LA VENGANZA DE BELIAL (TECNOLOGÍA DE VIDEOJUEGOS)

Grupo 7
David Ibáñez Díaz
Hisam Moreno Moral
Alberto Murillo Paredes
Ángel Oroquieta
Pablo Contreras

ÍNDICE

INTRODUCCIÓN 3

OBJETIVO DE LOS DESARROLLADORES 3

MECÁNICAS DEL JUEGO: REGLAS 3

- Jugador 4
- Habilidades 4
- Inventario e items 5
- Combates 5
- Avance del juego 6
- NPCS 7
- Toma de decisiones 7
- Menús 7

DESARROLLO DEL JUEGO 8

- Diseño de personajes 8
- Desarrollo de la historia 11
- Diseño de niveles 13
- Diseño de enemigos 15
- Diseño de diálogos 17
- Arquitectura del juego 17
- Organización del equipo 18

EXPLICACIÓN TÉCNICA DEL JUEGO 19

- Explicación de colisiones y eventos 19
  - Interacción del jugador con el mapa 19
  - Detección de colisiones y control de eventos 20
- Carga de nuevos Mapas 23
- La tienda 23
- Funcionamiento del Menú 24
- Escenas animadas 25
- Clases relacionadas con los personajes 27
- Clases relacionadas con Items 30
- Clase Habilidad 31
- Clase Inventario 32
- Clases Gestion 32
- Clase relacionadas con NPC 34
- Clases relacionadas con Combate 35
- Otras clases importantes del juego 40

Referencias de otros juegos y libros 41

- Referencias de otros juegos: 41
- Arte y música utilizada 42

Conclusiones y mejoras 43

INTRODUCCIÓN

“La Venganza de Belial” es un juego tipo RPG, basado en títulos antiguos tales como “Final Fantasy” o “Dragon Quest”. Esta aventura paródica guiará a un grupo de tres poco heroicos aventureros por el mundo de Reynos, un mundo de fantasía plagado de peligros y aventuras cliché con un toque cómico en ellas.

El objetivo final del juego es guiar al grupo de aventureros hasta completar la misión que se les ha encomendado, mientras desvelan los misterios que se ocultan tras los acontecimientos que se suceden. Para ello el jugador deberá hacer uso de las habilidades de los tres personajes que controlará para superar los distintos niveles diseñados, plagados de monstruos y peligros.

OBJETIVO DE LOS DESARROLLADORES

El objetivo propuesto por el equipo ha sido el desarrollar un juego de estilo RPG por turnos, desarrollando una historia paródica de una aventura épica, intentando despertar en el jugador un interés por unas mecánicas de juego fáciles de captar.

La base del juego ha sido centrada en dicha historia, intentando guiar al jugador junto a un grupo de aventureros en una misión que un inicio se presenta como “épica” y de vital importancia para el mundo que habitan.

Se ha buscado que la historia despierte intriga sobre los sucesos que ocurren en la aventura mientras que el jugador pueda tener una experiencia más inmersiva mediante la toma de decisiones que afectarán a los acontecimientos y e incluso, el final del juego. Además de ello se ha intentado alimentar el sentimiento de “sorpresa” en algunos puntos, intentando dar algo de impacto en algunos puntos para que la aventura no se convierta en monótona.

Al tratarse de una aventura paródica, podemos decir que uno de nuestros principales objetivos es la de conseguir que el jugador se divierta mientras avanza a lo largo del juego, por ello si durante el curso del juego conseguimos que el jugador se ría, habremos conseguido nuestro objetivo.

MECÁNICAS DEL JUEGO: REGLAS

A continuación, se muestras el conjunto de mecánicas (reglas) que posee el juego, así como las características que lo definen:

Jugador

El jugador cuenta con un equipo de tres personajes jugables, cada uno con sus propias características, fortalezas y defectos. A lo largo de la partida estos personajes irán adquiriendo experiencia mediante la lucha con enemigos, haciéndose más fuertes y dando una sensación de crecimiento al jugador. Dicho crecimiento se verá no solo en la subida de atributos del personaje si no también en su nivel, que crecerá desde el nivel 1 al empezar el juego, hasta un máximo de nivel 25 alcanzable.

Cada uno de los personajes, tanto aliados como enemigos, cuentan con una serie de estadísticas que determinarán su rendimiento en combate. Dichas estadísticas son las siguientes:

- ***Ataque:*** Estadística que determina el daño que pueda causar el personaje al objetivo al que desee golpear
- ***Defensa:*** Al contrario que ataque, reduce el daño causado por los diferentes ataques que reciba por parte de los adversarios
- ***HP:*** Puntos de vida del personaje. Deben ser reducidos a 0 para que dicho personaje muera
- ***MP:*** Puntos de mana con los que cuenta el personaje para llevar a cabo sus habilidades especiales
- ***Velocidad:*** Es la estadística base sumada a una variable aleatoria para determinar la iniciativa en los combates
- ***Crítico:*** Probabilidad de que el personaje cause un daño aumentado al enemigo. Esta estadística no la tienen los enemigos

Habilidades

Las habilidades son las acciones especiales que pueden usar los personajes para sobrepasar ciertas situaciones. Tanto los personajes del jugador como los enemigos cuentan con sus propias habilidades, las cuales podemos dividir en los siguientes grupos:

- ***Habilidades de ataque uniobjetivo:*** Realizan un daño mayor al normal sobre un único objetivo
- ***Habilidades de ataque mutiobjetivo:*** Golpean a varios objetivos al mismo tiempo, haciendo que el siguiente personaje tenga más probabilidad de acabar con cualquiera de ellos
- ***Habilidades de cura:*** Permiten restaurar un porcentaje de vida de un aliado que no esté muerto
- ***Habilidad de resurrección:*** Permite revivir a un personaje con un porcentaje de vida limitado
- ***Habilidad de drenaje:*** Es una variante de ataque uniobjetivo, donde su potencia es inferior, pero permite restaurar un porcentaje de vida proporcional al daño causado

Es interesante mencionar que mientras que el uso de las habilidades por parte del jugador está limitado a su MP, las habilidades de los enemigos pueden ser lanzadas de forma infinita, añadiendo un grado de desafío en especial cuando enfrenten a enemigos fuertes.

Inventario e items

A lo largo de la aventura el jugador contará con un inventario, sobre el cual almacenará diferentes objetos y dinero, necesarios para fortalecer sus personajes y sobrepasar algunas situaciones.

Dicho inventario contará siempre con espacio para cada uno de los tres consumibles presentes en el juego:

- ***Poción de vida:*** Permite la restauración de un porcentaje de vida a un aliado que no esté muerto
- ***Poción de maná:*** Permite la restauración de un porcentaje de los puntos de maná del personaje.
- ***Poción de resurrección:*** Permite resucitar a un aliado cuyo HP ha llegado a 0

Será posible portar hasta un máximo de 10 consumibles de cada tipo dentro de un inventario compartido entre los tres personajes.

Además de estos consumibles, el inventario contará con otros siete espacios para portar armas y armaduras, que podrás encontrar o comprar a lo largo de la aventura. Las armas aportarán ataque y crítico extra a los personajes, mientras que las armaduras proporcionarán más defensa.

Las armas y armaduras comprables en los pueblos han sido diseñadas añadiendo un toque cómico y limitadas por requisitos de dos tipos: un requisito de personaje, cada arma y armadura está diseñada para un único personaje, dándole personalidad al individuo. Y requisito de nivel mínimo, es decir, un personaje de nivel inferior a ese requisito no podrá equiparse el arma, esto hace una especie de sistema de recompensas en el cual el personaje recibe un premio por haber crecido/subido de nivel.

Combates

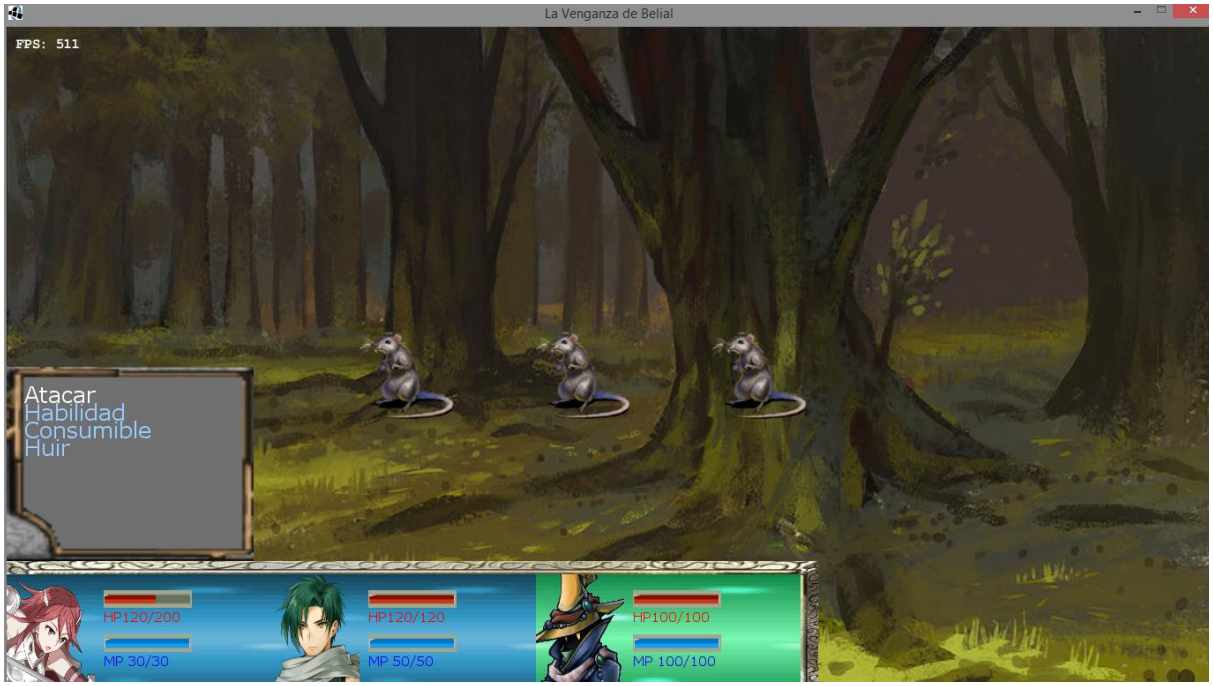
El mecanismo de combate se basa en el sistema de combate por turnos, en el cual los participantes de la refriega usan de forma estratégica sus bazas para ganar la batalla.

Todos los participantes del combate utilizarán su atributo de “Velocidad”, sobre el cual se aplica un valor de aleatoriedad añadido determinando así su “Iniciativa” en dicho combate. Esto implica que los personajes no tendrán siempre la misma “Iniciativa” aunque luchen contra el mismo grupo de enemigos dos veces, permitiendo cierta variedad en los combates.

Una vez establecidos el orden de turnos, cada personaje utiliza el suyo para realizar diferentes acciones. Los enemigos atacan basándose en sus propias estrategias entre las que incluirán poderosos ataques o curas entre ellos.

Por otro lado el jugador cuenta con cuatro posibilidades:

- ***Atacar:*** Ataca a un único objetivo utilizando el valor de su ataque sumado al daño de su arma para generar heridas en los enemigos
- ***Habilidad:*** Usar una de sus habilidades disponibles para dañar a sus enemigos o curar a sus aliados
- ***Consumibles:*** Permite el uso de uno de los tres consumibles mencionados sobre los aliados, siempre y cuando se tengan existencias de los mismos y cumpla los requisitos para poder usarlos sobre ellos
- ***Huir:*** Intentar huir de la batalla junto con todo el grupo



Al finalizar los combates en los que se salga victorioso, aquellos personajes que sigan con vida ganarán experiencia con la que podrán subir de nivel y aprender nuevas habilidades. Por otro lado, aquellos que hayan muerto durante el combate no ganarán experiencia y quedará la posibilidad de que queden rezagados.
Además de la experiencia se recibe una compensación monetaria que podrá ser utilizar en las tiendas que que hay a lo largo del juego.
El funcionamiento técnico del desarrollo del combate se explica más adelante.

Avance del juego

Para poder proseguir en el juego, el jugador deberá avanzar por los diferentes mapas/niveles propuestos en la historia, cumpliendo los objetivos impuestos en cada uno, alcanzando los eventos clave y derrotando a los jefes de mapa que se interpondrán en su camino.

NPCS

A lo largo de los diferentes mapas el jugador podrá encontrar diferentes NPCs, relevantes o no:

- ***NPCs de campo:*** Aquellos que se pueden encontrar sin más andando por los mapas. Chocando con ellos activarás un pequeño diálogo
- ***Tiendas:*** Permite comprar y vender diferentes artículos en función del mapa
- ***Healers:*** Restauran el HP y MP al completo
- ***Personajes de escenas y bosses:*** Personajes que se ven en las diferentes escenas animadas del juego. En algunas ocasiones, hay que enfrentarse a ellos

Toma de decisiones

Para crear una experiencia algo más inmersiva en el juego, se han añadido diversas decisiones y “acertijos” en los que el jugador deberá elegir entre diversas opciones, decidiendo así en algunos casos el rumbo que tomará el juego, dando así algo más de libertad en un juego que de otra forma puede parecer demasiado lineal.

Menús

A lo largo del juego el jugador debe desplazarse por varios menús que facilitarán su aventura:

- ***Menú de inicio:*** Presente al arrancar el juego. Te da la opción de empezar una nueva partida desde cero o continuar desde el último punto de guardado
- ***Menú de pausa:*** Usando la tecla Escape en el mapa, el jugador puede acceder a este menú que cuenta con las siguientes opciones:
    - *Continuar:* Volver al mapa
    - *Guardar partida:* Solo se puede guardar una partida a la vez. Esta opción permite guardar el estado del juego salvo por la posición del jugador dentro del mapa, que deberá iniciar desde el principio cuando se cargue dicha partida.
    - *Cargar partida:* Devuelve al jugador al estado de juego tras su último guardado, a excepción de su posición en el mapa.
    - *Personajes:* Permite acceder a los datos de los personajes dentro del grupo, dentro de cada cual se pueden ver sus características y estadísticas, así como ver sus habilidades y usar aquellas que se puedan utilizar sobre aliados. También se puede equipar nuevas armas y armaduras
    - *Inventario:* Permite ver los objetos dentro del inventario, y utilizar los consumibles sobre los aliados
    - *Salir:* Permite cerrar el juego

La explicación técnica acerca del desarrollo del menú está detallada más adelante.

Además de estos menús principales el uso de las tiendas, así como la elección de acciones dentro de los combates se llevan a cabo a través de sus propios menús de opciones.

DESARROLLO DEL JUEGO

Además de los objetivos explicados en apartados anteriores también hay que tener en cuenta que se ha intentado que el juego presente sorpresas y giros de trama inesperados, de forma que el jugador pueda tener una experiencia más impactante a medida que avanza en el juego.

Diseño de personajes

Es esencial para comprender parte de las decisiones tomadas en el desarrollo del juego, entender cómo se han diseñado los personajes junto con el trasfondo que se les ha dado, por ello este apartado se dedica a explicar aquellos personajes de más relevancia junto con todos los detalles de su diseño.

