UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA

TESIS

DESARROLLO DE VIDEOJUEGOS PARA DISPOSITIVOS PERSONALES MÓVILES (PDA)

QUE PRESENTAN
SILVIA DISNARDA RUIZ BARRERA
MANUEL TÉLLEZ GIRÓN GODÍNEZ

PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN

ASESOR
ING. SANTIAGO IGOR VALIENTE GÓMEZ

MÉXICO, D. F., 2006

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Dedicatorias

Quiero dedicarle esta tesis a mi madre ya que sin sus constantes presiones, quejas, apoyo y patrocinio me hubiera tardado más en terminarla, a mi hermano David por sus comentarios que me recuerdan mi lado humano y finalmente a Silvia por que sin ella la carrera me hubiera parecido insoportable.

Manuel Téllez Girón Godínez

Mamá, gracias por todo, esta tesis va para ti con cariño. Manuel, no necesito dedicártela, porque estamos juntos en esta y otras aventuras más…

Silvia Disnarda Ruiz Barrera.

Agradecimientos

A Igor, por el tiempo y el esfuerzo dedicados
a la revisión de esta tesis.

A David, por el talento desplegado en el arte conceptual.

A Alexis, por la dedicación, y sus conocimientos.

La imaginación es más importante que el conocimiento

Albert Einstein

ÍNDICE

1. INTRODUCCIÓN.

En el año de 1958 William A. Higinbotham, quien trabajaba en los Laboratorios Nacionales Brookhaven, se percató de que la gente que visitaba las exposiciones del laboratorio se aburría al observar equipo estático y simples fotografías. Ya que era confeso amante del juego de *Pinball*, se propuso desarrollar una exhibición que entretuviera y educara a los observadores.

Su idea fue usar una pequeña computadora analógica para desplegar en un osciloscopio la trayectoria de una pequeña pelota, con la cual los usuarios pudieran interactuar; con ayuda del técnico especialista Robert V. Dvorak, logró crear en sólo tres semanas el primer videojuego llamado "*Tennis for Two*" (Tenis para dos) el cual hizo su debut en octubre de 1958; aunque se trataba de un juego muy simple, los visitantes hacían fila por horas para poder jugar.

En nuestros días, la industria de los videojuegos es un negocio incluso más rentable que la industria del cine, cálculos de la consultora *NDP Group* estiman que el mercado de videojuegos, tan sólo en México, deja ganancias de $2 mil 535 millones de dólares al año y que vende alrededor de 73.4 millones de artículos. Por ello, se espera que sea una de las industrias más cotizadas en los próximos años, sobre todo en los países de habla hispana, en donde dichos productos han aumentado sus ventas hasta en un 80 por ciento.

Por otra parte, el desempleo en México constituye uno de los mayores problemas, no sólo para los recién egresados, sino para toda la población. El Instituto Nacional de Estadística, Geografía e Informática informó que la tasa de desempleo abierto se ubicó en 3.78% en noviembre del 2004, superior que en octubre cuando llegó a 3.60%.

El *INEGI* reportó que, durante el tercer trimestre del 2004, había 1.2 millones de personas sin empleo, mientras que la población ocupada ascendió a 42.3 millones en un país de aproximadamente 105 millones de habitantes. Por localidad, la tasa más alta de gente sin empleo durante octubre se registró en la ciudad de México, con 5.2%. La tasa incluye a cualquier persona mayor

de 12 años que durante el periodo de referencia no trabajó ni una hora a la semana, a pesar de que intentó buscar empleo o quiso ejercer una actividad por su cuenta.

En México la producción de videojuegos es muy poca, prefiriendo la importación del producto, lo que deja abierta la posibilidad de incursión y desarrollo, si se aúna esto con el hecho de que en la actualidad se esta viviendo una explosión en el uso de dispositivos móviles (teléfonos celulares, asistentes personales digitales, computadoras de mano) para desarrollar distintas actividades, ya sea de productividad, comunicaciones, ocio, etc., se presenta una gran oportunidad de negocios con la cual se puede aportar, aunque sea en cierta medida, a solucionar el problema del desempleo.

A lo largo de este trabajo de tesis el equipo presentará las bases teóricas necesarias para implementar un videojuego, así como un pequeño demo que aplique lo expuesto, esto es para demostrar que en México existe la capacidad no solo para desarrollar videojuegos, sino también para implementarlos en dispositivos móviles, lo que se logrará mediante la utilización de técnicas de ingeniería de software y diseño de videojuegos para crear un juego del tipo rol (*Role Playing Game, RPG*) que se ejecute sobre una plataforma portátil.

2. CONCEPTOS.

A lo largo de este trabajo se utilizarán varios términos y conceptos necesarios para cumplir el objetivo final de este trabajo, el cual es el desarrollo del videojuego. Dichos términos y conceptos serán listados y explicados a continuación.

2.1. Videojuegos.

Los videojuegos constituyen un término popular que se inserta en el proceso de desarrollo tecnológico que experimenta la sociedad. Tales videojuegos se introdujeron por primera vez en los Estados Unidos a principios de los años 70, con un éxito sin precedentes en los salones recreativos hasta entonces ocupados por máquinas tragamonedas y *pinballs*.

La producción masiva de videojuegos surgió a principios de la década de los setenta de las manos de Nolan Bushnell creador de la empresa *Atari*. Conocido como *Pong-Pong*, se trataba de un juego sencillo de tenis de mesa, compuesto por dos barras que simulaban las raquetas, y un cursor que, moviéndose, atravesaba la pantalla, apareció en un principio en una máquina que funcionaba con monedas de la que, en menos de un año, se vendieron aproximadamente 6000 unidades en los Estados Unidos. De forma paralela a la aparición de *Pong-Pong*, la compañía *Magnavox* comercializó un videojuego conocido como *Odyssey* que, en lugar de jugarse mediante máquinas de funcionamiento con monedas, podía utilizarse a través de las televisiones domésticas (por medio de una unidad de control acoplada al aparato de televisión, que permitía jugar distintos juegos insertando una tarjeta de programación).

En poco tiempo, tras la aparición de los primeros videojuegos señalados, se introdujeron mejoras considerables en los mismos y así, en 1975, *Atari* incursionó en el mercado del videojuego doméstico con una versión de *Pong* que ofrecía múltiples novedades: efectos sonoros para cada error, logro o

rebote, un marcador digital automático en la pantalla, etc. A partir de 1976, unas veinte compañías diferentes comenzaron a dedicarse a la producción de videojuegos domésticos, a los primeros juegos (*Pong* y *Odyssey*) les siguieron otros como *Space Invaders*, creado en 1979, probablemente el videojuego que ha conocido más versiones y adaptaciones desde los modelos para computadoras personales hasta los pensados para todo tipo de consolas, más tarde llegaron otros videojuegos célebres como son *Missile Command*, *Asteroids* y *PacMan*.

La popularidad del término videojuegos y el consecuente uso del mismo, contrastan con la falta de especificidad cuando se realiza una revisión de la literatura existente. Si bien se hallan numerosas investigaciones sobre distintos aspectos de los videojuegos, pocos son los autores que realizan una clarificación previa sobre el término, probablemente porque, en la mayoría de casos, su significado se da por supuesto. Sin embargo, llegar a un consenso sobre este significado no es una tarea fácil. En primer lugar, porque a menudo el concepto de videojuego es utilizado de forma indistinta para hacer referencia tanto a su componente tecnológico como al tipo de juego. En otras palabras, el videojuego designa tanto al *hardware* como al *software*; junto con ello y en segundo lugar porque la proliferación de innovaciones tecnológicas ha diversificado mucho el mercado y así nos hallamos ante multitud de posibilidades (consolas domésticas, máquinas recreativas, computadoras personales, etc.) que presentan sus propias ofertas diversificadas en el mercado de los videojuegos.

Así no es difícil imaginar que la mayoría de las investigaciones que se sitúan en torno a la década de los 80, cuando utilizan el concepto de videojuego, le atribuyen un significado (ya sea referido al *hardware* o al *software*) muy distinto al que puedan atribuirle aquellas investigaciones que se han desarrollado a lo largo de los últimos años.

Los videojuegos constituyen una de las actividades de ocio o entretenimiento más populares de nuestros días. Además, su campo de

actuación, desde la segunda mitad de la década de los 80, se ha ampliado y sobrepasado la frontera del entretenimiento abriendo posibilidades de uso en el ámbito educativo.

Hoy en día, limitar el concepto de videojuego a una actividad exclusivamente lúdica supone reducir las potencialidades instructivas o educativas del videojuego, estudiadas a partir de numerosas investigaciones. Por otra parte implica dejar a un lado todo un conjunto de videojuegos de carácter didáctico que, por combinar las funciones lúdica y pedagógica, cuentan con una gran difusión y aceptación en el mercado actual de videojuegos.

Los videojuegos son algo más que un producto informático, también son un negocio para quienes los manufacturan y venden, y un producto comercial sujeto, como todos, a las fluctuaciones del mercado. De igual forma pueden definirse como un instrumento de información que cumple importantes funciones hegemónicas en la perpetuación de muchos estereotipos, o como un campo de investigación en el que el investigador puede plantear cuestiones relacionadas con el sentido de nuestra propia cultura. Esta definición ampliada de los videojuegos, a partir de la suma de los distintos elementos que rodean a este concepto, confirma el hecho de que el videojuego puede ser entendido como un fenómeno social.

A efectos operativos, en función de los objetivos del presente trabajo y teniendo en cuenta las dificultades que conlleva hallar una definición comprensiva e inequívoca del término, se entenderá como videojuego: todo juego electrónico con objetivos esencialmente lúdicos que, sirviéndose de la tecnología de la información, se presente en distintos soportes (fundamentalmente consolas y computadoras personales). Haremos uso del término para designar todo el *software* específicamente diseñado para jugar, independientemente de que el soporte (o *hardware*) que se utilice sean máquinas de los salones recreativos, consolas portátiles, consolas domésticas o computadoras personales.

2.2. Juegos de Rol.

Juego de rol (*RPG* por sus siglas en inglés, *Role Playing Game*), es un tipo de juego en el que el jugador toma el papel de un personaje, y lo guía a través de diversas aventuras.

Un juego de rol establece básicamente un conjunto de reglas generales para que varios individuos interpreten una historia, bajo un marco de referencia común a todos.

Un RPG consta principalmente de cuatro elementos: un escenario en el que se desenvuelve el juego (incluyendo las reglas); un director que define, plantea y coordina la acción, jugadores que interpretan los roles y una historia que es el hilo conductor de todo.

- *Escenario*: De acuerdo con lo que se pretenda jugar hay diversos tipos de escenarios: fantasía, ciencia-ficción, históricos, de terror, etc.
- *Director*: También conocido como *dungeon master*, *game master*, o *storyteller*. Es la persona encargada de crear la historia inicial, plantear las situaciones a los personajes, resolver sus diferencias, y premiarlos o castigarlos de acuerdo con sus acciones. Es el responsable de dirigir el curso de la historia y adaptarlo de acuerdo con los eventos que ocurren a lo largo de la misma.
- *Jugadores*: Son aquellos que se asumen como personajes cuando comienza el juego. Así, la historia toma vida propia y se nutre de los aportes de todos y cada uno de los que participan en ella.
- *Historia*: En un juego de rol todos los elementos forman parte de la historia. La historia es aquello a lo que se juega, pero esta cambia constantemente como respuesta a las acciones de los jugadores y a sus aportes.

En el caso de los videojuegos, los elementos anteriormente mencionados existen, pero se encuentran limitados debido a la enorme complejidad que suponen, por ejemplo, la historia no puede ser completamente abierta, no puede depender completamente de las decisiones del jugador para evolucionar,

ya que esto conllevaría a un número infinito de posibilidades, y por consiguiente, a un número infinito de líneas de código.

Los RPG se clasifican dependiendo de los elementos que lo formen. De esta manera se tiene:

- *Fantasía*: Rol en un mundo imaginario que suele ser una mezcla de la Edad Media y el Renacimiento. Se caracteriza por héroes con espada y hechiceros que tienen como misión salvar al mundo de algún mal antiguo y abrumador. El ejemplo más conocido es *Dungeons & Dragons*.
- *Horror*: Rol en un mundo imaginario donde existen horribles criaturas, como las mostradas en las películas de terror. Los personajes suelen ser personas normales que tratan de detener el horror para que no conquiste a la gente que no sospecha nada. Un ejemplo puede ser *La llamada de Cthulhu*.
- *Superhéroes*: Rol en el mundo de las historietas, donde los impresionantes héroes con poderes especiales intentan detener a un malvado tirano que trata de conquistar el mundo.
- *Galáctico*: Suele ser en el futuro lejano, y los personajes (humanos, alienígenas o robots) exploran "extraños nuevos mundos" (como en *Star Trek*), o pueden ser luchadores por la libertad combatiendo a un imperio (como en *Star Wars*).
- *Cyberpunk*: Rol en un mundo imaginario, donde gigantescas corporaciones dirigen a los gobiernos. Los personajes son humanos, por lo general mejorados por implantes cibernéticos (como el *Hombre Biónico*). Algunos personas intentan hacer del mundo un lugar mejor, luchando contra las corporaciones, mientras otros solo quieren sobrevivir.
- *Viejo Oeste*: Rol en el ambiente histórico del Oeste norteamericano, con vaqueros e indios. Puede ser jugado en cualquier periodo de la era "pionera" (principios y mediados hasta fines del siglo XIX), pero

por lo general es después de la Guerra Civil norteamericana. Los personajes suelen ser hombres de la ley persiguiendo bandidos.

- *Bélico*: Rol en una de las muchas guerras de la historia humana. Suele ser en alguno de los conflictos más modernos (Segunda Guerra Mundial, Corea o Vietnam), pero puede ser en otros. Los personajes son soldados cumpliendo misiones para que su patria o bando gane la guerra.
- *Post-Holocausto*: Rol en un mundo después de un gran desastre nuclear, en el futuro lejano, donde la radiación ya no es tan alta. Los personajes pueden ser humanos, mutantes, *cyborgs* o robots que suelen explorar las ruinas de las antiguas grandes ciudades (como Chicago o Nueva York).

Además, hay juegos de muchos tipos más, como aquellos basados en *manga*, que es una historieta japonesa. Un juego no necesariamente debe estar dentro de estas clasificaciones, ya que puede mezclar dos o más categorías para adaptarse al gusto del director de juego y al de los jugadores, pues depende de ellos la atmósfera en la que deseen desenvolverse.

2.3. Motor del juego (*game engine*).

El motor de juego, mejor conocido como *game engine*, es una librería de programación que permite a los diseñadores de videojuegos crear juegos de una manera más simple y rápida.

El *game engine* reduce el trabajo de los desarrolladores al implementar mecanismos de bajo nivel como el *render* del modelo del mundo de juego, captura y algunas veces manejo de entradas del usuario (*joysticks*, ratón, entradas de teclado, etc.), reproducción de sonidos, soporte para juegos de varios jugadores a través de una red, simulación física y detección de colisiones, etc.

Es responsabilidad del *game engine* el manejar las tareas de preparación de un videojuego, esto lo logra al cerciorarse que se ejecute apropiadamente y

luego finalizando correctamente la aplicación. Aunque es verdad que estas tareas se realizan en cualquier programa, algunos aspectos de la inicialización, ejecución, y finalización son únicos de los videojuegos. Es por esto que es importante para un *game engine* el manejar las necesidades únicas de los videojuegos y ayudar para que el proceso de desarrollo de juegos sea tan simple como sea posible. Con un *game engine* bien diseñado, es posible crear videojuegos con mucho menos código que no utilizando ninguno, la idea es desarrollar una vez ciertas rutinas de juego, agregarlas al *game engine*, y nunca más volver a preocuparse de ellas.

En conclusión, se puede pensar en un *game engine* como un sistema operativo para videojuegos, el cual provee todo el soporte necesario para crear videojuegos completos en una pequeña fracción de tiempo.

2.4. Dispositivos móviles.

Un dispositivo móvil es cualquier aparato electrónico que por sus características de tamaño, forma, peso, fuente de alimentación, forma de interacción con el usuario y función que desempeña, es portátil.

En la actualidad se está viviendo una explosión en el uso de dispositivos móviles (principalmente en teléfonos celulares, *Personal Digital Assistants* o *PDAs*, computadoras de mano) para desarrollar distintas actividades, ya sea de productividad, comunicaciones, ocio, etc. Dentro de éstas se pueden resaltar:

- Aplicaciones para manejo de inventarios y registro de productos.
- Recaudación de información a través de cuestionarios.
- Consulta de información desde cualquier punto en cualquier momento (información en línea).
- Comunicación con otras personas o dispositivos.
- Localización y planeación de rutas.
- Aplicaciones médicas y de salud.
- Aplicaciones personalizadas.

- Agendas y organizadores personales.
- Aplicaciones de ocio y entretenimiento.

Además, el uso de dispositivos móviles proporcionan las siguientes ventajas:

- Movilidad.
- Acceso a la información a cualquier hora y en cualquier lugar.
- Interfaz sencilla e intuitiva.
- Personalización del dispositivo.

Los dispositivos inalámbricos que actualmente se pueden encontrar en el mercado son básicamente de dos tipos: los teléfonos móviles (celulares y satelitales) y los PDA.

La compañía *IDC* (dedicada a la consultoría e inteligencia de mercado) proyecta que se venderán más de 167 millones de dispositivos en el 2005, a una tasa de crecimiento compuesto anual del 40% a partir del 2000.

Muchas empresas distribuyen PDA o teléfonos celulares a sus empleados para aumentar la comunicación y productividad, lo que permite llevar con ellos todos los datos y aplicaciones que pudieran necesitar en su labor, esto reduce tiempos y costos en el manejo de la información.

Este fenómeno se debe a la gran versatilidad que ofrecen estos dispositivos. El desarrollo en el campo de los dispositivos móviles permite tener diversas funciones en un solo instrumento, que a su vez puede comunicarse fácilmente con otros dispositivos móviles, equipos de escritorio o conectarse a distintas redes.

Las capacidades de conectividad van desde la posibilidad de únicamente sincronizar con el equipo fijo hasta la capacidad de conexión inalámbrica continua. El mercado inalámbrico está, en estos momentos, en plena ebullición, con multitud de fabricantes ofreciendo soluciones capaces de conectar dispositivos móviles.

- *Teléfonos celulares*: Aquí se ha enfocado el mayor avance tecnológico, debido a su gran aceptación en el mercado.

  La principal causa de este éxito ha sido la posibilidad de

comunicación, lo que lo hace un accesorio común en la vida cotidiana.

- *Asistentes Personales*: Son dispositivos que, respecto a los teléfonos celulares, tienen mayores capacidades tanto de procesamiento como de memoria y aplicaciones. Estos dispositivos presentan mayores ventajas respecto a las limitantes de los teléfonos celulares, tales como pantallas más grandes, entrada por teclado o pantalla sensible al tacto, tarjetas de expansión de memoria, y la posibilidad de cargar fácilmente aplicaciones de toda índole, etc.

  Sin embargo su mercado presenta una tendencia a la baja: "De acuerdo al último reporte mundial del *IDC*, las ventas cayeron un 21% con respecto al año pasado [2002], tras venderse 2.45 millones de unidades. […] Debido a que la mayoría de las empresas no toman a las *handhelds* como un elemento clave de su infraestructura *IT*, las demandas decayeron al mismo tiempo que el presupuesto de estas empresas. En el mercado de consumidores, las *handhelds* son vistas principalmente como un lujo, por ende, una declinación en la confidencia de los usuarios ha impactado negativamente la demanda de éstos productos."1

- *Dispositivos híbridos*: Cada día que pasa se puede observar el lanzamiento de nuevas terminales de telefonía móvil que permiten a los usuarios la realización de más funciones con el mismo aparato. Primero fue simplemente la introducción de una agenda para guardar los números de teléfono de los contactos, más tarde la posibilidad de añadir logotipos personalizados a la pantalla del teléfono, o tonos para indicar la recepción de una llamada. Las aplicaciones Java fueron el siguiente paso, a la par, algunos fabricantes empezaban a producir teléfonos que funcionaban con los sistemas operativos

[1] La venta de handhelds bajó un 21% , *Zona PDA*, 2004

destinados al mundo de las computadoras de mano: *PalmOS* y *Windows CE*.

Por lo antes expuesto dejo de ser necesario adquirir dos aparatos diferentes para las funciones de PDA y teléfono, cuando se puede tener todo esto integrado en un solo dispositivo. Por el momento, lo único que ha frenado el crecimiento de este sector son los altos precios de estos aparatos híbridos.

No obstante, y según se desprende de uno de los últimos estudios publicados por la consultora *IDC*, la influencia de los dispositivos híbridos en las ventas de PDAs ya se ha empezado a notar: este año se prevé que las ventas de PDAs sean un 8.4% inferiores a las de 2002, dato que, sin la explicación anterior, sería sorprendente, pues el de las computadoras de mano es un sector de crecimiento casi constante.

Según el mismo informe, el principal protagonista de esta migración no sería *Microsoft* con sus *Windows Mobile* o *Smartphone*, sino el consorcio *Symbian*, con un sistema operativo mejor posicionado comercialmente y también más enfocado a los dispositivos híbridos.

2.5. Lenguajes de programación.

Se llama lenguaje de programación al lenguaje artificial que agrupa un conjunto de instrucciones utilizadas en la creación de programas de computadora. En la actualidad existen un gran número de lenguajes para todos los tipos de necesidades, es decir, algunos lenguajes de programación se enfocan en funciones matemáticas y analíticas, mientras otros pueden ser mejores en la creación de aplicaciones de proceso de datos o en el manejo de operaciones de negocios.

Como ya se ha mencionado, existe una gran variedad de lenguajes de programación, cada uno con sus aplicaciones específicas, aun así, para que un lenguaje de programación sea considerado “bueno” es necesario tener en cuenta los siguientes puntos:

1. *Claridad y simplicidad*: La sintaxis de un lenguaje afecta considerablemente la facilidad con la que un programa puede ser desarrollado y entendido bastante tiempo después de haber sido creado, es por eso que la legibilidad es un punto muy importante, el lenguaje debe de manejar conceptos claros y simples que puedan ser utilizados para crear algoritmos. Para esto lo mejor es que haya un número reducido de conceptos y que las reglas para combinarlos sean tan simples y regulares como se pueda.
2. *Ortogonalidad*: La ortogonalidad se refiere al hecho de que las diferentes partes de un lenguaje puedan ser combinados de cualquier forma posible y cada combinación tenga un significado lógico dentro del programa, un lenguaje así se vuelve fácil de leer y los programas se vuelven fáciles de escribir ya que hay menos excepciones y casos especiales que recordar.
3. *Naturalidad*: Una de las principales razones que ayudan a la difusión de un lenguaje es precisamente la naturalidad con la que se pueda utilizar, y que la sintaxis ayude a que la estructura del programa pueda reflejar el funcionamiento lógico del algoritmo. Un lenguaje que está diseñado para cierto tipo de aplicaciones puede simplificar significativamente la creación de programas enfocados a esa área.
4. *Soporte para abstracción*: Una parte importante del trabajo de un programador es diseñar abstracciones, con base en las características primitivas del lenguaje, con las cuales poder resolver los diferentes problemas que se le presentan. Es deseable que un lenguaje pueda ser muy natural, pero si dentro de su misma estructura facilita el diseño de abstracciones, será un apoyo que le dará aún más potencial.
5. *Fácil de verificar*: Una de las cualidades que todo programador busca es que sus programas puedan analizarse con facilidad, y así poder encontrar errores o pulir el código más rápidamente, los aspectos que podrán ayudar en este análisis son la simplicidad de la semántica y la estructura sintáctica del lenguaje.

6. *Portabilidad*: Para los desarrolladores de software siempre será de gran interés que sus programas puedan correr en el mayor número de computadoras posibles, sin importar si es *PC*, *Mac*, *SPARC*, etc., es por esto que es importante que sea fácil de implementar entre las diferentes arquitecturas. Un lenguaje ampliamente disponible, cuyas definiciones sean independientes de las características específicas de una máquina en particular, constituye una base sólida para la portabilidad de programas.
7. *Costo de uso*: Este es un factor muy importante en el desarrollo de videojuegos, se puede decir que existen cuatro tipos de costo (en tiempo) de uso de un lenguaje:
    - Costo en la ejecución del programa.
    - Costo de compilación.
    - Costo de creación, prueba y uso.
    - Costo de mantenimiento del programa.

Pero el principal de todos ellos y de vital importancia en un videojuego es indudablemente el costo de ejecución, se puede estar hablando del lenguaje más fácil de usar, el más portable que exista, con una ortogonalidad increíble, pero si es lento al momento de ejecutarse no sirve absolutamente de nada.

8. *Paradigma de programación*: Un último punto, pero no menos importante, es el enfoque de cómo el lenguaje maneja su estructura interna, los dos paradigmas de programación más conocidos son la programación estructurada y la programación orientada a objetos (*OOP*, *Object Oriented Programming*), la programación estructurada está diseñada para que al programador se le facilite enfocarse en las acciones del programa, en el "¿qué es lo que se va a hacer?" y "¿cómo se va a hacer?", en cambio la programación orientada a objetos está diseñada para enfocarse más en los "objetos" que están actuando en el programa y cómo interactúan dentro de él, es decir, en el "¿quién lo va a hacer?" y "¿qué puede hacer?".

2.6. Paradigmas de programación.

Un paradigma de programación es un modelo básico de diseño y desarrollo, que permite producir programas con unas directrices específicas, tales como: estructura modular, fuerte cohesión, alta rentabilidad, etc.

Para algunas personas puede resultar sorprendente que existan varios paradigmas de programación. La mayor parte de los programadores están familiarizados con un único paradigma, el de la programación estructurada, pero existen multitud de paradigmas atendiendo a alguna particularidad metodológica o funcional.

Un paradigma de programación es una colección de modelos conceptuales que juntos modelan el proceso de diseño y determinan, al final, la estructura de un programa.

Esa estructura conceptual de modelos está pensada de forma que determinen la forma correcta de los programas y controlen el modo en que pensamos y formulamos soluciones, y al llegar a la solución, ésta se pueda expresar mediante un lenguaje de programación. Para que este proceso sea efectivo las características del lenguaje deben reflejar adecuadamente los modelos conceptuales de ese paradigma.

Cuando un lenguaje refleja bien un paradigma particular, se dice que soporta el paradigma, y en la práctica un lenguaje que soporta correctamente un paradigma, es difícil distinguirlo del propio paradigma, por lo que se identifica con él.

Como ya se ha mencionado anteriormente, los dos paradigmas de programación más conocidos y utilizados son la programación estructurada y la programación orientada a objetos, los cuales serán definidos a continuación.

1. *Programación estructurada*: Es una teoría que consiste en construir programas de fácil comprensión. Ésta es especialmente útil cuando se necesitan realizar correcciones o modificaciones después de haber concluido un programa o aplicación. Cuando se ha utilizado la programación estructurada, es mucho más sencillo entender la

codificación del programa, ya que éste se organiza en diferentes secciones.

La programación estructurada tiene como base una metodología de desarrollo de programas llamada refinamiento sucesivo, la cual consiste en plantear una operación como un todo, luego dividirla en segmentos más sencillos o de menor complejidad; una vez terminados todos los segmentos del programa, se procede a unificar las aplicaciones realizadas por el equipo de programadores. Si se ha utilizado adecuadamente la programación estructurada, esta integración debe ser sencilla y no presentar problemas, en caso de presentar alguno, será rápidamente detectable para su corrección.

La representación grafica de la programación estructurada se realiza a través de diagramas de flujo, estos representan al programa con sus entradas, procesos y salidas.

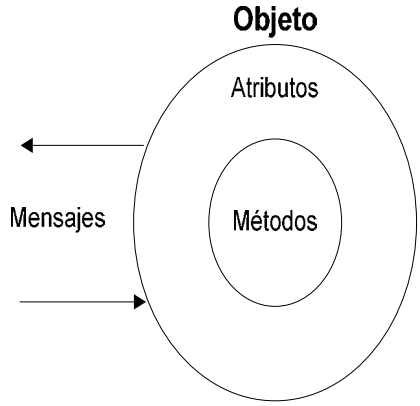
La programación estructurada propone segregar los procesos en estructuras lo más simples posible, las cuales se conocen como secuencia, selección e interacción. Dichas estructuras están disponibles en todos los lenguajes modernos de programación estructurada en forma de sentencias. Combinando esquemas sencillos se puede llegar a construir sistemas amplios y complejos pero de fácil entendimiento.

2. *Programación orientada a objetos*: La programación orientada a objetos, intenta simular el mundo real a través de la representación de objetos y sus interacciones. Los mecanismos básicos de la programación orientada a objetos son: objetos, atributos, mensajes y métodos.

   a) *Objetos*: Mientras que un programa tradicional se compone de procedimientos y datos, un programa orientado a objetos se compone solamente de objetos. Un objeto es una encapsulación genérica de datos y de los procedimientos para manipularlos, dicho de otra forma, un objeto es una entidad que tiene unos atributos particulares (las propiedades) y unas formas de operar sobre ellos

(los métodos). Por lo tanto, un objeto contiene, por una parte, operaciones que definen su comportamiento, y por otra, variables manipuladas por esas operaciones que definen su estado.

La estructura interna de un objeto está oculta para los usuarios, la única conexión que tiene con el exterior son los mensajes. Los datos que están dentro de un objeto solamente pueden ser manipulados por los métodos asociados al propio objeto.



b) *Mensajes*: Cuando se ejecuta un programa orientado a objetos, los objetos están recibiendo, interpretando y respondiendo a mensajes de otros objetos, esto marca una clara diferencia con respecto a los elementos de datos pasivos de los sistemas tradicionales. En otras palabras, un mensaje está asociado con un procedimiento, de tal forma que, cuando un objeto recibe un mensaje, la respuesta a ese mensaje es ejecutar el procedimiento asociado, el cual recibe el nombre de método.

c) *Métodos*: Un método se implementa en una clase y determina cómo tiene que actuar el objeto cuando recibe un mensaje, además, las propiedades permiten almacenar información para dicho objeto. Un método también puede enviar mensajes a otros objetos solicitando una acción o información.

La ejecución de un programa orientado a objetos realiza fundamentalmente dos acciones:

- Crea los objetos necesarios.
- Los mensajes enviados a unos y otros objetos dan lugar a que se procese internamente la información.

Finalmente, es necesario mencionar y definir las cuatro características fundamentales de la programación orientada a objetos, las cuales son: abstracción, encapsulamiento, herencia y polimorfismo.

a) *Abstracción*: Por medio de la abstracción se logra no detenerse en los detalles concretos de aquello que no interese en cada momento, sino generalizar y centrarse en los aspectos que permitan tener una visión global del tema, por ejemplo, el estudio de una computadora se puede realizar a nivel del funcionamiento de sus circuitos electrónicos, en términos de corriente, voltaje, etc., o a nivel de transferencia entre registros, centrándose así el estudio en el flujo e información entre las unidades que lo componen (memoria, unidad aritmética, unidad de control, registros, etc.), sin que importe el comportamiento de los circuitos electrónicos que componen estas unidades.

b) *Encapsulamiento*: Esta característica permite ver un objeto como una caja negra, la cual, contiene toda la información relacionada con dicho objeto. Esto permite manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

   La abstracción y el encapsulamiento están representadas por la clase. La clase es una abstracción, ya que en ella se definen las propiedades de un determinado conjunto de objetos con características comunes, y es una encapsulación, porque constituye una caja negra que encierra tanto los datos que almacenan los objetos como los métodos que permiten manipularlos.

c) *Herencia*: La herencia es el mecanismo para compartir automáticamente métodos y datos entre clases y subclases de

objetos.

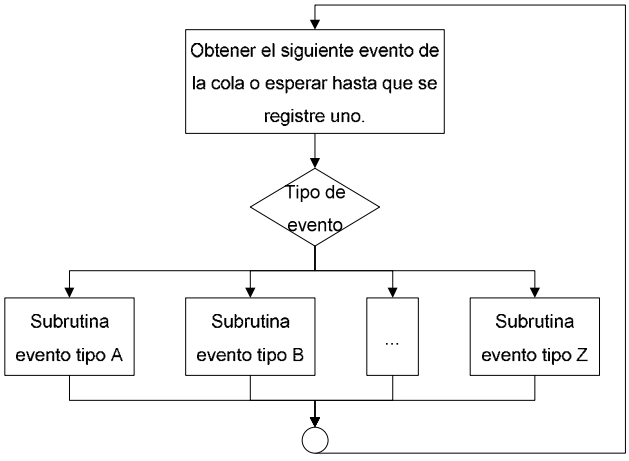
d) *Polimorfismo*: Esta característica permite implementar múltiples formas de un mismo método, dependiendo cada una de ellas de la clase sobre la que se realice la implementación, esto hace que se pueda acceder a una variedad de métodos distintos (todos con el mismo nombre) utilizando exactamente el mismo medio de acceso.

3. *Programación orientada a eventos*: Los eventos juegan un papel importante en el desarrollo de la mayoría del software que involucra interacción o simulación. Así, con este modelo se programa la respuesta del sistema a señales especificas (eventos) generadas por el mismo, o bien, por el usuario en su interacción con éste.

   Las aplicaciones creadas en este esquema contienen un ciclo (*event loop*) que continuamente está verificando los eventos que han sido registrados en una "cola de eventos" (*event queue*) a través de un despachador (*dispatcher*). Por cada evento en la cola, la aplicación debe verificar si es un evento de interés para el flujo del programa y atenderlo en una subrutina diseñada para ello (*event handler*).

   La subrutina informa si manejó completamente el evento y el *dispatcher* decide si debe pasarlo a otras capas del sistema (otros subsistemas o directamente al sistema operativo) antes de quitarlo de la cola.

   El flujo de este ciclo se representa en la siguiente figura:



2.7. *API*s y librerías.

Una librería es un conjunto de rutinas, previamente construidas, que se orientan a realizar algún trabajo en especial (como dibujar en pantalla, manipular sonidos, métodos numéricos, etc.). Así el programador ya no tiene que preocuparse por los detalles de cómo realizar esa tarea, simplemente incluye dentro de su propio código las instrucciones adecuadas que proporciona la librería y dichas instrucciones se ejecutarán y producirán el resultado deseado.

Una *API* (*Application Programmers Interface*) es la forma en la que se presentan al programador las funciones que provee una librería.

Por ejemplo, el estándar *ANSI C* especifica una serie de librerías que acompañan al lenguaje, comúnmente denominadas “biblioteca estándar”, estas librerías efectúan tareas como funciones matemáticas, de entrada salida, de manejo de cadenas etc. La especificación la proporciona la API a través de los prototipos de las funciones:

```
char *strcpy(char *s1, const char *s2);
```

La función strcpy copia la cadena a la cual apunta s2 (incluyendo el caracter nulo de fin de cadena) al arreglo al cual señala s1. Si la copia se lleva a cabo entre objetos que se superponen, el comportamiento queda indefinido, la función strcpy devuelve el valor de s1.

```
double cos (double x);
```

Calcula el coseno de x (medido en radianes);

```
int printf (const char *format,… );
```

La función printf imprime en la salida estándar el contenido de la cadena apuntada por format, aplicando previamente los modificadores indicados en la cadena.

Con esta información el programador sabe como llamar a las funciones que realizaran la tarea deseada.

2.8. *UML*.

El Lenguaje de Modelado Unificado (*UML* - *Unified Modeling Language*) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML es una especificación de notación orientada a objetos que entrega una forma de modelar conceptos como son procesos de negocio y funciones de sistema, además de ideas concretas como escribir clases en un lenguaje determinado, diseñar esquemas de base de datos y componentes de software reutilizables, etc.

UML divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto, estos diagramas juntos son los que representan la arquitectura del proyecto. Cada diagrama usa la notación pertinente y el conjunto de estos diagramas crea las diferentes vistas. Las vistas existentes en UML son:

1. *Vista de casos de uso*: Se forma con los diagramas de casos de uso, colaboración, estados y actividades.
2. *Vista de diseño*: Se forma con los diagramas de clases, objetos, colaboración, estados y actividades.
3. *Vista de procesos*: Se forma con los diagramas de la vista de diseño. Recalcando las clases y objetos referentes a procesos.
4. *Vista de implementación*: Se forma con los diagramas de componentes, colaboración, estados y actividades.
5. *Vista de despliegue*: Se forma con los diagramas de despliegue, interacción, estados y actividades.

Se dispone de dos tipos diferentes de diagramas: los que dan una vista estática del sistema y los que dan una vista dinámica. Los diagramas estáticos son:

- *Diagrama de clases*: Muestra las clases, interfaces, colaboraciones y sus relaciones. Son los más comunes y dan una vista estática del proyecto.
- *Diagrama de objetos*: Es un diagrama de instancias de las clases, que

muestra las instancias y como se relacionan entre ellas.

- *Diagrama de componentes*: Muestran la organización de los componentes del sistema. Un componente se corresponde con una o varias clases, interfaces o colaboraciones.
- *Diagrama de despliegue*: Muestra los nodos y sus relaciones. Un nodo es un conjunto de componentes. Se utiliza para reducir la complejidad de los diagramas de clases y componentes de un gran sistema. Sirve como resumen e índice.
- *Diagrama de casos de uso*: Muestran los casos de uso, los actores y sus relaciones. Permite visualizar las relaciones que existen entre actores y acciones (casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Lo diagramas dinámicos son:

- *Diagrama de secuencia* y *Diagrama de colaboración*: Muestran a los diferentes objetos y las relaciones que puede haber entre ellos, los mensajes que se envían. Son dos diagramas diferentes, se puede pasar de uno a otro sin perdida de información, ambos nos dan puntos de vista diferentes del sistema. En resumen, cualquiera de los dos es un diagrama de interacción.
- *Diagrama de estados*: Muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.
- *Diagrama de actividades*: Es un caso especial del diagrama de estados. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.

Como podemos ver, el número de diagramas es muy alto, en la mayoría de los casos excesivos, y UML permite definir solo los necesarios, ya que no todos son necesarios en todos los proyectos.

2.8.1. DIAGRAMA DE CLASES.

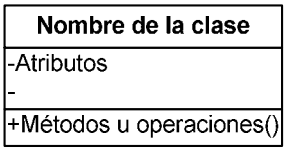
Los diagramas de clases son diagramas estáticos de UML. Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contención.

Un diagrama de clases esta compuesto por los siguientes elementos:

- *Clase*: Atributos, métodos y visibilidad.
- *Relaciones*: Herencia, asociación e instanciación.

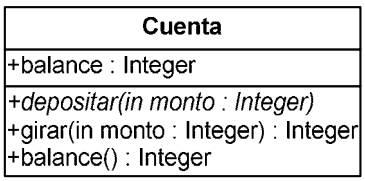
1. *Clase*: Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella se puede modelar el entorno en estudio (una casa, un auto, una cuenta corriente, etc.).

