internacional, España supone un mercado importante en la industria, ocupando el **cuarto lugar** a nivel europeo y el **octavo lugar** a nivel mundial en cuanto a ingresos.

Aunque, si tenemos en cuenta facturación de la industria, ese lugar debería ser **muy inferior** con respecto al mercado, ya que países como Francia, Alemania, Suecia y Finlandia obtienen una **facturación muy superior** a España estando compuestas por **menos empresas**.

INDUSTRIA DEL VIDEOJUEGO EN LOS PRINCIPALES PAÍSES

PAÍS	EMPRESAS	FACTURACIÓN (M €)
EEUU	SIN DATO	3.700
FRANCIA	250	3.677
CANADÁ	472	2.800
ALEMANIA	320	1.820
REINO UNIDO	1092	1.490
SUECIA	213	952
FINLANDIA	260	800
ESPAÑA	480	511
PAÍSES BAJOS	352	190
DINAMARCA	171	148
BÉLGICA	SIN DATO	41

Fuente: Dutch Game Garden

2.3 Motores de videojuegos más populares

Unity es un motor de videojuego multiplataforma creado por **Unity Technologies**. Unity está disponible como plataforma de desarrollo para **Microsoft Windows**, **OS X**, **Linux**. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataformas, ya sea para los distintos sistemas operativos de PC, para consolas, dispositivos móviles, entre otros.

Tiene 3 tipos de licencia:

Personal: Gratuito. Para principiantes, estudiantes y aficionados que desean explorar y empezar con Unity.

Plus: 35$ al mes. Para los creadores que tienen la seria intención de hacer realidad su visión y piensan publicar sus contenidos.

Pro: 125$ al mes. Para los profesionales que necesitan absoluta flexibilidad y anhelan una personalización avanzada.

Características principales:

- Unity puede usarse junto con **Blender**, **3ds Max**, **Maya**, **Softimage**, **Modo**, **ZBrush**, **Cinema 4D**, **Cheetah3D**, **Adobe Photoshop**, **Adobe Fireworks** y **Allegorithmic Substance**. Los cambios realizados a los objetos creados con estos productos se actualizan automáticamente en todas las instancias de ese objeto durante todo el proyecto sin necesidad de volver a importar manualmente.
- El motor gráfico utiliza **OpenGL** (en Windows, Mac y Linux), **Direct3D** (solo en Windows), **OpenGL ES** (en Android y iOS), e interfaces propietarias (Wii). Tiene soporte para mapeado de relieve, mapeado de reflejos, mapeado por paralaje, oclusión ambiental en espacio de pantalla, sombras dinámicas utilizando mapas de sombras, render a textura y efectos de post-procesamiento de pantalla completa.
- Se usa el lenguaje **ShaderLab** para la creación de sombreadores, similar a Cg/CgFx y **DirectX HLSL Effects** (.Fx). Pueden escribirse shaders en tres formas distintas: como *Surface shaders*, como *Vertex and Fragment shaders*, o como *shaders de función fija*. Un shader puede incluir múltiples variantes y una especificación declarativa de reserva, lo que permite a Unity detectar la mejor variante para la tarjeta de vídeo actual y si no son compatibles, recurrir a un *shader alternativo* que puede sacrificar características para una mayor compatibilidad.
- El soporte integrado para **Nvidia** (antes *Ageia*), el motor de física **PhysX**, (a partir de Unity 3.0) con soporte en tiempo real para mallas arbitrarias y sin piel, ray casts gruesos, y las capas de colisión.

- El scripting viene a través de **Mono**. El script se basa en **Mono**, la implementación de código abierto de **.NET Framework**. Los programadores pueden utilizar **UnityScript** (un lenguaje personalizado inspirado en la sintaxis *ECMAScript*), **C#** o **Boo** (que tiene una sintaxis inspirada en *Python*). A partir de la versión 3.0 añade una versión personalizada de **MonoDevelop** para la depuración de scripts.
- Unity también incluye **Unity Asset Server** - una solución de control de versiones para todos los assets de juego y scripts, utilizando *PostgreSQL* como backend, un sistema de audio construido con la biblioteca *FMOD*, con capacidad para reproducir audio comprimido **Ogg Vorbis**, reproducción de vídeo con códec **Theora**, un motor de terreno y vegetación , con árboles con soporte de *billboarding*, determinación de cara oculta con *Umbra*, una función de iluminación *lightmapping* y global con *Beast*, redes multijugador *RakNet* y una función de búsqueda de caminos en mallas de navegación.

Ori and the Blind Forest, videojuego desarrollado por Moon Studios GmbH con el motor Unity

Unreal Engine es un motor de juego de PC y consolas creado por la compañía **Epic Games**. Su primera versión apareció en **1998** y ha ido evolucionando hasta convertirse en uno de los motores de videojuegos más utilizados por las grandes productoras de videojuegos de hoy en día.

La plataforma es de uso gratuito, pero las desarrolladoras que lancen al mercado sus productos utilizando esta herramienta deben aportar el 5% de los ingresos obtenidos una vez superados los 3000$ obtenidos por producto cada trimestre.

Características principales:

- Representación fotorreal en tiempo real
- Código fuente completo del motor en C++ incluido
- **Blueprints**: crear sin código
- Robusto Framework multijugador
- Sistemas de VFX y de partículas
- Efectos post-proceso de calidad cinematográfica
- Editor de materiales flexible
- Amplio conjunto de herramientas de animación
- **Sequencer**: Cinemáticas de vanguardia
- Editor completo en modo VR
- Construido para VR, AR y XR
- Terreno y vegetación
- IA avanzada
- **Unreal Audio Engine**
- Navegador de contenido
- Integración perfecta de **Perforce**
- **Marketplace**: mercado de assets
- Extensibilidad ilimitada

Hellblade: Senua's Sacrifice, videojuego desarrollado por Ninja Theory con Unreal Engine 4

CRYENGINE®

CryEngine es un motor de juego creado por la empresa alemana desarrolladora de software **Crytek**, creadora de las sagas Crysis y Far Cry, videojuegos del género *First Person Shooter* que destacan por tener una gran calidad gráfica.

Dispone de 2 tipos de membresías, la **Base**, por 50€ al mes, y la **Premium,** por 150€ al mes.

Características principales:

- Actualización en tiempo real para todas las plataformas en el editor sandbox
- Vegetación y terreno integrados
- Sistema de partículas en tiempo real
- Herramientas de río y suelo
- Creador de vehículos
- Soporte multinúcleo
- Iluminación dinámica en tiempo real
- Iluminación natural y sombras suavizadas
- Niebla de distancia volumétrica
- Texturas normal y parallax
- Oclusión ambiental (SSAO)
- Tecnología "Uber Shader"
- Adaptación de retina y HDR
- Motion blur y profundidad de campo
- Sistema de animación de personajes
- Cinemática inversa
- Editor de animación facial
- Esparcimiento de superficie
- Edición de inteligencia artificial
- Buscador de caminos dinámico
- Efectos naturales del mundo
- Agua tridimensional de alta calidad
- Motor físico con soporte multinúcleo integrado
- Entorno destructible
- Físicas deformables
- Físicas de soga
- Herramientas de análisis de rendimiento
- Sonidos y música dinámicos
- Audio envolvente
- Renderización de texturas de alta velocidad
- Raytracing
- Editor de UVs
- Modelado 3D

Ryse: Son of Rome, videojuego desarrollado por Crytek con CryEngine 4

3 APORTACIONES Y COMPETENCIAS CUBIERTAS

3.1 Aportaciones

La realización de este Trabajo de Fin de Grado ha contribuido personalmente en lo siguiente:

- Participar en un proyecto software y pasar por las distintas etapas de su desarrollo.
- Poner en práctica los conocimientos adquiridos en Ingeniería del Software.
- Obtener nuevos conocimientos con herramientas de desarrollo del sector de desarrollo de videojuegos.

Además, el desarrollo del proyecto ha provocado personalmente un mayor interés en el sector de desarrollo de videojuegos, y ahora tengo una visión más clara de todo el proceso de desarrollo de un videojuego.

A nivel técnico, este proyecto es un ejemplo de desarrollo para el aprendizaje y la asimilación de técnicas de desarrollo de videojuegos, metodologías ágiles, conocimiento de motores de videojuegos, y desarrollo de inteligencia artificial.

Personalmente, he tenido problemas para encontrar documentación sobre inteligencia artificial de los personajes de un videojuego de lucha, por lo que este trabajo podría aportar documentación necesaria sobre el tema.

3.2 Competencias cubiertas

Con este Trabajo de Fin de Grado se han cubierto las competencias generales recogidas en el Proyecto Docente, y también varias competencias específicas que se detallarán a continuación.

CII08 - Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.

Se han llevado a cabo las etapas de análisis, diseño, desarrollo y mantenimiento durante la realización del Trabajo de Fin de Grado.

CII015 - Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.

La implementación de la inteligencia artificial de los diversos personajes presentes en el videojuego desarrollado incluye esta competencia.

CII016 - Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.

Se han empleado diversas técnicas de metodologías de Ingeniería de Software durante el desarrollo.

IS01 - Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

A lo largo de la etapa de desarrollo y mantenimiento de la aplicación, se realizaron frecuentes pruebas con usuarios que daban su aprobación con los objetivos alcanzados, y se ajustaron algunos aspectos que no los alcanzaron, por lo que esta competencia queda cubierta.

IS04 - Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

Se planificó la manera de abordar los distintos obstáculos que surgieron durante el desarrollo de este proyecto, y se superaron diversas dificultades mediante soluciones que fueron eficaces.

4 PLANIFICACIÓN ESTIMADA

Al comienzo se realizó una estimación con las horas que tomaría las distintas etapas de este Trabajo de Fin de Grado.

Fases	**Duración estimada *(horas)***	**Tareas**
Estudio previo / Análisis	20	Estudio del motor de desarrollo Unity
		Familiarización con el lenguaje C#
		Planificación inicial
Diseño	30	Elaboración del Product Backlog
		Búsqueda/Diseño del arte utilizado
		Diseño de la arquitectura
Desarrollo / Pruebas	150	Desarrollo del sistema de combate
		Desarrollo de la inteligencia artificial
		Diseño e implementación de interfaces de usuario
	50	Realización y prueba de versiones beta/prototipos
Documentación / Presentación	50	Realización de la memoria
		Realización de la presentación

A lo largo del desarrollo, surgieron varios problemas para obtener el arte necesario. En ocasiones tuve que crear o modificar mis propios recursos para la creación de las características que necesitaba en un momento dado, por lo que las horas para la búsqueda y diseño del arte utilizado fueron muchas más de las esperadas inicialmente.

Además, a medida que se realizaron las versiones beta o prototipos, los usuarios que probaron estas versiones aportaron nuevas ideas para ampliar o cambiar el videojuego, por lo que las horas empleadas para el desarrollo fueron mayores que las que estimaron.

Las versiones beta del videojuego se exportaron fácilmente a medida que iba avanzando el desarrollo. Además, se contó con usuarios que ayudaron a realizar las pruebas y nos dieron su opinión, por lo que no fueron necesarias tantas horas como se estimaron.

Las horas finales empleadas se presentan en la siguiente tabla:

Fases	**Duración final** ***(horas)***	**Tareas**
Estudio previo / Análisis	20	Estudio del motor de desarrollo Unity
		Familiarización con el lenguaje C#
		Planificación inicial
Diseño	**50**	Elaboración del Product Backlog
		Búsqueda/Diseño del arte utilizado
		Diseño de la arquitectura
Desarrollo / Pruebas	**175**	Desarrollo del sistema de combate
		Desarrollo de la inteligencia artificial
		Diseño e implementación de interfaces de usuario
	20	Realización y prueba de versiones beta/prototipos
Documentación / Presentación	50	Realización de la memoria
		Realización de la presentación

5 HERRAMIENTAS

Durante todo el desarrollo, se han utilizado múltiples herramientas para los objetivos necesitados.

5.1 Trello

Al comienzo del desarrollo, se planteó una **pila de producto** con las características más importantes que debía tener el videojuego. Consiste en un tablero en la plataforma **Trello** que nos permite organizar nuestro trabajo asignando cada tarea a una tarjeta, para luego moverlas entre las columnas, con el fin de conocer el estado de todas las tareas con un vistazo rápido, una práctica muy común del sistema *Kanban*.

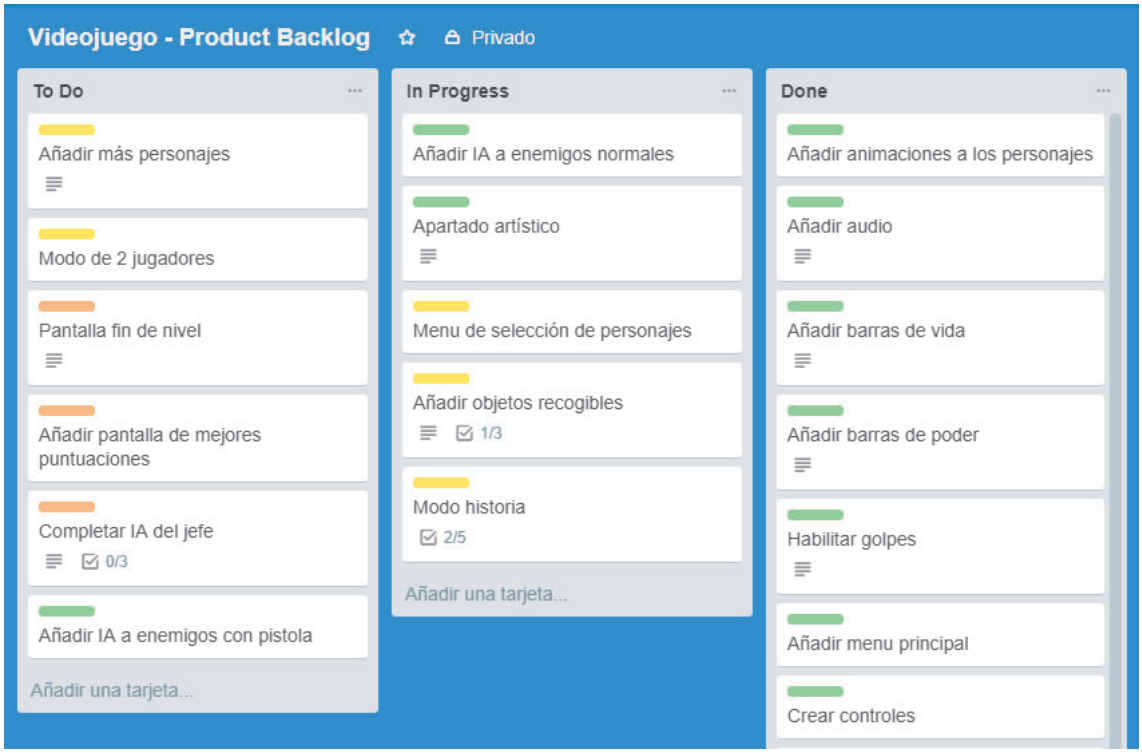


Imagen de tablero en Trello

5.2 Unity

Este es el motor de videojuegos que se utilizará durante todo el desarrollo. Con Unity podemos crear múltiples aspectos del videojuego, como nuestros niveles, menús de usuario, personajes, la música que tendrá cada nivel, etc. También podemos asociar a cada objeto un script que hayamos desarrollado para añadirle funcionalidad.

Imagen del nivel 1 del videojuego en Unity

5.3 Microsoft Visual Studio

Incluido con Unity, será el entorno de desarrollo utilizado para la implementación de todos los scripts necesarios. Incluye la extensión **Tools for Unity**, que nos facilita las labores a la hora de desarrollar y depurar contenido relacionado con Unity. Lo utilizaremos junto con el lenguaje de programación **C#**.

