

Comparativa de dos desarrollos

Grado en Ingeniería Multimedia



Trabajo Fin de Grado

Autor:

Jorge García Valera

Tutor/es:

Mireia Luisa Sempere

Mayo 2019



# AGRADECIMIENTOS

# DEDICATORIA

# CITAS

Cada época tiene su forma de contar historias, y el videojuego es una gran parte de nuestra cultura. Puedes ignorar los videojuegos o aceptarlos y empaparte de su gran calidad artística. Algunas personas están cautivadas con los videojuegos de la misma forma que a otras personas les encanta el cine o el teatro.

Andy Serkis

El objetivo claro de los videojuegos es entretener a la gente sorprendiéndoles con nuevas experiencias.

Shigeru Miyamoto

# INDICE

[AGRADECIMIENTOS 2](#_Toc534737736)

[DEDICATORIA 3](#_Toc534737737)

[CITAS 4](#_Toc534737738)

[INDICE 5](#_Toc534737739)

[INDICE DE FIGURAS 6](#_Toc534737740)

[INTRODUCCIÓN 7](#_Toc534737741)

[JUSTIFICACIÓN Y OBJETIVOS 8](#_Toc534737742)

[METODOLOGÍA 9](#_Toc534737743)

[MARCO TEÓRICO 11](#_Toc534737744)

[¿QUE ES UN VIDEOJUEGO HACK AND SLASH? 11](#_Toc534737745)

[ACTUALIDAD DE LOS VIDEOJUEGOS HACK AND SLASH 11](#_Toc534737746)

[HERAMIENTAS 12](#_Toc534737747)

[DE GESTIÓN 12](#_Toc534737748)

[DE DESARROLLO 12](#_Toc534737749)

[GRÁFICAS 13](#_Toc534737750)

[PRIMEROS PASOS 14](#_Toc534737751)

[BUSCAR Y ANALIZAR CARACTERISTICAS Y MECANICAS 14](#_Toc534737752)

[BUSQUEDA DE CARACTERISTICAS 14](#_Toc534737753)

[ANALISIS DE CARACTERISTICAS 14](#_Toc534737754)

[BUSQUEDA DE MECANICAS 15](#_Toc534737755)

[ANALISIS DE MECANICAS 15](#_Toc534737756)

[BUSCAR Y CREAR ASSETS 16](#_Toc534737757)

[SPRITE SHEETS 16](#_Toc534737758)

[PALETA DE COLORES 16](#_Toc534737759)

[MENUS / PANTALLAS 16](#_Toc534737760)

[INTERFACES 20](#_Toc534737761)

# INDICE DE FIGURAS

[Ilustración 1 - Ejemplo de Kanban 9](#_Toc534737630)

[Ilustración 2 - Ejemplo de Trello a mitad de desarrollo 10](#_Toc534737631)

[Ilustración 3 – Sprite sheet de entidades 16](file:///C:\Users\krudo\Desktop\TFG\Memoria%20TFG.docx#_Toc534737632)

[Ilustración 4 – sprite sheet de mapa 16](file:///C:\Users\krudo\Desktop\TFG\Memoria%20TFG.docx#_Toc534737633)

[Ilustración 5 - Paleta de colores 16](#_Toc534737634)

[Ilustración 6 - Menú principal 17](#_Toc534737635)

[Ilustración 7 - Menú selección modo de juego 18](#_Toc534737636)

[Ilustración 8 - Menú opciones 18](#_Toc534737637)

[Ilustración 9 - Menú selección jugador (solo) 19](#_Toc534737638)

[Ilustración 10 - Interfaz 1 jugador 20](#_Toc534737639)

[Ilustración 11 - Interfaz 2 jugadores 21](#_Toc534737640)

# INTRODUCCIÓN

La industria de los videojuegos es hoy en día una de las más grandes, superando incluso a la del cine. Es por eso por lo que cada día salen cientos de videojuegos, ya sean de grandes, medianas o pequeñas empresas, o hasta incluso de un grupo de gente que se junta para desarrollar y crear contenido en una de sus pasiones.

Ya sea por el interés económico o por crear contenido por una pasión, mucha gente se lanza a la industria del videojuego ya sea como desarrollador o como artista (2D o 3D). Una vez dentro de la industria ven que no es tan fácil llegar a lo mas alto de forma rápida, sino que tienes que empezar por el escalafón mas bajo y poco a poco ir subiendo e ir ganando experiencia.

# JUSTIFICACIÓN Y OBJETIVOS

La idea de este trabajo nace con la visión de la autosuperación y el crecimiento personal dentro del ámbito del desarrollo de videojuegos.

La realización de este trabajo se plantea de tal forma en que se volverá a desarrollar un antiguo proyecto, el cual cuando se desarrollo no se tenia mucha experiencia en el ámbito del desarrollo de videojuegos ni con él lenguaje de desarrollo. Ahora con más experiencia en ambos campos se pretende mejorar el resultado del producto final.

Para conseguir esta mejora se revisará el proyecto antiguo y se extraerán las ideas y mecánicas principales. Una vez extraídas las ideas y mecánicas se entrará en un proceso en el cual se decidirá si esa mecánica que se pensó en el inicio es valida o no, o si necesita una mejora para que se adapte todavía mas al producto final que deseamos.