El escuadrón “F” está compuesto por el grupo de aventureros que manejara el jugador a lo largo de la aventura. Dicho equipo está compuesto por:

- ***Horacia:*** Este personaje ha sido diseñado como una mujer de unos 20 años, hija de un antiguo caballero de nombre. Su personalidad es la de alguien simple, con toques de cobardía e indecisión, haciéndolo un personaje que puede llegar a impactar

debido a lo incompetente y débil que puede parecer. Sus objetivos han sido determinados como los de conseguir una vida fácil e ideal, honrando la memoria de su difunto padre.
Contrastando con su personalidad se ha decidido que este personaje actúe como la capitana del grupo, dando un fuerte impacto al jugador frente al contraste de su posición y personalidad. Además de ello, para acrecentar este contraste, este personaje ha sido desarrollado con el rol de "tanque" del grupo, habiendo destacado en su creación su vitalidad y su defensa.
Como justificación al rol que se la ha dado y a algunas estrategias de los enemigos que se centrarán en golpearla, se ha pensado que dichos ataques son dirigidos hacia ella debido a su personalidad y presencia, que la hacen parecer la más débil de grupo.
Entre las habilidades que se le han atribuido están, las habilidades de cura y ataque principalmente, haciéndola un personaje duradero y multifunción. No obstante, para compensar sus fortalezas se la ha diseñado como un personaje con baja velocidad, poco MP y un cantidad decente, pero sin exceder de ataque.
Como parte de su rol las armas que puede equiparse son las de cualquier caballero: espadas, hachas y lanzas. Las armaduras que puede equiparse son más pesadas y resistentes que las de sus compañeros

- ***Mordeim:*** Este personaje ha sido diseñado como un joven hombre en sus 20 años, con un pasado desconocido. Su personalidad es la de alguien arrogante, libre y que se cree por encima de todo y todos. Este personaje puede parecer en muchos momentos que va con la corriente hasta que decide hacer lo que quiere, llevando a situaciones cómicas e impactantes debido a ello. Aprovechando su personalidad se decidió que su rol fuera el de "DPS", es decir que fuera la ofensiva genérica del grupo, haciendo que su fuerza y velocidad sean su punto fuerte. Sus golpe básicos se convierten en una buena fuente de daño, mientras que su habilidades destacan por llegar a ser mortales.
  Puesto que cree poder hacer cualquier cosa, no tiene unos objetivos definidos simplemente decide seguir la corriente y hacer lo que se le antoja cuando lo desee.
  Para compensar su potencia ofensiva, sus demás estadísticas han sido diseñadas como promedio, llegando a tener un deficiencia en cuanto vida se refiere, siendo ligeramente más débil que los otro personajes del grupo.
  Sus armas son principalmente cuchillos, representando su arrogancia mediante el pensamiento de que él puede derrotar a cualquier enemigo incluso con el arma más pequeña. Sus armaduras son ligeras prendas de cuero que soportan los suficiente sin quitarle movilidad
- ***Kibito:*** Este hombre de edad desconocida, se ha propuesto como alguien que, tras varios experimentos mágicos acabó teniendo un aspecto poco humano. Siendo un personaje de tipo intelectual, su personalidad pesimista y victimita lo hacen quejarse constantemente de todo lo que ocurre al tiempo que explica el mundo gracias a sus conocimientos, haciéndolo perfecto para ayudar al jugador a aprender el mundo y su ambientación mientras aporta un toque cómico.
  Al igual que Mordeim no presenta un objetivo perfectamente definido, si no que se mueve por la curiosidad de ver hacia donde le guían sus misiones intentando no morir en el proceso.
  Su misión es la de "mago", habiendo combinado tanto las características de un "mago ofensivo" y un "curandero", haciendo que su principal punto sea su MP y la

variedad y potencia de sus habilidades. De esta forma presenta todos los tipos de habilidades presentes en el juego a excepción del drenaje.
Como desventaja a sus fuerte habilidades este personaje no posee una potencia de ataque útil en combate y su defensa es bastante baja, por suerte la gran mayoría de los enemigos no lo tendrán como objetivo. Debido a su debilidad física este personaje no puede equiparse grandes armaduras y sus armas son principalmente bastones que le permiten utilizar habilidades de mejor forma

En la imagen se puede ver el avatar de Horacia, Mordeim y Kibito en la barra de estados de un combate.

Siendo Cardinal la organización principal en la que se basan los acontecimientos como una fuerza opuesta al mal en el mundo, los personajes de esta que se han querido destacar por no sobrecargar al jugador con información son limitados pero remarcables e importantes en el desarrollo de la historia.

- ★ ***Hestia:*** Diseñada como una joven seria con sus labores como sacerdotisa de la organización, se presenta a sí misma como una joven preocupada por el futuro del mundo y la organización, encargando la misión principal a los personajes del escuadrón “F”. Durante las escenas en las que hace aparición se puede observar su preocupación hacia los personaje mientras mantiene un perfil bajo frente a sus compañeros de los altos cargos, haciéndola ver en todo momento como una aliada del jugador.
  ¡¡¡**ALERTA SPOILER!!!** Cuando se llega al final del juego y usándola como elemento principal de sorpresa, este personaje resulta haber engañado a todos siendo su objetivo real el de liberar al gran demonio que se menciona durante la historia, además para apelar a una justificación de sus acciones, resulta ser la propia hija del demonio, haciendo más creíble su situación y motivación.
- ★ ***Archi:*** Originalmente este personaje fue pensado como el “archienemigo principal” que se presenta como opuesto al jugador durante el juego. Sus primeras menciones introductorias lo presentan como el líder de todos los escuadrones y alto cargo de Cardinal, habiendo querido dar una personalidad seria y dura, siempre dedicado al trabajo como se esperaría de un gran líder.
  Respetado por todos y viendo que uno de sus escuadrones más problemáticos no da señales de vida, decide pensar que están haciendo algo en contra de sus creencias y mandar a sus agentes a detenerlos.
- ★ ***Capitanes, generales y soldados:*** Sin intentar darles especial relevancia se presenta a estos personajes como fieles seguidores de Archi cuya única motivación es seguir sus órdenes y los ideales de Cardinal.

El resto de personajes diseñados no cobran una relevancia real en el juego más allá de proporcionar una ambientación y apoyo al jugador, mediante el aporte de información, curas o venta y compra de objetos. Por este motivo muchos de sus diálogos han sido preparados

de acuerdo al nivel en el que se encuentran intentando sacar alguna risa al jugador e intentando alentar a buscar todos los NPCs del juego en busca de ver que dicen.

Desarrollo de la historia

Hemos basado la historia de nuestros protagonistas en las historias de los Final Fantasy, Dragon Quest,etc. añadiéndole un tono de parodia siendo este juego el Quijote de los RPGs.

La historia trata de un mundo medieval parecido al The Witcher en el que aparecen NPCs curativos, vendedores y gente que te informa de forma gratuita, no obstante, al contrario que en otros juegos del estilo, los personajes secundarios no aportan misiones secundarias bajo la premisa de “Te acabo de conocer y no me fio de que un desconocido se meta en mi vida”.

De aquí en adelante se explicará la trama que sucede en el juego con detalle, intentando explicar lo que los diseñadores han intentado hacer en cada momento, por tanto, los siguientes textos contienen “spoilers” del juego.

La trama del juego comienza con una pequeña presentación del mundo en el que se desarrolla y su historia, presentando un mundo que hace muchos años se halló en guerra debido a un gran demonio de otra dimensión llamado Belial. Dicha guerra y la liberación del demonio fueron impedidas por Cardinal, una organización creada para restaurar y preservar la paz en el mundo por la eternidad, quienes sellaron a Belial gracias a los “Cuatro Sellos de Luci”.
En este contexto en el que se ha intentado salvar el cliché de “antigua amenaza sellada” de muchos RPGs, se presentan las primeras escenas, en las cuales la gran Sacerdotisa de Cardinal en la actualidad encarga una misión secreta al “Escuadrón F”.
La misión es la de restaurar los sellos de Luci, que según Hestia se están deteriorando, además como excusa al secretismo dice no fiarse de las cosas que están pasando en los altos cargos de Cardinal, dando a entender que “alguien” está intentando despertar al demonio, proporcionando un elemento de intriga desde el primer momento.
Dicha misión como no, es la excusa para que nuestros personajes se embarquen en la misión, que en principio se plantea como un simple paseo de incógnito.

Tras el primer nivel que sirve como tutorial, se presenta a los personajes charlando despreocupados cuando un evento tan inesperado como una explosión provocada por unos bandidos, los lanza al fondo de un cañón, hacia un bosque supuestamente maldito. De esta forma se ha intentado conseguir el primer cambio impactante en la trama.
En el interior de dicho bosque los personajes lucharán contra los monstruos que allí habitan, en busca de una salida hacia la ciudad a la que se dirigían.

A medida que avanzan por el bosque se intenta que el jugador conozca un poco más acerca de su ambientación mediante diálogos entre los personajes hasta que, al llegar al final se presenta Yggdrasil, una deidad del bosque corrompida por el paso de los años y la

malicia desprendida por el sello cercano y origen de las leyendas de la maldición del bosque, al que los personajes se ven obligados derrotar.

Tras derrotar a este jefe, los personajes descubren que uno de los sellos que buscaban se encuentra justo al lado y deciden repararlo, presentando una escena que intenta romper el epicismo de la tarea, rompiendo con la idea de “leyenda épica”.
Tras esto los personajes observan como el árbol parece diferente y dudan de que hacer con él, presentándose uno de los elementos de decisión del juego, donde decidirán si rematar al árbol que les ha causado tantos problemas o dejarlo vivo y ver qué ocurre.
Dejándolo vivo se encontrarán a un NPC que se presenta como un participante de la antigua guerra y, de nuevo buscando una parodia de una leyenda, se ha convertido con el paso del tiempo en un anciano cuenta batallitas.
La otra opción lleva a Mordeim a quemar el bosque para asegurarse de que el árbol no se levanta de nuevo, dejando atrás una estela de destrucción que remarca la personalidad del personaje.

Tras abandonar el bosque los personajes llegan a su destino, donde se les presenta que los mismos bandidos que han provocado su caída al bosque, están asaltando la ciudad y les impiden llegar a su destino.
Con este nuevo problema presente, los personajes deben derrotar al jefe de los bandidos y con ello poder partir a su destino. De esta forma se presenta una pequeña transición entre niveles que ayuda al jugador a nivelarse o adaptarse a las circunstancias que van surgiendo, al mismo tiempo que caemos en el cliché del héroe que se encuentra con diversos problemas, algunos sin relación a su misión, a medida que avanza hacia su objetivo.

Tras resolver sus problemas en el puerto, los protagonistas consiguen llegar a su siguiente destino, siendo advertidos de antemano que algo extraño está ocurriendo en él. Así al llegar se encuentran con una ciudad en decadencia, más parecido a los restos de una ciudad que a una ciudad propiamente dicha, la falta de gente resalta la falta de vida en la ciudad.

Sin siquiera saberlo y solo con su objetivo en mente, los héroes exploran las catacumbas de la ciudad en busca del sello, metiéndose de lleno en los planes de una secta que está reviviendo a los muertos e intenta liberar a Belial.
La secta, pensando que estos han venido a detenerles se enfrentan a los protagonistas, creando una situación de conflicto a través de la confusión de ambos bandos, tras cuya conclusión los héroes no llegan a entender siquiera lo que han logrado hacer.

Dirigiéndose así a su último destino, escenas de lo que está ocurriendo en Cardinal se muestran, descubriendo, así como la trama avanza en paralelo a la aventura de los protagonistas y mostrando como su misión secreta ha sido descubierta de una u otra manera.
Con esto se presenta un nuevo conflicto que llevaba en espera desde el inicio de la trama, de forma que, Cardinal se vuelve el enemigo de los protagonistas, intentando impedir que sigan manipulando los sellos, lo cual se muestra al enviar a un capitán en su búsqueda, convirtiéndose en su enemigo en las montañas donde buscan el sello.

En las montañas los personajes se ha intentado dar la mentalidad de “mundo abierto”, intentando que el mapa no tenga un rumbo definido y que el jugador deba alcanzar el objetivo mediante la exploración del mapa.

Finalmente, los personajes deciden fijar su rumbo al último sello, descubriéndose que este se encuentra en Cardinal, y presentando una situación cómica que jamás se presentaría en una historia legendaria común.
Sin muchas más opciones que volver para continuar con su misión, los personajes regresan al punto de partida donde se encuentran con que el ejército entero de Cardinal intenta detenerlos sin dar explicaciones.
Confusos y teniendo en cuenta el “porqué” dado por la sacerdotisa para el secretismo, los personajes continúan avanzando hacia el último sello con el pensamiento de que están haciendo lo correcto en vistas de la autoridad que les ha encargado la misión, solo para descubrir la terrible verdad al llegar al final del último nivel.

La misma persona que le había encomendado su gran misión, la sacerdotisa Hestia, los había engañado, había conseguido que liberasen los sellos que contenían a Belial para traerlo a este mundo. Con esta revelación el último sello es liberado y Belial aparece en escena, presentando la verdad acerca Hestia y sus objetivos.

Toda esta última escena busca presentar el clímax y resolver todos los misterios del juego, jugando con la sorpresa al descubrir la verdad de la aventura y descubrir finalmente la orientación de cada facción en la historia.

Ante este clímax al jugador de le presentan dos opciones completamente opuestas: continuar apoyando a Belial hasta el final en busca del beneficio personal o enmendar su error intentando detener al demonio.

Cualquiera que sea la elección los protagonistas deberán acarrear con ella, enfrentándose al bando contrario y derrotándolo para así alcanzar el destino que hayan elegido y conseguir sus ambiciones.

Diseño de niveles

Los niveles han sido seleccionados y diseñados para coincidir con los escenarios más cliché de las aventuras de fantasía, pudiendo así transmitir una sensación de mundo fantástico y algo de nostalgia para aquellos acostumbrados a los RPG de fantasía. Los niveles se dividen en:

- ***Nivel tutorial:*** Ha sido diseñado con la idea principal de enseñar al jugador las dinámicas del juego sobre cómo interactuar con el mundo en general. Además, también ha sido uno de los niveles que han servido al equipo para aprender a diseñar los mapas y eventos que componen el juego, es decir es un nivel de aprendizaje tanto para el jugador como para el creador. En relación con la historia

este punto equivaldría a los preparativos de los personajes antes de aventurarse en su misión

- ***Nivel bosque:*** Ha sido desarrollado bajo una temática de bosque de fantasía, repleto de enemigos propios del mismo como goblins y arañas, que en un inicio no supondrán una dificultad excesiva para el jugador, aclimatándose así a las mecánicas de combate. Para ayudar en este punto de aprendizaje, el inicio del nivel cuenta con un tutorial de combate en el cual los personajes debatirán las diversas opciones que tiene el jugador en las peleas. Este bosque y su jefe final se ven presentados en una serie de escenas animadas antes y durante el mismo, que harán al jugador sumergirse en la historia

  Es ante todo, un nivel de calentamiento para que el jugador se vaya haciendo con las mecánicas del juego sin tener demasiados problemas superándolo.
- ***Nivel puerto:*** Como un buen puerto, cuenta con agua y barcos. Este mapa ha sido pensado como un nivel de transición, en el cual se presenta un problema tanto para el pueblo como para nuestros personajes, el cual debe resolver el jugador para continuar con la historia. Además, para apoyar el crecimiento del jugador, este mapa cuenta con enemigos opcionales contra los que se podrá luchar repetidamente hasta alcanzar un nivel adecuado en el caso de que se encuentre en dificultades para avanzar
- ***Nivel de catacumbas:*** Siguiendo con la historia, este nivel está diseñado como un pueblo en decadencias, arruinado por una malvada secta. Los personajes, siguiendo con su misión se verán envueltos con ella casi sin saberlo, debiendo adentrarse en unas catacumbas llenas de zombies y esqueletos.

  En este nivel la dificultad aumenta ligeramente, sobre todo por la disposición de jefes y curanderos algo menos amigable, y ante todo, habiendo aumentado la aparición de enemigos de forma que dé la impresión de una auténtica horda de muertos vivientes que pueden aparecer en cualquier punto del mapa.
- ***Nivel de montaña:*** Siendo el mapa que lleva al clímax de la aventura, este nivel ha sido desarrollado con la idea de “libertad” y “sorpresa”. A lo largo de este nivel se sucederán varias escenas que intrigan al jugador sobre lo que realmente está sucediendo. Además, el mapa ha sido diseñado de forma que no hay un único camino a seguir por lo que deberá encontrar la salida por sí mismo.

  Además de ellos el jugador se encontrará al final del nivel con un jefe clásico, un dragón, que hará honor a su nombre siendo posiblemente el boss más difícil con el que el jugador se pueda encontrar hasta el momento.
- ***Nivel Cardinal:*** De vuelta a la base de Cardinal, los jugadores deberán abrirse paso por los salones y pasillos que llevan hasta su último objetivo, enfrentándose a constantes hordas de enemigos. Este mapa ha intentado despertar la esencia de “último nivel”, haciendo uso de una banda sonora los más impactante posible e intentando dar la sensación de incesantes hordas de enemigos que se cruzan para detener su objetivo, no obstante, para compensar esa constante salida de enemigos se ha intentado que los enemigos decaigan un poco en potencia de fuego, buscando más el agotamiento de los personajes tras constante batallas que en una sola.

  Al llegar al final del nivel, se desvelarán los misterios de la aventura y se le dará a jugador la libertad de elegir su “destino” decidiendo entre uno de los dos finales alternativos que se han diseñado.

Todos los niveles cuentan con sus propios grupos de enemigos como se explicará a continuación, no obstante, en todos ellos de forma individual y en todo el juego en general se ha intentado que estos aporten una dificultad progresiva a medida que el jugador avanza a lo largo del juego. Este efecto se ha intentado desarrollar mediante el crecimiento del nivel de los enemigos conjunto al nivel de los jugadores, incluso dentro del mismo nivel y buscando que los enfrentamientos contra los jefes de nivel fueran cada vez más desafiantes.