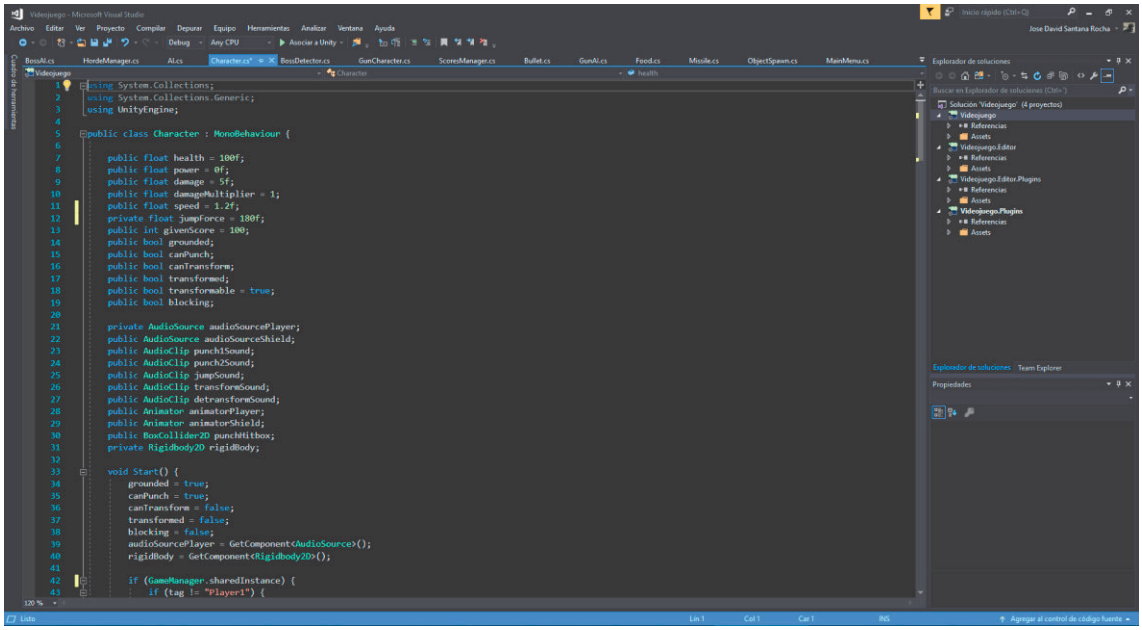


Imagen de uno de los scripts en Microsoft Visual Studio

5.4 GIMP 2

Esta herramienta es un programa de edición de imágenes que se ha utilizado para la creación o modificación del arte necesario para el videojuego. Nos permite exportar las imágenes en formato **.png**, que será el formato utilizado para el proyecto ya que soporta transparencia.

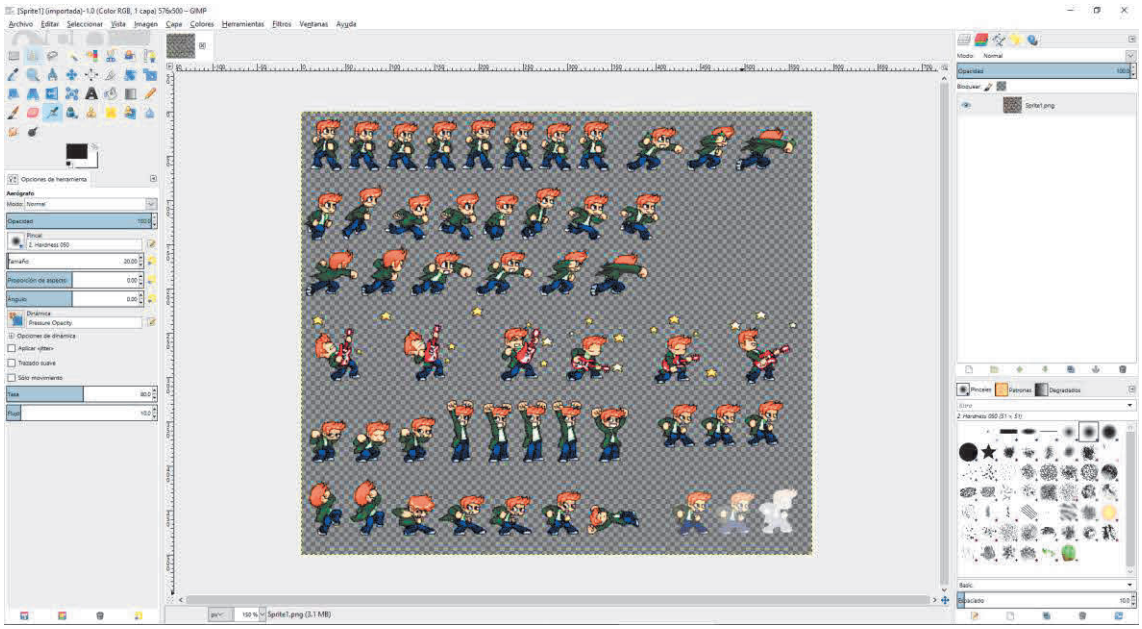


Imagen con la hoja de sprites de uno de los personajes del videojuego en GIMP 2

5.5 BeepBox

Consiste en una aplicación web que nos permite crear nuestros propios temas y canciones con ambientación retro. Permite exportar en formato **.wav**, **.midi** o incluso **.json**, por si posteriormente queremos cargar este archivo para editar el tema fácilmente.

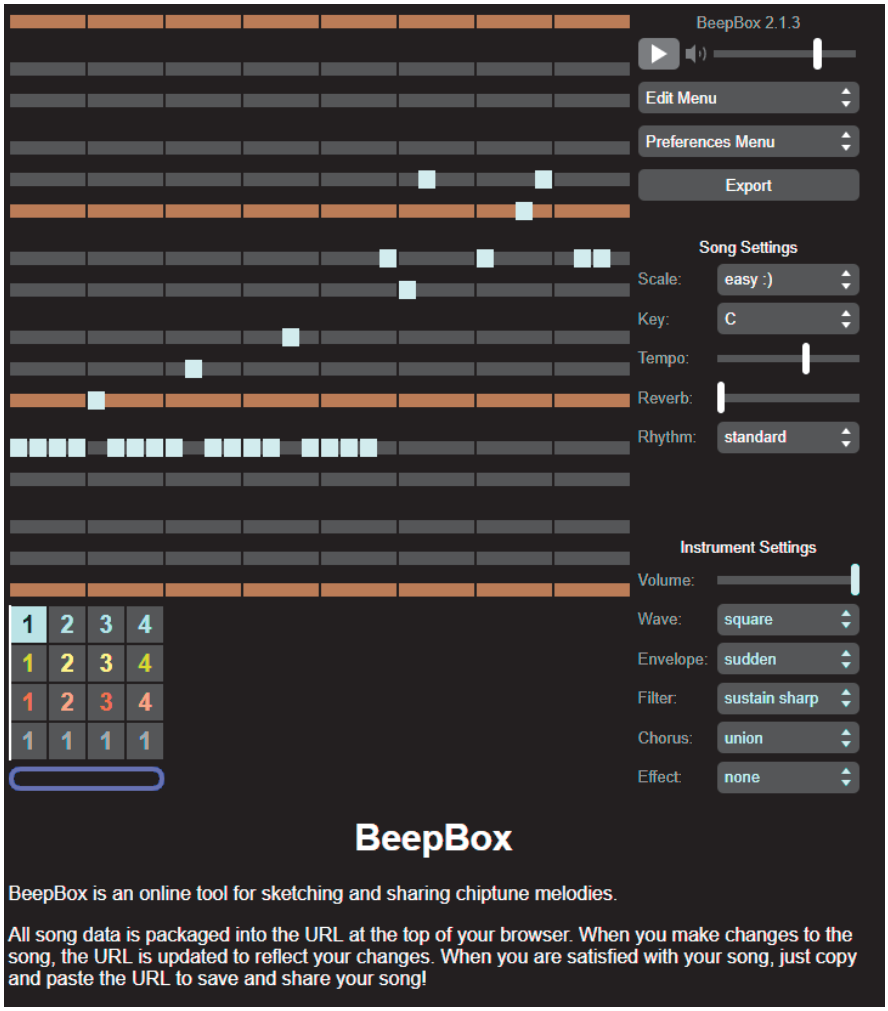


Imagen del diagrama de notas de la canción utilizada para el modo horda en BeepBox

5.6 Bfxr

Al igual que *BeepBox*, es una aplicación web que nos permite crear nuestros propios sonidos de efectos especiales como si fueran de videojuegos de **8 bits**. Podemos crear diversos tipos de sonidos, como por ejemplo **explosiones**, **saltos** o **golpes**, de manera aleatoria, y para luego editarlos a nuestro antojo si lo creemos conveniente. Nos permite exportar estos sonidos en formato **.wav**, el cual podemos usar fácilmente en Unity.

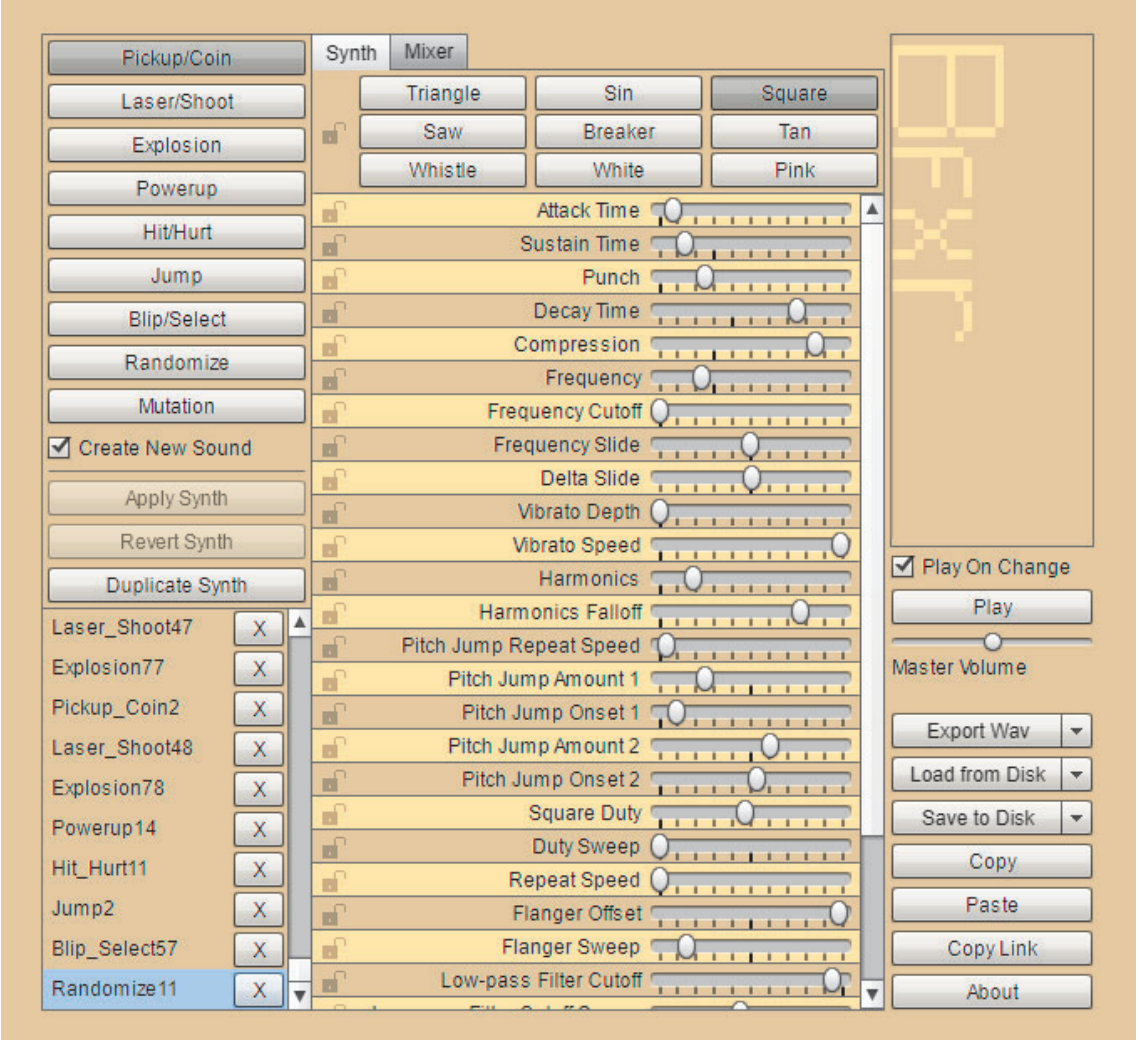


Imagen de un sonido de efecto especial en Bfxr

5.7 Audacity

Este programa nos permite editar y grabar archivos de audio para luego exportarlos en formato **.mp3**, **.ogg**, **.wav**, entre otros. Se empleó para realizar pequeños ajustes en los sonidos o para grabar sonidos necesarios para el videojuego.

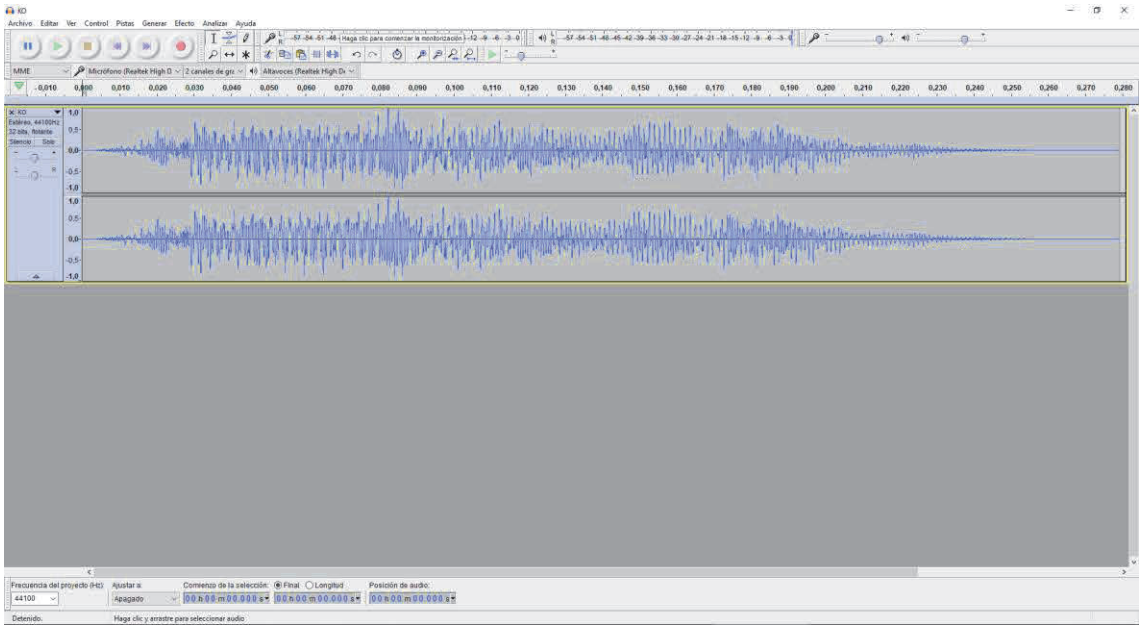


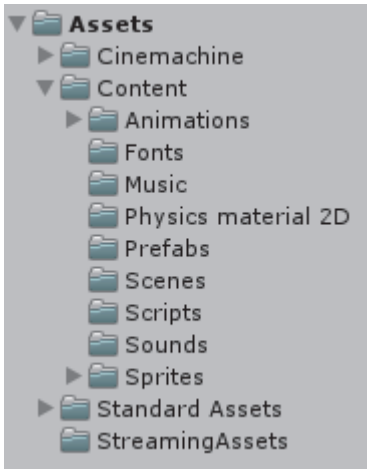
Imagen de la curva del sonido que hacen los enemigos al caer inconsciente en Audacity

6 ANÁLISIS Y DISEÑO

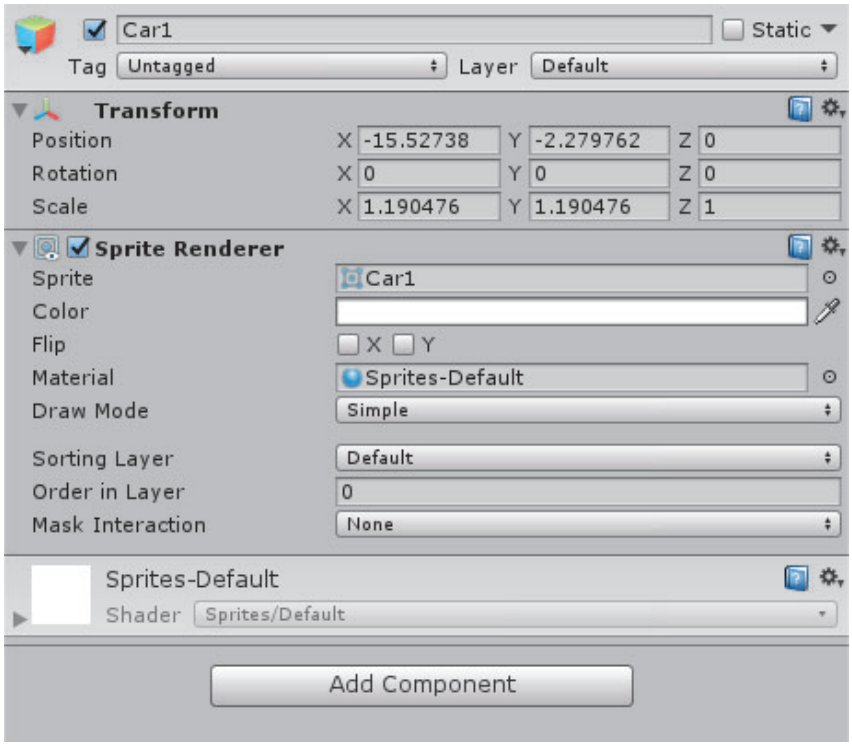
6.1 Terminología de Unity

A lo largo de la memoria se hará referencias a términos que quizás el lector no conoce. Para evitar esto, a continuación, se presentarán las palabras clave más utilizadas.