La idea para este proyecto en mi opinión es original, ya que se sale de la típica idea de únicamente desarrollar un videojuego para demostrar tus conocimientos. Esta idea puede servir para que generaciones futuras no le teman al desarrollo, ya que en sus inicios afrontar un desarrollo de un videojuego puede parecer algo imposible, pero como todo, es cuestión de dedicarle tiempo y mejorar en el entendimiento del lenguaje y de la librería que decidas utilizar. Todo esto estará recopilado al final de documento, ya que se va a estudiar el resultado de ambos proyectos.

A su vez, sirve para que el desarrollador vea como ha ido cambiado y mejorando su forma de programar y de afrontar los problemas que surgen durante un desarrollo.

Los objetivos por tratar en este trabajo son:

* Extraer las ideas y mecánicas originales y analizarlas para comprobar que son validas para el nuevo proyecto.
* Con las nuevas mecánicas e ideas desarrollar una nueva versión del videojuego original.
* Comparar ambas versiones y observar el cambio de rendimiento y de limpieza en el código.

# METODOLOGÍA

La metodología utilizada para realizar este proyecto se denomina Kanban.

La metodología Kanban consiste en un panel en el cual tenemos tantas columnas como fases queramos o necesitemos en nuestro proyecto (Por hacer, en proceso, en diseño, terminado, etc.).

Las columnas deberán ser rellenadas con tarjetas (normalmente post-it de diversos colores para una fácil y rápida interpretación del estado del proyecto), en las cuales se escribe el nombre de la tarea a realizar y una duración estimada de la misma.

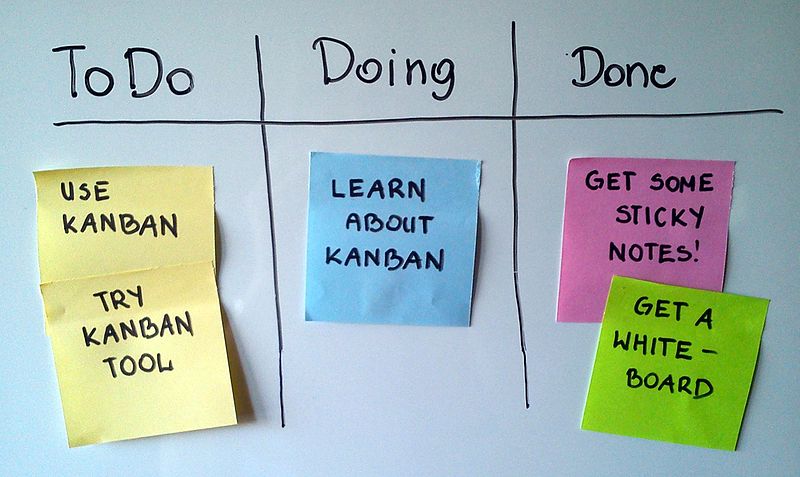


Ilustración 1 - Ejemplo de Kanban

Antes de empezar con el desarrollo y teniendo claro que es lo que tenemos que hacer para terminar nuestro producto, tenemos que empezar por rellenar todas las tarjetas necesarias, ordenarlas por orden de prioridad (si fuera necesario) y colocarlas en la columna de ‘Por hacer’. Cuando se empiece con el desarrollo. Lo primero que tenemos que hacer si no tenemos ninguna tarea asignada es seleccionar una tarjeta y ponerla en la columna de ‘En proceso’. Una vez tengamos la tarea terminada la ponemos en la columna de ‘Terminado’ y se repetiría todo el proceso desde la primera columna.

Si durante el desarrollo nos encontramos que una tarea es demasiado grande se puede subdividir esa tarea en otras más pequeñas añadiendo nuevas tarjetas a la columna de ‘Por hacer’. De esta forma conseguiremos agilizar el flujo de trabajo y conseguir que el producto tenga una calidad superior.

Para poner en marcha esta metodología se ha decidido utilizar la aplicación Trello, que es una aplicación web que nos permite crear las columnas y las tarjetas. Además, a las tarjetas se les puede añadir funcionalidades extras como una checklist (lista con tics para marcar que se ha realizado) y añadir comentarios a las tarjetas por si nos encontramos con alguna dificultad o queremos dejarlo algo remarcado para un futuro.