El juego ha sido testeado por los diseñadores habiendo nivelado en mayor o menor medida todos y cada uno de los niveles del juego de acuerdo al siguiente progreso:

- *Bosque:* Comenzando a nivel 1 es bastante asequible para cualquier jugador y se puede terminar entre el nivel 5-6 del grupo, habiendo terminado la mayoría de los test, finalizando el jefe a nivel 6
- *Puerto:* Preparado para entrar a nivel 6-7 y, dado sus circunstancias especiales nivelado para presentar un desafío menor que puede ser superables a nivel 7-8, aunque es recomendable conseguir nivel 9
- *Catacumbas:* Se ha testado su superación obteniendo en general la posibilidad de superarlo entrando a nivel 8 y saliendo a nivel 14
- *Montaña:* Entrando a nivel 14 el boss es superable a nivel 18 aunque conlleva cierta dificultad si se hace de este método.
- *Cardinal :* Los enemigos no presentan una amenaza que sobrecargue al jugador con un solo combate a nivel 18, no obstante, el desgaste tras varias oleadas, el largo del mapa y el mini jefe presente en el mapa conllevan un desafío considerable pudiendo ser derrotado a nivel 22-23.
- *Jefes finales:* Se recomienda que mínimo el jugador sea nivel 23 para poder pasarlos con un nivel de dificultad asequible

Diseño de enemigos

El diseño de enemigos ha sido llevado a cabo en función de los mapas creado en el diseño de niveles, y dentro de cada “grupo de mapa” los enemigos se han diferenciado en dos tipos:

- ***Enemigos de campo:*** Desarrollados haciéndolos coincidir con la temática de cada mapa, de forma que la reiterada aparición de los mismos en al avance a través del nivel resultará algo normal. Para no sobrecargar al jugador con una cantidad masiva de enemigos y así frustrar se ha decidido que, independientemente de la variante de enemigos de cada mapa, los enfrentamientos siempre serán contra grupos de tres enemigos, creando una sensación de igualdad entre ambos bandos.
  Cada tipo de enemigo cuenta con su propia estrategia en función de su concepto:
  - *Enemigos de inteligencia reducida:* Atacan principalmente a aquel aliado que posea más vida o que se encuentre más cerca de ellos.
  - *Enemigos inteligentes:* Atacan a los aliados más debilitados.
  - *Enemigos duraderos:* Aquellos que además de atacar se curan o regeneran a otros.
  - *Enemigos especialmente poderosos:* Capaces de atacar a todo el grupo a la vez.

- *Enemigos mortales:* Capaces de matar de un único golpe.

Cabe destacar que la mayoría de las estrategias tenderán a asestar su golpe a Horacia, dando la sensación, no solo de que se encuentra en primera línea de batalla defendiendo al resto del grupo, si no que también los enemigos la elegirían como objetivo al debido a su personalidad que la hace visualmente débil
Los enemigos por nivel son los siguientes:

- Bosque: Ratas, arañas y goblins
- Puerto: Bandidos
- Catacumbas: Esqueletos, zombies, fanáticos y mímicos
- Montañas: Slimes, minotauros, "murciégalos" y grifos
- Cardinal: Capitán, soldados y paladines

- ***Enemigos tipo jefe (bosses):*** Han sido diseñados de acuerdo a la ambientación y preparación de cada nivel, utilizados además para apoyar la presentación de diferentes escenas animadas.
  Cada jefe contará con habilidades propias correspondientes a su presentación, además de ello se diferencian principalmente de los enemigos de campo en que la estrategia de ataque está mucho más conseguida y que la cantidad de vida que presentan es mucho más elevada, diseñados especialmente para combatir en desventaja de 3 contra 1 y, aun así, presentar cierta dificultad al jugador. Dichos jefes además presentarán el final de cada nivel. Los jefes diseñados son los siguientes:
  - *Yggdrasil:* Es el jefe del primer nivel, siendo un árbol antiguo corrompido por el paso del tiempo. Entre sus habilidades principales está la de curarse a través de sus raíces y generar una miasma que dañará de forma constante a los aliados. Se trata al fin y al cabo de un boss menor para que el jugador vaya entrando en calor.
  - *Gran Bandido Crow:* Este peligroso bandido utilizará estrategias bastante humanas tendiendo principalmente a golpear a aquel que tiene al frente, es decir simulará golpear a todos los aliados de forma similar. Este jefe ha sido preparado más como un miniboss que como un jefe real de nivel, sus características y estrategias son idénticas a las de sus secuaces, pero sus estadísticas están muy por encima.
  - *Parca y líder Fanático:* Estos jefes vendrán en grupo, lanzando guadañazos a diestro y siniestro. La parca contará con una probabilidad de muerte instantánea, mientras que líder fanático intentará acabar con el aliado que más daño cause, demostrando una pequeña inteligencia al intentar acabar con el enemigo más peligroso.
  - *Bellafonte:* Es uno de los enviados de Cardinal para hacer regresar al grupo de su aventura, se centrará en atacar a todo el el grupo por igual, dando preferencia a aquel que se presente más débil.
  - *Dragón de las montañas:* Su aliento de fuego golpeará a todo el grupo hasta calcinarlo, mientras que su cola y garras le permitirán atacar a varios objetivos a la vez o varias veces a uno. Será un duro rival puesto que poseerá una gran cantidad de vida. Por suerte su limitada inteligencia le hará atacar principalmente a aquellos que se encuentren más cerca de él.
  - *Archi y General:* Se luchará contra ellos en caso de querer enfrentarse a Archi en vez de Belial. Archi intentará bajar la vida de nuestros personajes atacando a todos a la vez, para luego curarse las heridas recibidas e intentar

contraatacar con un ataque potente. Por su parte, el General, atacará de forma aleatoria a un objetivo o golpeará a todos a la vez.

- *Belial y su hija:* Si por el contrario, se decide que el enfrentamiento final sea contra Belial, también se tendrá que luchar contra su hija. Belial atacará a todos a la vez, hará un ataque fuerte contra un objetivo o atacará y se curará sus heridas. Por su parte, la hija de Belial, atacará parecido a su padre pero en vez de curarse haciendo daño a uno de los aliados, lo hará con una habilidad de cura.

Para hacer la experiencia lo menos aburrida posible, se ha intentado que en todo momento el nivel de los enemigos se adapte al del jugador dentro del margen del nivel, intentado mostrar una experiencia desafiante, sobre todo de cara a los bosses de nivel.

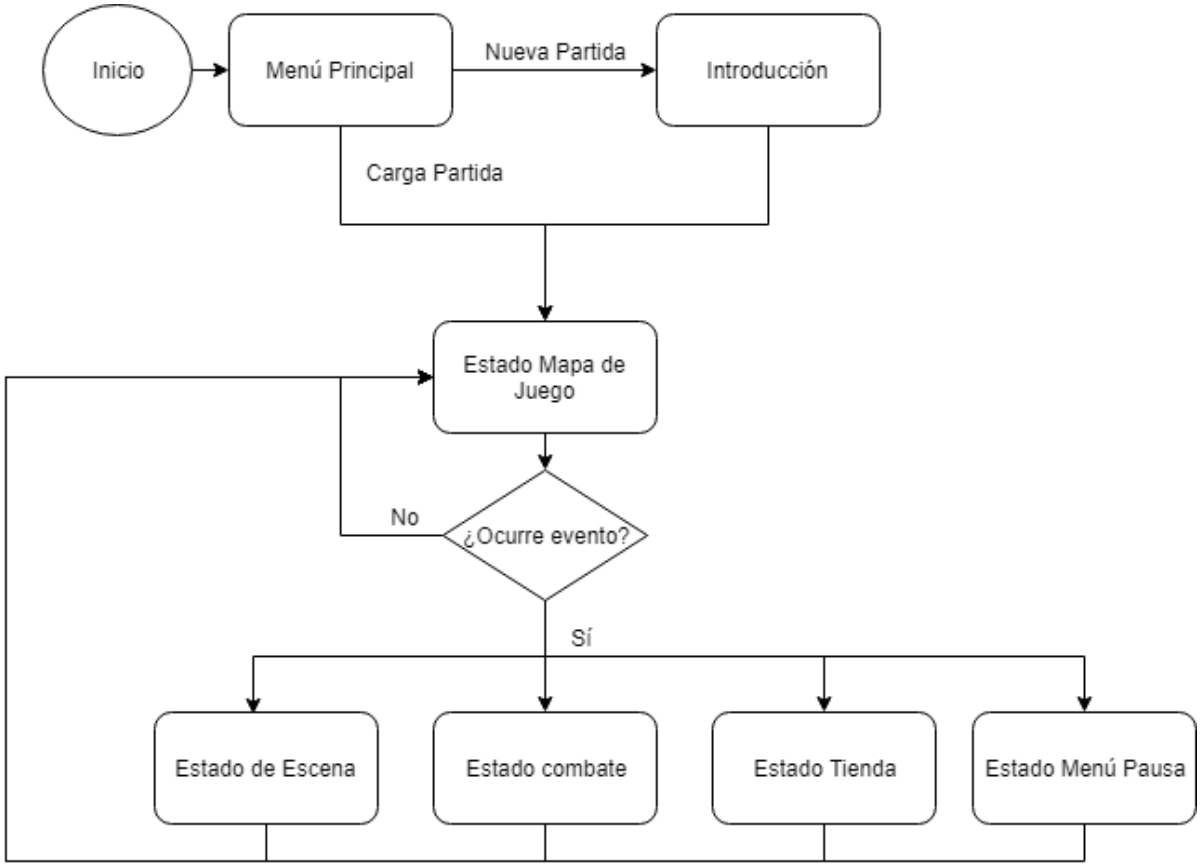
Diseño de diálogos

Como se ha mencionado repetidamente, al tratarse de una parodia, se ha intentado que todas las conversaciones y textos contengan cierto toque de humor, contrastando con la sensación de epicidad que cualquier otro juego del género suele intentar aportar.

Arquitectura del juego

Como es propio del motor utilizado, Slick2d, el juego utiliza dos bucles de control que manejan el control del juego y su renderizado. En nuestro caso, poseemos varios bucles de este tipo, en función del estado del juego en el que estemos para manejar los diversos estados y situaciones del juego mediante diversas clases en vez de utilizar un único bucle de cada tipo, esto quiere decir pues que, en cada uno de los estados se ejecutará un bucle de cada tipo, pero con diferente comportamiento en cada estado.

Basándonos en lo dicho, el juego se ha realizado utilizando una arquitectura de máquinas de estado. Además de ello, para realizar el cambio entre los estados se ha utilizado un subsistema basado en eventos, los cuales disparan los diversos estados diseñados.



Los eventos que disparan los cambios entre estados van desde colisiones en el mapa de juego a “inputs” por parte del jugador en las teclas de juego, además de ello en algunos momentos desde un “Estado de Escena” se dispara directamente un Estado Combate.

Organización del equipo

A lo largo de los meses que ha durado la asignatura, el equipo ha optado por depender menos de los métodos de comunicación que nos aporta GitHub, utilizando un método de reuniones presenciales semanales. Mediante este método el equipo se ha reunido de una a dos veces por semana, debatiendo en persona las tareas que debía ejecutar cada uno y resolviendo los problemas acerca del juego.

Las primeras dos semanas fueron dedicadas a la idea del juego y planteamiento de las reglas que utilizará el mismo, decidiendo por unanimidad los resultados mencionados en el apartado correspondiente.

Las tareas semanales y metas eran planteadas a cada integrante del juego, intentando que para la siguiente reunión se hubiera avanzado en el desarrollo en varios aspectos o, como mínimo, se pudieran detectar los problemas que impedían avanzar.

De esta forma se puede decir que hemos estado aplicando un método iterativo al desarrollo del juego en el cual, el grupo realizaba un análisis y un diseño de lo que se necesitaba

hacer, uno o varios miembros lo implementaban y probaban, trayendo así los resultados al grupo para volver a analizar si todo estaba correcto.

El mayor problema en el momento de organizar al equipo ha sido gestionar las reuniones debido a los horarios individuales de cada miembro del equipo, no obstante, en mayor o menor medida ese problema ha sido resuelto gracias a la predisposición participativa del grupo.

Se ha utilizado el sistema de issue de GitHub principalmente para dejar constancia de varias resoluciones de las reuniones presenciales para aquellos miembros que no hubieran podido asistir o para resolver algunos problemas/dudas en los momentos en lo que reunirnos no era posible. Al principio también se usó la blackboard debido a que no teníamos acceso aún de GitHub.

Aunque de alguna forma u otra todos los miembros han estado presentes en todas las tareas del desarrollo del juego, se puede atribuir varios trabajos en especial a los distintos miembros:

- ***Hisam:*** Artista jefe, Artista técnico, Audio, Inteligencia Artificial, Interfaces, Herramientas, Equipo de pruebas, Equipo de Diseño, Guionista principal.
- ***Alberto:*** Programador jefe, Equipo de Diseño, Inteligencia Artificial, físicas, Herramientas, Equipo de pruebas.
- ***Ángel:*** Programador, Equipo de Diseño, Inteligencia Artificial, físicas, Herramientas, Equipo de pruebas.
- ***David:*** Jefe de proyecto, Equipo de diseño, programador, diseñador gráfico, audio, interfaces, herramientas, equipo de pruebas, guionista secundario, artista secundario.
- ***Pablo:*** Equipo de diseño, programador.

EXPLICACIÓN TÉCNICA DEL JUEGO

Explicación de colisiones y eventos

Una de las partes más importantes del juego es poder controlar las colisiones del jugador en el mapa y de los eventos que se generan en dicho mapa.

Interacción del jugador con el mapa

Para explicar cómo el jugador interactúa con el mapa y los diferentes elementos, primero es necesario explicar las dos clases fundamentales sobre las que se crea la mecánica del juego, más allá de los combates, menús y personajes del mismo.
La primera clase fundamental es “EstadoMapaJuego”. Esta clase que hereda de “BasicGameState” es la encarga de renderizar, actualizar y controlar las acciones del jugador en el mapa. Para ello cuenta con un objeto de clase “TiledMap”, del cual se extraen todos los datos y atributos necesarios para el control de eventos y colisiones con el mapa

mediante código. Al crear dicho objeto, denominado “map” de aquí en adelante, se han diseñado 3 funciones principales para obtener el mapeado de todos los atributos de tipo booleano grabados en el mapa. Para hacer este “mapeado” de atributos mediante código, se implementan varias matrices, una por cada atributo implementado, de tamaño equivalente al número de “tileds” o cuadrados que posee el mapa cargado en el momento, haciendo que cada posición de la matriz equivalga al valor del atributo que trata en la misma posición del cuadrado del mapa.

Esto quiere decir que sabiendo en cada momento sobre que cuadrado se encuentra el jugador, para averiguar si hay algún tipo de evento o colisión en ese lugar solo debemos revisar dicha posición sobre las matrices de atributos. A modo de ejemplo: Mapa 7x7 y ejemplo de dos de sus matrices de control de atributos.

TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE

Array de control de colisiones

FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	TRUE	FALSE	FALSE

Array de control de NPC

Para controlar la posición del jugador, así como aquello con lo que choca y poder hacer la comprobación correspondiente, entra en juego la segunda clase: “Heroe”. Esta clase contiene el código que permite realizar tres tareas de gran importancia.
En primer lugar utilizando como dato de entrada la tecla de dirección que se esté pulsando en ese momento o la última pulsada, si no se está pulsando ninguna, actualiza como debe renderizar al héroe en la pantalla, ya sea quieto mirando hacia un lado o moviéndose en una dirección mientras la animación se ejecuta fotograma a fotograma.
En segundo lugar contiene su propia posición dentro de un vector bidimensional, dicho vector contiene la posición que guarda con respecto al mapa en el que se encuentra en píxeles, es decir nos proporciona la información donde deberemos dibujar a nuestro jugador
Además de ello, conociendo esta posición y puesto que conocemos perfectamente el ancho y alto de los cuadrados de nuestra cuadrícula, haciendo una simple transformación podemos conocer sobre qué cuadro del mapa estamos dando lugar a la tercera tarea importante que realiza esta clase, el control de colisiones.

Detección de colisiones y control de eventos

Como hemos mencionado el control de colisiones se lleva a cabo dentro de la clase “Heroe”, pero para ello es necesario que cuente con los arrays de control de atributos propuestos dentro de la clase “EstadoMapaJuego”, por ello ambas clases deben guardar algún tipo de relación. En nuestro caso nos hemos decantado por el uso de clases

anidadas, es decir crearemos un único objeto de la clase “EstadoMapaJuego” en cuyo interior se creará un atributo que se corresponda con el un objeto de la clase “Heroe”.
Con esto hecho contamos con la posibilidad de operar todas las funciones de ambas clases desde un único bucle de control del juego.