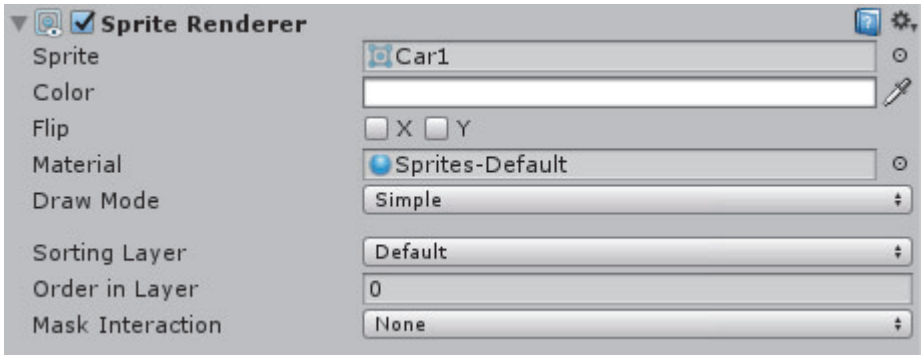
- **Assets:** son los recursos utilizados para crear el videojuego que se almacenan en la carpeta del proyecto, como imágenes, música, scripts, animaciones, etc.



- **GameObject:** son los principales elementos del proyecto. Son contenedores a los cuales se les pueden añadir, eliminar o modificar componentes u otros GameObject y que en conjunto forman un objeto en un videojuego. Pueden ser un personaje, un menú de opciones, o incluso el propio escenario.



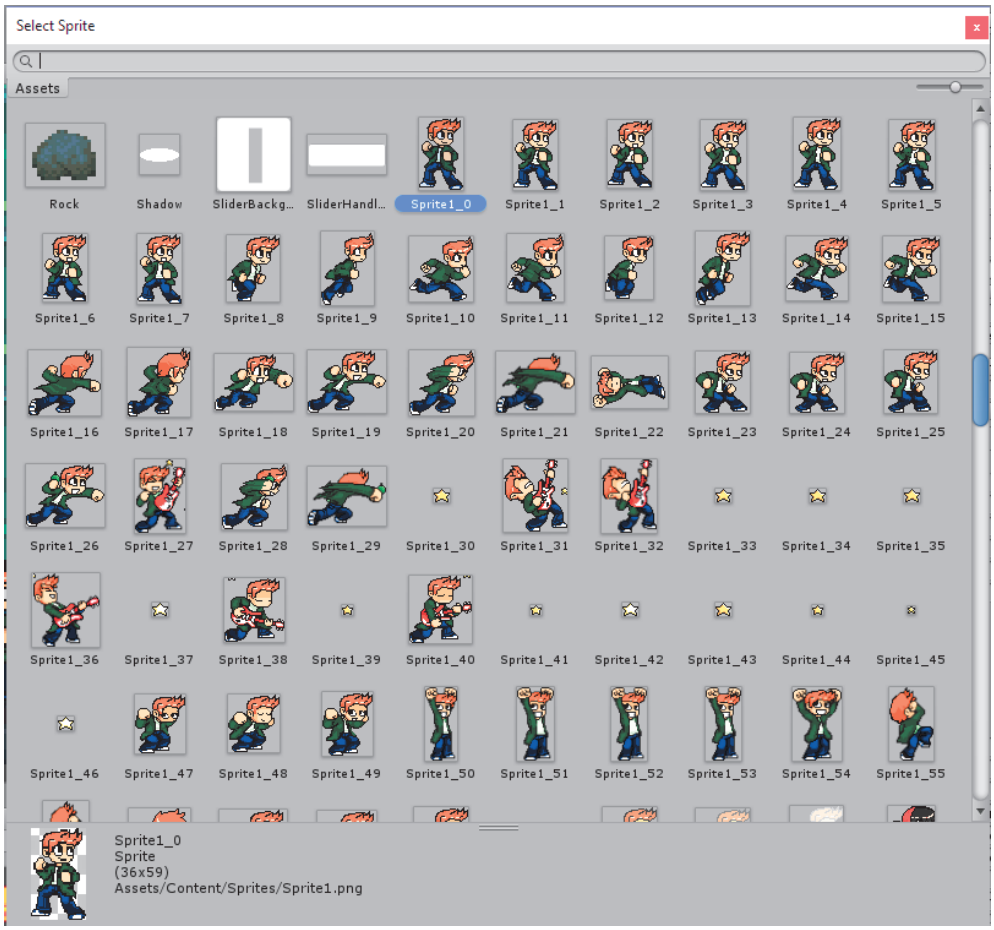
- **Componente:** son propiedades que se le pueden asignar a un GameObject y le dan diversas funcionalidades. Por ejemplo, una imagen asignada a un GameObject le otorga ese aspecto.



- **Scene:** la escena es el espacio donde se disponen los diversos GameObject y lo que conforman lo que será el propio videojuego.

- **Prefab:** es un GameObject el cual se almacena como un Asset y conserva todos sus componentes, para utilizarlo posteriormente más fácilmente. Se pueden colocar directamente en la escena sin la necesidad de crear el GameObject de cero, y también se pueden generar en la escena mediante programación.

- **Sprite**: es la imagen que se le asigna a un GameObject para que se muestre en la escena. Se le asigna mediante el componente **SpriteRenderer** y puede ser modificado con animaciones.



6.2 Aspectos del juego

El juego en sí también tiene cierta terminología que puede no ser entendida por el lector. A continuación, se recoge un pequeño glosario con las expresiones o características más importantes.

- **Barra de vida o salud:** se representa como una barra verde en la parte superior que va disminuyendo al recibir daño. La partida termina cuando se vacía. Se puede aumentar si el jugador recoge comida. Al recibir daño, podemos ver durante un instante una porción de la barra en rojo, que corresponde con el daño recibido.

- **Barra de poder:** se representa como una barra naranja bajo la barra de salud. Se va recargando al golpear al rival o levemente al recibir daño del rival. Si se llena es posible transformarse en un lobo para ganar velocidad y daño.

- **Modo versus:** modo de juego en el que cada jugador elige un personaje y luego uno se enfrenta al otro. Es posible jugar contra la inteligencia artificial o contra otro jugador. Gana el que reduzca la barra de salud del rival a cero primero.

- **Modo historia:** modo de juego en el que el jugador aparece en un escenario y debe llegar al final. Al paso le aparecen enemigos que deberá derrotar mientras esquiva misiles que caen desde arriba y minas que hay por el suelo. En este modo el jugador puede recoger granadas para lanzar a sus enemigos. Si tiene el dinero suficiente, también puede comprar comida en máquinas expendedoras para aumentar un poco su barra de salud. El nivel acaba cuando el jugador vence al jefe final situado al extremo derecho del escenario.

Jugador enfrentándose a los enemigos mientras esquiva misiles y minas. También se ve una máquina expendedora.

Combate contra el jefe final. Se puede ver una caja de granadas a la izquierda del jugador.

- **Modo horda:** modo de juego en el que el objetivo es aguantar el mayor número de oleadas de enemigos que aparecen a izquierda y derecha. Cada ronda debe ser terminada antes de que se agote el tiempo. Si se agota el tiempo o la barra del jugador llega a cero, se acaba la partida.

Imagen del modo horda. Los enemigos aparecen a ambos lados y hay que aguantar el máximo de rondas posibles para obtener mayor puntuación

6.3 Videojuegos de referencia

Tekken (saga)

Imágenes de juego de Tekken 3 de PlayStation (1997)

Inicialmente se pensó que el videojuego a desarrollar fuese algo similar. Un jugador contra el otro o contra la inteligencia artificial y ganaba el que redujera a cero antes la barra de salud del contrincante. En este videojuego hay rondas, por lo que se debe ganar más rondas que el rival para ganar el combate. Los personajes tienen una gran cantidad de combos, combinaciones de golpes que se pueden encadenar para dañar rápidamente al contrincante, habitualmente impidiéndole defenderse si falló al bloquear el primer golpe.

Esta idea se tomó para el *modo versus*.

Streets of Rage

Imágenes de juego de Streets of Rage de Sega Genesis / Mega Drive (1991)

Este videojuego es del género Beat 'em up, donde generalmente el jugador avanza a lo largo de un nivel mientras se enfrenta a grandes cantidades de enemigos al mismo tiempo. A diferencia de Tekken, en este tipo de juegos los controles no son muy complejos, por lo que la variedad de golpes que los personajes pueden realizar es muy limitada. Los jugadores pueden recoger objetos y armas con los que golpear a sus contrincantes. Al final de los niveles, hay un enemigo especialmente duro, un jefe final que el jugador debe vencer para avanzar en la aventura.

Esta idea se tomó para el *modo historia*.

Golden Axe

Imágenes de juego de Golden Axe de Sega Genesis / Mega Drive (1989)

Similar a Streets of Rage pero con ambientación medieval de fantasía, en Golden Axe los jugadores tienen que avanzar por el escenario mientras luchan contra toda clase de monstruos. Los personajes tienen poderes que pueden utilizar para vencer a grandes cantidades de enemigos a la vez. También pueden montarse en toda clase de dragones para aumentar su poder contra los enemigos.

Altered Beast

Imágenes de juego de Altered Beast de Sega Genesis / Mega Drive (1988)

Similar a los anteriores, pero sin desplazamiento vertical, el personaje se podía transformar tras conseguir el poder suficiente en toda clase de animales a lo largo de la aventura, y con ello ganaba nuevos movimientos y habilidades de combate.

Esta idea se tomó para la transformación de los personajes del videojuego.

6.4 Metodología

Con el fin de agilizar el desarrollo, se optó por seguir algunas técnicas de la metodología Scrum.

Inicialmente se recogieron todas las características principales que debía tener el videojuego en una pila de producto o *product backlog*, para posteriormente ordenarlas por prioridad, y luego se planificó la carga de trabajo en 4 sprints de 2 semanas.

Primer sprint:

Funcionalidad básica del videojuego en el modo versus. Dos jugadores se podían mover por el escenario y podían luchar en un 1 contra 1.

Se implementaron todos los menús e interfaces de usuario.

Segundo sprint:

Se intentó la implementación en dispositivos móviles, pero finalmente se descartó debido a dificultades para incluir todos los controles en la pantalla táctil.

Se implementó la inteligencia artificial del contrincante en el modo versus.

Se añadió un nuevo modo de juego: el modo historia. Aquí el personaje aparecía en un nivel donde le atacaban varios enemigos a la vez y tenía que avanzar.

Se añadieron misiles en posiciones determinadas en el modo historia que caían del cielo y el jugador tenía de esquivar.

Se añadió una máquina expendedora la cual, si el jugador tiene el dinero suficiente, proporciona comida al jugador con la que puede aumentar su salud.

Tercer sprint:

Se añadieron más personajes.

Se añadió un sistema que almacenaba las puntuaciones alcanzadas por los jugadores y mostraba una lista con las 10 mejores puntuaciones.

Se cambió el sistema de misiles del modo historia y ahora estos se generaban aleatoriamente en posiciones cercanas al jugador, en vez de generarse en posiciones fijas.

Cuarto sprint:

Se terminó de mejorar la IA de los enemigos para que bloquearan ataques y buscaran comida cercana.

Se realizó el combate contra el jefe final del modo historia y se creó el nivel 2.

Se añadió pantalla de cálculo de puntuación al finalizar la partida.

Se añadió un nuevo modo de juego: el modo horda. De 1 o 2 jugadores, el objetivo es aguantar el mayor número de oleadas de enemigos que aparecen a izquierda y derecha y acabar con ellos antes de que se agote el tiempo.

Durante la etapa de desarrollo, se hicieron a su vez versiones de demostración que fueron entregadas a usuarios que validaron nuestros objetivos, nos daban sugerencias de cambio y nos daban ideas de funcionalidades futuras.

Cada vez que terminaba un sprint se hacía una retrospectiva de todas las tareas y funcionalidades completadas y validadas, y las que no, se aplazaban para el siguiente sprint y se intentaba ajustar un poco mejor la planificación.

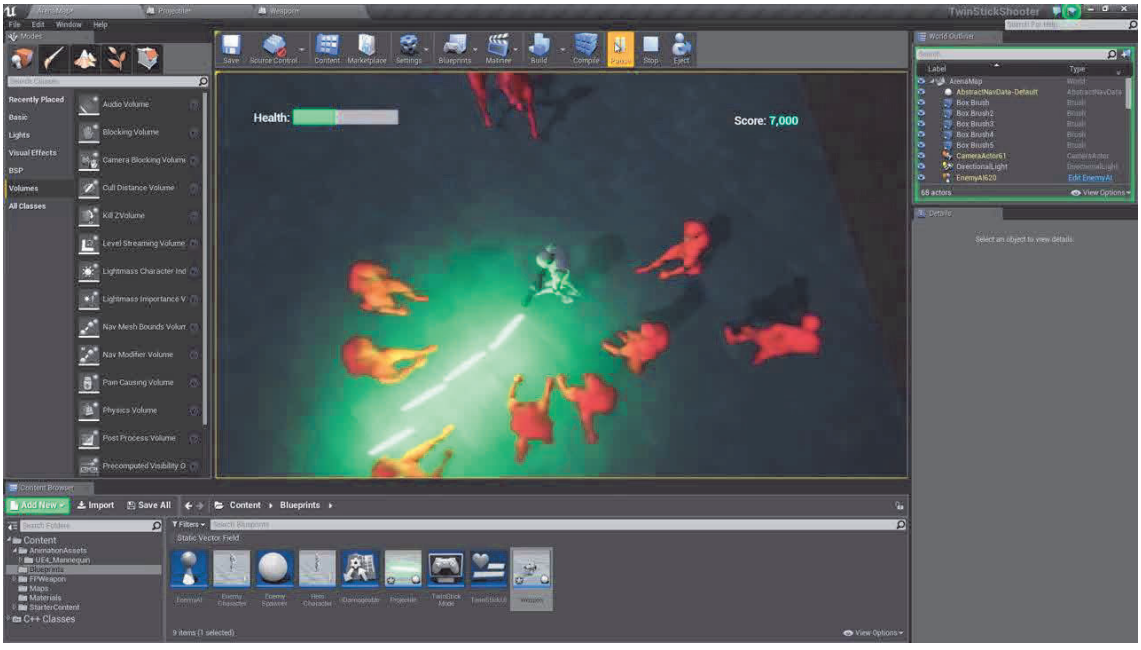
6.5 Análisis

Inicialmente, se pensó en una idea básica del concepto del proyecto. Sería un videojuego de lucha donde 1 o 2 jugadores podían luchar entre sí o contra una inteligencia artificial.

Después, se buscó información que tratara sobre inteligencia artificial en contrincantes de un videojuego de lucha. Este tipo de autómatas deben de ser muy rápidos a la hora de realizar sus acciones. Si el jugador le está golpeando, éste debería de cubrir sus golpes rápidamente, o si su rival no está haciendo ninguna acción debería aprovechar el momento para acercarse y golpearle.