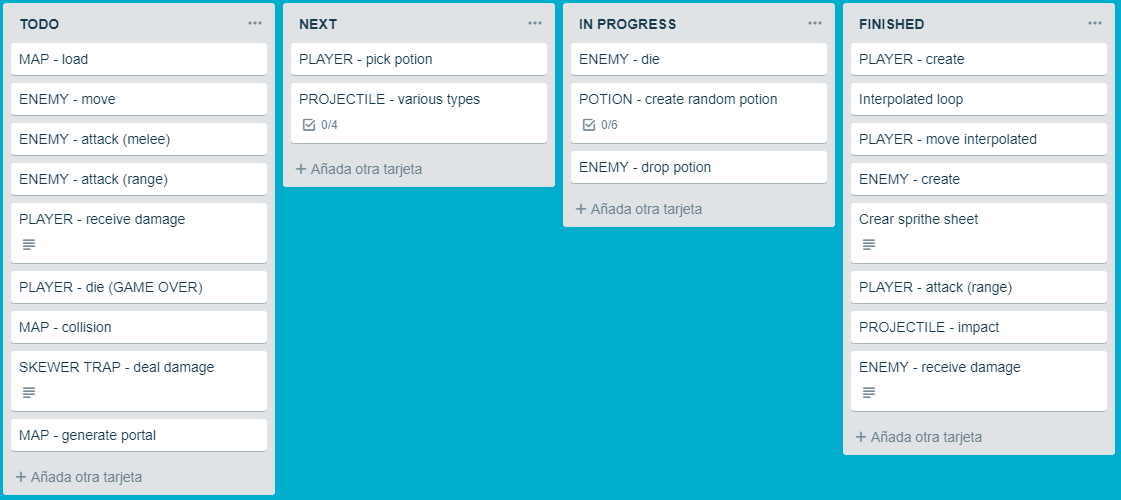


Ilustración 2 - Ejemplo de Trello a mitad de desarrollo

# MARCO TEÓRICO

## ¿QUE ES UN VIDEOJUEGO HACK AND SLASH?

Un videojuego del género Hack and Slash es un videojuego en el cual se enfatiza el combate cuerpo a cuerpo. En este género, el jugador se enfrenta en un mapa lineal a una gran cantidad de enemigos. Una vez que el jugador haya derrotado a todos los enemigos y llegue al final del nivel le esperará la lucha contra el jefe final, el cual supondrá un mayor desafío.

A su vez, existe otro género que podría ser considerado el padre de este género. El género del que hablamos es **Beat ‘em up** (también conocido como **Brawler**) en el cual el jugador se enfrenta también a una gran cantidad de enemigos.

Como podemos observar, ambos géneros son muy similares, por lo cual vamos a ver cuál es la principal diferencia entre ambos. La principal diferencia entre estos ambos géneros es que en el **Beat ‘em up** el estilo de lucha es **cuerpo a cuerpo sin armas**, directamente con los puños y/o piernas. Sin embargo, en el **Hack and Slash** el protagonista suele usar un **arma blanca**, ya sea para golpear y/o lanzar.

A su vez, existe otro subgénero llamado **Shoot ‘em up** el cual está enfocado a la lucha desenfrenada empuñando **armas de fuego**.

Otra característica que comparten estos géneros es que además de ser jugados en modo de un jugador pueden ser jugados en **modo cooperativo** en dos formas distintas.

* **Pantalla compartida:** en este modo dos jugadores pueden jugar a la vez desde una misma pantalla.
* **Modo online:** en este modo cuatro jugadores pueden jugar a la vez, cada uno desde su respectiva consola.

Lo interesante de este modo cooperativo es que los jugadores no tienen que luchar o competir entre sí para ver quién es mejor, sino que deben de unir fuerzas para derrotar a las hordas y así poder llegar aún más lejos en las infinitas rondas de enemigos que plantean estos géneros.

## ACTUALIDAD DE LOS VIDEOJUEGOS HACK AND SLASH

Actualmente, el género ha sido un poco reinventado ya que hoy en día hay consolas más potentes que las que había cuando este género fue creado. A su vez, el género ha sido combinados con otros como el de Plataformas, Aventura o Acción para dotar al producto final de una mayor diversión y jugabilidad juntando lo mejor de ambos géneros.

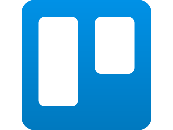
* **For Honor:** Mezcla los géneros de Acción y Hack and Slash con una temática medieval. For Honor trata de una lucha entre 2 equipos compuestos por 3 jugadores cada uno en el cual los jugadores tienen que luchar entre sí para conquistar puestos estratégicos para que su equipo consiga puntos y gane la partida.
* **Dead Rising (franquicia):** Mezcla los géneros de Acción, Aventura, Hack and Slash y Shoot ‘em up con temática zombi. Dead Rising 4 es un juego en el cual encarnamos a Frank West y debemos de sobrevivir a los infinitos zombis recolectando y creando armas mientras avanzamos en la historia.

# HERAMIENTAS

Para poder realizar este proyecto se han utilizado diversos tipos herramientas, cada una especializada en un campo en concreto.

## DE GESTIÓN

Las herramientas de gestión son aquellas que nos ayudan durante el desarrollo, ya sea mediante la organización o creando copias de seguridad por si en algún momento tenemos que retroceder debido a algún error o cambio en la estructura. Las herramientas de este tipo que hemos utilizado han sido:

**Trello:** Trello es una aplicación web que nos permite organizar información en forma de tarjetas las cuales podemos mover con total libertad entre las múltiples columnas que nos permite crear. Trello ha sido utilizado principalmente para crear el orden de desarrollo.

**Toggl:** Toggl es una aplicación web que nos permite llevar un registro de las horas que hemos empleado para la realización del proyecto. Para utilizarla basta con rellenar un campo con el nombre de la tarea y a continuación darle al botón de “Play” y el tiempo empezará a contarse.

**GitKraken:** GitKraken es una interfaz gráfica para git, lo que nos permite realizar en cuestión de unos pocos clics copias de seguridad de nuestro código a nuestra cuenta de GitHub.