La colisión con estos atributos se hace relacionando la trayectoria del avatar del jugador con las casillas que contienen los atributos. Para esto utilizamos como datos de entrada los botones de dirección que sirven como mandos del control de movimiento, la posición que tenga el jugador en un momento determinado y por último la casilla sobre la que se prevé que el jugador se encontrará atendiendo a los parámetros anteriores. Tras encontrar la casilla sobre la que se prevé que el jugador estará siguiendo esa dirección solo debemos analizar su posición en los diferentes arrays para encontrar si habrá o no algún efecto de respuesta.

Un buen ejemplo de esto sería un jugador pulsando la flecha de dirección hacia la derecha que se encuentra en la casilla (10,15). Atendiendo a estos dos factores se puede predecir que su posición en el eje X aumentará y su siguiente paso lo situará sobre la casilla (11,15). Con esta información, si quisiéramos comprobar si la siguiente casilla es accesible solo deberíamos comprobar nuestro array “blocked” en la posición (11,15), si el valor de dicha posición es “true” entonces la casilla es inaccesible y no debemos dejar al avatar actualizar su posición en esa dirección. De esta forma contamos pues con los siguientes array de atributos para cada mapa y cada colisión con alguno de ellos provocará un efecto u otro:

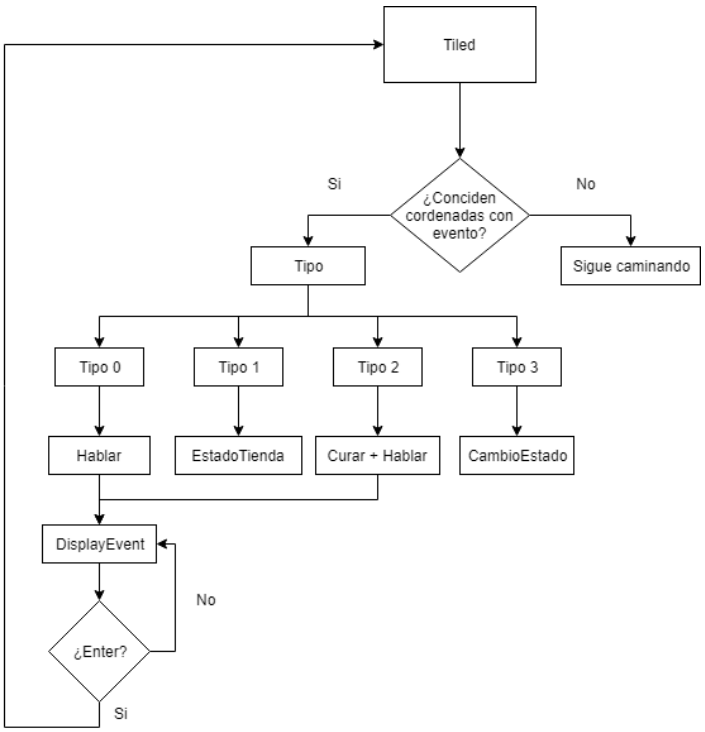
- ***Casilla con atributo“Blocked”:*** Cualquier casilla que posea este atributo será inaccesible por el jugador, es decir se encontrará bloqueada y por tanto la figura del personaje será incapaz de situarse sobre esas casillas. Corresponden a estas, las casillas de objetos grandes, NPCS o paredes sobre las que uno no se puede situar. Al chocar con las casillas que posean este atributo con un valor “false” el valor de la posición del jugador se actualizará en función de la dirección que se esté pulsando en ese momento, determinando su nueva posición en los ejes X e Y. Si por el contrario el atributo posee un valor “true” la actualización de posición no se llevará a cabo y el jugador permanecerá en el sitio contra la casilla bloqueada, dando la sensación de “choque” o colisión contra el objeto.
- ***Casilla con atributo “enemigos”:*** Estas casillas controlan la aparición de combates aleatorios en el mapa. La colisión del jugador con cualquier casilla con este atributo como “true” provocará la generación de un número aleatorio comprendido entre 0 y 1, el valor producido será comparado con un valor de tasa de aparición, y en caso de que entre dentro de unos valores establecidos se saltará de estado de juego hacia un combate con generación de enemigos aleatorio en función del mapa.
- ***Casillas con atributo “evento”:*** La colisión con estas casillas pueden provocar diversos efectos, tantos como distinto eventos existen en el juego:
    - *Eventos de diálogo con NPCS:* La colisión con estos eventos hará que se renderice un pequeño bocadillo junto al NPC con el que se ha chocado, permitiendo leer el diálogo del mismo y proporcionando información sobre el mundo al jugador. Para cerrar dicho diálogo se debe utilizar la tecla “Enter” que solo hace efecto cuando se ha activado uno de estos eventos
    - *Eventos de Tienda:* Estos eventos son, como su nombre indica, el salto hacia el estado de juego Tienda, sobre el cual se cargará un vendedor u otro con una serie de items en función del mapa en el que se encuentre. Como se

explica en el apartado sobre el estado de juego “Tienda” el jugador puede salir manualmente mediante el menú que se le presenta.

- *Eventos de escenas:* Son eventos más especiales, solo se activarán una vez al colisionar con una de las casillas del grupo que las activa. Esto es debido a que este tipo de evento realizó un salto sobre el estado de juego hacia uno de los estados que renderizan las escenas animadas que hacen avanzar la historia. Tras la escena esta se ocupará de redirigir al jugador hacia el siguiente combate o mapa de forma dinámica sin que el jugador deba hacer nada más que disfrutar con el avance de la historia. Este tipo de evento también es utilizado para el cambio de mapa
- *Eventos de Healer:* Este evento tiene el efecto de los eventos de diálogo, pero añade el la propiedad de recuperar la vida y maná de todos los personajes del grupo permitiendo la resurrección de aliados caídos.

Puesto que el detector de colisiones utilizado solo permite captar parámetros booleanos del mapa diseñado en Tiled Maps, se debe realizar la distinción entre los distintos eventos mediante código, esto se realiza mediante coordenadas de mapa.

Cada vez que se detecta la colisión con un “evento”, se capta la coordenada de la casilla que contiene el atributo “eventos=true”y se redirecciona sus coordenadas hacia la clase “EventosNPC” para su búsqueda en una base de datos. Sobre esta clase se ha diseñado algoritmo de control de eventos en función del mapa y sus coordenadas. Las funciones de la clase analizan la posición del evento y aportan a la clase “Heroe” todos los parámetros que necesita para saber de qué tipo son y cómo abordarlos, consiguiendo así que un único atributo booleano permite activar 4 tipos de eventos diferentes.



Carga de nuevos Mapas

El funcionamiento explicado antes, es genérico para todos los mapas y se lleva a cabo desde una única instanciación de la clase “EstadoMapaJuego” para todos los mapas, de forma que, a medida que avanza el juego se cargará un nuevo mapa sobre su atributo TiledMap, basándonos en los atributos de control del juego presentes en la clase Gestión.

Con cada nueva carga de mapa se reiniciará todos los array de atributos para adaptarlos al nuevo mapa, además de ello se cargarán a partir de las bases de datos diseñados nuevos grupos de enemigos de forma coincidente al nivel en el que nos encontramos.

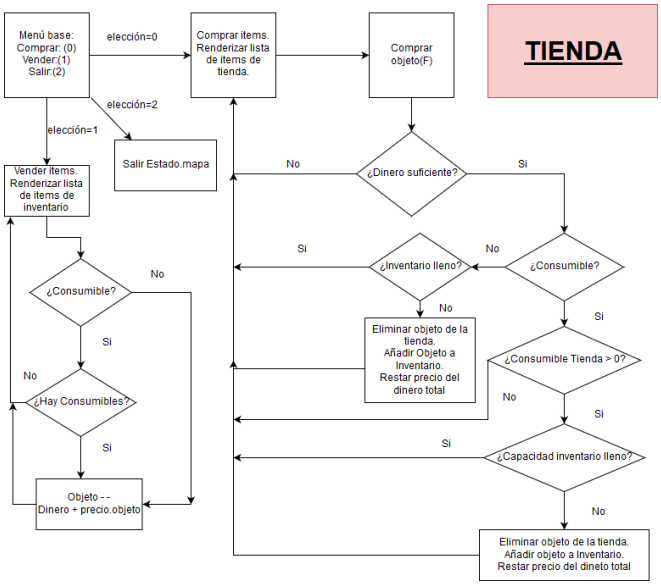
También es interesante mencionar, aunque sea considerado más una parte de los enemigos que del propio mapa, que, puesto que las imágenes no se pueden guardar en la bases de datos, con cada nueva carga de mapa o guardado del juego, se recorre el array de enemigos atribuyéndole a cada uno la imagen que le corresponde.

La tienda

Como en todo buen RPG se necesita una forma de conseguir objetos y negociar con ellos, esto se hace mediante el uso de la clase “EstadoTienda” o “Tienda” para abreviar. Dicha clase hereda sus propiedades de la clase “BasicGameState” por lo que cuenta tanto con un bucle de control como con un bucle de renderizado.

Como se explica en apartado correspondiente este estado se pone en acción cuando el jugador colisiona contra un evento de tipo de “Tienda”, en el cual se genera un objeto de tipo vendedor que posee un listado items sobre el que poder negociar

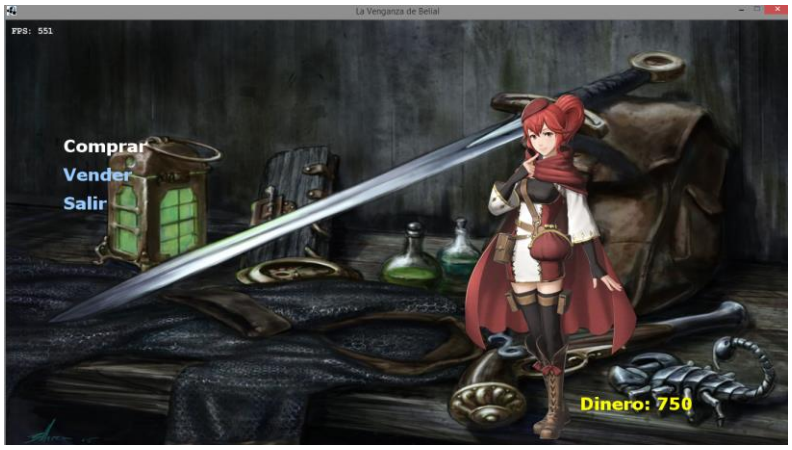
La tienda cuenta con una interfaz sencilla y una mecánica basada en un flujograma de funcionamiento simple



La interfaz gráfica de la tienda es bastante simple y renderiza una serie de menús en función del punto del flujograma en el que se encuentre el jugador, además de la ayudante

del vendedor “Anna” que siempre estará dispuesta a comentar las características de los objetos y decirte el dinero que te queda.

El menú principal en el que el jugador se encontrará al entrar poseerá 3 opciones simples: Comprar, Vender y Salir.



Saltándonos por un instante las dos primeras opciones, “Salir” devolverá al jugador al mapa en el que se encontraba y en la misma posición, es decir saltará del estado “Tienda” al estado “MapaJuego”.

Utilizando la opción “Comprar” el menú cambiará, y en esta ocasión se listarán los objetos que el vendedor posee en ese instante, junto con su coste y la descripción del objeto seleccionado en el instante por el jugador. En caso de que el objeto a comprar sea un consumible también se mostrará la cantidad restante que puede comprar al comerciante antes de quedarse sin existencias. Cuando el jugador utiliza el botón “Enter” para confirmar el objeto el programa comprobará el precio del objeto y lo comparará con la cantidad que posee el inventario del jugador, después comprobará si hay espacio en el inventario o dentro de su apartado de consumibles, una vez realizadas las comprobaciones necesarias, si el objeto puede ser comprado con éxito se eliminará de la lista de items en venta y se añadirá al inventario del jugador, descontando el dinero correspondiente.

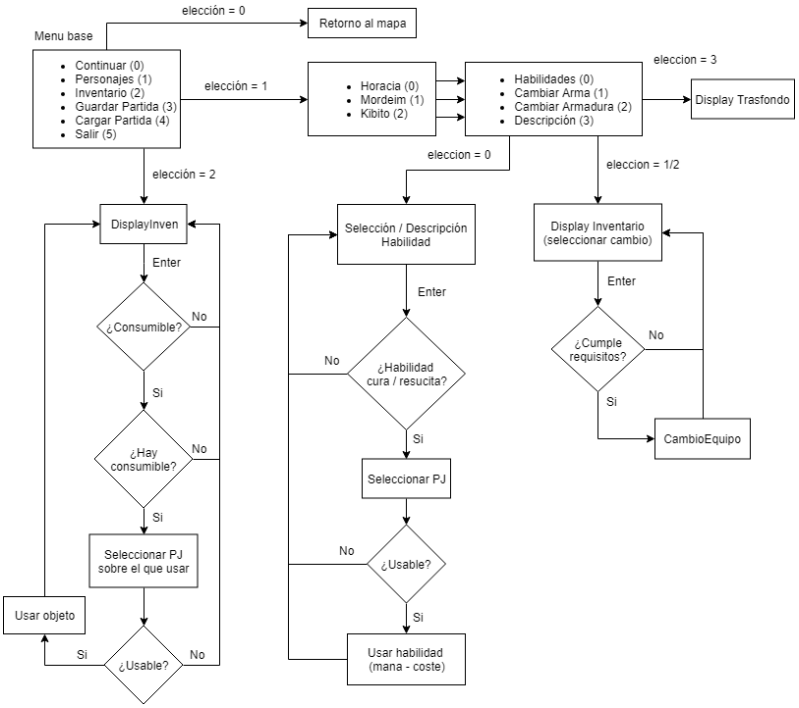
Por otro lado, si utilizamos la opción “Vender” el proceso es el contrario y no requiere de tantas comprobaciones. Se listarán los objetos presentes en el inventario del jugador junto con su precio de venta y la descripción del objeto seleccionado en el momento. Al seleccionar el objeto mediante “Enter” el objeto desaparecerá del inventario y se agregará el dinero de la venta.

Por añadir cierta dificultad al juego el objeto vendido no va a parar al inventario del comerciante, si no que este desaparece para siempre.

Funcionamiento del Menú

Tal como se ha mencionado en apartados anteriores el juego cuenta con diversos menús para uso del jugador, todos ellos desarrollados en base a máquinas de estado y flujogramas.

A pesar de haber sido diseñado como máquina de estados, para que el funcionamiento quede completamente claro, a continuación, se muestra un diagrama de flujo que explica más detalladamente el sistema:



Escenas animadas

Para dar vida al juego y poder ser considerado un RPG basado en en historia, hemos desarrollado diversas escenas animadas a través de las cuales iremos desarrollando la trama y los diálogos entre personajes.