Pero tras una larga búsqueda en foros, repositorios y documentación de sistemas inteligentes para tener una idea de base, no se logró encontrar nada útil sobre el tema. Esto supuso un problema a la hora de estimar el tiempo de desarrollo para la inteligencia artificial, ya que en un principio no se pensó en la complejidad que podría o no llegar a tener.

Posteriormente, se decidió aprender sobre el entorno de desarrollo **Unreal Engine**, una plataforma con una curva de aprendizaje moderada, pero que garantiza buenos resultados. Para familiarizarse con el entorno, se siguió una serie de tutoriales oficiales de **Unreal** y se creó una prueba base de un videojuego de vista superior en el que un personaje disparaba a su alrededor a numerosos enemigos que le seguían, llamado **Twin Stick Shooter**.



Proyecto de Unreal Engine llamado Twin Stick Shooter

Al comprobar la complejidad del entorno y de la documentación, los requisitos de ordenador necesarios para este tipo de proyectos y la dificultad para encontrar recursos de calidad para utilizar, se decidió descartar **Unreal Engine** como plataforma de desarrollo.

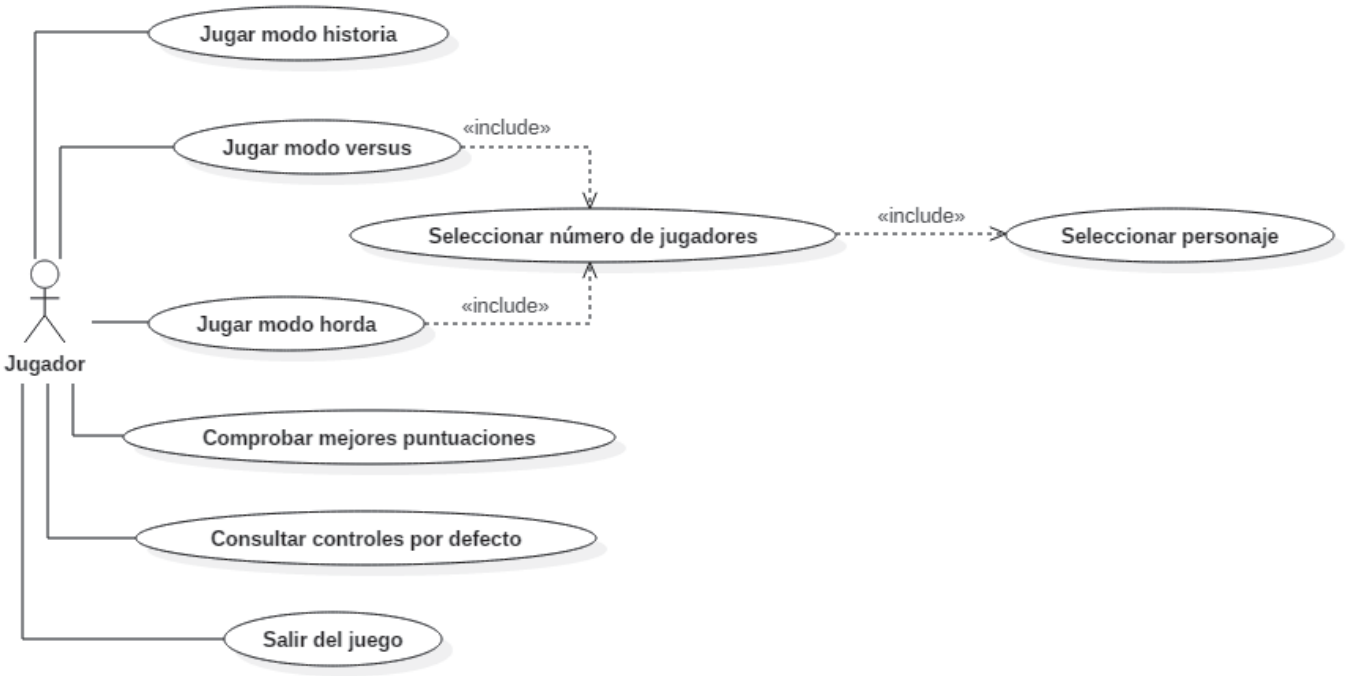
Luego, se procedió a probar el entorno **Unity**. Aquí la experiencia fue más satisfactoria, pues en cuestión de unos pocos minutos había un pequeño prototipo en el que el personaje se podía mover por el escenario.

Finalmente, se decidió utilizar **Unity** debido a la extensa comunidad y documentación que existía, y a la facilidad para usar imágenes como recursos propios del videojuego, sin necesidad de modelos ni animaciones 3D, ya que el soporte de **Unreal Engine** para 2D era muy limitado, y la calidad gráfica en nuestro Trabajo de Fin de Grado no es un apartado importante.

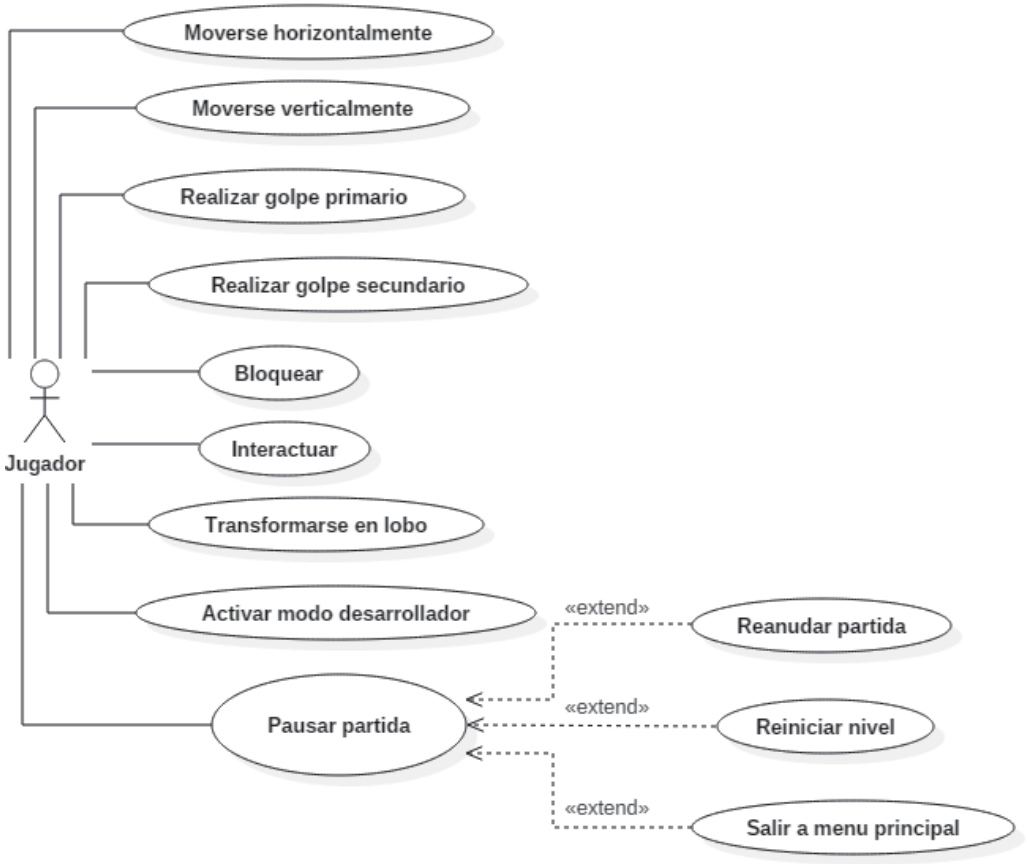
6.6 Diseño

A continuación, se detalla un conjunto de **diagramas de casos de uso** de cada aspecto del videojuego.

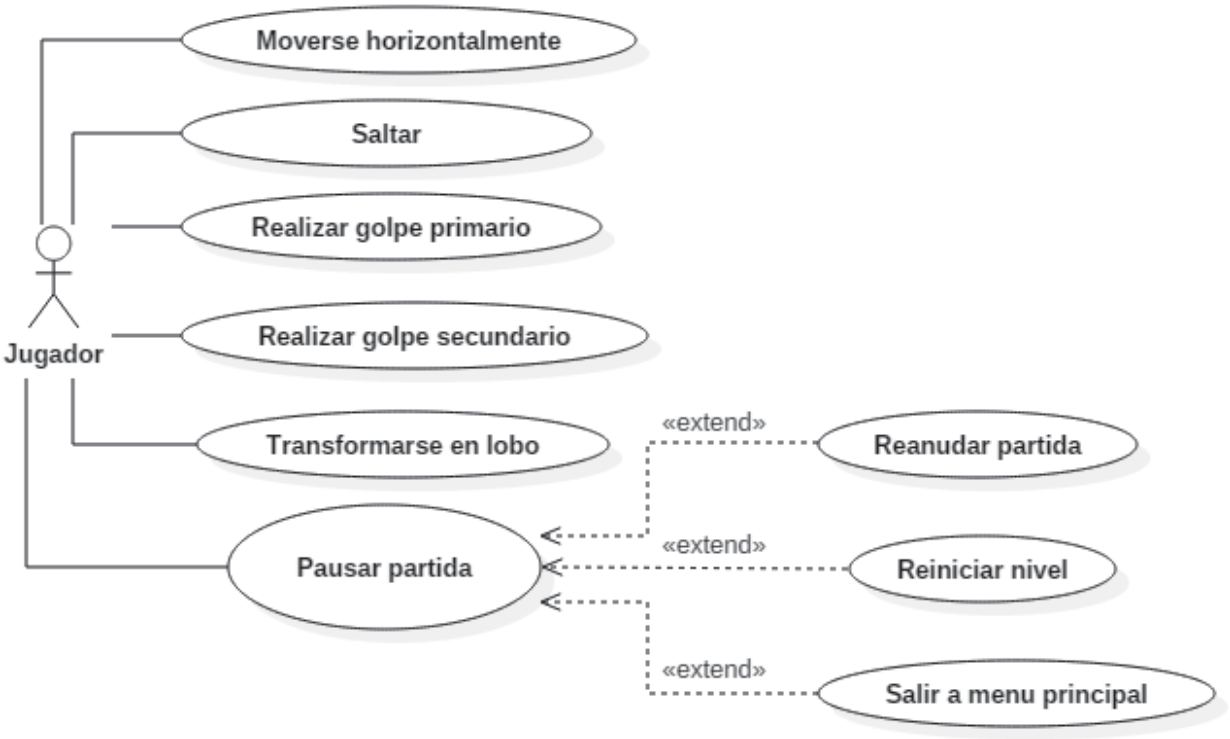
- Menú principal:



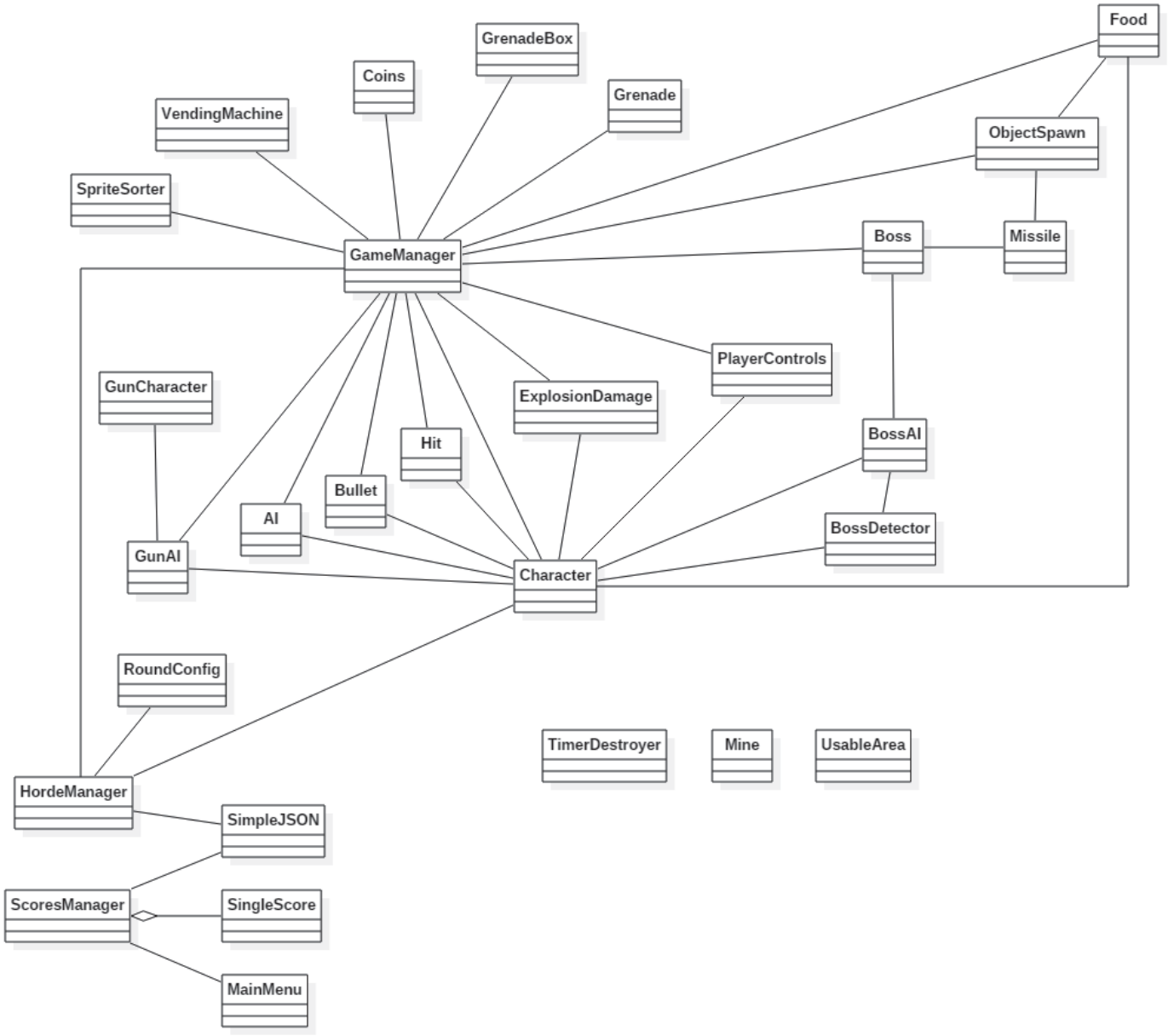
- Modo Historia/Modo Horda:



- Modo Versus


Moverse horizontalmente
Saltar
Realizar golpe primario
Realizar golpe secundario
Transformarse en lobo
Pausar partida
Jugador
«extend»
Reanudar partida
«extend»
Reiniciar nivel
«extend»
Salir a menu principal

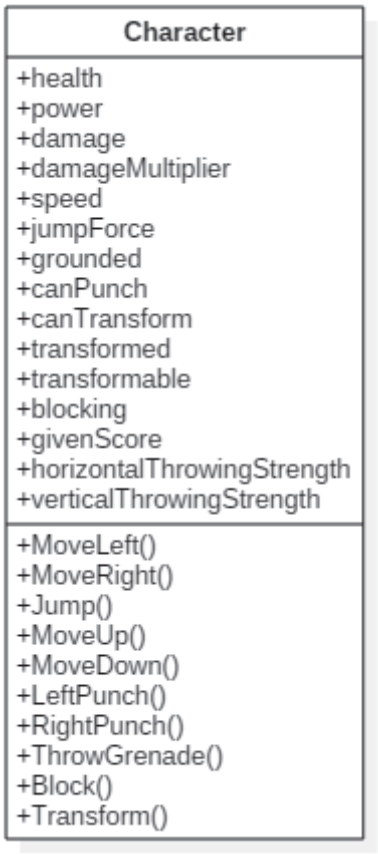
El **diagrama de clases** correspondiente al videojuego puede verse a continuación:



Podemos comprobar que las clases **Character** y **GameManager** son las que más interactúan con las demás, por lo que son las más importantes y las que más funcionalidades tienen. A continuación, se explicarán más en detalle.

6.6.1 Clase Character

Esta clase representa a un personaje, por lo que tendrá los atributos y funcionalidades que los personajes poseen en el videojuego.