   En UML, una clase es representada por un rectángulo que posee tres divisiones:



   La división superior contiene el nombre de la clase, la intermedia contiene los atributos (o variables de instancia) que caracterizan a la clase (pueden ser *private*, *protected* o *public*), y la división inferior contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: *private*, *protected* o *public*).

   Por ejemplo, una cuenta corriente que posee como característica balance puede realizar las operaciones de: depositar, girar y balance, por lo que el diseño asociado es:



a) *Atributos*: Los atributos o características de una clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:
   - *Public* (+): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
   - *Private* (-): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden acceder).
   - *Protected* (#): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de las subclases que se deriven (ver herencia).

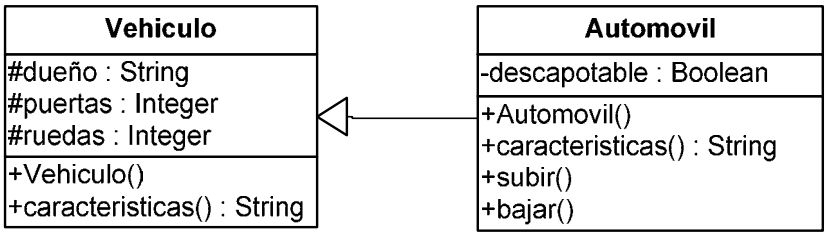
b) *Métodos*: Los métodos u operaciones de una clase son la forma en cómo ésta interactúa con su entorno, éstos pueden tener las características:
   - *Public* (+): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
   - *Private* (-): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden acceder).
   - *Protected* (#): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de métodos de las subclases que se deriven.

2. *Relaciones entre Clases*: Ya definido el concepto de Clase, es necesario explicar como se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes). Antes es necesario explicar el concepto de cardinalidad de relaciones, en *UML*, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser: Uno a muchos (1..*), cero a muchos (0..*) o número fijo (m), donde m denota al número.

a) *Herencia* (*Especialización*/*Generalización*):

Indica que una subclase hereda los métodos y atributos

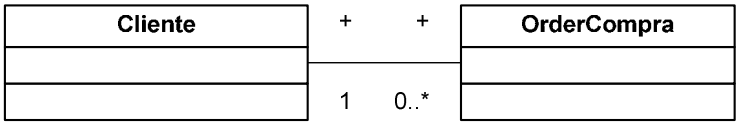
especificados por una súper clase, por ende la subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la súper clase (*public* y *protected*), ejemplo:



En la figura se especifica que Automóvil hereda de Vehículo, es decir, Automóvil posee las características de Vehículo, además de que posee algo particular que es descapotable. Cabe destacar que fuera de este entorno, lo único "visible" es el método características, aplicable a instancias de Vehículo y Automóvil, pues tiene definición publica, en cambio atributos como descapotable no son visibles por ser privados.

b) *Asociación*: 

La relación entre clases conocida como asociación, permite asociar objetos que colaboran entre sí. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro, por ejemplo:

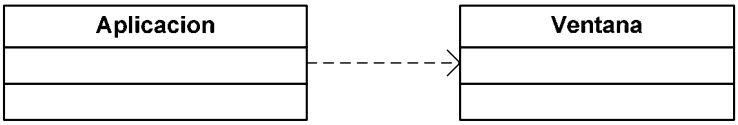


Un cliente puede tener asociadas muchas órdenes de compra, en cambio una orden de compra sólo puede tener asociado un cliente.

c) *Dependencia* o *instanciación* (*uso*):

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). Se denota por una flecha punteada.

Este tipo de relación se utiliza para denotar la dependencia entre clases, como por ejemplo una aplicación gráfica que instancia una ventana (la creación del Objeto Ventana está condicionado a la instanciación proveniente desde el objeto Aplicación):



Cabe destacar que el objeto creado (en este caso Ventana) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

2.8.2. DIAGRAMA DE CASOS DE USO.

Los casos de uso representan requisitos funcionales del sistema, se describen como conjuntos de secuencias. Cada una de estas secuencias refleja la interacción entre los elementos externos al sistema y el propio sistema (se trata de la descripción de escenarios o situaciones posibles donde se pone de relieve el comportamiento del sistema ante su uso por parte del usuario).

Así pues, los objetivos principales de la elaboración de casos de uso son:

- Definir el límite entre el sistema a desarrollar y los elementos externos a ese sistema (actores usuarios del sistema).
- Capturar el conjunto de funcionalidades y comportamientos del sistema a desarrollar.

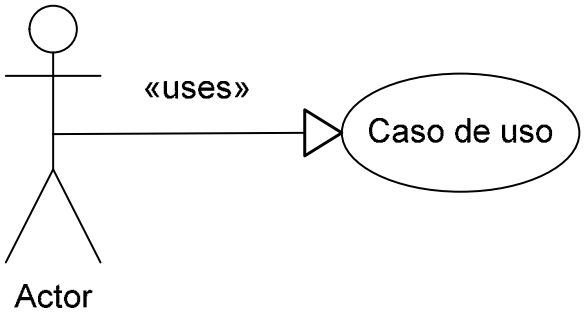
Cada caso de uso se documenta mediante una representación gráfica y un texto con la descripción de las situaciones o escenarios ante los que el usuario se pueda encontrar en su interacción con el sistema.

Los elementos que pueden aparecer en un diagrama de casos de uso son: actores, casos de uso y relaciones entre casos de uso.

1. *Actores*: Un actor es una entidad con comportamiento, como una persona (identificada por un rol), un sistema informatizado u organización, que realiza algún tipo de interacción con el sistema. Se representa mediante una figura humana dibujada con “palitos”. Esta representación sirve tanto para actores que son personas como para otro tipo de actores.

2. *Casos de uso*: Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el diagrama de casos de uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.
3. *Relaciones*: Un caso de uso, en principio, debería describir una tarea que tiene un sentido completo para el usuario. Sin embargo, hay ocasiones en las que es útil describir una interacción con un alcance menor como caso de uso. La razón para utilizar estos casos de uso no completos en algunos casos, es mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación de casos de uso. Para el caso de que queramos utilizar estos casos de uso más pequeños, las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes tipos:
   a) *Comunicación*: Es la relación entre un actor y un caso de uso con el que interactúa; se representa simplemente con una línea.
   b) *Uso* ("*include*", "*includes*", "*uses*"; se representa por una flecha apuntando en el sentido de la relación): Es una simple relación de inclusión, es decir, los escenarios o situaciones posibles detalladas en un caso de uso están incluidas en otro caso de uso (aquel del que, gráficamente, parte la flecha).
   c) *Extensión* ("*extend*", "*extends*"; gráficamente la representación es la misma que para "uso"): Este tipo de relación refleja situaciones particulares en un caso de uso que pueden ser extendidas por otro. En la descripción del caso de uso que es extendido debe haber una forma de indicar en que punto entra en juego el caso de uso que lo extiende (punto de extensión); esto se representa mediante una "etiqueta" (un texto significativo entre paréntesis) como referencia del lugar donde entraría a formar parte del caso de uso extendido.

d) *Generalización* (se trata del concepto de herencia, habitual en los diagramas de clases, pero aplicado entre casos de uso, e incluso entre actores; se representa por una flecha con un triángulo vacío por punta señalando en el sentido de la relación).



En este capítulo se pudieron repasar los conceptos y herramientas básicas que se utilizarán a lo largo del presente trabajo, dichos conceptos y herramientas se utilizarán tanto en el diseño, como en el desarrollo e implementación del videojuego.

3. DISEÑO Y DESAROLLO DE VIDEOJUEGOS.

Un videojuego es uno de los sistemas más complejos que existen, no solo por la cantidad de gente involucrada (equiparable a la de una producción de cine), sino por el hecho de que es una tarea multidisciplinaria la cual necesita de una planeación y desarrollo especial.

A lo largo de este capítulo se presenta metodología utilizada en el diseño y desarrollo de videojuegos, esto se hará tratando de enfocarse principalmente en la disciplina de ingeniería de *software* (con algunos otros tópicos que se consideran importantes), dejando un poco de lado, tópicos como la música, ambientación y algunos otros esto es por cuestión de complejidad.

3.1. La idea.

Los sueños son donde cada juego comienza, antes del código, antes de la planeación de *software*, antes del arte conceptual, aun antes del primer documento, el juego comienza como una chispa en la imaginación del diseñador, y es la idea, la entidad más persistente en el ciclo de desarrollo del juego. La idea va evolucionando y desarrollándose mientras el juego progresa, pero estaba ahí desde el inicio, es la semilla de la que un juego crece.

Así como los programadores no deben empezar a codificar sin planear, así los desarrolladores deben esperar antes de plasmar sus ideas en el papel, antes se deben ordenar y alimentar las ideas para luego comenzar a trabajar. El proceso creativo consta de cuatro fases las cuales son:

1. *Inspiración*: La inspiración se refiere a dónde conseguir ideas. En la mayor parte de los casos, la inspiración de los diseñadores de videojuegos viene siempre de las mismas fuentes, por ejemplo los *CRPGs* (*Computer Role Playing Games*) invariablemente están basados en las mecánicas del juego de mesa *Dungeons & Dragons*. Actualmente, casi todos los juegos parecen estar relacionados con otros en el mercado, haciendo que sean fácilmente encasillados dentro de géneros (estrategia en tiempo real,

tirador en primera persona, etc.).

Es por las razones ya explicadas que es necesario esforzarse para no crear otro clon de algún juego ya existente. La originalidad se puede presentar en muchos de los aspectos del juego como son: el modo de juego, la historia, el escenario, los personajes, la interfaz, o hasta la tecnología utilizada.

2. *Síntesis*: La síntesis se refiere a combinar ideas. No es suficiente tener una o muchas ideas y compilarlas, es necesario que esta amalgama funcione bien y para esto es necesario analizar cada idea y preguntar si contribuye a las demás para que el juego en creación haga pleno uso de todas ellas.
3. *Resonancia*: Es la creación de sinergia entre las ideas. La resonancia es una manera de hacer que el todo sea mayor que la suma de sus partes, esta es una manera efectiva de hacer que la historia sea significativa para los jugadores.
4. *Convergencia*: Se refiere a finalizar el concepto. En esta etapa es necesario ser autocrítico, es decir, juzgar si las ideas pueden trabajar juntas para hacer un buen juego. Es mejor anticipar las fallas para corregirlas en esta etapa antes de que el equipo de programación y arte empiecen a trabajar, ya que después el precio puede ser muy alto.

3.2. Modelado de la idea.

De acuerdo con el modelo del drama, existen cinco elementos que son: estilo, trama, personaje, escenario y tema. Estos elementos han sido combinados por más de veinte siglos en el teatro, opera, novelas, películas, etc.

Todos los buenos videojuegos deben entretener, y la mayoría gana mucho en este rubro a través del drama. El hecho de que el jugador tiene un mayor control del drama que en cualquier otro género del arte es una diferencia de intensidad y no de forma, por lo tanto, las reglas del drama son aplicables a los videojuegos.

- *Estilo*: El estilo en un videojuego esta dado por su género. Los géneros son identificables por la meta del diseñador, es decir, puede ser para asustar al jugador, retar su intelecto, o deleitarlo con imágenes fantásticas. Los principales géneros son:
    - *Acción*: Son juegos que requieren gran destreza manual.
    - *Aventura*: Juegos donde la historia es un factor primordial.
    - *Estrategia*: Son aquellos juegos donde la planeación y las decisiones son lo más importante.
    - *Simulación*: Se tratan de ejercicios de optimización.
    - *Acertijos*: Para estos juegos es necesario una gran cantidad de pensamiento analítico.
    - *Juguetes*: Se trata de software que sirve únicamente para diversión.
    - *Educacional*: Juegos con una filosofía de "aprender haciendo".

  Existen géneros híbridos como los RPGs los cuales son juegos de simulación con un toque de acción-aventura.
- *Trama*: En todos los videojuegos existe una trama, pero la mayoría de ella es creada por los mismos jugadores, en un buen juego, es el jugador el autor de los eventos del juego. El juego no es más que una herramienta que le permite al jugador crear historias.

  En los juegos de aventura la trama es particularmente importante, pero no se debe olvidar que los jugadores están impacientes por empezar a jugar, por lo que no se debe caer en el vicio de presentar videos muy largos.
- *Personaje*: La razón por la cual los diseñadores de videojuegos prefieren juegos con personajes identificables, es por que estos mejoran la historia (la cual está creando el jugador); esto se aplica tanto a los personajes creados por el jugador, así como a los propios del juego.
- *Escenario*: El escenario es un factor muy importante para darle

credibilidad a un videojuego, si, digamos en una ciudad asediada, el jugador se pregunta ¿quién vive ahí?, ¿cómo son?, o algo similar, entonces, se tiene un escenario bien planteado.

- *Tema*: El tema de un trabajo dramático es la idea filosófica que el autor está tratando de expresar. Aún con el hecho de que el verdadero autor de la narrativa de un juego debe ser el jugador, el diseñador puede encauzarlo hacia el tema que intenta exponer, siendo siempre consciente que el diseñador no puede hacer que un jugador piense como el.

3.3. Modo de juego (*Gameplay*).

En un videojuego, el *gameplay* se refiere a que el jugador puede ganar haciendo cosas inesperadas y haciéndolas funcionar, es decir, el *gameplay* es aquello que inspira al jugador a emplear estrategia.

Es durante el diseño del videojuego cuando se debe definir de manera clara el *gameplay*, esto se realiza con el objetivo de:

- Explicar a los diseñadores como debe funcionar el videojuego.
- Proveer una visión clara durante el desarrollo.
- Enfocar los puntos integrales que debe cumplir el videojuego.

Sid Meier, diseñador de exitosos juegos como *Civilization*, dijo: "Un juego es una serie de decisiones interesantes". Para que un videojuego sea exitoso, las decisiones del *gameplay* deben ser no triviales, es decir, cada estrategia que el jugador considere debe tener un lado positivo y uno negativo, en el caso, de que solo tenga lado positivo, la inteligencia artificial del juego debe encargarse de eso automáticamente, por otro lado, si solo tiene un lado negativo, entonces nadie usará esa estrategia.

Es importante recalcar que el videojuego debe ser una serie de decisiones interesantes, con cada decisión afectando la próxima, de manera que un juego bien diseñado no puede ser ganado sin una estrategia y una estrategia se manifiesta como una serie de decisiones interesantes.

3.4. Historia estructurada.

La historia se elabora mediante el estudio del material que entrega el creativo y se organiza de manera que parezca un guión cinematográfico. Se divide igual que un libro por capítulos, los cuales, se caracterizan por formar una serie de sucesos relacionados, por ejemplo, un ataque, un comienzo, la búsqueda de un objeto, una visita a un lugar, etc.

3.5. Sinopsis cronológica estructurada.

Por lo regular un equipo creativo presenta la historia acerca de las cosas que pasan en el juego al diseñador. Dicho texto tiene la estructura de un libro que cuenta los hechos mezclados con los diálogos, los lugares y las acciones, y muchas veces es un poco difícil de interpretar y usarlo para el desarrollo.

Para esto se usan las sinopsis que son breves descripciones de los acontecimientos en algunos lugares del juego. Se caracterizan por no tener diálogos ni acciones y por tener una estructura corta en su elaboración.

Su elaboración consiste en describir cronológicamente los acontecimientos con una estructura muy sencilla:

1. *Capítulos*: Es una pauta o titulo en el que se identifica un tiempo.
2. *Secciones*: En este identificador se describe el área en el que se desarrollan los hechos y describe algunas características de la atmósfera.
3. *Lugar*: Se describe el lugar en donde pasan los hechos.
4. *Acontecimientos*: Se desarrolla a grandes rasgos los personajes que están y que hacen.

Capitulo.- Primer capitulo (Aalok)	
Sección.- Darglia – noche – silencio.	Lugar.- Barracas, oficina de Aalok, escritorio con papeles, libreros.
Acontecimientos.- Aalok escribiendo su informe mientras recuerda a su esposa e hija.	

La sinopsis es muy parecida a un guión cinematográfico con la excepción de que no cuenta con los diálogos por personaje.

3.6. Diseño de búsquedas o misiones.

Las búsquedas o misiones son fracciones de la historia en donde el o los personajes tienen que hacer algo en especial, esto sirve para dar la mayor *jugabilidad* (qué tan divertido es) al producto y diferenciarlo de una animación. Aquí interviene mucho la creatividad del diseñador para dividir en partes cada tramo de la historia e hilarla de tal manera que no pierda el sentido ni se desvíe del camino deseado.

Una manera fácil de organizar las búsquedas o misiones es hacer una lista de cosas que hace un personaje, por ejemplo:

- Encontrar otro personaje.
- Platicar con alguien en especial.
- Obtener un objeto.
- Llegar a un lugar.
- Aprender o aumentar una habilidad.

Por ultimo se organizan en una lista procurando hacerlo en orden cronológico, y si están relacionados entre si se puede poner un paréntesis para indicarlo.

3.7. Diseño de personajes y diseño gráfico.

La mayoría de juegos necesitan personajes los cuales sirven para darle una identidad al producto. Existe una serie de niveles de importancia en los personajes para diferenciarlos:

- *Protagonista*: Es aquel personaje en torno al cual se desarrolla la historia.
- *Personajes principales*: Están ligados directamente con el protagonista.
- *Personajes secundarios*: No están ligados con el protagonista pero

tienen una interacción importante.

- *Personajes circunstanciales*: Actúan cuando se presenta un suceso especial.
- *Extras*: Son aquellos que no tienen interacción con los personajes y no intervienen en la historia, pero sirven para dar mayor realismo a la escena.

Según el nivel de participación de cada personaje, se considera la importancia de su diseño; este diseño tiene una manera muy sencilla para hacerse, y consta de un documento con los siguientes datos:

- Nombre.
- Genero.
- Altura.
- Complexión.
- Color de cabello.
- Rasgos notorios.
- Descripción.
- Historia resumida.
- Ropas características.

Los estudios profesionales se basan en el escrito del diseño antes planteado, para luego elaborar varios bocetos de los personajes. Tras elaborar estos bocetos eligen lo que mas se ajuste a sus necesidades y gustos, y lo incorporan a una hoja llamada diseño grafico por personaje.

Basándose en el documento de diseño de las búsquedas o misiones se observa que se puede sacar de ahí la mayoría de objetos que usan los personajes, llámense armas, aditamentos, prendas, comida, etc. A este tipo de objetos se les llama ítems u objetos. También existe otra clase de objetos que hay que diseñar, estos son los objetos de terreno o escenario (escenografía), como son, barriles, piedras, cajas, plantas, flores, etc. Se hace un documento en donde se enlistan estos objetos por jerarquías o importancia, personajes y

lugares. El diseño gráfico de los objetos es sencillo y se puede simplificar haciendo bocetos en páginas con casillas en donde se muestren varios objetos con su nombre o número.

A continuación se enlistan las diferentes escenas donde los personajes interactúan (guión), se organizan para ver si las mismas escenas se vuelven a usar. Se agrupan las que estén relacionadas y se les da una posición en un plano.

En el plano se hace un boceto con las escenas o lugares conectados por caminos y separados por muros. En los caminos se ponen las elevaciones del terreno y en los muros se ponen los edificios, árboles, paredes, objetos y otros elementos por donde los personajes no pueden andar.

3.8. Interfaz.

La interfaz es uno de los aspectos más difíciles del desarrollo de videojuegos, siempre se debe tener presente que la razón de la interfaz es la de ayudar al jugador a jugar el juego, por lo que debe ser intuitiva y el número de íconos debe mantenerse al mínimo. Un buen diseño de interfaz debe asegurarse de que el jugador no tenga que estar trabajando contra el sistema.

3.9. Abstracción.

Una vez que se encuentra completo el diseño conceptual del videojuego, es necesario analizarlo, para poder aplicar una planeación en su paso de una idea a la programación, para resolver tal asunto, se utiliza la abstracción.

La abstracción es un mecanismo fundamental para la comprensión de fenómenos o situaciones que implican gran cantidad de detalles. La idea de abstracción es uno de los conceptos más potentes en el proceso de resolución de problemas. Se entiende por abstracción la capacidad de manejar un objeto (tema o idea) como un concepto general, sin considerar la enorme cantidad de detalles que pueden estar asociados con dicho objeto. Sin abstracción no sería posible manejar, ni siquiera entender, la gran complejidad de ciertos problemas.

Por ejemplo, sería inimaginable pensar en el presidente de una gran multinacional que viese la empresa en términos de cada trabajador individual o de cada uno de los objetos que fabrica, en vez de departamentos especializados.

Se puede decir que la abstracción permite estudiar los fenómenos complejos siguiendo un método jerárquico, es decir, por sucesivos niveles de detalle. Generalmente, se sigue un sentido descendente, desde los niveles más generales a los niveles más concretos.

La solución de cualquier problema puede darse en varias formas o niveles de abstracción. Niklaus Wirth (1974) dijo: "Nuestra herramienta mental más importante para competir con la complejidad es la abstracción. Por tanto, un problema complejo no deberá considerarse inmediatamente en términos de instrucciones de un lenguaje, sino de elementos naturales del problema mismo, abstraídos de alguna manera".

El diseño *top down* consiste en encontrar la solución de un problema mediante la aplicación sistemática de descomposición en problemas cada vez más simples (aplicando la máxima de dividir para vencer).

Los programas son objetos complejos que pueden contener varios miles de instrucciones, cada una de las cuales puede dar lugar a un error del programa y que, por lo tanto, necesitan mecanismos de definición que eviten en la medida de lo posible que el programador cometa errores. Así, los lenguajes de programación de alto nivel permiten al programador abstraerse del sin fin de detalles de los lenguajes ensambladores y permiten trabajar de manera independiente respecto a las máquinas sobre las que finalmente se hará funcionar el programa.

En el proceso de programación se puede extender el concepto de abstracción tanto a las acciones, mediante la llamada abstracción procedural (uso de procedimientos), como a los datos, mediante los llamados tipos abstractos de datos.

Los procedimientos permiten encapsular partes de un algoritmo (hacer

módulos), localizando en una sección del programa aquellas proposiciones relacionadas con cierto aspecto del mismo. De forma que la abstracción procedural destaca qué hace el procedimiento ignorando cómo lo hace. El programador, como usuario de un procedimiento, sólo necesita conocer la especificación de la abstracción (el qué), limitándose a usar el procedimiento con los datos apropiados.

3.10. Estructura básica de un videojuego.

Cuando se está hablando de videojuegos, se está hablando de una forma completamente diferente de programar *software*, se trata de algo más complicado que un programa que sigue una secuencia lógica lineal como es el caso de la mayoría de los programas. Un navegador, un *chat*, un compilador, y demás no son más que programas que reaccionan cuando sucede algún evento, y después de que hayan terminado de hacer lo que tenían que hacer vuelven a esperar a que suceda algún otro evento.

Sin embargo, un videojuego es un programa que tiene que actuar en tiempo real, tiene que estar haciendo cálculos y dibujando en pantalla todo el tiempo, simplemente no puede esperar a que suceda un evento para poder actuar, aunque el jugador no haga nada, no presione una sola tecla, no mueva el ratón, no toque siquiera el *joystick*, el juego tiene que estar calculando el tiempo que lleva jugando en ese nivel, si le va a atacar algún enemigo, si esta cargando energía, y por supuesto dibujando en pantalla todo lo que sucede.

La mayoría de los juegos están construidos bajo una misma estructura básica, por supuesto, puede haber juegos que lo manejan con algunas variaciones, pero en esencia un juego trabaja de la siguiente forma:

1. Inicialización.
2. Ciclo de juego.
    a) Entrada.
    b) Procesamiento.
    c) Salida.

3. Finalización.

En la inicialización es donde se pone al juego en un estado ya predefinido, para que siempre esté de ese modo cuando empiece el juego, es decir, los valores iniciales del jugador como su energía, sus armas, la posición dentro del mapa, en qué mundo se encuentra, etc., además también en esta parte es donde se cargan las imágenes, sonidos, tabla de puntuación y demás cosas que se necesiten antes de iniciar el juego.

El ciclo de juego es la parte donde está toda la acción, es un ciclo que se va a estar repitiendo una y otra y otra vez hasta que el jugador pierda, gane o se salga del juego, en general se puede dividir esta sección en tres partes:

- *Entrada*: Aquí es donde se captura todo lo que hace el jugador, como presionar los botones del control, mover el ratón, presionar las flechas del teclado y toda la demás información que recibe el juego.
- *Procesamiento*: Esta es la parte donde se procesa toda la información que se recibió de entrada, quizás se podría decir que es el núcleo del juego, ya que en esta parte es donde se lleva a cabo la lógica del programa, los cálculos de la física del juego, la inteligencia artificial y, en sí, la forma como va a responder el juego a la entrada.
- *Salida*: En esta parte es donde se le envía al jugador toda la información que se procesó en el paso anterior, es decir, se le muestra la respuesta a lo que hizo; por lo general poniendo en pantalla los cambios, tocando sonidos, etc.

La parte de la finalización se ejecuta cuando el juego ha terminado, y todas las instrucciones que se ejecutan aquí tendrán que ver precisamente con dejar al juego en un estado óptimo, como guardar la puntuación que se logró durante el juego, los valores al finalizar, liberar recursos reservados durante el juego, etc.

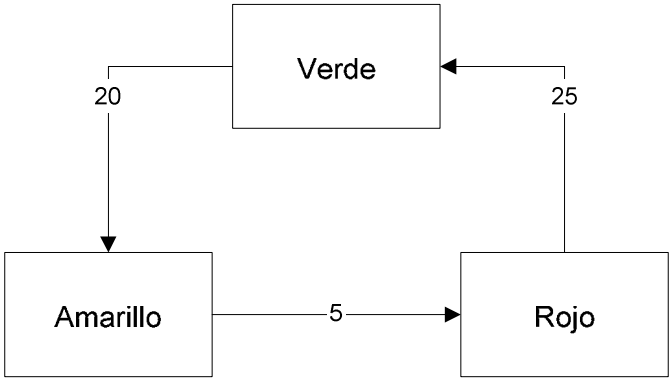
3.11. Autómatas (máquinas de estado).

Una de las técnicas que más ayuda en el desarrollo de un videojuego son sin duda los autómatas, también conocidos como máquinas de estado (en realidad

los autómatas son solo uno de los tipos de máquinas de estado, pero para fines prácticos se dejará así).

Los autómatas se suelen usar mucho en procesadores de palabras para corregir gramática, en editores de código y sobre todo en los compiladores en la fase de *escaneo* y *parseo* para revisar el manejo de símbolos y la gramática dentro del código que se trata de compilar.

Se puede pensar en un autómata como un objeto, un programa o una máquina que va cambiando de estados según lo que le vaya sucediendo o con lo que vaya interactuando. Por ejemplo, un semáforo puede estar en tres estados: rojo, amarillo y verde; sin embargo, no puede estar en dos estados al mismo tiempo, por lo que debe estar en solamente uno de los tres, además, se sabe que si está en verde después de algún tiempo, suponiendo, 20 segundos, va a pasar a amarillo, después de cinco segundos se pone en rojo y finalmente después de 25 segundos regresa nuevamente al verde. Esto se puede representar en un diagrama de estados de la siguiente forma:



Esto es fácilmente aplicable a un videojuego, ya que muchas de las cosas que hay dentro tienen estados, desde un cofre que contenga algún tesoro puede estar abierto o cerrado, o una antorcha que puede estar prendida o apagada, hasta el mismo enemigo principal del juego que puede estar tranquilo, alerta o atacando.

De esta forma se puede hacer que actúe el enemigo en forma diferente dependiendo del estado en el que se encuentre, si está atacando podría ir

revisando qué armas tienen más municiones y cuáles causan más daño, si está en alerta puede estar revisando qué enemigos tiene cerca, a cual le conviene más atacar o, dependiendo de qué tanto armamento y energía tenga, si es preferible escapar, si esta tranquilo puede estar buscando comida, etc.

3.12. Sincronización del juego.

Como ya se mencionó anteriormente, un videojuego siempre está en un ciclo constante, desde que se ejecuta el programa hasta que se sale de él, y es precisamente en este ciclo del juego donde se tendrán que realizar todos los cálculos que necesite, desde las animaciones de cada personaje que se encuentre en pantalla, los cálculos físicos, etc., es por esto que es muy importante siempre estar buscando la forma en como la computadora haría más rápido todos los cálculos, y al final, cuando ya esté listo el juego, optimizar el código.

Sin embargo, existe otro problema que hay que tener presente el cual se manifiesta cuando se ejecuta el videojuego en computadoras con diferentes velocidades de reloj. Suponiendo que se tiene una computadora con procesador a 200 MHz, y se esta haciendo un juego sencillo donde lo único que hay es un personaje que se mueve de izquierda a derecha en la pantalla y cuenta con cierto código dentro del ciclo principal del juego, el cual tarda 0.041 segundos en realizar todo el trabajo que tiene que hacer. Como se puede observar se esta dibujando el personaje en pantalla aproximadamente 24 veces por segundo, o, en términos correctos, a 24 *fps* (*Frames Per Second*), que es el máximo número de imágenes que puede percibir el ojo humano en un segundo, por lo tanto, la imagen del personaje moviéndose va a verse muy fluida, con una calidad bastante aceptable, y como solo va aumentando su posición en 2 píxeles por cada ciclo entonces cada segundo se estará desplazando 48 píxeles más a la derecha (2 x 24 = 48).

Hasta este momento no existe ningún problema, el personaje se mueve a 48 píxeles por segundo, y el juego corre a 24 *fps*, que es justo lo que se desea,

¿pero qué pasaría si se tratara de correr ese mismo juego en alguna otra computadora?, ¿que tal si su procesador es de 100 MHz o de 800 MHz y no de 200 MHz como la anterior?

Seguramente en la de 100 MHz el ciclo principal del juego en lugar de tardarse 0.041 segundos, se tardaría 0.082 segundos, lo que haría que en un segundo solo se pudieran ejecutar 12 ciclos, y por lo tanto se dibujaría en pantalla a una velocidad de 12 *fps*, y el personaje se movería solamente 24 píxeles cada segundo, que es mucho más lento que la velocidad deseada. Y en la computadora de 800 MHz el ciclo principal de juego tardaría solamente 0.01 segundos, que haría que el juego corriera 100 ciclos cada segundo, por lo tanto se dibujará en pantalla a una velocidad de 100 *fps*, y el personaje se movería a 200 píxeles por segundo.

Como se puede observar, el mismo juego, con exactamente el mismo código, incluso usando el mismo ejecutable, puede correr mucho más lento o más rápido dependiendo de la computadora en la que se ejecute, lo cual es necesario evitar. No es deseable que cuando se muevan en pantalla los objetos que se encuentran dentro del juego en algunas computadoras se vayan demasiado lento y desesperen al jugador, y en otras esos mismos objetos se muevan tan rápido que se vuelvan prácticamente imposibles de controlar.

Es por eso que, para evitar que el juego corra a diferentes velocidades en diferentes computadoras, es necesario sincronizar el juego, "forzar" a que el juego corra siempre de una forma al menos deseable si no es que óptima. Existen dos formas de sincronizar un juego: en base al *framerate* (velocidad de cuadro), y en base al tiempo.

1. *Sincronización por framerate*: Cuando se está sincronizando un juego en base a su *framerate* (número de frames por segundo), lo que se hace es detener el ciclo principal del juego hasta que queramos que se ejecute el siguiente ciclo, es decir, si se quiere que el juego corra a 24 *fps* lo que se tiene que hacer es fijar el *framerate*, o sea forzar a que cada ciclo dure 0.041 segundos, para lograr esto se deja que se ejecute todo el ciclo del

juego, y cuando termine de ejecutarse se revisa si pasaron 0.041 segundos desde que se inició, si no, se espera a que se cumplan los 0.041 segundos, y cuando se hayan cumplido se deja ejecutar el siguiente ciclo. Así pues, no importa si el juego se hizo en una computadora con un procesador a 200 MHz y se esta corriendo en otra con un procesador a 1.8 GHz, siempre se va a ejecutar el ciclo principal del juego 24 veces cada segundo.

El único problema con este tipo de sincronización es que si el juego se ejecutara en una máquina más lenta, este se alentaría mucho, ya que no se puede hacer que el procesador haga los cálculos más rápidos, y por consiguiente, el juego se estancaría con la velocidad a la que el procesador pueda ejecutar el ciclo principal del juego. Esto obliga a los desarrolladores de videojuegos a planear los requisitos mínimos de sistema para las computadoras a las que vaya dirigido el videojuego.

2. *Sincronización por tiempo*: Cuando se sincroniza un juego en base al tiempo no importa tanto el *framerate* que tenga el juego, de hecho es una de las grandes ventajas de sincronizarlo de esta forma, puede ser que el juego esté corriendo con un *framerate* de 8 *fps* o de 100 *fps*, lo cual no importa, ya que los objetos que se encuentren dentro del juego siempre se moverán a la misma velocidad.

   En si, está técnica consiste en introducir dentro del código operaciones que calculen el tiempo que le lleva al objeto moverse, y en base a ese tiempo manejar todas las animaciones. Esta es una técnica que se utiliza mucho en juegos 3D, ya que por como se manejan, suelen cambiar su *framerate* dependiendo que tan fácil o difícil les resulta *renderear* la imagen en pantalla y con esto se garantiza que la animación sea constante en todo momento. Los contra de esta técnica son principalmente que las operaciones necesarias para calcular el tiempo son un tanto pesadas en cuanto a procesamiento se refieren.

3.13. Profundidades de color (*Color modes*).

Antes de comenzar a trabajar con el ambiente gráfico de algún juego hay varias cosas que es necesario conocer, se necesita comprender como la computadora entiende y trabaja las imágenes y la forma como hace que queden plasmadas en la pantalla.

La pantalla del monitor está compuesta por píxeles, donde cada píxel puede tomar un color cualquiera (siempre y cuando la profundidad de color lo permita) y en conjunto con todos los demás píxeles que se encuentran en la pantalla en ese momento forman la imagen que se presenta. Entre más píxeles haya en la pantalla, las imágenes que se desplieguen van a ser más nítidas. Cuando se habla del número de píxeles que puede desplegar un monitor, se esta hablando de su resolución. Existen monitores de diferentes resoluciones, desde las de 640x480 píxeles hasta 800x600, 1024x768, 1280x1024 y más.

El color que se despliega en cada píxel es creado por la combinación de los 3 colores base, estos son el rojo, el verde y el azul, y dependiendo de la intensidad que tengan cada uno de estos colores es el color que se va a desplegar en pantalla. Los colores que son generados de esta forma se dice que son colores *RGB* (Rojo, verde y azul, o *Red*, *Green*, *Blue*).

Cando se dice que la profundidad de color es de 8 *bits* quiere decir que se van a utilizar ocho *bits* para representar todos los colores que se van a desplegar en pantalla, y por consecuencia solo se pueden alcanzar 256 diferentes colores (2^8), eso significa que para darle color a algún píxel en pantalla solamente se podrían utilizar ocho rojos, ocho verdes y cuatro azules diferentes (8 x 8 x 4 = 256) o alguna combinación semejante.

El primer problema con la profundidad de color de 8 *bits* es que alguno de los tres colores tendría solamente cuatro variantes, mientras que los otros dos tendrían ocho, el segundo problema es que la cantidad de colores que se podrían utilizar es muy limitada.

Es por esto que cuando se trabaja en 8 *bits*, normalmente se utiliza una paleta de colores en la cual se guardan todos los colores que se desean

desplegar en pantalla, y al momento de dibujar en pantalla solamente se hace referencia a los colores que estén guardados en la paleta. Es decir, una paleta de colores es como la típica paleta de colores que tienen los pintores cuando van a dibujar alguna obra; es un lugar donde se van a guardar todos los colores que se van a utilizar mientras dibujan sus obras, y cuando quieren utilizar algún color simplemente lo toman de la paleta. Esto significa que en la computadora una paleta de colores es un lugar (un arreglo) donde están guardados una colección de 256 colores RGB (representados por los colores rojo, verde y azul) que se van a utilizar para dibujar en pantalla.

Así pues, cada *byte* que represente un píxel en pantalla puede tener un valor entre 0 y 255, pero en lugar de que este número represente un color RGB, el número va a hacer referencia a uno de los colores que se guardan en la paleta de colores:

La ventaja de utilizar una paleta es que el usuario escoge cuales serán los 256 colores que va a utilizar, el problema es que no podría dibujar en pantalla otra imagen que usara otra combinación de colores, antes tendría que cambiar la paleta adecuada para dibujar la nueva imagen, o bien, podría crear una paleta con la cual se pudieran dibujar las dos imágenes. De este modo queda resuelto en gran medida el problema de la limitada cantidad de colores que puedes desplegar.

Representar un color por medio de 16 *bits* (llamado *Hi color*) es mucho más fácil que con 8 *bits*, ya que aquí no se utiliza ninguna paleta de colores, en este caso los 16 *bits* van a representar un color RGB, ya que con 16 *bits* se pueden generar 65,536 colores diferentes (2^{16}) para desplegar en pantalla.

Existen dos formas de representar un color RGB con 16 *bits*: por un lado se pueden dividir los 16 *bits* en tres partes (una parte para rojo, otra para verde y otra para azul), por lo que quedarían cinco *bits* para representar el rojo, seis *bits* para representar el verde y cinco *bits* para representar el azul.

Como se puede observar, el rojo y el azul son de cinco *bits* mientras que el verde es de seis *bits*, eso quiere decir que con el rojo y el azul solamente se

pueden generar 32 tipos diferentes (2^5), mientras que con el verde se puede generar 64 verdes diferentes (2^6). La razón por la que se escogió el verde para tener seis *bits* es sencilla, el ojo humano es más sensible al verde que a los demás colores, por lo tanto una persona notará más la diferencia entre los colores RGB usando 64 tipos de verdes que si usara colores RGB con 64 tipos de azul o rojo.

La otra manera de representar un color RGB con 16 *bits*, sería asignándole cinco *bits* a cada uno de los tres colores, de esta forma quedarían balanceados sin que hubiera algún color que predominara sobre los demás. Pero esto implica que queda un *bit* sobrando, por lo que se define ese *bit* como el *bit* de transparencia (*alpha*), donde si vale uno el píxel es transparente y si vale cero el píxel tiene el color que esté representado por los otros 15 bits, o si no también podría ser usado para definir la intensidad, si es uno el color es más brillante y si es cero el color es más opaco.

Para representar un color por medio de 24 *bits* (llamado *True color*) es prácticamente la misma idea, solo que en este caso se utilizan ocho *bits* (un *byte*) para representar cada uno de los colores primarios, es decir, ocho para el rojo, ocho para el verde y ocho para el azul (8 + 8 + 8 = 24). De esta forma se tienen 256 tipos de rojos, verdes y azules, que combinados dan un total de 16,777,216 colores diferentes (2^{24}).