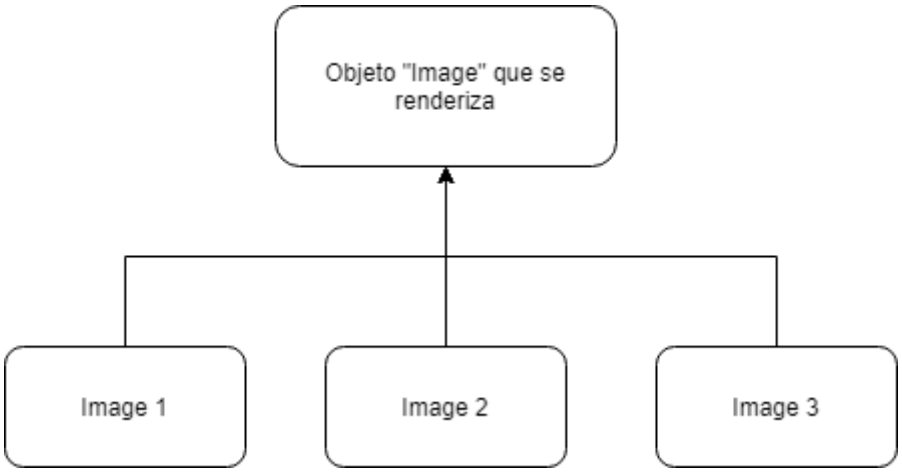
El funcionamiento de todas las escenas de este tipo sigue la estructura de máquina de estados que actualiza la situación de la escena, decidiendo qué se debe renderizar en cada momento. El paso de unos estados a otros se realiza conforme se dan los requisitos de input del jugador, posicionamiento de personajes en la escena o paso del tiempo, siguiendo varios algoritmos simples según corresponda:

```
            case 19://Temporización de animacion de explosiones
                time+=i;
                if(time/1000>3)//3 segundos de ejecución
                {
                    estado++;
                    time=0;
                }
                break;
```

```
            case 21:
                posicion.x+=0.1f*i;
                if(posicion.x>=400){
                    estado++;
                }
                break;
```

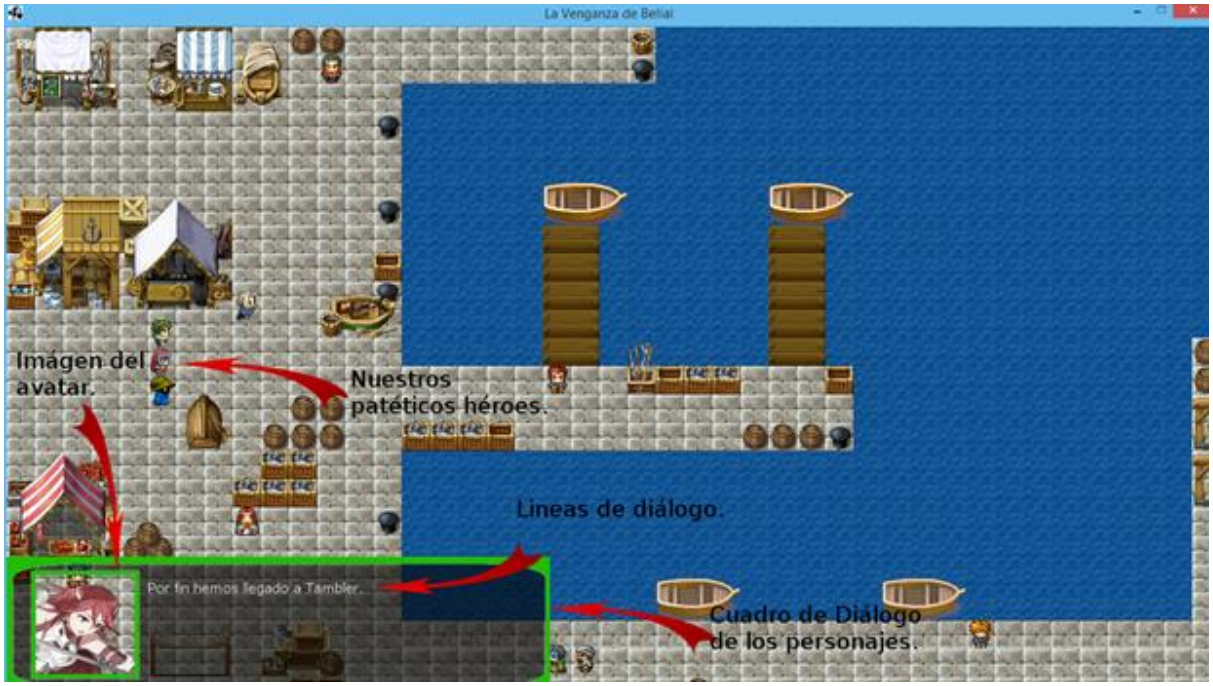
Cada uno de los estados dentro del bucle de control activa los diversos flags que hacen que el bucle de renderizado visualice imágenes o animaciones según corresponda. Es importante mencionar que todo el cuadro de diálogo consta de una imagen de avatar y varias líneas de tipo String, sobre las cuales se van cargando el diálogo y la imagen que se quiere mostrar en cada situación, utilizando así una única sentencia de renderizado para mostrar todos los diferentes diálogos.

```
Control loop:
        case 0:
            avatarDialogo=avatarMordeim;;
                    linea1="Hola soy Mordeim";
                    linea2="y esto es una prueba.";
                   linea3="";
                    linea4="";
             break;
        case 1:
            avatarDialogo=avatarHestia;;
                    linea1="Hola soy la Sacerdotisa Hestia.";
                    linea2="y este es mi dialogo.";
                    linea3="";
                    linea4="";
            break;
Render loop:
        avatarDialogo.draw(x,y,w,h);
        dialogo.draw(x,y,linea1);
        dialogo.draw(x,y,linea2);
        dialogo.draw(x,y,linea3);
        dialogo.draw(x,y,linea4);
```



De forma equivalente, los sprites de los personajes también están basados en este concepto de “renderizado genérico”, cargando sobre un objeto de tipo Animation aquellas animación que queremos que se muestre por pantalla.

Ejemplo de una escena y sus diferentes elementos:



Clases relacionadas con los personajes

Dentro de nuestros sistemas contamos con dos tipos de personajes: jugador y enemigo

Tanto las clases Jugador como Enemigo comparten varios atributos y por tanto se ha creado una clase padre llamada Personaje. Dicha clase está formada por los siguientes atributos, que muestran las características que comparten tanto los aliados como los enemigos:

- private String nombre
- private int nivel
- private int hp
- private int hpActual
- private int ataque: En el caso de jugadores será la suma del atributo “ataqueBase” (que tiene la clase Jugador) más el ataque del arma
- private int defensa: Igual que el anterior pero con atributo “defensaBase” y la defensa que proporciona la armadura
- private int velocidad: Utilizado para generar a la hora de combatir el orden de ataque tanto de enemigos como de jugadores
- private Imagen imagen

Los diferentes métodos que posee la clase son los siguientes:

- Getters and Setters para la utilización de dichos atributos
- public boolean estaVivo(): Usado para saber si un personaje está vivo o no

De la clase Personaje hereda la clase Jugador, que a su vez, es padre de las clases Horacia, Kibito y Mordeim, que son los tipos de personajes que contiene el juego. Cabe destacar que es una clase abstracta debido a que las diferentes clases hijas tendrán unos

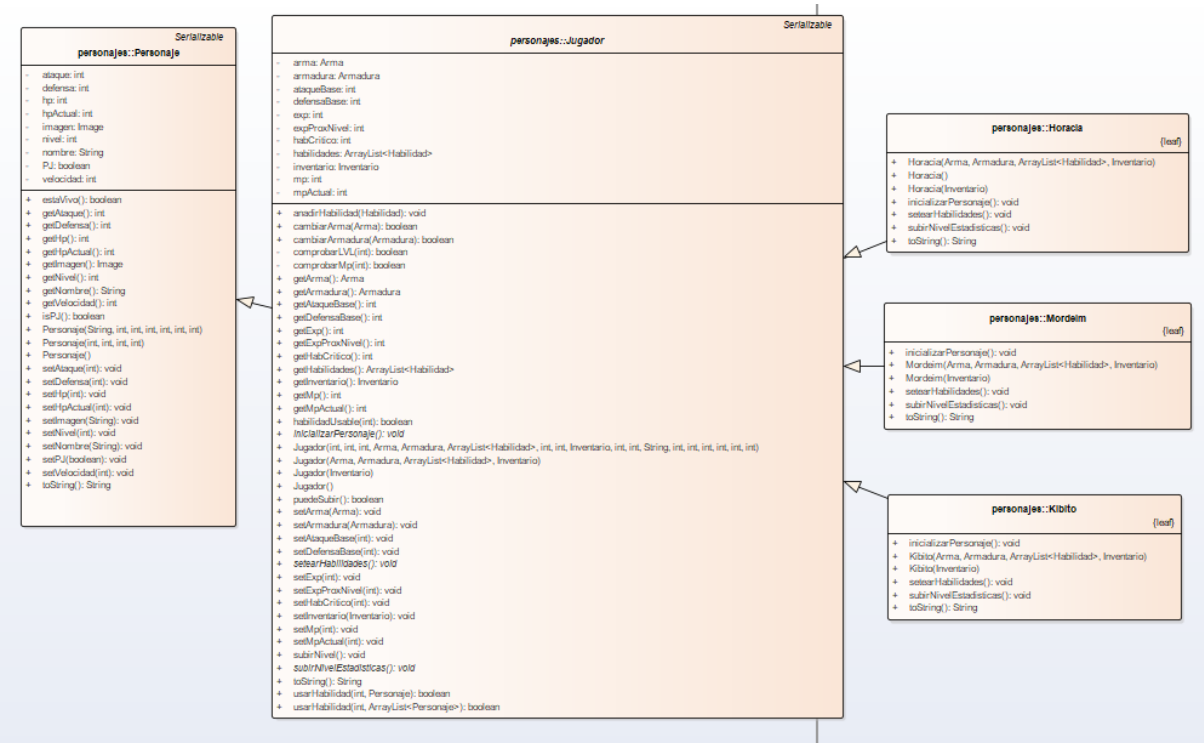
métodos abstractos que cada uno implementará de una manera. Jugador posee los siguientes atributos:

- private int mp
- private int mpActual
- private int habCritico
- private Arma arma
- private Armadura armadura
- private ArrayList<Habilidad> habilidades: Cada jugador tiene 5 habilidades, distribuidas a lo largo de sus niveles
- private int exp
- private int expProxNivel
- private Inventario inventario
- private int defensaBase
- private int ataqueBase

Por su parte los métodos que posee son:

- Getters and Setters usados para manejo de atributos anteriores
- public boolean cambiarArma(Arma arma)
- public boolean cambiarArmadura(Armadura armadura)
- public void anadirHabilidad(Habilidad hab)
- public boolean puedeSubir(): Comprueba si el jugador puede subir de nivel
- public void subirNivel(): Lleva a cabo la subida de nivel
- public boolean habilidadUsable(int nHabilidad)
- private boolean comprobarLVL(int nHabilidad)
- private boolean comprobarMp(int nHabilidad)
- public boolean usarHabilidad(int nHabilidad, Personaje objetivo): Método sobrecargado para usar habilidad contra un enemigo
- public boolean usarHabilidad(int nHabilidad, ArrayList<Personaje> objetivos): Método sobrecargado para usar una habilidad contra varios enemigos
- public abstract void inicializarPersonaje(): Como cada jugador se iniciará de una manera se tiene este método abstracto
- public abstract void subirNivelEstadisticas(): Es abstracto porque cada jugador sube las estadísticas de manera distinta
- public abstract void setearHabilidades()

Las clases hijas de Jugador lo que harán serán implementar los métodos abstractos de la manera que se especifique.



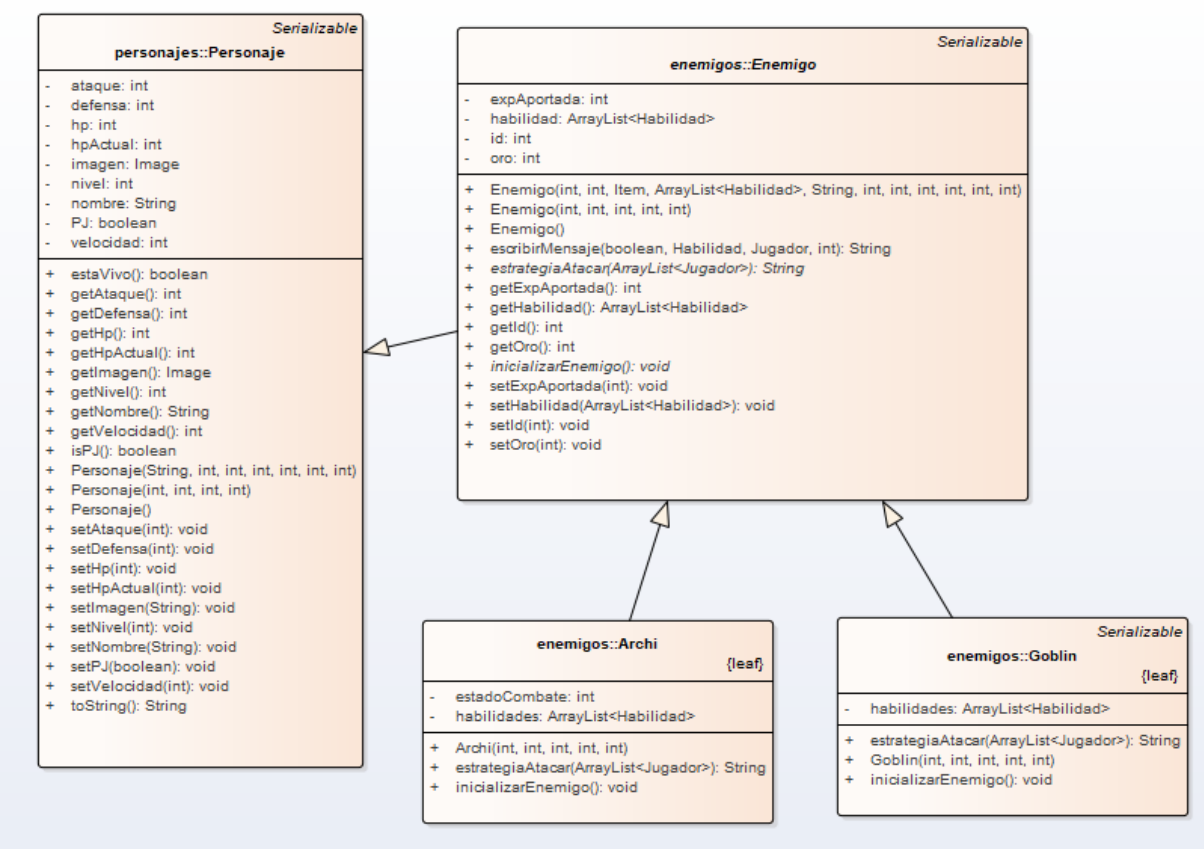
Por último, la clase Enemigos que es hija de Personaje, es a su vez padre de todos los tipos de enemigos como por ejemplo las clases MiniGrifo, Goblin, Rata, etc. También es abstracta porque tienen varios métodos abstractos. Los atributos que posee dicha clase son:

- private int exportada
- private int oro
- private int id
- private ArrayList<Habilidad> habilidades

Por otro lado, sus métodos son:

- Getters and Setters usados para manejo de atributos anteriores
- public String escribirMensaje(boolean habilidad, Habilidad hab, Jugador jugador, int danyo): Usado para mostrar en el combate un mensaje sobre el ataque de enemigo sobre los aliados
- public abstract String estrategiaAtacar(ArrayList<Jugador> jugadores): Usado para generar la estrategia que tiene el enemigo contra los aliados, como por ejemplo, atacar aleatorio, atacar al primero con vida, atacar en área, etc.
- public abstract void inicializarEnemigo(): Sirve para inicializar al tipo de enemigo

Las clases hijas de Enemigo lo que harán serán implementar los métodos abstractos de la manera que se estime oportuno.