Atributos:

- *health*: Los puntos de salud del personaje. Si llegan a cero, el personaje es vencido.
- *power*: Los puntos de poder del personaje. Si llegan a 100, el personaje podrá transformarse en lobo y obtener mejoras en combate.
- *damage*: Daño base que causa el personaje. El daño que el personaje causa a los demás se calcula como el daño base por el multiplicador de daño.
- *damageMultiplier*: Multiplicador del daño que causa el personaje. El daño que el personaje causa a los demás se calcula como el daño base por el multiplicador de daño.
- *speed*: Velocidad a la que se desplaza el personaje por el escenario.
- *jumpForce*: Fuerza aplicada a los saltos del personaje. A mayor valor, mayor es la altura que salta.
- *grounded*: Valor que nos indica si el personaje está en el suelo o está saltando. Utilizado en el modo versus.
- *canPunch*: Valor que nos indica si el personaje puede golpear. Nos permite saber si está golpeando en un momento dado.
- *canTransform*: Valor que nos indica si el personaje se puede transformar

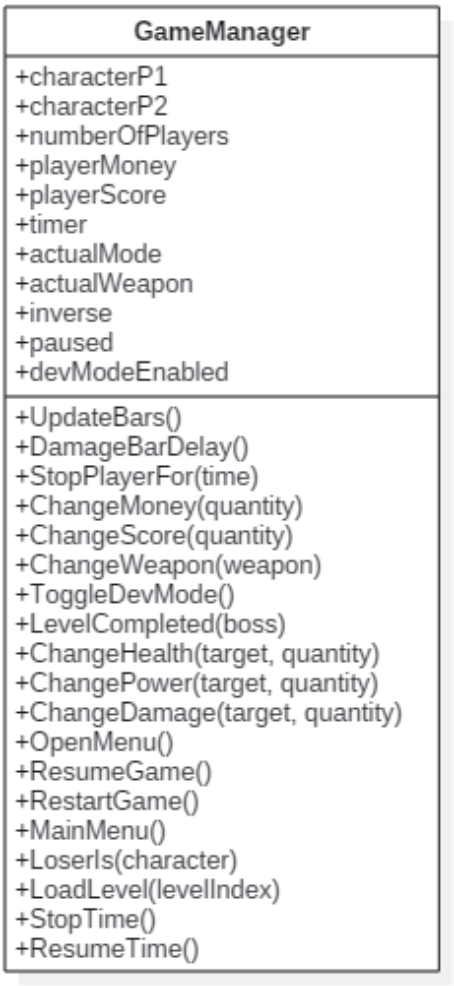
- *transformed*: Valor que nos indica si el personaje está transformado.
- *transformable*: Valor que nos indica si el personaje puede transformarse.
- *blocking*: Valor que nos indica si el personaje está bloqueando.
- *givenScore*: Puntuación que otorga el personaje al ser vencido. Utilizado en los enemigos del modo historia y horda.
- *horizontalThrowingStrength*: Fuerza horizontal con la que se lanzan las granadas.
- *verticalThrowingStrength*: Fuerza vertical con la que se lanzan las granadas.

Métodos:

- *MoveLeft()*: El personaje se mueve hacia la izquierda.
- *MoveRight()*: El personaje se mueve hacia la derecha.
- *Jump()*: El personaje salta en el modo versus.
- *MoveUp()*: El personaje se desplaza hacia arriba en el modo historia y horda.
- *MoveDown()*: El personaje se desplaza hacia abajo en el modo historia y horda.
- *LeftPunch()*: El personaje realiza un puñetazo izquierdo.
- *RightPunch()*: El personaje realiza un puñetazo derecho.
- *ThrowGrenade()*: El personaje lanza una granada.
- *Block()*: El personaje bloquea.
- *Transform()*: El personaje se transforma.

6.6.2 Clase GameManager

En esta clase se recogen todas las funcionalidades y características que son generales en el juego, y se gestionan todos los eventos relacionados a las partidas.



Atributos:

- *characterP1*: Indica el personaje escogido por el jugador 1.
- *characterP2*: Indica el personaje escogido por el jugador 2.
- *numberofPlayers*: Número de jugadores humanos en la partida.
- *playerMoney*: Cantidad de dinero obtenido por el jugador en el modo historia y horda.
- *playerScore*: Puntuación obtenida por el jugador en el modo historia y horda.
- *timer*: Temporizador de la partida.
- *actualMode*: Modo de juego actual.
- *actualWeapon*: Arma actual del jugador.
- *inverse*: Indica si los personajes están en su orientación inicial o no.
- *paused*: Indica si la partida está en pausa.
- *devModeEnabled*: Indica si el modo desarrollador está activado.

Métodos:

- *UpdateBars()*: Actualiza las barras de salud y poder a los valores actuales.
- *DamageBarDelay()*: Actualiza la barra de daño con un pequeño retraso.
- *StopPlayerFor(time)*: Detiene durante un tiempo un jugador de la partida y le impide realizar acciones.
- *ChangeMoney(quantity)*: Cambia el dinero del jugador en una cantidad especificada y actualiza la interfaz.
- *ChangeScore(quantity)*: Cambia la puntuación del jugador en una cantidad especificada y actualiza la interfaz.
- *ChangeWeapon(weapon)*: Cambia el arma actual del jugador por una especificada y actualiza la interfaz.
- *ToggleDevMode()*: Activa o desactiva el modo desarrollador. Esto incluye mostrar el área donde caerán los misiles, mostrar el rango de detección de los enemigos, mostrar área de las balas y potenciar los parámetros del jugador.
- *LevelCompleted(boss)*: Completa el nivel actual venciendo al jefe especificado.
- *ChangeHealth(target, quantity)*: Cambia los puntos de salud del personaje especificado por una cantidad dada. Si la cantidad es positiva se trata de una curación, por lo tanto, se reproducen los efectos y sonidos de curación, y si la cantidad es negativa se trata de un daño, por lo que se reproducen el sonido y la animación de daño. Si se trata de daño y los puntos de salud han llegado a cero, el personaje objetivo es derrotado, siendo el fin de la partida si se trata del jugador u obteniéndose puntuación y dinero si se trata de un enemigo.
- *ChangePower(target, quantity)*: Cambia los puntos de poder del personaje especificado por una cantidad dada. Si llegan a 100, el personaje se puede transformar.
- *ChangeDamage(target, quantity)*: Cambia el daño base del personaje especificado por una cantidad dada.
- *OpenMenu():* Abre el menú de pausa en la partida y detiene el tiempo.
- *ResumeGame()*: Reanuda la partida desde el menú de pausa.
- *RestartGame()*: Reinicia la partida.
- *MainMenu()*: Vuelve al menú principal.
- *LoserIs(character)*: Establece que jugador ha perdido la partida. Se muestra el letrero de quien es el ganador y se abre el menú.
- *LoadLevel(levelIndex)*: Carga el nivel especificado.
- *StopTime()*: Simula que el tiempo se para, deteniendo y anulando las físicas de los personajes. Utilizado a la hora de realizar la animación de la transformación.
- *ResumeTime()*: Simula que el tiempo se reanuda, restableciendo las físicas de los personajes. Utilizado a la hora de finalizar la animación de la transformación.

7 IMPLEMENTACIÓN

7.1 Modo versus

El objetivo inicial fue crear lo básico para que el juego fuese controlable por el jugador, teniendo en mente un **modo versus** de jugador contra jugador. Las primeras tareas que se tuvieron en cuenta fueron:

- Movimiento de personajes.
- Acciones de los personajes.
- Mecánicas de juego.

Primero se creó un escenario vacío muy básico, que consistía en una plataforma que mantendría a los personajes mientras se mueven sobre ella. Después se procedió a crear un personaje, el cual sería el mismo para ambos jugadores. Se creó un **GameObject** que tenía una imagen estática, se le añadieron físicas y se le añadió un script para poder controlarlo. Este sería el script de **PlayerControls**, el cual contiene todos los controles del juego asociados a los métodos de cada acción.

Una vez podíamos mover a estos personajes, se crearon las acciones básicas que podían realizar los personajes. Se creó el script **Character**, que incluye los métodos para las diferentes acciones de los personajes, como por ejemplo saltar, golpear o el movimiento.

Luego se realizaron las mecánicas del juego. Estas incluyen:

- **Parámetros de personajes**: puntos de salud, de energía, daño, velocidad, etc.
- **Daño a los golpes**: el puño del personaje colisiona con el cuerpo del otro jugador y le causa daño. Esto se realiza en el script **Hit**.
- **Barras de vida y poder**: en la parte superior aparecen unas barras que muestran la cantidad de salud y energía de los personajes y se actualizan cada vez que estos parámetros cambian.
- **Menús**: se crearon el menú principal y el menú de pausa, los cuales nos permiten navegar entre las escenas y realizar acciones como reiniciar la partida. Fueron necesarias algunas funciones para el menú principal que están en el script **MainMenu**, que cambian entre escenas y modifican la interfaz para mostrar las pantallas adecuadas.
- **Animaciones**: se añadieron las distintas animaciones que tiene el personaje, como la animación al correr, al saltar, al golpear, etc. Esto se consigue cambiando progresivamente el **sprite** del personaje. También se añadió una animación al fondo del escenario y del menú principal.
- **Inteligencia artificial**: se diseñó una inteligencia artificial para el otro personaje en el script **AI**, que utiliza los métodos del script **Character** para realizar su comportamiento.
- **Sistema de resultado de la partida**: cuando los puntos de salud de uno de los personajes llegan a cero, se llama a la función *LoserIs* que se encuentra en el script **GameManager**, y le pasa el jugador que ha perdido para que se muestre el resultado de la partida.

7.2 Modo historia

Tomando como base el modo versus, se procedió a implementar el **modo historia**.

- Se creó una escena nueva con un escenario mucho más grande que la plataforma del modo versus.
- Se eliminaron las físicas que tenían los personajes en el modo versus, por lo que los personajes no caerían por efecto de la gravedad, sino que los moveremos verticalmente con los controles, para así crear un efecto de profundidad en el escenario.
- Se le puso un área de colisión en la parte inferior del cuerpo del personaje para poder controlar el área en el que se podía mover, y además de la plataforma que soportaba a los jugadores en modo versus, se añadió otra un poco más arriba para impedir que suban a partir de cierto punto.

Se pueden ver los límites superior e inferior que impiden a los personajes salirse de la zona

- **Enemigos cuerpo a cuerpo**:

Luego se añadieron más personajes, que son los enemigos que el jugador va encontrando a lo largo del nivel. Éstos comienzan a seguir al jugador cuando está a cierto rango de distancia, por lo que hubo que hacer modificaciones en la inteligencia artificial de los enemigos, además de añadir la posibilidad de moverse verticalmente. Al detectar al jugador, los enemigos se acercan y actúan de manera similar al contrincante del modo versus. Cuando son derrotados, los enemigos dejan en el suelo un montón de monedas con una cantidad aleatoria de dinero, que se calcula en el script **Coins**.

- **Enemigos a distancia**:

Por otro lado, se creó otro tipo de enemigo que ataca a distancia, por lo que se generó otro tipo de inteligencia artificial distinta a la de los enemigos normales, el script **GunAI**. La acción de disparar se implementó en otro script llamado **GunCharacter**, y se hizo generando una bala a la altura de la pistola del personaje y aplicándole una cierta fuerza. Estos enemigos se acercan hasta mantener cierta distancia con el jugador, y le disparan si está a tiro. Si el jugador se acerca a ellos, retroceden unos pasos para mantener la distancia.

Hubo que crear un script nuevo para las balas llamado **Bullet**, que causa daño al jugador al entrar en contacto con ellos o se destruye al entrar en contacto con otra superficie, además de generar los sonidos de efectos especiales.

- **Trampas**:

Se implementaron trampas, como los misiles que caen desde el cielo y el jugador debe de esquivar para no recibir daño, o como las minas, que son estáticas y explotan al entrar en contacto con el jugador. Al activarse, estos generan una explosión que tiene el script **ExplosionDamage**, que, si detecta una colisión con algún tipo de personaje, le causa una cantidad de daño. Los misiles disponen del script **Missile**, que hace que, al generarse un misil, se genere a su vez una plataforma en el suelo en la que impactará el misil. Esto es para crear efecto de profundidad y que el misil no explote siempre en la misma posición del suelo. Al detectarse una colisión con una de estas plataformas o con algún personaje, generará una explosión y el script **ExplosionDamage** se encargará de realizar el daño si fuese necesario.

Plataformas de impacto de dos misiles distintos que caerán en posiciones diferentes.

Los misiles se generan en el cielo mediante un GameObject que tiene el script **ObjectSpawn**. Se obtiene la posición actual del jugador y se calcula al azar dentro del rango especificado una posición en la que se posicionará el objeto generado, un misil en este caso. También se puede especificar un rango de tiempo en el que se quiere que aparezcan los objetos generados.

Las minas tienen el script **Mine**, que comprueba si se produce una colisión con el jugador para generar una explosión, y el script **ExplosionDamage** se encargará de realizar el daño si fuese necesario.

- **Máquinas expendedoras**:

Se crearon máquinas expendedoras que proporcionan comida a cambio de una cantidad del dinero. En el script **VendingMachine**, cuando el jugador pulsa el botón de interactuar, se comprueba si el jugador tiene el dinero necesario, y si lo tiene, le resta la cantidad de dinero y genera una hamburguesa a sus pies. La comida tiene el script **Food**, que al entrar en contacto con el jugador le aumenta los puntos de salud en una cantidad dada. Posteriormente, el script se modificó y aparecieron 3 tipos de comida, dependiendo del atributo que aumentan: la salud, el poder o el daño. Para detectar si el jugador está cerca para poder interactuar con la máquina, se creó el script **UsableArea**, que muestra la tecla del botón interactuar sobre el GameObject.

- **Cajas de granadas**:

A lo largo del escenario se añadieron cajas de granadas. Estos objetos incluyen el script **GrenadeBox**, que comprueba si el usuario pulsa la tecla de interactuar para asignar que el arma actual del jugador es una granada. Solo se puede recoger una granada cada vez, y al coger una granada la caja queda vacía durante un tiempo. Luego el jugador puede lanzar las granadas y se genera una granada que cae delante del jugador. Al igual que las máquinas expendedoras, disponen del script **UsableArea**. Las granadas tienen el script **Grenade**, que hace una cuenta atrás y al acabar genera una explosión en el lugar donde estaba la granada.

- **Interfaz**:

Se creó un temporizador que va en aumento e indica al jugador el tiempo que lleva en la partida en la parte superior de la pantalla. Al ser una mecánica de juego, esto se realiza en el script **GameManager**.

Se añadió a la interfaz un indicador de puntuación. Al vencer cada enemigo se obtiene una puntuación que se acumula a lo largo del nivel, y que viene determinada por el tipo de enemigo vencido, y se muestra en el indicador superior derecho. Al ser una mecánica de juego, esto se realiza en el script **GameManager**.

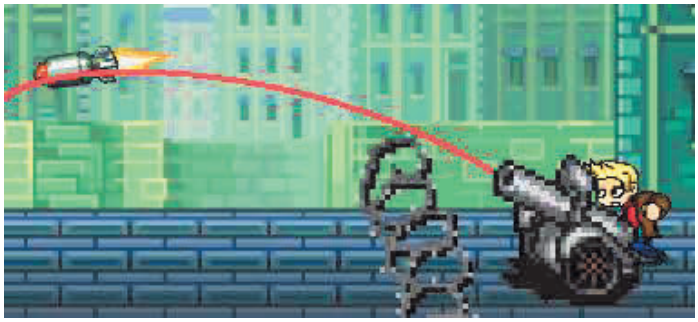
- **Jefe final**:

Luego se implementó otro tipo de enemigo, el jefe final del nivel. Este enemigo tiene el script **Character** como todos los personajes, y se creó el script **Boss** para las características más individuales o exclusivas del jefe final. El jefe final tiene un cañón que usa para atacar al jugador, y este cañón puede estar en posición baja, media o alta. Por tanto, se han creado funciones para que el jefe pase de una posición a otra adecuadamente con sus respectivas animaciones. Estas transiciones son las siguientes: de posición baja a media, de baja a alta, de media a baja, de media a alta, de alta a media, y de alta a baja.