Finalmente, para representar un color por medio de 32 *bits* (llamado *Ultra true color*) se utilizan ocho *bits* para el rojo, ocho *bits* para el verde, ocho *bits* para el azul y ocho *bits* para el color alpha (transparencia o intensidad), con este modo es posible generar 4,294,967,296 colores diferentes (2^{32}).

Es necesario mencionar que mientras mayor sea la profundidad de colores utilizada, también es mayor la capacidad necesaria para almacenar las imágenes, es por dicha razón que se debe analizar cada opción para encontrar la que mejor se acomode a las necesidades del sistema.

3.14. Creación de sprites.

Una vez que se cuenta con el arte conceptual (*sketches*), es necesario convertirlo en *sprites* (un *sprite* es un objeto multicolor dibujado en una cuadrícula rectangular). Para esto, es posible utilizar un scanner para la captura y luego, a través de un programa editor de imágenes, retocar, aplicar los colores deseados y finalmente, reducir su tamaño (generalmente los tamaños de los sprites son de 32×32, 32×16 o 16×16 píxeles).

El proceso anterior se repite para cada posición y gesto que el personaje utilice en el videojuego, al final se cuenta con un archivo llamado *sprite sheet* donde se tienen todas las facetas del personaje.

3.15. Mapeo por celdas.

Desde los primeros juegos que aparecieron a principios de los 80s, la técnica más utilizada para definir el mapa de un juego es construirlo en base a celdas, esta es una técnica muy flexible que permite editarlo y desplegarlo de una manera fácil y eficiente.

Es ineficiente tener que dibujar en una sola imagen el mapa completo de cada nivel que haya en el videojuego, sobre todo en los juegos de hoy en día donde los mundos son tan amplios, se tomaría mucho tiempo el crearlos además de que ocuparían una cantidad enorme de memoria.

Para evitar el hacer esto, se forma el mapa en base a un tablero lleno de celdas de la misma forma y tamaño colocadas unas junto a las otras, y donde cada una de ellas puede contener una imagen que represente, por ejemplo, un tipo de suelo.

Utilizando esta técnica, simplemente se repite la imagen en las partes del mapa donde se necesite que haya ese tipo de suelo, además, de que si por alguna razón es necesario volver a editar cierta parte del mapa solamente se tendría que cambiar la imagen de las celdas que están en esa zona. A cada una de las imágenes que se pueden poner en las diferentes celdas del mapa se les conoce como *tile* (panel).

3.16. *Double buffering*.

Esta es una de las técnicas más usadas, no solo en videojuegos, sino que también se usa en algunas películas y videos musicales, y en si, en cualquier lugar donde se encuentre alguna animación; y es que usar esta técnica puede es muy útil para que las animaciones que se dibujen en pantalla se vean con mucha mayor fluidez.

Para poder explicar a detalle en qué consiste esta técnica primero se tienen que explicar algunos detalles de cómo la tarjeta de video dibuja en pantalla.

- *Tiempo de refresco*: Cualquier monitor, tanto los de rayos catódicos (*CRT*), como los de cristal líquido (*LCD*) no dibujan la imagen instantáneamente de un momento para el otro, la imagen que se ve en pantalla es dibujada píxel por píxel desde la esquina superior izquierda hasta la esquina inferior derecha; sin embargo, se hace tan rápido que el ojo humano no lo nota.

  En el caso de los monitores *CRT* que tienen la mayoría de las computadoras y televisiones, hay un rayo que va dibujando en la pantalla cada uno de los píxeles que correspondan coloreándolos con el color *RGB* que le corresponde, dibujando renglón por renglón, de izquierda a derecha hasta llegar al último píxel, y volviendo a comenzar.

  Como se tiene solamente un rayo dibujando en el monitor sería lógico pensar que cuando se viera la pantalla lo único que se podría ver sería un punto que cambia de colores moviéndose por toda la pantalla, pero en realidad lo que pasa es que la pantalla que se ve, no es más que una especie de manta de un material especial, que al momento de alguna de sus partes es iluminado por el rayo absorbe el color y lo conserva al menos durante el tiempo suficiente como para que alcance a regresar el rayo y lo vuelva a iluminar. Al tiempo que tarda el monitor en dibujar en la pantalla desde el primero de los píxeles hasta el último se le conoce como tiempo de refresco.

- *Memoria de video*: Cada tarjeta de video tiene su propia memoria que se suele conocer como *Video RAM* o *VRAM* (la cual es completamente diferente a la memoria del sistema, *RAM*) y es aquí en donde se almacena todo lo que se va a dibujar en pantalla, de hecho, de aquí es de donde el monitor lee todo lo que va desplegar. Obviamente, entre más *VRAM* tenga la computadora (o más específicamente la tarjeta de video), es mejor; se podrán lograr mejores resoluciones, y utilizar una mayor profundidad de color. En concreto, cuando se quiera dibujar algo en el monitor lo que se hace es poner en la memoria de video la imagen que se quiere que aparezca en la pantalla, para que así durante el tiempo de refresco el monitor lea lo que se guardó y lo pueda poner en la pantalla.

  Si se desea que se vea alguna animación lo único que se tiene que hacer es dibujar la imagen deseada después borrarla y luego volver a dibujar la nueva imagen en una nueva posición, y así continuamente hasta que acabe la animación (tal y como en las películas se va poniendo una imagen tras otra).

Existe un gran problema, por el cual es necesario el *double buffering*, y es qué pasaría si a la mitad del refrescado se cambiara la imagen en la memoria de video; si se toma en cuenta que en una animación están cambiando constantemente las imágenes, es casi seguro que se presentará dicho problema, y lo que la persona que esté jugando va a ver en pantalla es una animación con imágenes traslapadas entre sí y con un parpadeo horrible.

El *double buffering*, como su nombre lo indica, utiliza dos búferes (espacios de memoria), por un lado se tiene la memoria de video (*VRAM*) que es el búfer donde hasta ahora siempre se había estado escribiendo, y por el otro se utiliza un búfer (del mismo tamaño que la *VRAM*) creado en la memoria del sistema (*RAM*).

La solución al problema anteriormente mencionado, consiste simplemente en dibujar todo en el búfer del sistema en lugar de en la memoria

de video (esto incluye borrar lo que había antes y volver a dibujar el personaje en la nueva posición) y cuando se haya terminado, simplemente se copia al búfer en *VRAM*.

3.17. *Dirty rectangles*.

La técnica de *dirty rectangles* (rectángulos sucios) es muy parecida a la de *double buffering*, de hecho surge de la necesidad de optimizarla (aunque no siempre es recomendable utilizarla).

En un juego relativamente sencillo quizás no se note mucho la diferencia, pero mientras el juego se va haciendo más y más grande y por consiguiente, complejo, el programa va a empezar a absorber más el tiempo de CPU, así es que se tiene que buscar nuevas formas de hacer más rápida la ejecución del programa, y una de esas formas es precisamente la técnica que se conoce como *dirty rectangles*.

Por ejemplo, en un juego que esta corriendo en una pantalla con resolución de 800×600 y con una profundidad de color de 16 *bits*, lo único que esta en movimiento es el personaje caminando por la parte superior izquierda y una pequeña explosión en la parte inferior de la pantalla. Utilizando *double buffering* se estarían copiando 480,000 píxeles (800×600) de 16 *bits* cada uno (2 *bytes*), desde la memoria del sistema a la memoria de video, es decir, 960,000 *bytes* (480,000 × 2). De esta situación nace el punto de que no es necesario copiar todo el búfer a *VRAM* cuando lo único que se está moviendo en la pantalla es el personaje y una explosión.

En la técnica de *dirty rectangles*, lo único que se hace es dibujar en la pantalla las partes que están cambiando, en el caso del ejemplo anterior, se copia del búfer el área de un rectángulo que rodeé al personaje (incluyendo el área donde se encontraba antes) y se pega justo en las mismas coordenadas en la memoria de video, aplicando lo mismo para el caso de la explosión, de manera, que en lugar de tener que sobrescribir toda la pantalla, solamente se cambian las partes necesarias.

La técnica *dirty rectangles* es conveniente utilizarla solamente cuando se puede saber con exactitud cuáles son las partes que están cambiando dentro del área del juego y se sabe que proporcionalmente resultaría más eficiente copiar solamente los rectángulos de esas partes que toda el área de la imagen.

3.18. Animaciones.

Una animación, al igual que en las películas, no es más que una secuencia de imágenes que se repiten constantemente una tras otra creando el efecto de que la imagen se está moviendo. Se puede decir que en un videojuego se tienen tres tipos de objetos en la pantalla:

- Objetos que basta con solo una imagen para ser desplegados en pantalla, como sería el caso de muebles, árboles, paredes, etc.
- Objetos que despliegan solamente un tipo de animación, como sería el caso de una antorcha, una fuente de agua, arbustos moviéndose por el viento, etc. En este tipo de animaciones lo único que se hace es implementar una función que cuente el tiempo para cambiar a la siguiente imagen, y cuando llegue al final de la animación reinicie con la primera imagen y así mantenerse en un ciclo constante.
- Objetos que dentro del juego llegan a utilizar más de una animación, como sería el caso del algún personaje (que tiene animaciones de frente, por detrás y por los lados). En este tipo de animaciones lo que se hace es que se crea un arreglo que mantenga el orden en el que se van a desplegar cada una de las imágenes durante la animación, además de una variable que contenga la imagen que se esta desplegando. La animación se cambia dependiendo de eventos, es decir, cuando el jugador presione alguna tecla, botón, o cuando muera, brinque, etc.

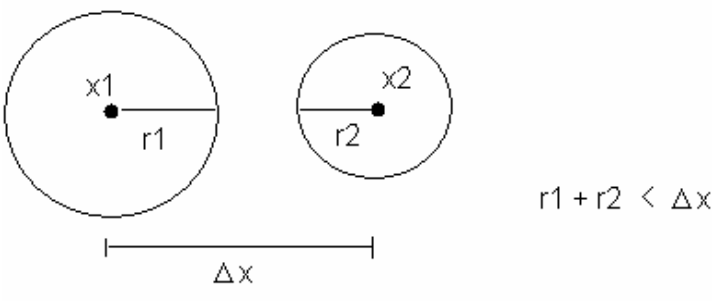
3.19. Detección de colisiones.

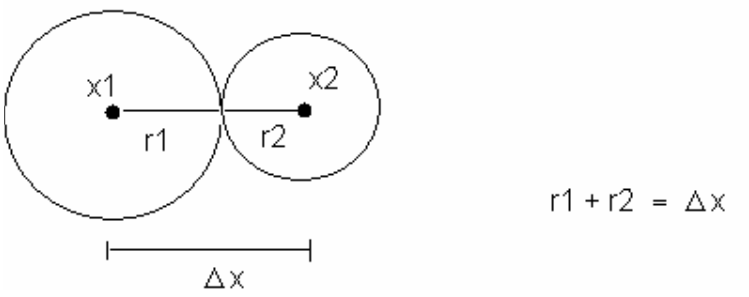
La base de la detección de colisiones es hacer que cuando los diferentes objetos que actúan en la pantalla se juntan lo suficiente entre sí parezca que están chocando y puedan reaccionar de alguna manera. La forma en la que van a reaccionar y como manejarlo depende mucho de qué objetos se traten y cómo sea el juego que se este haciendo.

Los juegos 2D tienen la característica de que la perspectiva desde la cual se desarrolla todo el juego siempre es desde arriba o desde un lado, todo el juego se lleva en 2 ejes, por lo que la detección de colisiones es sencilla.

Existen varias formas de representar el área de colisión de los objetos en un videojuego, las dos más usadas son por medio de círculos o por medio de rectángulos.

- *Círculos*: Para entender cómo saber cuándo los objetos están o no están ocupando la misma área se analiza el siguiente caso, donde se tienen dos imágenes, uno en el que las áreas de dos objetos están separadas y otra en la que se encuentran juntas provocando una colisión.





Δx simboliza la diferencia de distancias que hay entre los centros de cada objeto (x1 y x2), mientras que r1 y r2 señalan los radios de cada círculo. Como se puede observar, si Δx es igual o menor a la suma de ambos radios entonces quiere decir que existe una colisión.

Cabe mencionar que para encontrar la distancia entre dos puntos en un plano bidimensional se utiliza el teorema de Pitágoras, el cual nos dice que teniendo dos puntos (x1, y1) y (x2, y2), entonces:

$$\text{distancia} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Esto tiene un pequeño problema: computacionalmente sacar la raíz cuadrada de un número puede llegar a utilizar hasta 70 ciclos de CPU, y si a eso se le agrega que se están elevando dos números al cuadrado, seguramente el desempeño del juego bajará enormemente. Es por esto que se utiliza un algoritmo más eficiente:

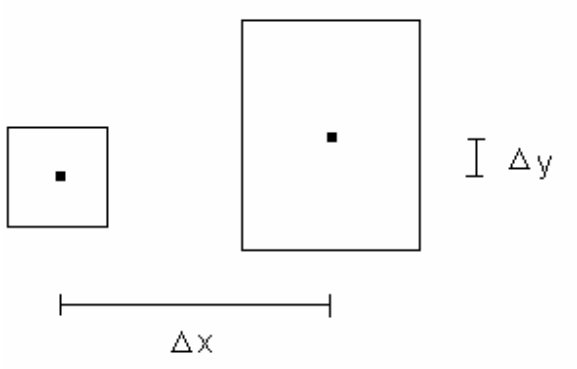
```
distancia(x1,y1,x2,y2)
      dX=|x2-x1|
      dY=|y2-y1|
      min= minimo(dX,dY)
      resultado= dX + dY – min/2 –min/4 + min/8
      return resultado
```

Este algoritmo esta basado en la Serie de McLaurin, que es un método numérico que sirve para calcular la distancia entre dos puntos mediante derivadas y aproximaciones matemáticas, el resultado no es totalmente exacto, sin embargo desde el punto de vista de un videojuego la aproximación es bastante buena, tiene una precisión del 97%.

- *Rectángulos*: En muchos casos puede resultar práctico utilizar círculos como área de colisión, pero también hay otras ocasiones en las que quizás no resulte tan adecuado, por ejemplo en un juego en el que tuvieran que interactuar de alguna forma dos personas, si se

trataran de unir lo más posible las áreas de colisión de ambos objetos chocarían incluso antes de que las imágenes de las dos personas pudieran llegar a estar cerca.

Las bases para detectar si hay colisión en un rectángulo son las mismas que la de los círculos, simplemente se calcula la diferencia de posiciones que hay entre los centros de los dos objetos al igual que como se hizo anteriormente, solo que en esta ocasión se calcula para el eje X y para el eje Y por separado, y después solamente se comparan con sus respectivos "radios", es decir, la distancia del centro del objeto al borde del rectángulo.



Así pues, la estructura del objeto solamente cambia en que se usan dos variables para guardar el "radio" del objeto, una en el eje X y otra en el eje Y.

En este capitulo se presentó una metodología para diseñar y desarrollar un videojuego, desde cómo formular y modelar la idea que se convertirá en la base del diseño de los escenarios, personajes, historia y demás componentes de la parte conceptual de un videojuego, hasta algunos algoritmos utilizados en la programación de las diferentes partes de un videojuego.

4. ANÁLISIS.

Para el desarrollo de un videojuego es necesario que primero se haga un análisis de los requerimientos y opciones que se tienen, con el objeto de lograr el objetivo de la mejor manera posible.

4.1. Estilo.

De entre la miríada de opciones que se tienen disponibles, se eligió el estilo de RPG, esta decisión se tomó ya que éste estilo de juegos presenta el mayor reto en cuanto a su análisis, planeación, diseño conceptual, y diseño técnico, lo cual nos permitirá aplicar una amplia gama de técnicas, tanto las necesarias para encauzar el proceso creativo, como las utilizadas dentro de la ingeniería de software.

4.2. Dispositivo.

El proyecto contempló como una de sus primeras etapas, una investigación y un estudio comparativo entre dispositivos, para seleccionar al que mejor se adaptase a las características del proyecto. Después de una exhaustiva investigación se decidió implementar en dispositivos Palm, ya que según estadísticas, se estima que este tipo de dispositivos controlan alrededor del 80% del mercado de las PDAs, esto gracias a su relación costo – capacidades.

El factor decisivo fue el tipo de equipo con que se cuenta –Palm- por lo cual la tecnología empleada deberá ser soportada por esta plataforma. Una vez seleccionado el tipo de dispositivo se inició la investigación técnica, la que permitirá conocer y dominar la tecnología a emplear.

4.3. Lenguaje.

Existen varios lenguajes que permiten el desarrollo de aplicaciones para el tipo de dispositivos que se contemplaron.

- Java.
- C++.
- Nativo de Palm.

Hay varias características que nos inclinaron hacia la opción de Java:

- En la actualidad existen una gran diversidad de sistemas operativos para los dispositivos móviles, por lo que la portabilidad de un lenguaje de programación es un factor muy importante, dicha portabilidad es encontrada en Java, gracias a su maquina virtual para dispositivos móviles (*KVM*, *Kilobyte Virtual Machine*).
- Java cuenta con un excelente manejador de errores.
- Finalmente, la curva de aprendizaje para el lenguaje Java es menor, ya que el equipo de desarrollo cuenta con experiencia previa en el uso del mencionado lenguaje.

Al seleccionar Java fue obligado el hecho de que se manejaría un paradigma de programación orientado a objetos, ya que este lenguaje trabaja sólo bajo dicho paradigma.

Después de la comparación se desarrollaron una serie de aplicaciones sencillas y “demos” con la finalidad de adquirir experiencia que permitiera un mejor diseño desde el punto de vista técnico. A la par, se inició el diseño conceptual del videojuego, es decir, la historia, interfaz, personajes, etc., que fueron discutidos junto con el plan de trabajo hasta elegir el más conveniente y el que mejor se adaptara al tiempo estimado del proyecto (6 meses).

4.4. Técnicas de modelado de software.

Para la óptima implementación del sistema consideramos necesario utilizar una metodología de modelado, es por esto que se decidió utilizar el Lenguaje Unificado de Modelado (*UML*, *Unified Modeling Language*) el cual nos permitirá tener una correcta documentación y un sencillo método de seguimiento del desarrollo del sistema.

Dentro de los modelos ofrecidos por UML se tiene contemplado utilizar los llamados: Diagrama de flujo, diagrama de clases, y diagrama de casos de uso.

Además se acordarán las convenciones de programación y desarrollo, como son los nombres de las clases, variables, jerarquía de carpetas, idioma de programación, e idioma de comentarios.

Cabe mencionar que cada una de las etapas descritas anteriormente quedará documentada con el fin de que esta experiencia sirva al equipo y a otros desarrolladores como referencia del tema.

4.5. API.

Al haber seleccionado Java como el lenguaje de desarrollo, era consecuente que se utilizaría *MIDP* (*Mobile Information Device Profile*), ya que es un paquete que define APIs para el desarrollo de aplicaciones para dispositivos móviles, entre las cuales se incluyen aquellas para el soporte de interfaz de usuarios, soporte de comunicaciones, soporte para el desarrollo de videojuegos, además de algunas clases como las de manejo de excepciones.

Anteriormente, la versión más utilizada era MIDP 1.0, sin embargo, su más reciente versión, MIDP 2.0, empieza a ser cada vez más común, esto es gracias a sus características como son:

- APIs para la programación de juegos.
- Componentes gráficos personalizables.
- Soporte para sonido.
- Soporte para utilizar las funciones propias del dispositivo, por ejemplo, la vibración de los celulares.
- Protocolos de redes.
- Modelos de seguridad.

De las características arriba mencionadas, la de mayor peso para utilizar esta versión, fue la del soporte para la programación de juegos, ya que MIDP 2.0

ofrece una serie de clases para el desarrollo de juegos avanzados, dichas clases son optimizadas por los fabricantes de los dispositivos al utilizar código nativo. Las clases más importantes dentro de esta API son: *Sprite*, *TiledLayer* y *GameCanvas*.

La clase *Sprite* maneja a los elementos animados dentro del juego, dichos elementos están formados por varios cuadros (*frames*), ya sean secuenciales o en un una secuencia especificada por el programador. Esta clase también permite la transformación de cuadros, como son rotaciones y detección de colisiones.

La clase *TiledLayer* representa a una matriz de celdas en la cuales se pueden desplegar tiles, los cuales son imágenes simples o animadas.

Finalmente, la clase *GameCanvas* es la encargada de proporcionar la base para la interfaz de usuario del juego. Además de proveer funciones heredadas como son los comandos y eventos de entrada, provee capacidades específicas de juegos como son un búfer gráfico y la habilidad de guardar el estado de las entradas de teclado.

4.6. *Game engine* (motor de juego).

El equipo de trabajo decidió crear su propio motor de juego, ya que de esta manera se tendría una mayor comprensión de lo que son y cómo funcionan, eso sin mencionar como se relaciona cada una de las partes que componen al motor.

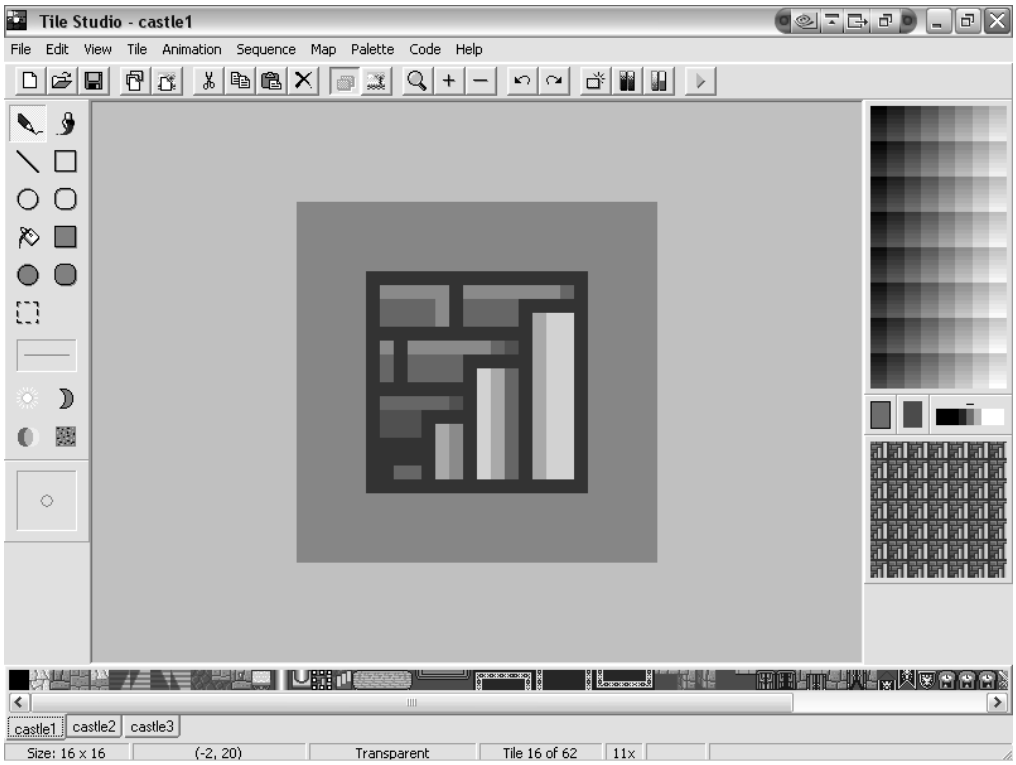
4.7. *Gameplay*.

En cuanto al *gameplay*, se decidió que el jugador empleará estrategia al enfrentar cada tipo de monstruo, es decir, ciertos tipos de ataques son más efectivos contra un tipo de monstruo que otros. Por otra parte, se tratará de que la historia no sea lineal en algunos capítulos del juego, permitiendo que el jugador elija el camino que desee tomar.

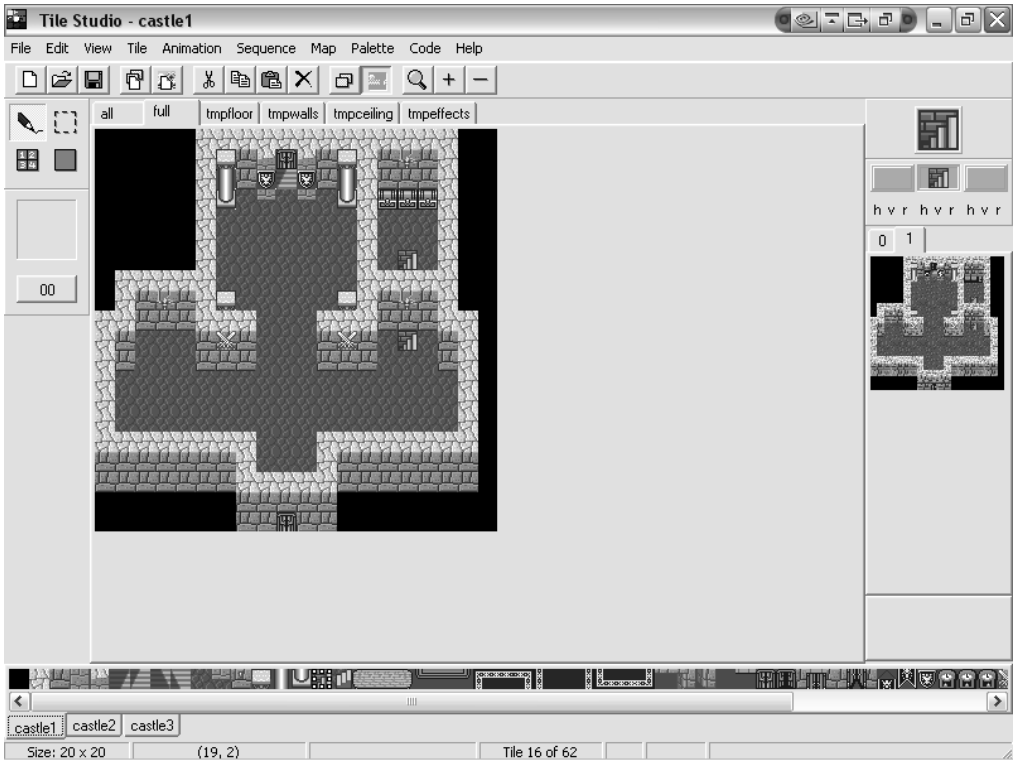
4.8. Diseño conceptual.

Para el diseño conceptual primero se decidirá un tema sobre el cual se escribirá una historia, se crearán los personajes, un breve trasfondo para cada uno de ellos, sus habilidades y sus roles dentro de la historia. Una vez estructurada y establecida dicha historia, se decidirá el estilo gráfico que tendrá el juego, con esto presente, se realizaran los *sketches* (dibujos) de los personajes y enemigos, además de los diagramas de los escenarios y objetos del mundo de juego.

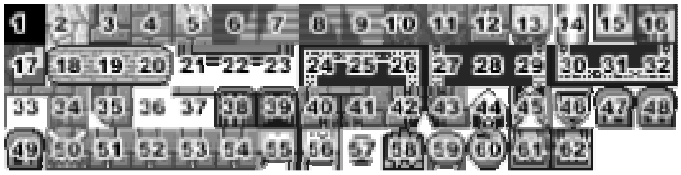
Una vez que se cuente con dichos elementos, el diseñador gráfico, los convertirá en *tiles* con la ayuda del software llamado TileStudio.



Terminados los *tiles*, se crearán los mapas al combinar varias imágenes en posiciones específicas para generar un arreglo semejante al diseño del artista.



Finalmente, se generará un arreglo de números enteros que representarán el índice de cada *tile* dentro de una imagen que contenga a todos los *tiles* utilizados en un mismo tipo de mapa.



```
full = {
{ 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,13, 3, 5,38, 5, 3,13, 2, 3,51, 3, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,14, 3,46, 7,46, 3,14, 2, 3, 3, 3, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,15,10,11,10,11,10,15, 2,61,61,61, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,10,10,10,10,10,10,10, 2,10,10,10, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,10,10,10,10,10,10,10, 2,10,10,10, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,10,10,10,10,10,10,10, 2,10,16,10, 2, 1, 1},
{ 1, 2, 2, 2, 2, 2,10,10,10,10,10,10,10, 2, 2, 2, 2, 2, 1, 1},
{ 1, 2, 3,51, 3, 2,13,10,10,10,10,10,13, 2, 3,51, 3, 2, 1, 1},
{ 2, 2, 3, 3, 3, 2, 2, 2,10,10,10, 2, 2, 2, 3, 3, 3, 2, 2, 1},
```

```
{ 2, 3,10,10,10, 3,50, 3,10,10,10, 3,50, 3,10,16,10, 3, 2, 1},
{ 2, 3,10,10,10, 3, 3, 3,10,10,10, 3, 3, 3,10,10,10, 3, 2, 1},
{ 2,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 2, 1},
{ 2,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 2, 1},
{ 2,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 2, 1},
{ 2, 2, 2, 2, 2, 2, 2, 2,10,10,10, 2, 2, 2, 2, 2, 2, 2, 2, 1},
{ 3, 3, 3, 3, 3, 3, 3, 2,10,10,10, 2, 3, 3, 3, 3, 3, 3, 3, 1},
{ 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 1},
{ 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 3, 3,38, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1}};
```

4.9. Profundidad de color.

Al tratarse de un videojuego que se desarrollará para un sistema personal, se decidió utilizar una profundidad de color de ocho bits, esto para que la carga de memoria y procesador sea la menor posible, ya que estos sistemas no cuentan con la potencia de los equipos normales de escritorio.

4.10. Sincronización.

Como se planea que el juego sea multiplataforma, es decir, que se pueda ejecutar tanto en teléfonos celulares, como en Palms o en Pocket PCs, es necesario que la sincronización sea por tiempo, ya que de esta forma no importa que *framerate* estén manejando las animaciones. La única razón en contra que tenemos al utilizar este tipo de sincronización, es que ocupa más operaciones, y por lo tanto, más recursos de procesador.

4.11. Colisiones.

MIDP2 nos ofrece la opción de detección de colisiones por rectángulos o por píxeles, por el tipo de juego a desarrollar (un RPG), conviene más utilizar la técnica de detección de colisiones por rectángulos, ya que en este tipo de juegos no es necesario que las colisiones sean muy exactas.

4.12. Optimización de código.

Como ya se ha mencionado, al desarrollar para dispositivos móviles, es de vital importancia optimizar el código, para así evitar una carga innecesaria del procesador y de la memoria. Para esto se utilizarán tipos de datos byte en lugar de enteros (siempre que sea posible), además, se tratará de utilizar corrimientos de bits en lugar de multiplicaciones y divisiones para evitar las operaciones matemáticas complejas, finalmente, se usará la técnica de *Dirty Rectangles*, para no tener que volver a dibujar la pantalla completa, sino solamente las partes que cambien.

A lo largo de este capitulo se presentaron las decisiones que se tomaron y sus respectivas justificaciones, entre ellas, las más importantes fueron:

- Implementar en dispositivos Palm, ya que según las investigaciones realizadas, dichos dispositivos controlan la mayor parte del mercado, es decir, son los más comunes.
- -El lenguaje con el que se desarrollará será Java, esto porque es portable y tiene APIs especializadas para la programación de videojuegos (MIDP2).
- Finalmente, es importante enfatizar que se utilizará una profundidad de color de 8 bits para no cargar demasiado los limitados recursos del dispositivo.

5. DESARROLLO CONCEPTUAL.

Un videojuego, en especial uno del tipo RPG, necesita un sólido marco conceptual, que permita al jugador sumergirse en la historia sin encontrar elementos discordantes. En este capítulo se describirá paso por paso esta parte del desarrollo.

5.1. ¿Dónde toma lugar el juego?

El juego comienza en el continente de Zar'an, un pacífico y hermoso lugar donde según las leyendas, los dioses comenzaron a crear al mundo. Dos reinos principales, los cuales desde el principio de los tiempos siempre han sido aliados, gobiernan justamente Zar'an. Darglia, el reino de la luz, es famoso por sus maravillas tecnológicas, ubicado en la parte este del continente es la puerta de entrada del comercio. Ric'hem, el reino mágico, es conocido como la cuna de la magia en todo el mundo, se dice que la ciudad fue construida en solamente un día utilizando poderosas fuerzas místicas.

La vida de la gente transcurre pacíficamente y sin preocupaciones en este bello continente, pero algo está a punto de desencadenar hechos terribles, es en este momento que se necesitan héroes, y la historia de estos héroes comienza aquí.

5.2. Objetivo principal.

El objetivo principal del jugador es descubrir el secreto que encierra el robo de la Llave de los Creadores.

Durante el juego, el jugador irá conociendo el pasado y motivaciones de cada uno de los personajes, así como tomará decisiones que afecten el futuro de estos.

5.3. Juegos similares.

- Saga de *Zelda* (*Nintendo*).
- *Golden Sun* (*Rare*).
- Saga *Final Fantasy* (*SquareSoft*).
- *Tales of Phantasia* (*Namco*).
- *Chrono Trigger* (*SquareSoft*).
- *Illusion of Gaia* (*Enix*).

5.4. Características.

5.4.1. CARACTERÍSTICAS GENERALES.

- Todo un continente para explorar.
- Múltiples armaduras y armas.
- Gráficos 2D con perspectiva aérea 3ra persona.
- Tiempos de carga de juego mínimos.
- Se puede guardar los avances en cualquier momento.
- Puede ser jugado hasta por dos personas en el mismo dispositivo.
- Se tienen previstas tres expansiones más, ya que en realidad esto es tan solo el principio de la historia.

5.4.2. CARACTERÍSTICAS PARA MULTIJUGADOR.

Por motivos de tiempo y recursos, en la primera etapa de desarrollo y primera liberación el juego no contendrá opción para múltiples jugadores, sin embargo se espera que en futuras versiones si se pueda incluir esta característica.

5.5. El mundo de juego.

5.5.1. SINOPSIS.

El juego se desarrolla en el continente de Zar'an, en una de las dos lunas del planeta Palathal, Mhyte. Este continente contiene diversos ecosistemas, como pestilentes pantanos, agobiantes desiertos, frondosos bosques y verdes praderas.

Esta variedad de ecosistemas ha propiciado que se genere una gran diversidad de formas de vida en todo el continente, muy diferentes a las que conocemos en la actualidad en nuestro planeta.

5.5.2. ECOSISTEMA.

- Pantanos.
- Desiertos.
- Bosques.
- Praderas.
- Volcanes.

5.5.3. FORMAS DE VIDA.

En cada uno de los ecosistemas existen formas de vida particulares, adaptadas a su entorno, en algunos casos su medio de subsistencia no es solamente físico, sino que pueden alimentarse de la magia de la misma luna, o incluso de la que producen los seres que viven o pasan cerca de ellos.

5.5.4. MAGIA Y TECNOLOGÍA.

Al ser el primer lugar creado por Los Dioses, Zar'an tiene una fuerte presencia de magia, la cual según la leyenda, es el remanente de su presencia.

Algunas culturas, como en el reino de Ric'hem, estudian y manejan esa fuerza, al punto de haberla incorporado a su vida diaria. Otros, como Darglia, creen que son tan solo supersticiones o trucos, y prefieren el uso de la tecnología, que ellos mismos desarrollan, para cubrir sus necesidades.

5.6. El mundo físico.

5.6.1. SITIOS IMPORTANTES.

- Castillo de Darglia.
- Ciudad de Darglia: Centro de desarrollo tecnológico de Zar'an.

- Mina de Darglia: Mina de hierro y otros minerales exóticos.
- Puerto de Darglia: Entrada comercial de los productos que vienen de los otros continentes.
- Edificio Real de Ingeniería (Darglia).
- Castillo de Ric'hem.
- Ciudad de Ric'hem.
- Bosque oscuro.
- Dalcidia.
- Cueva Tazer.
- Pantano Kamar.
- Mina olvidada.
- Puerto Broa.
- Isla Pirata.
- Torre de Dalzac.

5.6.2. TRANSPORTE.

Los personajes normalmente se desplazan caminando, pero en ciertas ocasiones también tienen a su disposición un barco o bote.

5.7. Personajes del juego (PJ).

5.7.1. DESCRIPCIÓN GENERAL.

Se puede seleccionar a uno de cuatro personajes iniciales, y dependiendo de dicha selección, habrá un principio de la historia diferente. Por otra parte, también hay otros dos personajes que se pueden unir al equipo más adelante, además de que algunos de los eventos que pasan en el mundo de juego, y también el final, dependerán de los personajes presentes en el equipo.

5.7.2. PERSONAJES PRINCIPALES.

Aalok.

Guardia veterano del reino de Darglia, es un combatiente honorable que lucha siempre de manera limpia. Aunque siempre se comporta de manera segura y amable con los demás realmente se encuentra un tanto frustrado por el hecho de que nunca pudo escalar en los rangos de la guardia más allá de teniente.

Rhiannon.

Joven estudiante de la academia de magia de Ric'hem, aunque apenas tiene 12 años ya es una talentosa usuaria de los poderes místicos. Inocente y alegre, siempre busca resolver los problemas de una manera pacífica, por lo que no le gusta utilizar su magia para dañar a los demás.

Vladimir.

Ingeniero del reino de Darglia. Vladimir utiliza instrumentos y partes mecánicas para crear poderosas armas. Vladimir es callado y reservado por naturaleza, desconfía de aquellos que utilizan magia a quienes llama "timadores", ya que considera a la magia como trucos baratos.

Aramil.

Joven miembro del grupo espiritual de "Los hijos de los creadores", este grupo busca reconfortar y curar cualquier mal o padecimiento que sufra la gente no sólo de Ric'hem (base del grupo), sino de cualquier parte del mundo. Aramil es un idealista que busca siempre la justicia aunque en el fondo realmente es un poco cobarde.

5.7.3. PERSONAJES SECUNDARIOS.

Los siguientes personajes se pueden unir a la aventura al avanzar en la historia.

Yozan.

Un ser misterioso que por culpa de poderes prohibidos, está condenado a vagar

eternamente por el mundo sin mas estímulo que el deseo de venganza. Callado y distante, tiene un extraña y fría aura que lo rodea siempre.

Elizabeth.

Joven alegre y de carácter aventurero, inquieta por naturaleza no puede mantenerse inactiva por un segundo. Elizabeth es una huérfana que a través de su ingenio y picardía ha logrado salir adelante en la vida, algunas de las veces de una manera no muy honesta.

5.8. Enemigos y monstruos.

Dependiendo de la zona en la que se encuentre el personaje, cambiarán los enemigos. Esto es porque cada uno de ellos tiene un tipo y un ecosistema al que pertenece.

5.9. Personajes no jugadores (PNJ).

Hay una gran variedad de personajes no jugadores, ya sean mercaderes, pueblerinos, o aquellos que otorgan misiones; además, hay un tipo especial de PNJ que es aquel que se une temporalmente al equipo y se convierte en un PJ más.

5.10. Atributos de los personajes.

Las estadísticas de cada uno de los personajes y enemigos del juego se dividen en los siguientes atributos:

- *Fuerza (Fz)*: Determina la cantidad de daño físico que un personaje o enemigo hace cuando ataca cuerpo a cuerpo.
- *Destreza (Dz)*: Determina si un personaje o enemigo es capaz de acertar el golpe cuando ataca cuerpo a cuerpo, además de que de este atributo también depende si son capaces de evadir los ataques físicos.