Clases relacionadas con Items

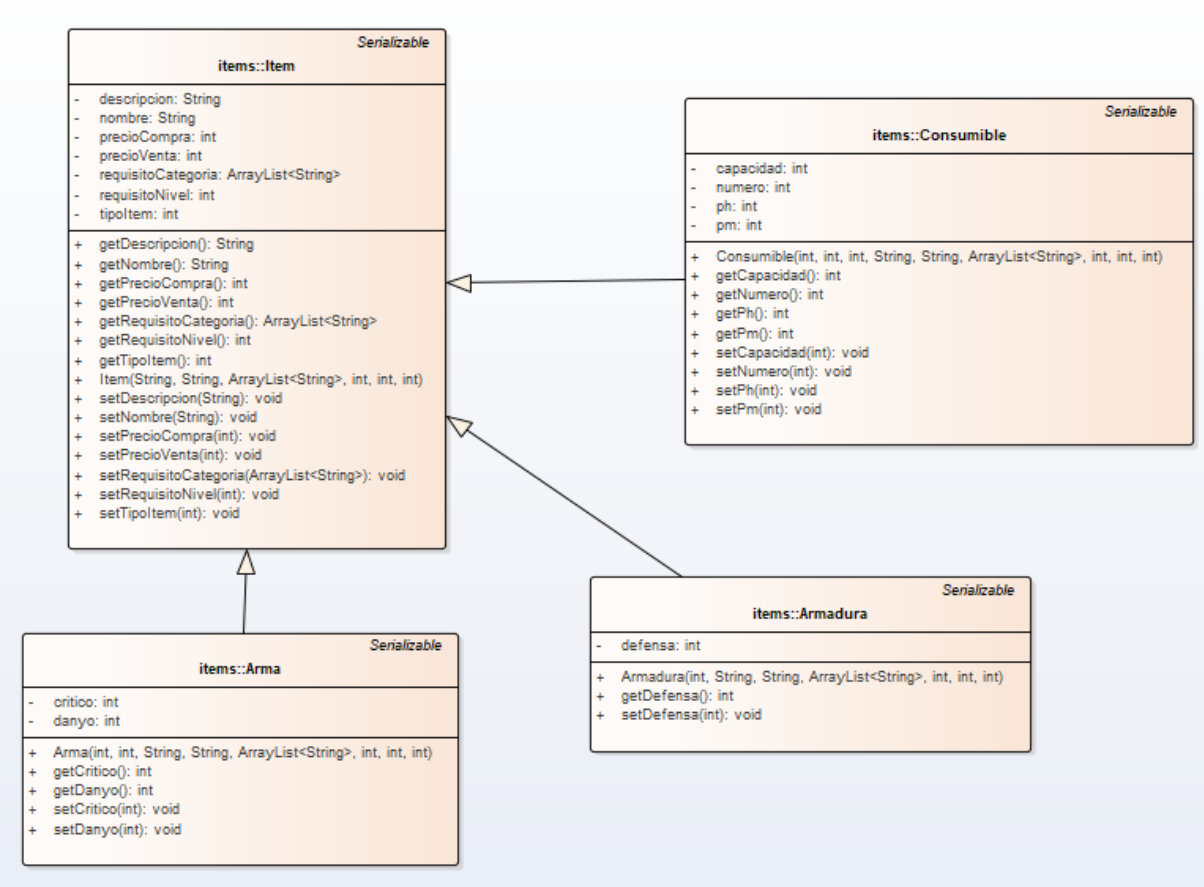
En este apartado se explica la clase padre Item de la cual heredan Armadura, Arma y Consumible, debido a que estas tres clases comparten varias características. La clase Item cuenta con los siguientes atributos:

- private String nombre
- private String description
- private ArrayList<String> requisitoCategoria: Sirve para saber qué personajes pueden usar dicho ítem
- private int requisitoNivel: Utilizado para saber a partir de qué nivel se puede usar dicho item
- private int precioCompra
- private int precioVenta
- private int tipoItem: Usado en los estados del juego para su tratamiento. 0 consumible, 1 arma y 2 armadura

Además, cuenta con los siguientes métodos:

- Getters and Setter para manejo de atributos anteriores (encapsulación)

La diferencia entre las clases hijas son los atributos que añaden: Arma añade danyo y critico, Armadura defensa y Consumible ph, pm, num y capacidad.



Clase Habilidad

Como se ha explicado con anterioridad, los personajes cuentan con diferentes habilidades que realizan diferentes efectos tanto en aliados como en enemigos, para manejar dichos efectos y guardar los atributos de dichas habilidades se ha creado una clase genérica para todas ellas que contiene un código genérico para utilizar todas ellas.

Esta clase contiene los siguientes atributos

- String nombre
- int nivel: Contiene el requisito de nivel para usar dicha habilidad
- float danyo: Este parámetro se utiliza como multiplicador de ataque del personaje cuando la habilidad debe realizar un cálculo de daños
- int costeMP: Indica cuánto MP debe gastar el personaje para usar la habilidad
- String descripción:
- int tipoHabilidad: Este identificador nos permite identificar el tipo de habilidad que es, y de ese modo saber cómo debe ejecutarse y que parámetros se le debe pasar para ello

Además de los correspondiente atributos, esta clase cuenta con una serie de métodos:

- Getters and Setters para el manejo de los atributos de la clase
- public boolean usarHabilidad(Jugador usuario, Personaje objetivo): Método sobrecargado para usar habilidad contra un objetivo

- public boolean usarHabilidad(Jugador usuario, ArrayList<Personaje> objetivos): Método sobrecargado para usar habilidad contra varios enemigos
- private void tipoAtaque(Jugador usuario, Personaje objetivo)
- private void tipoDrenar(Jugador usuario,Personaje objetivo)
- private void tipoAOE(Jugador usuario, ArrayList<Personaje> objetivos)
- private boolean tipoCura(Personaje objetivo)
- private boolean tipoResucitar(Personaje objetivo)
- private void descontarMP(Jugador usuario)

Clase Inventario

Se ha utilizado el patrón de diseño “Composición” para la utilización de la clases Inventario junto a la de Item. Los tres personajes jugables comparten un inventario que puede estar compuesto por todo tipo de items.

Gracias al polimorfismo que nos proporciona la Programación Orientada a Obejtos esta clase posee, un array de Item que le permite guardar en su interior objetos hijos como Consumibles, Armadura y Arma.

Además de ello también cuenta con un atributo “dinero” que, como su nombre indica sirve como indicador del dinero que posee el jugador para su uso en la tienda.

El array de Item cuenta con una capacidad limitada, impidiendo que se puedan tener más de 10 objetos en el inventario. Además, sus 3 primeras posiciones siempre estarán rellenas por objetos de la clase Consumible, para los 3 tipos de pociones del juego.

Para gestionar el uso de este inventario, la clase cuenta con varios métodos como son los correspondientes getter y setters de los atributos, métodos para agregar y quitar objetos del inventario o para el uso de las diferentes pociones utilizando como argumento el jugador objetivo.

Clases Gestion

Es una de las clases más importantes del sistema puesto que se encarga de coordinar varios elementos del juego durante la ejecución de la aplicación. Además, se encarga de guardar algunos ficheros que actúan como base de datos en nuestro sistema y de poder cargar y guardar el avance de nuestra partida.

La clase principal del juego “VenganzaBelial” tiene un atributo que es la instanciación de esta clase.
Está formada por los siguientes atributos:

- private ArrayList<Jugador> jugs: Se guardarán los personajes utilizados por el usuario para su aventura (Horacia, Mordeim y Kibito).

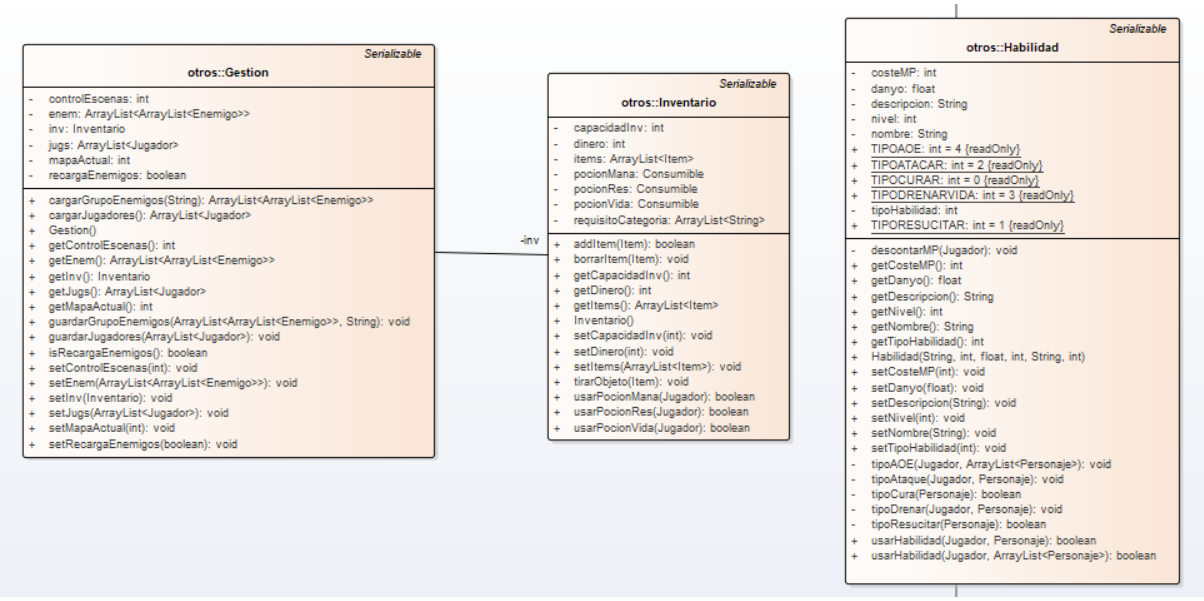
- private ArrayList<ArrayList<Enemigo>> enem: Es un ArrayList formado por un grupo de ArrayList que simularán las partys de enemigos que se pueden encontrar en un mapa en concreto (cambia con el mapa en ejecución) .
- private Inventario inv = new Inventario(): Es el inventario que tendrán los personajes que utiliza el usuario.
- private int mapaActual: Es una variable de control que nos indica en qué mapa nos encontramos, ya que el sistema posee una enumeración de mapas. Es el atributo que nos indicará qué mapa debemos cargar sobre el "EstadoMapaJuego", a que punto debemos regresar tras un combate o cuando debemos desarrollar una batalla contra enemigos específicos.
- private int controlEscena: es una variable de control utilizada en la clase que controla los eventos de forma que las escena animadas se realicen de forma ordenada y no se repitan incluso tras cargar una partida.

Los métodos que posee esta clase son:
- Getters and Setter usados para el manejo de los atributos anteriores
- ArrayList<Jugador> cargarJugadores()
- void guardarJugadores(ArrayList<Jugador> jugadores)
- void guardarGrupoEnemigos(ArrayList<ArrayList<Enemigo>> enemigos, String ruta)
- ArrayList<ArrayList<Enemigo>> cargarGrupoEnemigos(String ruta)

Los métodos de guardar y cargar son utilizados para generar nuestras propias bases de datos internas.

A lo largo de la partida, cuando hagamos referencia a uno de los personajes (Horacia, Kibito o Mordeim) lo haremos a través del atributo jugs. Del mismo modo, si hacemos referencia de alguna party de enemigos lo haremos a través del atributo enem.

A la hora de guardar la partida, lo que haremos es guardar el objeto que hace referencia a esta clase que tiene "VenganzaBelial" para mantener el juego como estaba. Después para cargar, lo que haremos será cargar de un fichero el objeto Gestión que guardamos anteriormente, para restaurar el estado de la partida como se quedó.



Clase relacionadas con NPC

Como Evento y Vendedor comparten varios atributos se ha creado la clase padre NPC que cuenta con los siguientes atributos:

- String nombre
- String saludo

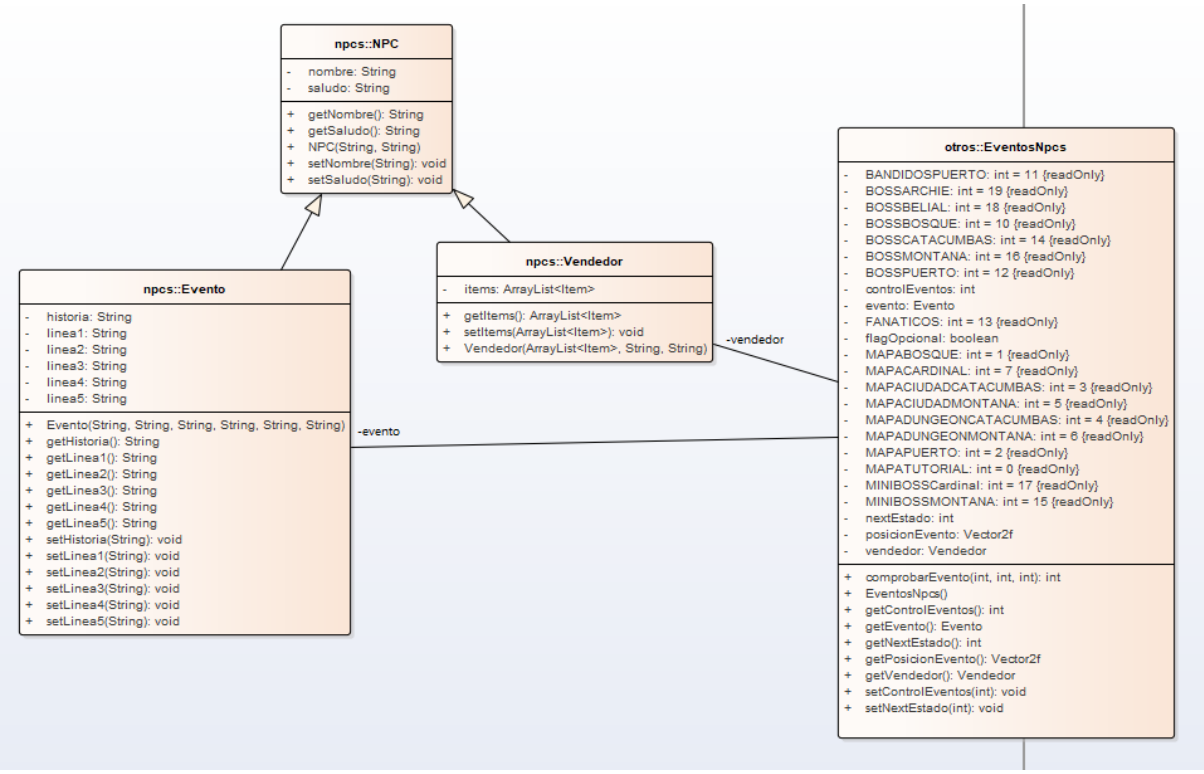
Además de estos atributos, posee los métodos Getters y Setters pertinentes para poder utilizar dichos atributos.

La clase Evento que hereda de NPC contiene el atributo de historia (linea 1-5) donde se guarda la historia que te cuenta un NPC que es de tipo diálogo o tipo healer antes de curarte. Por su parte el Vendedor, tiene como atributo un ArrayList con los objetos que vende.

Por último, cabe destacar el uso de la clase EventosNpcs, que es la encargada de gestionar todos los eventos que tiene el juego. Por ello, cuenta con los atributos:

- int controlEventos: Usado para saber por qué escena se encuentra el jugador
- Evento evento: Necesario para instanciar un evento de diálogo
- int nextEstado: Cambiar al siguiente estado en la máquina de estados
- Vector2f posicionEvento: Localizar en x e y al evento en cuestión
- Vendedor vendedor: Necesario para instanciar un tipo de vendedor
- También se añaden constantes que sirven para hacer un manejo de eventos más amigable

En esta clase destaca el método “comprobarEvento” que sirve para el manejo de cada evento según coordenadas. También posee los típicos Getters and Setters para el manejo de atributos



Clases relacionadas con Combate

El mecanismo de combate de “La venganza de Belial” se basa en el sistema de combate por turnos, en el cual los participantes de la refriega usan de forma estratégica sus bazas para ganar la batalla y eliminar al “equipo” contrario de la forma más eficaz posible.
El combate se lleva a cabo mediante el uso simultáneo de varios objetos interrelacionados entre ellos. Entre dichos objetos se encuentran como no, los personajes del equipo del jugador, los enemigos y el inventario, todos presentes en la clase Gestión explicada en su apartado correspondiente.

Aunque los objetos mencionados son de real importancia para el desarrollo del combate, no son si no los datos de entrada hacia las dos clases principales que desarrollan y organizan todo el control y renderizado del combate de principio a fin. La clase “EstadoCombate”, que hereda de “BasicGameState”, ocupada de renderizar y controlar la máquina de estados y el flujo de la misma, además de renderizar el aspecto gráfico del juego en este modo. La otra clase es “Combate”, la cual se compone de todos los atributos y métodos necesarios para llevar a cabo el funcionamiento interno del combate, es decir, yendo más allá del mero control de la máquina de estados que puede observar el jugador, controlando el flujo de turnos, daños, enemigos muertos, experiencia ganada, resurrecciones y determinar el resultado del combate o del final del turno en el que se encuentra el jugador, es decir, se compone de todo aquello que hace funcionar correctamente la máquina de estados que controla la clase “EstadoCombate”.