## DE DESARROLLO

Las herramientas de desarrollo son aquellas que nos han permitido dar forma a nuestro proyecto, ya sea mediante un editor de texto o mediante librerías para añadir funcionalidad. Las herramientas de este tipo que hemos utilizado han sido:

**Visual Studio Commuity 2017:** Visual Studio ha sido el IDE utilizado para desarrollar el proyecto. Se ha decidido utilizar este IDE ya que nos facilita mucho el trabajo con atajos de teclado, compilador integrado, autocompletar, texto predictivo, creación de clases con dos clics y muchas otras funcionalidades.

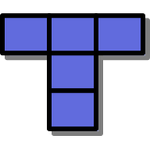
**SFML: ‘**Simple and Fast Multimedia Library’, es la librería que ha hecho posible el desarrollo de este proyecto junto con el lenguaje C++. Esto es gracias a que nos permite crear una ventana en la cual nosotros podemos pintar los assets (sprites, imágenes y más) de nuestro proyecto. También nos permite recoger los inputs tanto de teclado y ratón como los de un joystick para así poder interaccionar con los assets.

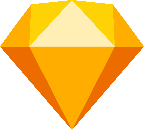
**TinyXML-2:** TinyXML-2 es una librería que nos permite leer de forma rápida y sencilla el contenido de un fichero XML. Esta librería la usaremos para leer el fichero XML que nos generará Tiled con toda la información del mapa del juego.

## GRÁFICAS

Las herramientas gráficas son aquellas que nos han permitido crear todos los assets visuales necesarios para poder realizar el proyecto, ya sean bocetos, imágenes sueltas o ‘Sprite sheets’ (conjunto de imágenes o animaciones de un sprite).

**Photoshop:** Photoshop es un programa de creación y edición de imágenes. Este programa nos permite crear los múltiples assets necesarios para nuestro proyecto. Más en concreto me ha permitido crear los SpriteSheets (imagen que contiene todas las texturas de una o varias animaciones) necesarios para luego convertirlos en texturas y así poder aplicarlos a los sprites dentro del videojuegos.

**Tiled:** Tiled nos permite crear de forma rápida y simple mapas en 2D con múltiples capas para luego utilizarlos en nuestros proyectos. Para poder usar Tiled primero tenemos que crear un SpriteSheet. Una vez tenemos creado el SpriteSheet lo cargamos en Tiled y con unos cuantos clics podremos crear nuestro mapa.

**Sketch**: Sketch es una aplicación que nos permite crear de forma rápida bocetos para nuestras aplicaciones. En concreto en este proyecto se ha utilizado Sketch para diseñar y bocetar todos los menús y la interfaz del juego.

# PRIMEROS PASOS

Antes de empezar con el desarrollo y una vez establecidas las herramientas principales para poder llevar a cabo el proyecto se empezó con la búsqueda y recuperación de las ideas y mecánicas principales que formaron el juego original. En esta búsqueda también se recuperaron los ‘assets’ del juego (principalmente se recuperaron los sprites de los personajes, los enemigos y los tiles que forman el mapa), ya que este trabajo está enfocado a la parte de la programación de código y no a la parte del arte.

## BUSCAR Y ANALIZAR CARACTERISTICAS Y MECANICAS

Una vez recuperada la versión final del proyecto original se empezó por estudiar el documento que resumió el desarrollo y el estado final del primer proyecto. Gracias a este documento se pudieron rescatar de forma rápida todas las características y mecánicas básicas que contenía el juego base, implementadas o no.

Una vez se identifiquen todas las características y mecánicas se procederá con el análisis de todas y cada una de ellas para ver si encajan bien con el nuevo proyecto o necesitan ser descartadas, o si por el contrario necesitan una mejora para poder encajar mejor en el nuevo proyecto.

### BUSQUEDA DE CARACTERISTICAS

Las características encontradas fueron las siguiente:

* Modos de juego
  + Modo individual.
  + Modo cooperativo local (no implementado).
* Personajes
  + Personaje jugable azul.
  + Personaje jugable verde.
  + Personaje jugable amarillo (no implementado)
  + 1 enemigo.
  + 1 enemigo jefe.
* Mapa
  + Mapa finito predefinido.
  + Minijuego predefinido.

### ANALISIS DE CARACTERISTICAS

En cuanto a los **modos de juego** no se ha encontrado ningún contra por lo que ambos modos de juego seguirán presentes en la nueva versión del juego.

Respecto a los **personajes** se van a mantener los 3 personajes jugables. Con respecto a los personajes enemigos encontramos que el número incrementa de ser únicamente 1 a 3. Se conserva el enemigo original que ataca cuerpo a cuerpo y se añaden dos nuevos:

* Enemigo a distancia: lanza una bola viscosa que hace daño al jugador si impacta con él.
* Enemigo con carga: se inmoviliza y carga un ataque rabioso. Cuando termina de cargar el ataque este sale corriendo en línea recta hacia el jugador, si impacta el jugador recibe daño.

Respecto al **mapa** se descarta el mapa con el minijuego que aparece cada cierto tiempo porque rompe un poco con el frenesí del juego y también porque no se puede recompensar de forma correcta al jugador.