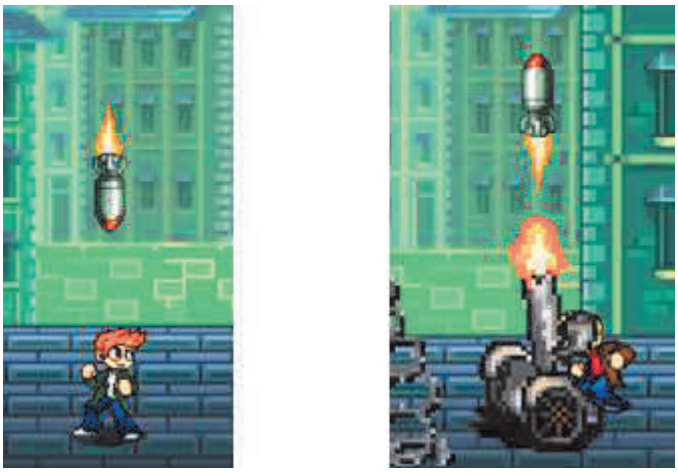
Diferentes posiciones del cañón del jefe final (media, baja y alta)

Por otro lado, dependiendo de la posición en la que el jefe tenga su cañón, realizará distintos tipos de disparos con cañón. Si el cañón está en posición baja, el disparo generará un misil que sale en línea recta hacia el jugador. Si el cañón está en posición media, los misiles saldrán en diagonal y caerán con forma parabólica. La fuerza con la que se disparan estos misiles se calcula dependiendo de la distancia que tenga con el jugador. Si el cañón está en posición alta, los misiles irán directos hacia el cielo, para que, pasados unos segundos, se generen en caída libre por encima del jugador.

Disparo con cañón en posición baja



Disparo con cañón en posición media



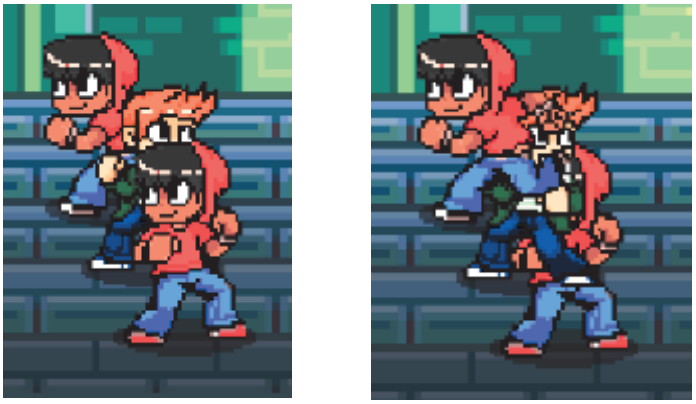
Disparo con cañón en posición alta

Para controlar todo este tipo de acciones, se diseñó su comportamiento con una inteligencia artificial nueva que está en el script **BossAI**.

Se creó una zona invisible situada antes del combate contra el jefe para activar este evento. En el script **BossDetector**, se detecta la presencia del jugador y se aleja la cámara de juego, se cambia la música, se desactiva la aparición de misiles desde el cielo, se activa la inteligencia artificial del jefe final, se muestra el diálogo del jefe y se activa una barrera invisible que impide al jugador salir del combate.

- **Gestión de capas de sprites**:

Se diseñó un script llamado **SpriteSorter** que dependiendo de la altura física a la que se encuentra el GameObject que tiene este script, modificaba la capa a la que se muestra el sprite del GameObject. Esto es para lograr una mayor capacidad de profundidad de los elementos que se dibujan en pantalla, como personajes, misiles, decorado, etc.



A la izquierda podemos ver como los personajes se superponen correctamente dependiendo de la profundidad a la que se encuentran. A la derecha podemos verlo incorrectamente.

- **Sistema de puntuaciones**:

Se creó una pantalla de puntuaciones que aparece cuando el jugador supera el nivel o cuando éste pierde la partida. Para calcular la puntuación final, se tienen en cuenta los siguientes parámetros:

- Puntos de salud del jugador
- Puntos de poder del jugador
- Dinero al terminar la partida
- Puntuación actual

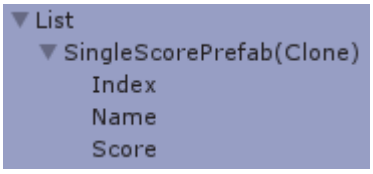
El jugador puede decidir entonces si quiere guardar esa puntuación en el sistema, introduciendo su nombre para ello. Las puntuaciones quedan guardadas en un archivo JSON que tiene la siguiente estructura:


```
{
  "scores": {
    0: {
      "name": "AAA",
      "score": 200
    },
    1: {
      "name": "BBB",
      "score": 100
    },
    2: {
      "name": "CCC",
      "score": 300
    },
    3: {
      "name": "DDD",
      "score": 450
    },
    4: {
      "name": "EEE",
      "score": 500
    },
    5: {
      "name": "FFF",
      "score": 150
    },
    6: {
      "name": "GGG",
      "score": 250
    },
    7: {
      "name": "HHH",
      "score": 550
    },
    8: {
      "name": "III",
      "score": 350
    },
    9: {
      "name": "JJJ",
      "score": 600
    }
  }
}
```

Cada vez que un usuario añade su puntuación, se añade un registro que contiene su nombre y la puntuación obtenida.

Cuando el jugador accede al menú de mejores puntuaciones, el sistema obtiene las 10 mayores y las muestra en pantalla ordenadas de mayor a menor.

Toda la gestión de puntuaciones se hace en el script **ScoresManager**, se carga y se trata el JSON utilizando para ello el plugin **SimpleJSON** y nuestra clase **SingleScore**, que es necesaria para crear objetos con la estructura que nos aporta **SimpleJSON**. Un objeto de tipo **SingleScore** solamente tiene los campos *rank*, que nos indica la posición de la puntuación, *playerName*, que indica el nombre del jugador, y *score*, que almacena su puntuación. El script **SingleScore** irá asociado a un GameObject que representa una de las puntuaciones en la lista de mejores puntuaciones, por lo tanto, ese GameObject a su vez contiene los textos de posición o rango de la puntuación, el nombre del jugador, y su puntuación. Fue necesario añadir una función al script **SingleScore** que actualizara estos textos al crear una puntuación.



En la jerarquía de Unity tenemos un GameObject que es la lista de puntuaciones y que almacena todas las puntuaciones. Cada vez que se carga una de las 10 mejores puntuaciones, se hace una copia del GameObject *SingleScorePrefab*, que contiene los textos de la posición o rango de la puntuación, el nombre del jugador, y su puntuación, y se añade a esta lista.

Uno de los GameObject que representa una puntuación y que contiene el script SingleScore

- **Modo desarrollador:**

Durante la etapa de testeo, se perdía mucho tiempo avanzando por el nivel para comprobar las funcionalidades añadidas o los cambios realizados. Para una mayor comodidad y rapidez, se pensó en implementar un **modo desarrollador**, que al ser activado hace que el personaje gane una serie de características que facilitarían las labores de pruebas. Estas mejoras consisten en:

- El personaje es invencible, no recibe daño de los enemigos ni de las trampas.
- El personaje gana mucho daño con sus golpes.
- El personaje obtiene el máximo de poder, con lo que se puede transformar al instante.
- El personaje se mueve mucho más rápido.
- El dinero del jugador aumenta en 10$.
- Se visualiza con un contorno rojo el área de detección de los enemigos.
- Se visualiza con una plataforma roja la zona en la que caerán los misiles.
- Se visualiza con un contorno azul las balas disparadas por los enemigos con pistola.

7.3 Modo horda

Tomando algunas características del modo historia, se creó el modo horda. El escenario pasa a ser del ancho de la pantalla como en el modo versus, pero manteniendo el aspecto de profundidad con el movimiento vertical del modo historia. El temporizador se cambió para que lugar de ser ascendente, fuera disminuyendo, partiendo desde una cierta cantidad. Si el temporizador llega a cero, se acaba la partida.

Fuera de la cámara y a ambos lados, hay dos GameObject que se encargarán de generar los enemigos. Los distintos parámetros utilizados en este modo son almacenados en un archivo **JSON**, que tiene la siguiente estructura:

```
{
  "rounds": {
    "1": {
      "time": 30,
      "punchEnemy": 2,
      "gunEnemy": 1,
      "heal": false
    },
    "2": {
      "time": 35,
      "punchEnemy": 3,
      "gunEnemy": 1,
      "heal": true
    },
    "3": {
      "time": 40,
      "punchEnemy": 4,
      "gunEnemy": 2,
      "heal": false
    },
    "4": {
      "time": 45,
      "punchEnemy": 5,
      "gunEnemy": 2,
      "heal": true
    },
    "5": {
      "time": 50,
      "punchEnemy": 6,
      "gunEnemy": 2,
      "heal": true
    }
  }
}
```

Aquí almacenamos la información referente a cada ronda, como el tiempo que dura cada una, los enemigos que aparecen de cada tipo, y un parámetro que nos indica si esa ronda proporciona una pequeña curación a los puntos de salud del personaje.

Para tratar este archivo JSON, se creó el script **HordeManager**, que al igual que **ScoresManager**, lo hace utilizando el plugin **SimpleJSON**.

Una vez empieza la partida en el modo horda, se obtienen los parámetros de la ronda actual y se realizan los siguientes ajustes:

- Se establece el tiempo que corresponde a la ronda y se activa la cuenta atrás.
- Se toma el número de enemigos del primer tipo y se generan aleatoriamente en el GameObject de la izquierda o de la derecha.
- Se toma el número de enemigos del segundo tipo y se generan aleatoriamente en el GameObject de la izquierda o de la derecha.
- Se comprueba si en esa ronda se debe dar al jugador una bonificación de puntos de salud.

En una variable se almacena el número de enemigos restantes en la ronda actual, que corresponde con la suma de los enemigos de ambos tipos. Cuando este número llega a cero, se pasa a la siguiente ronda.

Si el temporizador o los puntos de salud del jugador llegan a cero, es el fin de la partida y aparece la pantalla con su puntuación.

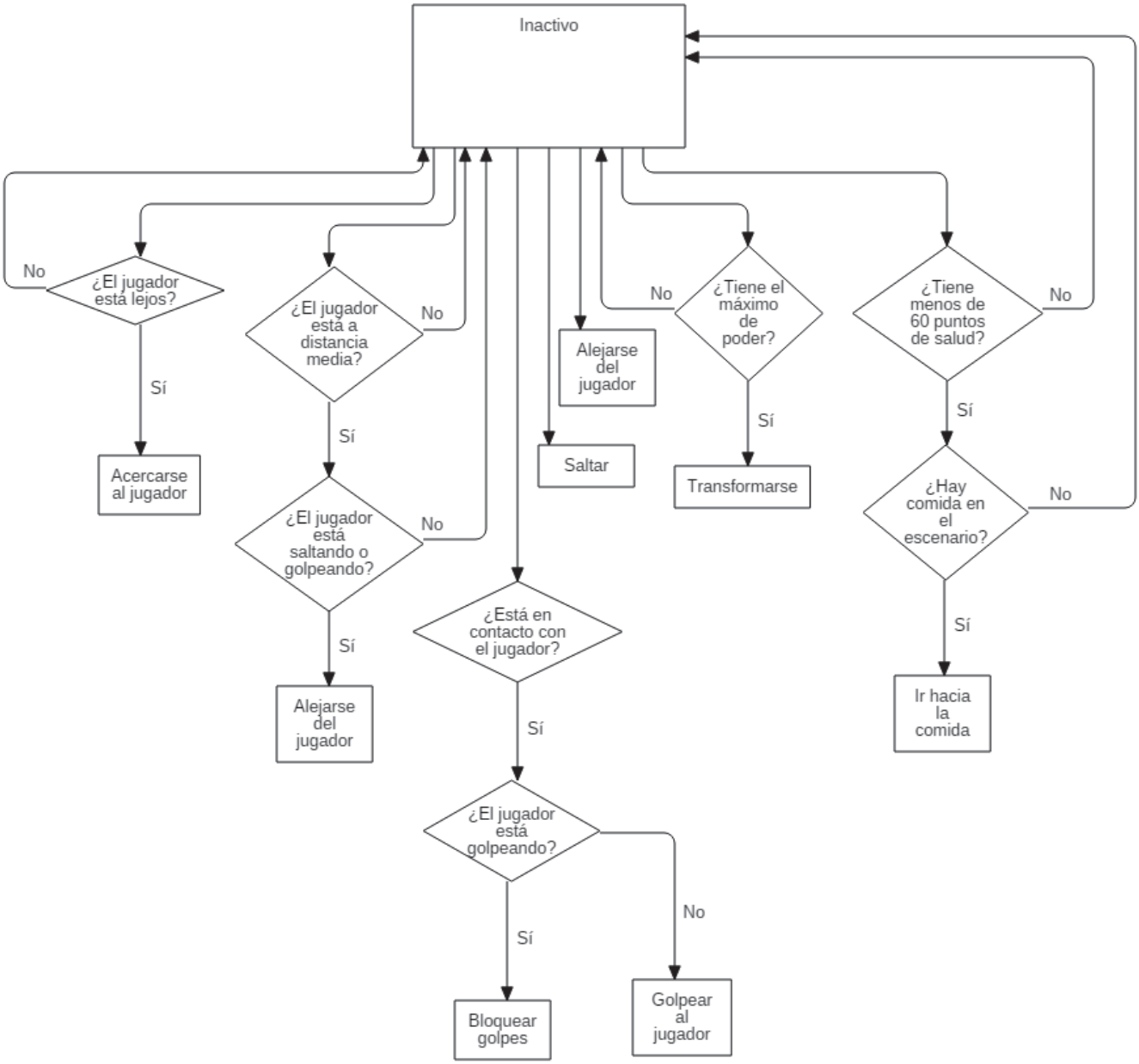
7.4 Inteligencia artificial

Como hemos dicho anteriormente, el videojuego tiene 3 tipos de inteligencia artificial diferentes: la del rival en el modo versus y la de los enemigos cuerpo a cuerpo en el modo historia (script **AI**), la de los enemigos con pistola del modo historia (script **GunAI**) y la del jefe final del modo historia (script **BossAI**).

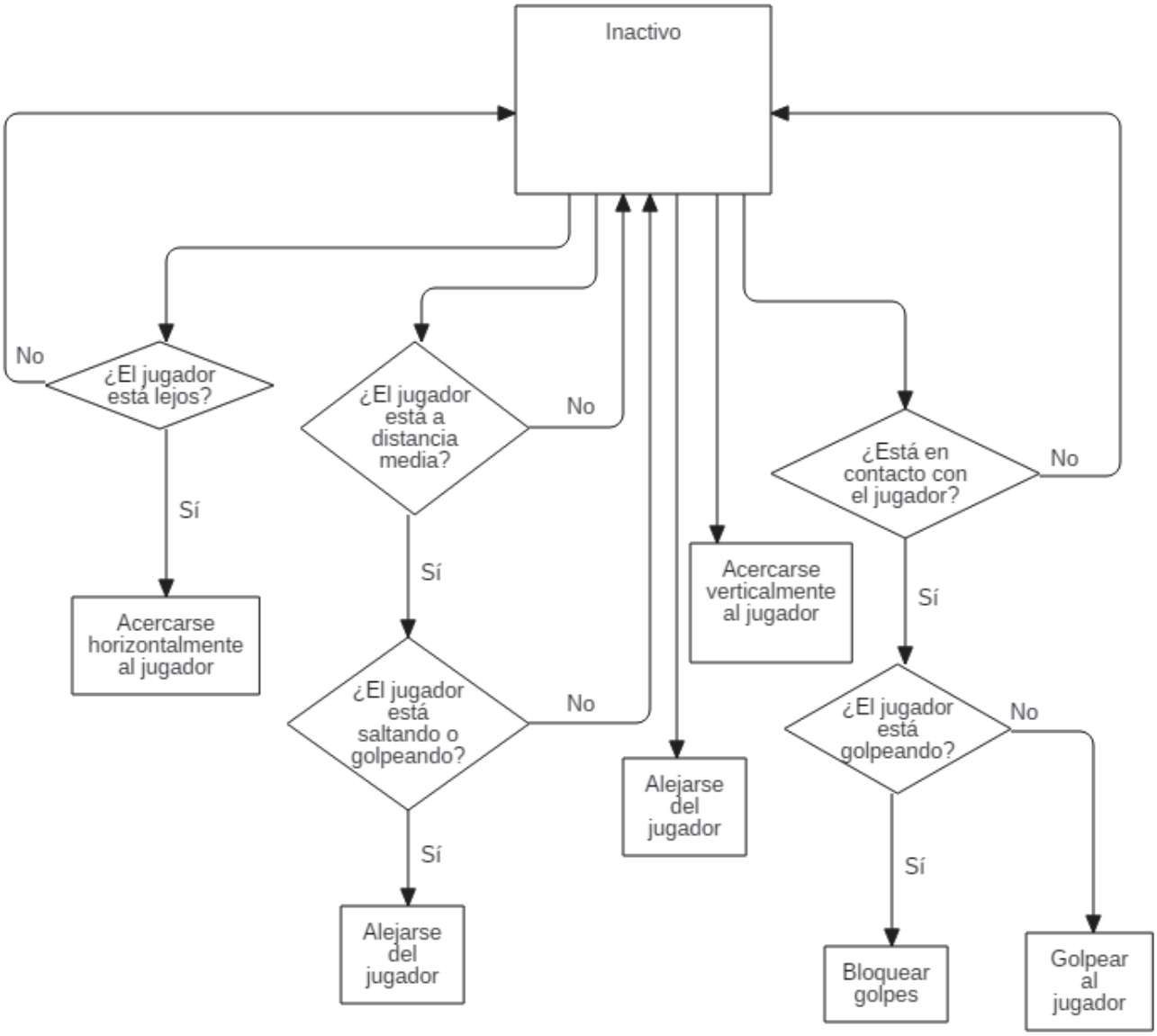
7.4.1 Enemigos cuerpo a cuerpo

El contrincante en el modo de juego versus y los enemigos cuerpo a cuerpo del modo historia tienen la misma inteligencia artificial, aunque hay pequeños ajustes que se tienen en cuenta con los enemigos del modo historia.