Para explicar más detalladamente todos los procesos que componen el sistema de combate y sus componentes a continuación se expone el desarrollo de un combate genérico:

- En primer lugar, el combate solo se activa cuando el juego hace saltar un evento y cambiando al “EstadoCombate”, ya sea haciendo el cambio mediante un evento del mapa o una escena que sirve de prólogo a una batalla contra un “boss”.
  Cuando el bucle de control de “EstadoCombate” se inicia, comprueba si debe o no generar un nuevo combate mediante un flag internos que se resetea siempre que se sale de este estado, por lo que siempre que se entre a este estado se deberá generar uno nuevo.
- Cuando se genera un nuevo combate se crea un objeto de la clase “Combate”, pasándole como argumento los personajes del jugador y el mapa en el que se encuentran. La inicialización de este objeto llama a un método denominado “generaArrayEnemigos” que utilizando como argumentos de entrada el mapa en el que nos encontramos y el nivel de los personajes, ejecuta una búsqueda aleatoria en la base de datos de enemigos cargado en ese momento y los carga sobre un array.
- Una vez tenemos a todos los participantes del combate, una función se ocupa de ordenándolos atendiendo a su velocidad, pero añadiendo un factor de aleatoriedad: la estadística de Velocidad de todos los participantes es multiplicada, de forma individual, por un factor aleatorio diferente comprendido entre 0.5 y 1, esto hace que la experiencia sea más entretenida pues el mismo enemigo puede llegar a atacar antes o después que cualquiera de tus personajes en diferentes combates.
- Una vez establecido el orden todos los participantes serán cargados en un único Array de Personajes denominado “ordenPersonajes”. Esto es posible debido al polimorfismo debido a que las clases Jugador, Horacia, Mordeim, Kibito, Enemigo y sus sucesivos heredan de esta clase padre (Personaje). Una vez determinado el orden a seguir a lo largo del combate, la clase “EstadoCombate” se ocupará de renderizar en todo momento a los enemigos que queden vivos, así como los avatares de los personajes del jugador, variando el fondo de los mismos en función de la situación en la que se encuentren. Además, cuenta con varios métodos de renderización que muestran en pantalla mensajes, además de un fondo en función del mapa en el que se encuentre y menús de opciones diferentes dependiendo del estado en el que se encuentre el combate dentro de la máquina de estados de combate. Si por ejemplo el personaje se encuentra muerto, su fondo aparecerá en rojo, si es el turno del personaje aparecerá en verde y si no es ninguno de los anteriores su fondo será azul, mostrando siempre la barra de vida y de mana del personaje.
- Para realizar el combate de forma ordenada y renderizar lo que se debe en cada momento se cuenta en con dos métodos por cada uno de los estados diseñados, uno de control y otro de renderizado. Primero se comprueba si el turno que toca pertenece al jugador o a un enemigo y en función de eso la máquina de estado salta a uno u otro estado:
  - Si el turno fuera de un enemigo se saltaría al estado número 5 o “TurnoEnemigo”, donde se ejecutaría una función abstracta de la clase “Enemigo”, “estrategiaAtacar”. Cuando esta función se ejecuta se desarrollará en función del enemigo al que le toque una seríe de acciones o ataques y saltará al estado número 8 o “FinTurno”, donde la información de lo ocurrido se mostrará en pantalla informando al jugador. Para finalizar el turno definitivamente se utilizará tecla “Enter”, provocando la llamada a una serie de métodos de la clase “Combate” que comprobarán que participantes

han muerto y los removerán de la lista de turno. Luego comprobarán si el combate ha acabado y, si no es el caso, determinarán en función de la lista de turnos de quien es el siguiente turno

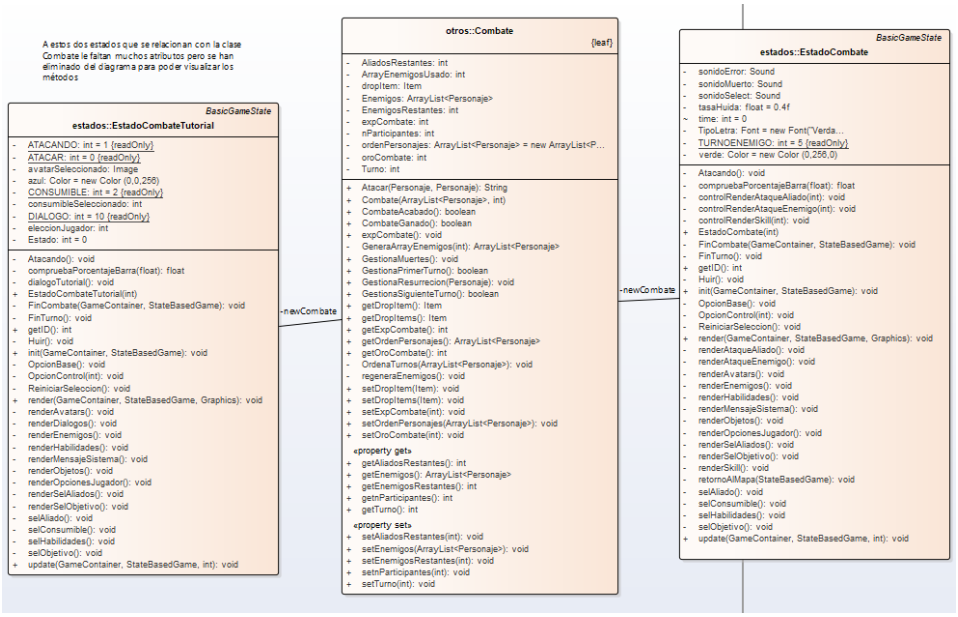
- Si el turno es de un aliado se saltará al estado número 0 o “OpcionesBase”, donde se mostrarán las opciones del personaje entre las que se encuentran: Atacar, Habilidades, Inventario y Huir.
  Mediante el uso de un método genérico para todos los menús denominado “OpcionControl” que utiliza como argumento el número de opciones en cada momento, el jugador podrá disminuir e incrementar un índice que permitirá la selección de otra opción. Para que la experiencia de jugabilidad sea mejor el método de renderizado de cada estado permitirá observar visualmente que opción se está eligiendo si estar pendiente de número del índice. Al presionar “Enter” sobre cualquiera de las opciones se realizará un cambio de estado hacia el correspondiente:
  - Si presionamos la opción “Atacar”, saltaremos al estado número 1 o “Atacanado” y se mostrará en pantalla el nombre del enemigo al que estamos apuntando. Luego por medio del mismo método “OpcionControl” se nos permitirá desplazarnos por el array de enemigos decidiendo a cuál de ellos deseamos atacar. Cuando la decisión esté tomada por medio del botón “Enter” se ejecutará el método de cálculo de daños presente en la clase “Combate”, tras lo cual se remitirá hacia el estado “FinTurno” que ejecutará los mismos procesos explicados anteriormente
  - Si presionamos la opción “Habilidades”, se saltará al estado número 2, o “Seleccionar Habilidades”, donde se renderiza la habilidad del personaje en cuestión. Sobre este menú podremos desplazarnos sobre el array de habilidades del personaje para seleccionar aquella habilidad que deseemos. Cuando seleccionamos cualquier habilidad se realiza una comprobación de requisitos atendiendo al nivel y coste MP de la habilidad, si ambas cumplen satisfactoriamente se podrá usar la habilidad entendiendo que existen 3 tipos principales y que cada una llamará a un estado diferente. Si utilizamos una habilidad multiobjetivo esta se ejecutará automáticamente realizando el cálculo de daños correspondiente y remitiendo directamente al estado “FinTurno”. Si utilizas una habilidad de ataque uniobjetivo se remitirá al estado nº6 o “SelObjetivos”, donde se seguirá un proceso igual al explicado en el estado “Atacando”. Si utilizamos una habilidad de apoyo, es decir que va dirigida contra un aliado, el estado saltará al número 7 o “Seleccionad Aliado”. Este estado cumple con un renderizado y menú, similares a los del estado “Atacando” pero en este caso no nos desplazamos por el array de enemigos si no por el de aliados, permitiéndonos seleccionar a uno de los mismos. Cuando el objetivo es seleccionado, se realizan las comprobaciones pertinentes a la habilidad seleccionada para saber si realmente se puede usar esa habilidad sobre el objetivo determinado. Si se puede utilizar la habilidad, ejecutará sus funciones y restará el coste de MP sobre el personaje que la ha ejecutado y, en caso de haber resucitado a alguien se utilizará un método de gestión resurrecciones presente

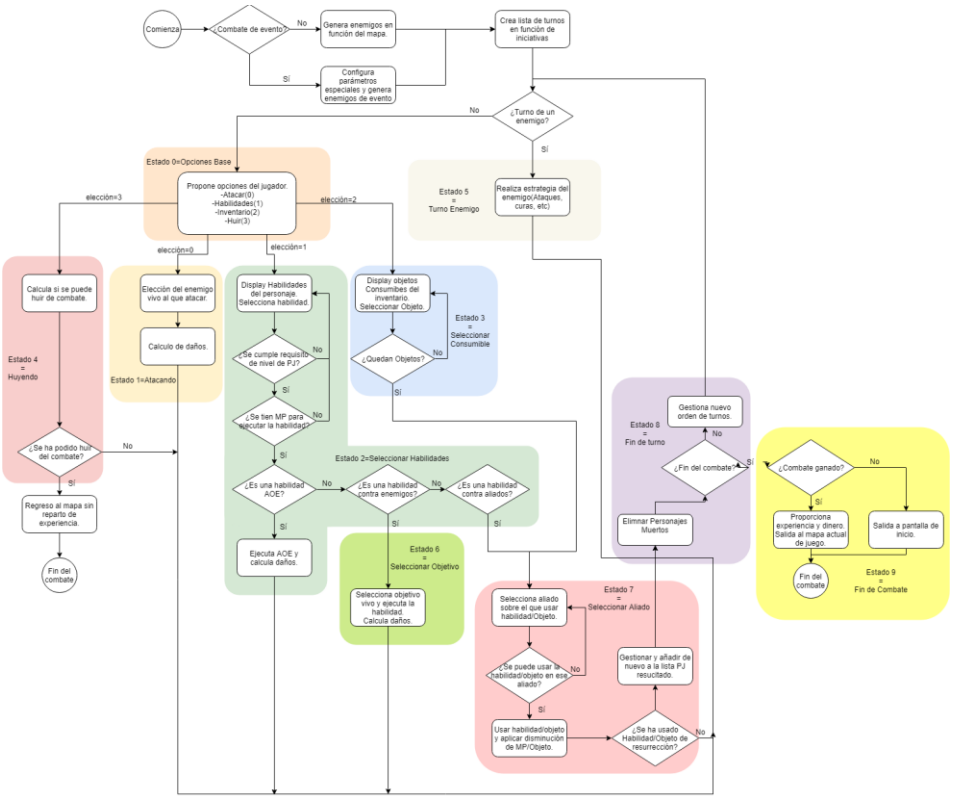
en la clase “Combate” que añadirá al personaje resucitado al final del array de turnos, tras ello se remitirá al estado “FinTurno”.
Es importante mencionar que el estado “Seleccionar Aliado” es genérica para el uso de habilidades y objetos por lo que para poder ejecutarse de una forma u otra según la situación, realiza una comprobación sobre cuál fue el último estado antes de él.

- Si se usa la opción “Inventario” se saltará al estado número 3 o “Seleccionar Consumible”. En este estado se mostrarán los diferentes objetos consumibles presentes en el inventario, así como su cantidad, es decir nos desplazamos a modo de menú por las 3 primeras posiciones del array de objetos del inventario para seleccionar cual queremos utilizar. Al seleccionar cualquiera de ellos se comprobará que quedan objetos de ese tipo en el inventario antes de saltar al estado “Seleccionar Aliado” donde se seguirá el mismo proceso explicado antes, pero aplicado a objetos.
- Si se utiliza la opción “Huir” se saltará al estado número 4 o “Huyendo”, donde se generará un número aleatorio. Si dicho aleatorio está comprendido dentro de la tasa de Huida establecida para el combate, se activará un flag especial, en caso contrario el flag permanecerá apagado. Si el flag permanece apagado se cambiará al estado “FinTurno” donde se ejecutarán los procesos pertinentes y se cambiará al turno del siguiente participante. En caso de que el flag se active se cambiará al estado número 9 o “Fin Combate” donde se gestionará la finalización del combate y el regreso al mapa de juego o hacia la escena que corresponda en función del identificador de mapa con el que se ha llevado a cabo el combate.

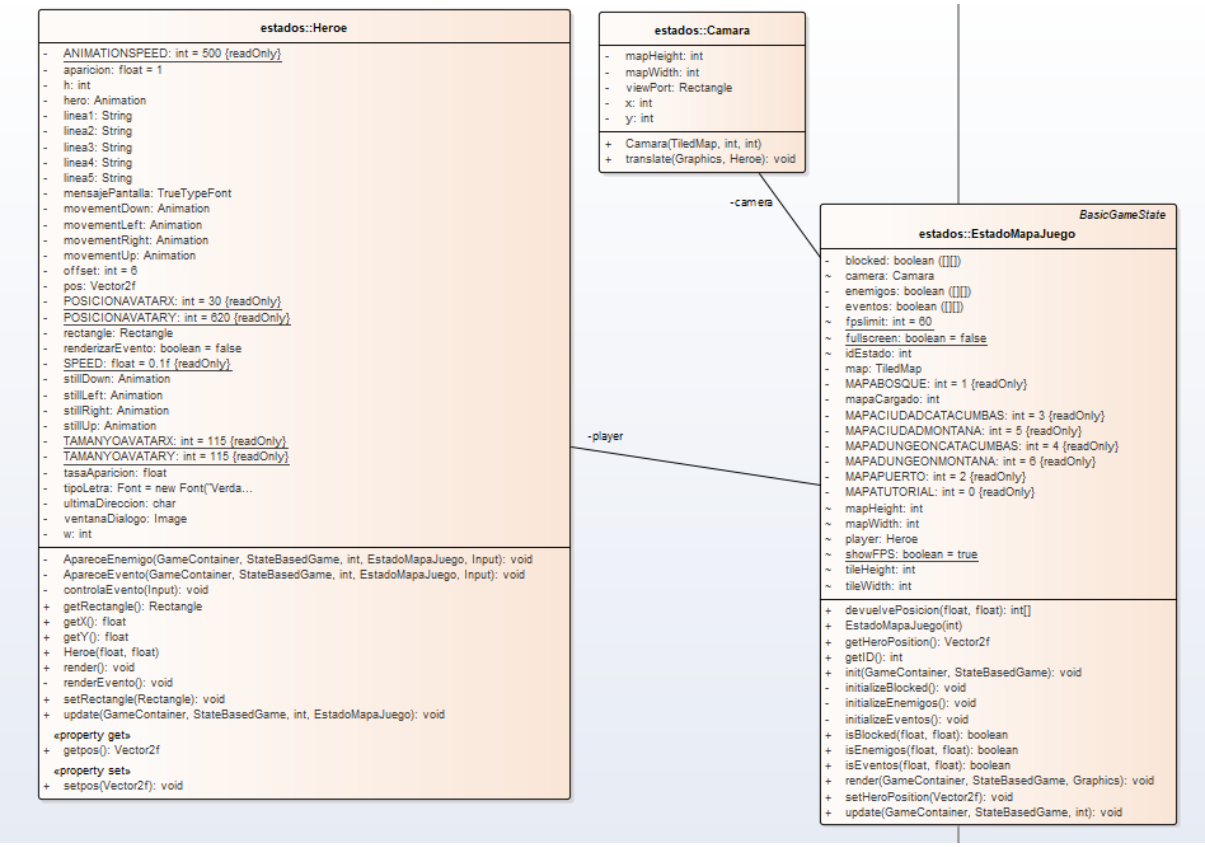
- A la hora de llamar a la finalización de combate siempre se reiniciará el flag que permitirá la creación de un nuevo objeto “Combate” cuando se vuelva a cambiar al estado de juego “EstadoCombate”, además de ello se valorará la forma en la que se ha acabado el combate dando lugar a tres opciones:
  - El combate se ha acabado, no quedan enemigos vivos y por tanto el jugador ha ganado, obteniendo como premio oro y experiencia
  - El combate ha finalizado pero no hay aliado vivos, por tanto el jugador ha perdido y debe comenzar desde el último punto guardado o desde el inicio
  - El jugador ha activado el flag de huida, el jugador no recibe premios puesto que no ha ganado el combate, pero puede continuar su aventura