### BUSQUEDA DE MECANICAS

Las mecánicas encontradas fueron las siguientes:

* Modo de juego infinito basado en oleadas.
  + Cada oleada aumenta la cantidad de enemigos y resistencia.
  + Cada 5 oleadas aparece el jefe enemigo.
* Personajes jugables
  + Libertad de movimiento 2D (arriba y abajo, izquierda y derecha).
  + Atacan a los enemigos con diversos ataques
    - Ataque básico infinito.
    - Habilidades especiales con tiempo de reutilización (no implementado).
  + Pierden/ganan vida.
  + Ganan experiencia y suben de nivel (no implementado).
* Personajes enemigos
  + Persiguen a los jugadores.
  + Atacan a los jugadores cuerpo a cuerpo.
  + El jefe ataca cuerpo a cuerpo y a distancia.
  + Al morir el personaje jugable que lo derrota recibe experiencia.
  + Al morir pueden soltar una bola de vida.
* Mapa
  + Contiene trampas de pinchos que al ser pisadas infligen deño a los personajes.
  + Contiene elementos que bloquean el paso a los personajes.
  + Genera un portal con un minijuego cada cierto tiempo.
    - Al superar este minijuego el jugador recibe vida.

### ANALISIS DE MECANICAS

Respecto a las mecánicas del **modo de juego** no se ha encontrado ninguna contra, por lo cual el sistema de oleada seguirá siendo utilizado.

Respecto a los **personajes jugables** el movimiento seguirá siendo en 2D con las mismas direcciones. En los ataques seguirá habiendo 1 ataque básico, pero ahora los ataques especiales aumentan a 3 para dar un poco mas de variedad al rol del jugador. También se añadirán puntos de magia al jugador que será el limitante para poder lanzar mas o menos habilidades especiales.

Los **personajes enemigos** seguirán actuando de la misma forma, pero con el cambio de que ahora en vez de poder soltar una bola de vida tienen una posibilidad de soltar una poción (de 6 tipos distintos que existirán).

En cuanto al **mapa** como en las características se ha eliminado el mapa con el minijuego se elimina también la mecánica relacionada con este tipo de mapa.

## BUSCAR Y CREAR ASSETS

Una vez teniendo todos los assets localizados se empezó por seleccionar cuales iban a estar presentes en esta nueva versión. A su vez se crearon algunos nuevos que no estaban en la versión original del juego y otros fueron ligeramente modificados para que encajaran mejor en esta nueva versión.

### SPRITE SHEETS

Con todos los sprites ya listos se crearon 2 sprite sheets: uno para el mapa y otro para el resto de las entidades.

Ilustración 3 – Sprite sheet de entidades

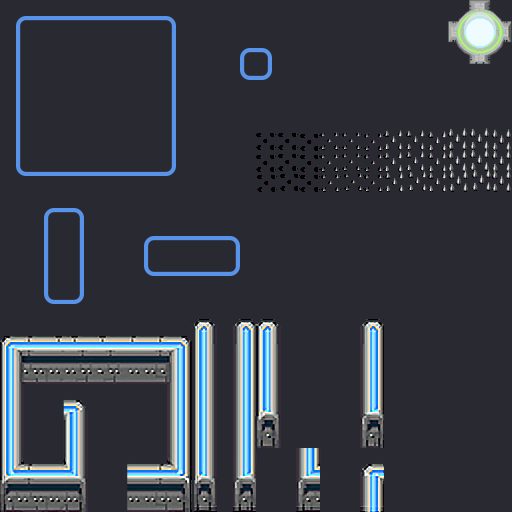


Ilustración 4 – sprite sheet de mapa

### PALETA DE COLORES

Tras observar como quedaron los sprite sheets se continuó creando una paleta de colores con los principales colores que componen la mayoría de los elementos para así poder ser usados en el menú y en la interfaz del juego.



Ilustración 5 - Paleta de colores

### MENUS / PANTALLAS

Cuando ya se tuvo la paleta de colores completa, se procedió a crear los bocetos de los menús y el de la interfaz, para así agilizar luego el proceso de creación de los mismo y no tener que parar el desarrollo para bocetarlos y posteriormente crearlos.

Este será el menú que se verá nada más se ejecute el proyecto. En este menú podemos observar cómo se encuentran todos los colores de la paleta que hemos mostrado anteriormente (Ilustración 5). En la parte superior en el centro observamos se encuentra el nombre del videojuego “ALPHAS”. En la parte izquierda encontramos 3 botones con las posibles acciones que podemos realizar. Si la opción que seleccionamos es la primera “JUGAR” aparecerá el menú mostrado en la Ilustración 6. Si seleccionamos “OPCIONES” aparecerá el menú mostrado en la Ilustración 8. Si por el contrario seleccionamos “SALIR” el juego se cerrará.



Ilustración 6 - Menú principal

En **el menú de selección de modo de juego** podemos seleccionar el modo de juego que queremos, ya sea jugar solo o con un amigo de forma cooperativa. Una vez seleccionada la opción deseada pasaremos al menú de selección de personaje (Ilustración 9) si hemos seleccionado “SOLO” o a (poner ilustración) si hemos seleccionado “COOPERATIVO”.



Ilustración 7 - Menú selección modo de juego

En el menú de **opciones** podremos modificar todas las opciones referentes con los sonidos del videojuego. Subir o bajar el volumen del general del juego, el volumen de los efectos o el volumen de la música.