- *Magia (Mg)*: Determina tanto la resistencia contra conjuros mágicos así como la magnitud del efecto de los hechizos de este tipo que lanza el personaje o enemigo.
- *Espíritu (Es)*: Determina tanto la resistencia contra conjuros espirituales así como la magnitud del efecto de los hechizos de este tipo que lanza el personaje o enemigo.
- *Ataque*: Es una medida de daño físico que puede infligir un personaje o enemigo, la cual esta relacionada con el arma utilizada y la fuerza.
- *Defensa*: Es una medida de resistencia al daño físico que puede absorber un personaje o enemigo, la cual está relacionada con la armadura utilizada y la defensa base.
- *Vida*: Determina la resistencia de un personaje o enemigo para seguir luchando, cada vez que un personaje reciba daño el valor de este atributo baja hasta llegar a cero que significa que el personaje está noqueado o el enemigo ha sido vencido.
- *Energía*: Determina la cantidad de habilidades que un personaje o enemigo puede realizar, cada vez que hagan ya sea un hechizo o alguna habilidad especial, estos puntos bajan hasta llegar a cero.
- *Puntos de experiencia*: Cada vez que los personajes derroten a un enemigo consiguen una cierta cantidad de estos puntos. En el caso de los enemigos este campo no es necesario.
- *Nivel*: Cada que un personaje consiga una cierta cantidad de puntos de experiencia su nivel se incrementará en uno, lo cual trae consigo un aumento determinado en sus otros atributos, así como la posible obtención de nuevas habilidades. En general, el nivel es una medida de que tan fuerte es el personaje o el enemigo.

5.11. Habilidades de los personajes.

Aalok.

- *Corte*: Es un golpe rápido con la espada que daña al enemigo.
- *Cubrir*: Con esta habilidad Aalok es capaz de cubrir y recibir el daño físico dirigido a uno de sus compañeros.
- *Golpe de roca*: Aalok es capaz de levantar rocas con un movimiento de su espada, las cuales vuelan y golpean a varios enemigos.
- *Doble corte*: Es un golpe que combina tanto la fuerza como la velocidad impactando dos veces a un enemigo.

Rhiannon.

- *Fuego*: Ataque mágico elemental de fuego, puede golpear a uno o varios enemigos, si golpea a varios, el daño se divide entre el número de enemigos atacados.
- *Agua*: Ataque mágico elemental de agua, puede golpear a uno o varios enemigos, si golpea a varios, el daño se divide entre el número de enemigos atacados.
- *Viento*: Ataque mágico elemental de viento, puede golpear a uno o varios enemigos, si golpea a varios, el daño se divide entre el número de enemigos atacados.
- *Roca*: Ataque mágico elemental de tierra, puede golpear a uno o varios enemigos, si golpea a varios, el daño se divide entre el número de enemigos atacados.
- *Veneno*: Ataque mágico que causa un daño inicial a un enemigo y luego un pequeño daño constante a través del tiempo.
- *Ceguera*: Este hechizo puede cegar a un enemigo bajando su destreza.
- *Geiser*: Ataque mágico de fuego y agua que causa daño a todos los enemigos.

- *Ventisca*: Ataque mágico de tierra y viento que causa daño a todos los enemigos y les puede causar ceguera (efecto similar a la magia Ceguera).
- *Magma*: Ataque mágico de fuego y tierra que causa daño a todos los enemigos y puede bajarles la defensa.
- *Lluvia ácida*: Ataque mágico de agua y viento que causa daño a todos los enemigos y puede envenenarlos (efecto similar a la magia Veneno).

Vladimir.

- *Examinar*: Al utilizar esta habilidad, Vladimir es capaz de identificar cuanta vida tiene el enemigo y sus puntos débiles.
- *Bomba*: Vladimir crea y lanza una bomba que daña a un enemigo.
- *Bomba de agua*: Con este artefacto Vladimir es capaz de lanzar un chorro de agua a presión que puede dañar a uno o a todos los enemigos de manera aleatoria. El daño que hace este artefacto es del tipo elemental agua.
- *Bomba especial*: Es una habilidad parecida a la bomba, si embargo, este ataque puede dañar de manera aleatoria alguno de los atributos del enemigo.
- *Taladro*: Con este artefacto es posible dañar a un enemigo de manera considerable.

Aramil.

- *Curar*: Recupera una pequeña cantidad de vida a un personaje.
- *Antídoto*: Con este hechizo, Aramil es capaz de neutralizar el veneno que aflija a un compañero.
- *Curación*: Recupera una pequeña cantidad de vida a todos los personajes.

- *Defensa*: Aumenta por un breve periodo de tiempo la defensa de un personaje.
- *Visión*: Neutraliza el efecto de ceguera en un personaje.
- *Ralentizar*: Hace que un enemigo ataque con menor frecuencia.
- *Curar 2*: Recupera cierta cantidad de vida a un personaje.
- *Rapidez*: Hace que un personaje ataque con mayor frecuencia.
- *Curación 2*: Recupera una cierta cantidad de vida a todos los personajes.
- *Revivir*: Recupera a un personaje de estar noqueado dejándolo con una pequeña cantidad de vida.

Yozan.

- *Inmortalidad*: Al ser noqueado en las batallas, Yozan es capaz de levantarse de nuevo tras algún tiempo, esto sin necesidad de alguna habilidad externa para ello.
- *Absorción*: Esta habilidad le roba la vida, haciéndole daño, a un enemigo para dársela a Yozan quien a su vez es curado.
- *Gritos en la oscuridad*: Afecta a todos los enemigos bajando de manera temporal su fuerza.
- *Cosecha oscura*: Hace daño a un enemigo y tiene cierta probabilidad de destruirlo con ese golpe.
- *Regeneración*: Le permite a Yozan el ir recuperando su vida poco a poco.
- *La marcha de los condenados*: Daña a todos los enemigos y cuenta con la posibilidad de bajarles de manera temporal la fuerza.

Elizabeth.

- *Robar*: Con esta habilidad, Elizabeth es capaz de robarle a los enemigos los objetos o dinero que tengan.

- *Primeros auxilios*: Puede curar a cualquier personaje (incluido Yozan).
- *Trampa*: Daña a un enemigo y puede inmovilizarlo por cierto tiempo.
- *Evadir*: Aumenta de manera temporal la destreza de un personaje.
- *Ataque crítico*: Puede dañar fuertemente a un enemigo.

5.12. Avance por niveles de los personajes.

Puntos de experiencia	Nivel del PJ
100	1
200	2
300	3
500	4
800	5
1300	6
2100	7
3400	8
5500	9
8900	10 (Nivel límite)

Nombre	Atributo Principal	Atributo Secundario	Vida por Nivel	Energía por Nivel	Defensa base
Aalok	Fuerza	Destreza	10	2	10
Rhiannon	Magia	Espíritu	6	5	2
Vladimir	Destreza	Fuerza	8	3	6
Aramil	Espíritu	Magia	7	4	8
Yozan	Fuerza	Magia	9	4	4
Elizabeth	Destreza	Espíritu	7	3	6

- *Atributo principal*: Cada vez que un personaje haga un nivel su atributo principal aumenta en uno.
- *Atributo secundario*: Cada dos niveles de un personaje este atributo aumenta en uno, es decir, al nivel dos gana un punto, en el nivel cuatro gana otro punto y así sucesivamente.
- *Otros atributos*: Cada tres niveles, el personaje gana un punto en el resto de los atributos, por ejemplo, Aalok al nivel tres gana un punto en magia y un punto en espíritu, luego en el nivel 6 gana otro punto en magia y otro en espíritu y así sucesivamente.
- *Vida ganada por nivel*: En cada nivel que alcance un personaje, su máximo de vida aumenta en el número de la tabla, por ejemplo, Vladimir en el nivel uno tiene un máximo de 8 puntos de vida, al alcanzar el nivel dos el máximo número de puntos de vida aumenta a 16 y así sucesivamente.
- *Energía ganada por nivel*: Cada ocasión que un personaje aumente de nivel, su máximo número de puntos de energía aumenta en el número indicado en la tabla, de manera que Rhiannon en el nivel uno tiene un máximo de cinco, en el nivel dos su máximo aumenta a diez y así sucesivamente.
- *Defensa base*: Cada cinco niveles, el personaje gana dos puntos de defensa base, al aumentar su resistencia.

5.13. Equipo.

La unidad monetaria en el continente de Zar'an es el *ams*.

5.13.1. ARMAS.

Aalok.

Arma	Puntos de ataque	Costo [ams]
Espada de bronce	5	100 (arma inicial)
Espada de hierro	9	500

Espada de acero	13	2000
Espada de la luz	17	5,000
Corazón de Guerrero	22	-

Rhiannon.

Arma	**Puntos de ataque**	**Costo [ams]**
Cetro de pino	2	50 (arma inicial)
Cetro de roble	4	300
Bastón de plata	6	1,600
Bastón de oro	8	3,000
Renacer del Fénix	10 + daño de fuego (30%)	-

Vladimir.

Arma	**Puntos de ataque**	**Costo [ams]**
Pistola de aire	5	200 (arma inicial)
Mosquete	9	800
Rifle	13	3000
Pistola de asalto	17	6,500
Buscadora de la Verdad	22	-

Aramil.

Arma	**Puntos de ataque**	**Costo [ams]**
Honda	4	50 (arma inicial)
Arco corto	8	400
Arco largo	12	1500
Ballesta	16	5,000
Esperanza	20	-

Yozan.

Arma	Puntos de ataque	Costo [ams]
Hoz	10	300 (arma inicial)
Hoz de plata	14	1,200
Guadaña	19	4,000
Sacrificio del Alma	25 + envenenamiento (25%)	-

Elizabeth.

Arma	Puntos de ataque	Costo [ams]
Daga	9	150 (arma inicial)
Espada corta	13	1,000
Florete	17	3,200
Sangre Indomable	22 + disminuye la defensa del enemigo (20%)	-

5.13.2. PROTECCIÓN.

	Aalok	Rhiannon	Vladimir	Aramil	Yozan	Elizabeth	Defensa	Costo [ams]
Capa	S	S	S	S	S	S	2	50
Túnica de algodón		S	S		S	S	4	200
Armadura de cuero	S			S		S	8	450
Túnica de batalla		S	S		S		8	1,800

	Aalok	Rhiannon	Vladimir	Aramil	Yozan	Elizabeth	Defensa	Costo [ams]
Armadura de bronce	S			S			12	4,800
Brazalete de La Orden		S					12	-
Cota de mallas	S		S	S	S	S	16	6,000
Armadura celestial	S			S			20	-

- S: Significa que sí lo puede utilizar.
- Los campos marcados con "-" representan un objeto que no se puede comprar.

5.13.3. POCIONES Y OBJETOS.

- *Medicina*: Recupera poca cantidad de vida. 50 ams
- *Poción*: Recupera poca cantidad de energía 100 ams
- *Elixir*: Recupera algo de vida y energía. 250 ams
- *Tratamiento*: Recupera una gran cantidad de vida. 200 ams
- *Java*: Recupera una gran cantidad de energía. 400 ams
- *Ambrosía*: Recupera toda la vida y energía. No se vende
- *Antídoto*: Cura el envenenamiento. 50 ams
- *Gotas*: Cura la ceguera. 50 ams
- *Remedio*: Cura envenenamiento y ceguera. 200 ams
- *Pluma del Fénix*: Reanima a un personaje noqueado, dejándolo con poca vida. 200 ams
- *Lágrima del Fénix*: Reanima a un personaje noqueado con 100% de vida. 10,000 ams

- *Huevo del Fénix*: Reanima a todos los personajes noqueados con 100% de vida. Especial[1]

5.14. Secuencia de Batalla.

Cada personaje y enemigo tiene una velocidad de ataque diferente, mientras mayor sea el número, más veloz es. Cada turno se escoge el número más alto, ese personaje ataca, y su contador vuelve a cero, mientras que a los demás personajes se les suma nuevamente su contador de velocidad, y al turno siguiente se vuelve a repetir el proceso.

5.15. Huída.

Cuando los personajes eligen huir de la batalla, todos salen de ésta; sin embargo, de manera aleatoria (dependiendo de qué tan herido esté, qué tan difícil sea el enemigo, y si ya no tiene magia) Aramil "huirá" sólo, o se esconderá, de tal manera que el usuario ya no podrá controlarlo durante el resto de esa batalla.

5.16. Cálculos de batalla.

Hay defensa física (que depende de la destreza del personaje) y defensa mágica (depende de su magia), pero la defensa mágica puede ser de varios tipos: contra agua, curación, fuego, viento, roca, ceguera y veneno.

5.16.1. ATAQUE.

Para determinar la magnitud del ataque se usará la siguiente fórmula:

```
Ataque = fuerza + modificador del arma + modificador mágico
```

[1] Solo hay uno en el juego.

5.16.2. DEFENSA.

Para determinar la magnitud de la defensa se usará la siguiente fórmula:

```
Defensa = defensa base + modificador de armadura + modificador
mágico
```

5.16.3. ACIERTO FÍSICO.

Para ver si se acierta un golpe, tomaremos la siguiente fórmula:

```
número aleatorio positivo > destreza del agredido – destreza del
agresor
```

Si se cumple la desigualdad, el golpe acierta.

5.16.4. DAÑO FÍSICO.

Para determinar los puntos de vida que se restarán al acertar un golpe:

```
Daño = ataque del agresor - defensa del agredido
```

5.16.5. ACIERTO MÁGICO.

Para determinar si una magia puede afectar a un personaje se utilizará:

```
número aleatorio positivo > magia del agredido – magia del agresor
```

Si se cumple la desigualdad, la magia funciona.

5.16.6. DAÑO MÁGICO.

Para determinar los puntos de vida que se quitarán al hacer una magia:

```
½ (daño máximo que puede hacer la magia) < número aleatorio <= daño
máximo
Daño = número aleatorio * porcentaje de vulnerabilidad a ese tipo de
magia
```

En este caso, con una vulnerabilidad de 100%, el daño sería igual al número aleatorio.

Es vital resaltar la importancia de un buen desarrollo conceptual ya que este es la base sobre la que se planificará y desarrollará el videojuego, es decir, si el desarrollo conceptual esta bien planteado, no existirán problemas ni confusiones al momento de implementar el videojuego.

6. DESARROLLO TÉCNICO.

Una vez que se tiene el desarrollo conceptual, este se toma para analizarlo y prepararlo para su implementación. Dicho análisis es el desarrollo técnico, el cual se trata de tomar lo general y dividirlo en pequeños módulos que sean fáciles de abordar.

Para lograrlo, se realizará un análisis de la forma de interacción del usuario con el juego, los casos de uso, qué tipo de pantallas y menús se considerarán necesarios, y otros detalles específicos relativos a la implementación y optimización del juego en sí.

6.1. Interfaz de Usuario.

6.1.1. DESCRIPCIÓN GENERAL.

El sistema de interacción se basará en el uso de 6 botones: 4 de dirección (arriba, abajo, izquierda y derecha) y 2 de acción (A y B). La función de los botones de acción dependerá del modo en el que esté el juego en ese momento, ya sea en el mapa, dentro del menú, en batalla, durante un diálogo, durante un cinema, o en pausa.

6.1.2. MODOS DE JUEGO.

Modo Mapa.

- Botones de dirección: Se tienen 4 direcciones: arriba, abajo, izquierda y derecha, dependiendo del botón seleccionado, el personaje caminará hacia esa dirección.
- Botones de acción: Se tienen 2 botones, A y B; la función de A será interactuar con el entorno, ya sean personajes u objetos; la de B será abrir el menú de mapa.

Modo de Menú.

- Botones de dirección: Se utilizarán las 4 direcciones para moverse entre las opciones del menú, las cuales se presentarán de 2 por línea.
- Botones de acción: La función de A será aceptar la opción, o bien, abrir el submenú; la de B será regresar al menú inmediato anterior, o cerrar el menú en el caso necesario. En el caso de que se trate del menú de batalla, el menú raíz no se cierra.

Modo de Batalla.

En este modo solo se tiene interacción mediante el menú de batalla.

Modo Diálogo.

- Botones de dirección: Se utilizan para elegir opciones cuando se presenten.
- Botones de acción: La función de A será aceptar la opción de diálogo ofrecida al personaje, y la de B será salir de la conversación.

Modo Cinema.

- Botones de dirección: No se utilizan.
- Botones de acción: La función de A y B será la de continuar con el dialogo.

Modo Pausa.

- Botones de dirección: No se utilizan.
- Botones de acción: A no se utiliza, B cancela la pausa.

6.1.3. MENÚS.

Menú inicial.

- Nuevo juego: Comienza un juego nuevo.

- Cargar juego: Carga los avances de uno de los juegos guardados por el usuario; se contará con 2 espacios.

Menú de selección de archivo.

Muestra los dos archivos disponibles para guardar los avances del juego; una vez que se escoge uno solo se podrá utilizar ese a lo largo del juego. Si se escoge un espacio ya utilizado, se deberá mostrar un mensaje diciendo que se sobrescribirá su contenido.

Menú de configuración de juego.

Muestra las opciones de configuración del juego.

- Idioma.
- Velocidad de despliegue del texto.

Menú de selección de personaje.

Muestra la pantalla de selección de personaje: nombre, fotografía (arte conceptual) y una pequeña descripción.

Menú Principal.

Estadísticas: Ver las estadísticas de los personajes.

- Nombre.
- Nivel.
- Vida.
- Energía.
- Espíritu.
- Fuerza.
- Destreza.
- Magia.
- Ataque.

- Defensa.
- Experiencia.
- Inventario: Ver, organizar y administrar los elementos del inventario.
- Equipo: Ver armaduras y armas, así como equiparlas.
- Habilidades: Ver y utilizar las habilidades propias del personaje.
- Guardar: Guarda el estado del juego en el archivo actual.
- Salir: Pregunta si se desea guardar el avance, y sale del juego al menú inicial.

Menú de Inventario.

Se divide en dos pantallas, una de ellas muestra la lista de ítems en el inventario (excepto las armas y armaduras), y la otra despliega la descripción del elemento resaltado. Al seleccionar un elemento que se pueda usar en el mapa, se despliega un submenú desde donde se elige la acción a llevar a cabo, y el destinatario de dicha acción.

- Usar.
- Elegir blanco del ítem (personaje).
- Tirar.

Menú de Equipo.

Es una pantalla dividida en dos partes, la primera muestra lo que trae equipado el personaje y sus estadísticas actuales de ataque y defensa, y la segunda, lo que se le puede equipar. Al resaltar un elemento para equipar, las estadísticas reflejarán el cambio posible. Al seleccionarlo, el elemento es equipado, y el que tenía antes pasa al inventario de equipo que puede usar el personaje actual.

Menú de Habilidades.

Se divide en dos pantallas, una de ellas muestra la lista de habilidades que tiene el personaje, y la otra despliega una breve descripción. Al seleccionar una

habilidad que se pueda usar en el mapa, se despliega un submenú desde donde se elige el destinatario de esta.

- Elegir personaje.

Menú de Batalla.

- Atacar.
- Usar ítem.
- Selecciona el ítem.
- Selecciona el destinatario.
- Usar habilidad.
- Selecciona la habilidad.
- Selecciona el destinatario
- Pasar turno: no hace nada y deja pasar al personaje siguiente, después del cual vuelve a ser su turno. Es útil para cambiar el orden del turno de los personajes en batalla.
- Defensa: se cubre, aumentando temporalmente su defensa hasta su turno siguiente.
- Huir: huye todo el equipo.

Menú de Comercio.

Es una pantalla dividida en cuatro partes, la primera de ellas corresponde al inventario general del usuario, la segunda, al del vendedor. Ambas secciones tendrán el nombre del elemento, la cantidad, y su precio. Las otras dos secciones mostrarán una breve descripción del elemento resaltado, quienes de los personajes lo pueden usar, y cual es el efecto que tendría, en el caso de armas y armaduras. Al seleccionar un elemento, automáticamente se compra o vende, según sea el caso.

6.1.4. PANTALLAS.

Pantalla de exploración.

Se muestra el mapa, el personaje que encabeza al equipo y los objetos con que se puede interactuar.

Pantalla de Batalla.

Tiene 4 secciones:

1. Menú de batalla.
2. Pantalla con estadísticas de vida, energía y el contador de ataque, que será una barra.
3. Sección de mensajes, aquí se despliegan las descripciones de ataques, objetos y experiencia ganados y otros mensajes.
4. Zona de despliegue, es donde se muestran los personajes y monstruos distribuidos.

Pantalla de Comercio.

Aparece el menú de comercio.

Pantalla de inventario.

Aparece el menú inventario.

6.2. Catálogo de casos de uso.

6.2.1. JUEGO NUEVO.

Descripción: Este es el caso que se presenta al entrar al juego por primera vez o si el usuario eligió salir del juego.

- Aparece la pantalla de presentación del juego “La Llave de los Creadores” en desvanecimiento.
- Se muestra el Menú inicial.

6.2.2. PRIMERA VEZ EN EL JUEGO.

Descripción: Ocurre cuando el usuario eligió la opción “Nuevo juego” del menú inicial entrando por primera vez al juego, aquí se configura este.

- Aparece el menú de selección de archivo: si selecciona uno no vacío se le preguntará si desea sobrescribir los datos.
- Aparece el menú de configuración.
- Se muestra el menú de selección de personaje en una pantalla con el nombre, fotografía (arte conceptual) y una pequeña descripción de cada uno de ellos.

6.2.3. CARGAR JUEGO.

Descripción: Si el usuario eligió la opción “Cargar juego” del menú inicial, se presenta la lista de archivos que se pueden cargar. La lista depende de si hay archivos guardados, lo que divide a este caso de uso en otros dos casos.

No hay juegos guardados.

Descripción: No aparece el listado de los juegos y se muestra un mensaje indicando que no los hay y el botón de regresar.

Si hay juegos guardados.

Descripción: Se presenta la lista de archivos que se pueden cargar.

6.2.4. EN EL MAPA.

Descripción: Se presenta cuando el jugador explora una habitación, ciudad, calabozo, continente, etc.

- Puede caminar libremente por el área usando los botones de dirección.
- Si usa los botones de acción se ejecuta uno de los siguientes casos:

Usa el botón A.

Descripción: al tratarse del botón de interacción, la respuesta dependerá del actor ante el que se encuentra el personaje.

Ante ningún objeto.

Descripción: Ocurre cuando no se esta frente a un objeto con el que se pueda interactuar, como paredes, árboles, cascadas y otras texturas.

- No hay respuesta.

Ante objetos examinables.

Descripción: Se presenta ante objetos que muestran un mensaje breve, sin más interacción, por ejemplo: letreros, ventanas, cuadros, libros, etc.

- Se muestra un cuadro de dialogo con el mensaje, al presionar cualquier botón se cierra dicho mensaje.

Ante contenedores.

Descripción: Ocurre ante objetos que agregan elementos al inventario, tales como cofres, barriles, etc.

- Muestra un mensaje por cada elemento indicando qué se encontró dentro del contenedor.
- Se actualiza el inventario.

Ante puertas.

Descripción: La puerta puede verificar si se necesita una llave para abrirla, y debe buscarla en el inventario del jugador.

- Si es necesario, verificar si se cuenta con la llave.
- La puerta presenta la animación de abrirse.
- Se deja libre la zona para que el personaje pueda cruzar.

Ante personajes no jugadores (PNJ).

Descripción: Ocurre cuando se interactúa con cualquier PNJ, no importa si es mercader, misionero, etc.

- Presenta una secuencia de dialogo.
- Si es necesario se disparara un evento como abrir menú de comercio o activar misión.

Usa el botón B.

Descripción: El uso del botón B en el mapa esta restringido a desplegar la pantalla de menú principal.

Dentro de un menú.

Descripción: Se presenta cuando el jugador utiliza alguno de los botones de dirección o acción dentro de un menú.

- Los botones de dirección se utilizan para navegar entre los elementos que conforman dicho menú.
- El botón A sirve para aceptar la opción sobre la que se encuentra el cursor. Si dicha opción es una acción final se ejecutara, si es un submenú este se abrirá.
- El botón B cancela una selección en caso de que se tenga elegida, y en caso de que no se tenga nada elegido el menú se cerrara presentando su menú padre. Si ya se encuentra el menú de mayor jerarquía se cierra el menú.

Algunos menús varían el comportamiento, lo que genera los siguientes casos:

Menú Batalla.

Descripción: este menú aparece en la pantalla de batalla y el menú raíz no se cierra al presionar B.

Diálogos.

Descripción: Cuando se abre un cuadro de dialogo la interacción es como sigue:

- El botón A muestra el siguiente cuadro de dialogo si existe, o bien selecciona la opción marcada.
- El botón B cancela el dialogo.

En modo pausa.

Descripción: El jugador se encuentra en modo pausa porque regreso a la aplicación.

- No responde a los botones de dirección ni al botón A.
- Con el botón B se termina el estado de pausa.

6.2.5. INTRODUCCIÓN DEL JUEGO.

Descripción: Se muestra el mapa de Zar'an mientras se despliega la historia en un cuadro de diálogo:

El continente de Zar'an, un pacífico y hermoso lugar donde según las leyendas, los dioses comenzaron a crear al mundo. Dos reinos principales, los cuales desde el principio de los tiempos siempre han sido aliados, gobiernan justamente Zar'an. Darglia, el reino de la luz, es famoso por sus maravillas tecnológicas, ubicado en la parte este del continente es la puerta de entrada del comercio. Ric'hem, el reino mágico, es conocido como la cuna de la magia en todo el mundo, se dice que la ciudad fue construida en solamente un día utilizando poderosas fuerzas místicas.

La vida de la gente transcurre pacíficamente y sin preocupaciones en este bello continente, pero algo está a punto de desencadenar hechos terribles, es en este momento que se necesitan héroes, y la historia de estos héroes comienza aquí.

Comienza la historia dependiendo del personaje inicial seleccionado.

6.3. Especificaciones de Desarrollo.

6.3.1. CONVENCIONES.

- Nombre de las funciones, objetos, variables, clases y archivos en inglés.
- Comentarios en español.
- No se utilizarán acentos dentro de los archivos de código (ni en los comentarios).
- Las líneas se cortarán a los 80 caracteres.
- Los nombres de clase empiezan con mayúsculas y se utilizará la notación *CamelCase* (consiste en escribir la primera letra de cada palabra que componga al identificador con mayúscula).
- Los nombres de variables y funciones empiezan con minúsculas y se utilizará la notación *CamelCase*.
- El nombre de las constantes se escribirá con mayúsculas y guiones bajos separando las palabras.
- No habrá variables globales, todo se manejará mediante clases estáticas.
- Todas las variables deben tener un nombre descriptivo de su uso
- Todas las variables se declararan al inicio de las funciones o de las clases, en un solo bloque, ordenadas descendentemente primero por especificador de acceso y después por tipo de dato. Las variables contadores de los ciclos se declararan en el ciclo, no al inicio.
- Las llaves de un bloque se escriben en la línea siguiente.
- Los operadores de asignación y comparación se separaran por un espacio de sus operandos.
- La identación del código es obligatoria y se hará con tabuladores.

6.3.2. PLATAFORMA DE DESARROLLO.

Utilizaremos como ambiente de desarrollo la suite de desarrollo para Palm OS (Palm OS Developer Suite).

6.3.3. JERARQUÍA DE CARPETAS.

Tesis

- *Marco teórico*: Aquí se encuentra un historial del desarrollo del documento de tesis.
    - *Investigación previa*: Contiene las fuentes consultadas "en bruto" (páginas, tesis, documentos, libros).
    - *Resúmenes*: Contiene un resumen de lo que investigamos, si deseamos profundizar más en alguno de los contenidos podemos buscar la fuente dentro de Investigación previa.
    - *Capítulos:* Guarda los capítulos de la tesis conforme éstos se van desarrollando.
- *Proyecto*: Aquí se guardará el desarrollo del sistema.
    - *Análisis y diseño del sistema*: Diagramas, especificaciones, etc.
    - *Desarrollo*: Contiene los archivos fuente del desarrollo del proyecto.
        - *Programación*: Versiones de los archivos fuente de programación.
        - *Recursos*: Archivos fuente de generación de gráficos, sonidos y archivos externos.
            - *Gráficos*: Versiones de la evolución de los gráficos.
            - *Sonidos*: Versiones de la evolución de los sonidos.
            - *Archivos externos*: Versiones de la evolución de los archivos externos.

- *Librerías*: Versiones de las librerías externas.
- *Compilación*: Jerarquía de archivos necesaria para la compilación del proyecto

o *Producto*: Contiene solo el proyecto terminado (archivos ejecutables)

- *Juego final*: Contiene la liberación más reciente del proyecto, con las limitaciones controladas.

6.3.4. CONTROL DE VERSIONES.

Se utilizará *CVS*[1] para llevar el control de las versiones.

6.3.5. DOCUMENTACIÓN.

La documentación de código se hará de acuerdo al siguiente modelo:

Documentación de diseño:

Archivo:

```
/*************************************************************
 * ARCHIVO:    nombreDeArchivo
 * DESCRIPCION: Breve descripcion del archivo.
 * CLASES:     Listado de las clases contenidas, con historial
 *             Nombre
 *             ------
 * MODIFICACIONES:
 *  Quien Fecha       Codigo afectado
 *  ----- --------    ---------------
 *  sdrb  aa/mm/dd    nombreDeLoModificado
*************************************************************/
```

1 *Concurrent Versioning System*: Implementa un sistema de control de versiones manteniendo el registro de todo el trabajo y los cambios en la implementación de un proyecto y permite que distintos desarrolladores colaboren.

Clases:

```
/*****************************************************************
 * NOMBRE:           NombreDeLaClase
 * DESCRIPCION:Breve descripcion de la clase.
 * INTERFAZ:   Listado de interfaces que utiliza.
 * HERENCIA:   Listado de clases de las que hereda.
 * MODIFICACIONES:
 *  Quien Fecha       Descripcion
 *  ----- --------    -----------
 *  alxs  aa/mm/dd    nombreDeLoModificado
 *****************************************************************/
```

Funciones:

```
/*****************************************************************
 * NOMBRE:           nombreDeLaFuncion
 * DESCRIPCION:Breve descripcion de la funcion.
 * PARAMETROS:
 *  Tipo             Nombre           Descripción
 *  ----             ------           -----------
 *
 * REGRESA:          tipoDato    Descripcion
 * MODIFICACIONES:
 *  Quien Fecha       Descripcion
 *  ----- --------    -----------
 *  man   aa/mm/dd    nombreDeLoModificado
 *****************************************************************/
```

Constantes públicas:

```
/*****************************************************************
 * NOMBRE:           NOMBRE_DE_LA_CONSTANTE
 * DESCRIPCION:Breve descripcion de la constante.
 * TIPO:       Tipo de dato.
 *****************************************************************/
```

Documentación de implementación:

1. Variables:

```
variable  // Esta descripción de variable es muy larga, por lo que
          // se necesitan dos lineas
```

2. Lineas de codigo:

```
// La siguiente linea hace algo muy complejo, por lo
// que necesita documentación.
variable=funcion(x,y,z);
```

3. Estructuras de control de flujo:

```
// Este ciclo hace algo muchas veces controlado por la expresión
// que verifica algo, con alguna finalidad.
for(int i=0;i<control;i++)

// Este if verifica algo
if(expresion)
```

6.4. Mapa.

Cada mapa esta compuesto de 4 pantallas (ver apartado “Medidas”). Cada pantalla tiene asociado un tipo de escena, el cual servirá para mostrar el tipo de fondo en el modo batalla.

6.4.1. ENCUENTROS ALEATORIOS.

Densidad de encuentros.

Existirá un hilo que periódicamente calcule, a través de un número aleatorio, si se va a presentar un encuentro. La densidad de encuentros estará controlada con este periodo, en zonas muy densas el periodo será corto, en zonas sin encuentros el periodo será infinito o cero. Cada mapa especificará su propio periodo y si este depende del estado de la historia.

Tipo de encuentros.

Cada mapa especificara el tipo de enemigos que se presentaran en batalla, dependiendo del estado de la historia, que estará indicado por las banderas. El mapa indica qué banderas revisar y en base al estado de éstas indica que tipo de monstruos pueden presentarse.

6.5. Motor de batalla.

6.5.1. CÓDIGO DE LAS MAGIAS.

Los efectos que aplique el motor de batalla serán controlados por éste código. Es una variable entera de 32 bits, donde cada uno indica lo siguiente:

Bit	Efecto	Descripción
0	Físico.	Es un ataque físico.
1	Mágico.	Es un ataque mágico.
2	Fuego.	
3	Agua.	
4	Tierra.	
5	Viento.	
6	Veneno.	
7	Velocidad.	
8	Fuerza.	
9	Destreza.	
10	Magia.	
11	Espíritu.	
12	Vida.	
13	Energía.	
14	Nivel.	
15	Daño directo.	
16 - 31	N/A	

- N/A: No asignado.

6.5.2. APLICACIÓN DE LOS ATAQUES.

Las magias con sus efectos estarán representadas por una clase, esta clase contendrá el método effect() y el código de la magia, el método será llamado por el motor de batalla al momento de aplicar una magia. Este método contiene la lógica que el usuario quiera en la magia, pero para aplicar los resultados de esta sobre el objetivo, deberá hacer llamadas al método damage() del motor de batalla.

El método damage() recibe la cantidad de daño que se quiere hacer y la propiedad que se quiere afectar. Este método, en base al código de la magia hace el cálculo de cuánto daño realmente pasa al objetivo.

6.6. Manejo de recursos.

6.6.1. MANEJO DE BANDERAS.

Registro.

Se tiene un arreglo de 256 posiciones de 64 bits (suficientes para representar 16,384 banderas).

Direccionamiento.

Se utiliza un dato de 16 bits para el direccionamiento, con el siguiente formato: Los ocho bits más significativos indican la posición en el arreglo; los ocho bits menos significativos indican el bit a revisar.

Algoritmo para convertir el direccionamiento.

1. Se guardan los 8 bits menos significativos en una variable de 8 bits. Esto proporciona la posición del bit a revisar.
2. Se hace un corrimiento a la derecha de 8 bits sobre el valor de direccionamiento.
3. Se guardan los 8 bits menos significativos en una variable de 8 bits. Esto proporciona la posición en el arreglo.

Para generar la máscara: se hace un corrimiento hacia la izquierda sobre un

dato de 64 bits con valor inicial 1, el número de veces indicado por la posición de bit a revisar.

Operaciones.

Leer valor.

Recibe como parámetros la dirección de la bandera y devuelve su valor (0 y 1). Se hace un AND entre el valor y la máscara.

Establecer valor.

Recibe como parámetros la dirección de la bandera y el valor al que se le establecerá (0 y 1). De acuerdo a eso, hace la operación necesaria: OR con la máscara para establecer un 1; AND con la máscara negada para establecer un 0.

Archivo de mnemónicos.

Este archivo contendrá la correspondencia entre el mnemónico y el valor de direccionamiento de la bandera. Su formato es el siguiente:

MNEMONICO_DE_LA_BANDERA1=valor

MNEMONICO_DE_LA_BANDERA2=valor

6.6.2. MANEJO DE OBJETOS GRÁFICOS.

Se manejan dos tipos de elementos gráficos: las texturas y los objetos.

Texturas.

Son los elementos estáticos de un ambiente, como paredes, pisos y objetos con los que no se tiene interacción. Para manejo de texturas se cuenta con cuatro niveles:

1. *Piso*: Texturas que siempre están bajo el personaje.
2. *Paredes*: Texturas contra las cuales hay detección de colisiones, es decir, están al mismo nivel que el personaje.

3. *Techo*: Texturas que siempre están sobre el personaje.
4. *Efectos ambientales*: Es la capa más cercana al jugador, presenta los efectos ambientales que afectan a toda la escena, como niebla, anochecer, efectos de iluminación, sombras, etc.

Objetos.

Son los elementos con los que se puede tener interacción o presentan animación, como los personajes, elementos como cofres, puertas, etc. Se encuentran en al nivel de la capa Paredes (es decir, hay colisión).

Imágenes de texturas.

Los archivos de texturas estarán en el formato *PNG*. Se tienen diferentes archivos para agrupar diferentes elementos de un mismo estilo, por ejemplo texturas de cuevas, exterior, texturas de personajes, objetos, etc.

Cada imagen se dividirá en tiles, cuyo tamaño puede variar de imagen a imagen, sin embargo debe ser múltiplo de la unidad mínima de imagen que se describa mas abajo.

Medidas.

Unidad mínima de imagen (*umi*).	16 x 16 píxeles.
Tile de textura.	1 x 1 umi.
Tile de objeto.	2 x 2 umi.
Tile de monstruo en batalla.	4 x 4 umi.
Pantalla.	20 x 20 umi.
Mapa.	2 x 2 pantallas.

6.7. Propuestas adicionales.

Durante el desarrollo del juego, se han tenido algunas ideas adicionales que se podrían agregar en un futuro, las dejamos expresadas con el fin de que si alguien retoma el proyecto las evalúe y quizá incorpore a su desarrollo:

- Incluir un teclado virtual.
- Dependiendo del dispositivo, aprovechar sus características (*touch screen, joystick,* botones)
- Personalizar la apariencia de los personajes al principio del juego.
- Elaborar varios módulos que continúen con la historia.
- Llegar a un juego multijugador.
- Hacer un editor de niveles.

En este capitulo se tomaron la historia, personajes y escenarios para separar el juego en partes mas pequeñas, como los modos de juego e interfaces necesarias para la interacción con el usuario. Además, se crearon las convenciones a utilizar dentro de la implementación del videojuego, se definieron las especificaciones de implementación, el manejo de recursos, y los algoritmos de direccionamiento.

7. CONCLUSIONES

Durante el desarrollo de la tesis fuimos descubriendo mucho, no sólo sobre la industria de los videojuegos, sino de la ingeniería de software en general, reafirmando conceptos conocidos y descubriendo algunos nuevos. Aprendimos sobre algunos algoritmos empleados en el *rendereo* de animaciones, por ejemplo, el uso de *dirty rectangles* en la detección de colisiones, etc.

Además, utilizamos los conocimientos aprendidos durante la carrera, particularmente, los obtenidos en las materias:

- Métodos numéricos: Aplicación en la optimización de código, es decir, simplificación de operaciones lógicas y matemáticas.
- Estructura de datos: Uso de algoritmos de direccionamiento, tipos de datos y estructuras.
- Ingeniería de programación: Usos y elaboración de diagramas de flujo y UML.
- Lenguajes formales y autómatas: Diseño e implementación de máquinas de estado.
- Recursos y necesidades de México: Estudio de la problemática de desempleo en nuestro país.
- Seminario de ingeniería en computación: Programación orientada a objetos con Java.
- Graficación por computadora: Modularización de los videojuegos, *redereo* y APIs graficas.