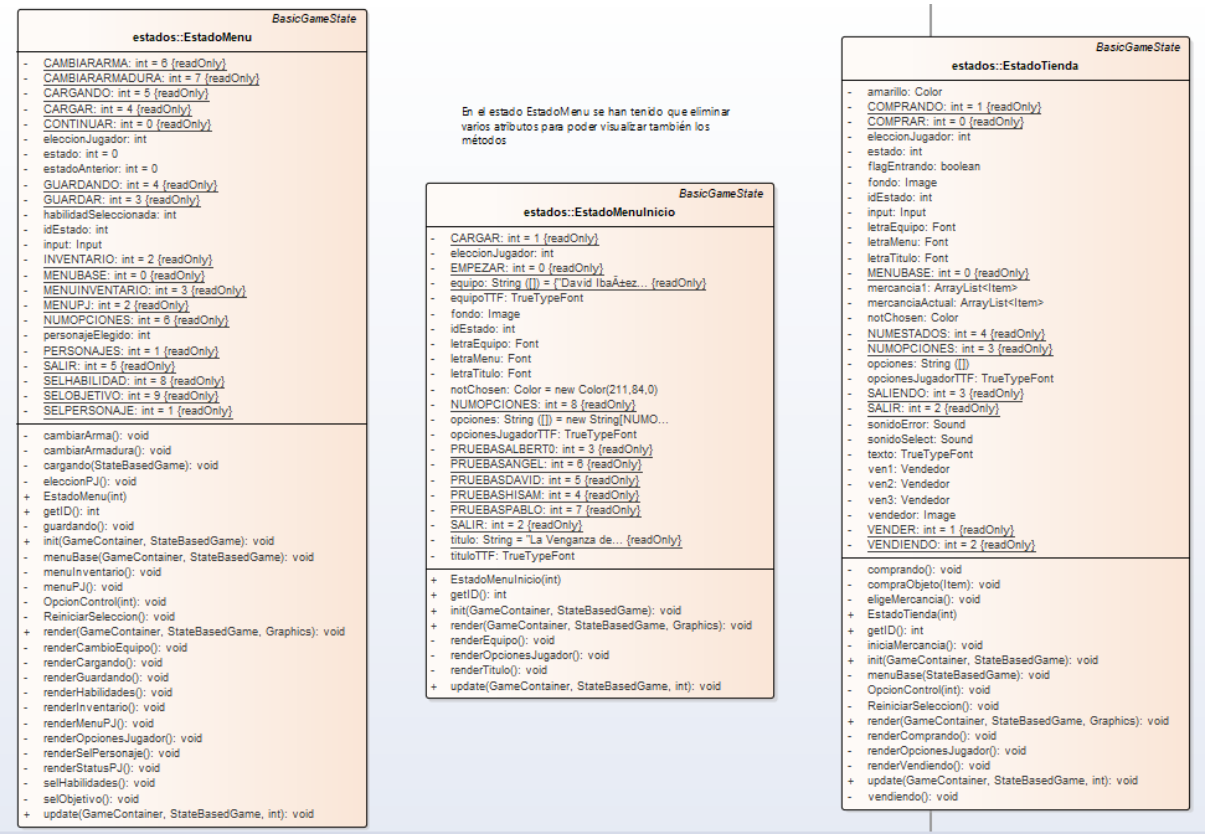
A modo de ayuda para entender toda esta explicación se muestran a continuación el diagrama de clases de dichas clases y un diagrama de estados explicativo


A estos dos estados que se relacionan con la clase Combate le faltan muchos atributos pero se han eliminado del diagrama para poder visualizar los métodos
BasicGameState
estados::EstadoCombateTutorial
- ATACANDO: int = 1 {readOnly}
- ATACAR: int = 0 {readOnly}
- avatarSeleccionado: Image
- azul: Color = new Color (0,0,256)
- CONSUMIBLE: int = 2 {readOnly}
- consumibleSeleccionado: int
- DIALOGO: int = 10 {readOnly}
- eleccionJugador: int
- Estado: int = 0
- Atacando(): void
- compruebaPorcentajeBarra(float): float
- dialogoTutorial(): void
+ EstadoCombateTutorial(int)
- FinCombate(GameContainer, StateBasedGame): void
- FinTurno(): void
+ getID(): int
- Huir(): void
+ init(GameContainer, StateBasedGame): void
- OpcionBase(): void
- OpcionControl(int): void
- ReiniciarSeleccion(): void
+ render(GameContainer, StateBasedGame, Graphics): void
- renderAvatars(): void
- renderDialogos(): void
- renderEnemigos(): void
- renderHabilidades(): void
- renderMensajeSistema(): void
- renderObjetos(): void
- renderOpcionesJugador(): void
- renderSelAliados(): void
- renderSelObjetivo(): void
- selAliado(): void
- selConsumible(): void
- selHabilidades(): void
- selObjetivo(): void
+ update(GameContainer, StateBasedGame, int): void
-newCombate
otros::Combate
{leaf}
- AliadosRestantes: int
- ArrayEnemigosUsado: int
- dropItem: Item
- Enemigos: ArrayList<Personaje>
- EnemigosRestantes: int
- expCombate: int
- nParticipantes: int
- ordenPersonajes: ArrayList<Personaje> = new ArrayList<P...
- oroCombate: int
- Turno: int
+ Atacar(Personaje, Personaje): String
+ Combate(ArrayList<Personaje>, int)
+ CombateAcabado(): boolean
+ CombateGanado(): boolean
+ expCombate(): void
- GeneraArrayEnemigos(int): ArrayList<Personaje>
+ GestionaMuertes(): void
+ GestionaPrimerTurno(): boolean
+ GestionaResurrecion(Personaje): void
+ GestionaSiguienteTurno(): boolean
+ getDropItem(): Item
+ getDropItems(): Item
+ getExpCombate(): int
+ getOrdenPersonajes(): ArrayList<Personaje>
+ getOroCombate(): int
- OrdenaTurnos(ArrayList<Personaje>): void
- regeneraEnemigos(): void
+ setDropItem(Item): void
+ setDropItems(Item): void
+ setExpCombate(int): void
+ setOrdenPersonajes(ArrayList<Personaje>): void
+ setOroCombate(int): void
«property get»
+ getAliadosRestantes(): int
+ getEnemigos(): ArrayList<Personaje>
+ getEnemigosRestantes(): int
+ getnParticipantes(): int
+ getTurno(): int
«property set»
+ setAliadosRestantes(int): void
+ setEnemigos(ArrayList<Personaje>): void
+ setEnemigosRestantes(int): void
+ setnParticipantes(int): void
+ setTurno(int): void
-newCombate
BasicGameState
estados::EstadoCombate
- sonidoError: Sound
- sonidoMuerto: Sound
- sonidoSelect: Sound
- tasaHuida: float = 0.4f
~ time: int = 0
- TipoLetra: Font = new Font("Verda...
- TURNOENEMIGO: int = 5 {readOnly}
- verde: Color = new Color (0,256,0)
- Atacando(): void
- compruebaPorcentajeBarra(float): float
- controlRenderAtaqueAliado(int): void
- controlRenderAtaqueEnemigo(int): void
- controlRenderSkill(int): void
+ EstadoCombate(int)
- FinCombate(GameContainer, StateBasedGame): void
- FinTurno(): void
+ getID(): int
- Huir(): void
+ init(GameContainer, StateBasedGame): void
- OpcionBase(): void
- OpcionControl(int): void
- ReiniciarSeleccion(): void
+ render(GameContainer, StateBasedGame, Graphics): void
- renderAtaqueAliado(): void
- renderAtaqueEnemigo(): void
- renderAvatars(): void
- renderEnemigos(): void
- renderHabilidades(): void
- renderMensajeSistema(): void
- renderObjetos(): void
- renderOpcionesJugador(): void
- renderSelAliados(): void
- renderSelObjetivo(): void
- renderSkill(): void
- retornoAlMapa(StateBasedGame): void
- selAliado(): void
- selConsumible(): void
- selHabilidades(): void
- selObjetivo(): void
+ update(GameContainer, StateBasedGame, int): void


Comienza
Combate de evento?
No
Sí
Genera enemigos en función del mapa.
Configura parámetros especiales y genera enemigos de evento
Crea lista de turnos en función de iniciativas
¿Turno de un enemigo?
No
Sí
Estado 0=Opciones Base
Propone opciones del jugador.
-Atacar(0)
-Habilidades(1)
-Inventario(2)
-Huir(3)
elección=3
elección=2
elección=0
elección=1
Estado 5
=
Turno Enemigo
Realiza estrategia del enemigo(Ataques, curas, etc)
Calcula si se puede huir de combate.
Estado 4
=
Huyendo
¿Se ha podido huir del combate?
No
Sí
Regreso al mapa sin reparto de experiencia.
Fin del combate
Elección del enemigo vivo al que atacar.
Calculo de daños.
Estado 1=Atacando
Display Habilidades del personaje. Selecciona habilidad.
¿Se cumple requisito de nivel de PJ?
No
Sí
¿Se tien MP para ejecutar la habilidad?
No
Sí
Estado 2=Seleccionar Habilidades
¿Es una habilidad AOE?
No
¿Es una habilidad contra enemigos?
No
¿Es una habilidad contra aliados?
Sí
Ejecuta AOE y calcula daños.
Display objetos Consumibles del inventario. Seleccionar Objeto.
Estado 3
=
Seleccionar Consumible
¿Quedan Objetos?
No
Sí
Estado 6
=
Seleccionar Objetivo
Selecciona objetivo vivo y ejecuta la habilidad. Calcula daños.
Estado 7
=
Seleccionar Aliado
Selecciona aliado sobre el que usar habilidad/Objeto.
¿Se puede usar la habilidad/objeto en ese aliado?
No
Sí
Usar habilidad/objeto y aplicar disminución de MP/Objeto.
¿Se ha usado Habilidad/Objeto de resurrección?
No
Gestionar y añadir de nuevo a la lista PJ resucitado.
Estado 8
=
Fin de turno
Gestiona nuevo orden de turnos.
¿Fin del combate?
No
Sí
Eliminar Personajes Muertos
¿Combate ganado?
No
Sí
Proporciona experiencia y dinero. Salida al mapa actual de juego.
Salida a pantalla de inicio.
Fin del combate
Estado 9
=
Fin de Combate

Otras clases importantes del juego





Referencias de otros juegos y libros

Referencias de otros juegos:

En este apartado vamos a explicar los distintos tipos de videojuegos de los que hemos sacado la idea de narrativa, mapeado y la personalidad de los personajes no jugables.

Narrativa:

Para hacer la parte narrativa del juego nos hemos basado en juegos de tipo medieval con fantasía como “Dragon Quest”, “Final Fantasy”, “The Witcher” y “Sacred”.

- ***“Dragon Quest”:*** Idea de monstruos clásicos como el Slime, los Minotauros y los Dragones
- ***“Final Fantasy”:*** Nos hemos basado concretamente en el “Final Fantasy IV” con una fuerte historia oculta hasta el final para el jugador haciendo que le entren ganas de seguir jugando para descubrir la verdad
- ***“The Witcher”:*** Hemos cogido easter eggs como el easter egg de Game of Thrones con Tyrion Lannister en el mapa Catacumbas, o el de algunos guiños nuestros que surgían en las reuniones para ideas que no se pudieron añadir, aunque otras sí como el caso del enemigo del mapa Montaña: Murciégalo
- ***“Sacred”:*** Nos hemos basado en algunas armas y armaduras dentro del juego como un tributo a esta gran saga de juegos

Mapeado:

Para los mapas se han cogido ideas de otros mapas para hacer este juego.

Para el “Bosque de Tamberl” nos hemos basado en los bosques de Brecilian Forest del juego “Dragon Age Origins”, un mapa donde es fácil perderse y a la vez fomenta la exploración de dicho mapa.

Del mapa “Puerto de Tambler” hemos cogido la idea de Ciudad Carmín de la región de Kanto de “Pokémon”, con la gran reconocida referencia del mítico barco, el “SSAnne”.

Para el mapa “Catacumbas” nos hemos basado en el exterior de las Catacumbas de “Dark Souls”, un lugar lleno de esqueletos que intentarán acabar con nuestra vida. Para los sectarios hemos cogido la idea de la religión por la parte más fanática.

Del mapa “Montaña” hemos cogido la idea de la mitología griega del “God of War”, donde aparecen enemigos de este Hack and Slash tan popular, además de enemigos griegos como Belerofonte, el domador del caballo alado Pegaso.

Para el mapa “Cardinal” cogimos la idea del Castillo de N en “Pokémon” Blanco y Negro, siendo uno de los lugares más míticos de este juego que marco un tono más serio a esta saga de juegos.

NPC’s y su personalidad:

Los NPC del tipo vendedor los hemos sacado del épico “Fire Emblem”, más en concreto Anna, la cual nos vende los consumibles y el nuevo equipo continuar con nuestra aventura.

El curador o Healer, es el mítico personaje que cura a todo el equipo con sus sobrenaturales poderes, un clásico en el mundo de los videojuegos.

Algunos NPCs fomenta al jugador la exploración y el hablar con todos los NPC para tener información del lugar o simplemente para reírse con las frases que sueltan. Para estos NPC nos hemos basado en que fueran sus diálogos lo más humano posible dándole ese tono humorístico o serio en algunas situaciones.

Arte y música utilizada

A continuación, se especifica de donde se han obtenido los diferentes aspectos del arte y sonido, teniendo en cuenta que casi en su totalidad no han sido desarrollados por el equipo, se especifican aquellos que no pertenecen al equipo y han sido captados de Internet u otros medios de forma gratuita.

Las imágenes o cualquier otro material sujeto a derechos de autor se usan de forma gratuita, de haber algún inconveniente con los propietarios, serán retiradas y modificadas. Este juego ha sido desarrollado sin ánimo de lucro, únicamente como trabajo académico.

Arte:

- *Diseño de mapas:* Propio, diseñado mediante Tiled Maps
- *Diseños de Mordeim:* Obtenido de “Sothe”, personaje de “Fire Emblem: Radiant Dawn” diseño de “Fire Emblem Heroes”
- *Diseños de Horacia:* Ha sido obtenido de “Cordelia”, personaje de “Fire Emblem: Awakening” diseño de “Fire Emblem Heroes”
- *Diseños de Kibito:* Obtenidos de “Vivi” o “Mago Negro” de la saga de juegos de “Final Fantasy”
- *Diseño de Hestia:* Se ha sacado de “Ren Kougyoku”, personaje del manga “Magi: the labyrinth of magic”
- *Ayudante de la tienda:* Obtenido de “Anna” de la saga “Fire Emblem”
- *Diseños de sprites y tilesets casi en su totalidad:* Diseñados mediante RPG Maker u obtenidos de Google imágenes
- *Fondos de batalla:* Obtenidos de RPG Maker o Google imágenes
- *Fondo del menú:* Sacado de Google imágenes mediante la búsqueda “rune background”
- Otros diseños artísticos: obtenidos de forma gratuita de Internet, como personajes de los juegos mentados en las referencias u otros pero que de igual manera no son propiedad del equipo.

Música:

- *Música de batallas:* “Shout your soul” de la banda sonora original de “Tales of Berseria”.
- *Efectos de sonido en su totalidad:* Provenientes de Youtube, RPG Maker e internet, y convertidos a formato .wav para hacerlos funcionar en la aplicación
- *Música sacada de forma gratuita y sin ánimo de lucro:* Sacada de videojuegos como Pokémon, God of War, The Witcher, Metal Gear Rising Revengeange, Final Fantasy, Dark Souls, Skyrim y Mario y Luigi Bowser’s Inside y de animes como Katekyo Hitman Reborn y One Punch Man.

Conclusiones y mejoras

Como resultado del trabajo hemos obtenido un juego entretenido, con mecánicas simples que presentan facilidad para su aprendizaje por parte del jugador y una historia lo suficientemente entretenida como hacer que los jugadores se mantengan frente a la pantalla solo para ver el desenlace, o al menos eso es lo que se espera.

De cara a futuros proyectos o de haber tenido más de tiempo para este nos hubiera gustado mejorar tres aspectos fundamentales que, debido a diversos motivos no hemos podido llegar a explotar:

- *Animaciones de combate:* Posiblemente uno de nuestros mayores arrepentimientos es que durante los combates apenas hemos podido añadir animaciones específicas

que escenifiquen lo que ocurre, más allá de 3 tipos de animaciones genéricas y una explicación en texto con el resumen de lo ocurrido cada turno

- *Habilidades más variadas:* Si bien es cierto que hemos cumplido con las mecánicas normales que tiene un RPG de este estilo, nos hubiera gustado añadir más variedad en las habilidades, como por ejemplo que nuestro “tanque” pudiera interponerse entre los enemigos y los aliados o añadir curas en área. Por desgracia la duración del juego, el intento por hacer el código lo más genérico posible y la falta de tiempo al final se han vuelto un impedimento para ello
- *Detección de colisiones más refinada:* La detección de colisiones actual funciona como corresponde, pero a veces al mecanismo de colisiones le cuesta más identificar la colisión de tipo evento, por ello una de las mejoras sería la de refinar el mecanismo de detección de colisiones.

De cara a la memoria realizada queremos mencionar que hemos intentado en todo momento explicar todo el trabajo que hemos realizado y tiempo dedicado, no obstante, al mismo tiempo hemos intentado que no se haga especialmente pesado por lo que hemos omitido la resolución de algunos problemas encontrados durante el desarrollo del proyecto centrándonos en aquellos que hemos considerado de mayor importancia para la asignatura, esperando no habernos quedado cortos.

En caso de que alguno de los diagramas no pueda ser apreciado en la memoria, todos ellos se encuentran en la carpeta “Desarrollo de Memoria” dentro del repositorio de trabajo del gupo.