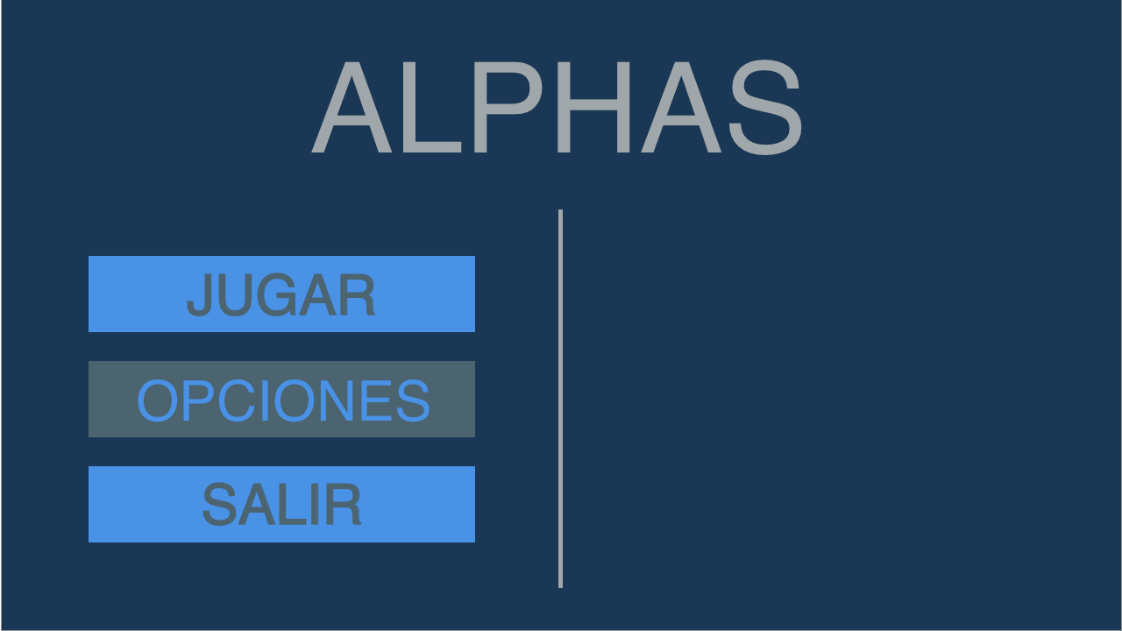


Ilustración 8 - Menú opciones

La pantalla de **selección de personaje (solo)** consta en el inferior derecho de los 3 personajes disponibles para ser usados. El jugador podrá desplazarse sobre ellos y seleccionar el que más les guste. Mientras va desplazándose sobre ellos va cambiando la imagen en grande del seleccionado junto con la información de los ataques y las estadísticas base.

Una vez el jugador seleccione un personaje el botón de “JUGAR” estará desbloqueado.



Ilustración 9 - Menú selección jugador (solo)

La **pantalla de selección de personaje (cooperativo)** es igual a la pantalla de selección de personaje (solo) con la pequeña diferencia de que ahora no podemos observar las características de los personajes, sino que en su lugar aparecerá la imagen del personaje que ha seleccionado el segundo jugador.

Una vez los dos jugadores seleccionen un personaje distinto al del otro el botón de “JUGAR” estará desbloqueado.

(INSERTAR IMAGEN SELECCIÓN JUGADOR COOPERATIVO)

### INTERFACES

Una vez bocetados los menús procedemos con el bocetado de las interfaces del juego. Como tenemos dos modos de juego (un solo jugador o dos jugadores) tendremos que crear dos interfaces con la máxima similitud posible para que los jugadores no se pierdan independientemente del modo de juego en el que estén.

En la interfaz de 1 jugador observamos como en la parte inferior en el centro tenemos toda la información relevante sobre el personaje que esta siendo utilizado. Encontramos:

* Dos barras: una roja (vida) y otra azul (magia).
* Entre medias de estas dos barras encontramos los iconos de las 3 habilidades especiales del personaje junto con su respectivo ‘feedback’ visual.
  + Si la habilidad esta con colores vivos es que podemos usar la habilidad.
  + Si la habilidad tiene el color apagado es que tenemos que esperar un tiempo para poder volver usar esa habilidad.
  + Si la habilidad tiene un tono azul significa que la habilidad está cargada pero que no tenemos la magia necesaria para lanzarla.
* Debajo de las barras encontramos 4 iconos que hacen referencia a los puntos de daño, armadura, velocidad de ataque y velocidad de movimiento.

Por otro lado, en el centro de la parte superior encontramos en que numero de oleada estamos y cuantos enemigos quedan para que termine la oleada actual.



Ilustración 10 - Interfaz 1 jugador

La interfaz de dos jugadores es exactamente idéntica a la de 1 jugador, pero ahora en vez de tener abajo en el centro la información de un jugador tenemos la información de los dos jugadores, una a la izquierda (jugador 1) y otra a la derecha (jugador 2).



Ilustración 11 - Interfaz 2 jugadores

# DESARROLLO NUEVA VERSION

Teniendo ya clara la idea junto con las características principales y las mecánicas básicas se procederá al desarrollo de la nueva versión de ALPHAS. Es posible que durante el desarrollo de la nueva versión nos encontremos con algún problema o debamos de cambiar un poco el planteamiento principal. Todas estas dudas o problemas se verán reflejadas en sus respectivos apartados.