Por otra parte, tomamos decisiones basadas en el análisis del problema, como el limitar los alcances de la tesis para ajustarla a nuestros recursos, tanto materiales (disponibilidad de dispositivos portátiles, instalación y uso de la maquina virtual de Java, dispositivos de red especializados para la descarga de la aplicación) como de tiempo (falta de coordinación de tiempo y disponibilidad de los integrantes debido a sus actividades tanto profesionales como personales).

Aprendimos a aplicar la metodología de modelado de ideas al crear una historia que serviría de base para todo el desarrollo del videojuego. Esto fue muy importante para nosotros ya que, como ingenieros, muchas veces debemos desarrollar o inventar nuevas soluciones a los problemas, y ese tema es pocas veces tocado durante nuestra estancia en la facultad.

Otra parte del trabajo de desarrollo de la tesis que nos pareció muy interesante fue la oportunidad de trabajar con gente de otras disciplinas, las cuales nos ayudaron con el diseño gráfico de los personajes, esto nos permitió conocer y practicar el proceso de digitalización de imágenes y creación de texturas, así como la forma de utilizar los productos terminados dentro de los videojuegos.

Además, planeamos y estructuramos de manera lógica la historia para así poder crear un documento que sirviera de base para la programación del videojuego, que dicho de paso, será la continuación de esta tesis. Además, fuimos capaces de crear un demo del juego, basándonos en el presente documento.

Los problemas con los que nos encontramos fueron que, al presentar la propuesta de tesis, el equipo de trabajo tenía grandes expectativas con respecto a ésta, se había planeado no solo el desarrollo del videojuego, sino también la creación de un motor de juego con el cual se podrían hacer otras dos partes de la historia que se tenían contempladas para este desarrollo, pero durante el desarrollo se presentaron muchos problemas que dieron como consecuencia que solamente se creara una demo del producto que se tenía planeado.

El primer problema con el que nos enfrentamos fue la conceptualización misma de la tesis, ya que en realidad nos faltó definir de una manera real los alcances y limitaciones del tema, lo que causó confusión entre el equipo ya que no se sabía con certeza como comenzar el desarrollo, es decir, no estábamos seguros si utilizar un *engine* ya creado (los cuales, generalmente se encontraban en versiones alfa), adecuar alguno a nuestras necesidades, o

desarrollar uno completamente nuevo. Cabe destacar que al analizar algunos ya creados nos encontramos con que no cubrían las necesidades o expectativas para el desarrollo del juego que teníamos en mente.

Cuando tomamos la decisión de crear un *engine* nuevo, el siguiente problema que se presentó fue el de separar la creación de un API con la programación del juego en sí, ya que no sabíamos desde que punto empezar a crear las clases, es decir, en varias ocasiones nos encontramos diseñando clases que correspondían al juego e incorporándolas a la API.

Dicho problema se reflejó al comenzar con el diagrama de clases, ya que siempre nos encontrábamos dando vueltas y preguntándonos cómo relacionar cada clase o cómo podríamos hacer cada método.

De manera paralela, se fue desarrollando el diseño conceptual del juego, en el que se presentó el problema del diseño gráfico, ya que ninguno de los integrantes del equipo contaba con grandes capacidades artísticas, por lo que fue necesario pedir la cooperación de un artista gráfico el cual diseñó un par de personajes y algunos enemigos para el juego. Otro problema en este rubro fue la clasificación de objetos dentro del juego; por ejemplo, se definía un tipo aldeano, con ciertos atributos, pero al avanzar el desarrollo se descubría que era necesario modificar ese tipo o que simplemente el tipo estaba de más.

Finalmente, una vez que la demo se encontraba terminada, se presentó la complicación de implementar dicha demo en un dispositivo portátil, en donde se tuvo la dificultad que la API utilizada (MIDP 2.0) al ser relativamente nueva, no era compatible con los dispositivos portátiles con los que contábamos. Luego el asunto se complico aún más al percatarnos que era necesaria una conexión Bluetooth para poder instalar el demo en los dispositivos ya mencionados, y que para subir la demo a Internet es necesaria una licencia de desarrollador de MIDlets (aplicaciones Java para dispositivos móviles).

Al tratarse de un tema muy ambicioso, el desarrollo del videojuego se truncó a la presentación de un demo, aún así, creemos que si con un pequeño equipo se pudo desarrollar un demo a este nivel, es posible que con mayores

recursos tanto humanos (multidisciplinarios) como económicos, se pueda crear una empresa capaz de competir a nivel nacional en este mercado tan grande y así contribuir, aunque sea un poco, en resolver el creciente problema del desempleo en este país, ya que por razones geográficas, económicas y tecnológicas, México presenta un alto potencial de crecimiento en la industria de desarrollo de software, el cual, debidamente sustentando, puede llegar a dar resultados como los de la India, país que es una referencia en el sector, y que exporta anualmente más de 8.000 millones de dólares en tecnología de la informática, es decir, el 22% del total de sus exportaciones[1].

En conclusión, este trabajo de tesis nos sirvió para conocer el funcionamiento interno de un *game engine,* así como la complejidad multidisciplinaria que presenta la creación de un sistema de este tipo, la necesidad de planear minuciosamente cada parte del desarrollo tanto conceptual como técnico, y finalmente pudimos experimentar con las capacidades y limitaciones de algunos dispositivos móviles (Palm Zire 72).

[1] http://www.lukor.com/ordenadores/noticias/0408/03100554.htm

8. APÉNDICE 1: GUIÓN DEL JUEGO LA LLAVE DE LOS CREADORES.

8.1. Introducción.

El continente de Zar'an, es un pacífico y hermoso lugar donde según las leyendas, los dioses comenzaron a crear al mundo. Dos reinos principales, los cuales han sido aliados desde el principio de los tiempos, gobiernan Zar'an con justicia. Darglia, el reino de la luz, famoso por sus maravillas tecnológicas, está ubicado en la parte este del continente y es la puerta de entrada del comercio. Ric'hem, el reino mágico, es conocido como la cuna de la magia en todo el mundo, y se dice que la ciudad fue construida en solamente un día utilizando poderosas fuerzas místicas.

La vida de la gente transcurre pacíficamente y sin preocupaciones en este bello continente, pero algo está a punto de desencadenar hechos terribles, es en este momento que se necesitan héroes, y la historia de estos héroes comienza aquí.

8.2. Sinopsis de la aventura.

Todo comienza una noche normal donde cada personaje se encuentra ocupado de alguna manera, sin previo aviso la ciudad del personaje es atacada por un ejercito del reino contrario, destruyendo casi toda la ciudad y robando una parte de "la llave de los creadores", un antiguo artefacto que según cuenta la leyenda posee poderes asombrosos y secretos.

Al principio del juego es posible seleccionar alguno de los cuatro personajes iniciales para seguir su historia a lo largo de este capítulo.

8.3. Capítulo 1. Encuentros.

Aalok.

Es una noche de rutina, Aalok se encuentra en las barracas de los guardias escribiendo su informe y recordando a su esposa e hija cuando una explosión se escucha a lo lejos, un guardia entra a la habitación para informar que están atacando la ciudad al parecer se trata de las fuerzas armadas del reino Ric'hem, Aalok da la orden de que la guardia se desplace a sus posiciones de ataque y que evacuen a los civiles, una vez dada la orden Aalok toma su espada y sale a la calle, acompañado de dos guardias, para defender a la ciudad.

> *Aquí es donde el jugador toma control de Aalok, explora las barracas para encontrar su arma y acompañantes. Saliendo de las barracas, y a lo largo del camino hasta la parte donde se encuentra el ejército enemigo, Aalok debe abrirse camino luchando contra algunos zombis y salvando ciudadanos.*

Mientras Aalok avanza por las calles ve que un pequeño escuadrón de zombis que se dirige sigilosamente hacia otra parte de la ciudad, además parece reconocer al líder de dicho agrupamiento, así que decide seguirlos para detenerlos, tras algunas batallas más con zombis, Aalok y sus dos compañeros llegan a la entrada del Edificio Real de Ingeniería (ERI), ahí es donde Aalok detiene a la brigada enemiga, y tras combatirla y vencerla, reta en un combate al líder del otro escuadrón.

> *Esta parte comprende la persecución del escuadrón hasta llegar a la entrada del ERI, y dos batallas, en la primer Aalok y sus dos guardias se enfrentan a los soldados (todos no muertos) del escuadrón enemigo, una vez que los derrote, se enfrentará a Jarein (el líder del escuadrón enemigo) quien matará primero a los guardias y luego vencerá a Aalok.*

Antes de irse, Jarein le pregunta al derrotado Aalok como están su esposa e

hija, tras lo cual el teniente de la guardia se desmaya. Algunos minutos después otro guardia de la ciudad llega y despierta a Aalok, al despertar el teniente encuentra que el ERI está en llamas, un ingeniero sale del edificio e informa que todavía hay gente atrapada adentro. Aalok se levanta, le encarga al guardia que cuide al ingeniero herido y entra al edificio en busca de sobrevivientes.

Aquí el jugador debe llevar a Aalok a través del edificio en llamas, esquivando las flamas y abriéndose paso a través de algunos enemigos mecánicos que cuidan el lugar.

Adentro del edificio encuentra a Vladimir inconsciente, el cual es el único sobreviviente, y lo despierta.

A partir de aquí la historia sigue como en el apartado Darglia de este capítulo

Vladimir.

Vladimir se encuentra observando la mitad de la llave de los creadores cuando un compañero ingeniero llamado Agustín lo interrumpe, Agustín se burla de Vladimir preguntándole si todavía sigue con su estúpida investigación de esa vieja cháchara. Le dice que como es posible que el joven genio de la academia pierda su tiempo observando una reliquia inservible si hay cosas más importantes que desarrollar. Vladimir hace caso omiso a estos comentarios y Agustín se retira.

Algunos momentos más tarde, se escucha una explosión a lo lejos, Vladimir se asoma a la ventana y escucha a los guardias gritar que los civiles deben refugiarse en el castillo, que esto es un ataque enemigo.

Aquí el jugador toma control de Vladimir para explorar el cuarto, encontrar sus armas, tomar la mitad de la llave y tratar de salir de la habitación.

Vladimir toma la mitad de la llave y trata de escapar, cuando un grupo de monstruos seguidos por una extraña mujer entran volando por la ventana. Los monstruos atacan a Vladimir.

Aquí el jugador toma control de Vladimir en su primera batalla contra los monstruos voladores.

Al ser derrotados los monstruos, la extraña mujer le pregunta su nombre al joven ingeniero, a lo cual Vladimir responde cortésmente y hace la misma pregunta a la mujer la cual se identifica como Johanna y le demanda la llave de los creadores, Vladimir hace caso omiso a lo que Johanna responde invocando más monstruos, el joven ingeniero al ver esto la llama timadora y escapa del cuarto no sin antes dejar una bomba en el camino que ocasiona que el cuarto explote en llamas.

Johanna sale sin un rasguño del cuarto e invoca más monstruos que manda a que persigan a Vladimir y maten a todo aquel que se interponga en el camino.

Vladimir debe abrirse camino combatiendo contra los monstruos voladores hasta llegar a la puerta principal del ERI, para lograr esto puede utilizar algunos dispositivos e inventos de otros ingenieros (accionar palancas, oprimir botones, jalar cuerdas, resolver secuencias) que se encuentra en el camino de manera que se libre de algunos monstruos sin tener que luchar.

Los monstruos persiguen al joven ingeniero matando a todo aquel que se interponga en su camino y cuando Vladimir está a punto de llegar a la salida, Johanna le corta el paso y le demanda la llave, a lo que el joven ingeniero responde enfrentándose y destruyendo a los monstruos que lo seguían.

Aquí el jugador vuelve a enfrentarse a los monstruos voladores.

Johanna, loca de furia, se lanza hacia él tomándolo por el cuello, volando con él a través de las llamas hasta el cuarto de atrás, y estrellándolo contra la pared,

dejando a Vladimir inconsciente. Agustín observa esto mientras está escondido y aprovecha el momento para salir del edificio en busca de ayuda. Momentos después, Vladimir es despertado por Aalok.

Es aquí donde Vladimir es encontrado por Aalok así que la historia continua como la que se relata en el apartado de Darglia.

DARGLIA.

Al encontrarse juntos los personajes, Aalok menciona que el edificio está a punto de derrumbarse y que tienen que salir de aquel infierno, a esto Vladimir responde que sabe de un pasaje secreto que los lleva a través de las cloacas a las afueras de la ciudad, así que tienen que buscarlo.

Aalok y Vladimir deben explorar el ERI para encontrar la entrada a las cloacas. Una vez en las cloacas Aalok y Vladimir deben enfrentarse a ratas, gelatinas, moscas y mosquitos gigantes para finalmente combatir contra un cocodrilo que les bloquea el paso para salir de ese lugar.

Al salir, se dan cuenta que el ataque ha terminado, y aparece un guardia que reclama su presencia ante el rey.

Aquí el jugador toma el control de ambos durante la exploración de Darglia hasta llegar al castillo. Al llegar a él, deben encontrar el cuarto del trono.

Al llegar al cuarto del trono el rey los interroga y descubren que el ataque era una distracción para robar la mitad de la llave de los creadores, por otra parte, Aalok empieza a recordar que hace algunos años, cuando tan sólo era un guardia común lleno de ambiciones, el se ofreció para infiltrarse en la organización criminal llamada La Serpiente, haciéndose pasar por un bandido, descubrió los planes de dicha organización y la desmanteló, al momento de la redada se enfrentó al líder, Jarein, al cual venció pero este último logró escapar

El rey da la orden de preparar una contra-ofensiva al reino de Ric'hem, Aalok debe quedarse en el castillo para proteger al rey y Vladimir puede irse a descansar hasta que el edificio real de ingeniería sea restaurado.

Aalok, no conforme y molesto por alguna razón, decide desobedecer las ordenes y salir por su cuenta a investigar y posiblemente a terminar con Jarein. Vladimir por su parte observa con cierta curiosidad al teniente de la guardia y decide seguirlo.

Aalok reúne a todos los guardias de la ciudad y les ordena que cuiden y protejan la ciudad, ya que él tiene que salir por un tiempo, es así que, al amanecer, Aalok deja la ciudad de Darglia y empieza a seguir el rastro de Jarein, sin notar que Vladimir lo sigue a la distancia, cuando de pronto, escucha ruidos de batalla en el bosque y se dirige al lugar de donde provienen.

En esta parte, el Aalok debe recorrer Darglia para encontrar 6 guardias, a los que manda a proteger al rey. Vladimir sale en busca de Aalok

Rhiannon.

Rhiannon se encuentra en su casa, cenando con sus padres, mientras platican como les fue en sus días son interrumpidos por gritos de personas en la calle, seguidos por personas que llaman a los ciudadanos a refugiarse ya que las defensas mágicas de la ciudad han caído.

Aquí el jugador toma el control de Rhiannon mientras explora su casa, para encontrar sus armas.

Los padres de Rhiannon salen llevándola de la mano para dirigirse al palacio en busca de refugio cuando ven que un grupo de monstruos mecánicos se dirigen hacia el palacio y se encuentran muy cerca de ellos. Rhiannon y sus padres se esconden en un callejón mientras el ejercito mecánico destruye y quema todo a su paso es aquí donde Rhiannon observa otro grupo de personas que están raptando a un niño, por lo que decide ayudarlo. Al principio Rhiannon enfrenta

a los secuestradores sólo con amenazas, pero estos se acercan amenazadoramente a ella, es cuando ve que no son personas, sino un grupo de no muertos que la comienzan a rodear.

Aquí el jugador toma el control de Rhiannon en una lucha contra los zombis.

Al vencerlos, la madre del niño le da las gracias y huye con su hijo, mientras tanto, los padres de Rhiannon la llaman al callejón cuando con horror, Rhiannon se da cuenta de que una sombra aparece tras ellos y los noquea, acto seguido un grupo de monstruos mecánicos la atacan.

Aquí el jugador toma el control de Rhiannon en una lucha contra los monstruos mecánicos.

Al vencer a los monstruos mecánicos, la sombra toma forma y se acerca a Rhiannon diciéndole que su hazaña es impresionante y que su fuerza vital lo alimentará por mucho tiempo, la pequeña niña corre asustada hacia el castillo.

El jugador debe abrirse camino derrotando a los monstruos mecánicos que ataquen a Rhiannon. A lo largo del camino, Rhiannon verá como los monstruos no muertos secuestran a otros niños, robándoselos a sus padres.

A medio camino, Rhiannon ve a uno de sus maestros de magia luchando contra los invasores, alegre corre hacia el sólo para mirar con horror como una sombra aparece tras de el y lo mata de un golpe, la sombra toma forma y le corta el paso a Rhiannon, la cual se dispone a pelear cuando ve que un grupo de no muertos trae a sus padres inconscientes, el sombrío personaje se ríe de sus inútiles esfuerzos y le ordena rendirse o matará a sus padres, la niña no viendo otro remedio se da por vencida y el oscuro personaje la duerme y luego se la lleva.

Al despertar descubre que tiene un extraño brazalete que no se puede quitar y que está amarrada en un campamento, un grupo de no muertos la vigila

y cuida el campamento, momentos después escucha una voz de los arbustos que le dice que no tenga miedo, que el la va a sacar de ahí, y se oye un ruido del otro lado del campamento.

De aquí en adelante, la historia se desarrolla como en el apartado Bosque.

Aramil.

La historia de Aramil comienza mientras se encuentra en el palacio de Ric'hem curando al hijo enfermo del rey, cuando un alboroto que viene del cuarto del trono hace que vaya a averiguar que está pasando, cuando llega ve que los consejeros del rey y algunos miembros de la orden de los hijos de los creadores, orden a la que él pertenece, se encuentran discutiendo, de pronto, un guardia mal herido entra al salón y cae, antes de morir dice que las fuerzas de Darglia están atacando la ciudad,

Aquí el jugador toma el control de Aramil mientras explora el castillo hasta llegar al cuarto del trono.

Al llegar al cuarto del trono, se percata de que han sido convocadas las fuerzas armadas para proteger la ciudad, y la Orden de los Hijos de los Creadores para ayudarles, es así como Aramil con otros tres acompañantes forman un batallón y se dirigen a la zona de conflicto.

El jugador debe explorar el castillo para buscar la salida.

Cuando el primer batallón sale del palacio una explosión acaba con todos ellos, un guardia grita que el enemigo ya llegó al palacio, que es necesario defenderlo, así los batallones se guarecen en el palacio para esperar al enemigo, mientras tanto, Aramil es encargado de estar en la última línea de defensa para proteger al rey.

Gritos de dolor y agonía llegan al cuarto del trono, así como el sonido de una feroz batalla que se libra en las cámaras que rodean al gran salón, en un

momento, todo queda en silencio, la gente en el salón se pregunta si la batalla habrá terminado cuando la puerta del cuarto se abre de golpe, un grupo de monstruos mecánicos entra y empieza una feroz pelea.

Aramil, junto con tres compañeros, debe enfrentarse a varias olas de enemigos, con cada nueva embestida, Aramil pierde cada vez más su coraje y se vuelve menos efectivo en las batallas.

De pronto un hombre con armadura entra por la puerta, se presenta como Dieter y exige la mitad de la llave de los creadores que tiene el rey, los soldados restantes, todos menos Aramil se lanzan al ataque, pero todos son muertos por el misterioso caballero, el cual se aproxima al rey a quien de un golpe mata y le quita la mitad de la llave, antes de irse, Dieter voltea hacia donde se encuentra el paralizado Aramil y le dice que no se preocupe, que el no mata a basura cobarde como él, tras decir esto, Dieter sale caminando del palacio.

Aramil, avergonzado por su conducta cobarde, decide seguir al extraño caballero y recuperar la llave aunque le cueste la vida, es así como empieza a seguir a Dieter por la ciudad.

Aramil debe irse escondiendo por varios callejones al seguir a Dieter, además de que de manera aleatoria será atacado por algunos monstruos mecánicos.

Es así como Aramil sigue a Dieter a lo largo de gran parte de la noche, hasta que llega a un extraño campamento en el bosque, ahí Dieter se detiene por algunos momentos, habla con los monstruos que están cuidando a una pequeña niña que se encuentra inconsciente y atada y luego se retira. Aramil decide que es mejor primero salvar la vida de la niña y espera a que esta recobre la conciencia, una vez que ve que la niña despierta, lanza una piedra al extremo opuesto del campamento.

De aquí en adelante, la historia se desarrolla como en el apartado Bosque.

BOSQUE.

El ruido provocado por Aramil distrae a los monstruos mientras éste se desliza en la oscuridad y desata a Rhiannon, una vez libre, Aramil llama a Rhiannon para que la siga y escapen del campamento.

El jugador debe explorar el bosque mientras enfrenta a zombis, esqueletos, osos, y lobos.

Rhiannon y Aramil recorren el bosque hasta el amanecer en busca de una salida cuando un grupo de no muertos los rodean, uno de ellos se adelanta hacia ellos y les dice, el maestro no está, así que no hay nadie que evite que los devoremos.

Aquí el jugador debe derrotar a un no muerto que funge como jefe de la escena.

Al derrotar al no muerto, los demás se acercan amenazadoramente a los personajes, cuando otra persona se les une, por un momento creen que se trata de un enemigo, pero la persona los ayuda contra los monstruos.

Aquí empieza una gran batalla en la que Aalok, Rhiannon y Aramil luchan desesperadamente contra los zombis.

Después de derrotarlos, otro grupo de ellos los vuelven a rodear, cuando todo parecía perdido un grupo de explosiones acaban con los enemigos y ven como del humo sale otra persona (Vladimir) que les dice que lo sigan rápido antes de que más monstruos vengan, ya que esas fueron sus últimas bombas, así es como después de presentarse, los cuatro salen del siniestro bosque.

8.4. Capítulo 2. Buscando Respuestas.

A la orilla del bosque los personajes se presentan formalmente, pero al enterarse de que pertenecen a distintos reinos empieza una serie de reclamaciones y acusaciones. Vladimir detiene las acusaciones y escucha la historia de lo que paso en Ric'hem, luego cuenta lo que paso en Darglia y como saca como conclusión que es imposible que Ric'hem atacara si sus fuerzas armadas estaban defendiendo la ciudad, para disipar las dudas de que si la historia de Rhiannon y Aramil es verdad, decide ir a Ric'hem a investigar, además de que desea saber mas sobre la mitad de la llave que tenían ahí, por otra parte, Aalok desea atrapar a los ladrones por lo que deciden dividirse en dos equipos, Aalok y Rhiannon rastrearan a los ladrones mientras que Vladimir y Aramil regresaran a Ric'hem para verificar que la historia es cierta y recopilar información sobre la mitad de la llave que estaba ahí. Deciden reunirse de nuevo en el pueblo llamado Dalcidia, al pasar una semana.

El jugador tomará el control de Vladimir y Aramil o de Aalok y Rhiannon de acuerdo a su personaje inicial.

VLADIMIR Y ARAMIL.

Ric'hem.

Una vez en la ciudad de Ric'hem, Vladimir y Aramil encuentran el pueblo casi destruido, lo que corrobora la historia de Rhiannon, además la gente que reconoce a Aramil lo insulta y lo llama cobarde.

El jugador explora Ric'hem hasta encontrar la biblioteca.

Vladimir descubre que hace varios años un viejo llamado Dalzak analizó detalladamente la mitad de la llave y escribió sus observaciones, también se entera de que Dalzak vive o vivía en una apartada torre.

El jugador explora Ric'hem hasta encontrar la salida.

Cuando los personajes se disponen a dejar la ciudad, ven que a lo lejos se

aproxima el ejército de Darglia, el cual comienza un ataque despiadado sobre Ric'hem, este último responde con ataques mágicos y así comienza la guerra entre ambos reinos.

Vladimir y Aramil deciden que para detener la guerra es necesario inutilizar las armas de ambos bandos, de manera que primero se dirigen a las afueras del pueblo para acabar con la ofensiva de Darglia.

> *Vladimir y Aramil se encuentran en medio de una guerra, así que son atacados tanto por soldados de Darglia como por soldados de Ric'hem. El jugador debe sabotear las máquinas de guerra de Darglia, para eso tiene que abrirse paso a través del campo de batalla hasta llegar a cada una de las tres máquinas, la última máquina funge como jefe, ya que ataca a los personajes.*

Luego se dirigen al calabozo del Castillo Ric'hem para inutilizar las armas mágicas. Para acceder al calabozo se esconden entre los callejones de la ciudad y entran al castillo por una entrada mágica que sólo los miembros del grupo Los Hijos de los Creadores conocen.

> *En el calabozo, el jugador se enfrentará a guardianes mágicos para finalmente llegar ante un gran cristal que Vladimir destruye con una bomba.*

Una vez que acaban con las armas de ambos ejércitos, la batalla se dispersa hasta finalmente terminar. Vladimir y Aramil, ahora considerados como traidores huyen de la ciudad y se dirigen a Dalcidia para encontrarse con sus compañeros.

AALOK Y RHIANNON.

La parte de este equipo comienza en la noche, mientras están sentados frente a una fogata, platicando sobre ellos y su pasado, Aalok solo menciona que es, o

era, teniente de la guardia de Darglia, ya que al desobedecer al rey seguramente será dado de baja de su puesto, pero que eso no le importa, Rhiannon le hace la observación de que parece no gustarle el hecho de ser teniente en la guardia a lo que el veterano guerrero sólo responde que el solo llegar a ese rango le ha costado lo mas importante de su vida. Para cambiar de tema Rhiannon menciona lo que le paso el día de la batalla, además menciona que la magia para levantar a los muertos es nigromancia y está prohibida en Ric'hem, por lo que le sorprendió ver que alguien la usara, en ese momento Rhiannon siente un extraño dolor en el brazo donde tiene el brazalete que no se puede quitar, Aalok le pregunta que le pasa y ella le responde que de pronto se sintió débil, pero que ya está bien. Aalok le dice a la pequeña que es mejor que duerma ya que en la mañana seguirán tras el rastro de los ladrones.

Mina Olvidada.

Los personajes entran a la mina donde encuentran una extraña neblina que lo cubre todo, en caso de que lleven a Yozan este les dice que el lugar esta impregnado con el olor de la muerte y el sufrimiento. Mientras recorren la mina encuentran un registro donde descubren que en el lugar trabajaban como esclavos tanto adultos como niños y que cientos murieron mientras excavaban la plata de la mina, esto terminó cuando se agotó la plata.

Aquí los personajes avanzan mientras se enfrentan a no muertos, murciélagos e insectos.

Al explorar la mina, llegan hasta un muro de huesos que no les permite seguir adelante, escuchan varias voces que les dicen que los ayuden y que los liberen, en ese momento una extraña luz aparece y un enorme capataz no muerto los ataca

El jugador se enfrenta contra el capataz no muerto.

Al vencerlo, el muro desaparece y la mina es exorcizada. Al acercarse a lo que parece ser la salida de la mina cuatro seres les cierran el camino, estos seres

de los cuales Aalok reconoce a Jarein y Rhiannon a aquel que la secuestró y le puso el brazalete, llaman a un grupo de soldados para que los maten, los personajes se enfrentan a los soldados.

Durante esta exploración ya no encuentran no muertos que los ataquen.

En esta batalla Rhiannon tiene dudas y no ataca, Aalok le dice que la necesita, que por qué no actúa, la niña responde que no quiere lastimar a nadie, que hasta ahora nunca ha usado su magia para dañar a la gente, sólo la utiliza para protegerse de monstruos.

Esta batalla se puede ganar fácilmente sin Rhiannon, la cual no actúa en ella.

Una vez derrotados los guardias uno de los cuatro seres se adelanta y les dice a los demás que continúen, que el se encargará de ellos, Aalok le pregunta quién es a lo que el ser responde diciendo que un viejo amigo de Rhiannon y que su nombre es Mephisto. En caso de que Yozan esté en el equipo este se adelanta y le dice: "tu, tu fuiste el que me hizo esto" a lo que Mephisto responde diciendo que el no lo conoce.

Los personajes son derrotados fácilmente por Mephisto, no importa lo que hagan esta pelea es parte de la historia.

Luego Mephisto se retira sellando derrumbando la salida tras el, en ese momento, la mina se empieza a derrumbar, Aalok se levanta y despierta a los demás y corren de regreso para poder escapar del lugar.

El jugador debe regresar al inicio de la mina en cierto tiempo o la mina caerá acabando con los personajes.

Al escapar de la mina, Aalok le pregunta a Rhiannon y, en el caso de estar con ellos, a Yozan si conocen a Mephisto. Rhiannon explica que el fue quien la secuestró durante la batalla donde robaron la llave, por otra parte, Yozan explica que el era un soldado de Ric'hem que murió defendiendo la ciudad en

ese ataque y que Mephisto al utilizar su magia lo volvió a la vida como un no muerto, ahora en ese estado no puede volver con su familia o con sus amigos y ya no tiene una identidad o una razón para existir, así que lo único que lo impulsa es la idea de destruir a Mephisto.

Después de eso, los personajes se dirigen a Dalcidia para reunirse con sus compañeros.

CARAVANA Y CUEVA TAZER.

Mientras este equipo se dirige a Ric'hem, encuentran una caravana que esta siendo atacada por bandidos y se lanzan para ayudar a los comerciantes.

Aquí el jugador debe derrotar a todos los ladrones para rescatar a los comerciantes.

Cuando terminan con los ladrones ven que otro grupo de ellos escapa, el líder de la caravana les pide ayuda, ya que el grupo que escapó se llevó todo su dinero.

El jugador puede escoger si ayuda o no al comerciante, en el caso en que lo ayude, Vladimir rastrea a los bandidos hasta la Cueva Tazer, ahí debe recorrerla enfrentando a murciélagos, bandidos e insectos para encontrar y vencer al líder de los bandidos y recuperar el dinero, si le regresan el dinero al comerciante, este como agradecimiento les obsequia la bomba de agua (un artefacto para obtener la habilidad de Vladimir), en caso de no regresarle el dinero y mentirle diciéndole que ya acabaron con los bandidos pero no encontraron nada ahí, el comerciante les da las gracias y les da como recompensa un poco de dinero y luego se va diciendo que debe volver a empezar a reunir sus ahorros (con esta opción es posible obtener la bomba de agua, pero deben comprarla a un gran precio).

En caso de no ayudar al comerciante el jugador puede seguir libremente su camino o ir a la Cueva Tazer, donde puede luchar

contra el líder de los bandidos y quedarse con el dinero (en este caso, como no pueden regresarle el dinero al comerciante, no obtienen la bomba de agua de manera gratuita y debe comprarla por un gran precio).

PANTANO KAMAR.

En la mañana los personajes entran al Pantano Kamar, lugar en donde en repetidas ocasiones ven a alguien cubierto por una túnica que se mueve rápidamente por lo que sólo lo pueden ver por instantes, al parecer el extraño ser los está observando. En el centro del pantano se encuentran con un extraño hongo al que derrotan para continuar adelante, pero cuando se disponían a seguir su camino, el extraño ser les corta el paso y los reta a un duelo para comprobar que tan fuertes son. Tras el último ataque, el extraño ser parece caer muerto y cuando los personajes se disponen a seguir su persecución, el ser se levanta de nuevo, se sacude y se presenta como Yozan, les pregunta si se puede unir a ellos para buscar a los que robaron la llave. Ya sea con o sin el extraño Yozan, los personajes salen del pantano y descubren que las huellas se dirigen a una antigua mina, llamada por la gente la Mina Olvidada.

En el pantano los personajes se enfrentan a monstruos tipo insecto, planta y algunas bestias. El hongo funge como jefe de este lugar y al vencerlo, el jugador tiene la opción de permitirle a Yozan que se una al equipo o rechazarlo. En caso de que acepten, Aalok demanda saber por qué Yozan sigue a los ladrones a lo que este último responde simplemente con la palabra venganza. En caso de que no acepten, Yozan se retira decepcionado y nunca lo vuelven a ver.

DALCIDIA.

Ya en Dalcidia los personajes se reúnen y se informan de lo que les sucedió, Aalok menciona que le perdieron la pista a los ladrones al derrumbarse la mina, por otra parte, Vladimir aconseja que sería fácil encontrarlos si supieran más sobre la llave y su historia por lo que recomienda que se dirijan a la Torre de Dalzak, lugar donde un hechicero hizo varias investigaciones sobre la llave. Es así como todos deciden ir hacia dicho lugar.

La Torre de Dalzak se encuentra en una pequeña isla por lo que para llegar los personajes deben encontrar un medio de transporte y deciden ir al Puerto Broa y ahí buscar una embarcación.

Para viajar los personajes deciden que un equipo pequeño es mas rápido por lo que deciden que solo vayan 4 a la vez, es decir, un personaje se quedará atrás y los alcanzará en el puerto.

PUERTO BROA.

Al llegar al puerto descubren al platicar con la gente que ninguna embarcación dejará el puerto, ya que piratas asaltan a todo aquel que se embarca. Cuando los personajes están a punto de darse por vencidos, una joven se acerca a ellos y les pregunta si son ellos quienes están buscando transporte, una vez que los personajes le responden, ella les propone un trato, si ellos la ayudan a ir por su tesoro de los piratas, ella los ayudará en su misión. No teniendo otra opción los personajes aceptan y ella los conduce hasta un pequeño bote, les dice que solo puede llevar a tres más por lo que los demás deben quedarse en el puerto.

El jugador debe escoger a tres personajes para llevar, no es posible cambiar la selección hasta que regresen al puerto.

En el camino la joven se presenta como Elizabeth y les dice que se dirigen a una pequeña isla donde los piratas tienen su guarida, además les indica que es necesario que una vez que estén en la isla se conduzcan lo más cautelosamente posible.

ISLA PIRATA.

Una vez en la isla, el jugador debe moverse por un pequeño pueblo de piratas, es necesario que se vaya ocultando, si el jugador guía a los personajes frente a algún pirata, este los atacará. Luego, al pasar por el pueblo debe entrar a una caverna donde se enfrentará a lo largo del camino con piratas para al final enfrentarse al capitán de los piratas (jefe de la escena) y conseguir el tesoro de Elizabeth.

Al recuperar el tesoro de Elizabeth que es un barco, los personajes huyen de la isla, pero los piratas los persiguen librando una feroz batalla en el mar, al vencer los personajes regresan al puerto por sus compañeros. En el camino alguno de los personajes hace la observación de que es muy extraño que los piratas los persiguieran con tanto empeño, ya que se estaban arriesgando mucho al acercarse tanto al puerto, Elizabeth contesta a esto con una risa y les dice que se arriesgaron tanto por que estaban buscando recuperar su mejor barco. Elizabeth explica que realmente utilizó a los personajes para conseguir un barco y que mejor barco que uno pirata, los personajes tratan de discutir con ella pero ella termina el problema diciendo que ladrón que roba a ladrón…

La batalla naval puede ser desarrollada como un mini juego donde los barcos se disparen, puede ser un grupo de peleas normales en el barco, o una combinación de ambos.

Ya en el puerto Elizabeth les pide que esperen un poco ya que necesita reparar el barco, y conseguir provisiones para el viaje, que descansen y que al día siguiente se vean en el puerto para zarpar.

El jugador puede utilizar este tiempo para comprar cosas en el puerto, ya que ahora que los piratas ya no son un problema, los comerciantes tienen mejores y más cosas que vender. Por otra parte, al hablar con la gente, los personajes conocen a Susana, una amiga de Elizabeth, esta les cuenta que ella y Elizabeth son amigas

desde pequeñas, que fueron huérfanas y que gracias al apoyo de Elizabeth (dinero que no sabe de donde sacó) fue capaz de poner una tienda con lo que ambas podían sobrevivir, pero que pronto Elizabeth se cansó de la vida tranquila y le dejo la tienda a ella. Les dice que Elizabeth es una persona muy inquieta y que para ella todo es una nueva aventura.

TORRE DE DALZAK.

Al otro día los personajes dejan el puerto y se dirigen a la isla donde se encuentra la Torre de Dalzak, de nuevo es necesario que sólo vayan 4 personajes.

El jugador debe escoger a cuatro personajes con los cuales avanzará por la torre enfrentándose a monstruos mágicos para llegar a la cima y luchar contra un gólem guardián que es el jefe de esta escena.

Al vencer al guardián de la torre, pueden pasar a un último cuarto donde descubren un esqueleto que yace sobre una silla, al parecer era Dalzak y murió hace ya varios años, al investigar en el cuarto los personajes encuentran algunas notas sobre la investigación de Dalzak, en ellas descubren que la llave abre la puerta de la destrucción del mundo, puerta donde los dioses creadores encerraron a todo el mal, también encuentran que para volver a unir ambas mitades de la llave es necesario que sean llevadas a un sitio especial, el nombre del sitio no es legible, ya que el tiempo y la humedad ha destruido casi todo en ese cuarto. Además en uno de los estantes encuentran una formula química que le permite a Vladimir crear las Bombas Especiales.

El sitio especial depende del personaje que el jugador haya elegido para la introducción, por otra parte, al encontrar la formula química, Vladimir obtiene la habilidad de Bomba Especial.

Cuando regresan al puerto, los personajes saben que deben detener a los

ladrones antes de que unan los fragmentos de la llave, así que se disponen a buscar el lugar donde esto debe hacerse. Por su parte, Elizabeth decide unirse a la aventura de los demás ya que según ella, eso será muy divertido.

8.5. Capítulo 3. Revelación.

Este capítulo no tiene un orden de evolución fijo, es decir, el jugador puede escoger a donde ir y cada lugar presenta un reto y algunos contienen pistas o la respuesta al enigma de lugar donde los ladrones llevaron las mitades de la llave de los creadores. A continuación se presenta una lista de los sitios importantes, así como los posibles tesoros que pueden hallar ahí.

PUEBLO MARILLY.

En este remoto y pequeño pueblo vive un anciano sabio y enfermo que ayudará a los personajes si estos encuentran la medicina que necesita, la medicina es una planta que crece en el pico del Monte Ukeret, para llegar a dicho lugar, es necesario cruzar el extraño Bosque Flammarion. Como recompensa el anciano les ofrece información, les dice que para encontrar el lugar donde se deben unir ambas mitades de la llave deben preguntarle al oráculo del Templo de Selein, el cual se encuentra escondido en las profundidades de la tierra, para llegar a él es necesario que sigan el Río Fatuh.

COLISEO DE MARILLY.

Este lugar se encuentra cerca del pueblo que lleva el mismo nombre, al llegar los personajes, se enteran de que un torneo está a punto de iniciar y que el premio en efectivo es muy tentador, en un principio los personajes no quieren participar, pero Elizabeth está deseosa de quedarse con ese dinero por lo que decide infiltrarse en la noche y tratar de robarse el dinero, es atrapada y los personajes son obligados a participar para obtener su libertad.