A continuación, se presenta el diagrama de flujo del comportamiento de la inteligencia artificial del contrincante en el modo versus:



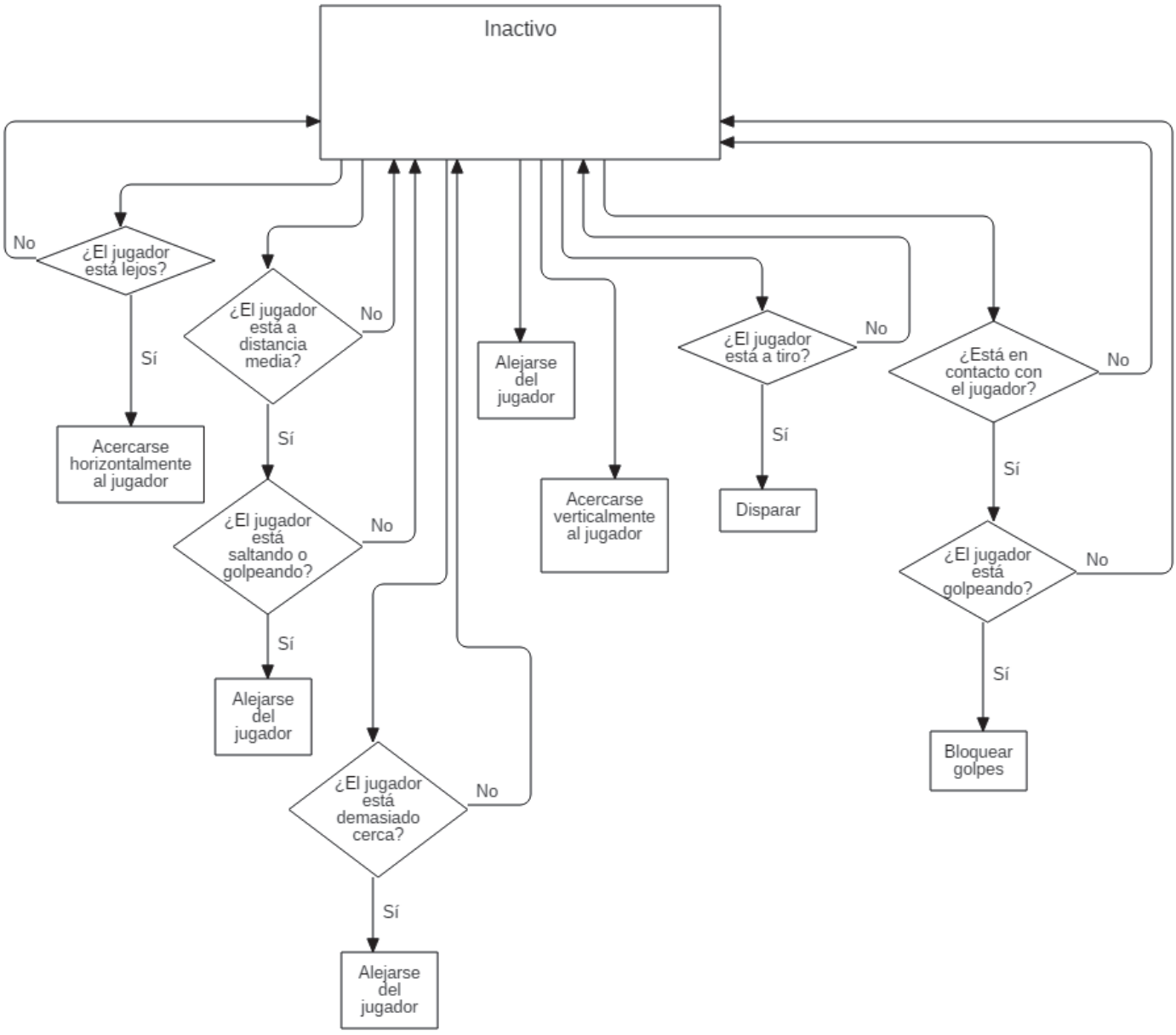
Con ligeros cambios, aquí se expone el diagrama de flujo de la inteligencia artificial de los enemigos cuerpo a cuerpo del modo historia:



7.4.2 Enemigos con pistola

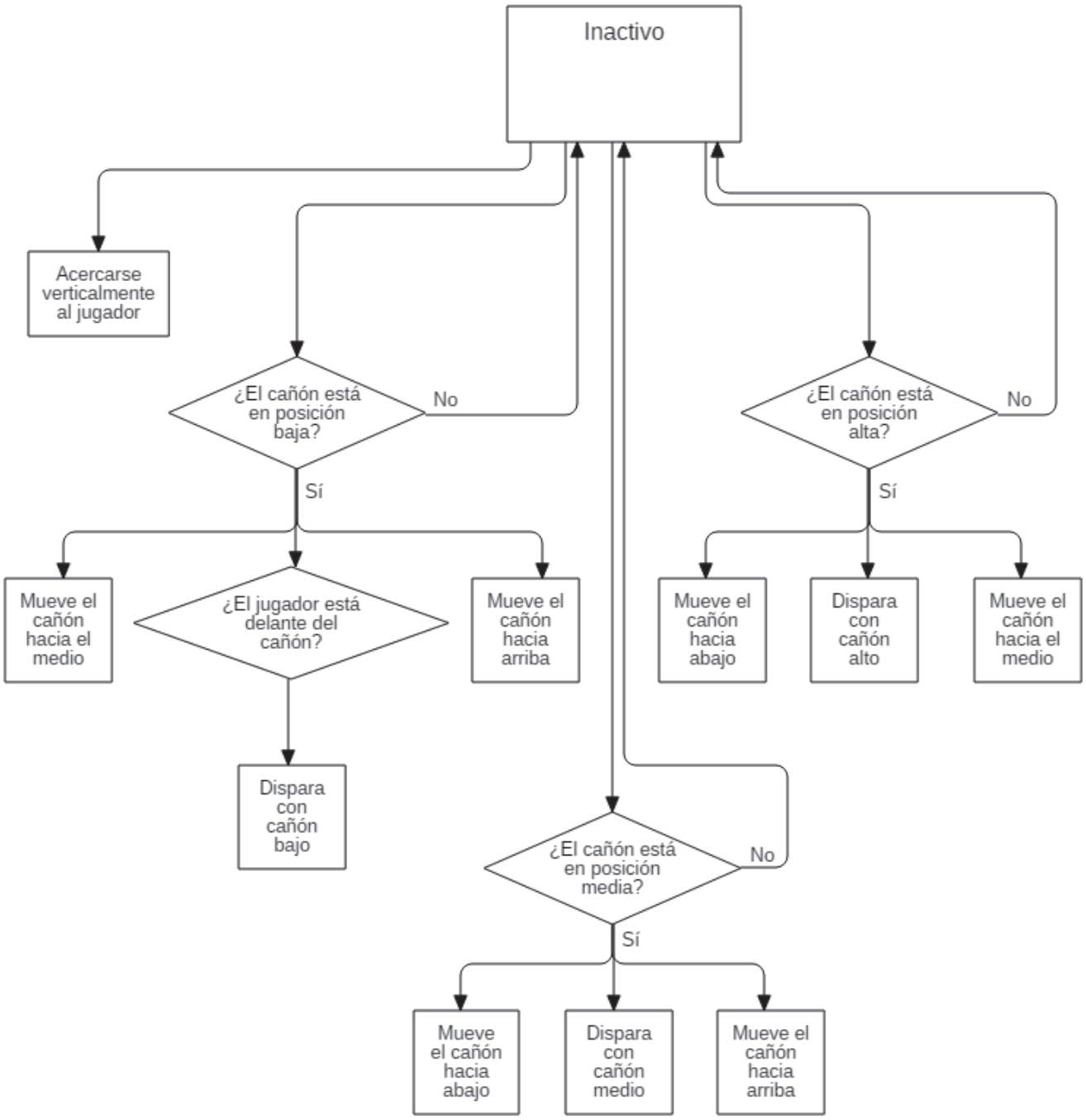
Los enemigos que atacan a distancia con una pistola en el modo historia tienen otra inteligencia artificial diferente a los anteriores.

A continuación, se muestra cómo es su comportamiento con un diagrama de flujo:



7.4.3 Jefe final

Por último, el jefe final que el jugador debe vencer en el modo historia tiene otra inteligencia artificial.



7.5 Evaluación y pruebas

Se realizaron frecuentemente diversos prototipos del videojuego que se entregó a **4 usuarios**, los cuales nos sirvieron para validar las funcionalidades que habíamos implementado y para realizar cambios que se consideraban necesarios.

El primer prototipo consistió en un juego que solo disponía del modo versus. Aun no se había implementado la inteligencia artificial, pero **2 jugadores** podían jugar entre sí. El objetivo era validar la jugabilidad y mecánicas que se habían implementado.

Cuando se implementó la **inteligencia artificial**, se hicieron nuevos prototipos que nos ayudaron a pulirla. Los usuarios nos indicaron que era muy fácil ganarle, ya que había momentos en los que se quedaba quieto durante mucho tiempo. Reducir estos tiempos de espera en el script de inteligencia artificial nos ayudó a arreglar esto.

Se encontró un problema con la representación de la **interfaz** en **distintas resoluciones**. Uno de los usuarios se dio cuenta de que la resolución cambiaba la posición relativa de los elementos de la interfaz, por lo que no estaban en la misma posición en la que se había diseñado. Se cambió para que se adaptara a las diferentes resoluciones y siempre se visualizara igual.

Más adelante, se creó el **modo historia** y nuevamente las pruebas nos ayudaron enormemente. El primero de los problemas apareció con los misiles, ya que, al impactar contra el jugador, le hacía el doble del daño planeado. El fallo era que la explosión detectaba 2 colisiones, una con el jugador, y otra con la plataforma con la que se desplaza.

Con cada prototipo se fueron añadiendo nuevas funcionalidades, iban surgiendo sugerencias de cambio y se reportaron fallos que fueron corregidos. Muchos usuarios coincidieron en que era muy difícil vencer al jefe final del modo historia, debido a que se movía demasiado y las granadas tenían muy poca área de efecto, por lo que se aumentó esta área y resultaba algo más sencillo derrotarle. Por lo tanto, las pruebas sirvieron para mejoras de usabilidad.

Otro de los usuarios nos recomendó añadir **diferentes patrones** de ataque del jefe final. Inicialmente, el jefe solo lanzaba misiles horizontalmente con el cañón en posición baja. Se añadieron los ataques con cañón en posición media y posición alta.

El **modo horda** fue una sugerencia de adición por parte de mi tutor, que me dio la idea de añadir un modo de juego en el que aparecieran multitud de enemigos y que hubiese que sobrevivir. Junto con este modo de juego, se añadió un sistema de puntuaciones que almacenaba las puntuaciones de los jugadores y mostraba las 10 más altas.

Se hicieron pruebas para portar el juego a plataforma **WebGL** y poder jugarlo en navegadores, alojando el videojuego en portales web como **itch.io**. Hubo problemas para hacer funcionar el sistema de puntuaciones, ya que al leer el archivo JSON se obtenía un problema para obtener la ruta. Se logró que el sistema cargara las puntuaciones, pero no que se pudieran añadir nuevas. Debido a la falta de tiempo, este último problema no pudo solucionarse a tiempo y quedó descartado.

8 CONCLUSIONES Y TRABAJOS FUTUROS

Los resultados obtenidos en el desarrollo de este proyecto han sido bastante satisfactorios, pudiendo alcanzar un producto de calidad. Si nos basamos en los objetivos iniciales, se han logrado superar con creces, añadiendo aspectos que no se consideraron al comienzo del desarrollo.

- **Mayor variedad de enemigos:** la cantidad de personajes hostiles actualmente es escasa, y una mayor cantidad de enemigos diferentes podría añadir diversidad y originalidad al producto.
- **Mayor variedad de transformaciones:** la transformación que puede llevar a cabo los personajes al alcanzar el máximo nivel de poder es común para todos los personajes. Una transformación diferente para cada personaje que tengan cambios jugables en cada una podría añadir diversidad.
- **Mejorar jugabilidad:** los enemigos pueden llegar a ser muy duros, así que se debería ajustar la dificultad de los niveles haciendo más débiles a los enemigos o proporcionando ayudas al jugador.
- **Pulir mecánicas jugables:** algunos aspectos como golpear o bloquear son un poco toscos actualmente. Se podría dinamizar la jugabilidad añadiendo diversos combos para los golpes, ajustando el aturdimiento que estos causan al contrincante y que el bloqueo de ataques tenga alguna ventaja contra el rival.
- **Más niveles para el modo historia:** sería posible añadir más niveles y con mayor variedad, como por ejemplo niveles contrarreloj, donde el jugador debe acabar el nivel antes de que la cuenta atrás termine, o niveles auto-scrolling, donde la cámara avanza automáticamente sin el jugador y éste debe permanecer siempre dentro de la escena.
- **Más armas:** es posible añadir nuevas armas, como pistolas para atacar a distancia, bate de béisbol para atacar cuerpo a cuerpo, o armas especiales como minas que el jugador pueda usar contra los enemigos.
- **Añadir poderes:** se podría hacer que la barra de poder no sólo sirviera para transformar al personaje, sino que además permitiera utilizar una variedad de habilidades, como por ejemplo lanzar bolas de fuego.
- **Añadir número de rondas y tiempo al modo versus:** se podrían añadir rondas para que el jugador con el mayor número de rondas ganadas sea el ganador. Además, los combates se realizarán en un periodo de tiempo determinado.
- **Adaptar a dispositivos móviles:** adaptar los controles del juego a pantallas táctiles podría ser complejo, pero se podría buscar la manera de simplificarlos para que el juego se pudiera portar a dispositivos móviles.
- **Multijugador online:** relacionado con lo anterior, se podría permitir que dos jugadores pudiesen jugar juntos, cada uno con su propio móvil, ya sea mediante una conexión Bluetooth o Wi-Fi.

9 FUENTES DE INFORMACIÓN

9.1 Documentación

Se han utilizado diversas fuentes durante el desarrollo de la aplicación y para la creación de este informe.

- Manual de Unity
  https://docs.unity3d.com/Manual/
- Comunidad de Unity
  https://forum.unity.com/

- Libro Blanco del Desarrollo Español de Videojuegos 2016
  http://www.dev.org.es/publicaciones/libro-blanco-dev-2016
- Historia de los videojuegos
  https://es.wikipedia.org/wiki/Historia_de_los_videojuegos
- Spacewar!
  https://es.wikipedia.org/wiki/Spacewar!
- Magnavox Odyssey
  https://es.wikipedia.org/wiki/Magnavox_Odyssey
- Atari Pong
  https://es.wikipedia.org/wiki/Atari_Pong
- Unity
  https://es.wikipedia.org/wiki/Unity_(motor_de_juego)
- Unreal Engine
  https://es.wikipedia.org/wiki/Unreal_Engine
  https://www.unrealengine.com/en-US/features
- CryEngine
  https://es.wikipedia.org/wiki/CryEngine

9.2 Recursos

Me gustaría agradecer a toda la gente que me ha aportado los materiales necesarios para la realización de este proyecto.