A continuación, se procederá a explicar como se han ido creando las múltiples entidades que componen la nueva versión de ALPHAS, cuáles son sus características y cuál es su finalidad dentro del videojuego.

## CLASE ENGINE MANAGER

La clase **Engine Manager** es una clase que implementa el patrón de diseño **Facade** junto con un **Singleton**.

Esta clase nos permite desacoplar por completo el motor que usará y moverá nuestro juego (SFML) de tal forma que todas las llamadas que se realizan a SFML se invocan desde esta clase. Esta estructura nos facilitaría el cambio de motor en un futuro si fuera necesario, ya que solamente deberíamos de modificar esta clase y no el resto de los ficheros.

## CLASE ENTIDAD (ENTITY)

La clase entidad es la clase base y clave para el desarrollo del juego. Esta clase contiene los métodos y elementos base para la gran mayoría de los elementos que formaran ALPHAS.



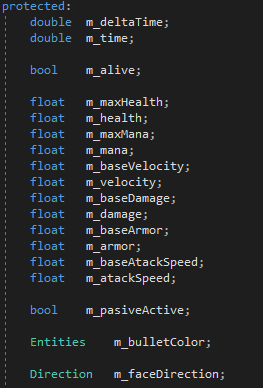
Ilustración - Fichero de cabecera de la clase Entity

Como se puede observar en la anterior Ilustración, la clase Entidad contiene el identificador del Sprite, el tamaño del Sprite, la posición actual y la última posición del Sprite. Además, contiene una variable que nos permite saber qué tipo de entidades en concreto.

En la parte de los métodos observamos que la mayoría son getters de las variables nombradas anteriormente. No es necesario utilizar setters debido a que los valores son modificados en las clases derivadas.

Los únicos métodos que no son getters son update() y draw(). Estos dos métodos son virtuales debido a que no es necesario que sean implementados en la clase base, sino que deben ser implementados en las clases derivadas.

## CLASE JUGADOR (PLAYER)

La clase Jugador es derivada de la clase Entidad. Esta clase no es la clase final, sino que es una clase intermedia antes de llegar al verdadero personaje que manejará el jugador.

Como los 3 personajes jugables que hay en ALPHAS tienen las mismas mecánicas y variables esta clase nos sirve para implementar las mecánicas y no tener que repetir el mismo código 3 veces, 1 por cada personaje.

Como podemos observar, el jugador tiene los parámetros de vida, magia, velocidad de movimiento, daño, armadura y velocidad de ataque.

Junto a las variables de vida y magia encontramos las mismas, pero con un ‘max’ delante. Esto quiere decir que no pueden sobrepasar ese valor.

Junto a las variables de velocidad de movimiento, daño, armadura y velocidad de ataque encontramos las mismas, pero con un ‘base’ delante. Esto quiere decir que es el valor base al que deben de volver cuando se acabe el efecto de una poción o una habilidad.

Ilustración - Variables clase Jugador

Estas variables serán luego diferentes dependiendo del personaje final que elijamos, lo que nos permitirá que cada personaje se adapte mejor a un rol (agresivo, defensivo o supervivencia).

En la parte de los métodos (Ilustración 14), podemos observar que al igual que en la clase base Entidad solo existen métodos getter debido a que los valores se actualizan en esta clase o en la derivada que será ya el propio jugador.

Además de estos métodos encontramos los métodos para recibir daño ya sea desde un enemigo o desde una trampa. Se ha creado esta distinción ya que si se recibe daño de un enemigo entra en juego el factor de la armadura, pero si es una trampa la que inflige daño el factor armadura no juega ningún papel, por lo que recibe todo el daño que inflige esa trampa.

Debajo de estos métodos encontramos los correspondientes de cada poción. Como el efecto es el mismo independientemente de que personaje la coja se han creado en esta clase y no en las derivadas.



Ilustración - Métodos de la clase Jugador

Debajo de los métodos de las pociones encontramos el método para poder mover a los jugadores por el mapa.

Los últimos métodos que encontramos son los relacionados con los ataques. El primero es el ataque básico (rangeAtack()) el cual lanza una aro en línea recta en la dirección a la que esta apuntando la cabeza del personaje. Debajo encontramos el método enoughMana() el cual sirve para saber si tenemos suficiente magia para poder lanzar una habilidad especial. Los 3 últimos son los correspondientes a las 3 habilidades que tendrá cada jugador y por lo tanto son virtuales puras, lo que significan que deben de ser implementadas en la clase derivada sí o sí.

### CLASE JUGADOR FINAL (PLAYER BLUE/GREEN/YELLOW)

Esta clase es derivada de la clase Jugador. La clase Jugar final solo contiene como métodos los 3 métodos virtuales puros de la clase Jugador ya que deben de implementarse sí o sí en esta clase ya que es la derivada. Estos 3 métodos están formados por diversos tipos de aros, dependiendo del rol del personaje que sea.