En este escenario, los personajes deben luchar a través de cuatro eliminatorias, cada lucha tendrá una regla específica. En la primera eliminatoria, los personajes luchan contra otros cuatro participantes, en la segunda eliminatoria lucharán uno contra uno (hasta vencer a

los cuatro integrantes del equipo contrario), en la tercera los personajes luchan contra otros cuatro participantes, pero esta vez serán atacados al inicio de la pelea con algún efecto aleatorio (veneno, ceguera, etc.), finalmente, en la última batalla, los personajes se enfrentarán cuatro contra uno, es decir, los cuatro personajes deben vencer a un luchador que funge como jefe de este escenario.

Al ganar el torneo, los personajes obtienen una gran cantidad de dinero y el arma más poderosa para Elizabeth.

BOSQUE FLAMMARION.

El Bosque Flammarion es un bosque mágico y misterioso, al entrar los personajes observan con asombro el hermoso lugar. Dentro de este bosque encuentran un hermoso lago, si en el equipo se encuentra Aalok, este se acerca y se ve reflejado en el agua, comienza recordar lo que pasó hace ya tantos años.

Se ve una mujer y una niña preparando la comida cuando un Aalok más joven entra a la casa, la mujer lo saluda con un beso y la niña corre hacia el gritando "¡Papá, qué bueno que llegaste!". Aalok se sienta a la mesa con su hija, su esposa sirve la comida y se sienta también. La esposa le pregunta a Aalok como había estado su día, a lo que el joven guardia contesta que igual que todos los demás, que desearía tener una misión más importante que solo vigilar y que desearía poder ser más que un simple guardia, la esposa le dice que no se preocupe, que ella está segura que algún día lo ascenderán y será capitán de la guardia.

La escena cambia a las barracas, donde el capitán de la guardia pide un voluntario para una peligrosa misión que el rey encargó, sin dudar ni un momento, Aalok se presenta como voluntario. El capitán le describe la misión diciéndole que un grupo criminal llamado La Serpiente, tiene aterrorizada a la

gente de Darglia, que es necesario que alguien se infiltre para encontrar su cuartel general.

Nuevamente cambia la escena y se ven que Aalok está en un cuarto con varios criminales, a la cabeza de la reunión está Jarein, se encuentran discutiendo cuanto han subido los ingresos del "negocio" cuando un fuerte golpe se escucha de fuera del cuarto, un hombre mal herido entra al cuarto y grita: "Es un redada, la guardia nos encontró", después de eso el hombre muere. Todos los criminales comienzan a salir corriendo del cuarto, Jarein furioso grita: "¿Cómo nos encontraron? ¡Un traidor, debe haber un traidor!". En ese momento Aalok saca su espada y se lanza sobre Jarein, comenzando una feroz lucha entre los dos. Aalok hiere a Jarein, pero este escapa a través de un pasaje secreto en la pared.

De nueva cuenta la escena cambia al cuarto del trono en el palacio de Darglia, aquí Aalok recibe del rey su nuevo rango de teniente de la guardia, Aalok feliz se dirige hacia su casa para encontrar a su familia, pero cuando llega queda horrorizado al encontrar tanto a su esposa como a su hija asesinadas, y en la mesa se encuentra una nota que dice:

"Felicidades por tu ascenso y por destruir mi organización, pero todo tiene un precio, espero no me olvides. Jarein."

La escena cambia de nuevo al Bosque Flammarion, Aalok se encuentra de rodillas frente al lago inmóvil y sollozando, comienza a llover y del lago aparece un hada obscura que se acerca a Aalok, el hada le dice que no sufra más, que ella lo puede reunir con sus seres amados, que lo guiará a la oscuridad donde se encuentran su esposa y su hija. Los demás personajes se adelantan y le dicen al hada que no se lo permitirán, pero el hada los paraliza con un hechizo y se sigue acercando a Aalok, cuando lo tiene a la mano el teniente de la guardia se levanta y le dice al hada que no es el momento de que se reúna con su familia, que tiene cosas que hacer, en ese momento el hechizo del hada sobre los demás termina y empieza una lucha. Al vencer al hada, en su lugar aparece un espada la cual toma Aalok, y luego siguen adelante.

En el Bosque Flammarion, los personajes se enfrentaran con criaturas de tipo mágico como son hadas, dríadas, goblins y duendes. El hada obscura funge como jefe de este lugar, la espada que encuentra Aalok es el arma más poderosa para él en este juego.

MONTE UKERT.

El Monte Ukert se encuentra en medio del Bosque Flammarion, por lo que es necesario que primero atraviesen el bosque para llegar aquí. En este escenario los personajes deben subir hasta el pico del monte, en algunos puntos del camino tendrán que esquivar enormes rocas que les lanzarán gigantes. En el pico los ataca el rey de los gigantes.

A lo largo de su camino, los personajes se enfrentarán con aves, algunas bestias, y cada vez que pasen un área en la que los gigantes les lancen rocas, se enfrentarán a dichos enemigos. El jefe de la escena es el rey de los gigantes, al vencerlo los personajes obtendrán una planta medicinal.

PUEBLO FATUH.

Este escenario comienza con los personajes llegando a una pequeña villa de pescadores, tras recorrerla se encuentran que necesitan una balsa para navegar por el río, para obtenerla, es necesario que le lleven al pescador una gran cantidad de pescado. Así comienza una escena donde se ve que Aalok, Rhiannon y Aramil llevan varias horas pescando con cañas y no han atrapado nada, por otra parte Yozan (en caso de haber aceptado que se uniera al equipo) está molesto por la pérdida de tiempo, Elizabeth está tomando el sol en una silla y Vladimir está observando el río. De pronto, Vladimir saca una bomba y la lanza al río, una explosión hace que muchos peces vuelen a la orilla, y todos sorprendidos los recogen, es así que obtienen la balsa que necesitaban.

Una vez que obtuvieron la balsa, los personajes navegan a través del río hasta llegar a unos rápidos, aquí se dan cuenta de que adelante hay rocas filosas y luego una cascada, con horror ven como su balsa se parte por la mitad al golpearse contra las rocas y luego caen por la cascada.

CUEVA FATUH Y TEMPLO DE SELEIN.

Al recobrar la conciencia, los personajes se encuentran en una extraña cueva, separados de sus demás compañeros deciden explorarla en busca del Templo de Selein, esperando hallar ahí a sus amigos.

> *En este punto, el jugador debe formar dos equipos, ya que manejará a ambos a lo largo de un complejo de cuevas subterráneas, los equipos se ayudarán mutuamente al abrir caminos o activar mecanismos que les permitan seguir adelante. A lo largo del camino se enfrentarán contra murciélagos, insectos, goblins, gigantes y plantas peligrosas.*

Una vez que llegan al templo, el cual es un edificio construido bajo tierra, los personajes deben enfrentar a los guardianes, un par de estatuas gigantes que cobran vida y los atacan.

> *En esta pelea, los personajes enfrentan a los guardianes (jefes de la escena) con los equipos que ya habían formado, un guardián para cada equipo.*

Dentro del templo encuentran una gran esfera de cristal y Aramil escucha una voz que lo llama, al acercarse a la esfera, un grupo de imágenes se forman en su mente, ve la llave ya unida, a Rhiannon, una extraña y espantosa puerta y finalmente, su propia muerte, es cuando se da cuenta de que está tirado en el suelo y todos sus amigos están a su alrededor. Luego escucha una voz que le dice cuál es el sitio para reunir las mitades de la llave y que para entrar necesita romper el sello mágico, que protege al lugar, utilizando un rubí del Desierto Somara.

El sitio para volver a unir la llave depende del personaje inicial que se haya escogido.

DESIERTO SOMARA.

A lo largo de esta escena los personajes se encontrarán con tormentas de arena que los regresarán al principio de la misma, para finalmente encontrar un oasis en cuyo centro verán que se encuentra un islote y en él hallarán un pedestal sobre el que descansa un hermoso rubí.

Los personajes atraviesan el oasis (el cual no es muy profundo) y toman el rubí, al hacerlo el agua a su alrededor se levanta como un gran muro y los ataca.

A lo largo de esta escena los personajes se enfrentarán con seres de arena, algunos insectos como escorpiones y arañas, así como algunos seres de fuego, el jefe de la escena es un gran muro de agua. En esta escena pueden encontrar un cofre con el arma más poderosa de Aramil.

8.6. Capítulo 4. ¿Final?

El sitio de la batalla final depende del personaje inicial, pero aunque dicha batalla no se desarrolle en cierto lugar, eso no quiere decir que el jugador no pueda explorar el sitio, es por eso que se describirá cada uno de los sitios posibles donde será la batalla final, así como los eventos que en dichos lugares se lleven a cabo. La última batalla y los sucesos que en ella se desarrollan será descrita como un apartado al final de este capítulo.

PUEBLO NAMODE Y CASTILLO UTUAL.

Al llegar los personajes, se dan cuenta de que algo extraño pasa en ese pueblo, la gente parece que actuara como autómatas, al acercarse a investigar se dan cuenta de que son muertos vivientes, condenados a repetir sus acciones cotidianas por toda la eternidad. Los personajes se preguntan que fue lo que paso en este lugar, en ese momento Rhiannon se siente débil y cae al suelo, los demás la ayudan a levantarse y ella dice: "Mephisto…".

Al avanzar por el pueblo, escuchan campanadas que provienen del castillo, y todos los habitantes del pueblo se vuelven agresivos y comienzan a atacar a los personajes.

Los personajes tienen que luchar contra zombis y esqueletos mientras avanzan por el pueblo, hasta que lleguen al castillo.

Al llegar al castillo los personajes escuchan de nuevo las campanadas, así que deciden dirigirse a la torre para investigar quien está tocando la campana. En su camino llegan al cuarto del trono, ahí el rey (un no muerto también) activa una trampa que abre el suelo, y hace que los personajes caigan en una mazmorra del castillo.

Al entrar al castillo y en adelante los personajes lucharan contra guardias y soldados no muertos.

Dos días después, en la mazmorra, suenan las campanas y los personajes que no iban en el equipo caen, encontrando a sus compañeros. Los personajes intercambian sus impresiones del lugar y descubren que Mephisto debió matar a todos los habitantes y convertirlos en no muertos bajo su control. Vladimir utiliza una de sus bombas para abrir la puerta y los personajes salen de la mazmorra para encontrarse en un complicado calabozo.

> *Aquí el jugador debe formar dos equipos los cuales avanzaran por el calabozo abriéndose camino mutuamente hasta que salgan del calabozo y se encuentren en el cuarto del trono, ahí lucharán contra el rey y su ejercito de no muertos, para esto los dos equipos lucharan contra este jefe y su ejercito a la vez, uno enfrentándose al rey mientras otro equipo lucha contra su ejercito y el mago que protege con su magia a su amo.*

Al vencer al rey, los personajes encuentran libre su camino y se dirigen a la torre donde encuentran una extraña campana, la cual destruyen con lo cual liberan a todas las almas del pueblo y el castillo.

> *Si el personaje inicial escogido fue Rhiannon, este es el sitio de la batalla final y ante los personajes aparece Mephisto, lo acontecido después de esto se encuentra en el apartado de la Batalla Final.*

RUINAS DE LA ANTIGUA CIUDAD DENAYO.

Al entrar a este lugar, los personajes se encuentran maravillados por las extrañas ruinas que encuentran a orillas del mar, Elizabeth es la más impresionada y propone explorar a fondo el lugar ya que seguramente hay grandes tesoros ocultos en la zona.

Entre las ruinas, los personajes descubren un extraño dispositivo que no funciona (en caso de que se encuentre Vladimir en el equipo, este examina el dispositivo y comenta que parece ser una especie de transporte al subsuelo y que es necesario encontrar dos medallones que funcionan como llaves para

hacerlo trabajar). Después de explorar las ruinas y encontrar los medallones, los personajes abordan el extraño transporte y se encuentran a las puertas de un fantástico palacio en el fondo del mar, se dan cuenta de que pueden respirar gracias a que una burbuja de aire está atrapada dentro del palacio.

En la superficie, los personajes se enfrentan contra enemigos de tipo mágico, como estatuas y armaduras vivientes, armas animadas y cristales que lanzan hechizos. Una vez en el palacio, los enemigos cambian por criaturas marinas y hombres pez. Dentro del palacio submarino, los personajes pueden encontrar el artefacto que le da a Vladimir la habilidad de Taladro, así como el arma más poderosa para el. El jefe de esta escena es un calamar gigante el cual se encuentra en el cuarto del trono una vez derrotado y en el caso de que el personaje inicial haya sido Aalok, los personajes se encuentran con Jarein y lo que acontece a continuación está detallado en el apartado llamado Batalla Final.

PICO DE GAUSS.

El Pico de Gauss es un volcán que se encontraba dormido y que recientemente despertó.

Los personajes deben avanzar enfrentándose con salamandras, elementales de fuego, pequeños dragones y como jefe del lugar vencer a un enorme dragón. En este lugar los personajes pueden encontrar el arma más poderosa de Rhiannon.

Tras vencer al jefe de la escena y si el personaje inicial fue Aramil, aparece Dieter y lo que pasa a continuación se explica en el apartado llamado Batalla Final.

TEMPLO DE LA MALDAD PRIMORDIAL.

Al entrar a este abandonado y viejo templo, los personajes sienten escalofríos, Yozan menciona que siente una maldad muy poderosa en el ambiente y es cuando escucha una voz que lo llama, la voz le promete gran poder, pero este poder tiene un precio, el cual es que por siempre será un ser de la oscuridad sin la menor esperanza de regresar a su antigua forma.

> *En el caso de que el jugador no haya aceptado a Yozan, la parte referente a el no ocurre y simplemente los personajes mencionan que sienten algo ajeno a este mundo en este lugar. A lo largo de todo el templo, los personajes se enfrentan contra extraños seres sin forma y contra demonios.*

En cierto punto los personajes son atacados y capturados en una emboscada, todos menos Aramil, el cual se esconde al ver a los enemigos que los rodean, Aramil temeroso y sólo, busca a sus amigos encontrándolos uno a uno y liberándolos de los demonios.

> *Al ser capturados los personajes, el jugador toma el papel de Aramil, que debe buscar y liberar a sus compañeros, cada uno de los personajes se encuentra custodiado por un demonio el cual debe ser derrotado antes de liberar al personaje, el derrotar al demonio puede ser a través de una batalla, resolviendo algún acertijo, encontrando algún objeto o realizando alguna otra tarea.*

Al encontrar a Yozan, los personajes descubren que este no está preso, al contrario, se encuentra frente a una extraña arma la cual se haya clavada en un altar, un demonio que se encuentra a su lado le dice: "Ya lo sabes, poder a cambio de la esperanza de recuperar tu humanidad, ¿qué decides?". Yozan da un paso adelante y toma el arma diciendo "Intercambio mi humanidad por el poder para lograr mi venganza", el demonio ríe y luego desaparece, Yozan da vuelta y ve a los demás personajes que los están observando, el no muerto les pregunta que están mirando y les dice que no tienen tiempo para perderlo en

niñerías que es mejor que todos se pongan en marcha.

El arma que tomó Yozan es la más poderosa que puede tener en el juego.

Al llegar al altar principal del templo, los personajes se encuentran con un gran demonio, el cual les da la bienvenida a su muerte y al sufrimiento eterno.

El demonio en el altar principal funge como jefe de esta escena, al vencerlo y en el caso de que el personaje inicial haya sido Vladimir, Johanna aparece y lo que sucede después esta detallado en el apartado llamado Batalla Final.

BATALLA FINAL.

Dependiendo del personaje inicial que se halla elegido, la batalla final se desarrolla contra un enemigo diferente, de manera que si fue Aalok el enemigo será Jarein, en el caso de Rhiannon será Mephisto, con Vladimir se trata de Johanna y finalmente en el caso de que haya sido Aramil se tendrán que enfrentar con Dieter.

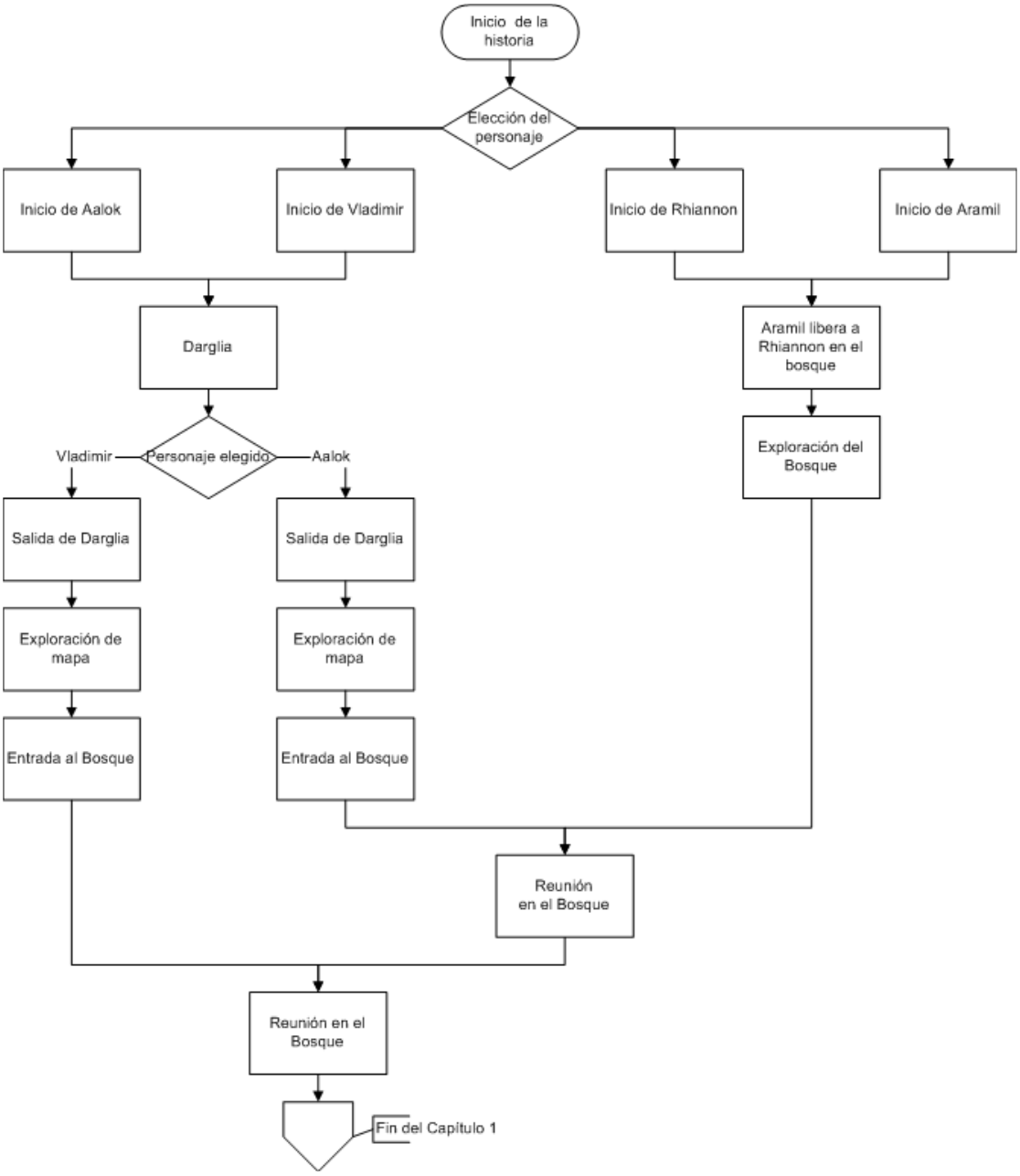
El enemigo final felicita a los personajes por haber llegado tan lejos, pero les dice que es una lástima ya que la llave ya ha sido reconstruida, los personajes le preguntan para que quiere la llave a lo que este responde diciendo que la verdad no puede ser negada, los personajes no entienden esta respuesta, pero saben que su supuesta verdad no es mas que destrucción y muerte por lo que tienen que detenerlo.

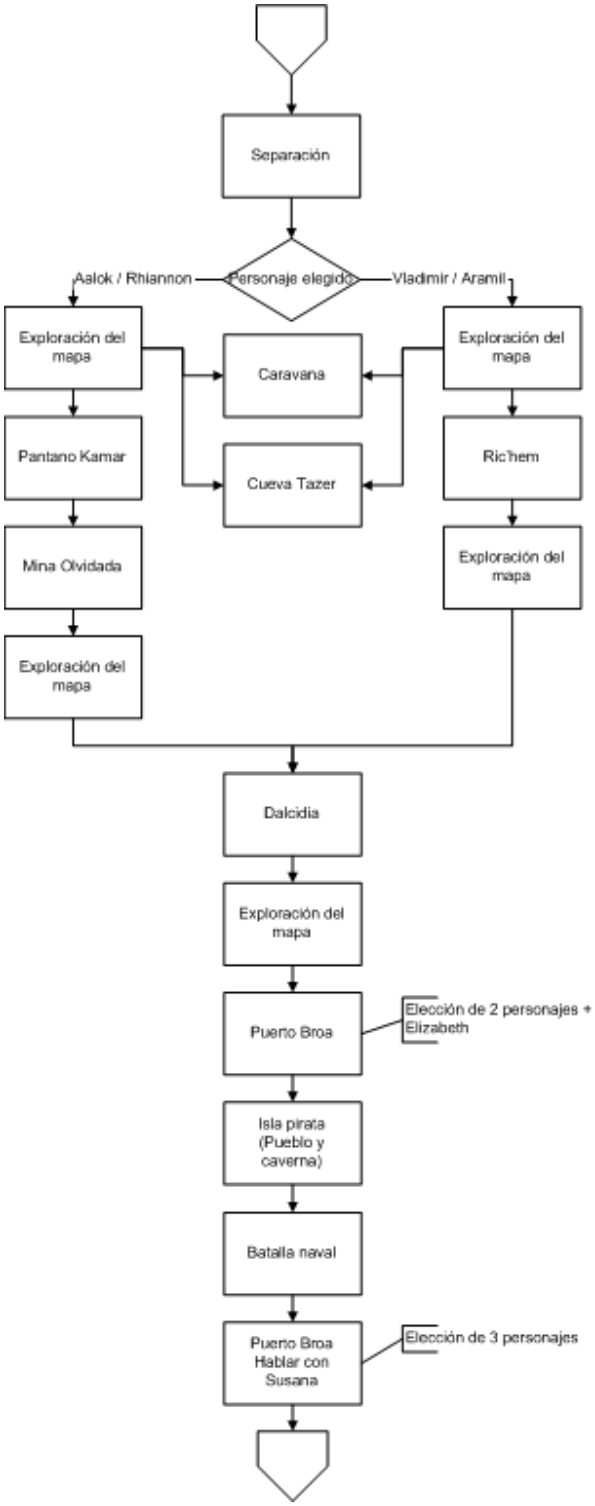
Antes de comenzar la batalla el jugador debe seleccionar a tres personajes que acompañaran al personaje inicial. A lo largo de la batalla aparecerá de manera aleatoria algún personaje que no este luchando para utilizar alguna habilidad y luego desaparecer, por ejemplo, si Aramil no se encuentra luchando puede aparecer en cierto momento, lanzar un hechizo de curación y desaparecer.

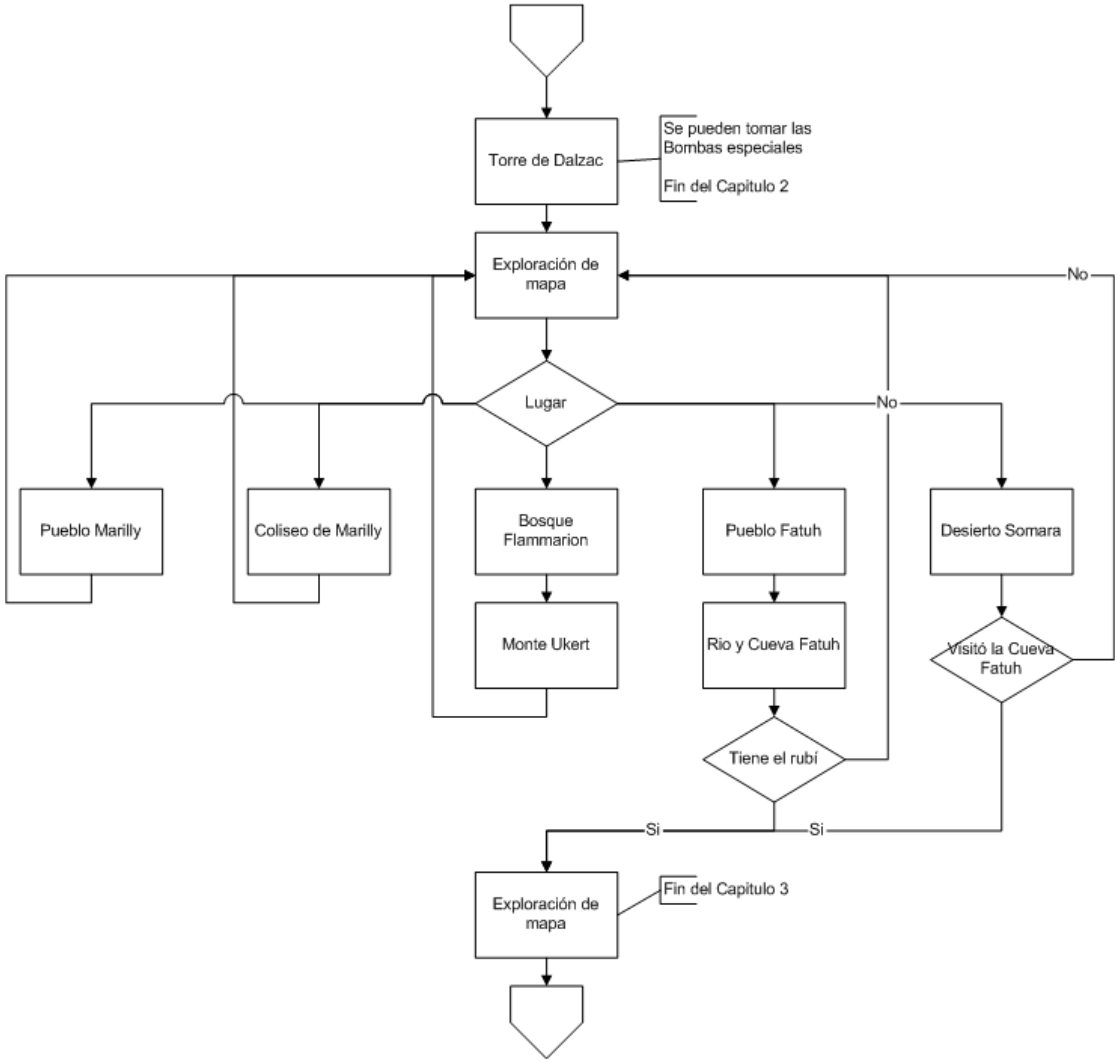
Al derrotar al enemigo, este jadeante y moribundo se pregunta como es posible que halla sido derrotado, luego les pregunta que por qué lo hacen, por qué evitan que todo termine, antes de morir mira fijamente a Rhiannon y dice, si he de desaparecer entonces no tengo nada que perder y ataca a la pequeña niña, Aramil cubre a Rhiannon y recibe de lleno el ataque, todos corren a su alrededor sólo para descubrir que no pueden hacer nada para salvarlo y es así como, tanto el joven Aramil como el enemigo final mueren al mismo tiempo.

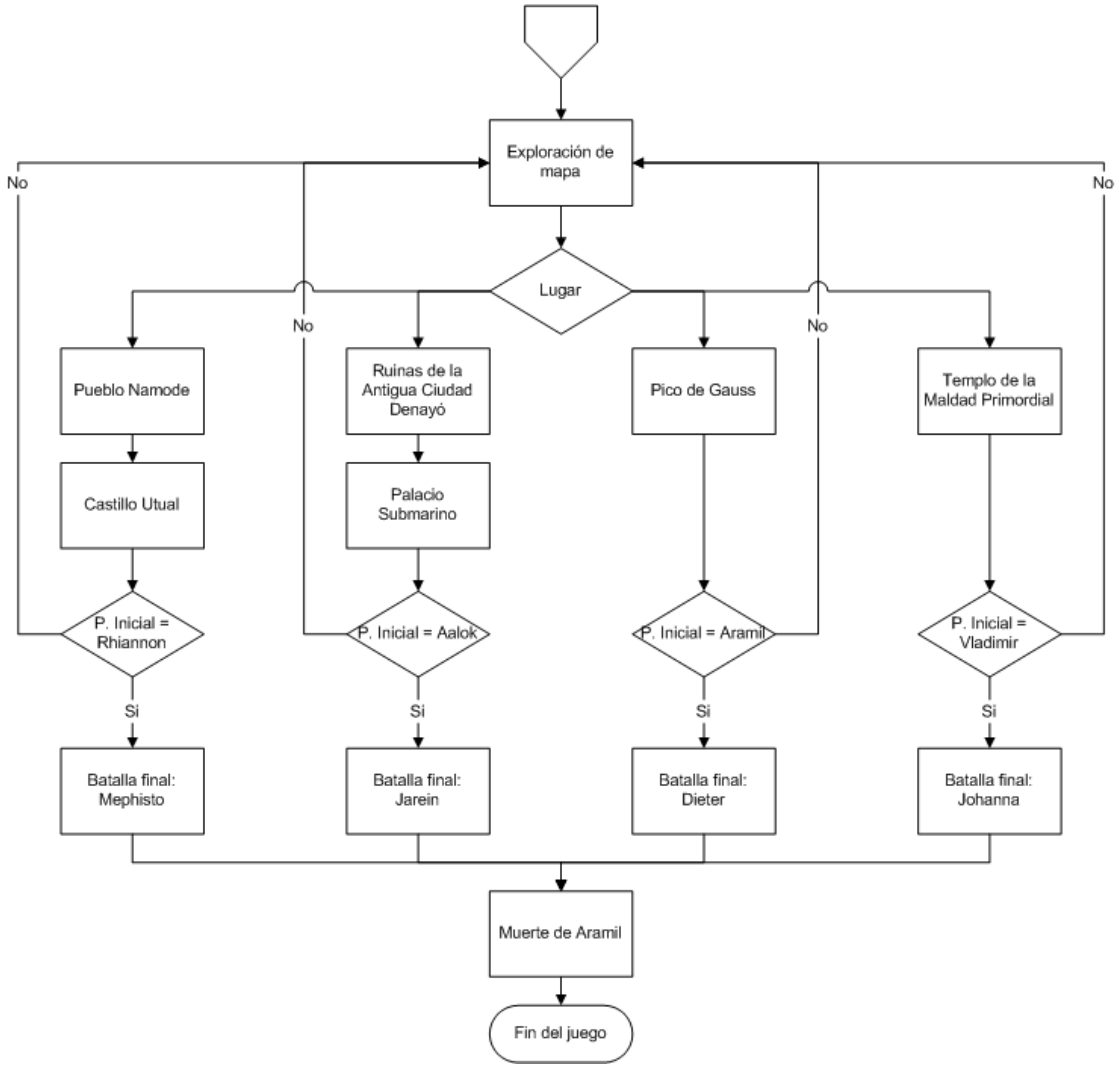
Al final los personajes abatidos por los hechos y con grandes interrogantes se separan, algunos para regresar a sus casas mientras otros para buscar nuevos significados a sus vidas.

9. APÉNDICE 2: DIAGRAMAS DE FLUJO DEL GUIÓN.

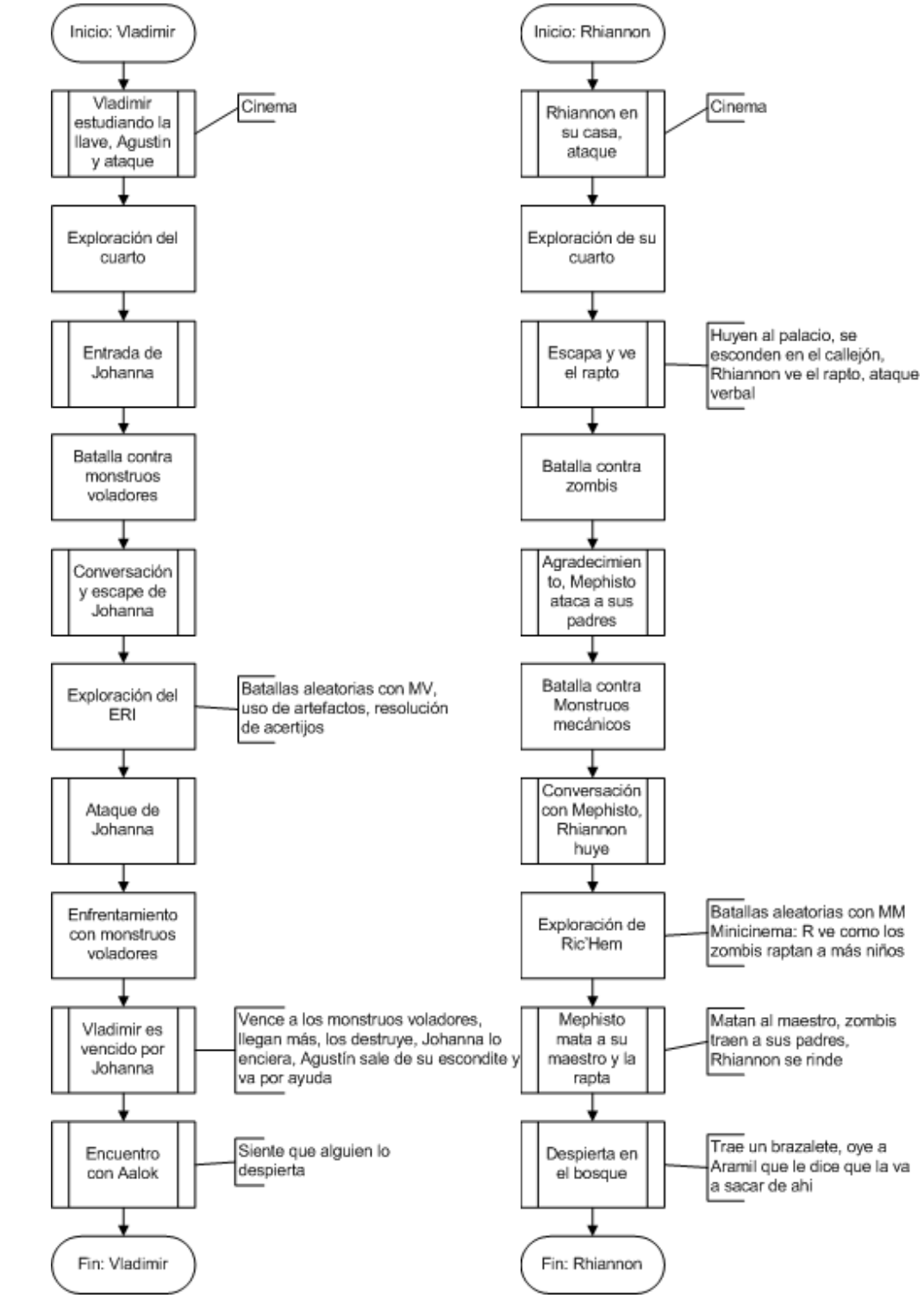


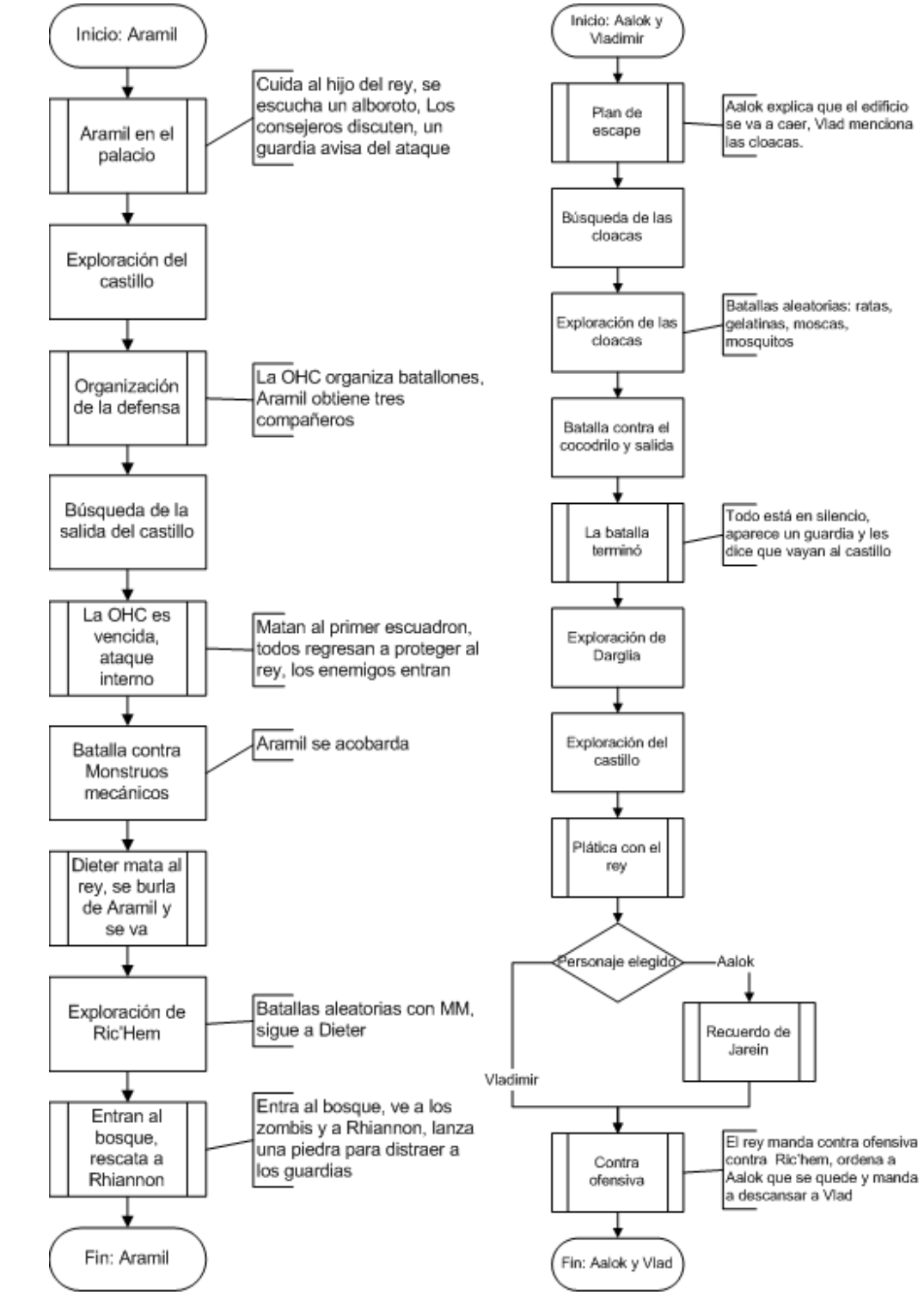

Separación
Personaje elegido
Aalok / Rhiannon
Vladimir / Aramil
Exploración del mapa
Caravana
Exploración del mapa
Pantano Kamar
Cueva Tazer
Ric'hem
Mina Olvidada
Exploración del mapa
Exploración del mapa
Dalcidia
Exploración del mapa
Puerto Broa
Elección de 2 personajes + Elizabeth
Isla pirata (Pueblo y caverna)
Batalla naval
Puerto Broa Hablar con Susana
Elección de 3 personajes