9.2.1 Imágenes

- **Personajes**

Las hojas de sprites de personajes, imágenes con los fotogramas de las animaciones de cada personaje, están basadas en el videojuego Scott Pilgrim vs. the World: The Game, aunque han sido completamente hechas por fans con ligeras modificaciones personales.

- Cozah: https://cozah.deviantart.com/art/Human-Coza-Sprite-Sheet-279561640
- Romuluspixels: https://romuluspixels.deviantart.com/art/Romulus-scott-pilgrim-sprite-258614690
- Nick Sampson: https://www.pinterest.es/pin/311452130455295396/

- **Escenarios**

- RudeBootie: https://www.reddit.com/r/gaming/comments/1d7g1p/i_have_a_huge_collection_of_fighting_game/
- Neoriceisgood: http://neoriceisgood.deviantart.com/art/City-of-Gloria-504814284
- SirDrinksAlot: https://imgur.com/gallery/FWvFz
- Ansimuz: https://ansimuz.itch.io/bulkhead-walls-environment

- **Diversos objetos**

- Cañón, de AziasCreations: https://aziascreations.deviantart.com/art/Advance-Wars-Little-Cannon-414483740
- Misil de Commando 2, de Miniclip: http://commando2.wikia.com/wiki/File:C25_Marrugo_missile_burst.png
- Explosión, de wanyo: https://www.reddit.com/r/PixelArt/comments/1itheg/occchavent_been_around_much_so_i_return_with_an/
- Monedas, de Scrillrock: http://piq.codeus.net/picture/359281/ca_h_money_
- Máquina expendedora, de ThePrincessParadox: https://theprincessparadox.deviantart.com/art/Vending-Machines-Pixel-Art-340119251
- Coche 1, de Dafu: http://pixel-troll.blogspot.com.es/2010/10/car-pixel-art.html
- Coche 2, de JaeBum Joo: https://www.behance.net/gallery/6806353/The-car-series-(Pixel-Art)
- Pistola, de EvilEagles: http://pixeljoint.com/pixelart/81398.htm
- Puño, de TGBB: http://piq.codeus.net/picture/45116/falcon_punch_
- Roca, de Super Walrus Land: http://superwalrusland.com/ohr/issue26/pa/pixelart.html
- Lobo de Fire Emblem The Sacred Stones, de Intelligent Systems: https://www.spriters-resource.com/game_boy_advance/fireemblemthesacredstones/sheet/53710/
- Comida, de Camilla Lopes: https://www.pinterest.com.mx/pin/360991726363448836/
- Controles de teclado realizados en: http://www.keyboard-layout-editor.com/

9.2.2 Música y sonidos

Todas las melodías han sido realizadas con la aplicación web **BeepBox**.

http://www.beepbox.co/

La gran mayoría de los efectos especiales de sonido han sido realizados con la aplicación web **Bfxr**.

https://www.bfxr.net/

Otros sonidos han sido recogidos de diversas fuentes. Son los siguientes:

- Sonido al introducir moneda en máquina expendedora, de tweeterdj: https://freesound.org/people/tweeterdj/sounds/29649/
- Sonido al salir producto de máquina expendedora, de morgantj: https://freesound.org/people/morgantj/sounds/59669/
- Sonido al recoger granada de la caja, de Freeify Music: https://www.youtube.com/watch?v=mVwcuXDMQlg
- Sonido al recargarse la caja de granadas, de Murtaja Gamer: https://www.youtube.com/watch?v=c-BKVzTLqlc
- Sonido de pistola silenciada, de ProFX Sound Effects: https://www.youtube.com/watch?v=zdhFjB-U9e4

9.2.3 Fuentes de texto

- Bit Wonder, de Joiro Hatgaya: https://www.dafont.com/es/8bit-wonder.font
- Joystix Monospace, de Hypodermic Fonts: https://www.dafont.com/joystix.font

9.2.4 Scripts o API externas

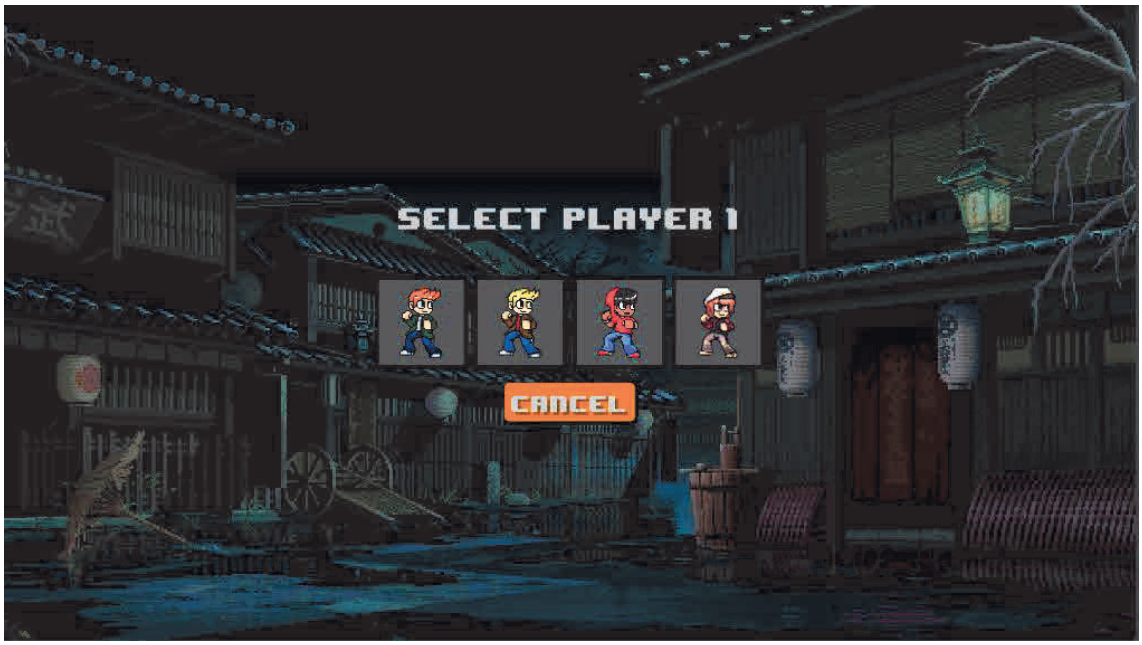
- SimpleJSON: http://wiki.unity3d.com/index.php/SimpleJSON

10 ANEXOS

10.1 Manual de usuario

- Menú principal:

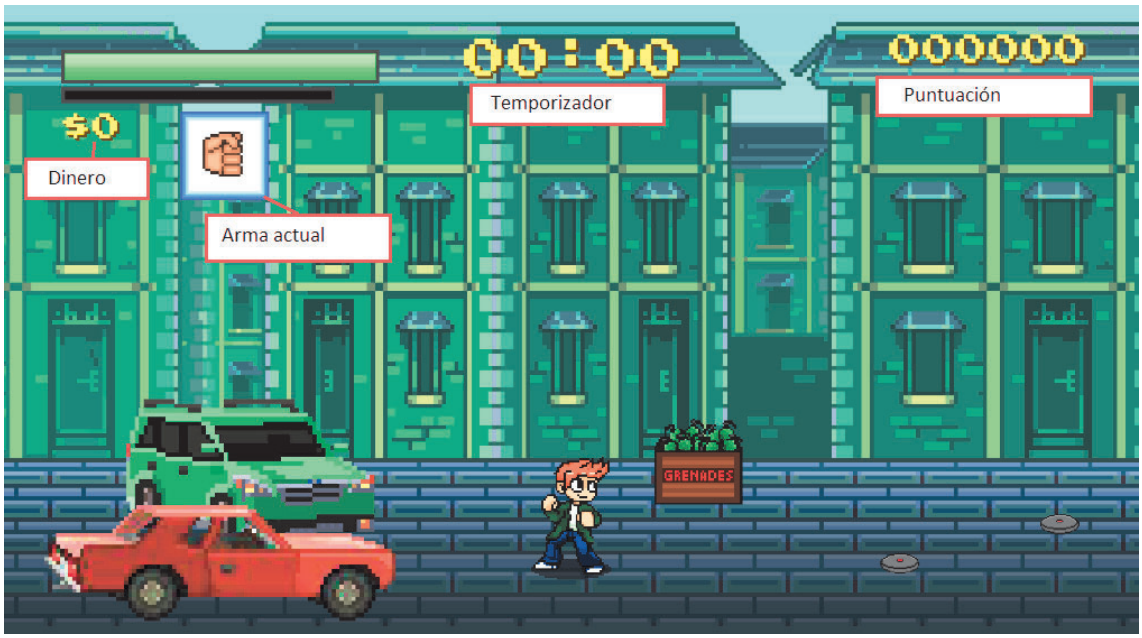
Esta es la primera pantalla que ve el usuario al ejecutar la aplicación. Se pueden ver varias opciones que se explican en la tabla inferior.



Opción	Descripción
Story Mode	Inicia el modo historia
Versus Mode	Inicia el modo versus. Se deberá de escoger el número de jugadores y el personaje de cada jugador.
Horde Mode	Inicia el modo horda. Se deberá de escoger el número de jugadores y el personaje de cada jugador.
Top Scores	Muestra las 10 puntuaciones más altas del sistema.
Default Controls	Muestra los controles del juego por defecto con teclado.
Exit	Se abandona la aplicación.

- Modo historia:

Al entrar en el modo historia, el jugador empezará en el primer nivel y se le presentará la siguiente interfaz de usuario.



En la esquina superior izquierda, se pueden observar dos indicadores. Estos corresponden a la **barra de salud** y a la **barra de poder**.

- **Barra de vida o salud:** se representa como una barra verde en la parte superior que va disminuyendo al recibir daño. La partida termina cuando se vacía. Se puede aumentar si el jugador recoge comida.

- **Barra de poder:** se representa como una barra naranja bajo la barra de salud. Se va recargando al golpear al rival o levemente al recibir daño del rival. Si se llena es posible transformarse en un lobo para ganar velocidad y daño.

Controles por defecto del modo historia y del modo horda

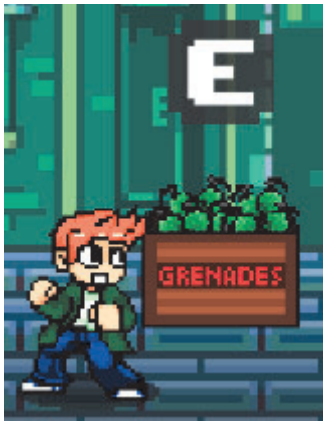
El **movimiento** del personaje en este modo de juego se corresponde con las teclas **W** y **A** para el movimiento horizontal, y **S** y **D** para el movimiento vertical.

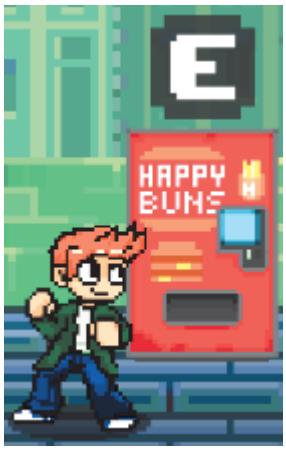
El personaje puede **golpear** a sus enemigos con las teclas **K** para el golpe fuerte y **L** para el golpe rápido.

Una vez la barra de poder se ha llenado, el jugador podrá **transformarse** pulsando la tecla **J**.

Es posible **bloquear** para recibir menos daños de los golpes de los enemigos o las trampas pulsando la **barra espaciadora**.

El jugador puede **interactuar** con las cajas de granadas y las máquinas expendedoras que encontrará a lo largo de los niveles pulsando la tecla **E** cerca de ellas.





Controles por defecto del modo versus

Los controles de este modo son iguales que el modo historia y el modo horda, pero al no haber desplazamiento vertical, los controles para este tipo de movimiento son sustituidos por la capacidad de saltar, con la tecla **W**.

Modos de juego

El videojuego está compuesto de 3 modos de juego.

- **Modo historia**:

Para 1 jugador. En este modo el jugador debe superar los niveles avanzando por el escenario venciendo a todos los adversarios que se le crucen en el camino, incluido el jefe final. El jugador puede coger granadas de las cajas que hay esparcidas por el escenario para luego atacar a sus rivales. Solo se puede coger una granada cada vez.

Cuando el jugador va derrotando enemigos, éste va obteniendo dinero que puede guardar para obtener más puntuación, o si lo necesita, gastarlo en las máquinas expendedoras que hay en los escenarios para comprar comida y curar una porción de la barra de salud.

- **Modo versus**:

Para 1 o 2 jugadores. Este modo es una lucha directa de un jugador contra el otro y gana el que reduzca antes a cero la barra de salud de su adversario. Es posible escoger entre 4 personajes diferentes.

- **Modo horda**:

Para 1 o 2 jugadores. El objetivo de este modo de juego es aguantar el mayor número de rondas posibles derrotando a todos los enemigos que nos aparezcan. Ciertas rondas darán al jugador una pequeña regeneración de salud, y aleatoriamente aparecerán potenciadores de daño por el escenario que ayudarán al jugador a llegar más lejos.

Personajes – Modo historia

- **Protagonista**

El personaje que maneja el jugador en el modo historia. Dispone de todos los movimientos y características de los personajes.

- **Enemigos cuerpo a cuerpo**

Son los personajes más abundantes en los niveles del modo historia. Enemigos que al entrar en contacto con el protagonista se disponen a atacarle cuerpo a cuerpo. No pueden transformarse. Vencerlo proporcionará dinero al jugador. Proporciona 100 puntos a la puntuación.

- **Enemigos a distancia**

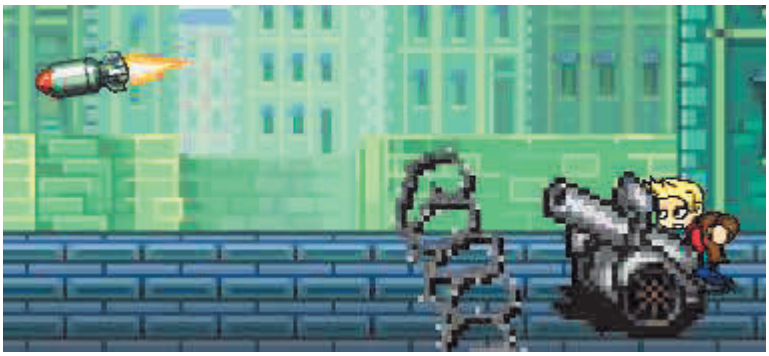
Personajes más raros de ver en el modo historia. Enemigos que se mantienen a una distancia segura y atacan al protagonista con una pistola. No pueden atacar cuerpo a cuerpo ni transformarse. Vencerlo proporcionará dinero al jugador. Proporciona 250 puntos a la puntuación.

- **Jefe del nivel**

Solo aparece uno al final del nivel. Desde la distancia y tras una verja, busca atacar al protagonista con su cañón. Solo se le puede dañar lanzándole granadas. No puede golpear cuerpo a cuerpo ni transformarse. Proporciona 1000 puntos a la puntuación. Puede realizar 3 tipos de ataque:

Ataque horizontal: el jefe baja su cañón y dispara misiles en línea recta hacia el protagonista.

Ataque diagonal: el jefe mantiene su cañón en la posición de en medio y dispara misiles que caen hacia el jugador en forma de parábola.



Ataque vertical: el jefe sube su cañón y dispara misiles al cielo, que tras unos segundos bajan en caída libre hacia el jugador.

Pantalla de fin de partida

Cuando la partida se acaba, ya sea porque el jugador ha perdido o porque ha superado el nivel, se muestra una pantalla con la puntuación obtenida. Se da la oportunidad al jugador a que introduzca su nombre y guarde esa puntuación en el sistema. Si ha obtenido una puntuación lo suficientemente buena, aparecerá entre las 10 mejores puntuaciones del menú principal.

Pantalla de fin de partida en el modo historia

Pantalla de fin de partida en el modo horda

Las 10 mejores puntuaciones registradas en el sistema