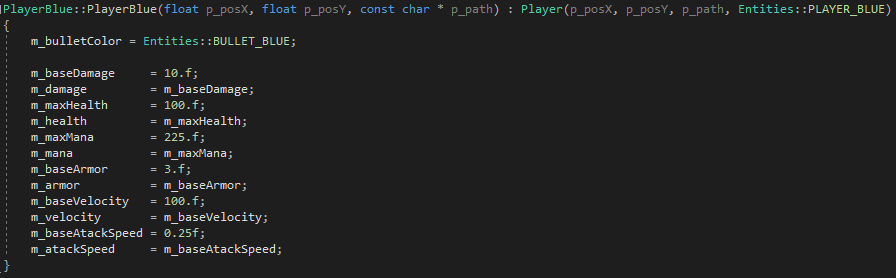
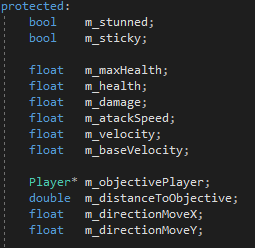


Ilustración - Constructor Jugador Azul (Jugador Final)

Como se puede observar en la imagen anterior el constructor del Jugador final (el azul en este caso) implemente todas las variables que se nombraron en la clase Jugador. De esta forma podemos crear jugadores con características diferentes y modificarlas de forma fácil.

## CLASE ENEMIGO (ENEMY)

La clase Enemigo es derivada de la clase Entidad. Al igual que Jugador esta es una clase intermedia que nos permitirá configurar los enemigos antes de darles su verdadero rol.

Como podemos observar, los enemigos tienen al igual que los jugadores vida, daño, velocidad de ataque y velocidad de movimiento, pero no tiene armadura.

Las dos primeras variables de tipo bool son efectos de estados que les pueden causar los jugadores con sus diferentes ataques especiales.

Para terminar, encontramos que cada enemigo tiene una variable para guardar cual es el jugador objetivo al que tiene que perseguir dependiendo de varios factores. Debajo encontramos la distancia que los separa (se utiliza para saber si el jugador esta en rango para poder inicializar un ataque). Las ultimas variables corresponden con la dirección de movimiento que deben de seguir para poder llegar hasta el personaje objetivo.

Ilustración - Variables clase Enemigo

En la parte correspondiente a los métodos de la clase Enemigo encontramos dos métodos que son parecido a los de la clase Jugador, estos métodos son los de recibir daño. La diferencia es que el enemigo guarda cual ha sido el ultimo proyectil que le ha causado daño para que no le haga daño durante una pequeña cantidad de tiempo. Al mismo tiempo que el enemigo comprueba si ha recibido daño o no, este comprueba también si está muerto o no.

A continuación, encontramos los métodos que nos permiten mover al enemigo por el mapa. Para mover al enemigo lo primero que hacemos es identificar cual es el jugador que está mas cerca o cual es el que menos vida tiene para poder ir a derrotarlo. Una vez el jugador objetivo es identificado el enemigo lo perseguirá.

Para terminar, encontramos los métodos correspondientes con los ataques. Estos métodos son virtuales puros ya que cada tipo de enemigo tiene uno diferente con unos requisitos de activación y actualización distintos. Es por estas razones que deben ser implementados en la clase derivada sí o sí.

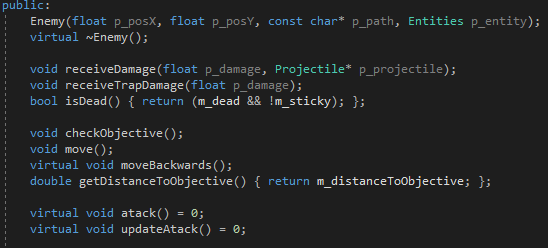


Ilustración - Métodos de la clase Enemigo

### CLASE ENEMIGO GUERRERO (ENEMYWARRIOR)

La clase Enemigo guerrero es derivada de la clase Enemigo. Es el enemigo más básico de todo el juego y más equilibrado. Es el enemigo que menor daño hace, pero se compensa con mayor velocidad de movimiento y resistencia. Su función es perseguir al jugador objetivo y atacarle cuerpo a cuerpo.

### CLASE ENEMIGO A DISTANCIA (ENEMYRANGER)

La clase Enemigo a distancia es derivada de la clase Enemigo. La característica de este enemigo es que su ataque es a distancia a gran velocidad y que tiende a mantener la distancia con el jugador objetivo. Este tipo de enemigo cuenta con una daño de ataque superior al del guerrero, pero tiene la menor resistencia de todos los tipos de enemigos.

### CLASE ENEMIGO CON CARGA (ENEMYCHARGER)

La clase Enemigo con carga es derivada de la clase Enemigo. La característica de este enemigo es que cuando detecta que tiene al jugador en rango se queda inmóvil y vulnerable, pero empieza a cargar un ataque lleno de ira. Una vez cargado el ataque se lanza en línea recta sobre su objetivo. Este tipo de enemigo cuenta con el mayor daño de todos los enemigos y su resistencia es normal.

## CLASE PROYECTIL (PROJECTILE)

La clase Proyectil es derivada de la clase Entidad. Al igual que las anteriores es una clase intermedia que nos permitirá configurar los valores de los múltiples proyectiles existentes.

## CLASE POCION (POTIONS)

La clase Poción es derivada de la clase Entidad. Al igual que las anteriores es una clase intermedia que nos permitirá configurar los valores de las pociones.