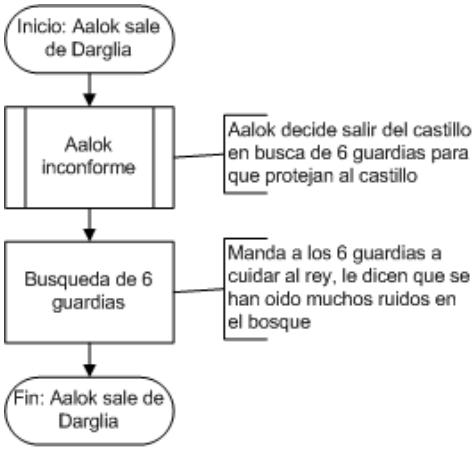

Torre de Dalzac
Se pueden tomar las Bombas especiales
Fin del Capitulo 2
Exploración de mapa
No
Lugar
No
Pueblo Marilly
Coliseo de Marilly
Bosque Flammarion
Pueblo Fatuh
Desierto Somara
Monte Ukert
Rio y Cueva Fatuh
Visitó la Cueva Fatuh
Tiene el rubí
Si
Si
Exploración de mapa
Fin del Capitulo 3


Exploración de mapa
Lugar
No
No
No
No
Pueblo Namode
Castillo Utual
P. Inicial = Rhiannon
Si
Batalla final: Mephisto
Ruinas de la Antigua Ciudad Denayó
Palacio Submarino
P. Inicial = Aalok
Si
Batalla final: Jarein
Pico de Gauss
P. Inicial = Aramil
Si
Batalla final: Dieter
Templo de la Maldad Primordial
P. Inicial = Vladimir
Si
Batalla final: Johanna
Muerte de Aramil
Fin del juego

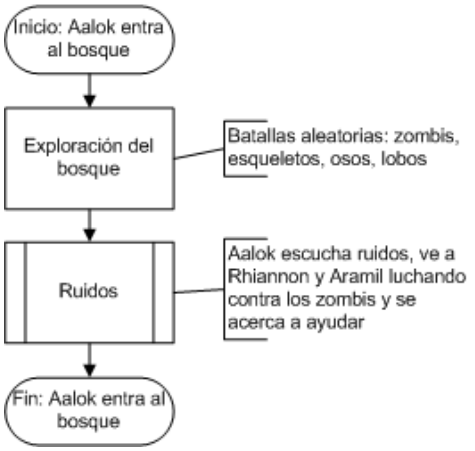

Inicio: Aalok
Aalok en las Barracas
Cinema
Barracas
Puede explorar para encontrar su espada, así como pedir a dos guardias que lo acompañen
Exploración de Darglia
Batallas aleatorias con zombis
Escuadrón de Zombis
Seguir al escuadrón
Batallas aleatorias con zombis
Detención del Escuadrón
Enfrentamiento con Zombis
Reto a Jarein
Enfrentamiento con Jarein
Jarein los vence
Edificio en llamas, misión de rescate
Exploración del ERI
Batallas aleatorias con monstruos mecánicos
Encuentro con Vladimir
Llega, lo ve incosciente y lo despierta
Fin: Aalok

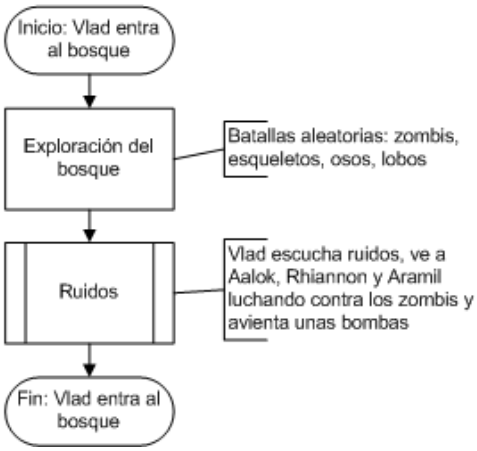

Inicio: Vladimir
Vladimir estudiando la llave, Agustin y ataque
Cinema
Exploración del cuarto
Entrada de Johanna
Batalla contra monstruos voladores
Conversación y escape de Johanna
Exploración del ERI
Batallas aleatorias con MV, uso de artefactos, resolución de acertijos
Ataque de Johanna
Enfrentamiento con monstruos voladores
Vladimir es vencido por Johanna
Vence a los monstruos voladores, llegan más, los destruye, Johanna lo enciera, Agustín sale de su escondite y va por ayuda
Encuentro con Aalok
Siente que alguien lo despierta
Fin: Vladimir
Inicio: Rhiannon
Rhiannon en su casa, ataque
Cinema
Exploración de su cuarto
Escapa y ve el rapto
Huyen al palacio, se esconden en el callejón, Rhiannon ve el rapto, ataque verbal
Batalla contra zombis
Agradecimiento, Mephisto ataca a sus padres
Batalla contra Monstruos mecánicos
Conversación con Mephisto, Rhiannon huye
Exploración de Ric'Hem
Batallas aleatorias con MM Minicinema: R ve como los zombis raptan a más niños
Mephisto mata a su maestro y la rapta
Matan al maestro, zombis traen a sus padres, Rhiannon se rinde
Despierta en el bosque
Trae un brazalete, oye a Aramil que le dice que la va a sacar de ahi
Fin: Rhiannon

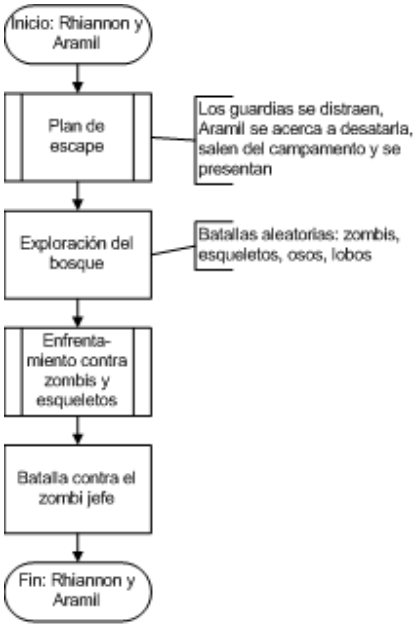

Inicio: Aramil
Aramil en el palacio
Cuida al hijo del rey, se escucha un alboroto, Los consejeros discuten, un guardia avisa del ataque
Exploración del castillo
Organización de la defensa
La OHC organiza batallones, Aramil obtiene tres compañeros
Búsqueda de la salida del castillo
La OHC es vencida, ataque interno
Matan al primer escuadron, todos regresan a proteger al rey, los enemigos entran
Batalla contra Monstruos mecánicos
Aramil se acobarda
Dieter mata al rey, se burla de Aramil y se va
Exploración de Ric'Hem
Batallas aleatorias con MM, sigue a Dieter
Entran al bosque, rescata a Rhiannon
Entra al bosque, ve a los zombis y a Rhiannon, lanza una piedra para distraer a los guardias
Fin: Aramil
Inicio: Aalok y Vladimir
Plan de escape
Aalok explica que el edificio se va a caer, Vlad menciona las cloacas.
Búsqueda de las cloacas
Exploración de las cloacas
Batallas aleatorias: ratas, gelatinas, moscas, mosquitos
Batalla contra el cocodrilo y salida
La batalla terminó
Todo está en silencio, aparece un guardia y les dice que vayan al castillo
Exploración de Darglia
Exploración del castillo
Plática con el rey
Personaje elegido
Aalok
Recuerdo de Jarein
Vladimir
Contra ofensiva
El rey manda contra ofensiva contra Ric'hem, ordena a Aalok que se quede y manda a descansar a Vlad
Fin: Aalok y Vlad


Inicio: Aalok sale de Darglia
Aalok inconforme
Aalok decide salir del castillo en busca de 6 guardias para que protejan al castillo
Busqueda de 6 guardias
Manda a los 6 guardias a cuidar al rey, le dicen que se han oido muchos ruidos en el bosque
Fin: Aalok sale de Darglia

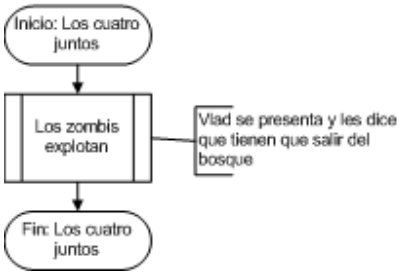

Inicio: Vlad sale de Darglia
Aalok inconforme
Vlad ve salir a Aalok, desobedeciendo sus órdenes, y decide seguirlo
Exploración de Darglia
La gente le dice que se han oido muchos ruidos en el bosque. En la salida le mencionan que Aalok fué a investigar
Fin: Vlad sale de Darglia

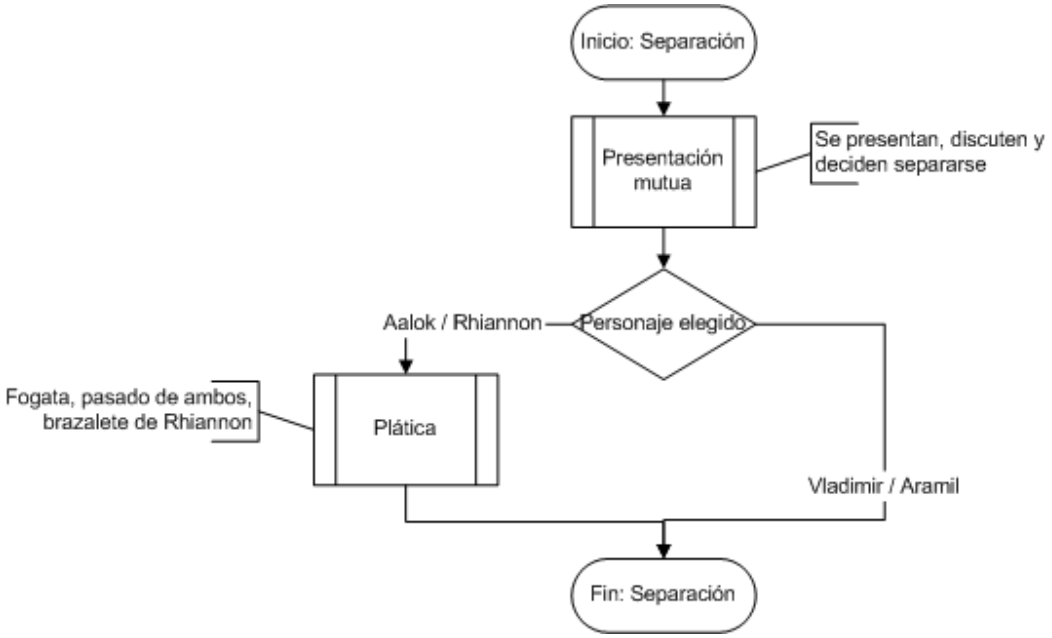

Inicio: Aalok entra al bosque
Exploración del bosque
Batallas aleatorias: zombis, esqueletos, osos, lobos
Ruidos
Aalok escucha ruidos, ve a Rhiannon y Aramil luchando contra los zombis y se acerca a ayudar
Fin: Aalok entra al bosque

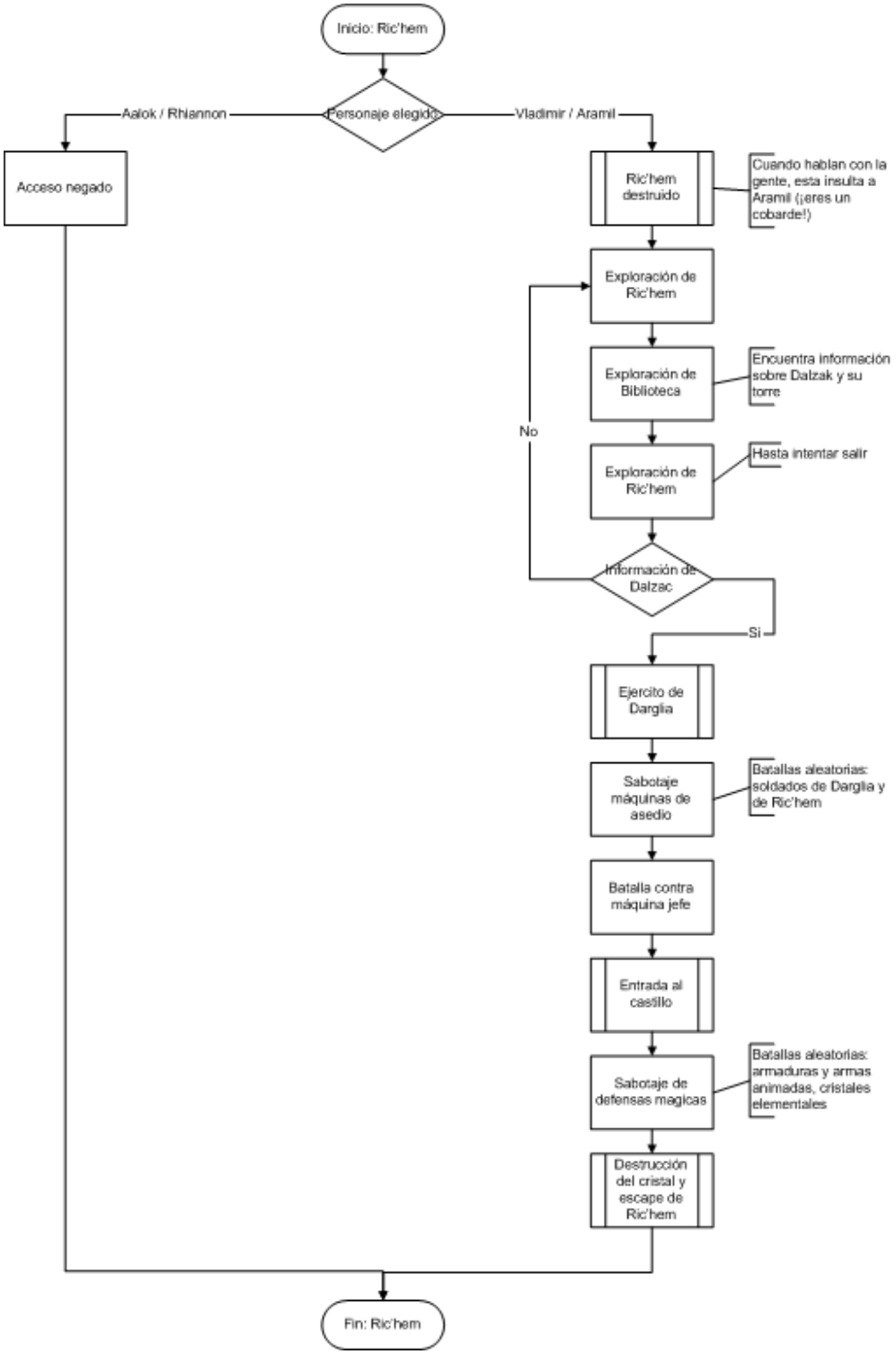

Inicio: Vlad entra al bosque
Exploración del bosque
Batallas aleatorias: zombis, esqueletos, osos, lobos
Ruidos
Vlad escucha ruidos, ve a Aalok, Rhiannon y Aramil luchando contra los zombis y avienta unas bombas
Fin: Vlad entra al bosque

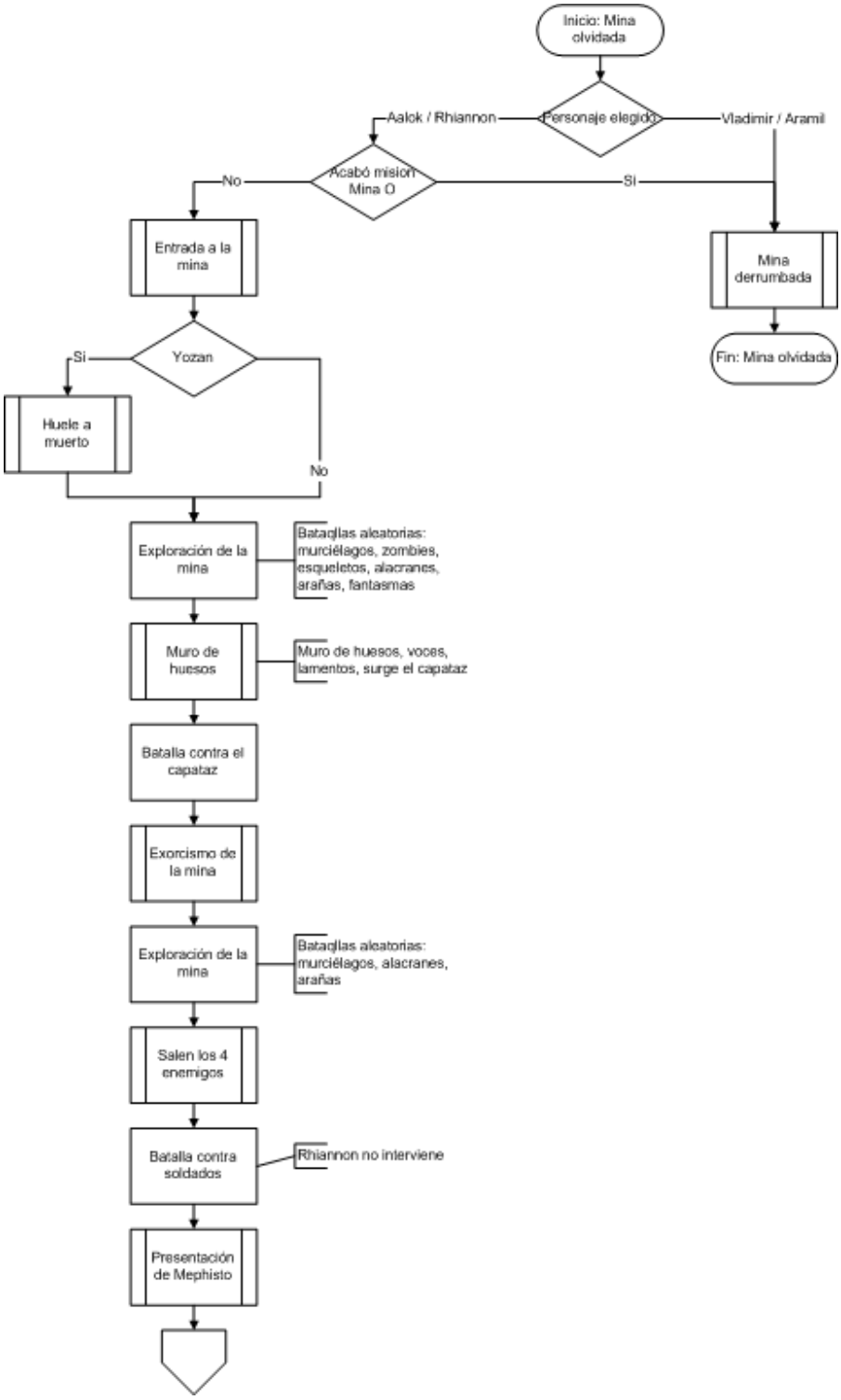

Inicio: Rhiannon y Aramil
Plan de escape
Los guardias se distraen, Aramil se acerca a desatarla, salen del campamento y se presentan
Exploración del bosque
Batallas aleatorias: zombis, esqueletos, osos, lobos
Enfrenta-miento contra zombis y esqueletos
Batalla contra el zombi jefe
Fin: Rhiannon y Aramil

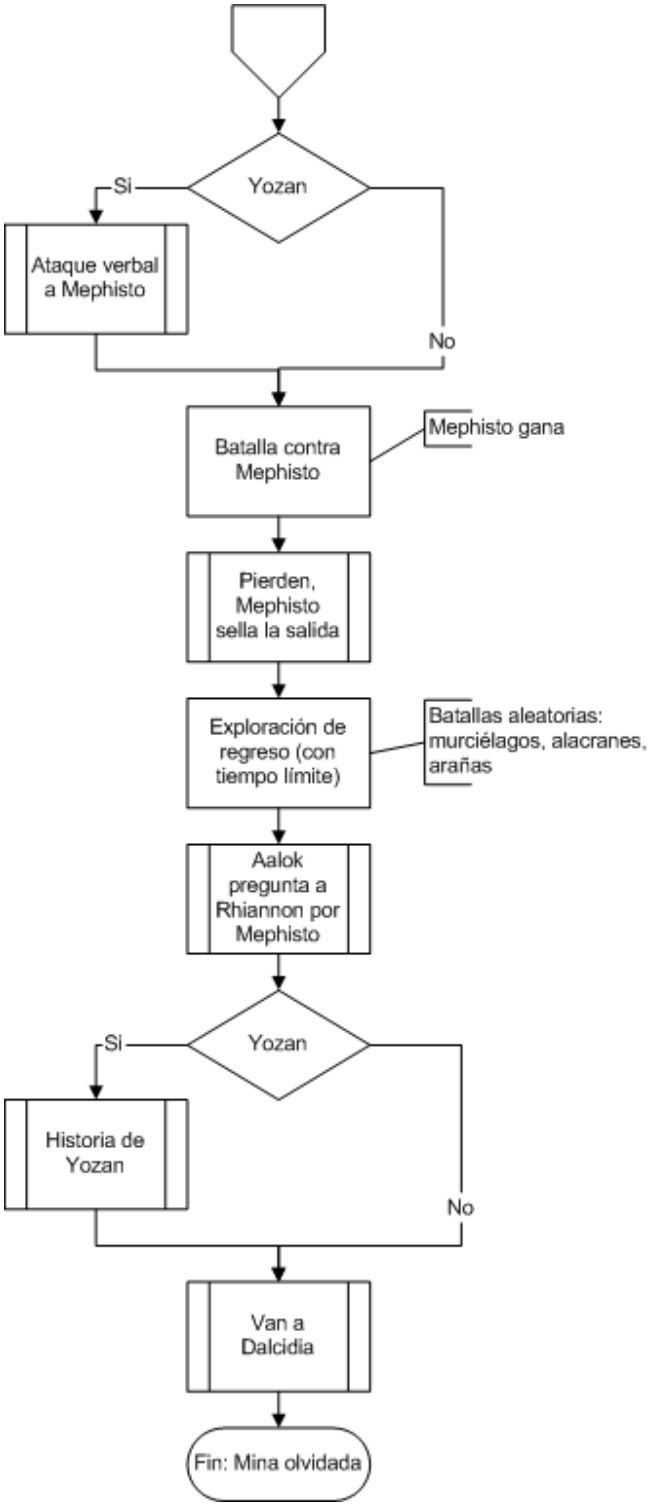

Inicio: Rhiannon, Aramil y Aalok
Zombis atacan a Rhiannon y Aramil
Los zombis los rodean, aparece Aalok, ellos se asustan pero él ataca a los zombis
Batalla con los zombis
Fin: Rhiannon, Aramil y Aalok

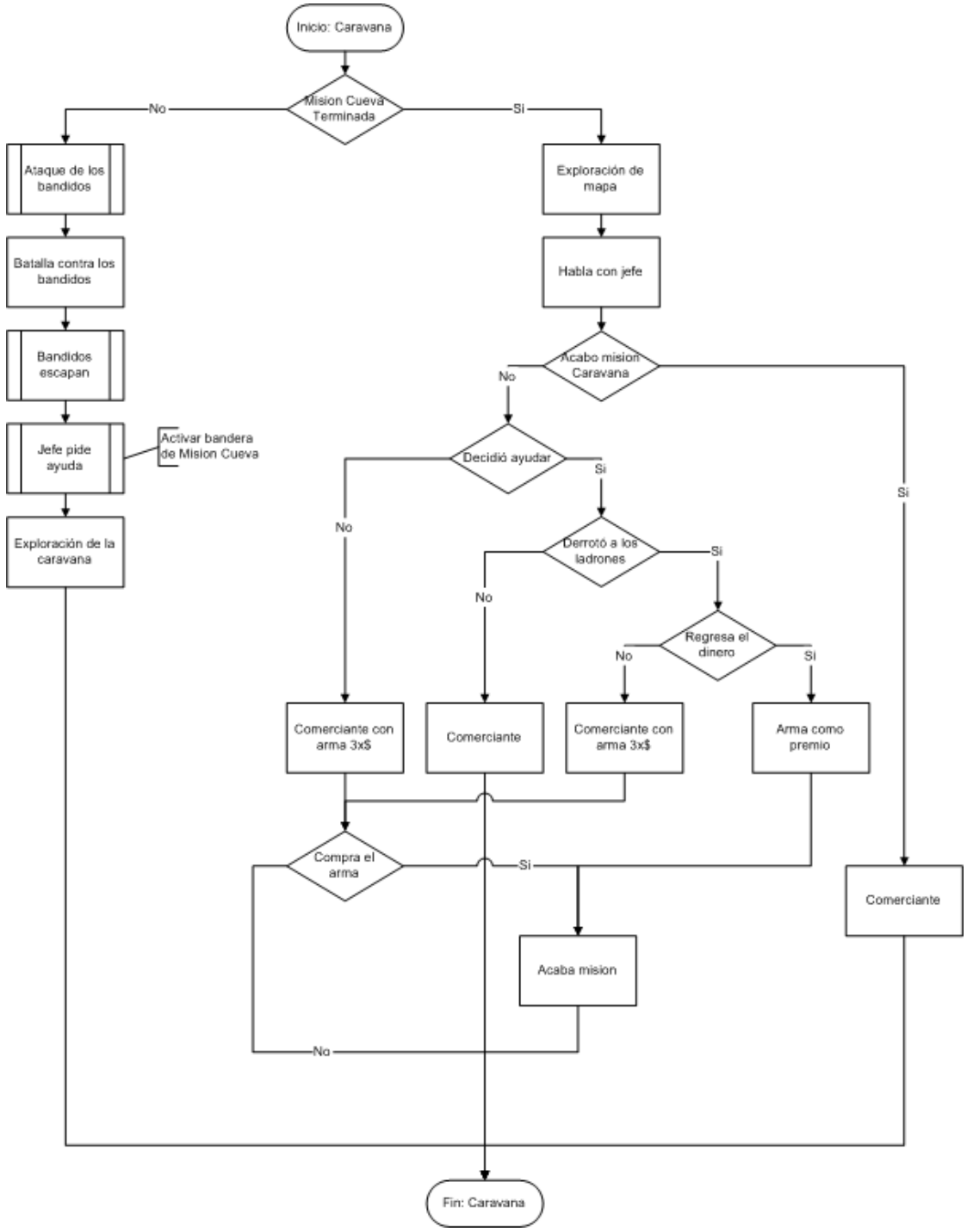

Inicio: Los cuatro juntos
Los zombis explotan
Vlad se presenta y les dice que tienen que salir del bosque
Fin: Los cuatro juntos

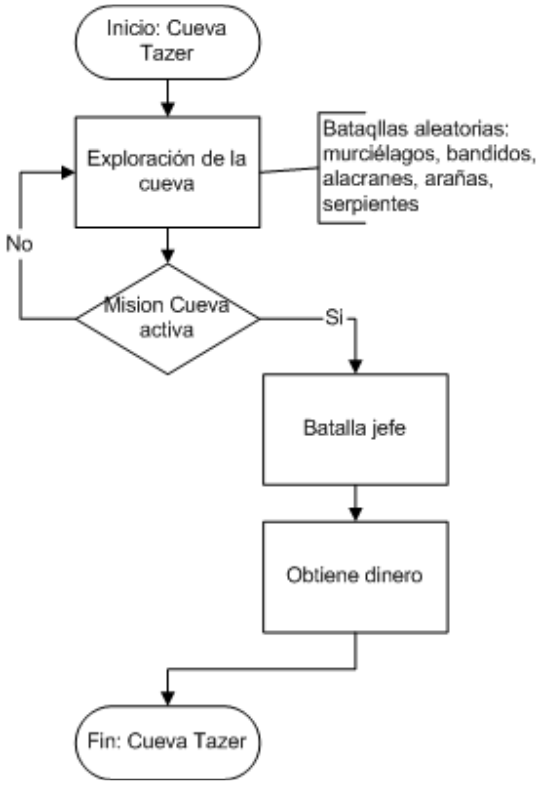

Inicio: Separación
Presentación mutua
Se presentan, discuten y deciden separarse
Personaje elegido
Aalok / Rhiannon
Fogata, pasado de ambos, brazalete de Rhiannon
Plática
Vladimir / Aramil
Fin: Separación

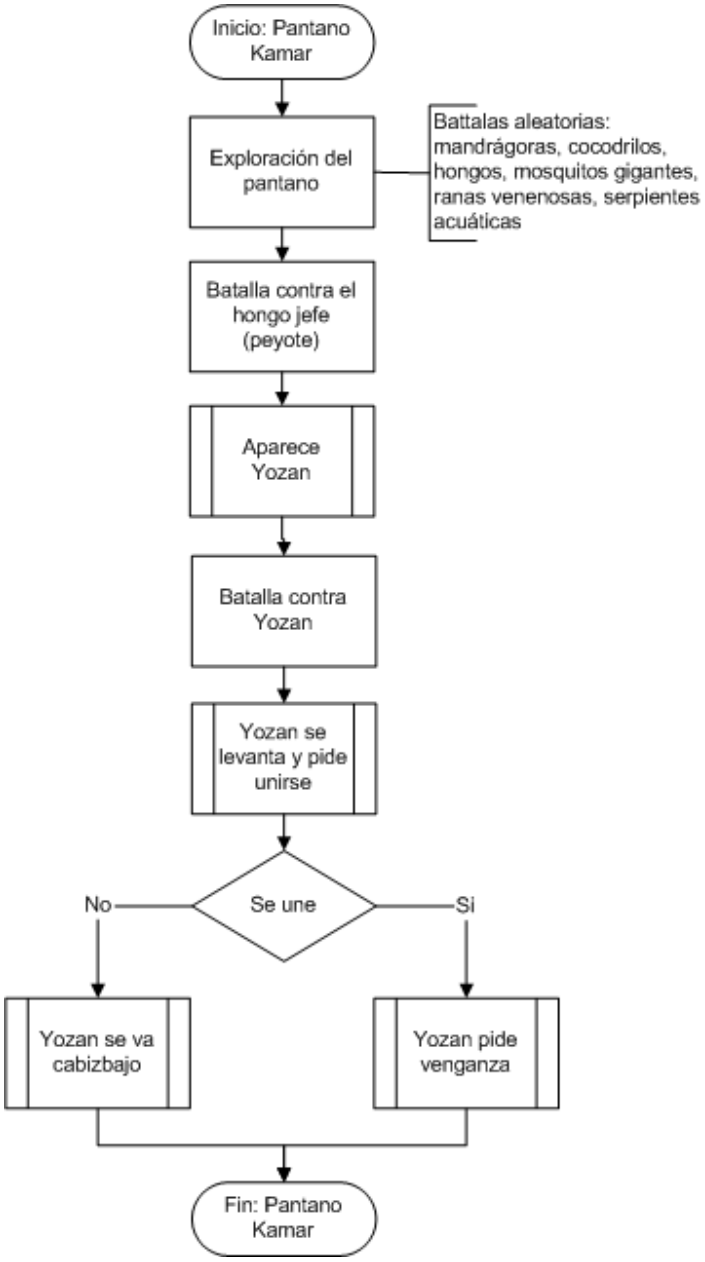

Inicio: Ric'hem
Personaje elegido
Aalok / Rhiannon
Vladimir / Aramil
Acceso negado
Ric'hem destruido
Cuando hablan con la gente, esta insulta a Aramil (¡eres un cobarde!)
Exploración de Ric'hem
Exploración de Biblioteca
Encuentra información sobre Dalzak y su torre
No
Exploración de Ric'hem
Hasta intentar salir
Información de Dalzac
Si
Ejercito de Darglia
Sabotaje máquinas de asedio
Batallas aleatorias: soldados de Darglia y de Ric'hem
Batalla contra máquina jefe
Entrada al castillo
Sabotaje de defensas magicas
Batallas aleatorias: armaduras y armas animadas, cristales elementales
Destrucción del cristal y escape de Ric'hem
Fin: Ric'hem

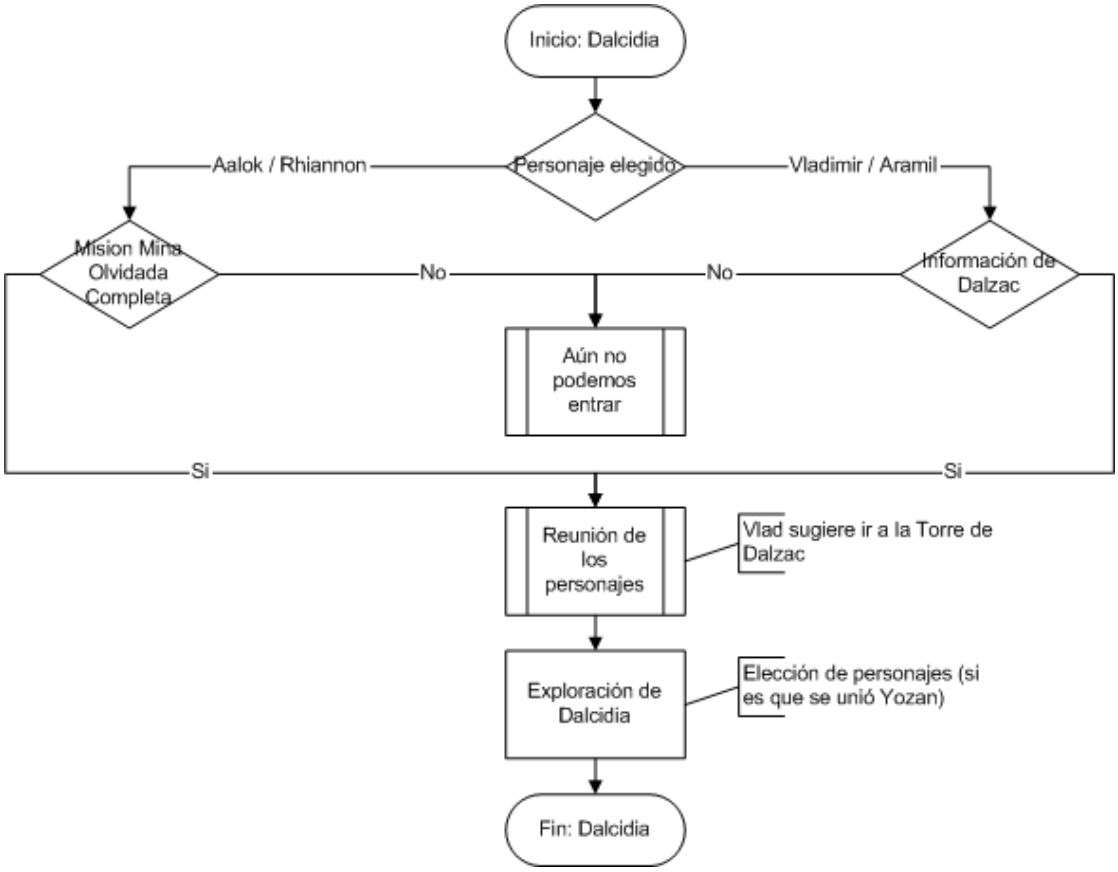

Inicio: Mina olvidada
Personaje elegido
Aalok / Rhiannon
Vladimir / Aramil
Acabó mision Mina O
No
Si
Entrada a la mina
Mina derrumbada
Fin: Mina olvidada
Yozan
Si
No
Huele a muerto
Exploración de la mina
Batallas aleatorias: murciélagos, zombies, esqueletos, alacranes, arañas, fantasmas
Muro de huesos
Muro de huesos, voces, lamentos, surge el capataz
Batalla contra el capataz
Exorcismo de la mina
Exploración de la mina
Batallas aleatorias: murciélagos, alacranes, arañas
Salen los 4 enemigos
Batalla contra soldados
Rhiannon no interviene
Presentación de Mephisto

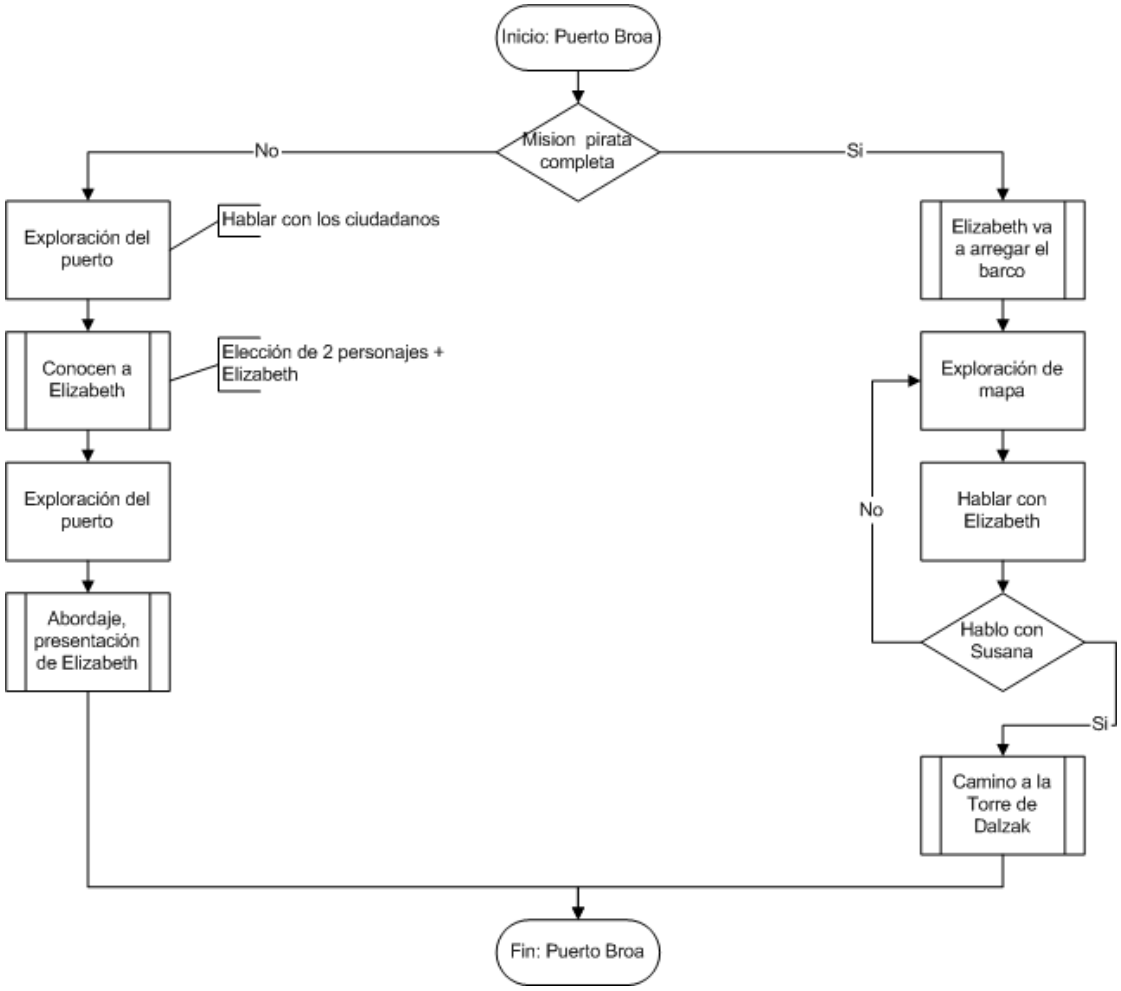

Yozan
Si
Ataque verbal a Mephisto
No
Batalla contra Mephisto
Mephisto gana
Pierden, Mephisto sella la salida
Exploración de regreso (con tiempo límite)
Batallas aleatorias: murciélagos, alacranes, arañas
Aalok pregunta a Rhiannon por Mephisto
Yozan
Si
Historia de Yozan
No
Van a Dalcidia
Fin: Mina olvidada

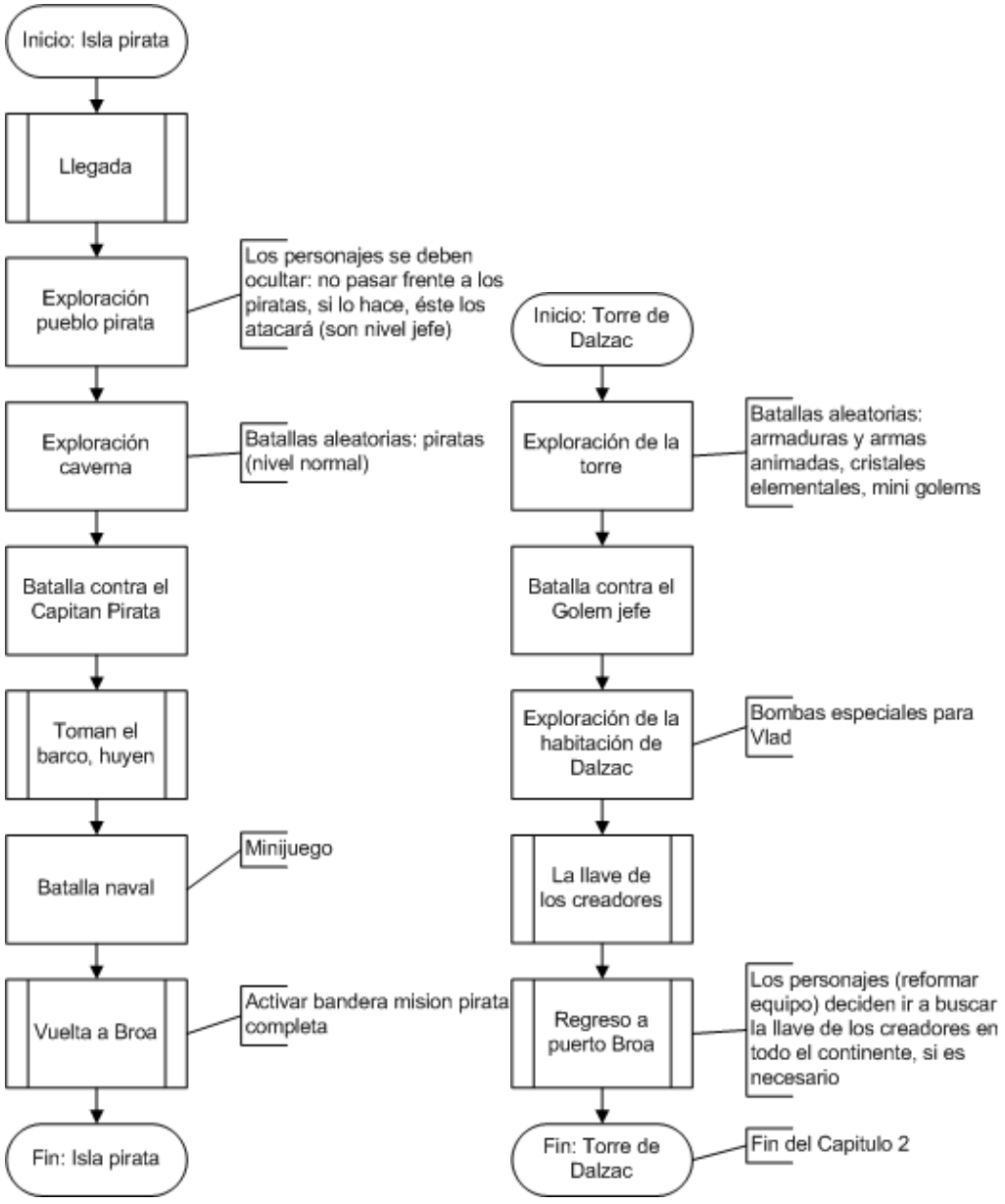

Inicio: Caravana
Mision Cueva Terminada
No
Si
Ataque de los bandidos
Batalla contra los bandidos
Bandidos escapan
Jefe pide ayuda
Activar bandera de Mision Cueva
Exploración de la caravana
Exploración de mapa
Habla con jefe
Acabo mision Caravana
No
Si
Decidió ayudar
No
Si
Derrotó a los ladrones
No
Si
Regresa el dinero
No
Si
Comerciante con arma 3x$
Comerciante
Comerciante con arma 3x$
Arma como premio
Compra el arma
Si
No
Acaba mision
Comerciante
Fin: Caravana

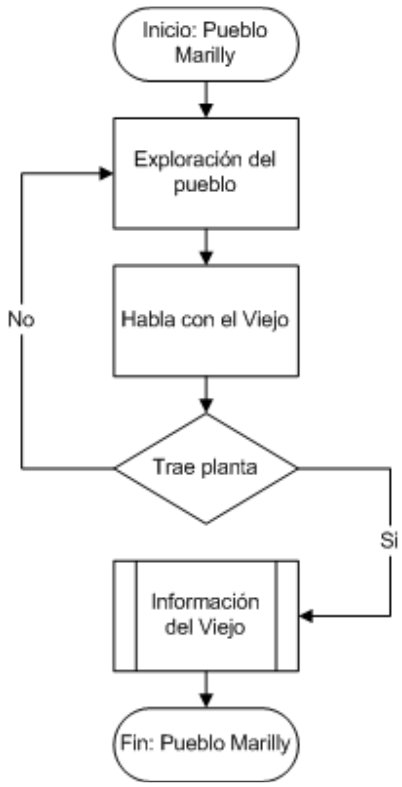

Inicio: Cueva Tazer
Exploración de la cueva
Bataqllas aleatorias: murciélagos, bandidos, alacranes, arañas, serpientes
No
Mision Cueva activa
Si
Batalla jefe
Obtiene dinero
Fin: Cueva Tazer

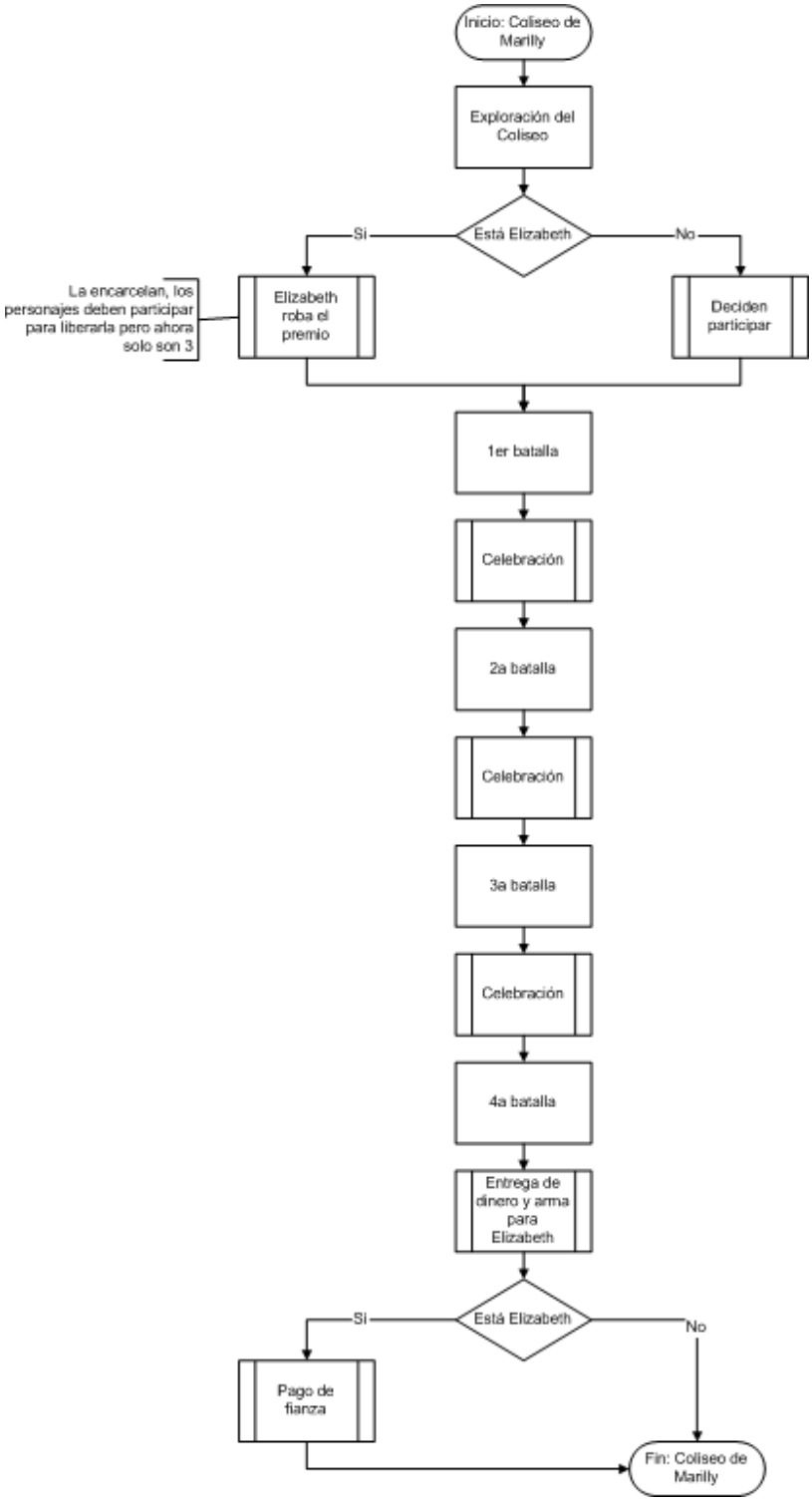

Inicio: Pantano Kamar
Exploración del pantano
Battalas aleatorias: mandrágoras, cocodrilos, hongos, mosquitos gigantes, ranas venenosas, serpientes acuáticas
Batalla contra el hongo jefe (peyote)
Aparece Yozan
Batalla contra Yozan
Yozan se levanta y pide unirse
Se une
No
Si
Yozan se va cabizbajo
Yozan pide venganza
Fin: Pantano Kamar

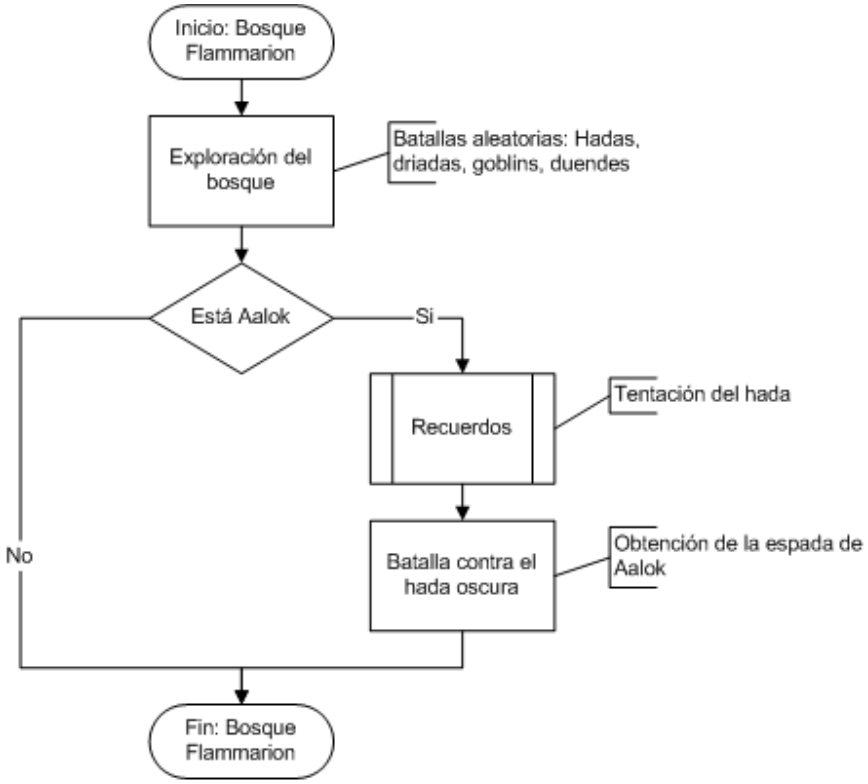

Inicio: Dalcidia
Personaje elegido
Aalok / Rhiannon
Vladimir / Aramil
Mision Mina Olvidada Completa
Información de Dalzac
No
No
Aún no podemos entrar
Si
Si
Reunión de los personajes
Vlad sugiere ir a la Torre de Dalzac
Exploración de Dalcidia
Elección de personajes (si es que se unió Yozan)
Fin: Dalcidia

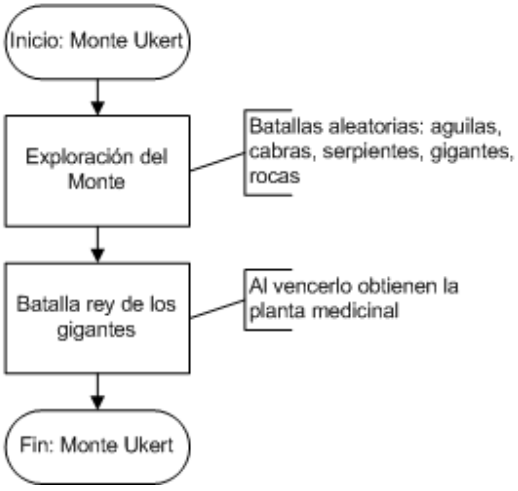

Inicio: Puerto Broa
Mision pirata completa
No
Si
Exploración del puerto
Hablar con los ciudadanos
Conocen a Elizabeth
Elección de 2 personajes + Elizabeth
Exploración del puerto
Abordaje, presentación de Elizabeth
Elizabeth va a arregar el barco
Exploración de mapa
Hablar con Elizabeth
Hablo con Susana
No
Si
Camino a la Torre de Dalzak
Fin: Puerto Broa

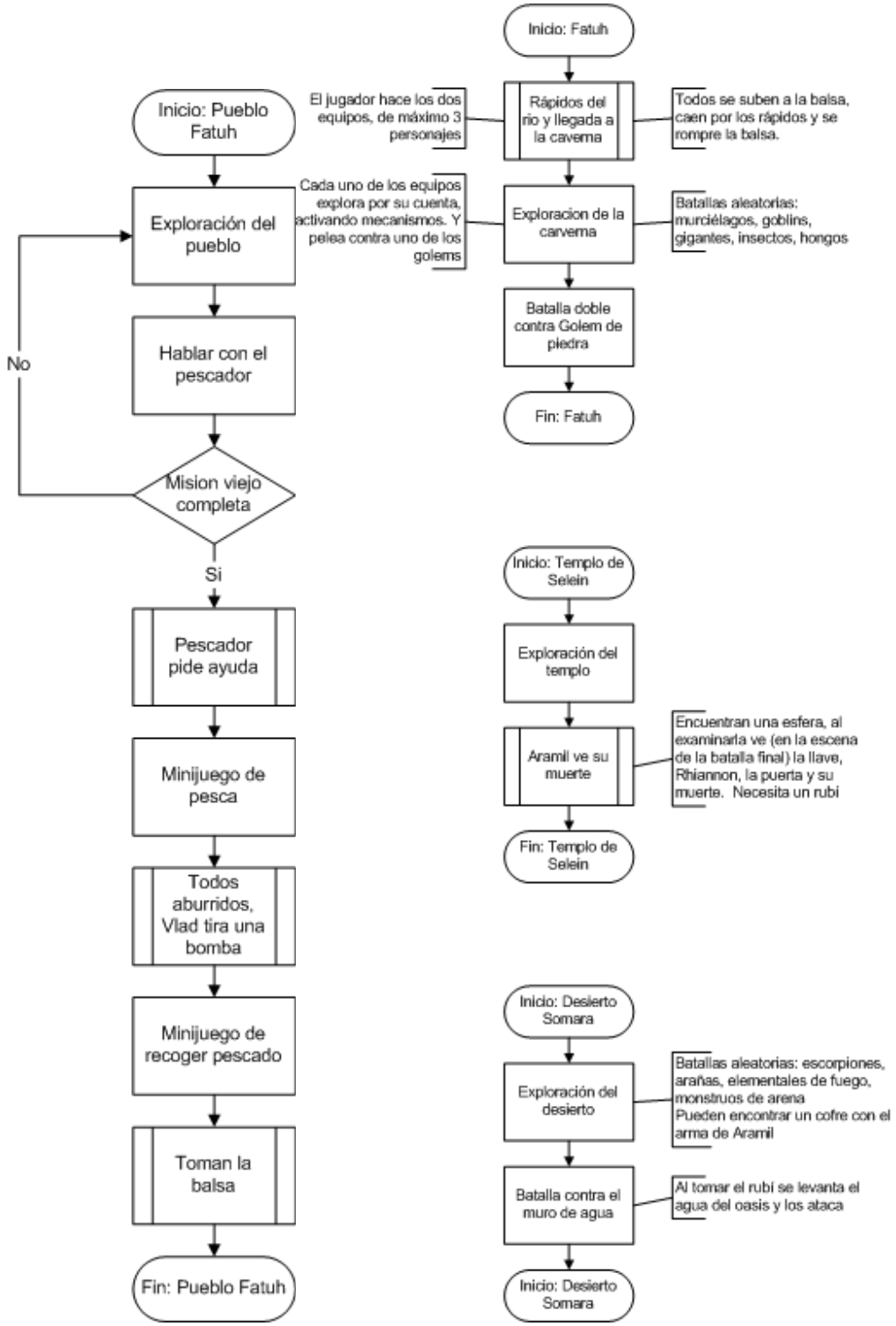

Inicio: Isla pirata
Llegada
Exploración pueblo pirata
Los personajes se deben ocultar: no pasar frente a los piratas, si lo hace, éste los atacará (son nivel jefe)
Exploración caverna
Batallas aleatorias: piratas (nivel normal)
Batalla contra el Capitan Pirata
Toman el barco, huyen
Batalla naval
Minijuego
Vuelta a Broa
Activar bandera mision pirata completa
Fin: Isla pirata
Inicio: Torre de Dalzac
Exploración de la torre
Batallas aleatorias: armaduras y armas animadas, cristales elementales, mini golems
Batalla contra el Golem jefe
Exploración de la habitación de Dalzac
Bombas especiales para Vlad
La llave de los creadores
Regreso a puerto Broa
Los personajes (reformar equipo) deciden ir a buscar la llave de los creadores en todo el continente, si es necesario
Fin: Torre de Dalzac
Fin del Capitulo 2

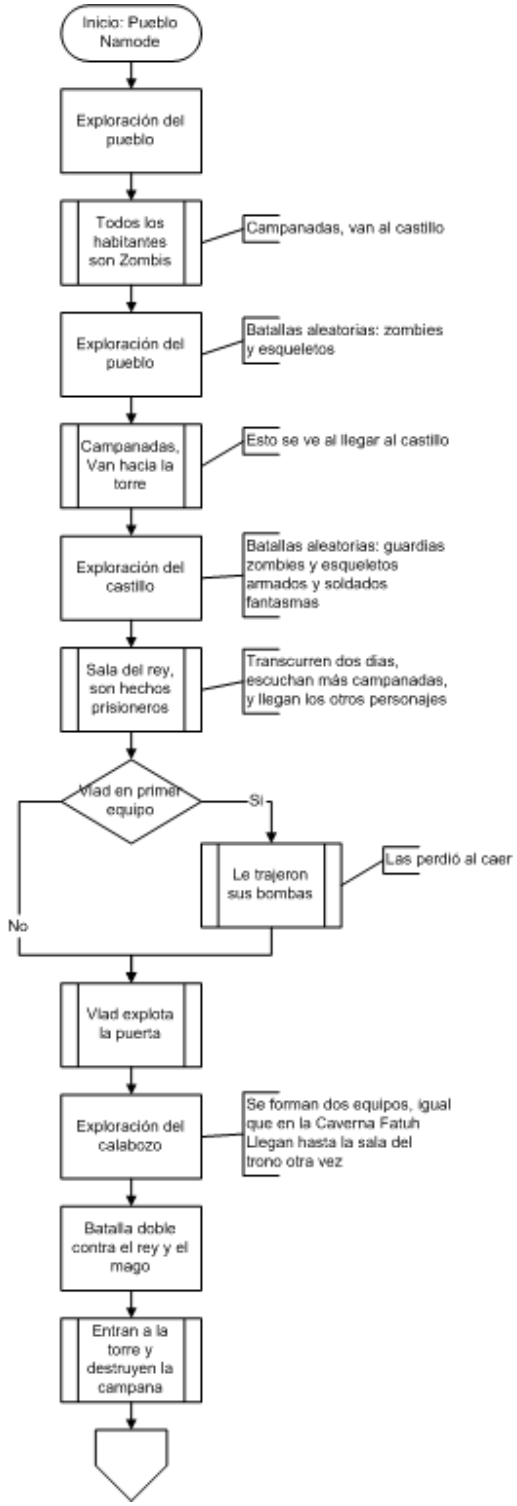

Inicio: Pueblo Marilly
Exploración del pueblo
Habla con el Viejo
Trae planta
No
Si
Información del Viejo
Fin: Pueblo Marilly

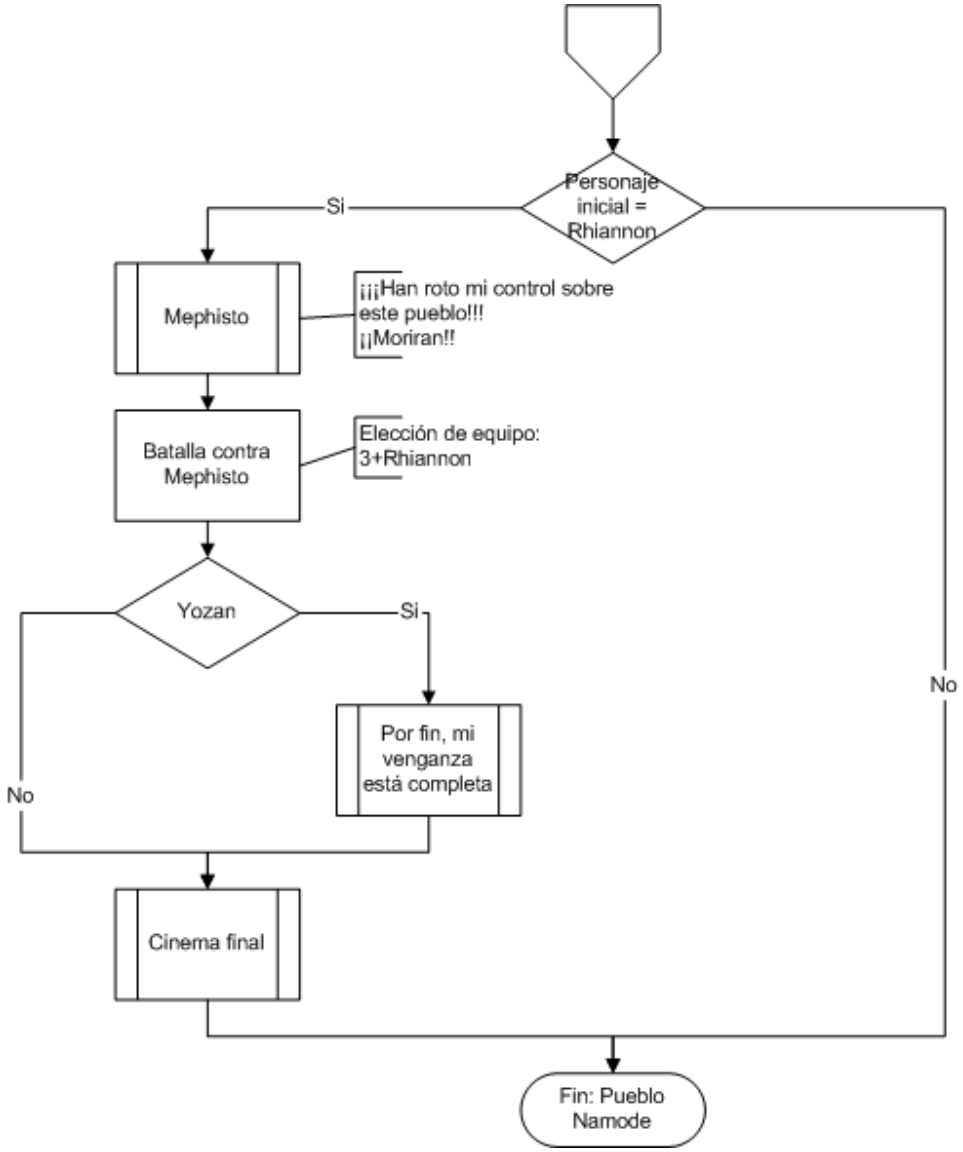

Inicio: Coliseo de Marilly
Exploración del Coliseo
Está Elizabeth
Si
No
La encarcelan, los personajes deben participar para liberarla pero ahora solo son 3
Elizabeth roba el premio
Deciden participar
1er batalla
Celebración
2a batalla
Celebración
3a batalla
Celebración
4a batalla
Entrega de dinero y arma para Elizabeth
Está Elizabeth
Si
No
Pago de fianza
Fin: Coliseo de Marilly

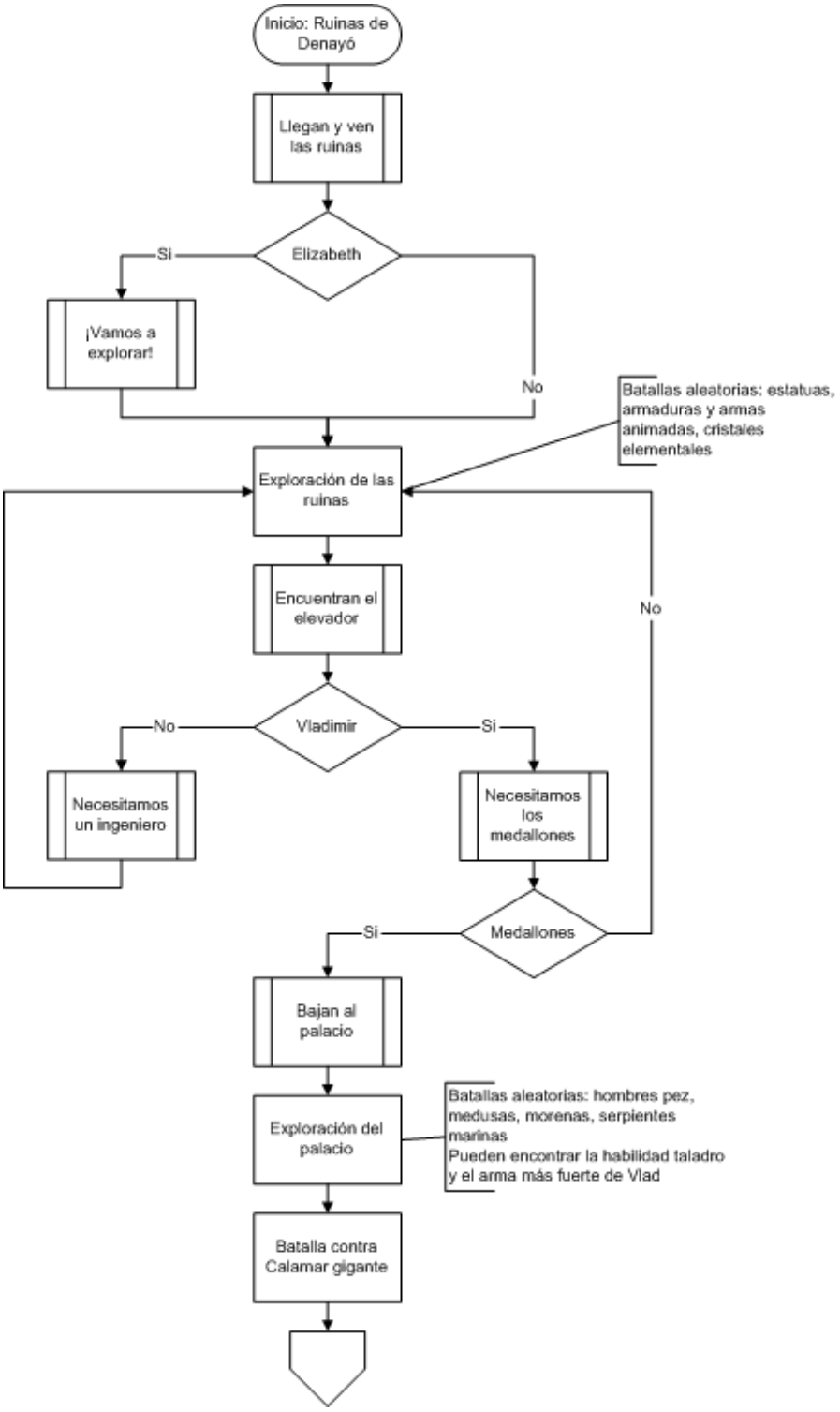

Inicio: Bosque Flammarion
Exploración del bosque
Batallas aleatorias: Hadas, driadas, goblins, duendes
Está Aalok
Si
No
Recuerdos
Tentación del hada
Batalla contra el hada oscura
Obtención de la espada de Aalok
Fin: Bosque Flammarion

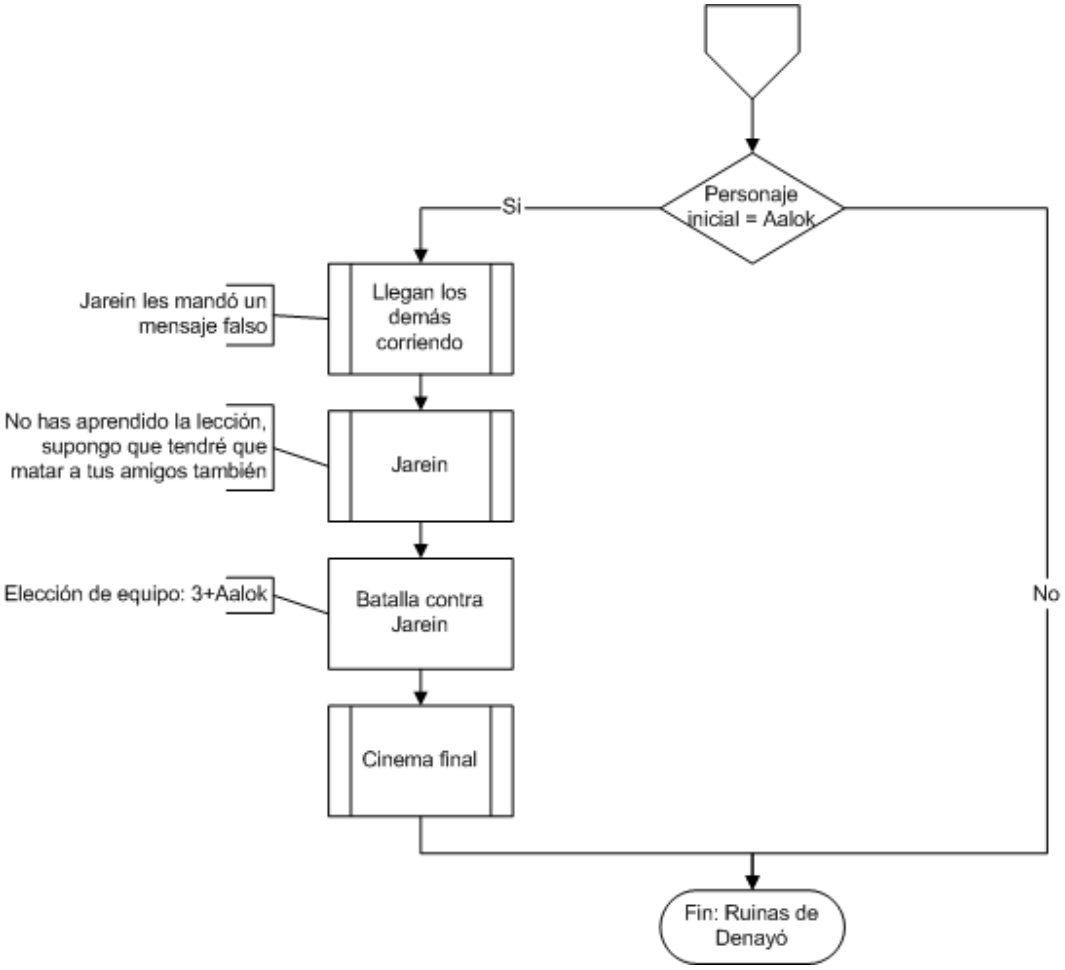

Inicio: Monte Ukert
Exploración del Monte
Batallas aleatorias: aguilas, cabras, serpientes, gigantes, rocas
Batalla rey de los gigantes
Al vencerlo obtienen la planta medicinal
Fin: Monte Ukert

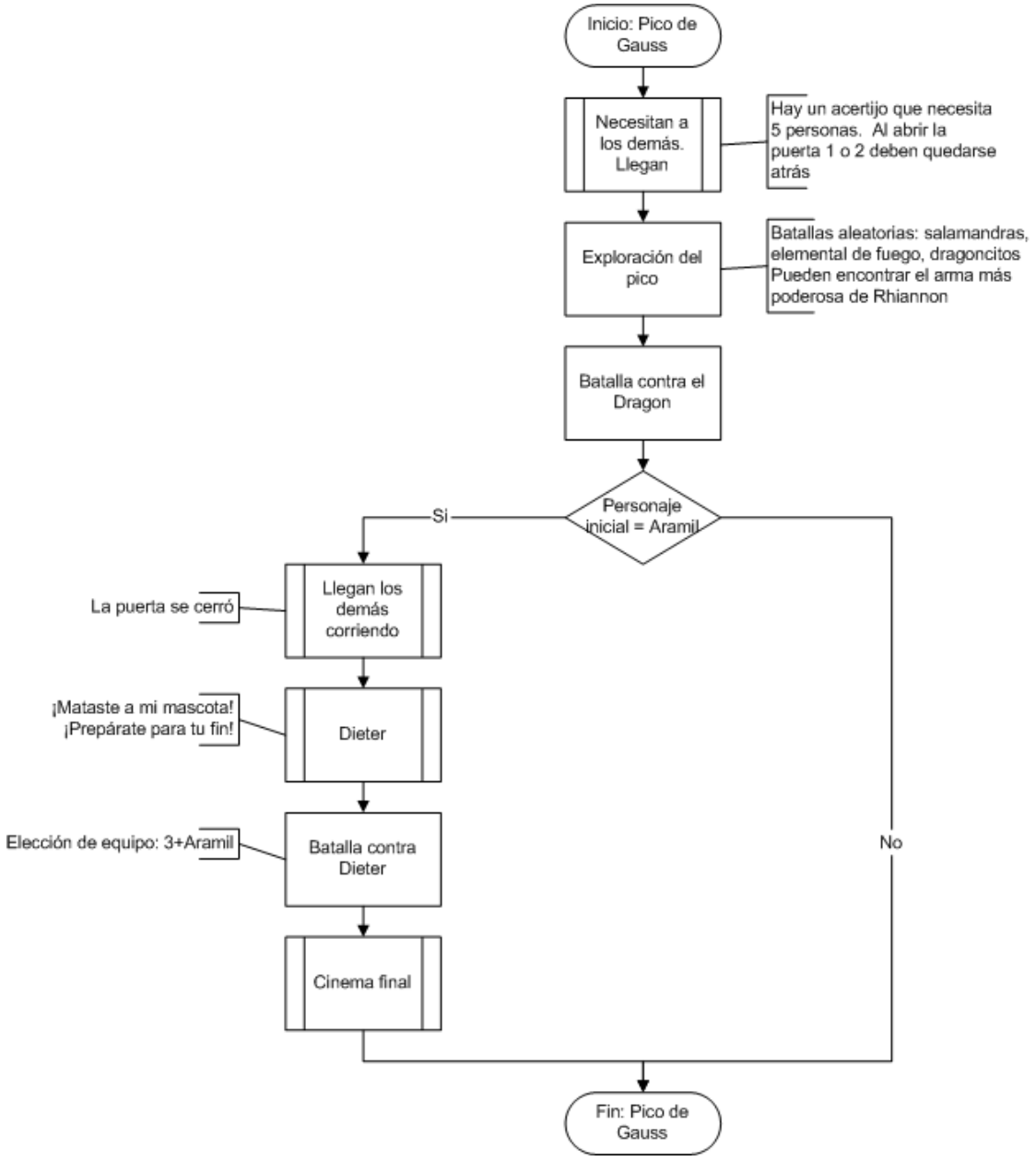

Inicio: Pueblo Fatuh
Exploración del pueblo
Hablar con el pescador
Mision viejo completa
No
Si
Pescador pide ayuda
Minijuego de pesca
Todos aburridos, Vlad tira una bomba
Minijuego de recoger pescado
Toman la balsa
Fin: Pueblo Fatuh
Inicio: Fatuh
Rápidos del rio y llegada a la caverna
El jugador hace los dos equipos, de máximo 3 personajes
Todos se suben a la balsa, caen por los rápidos y se rompre la balsa.
Exploracion de la carverna
Cada uno de los equipos explora por su cuenta, activando mecanismos. Y pelea contra uno de los golems
Batallas aleatorias: murciélagos, goblins, gigantes, insectos, hongos
Batalla doble contra Golem de piedra
Fin: Fatuh
Inicio: Templo de Selein
Exploración del templo
Aramil ve su muerte
Encuentran una esfera, al examinarla ve (en la escena de la batalla final) la llave, Rhiannon, la puerta y su muerte. Necesita un rubí
Fin: Templo de Selein
Inicio: Desierto Somara
Exploración del desierto
Batallas aleatorias: escorpiones, arañas, elementales de fuego, monstruos de arena
Pueden encontrar un cofre con el arma de Aramil
Batalla contra el muro de agua
Al tomar el rubí se levanta el agua del oasis y los ataca
Inicio: Desierto Somara

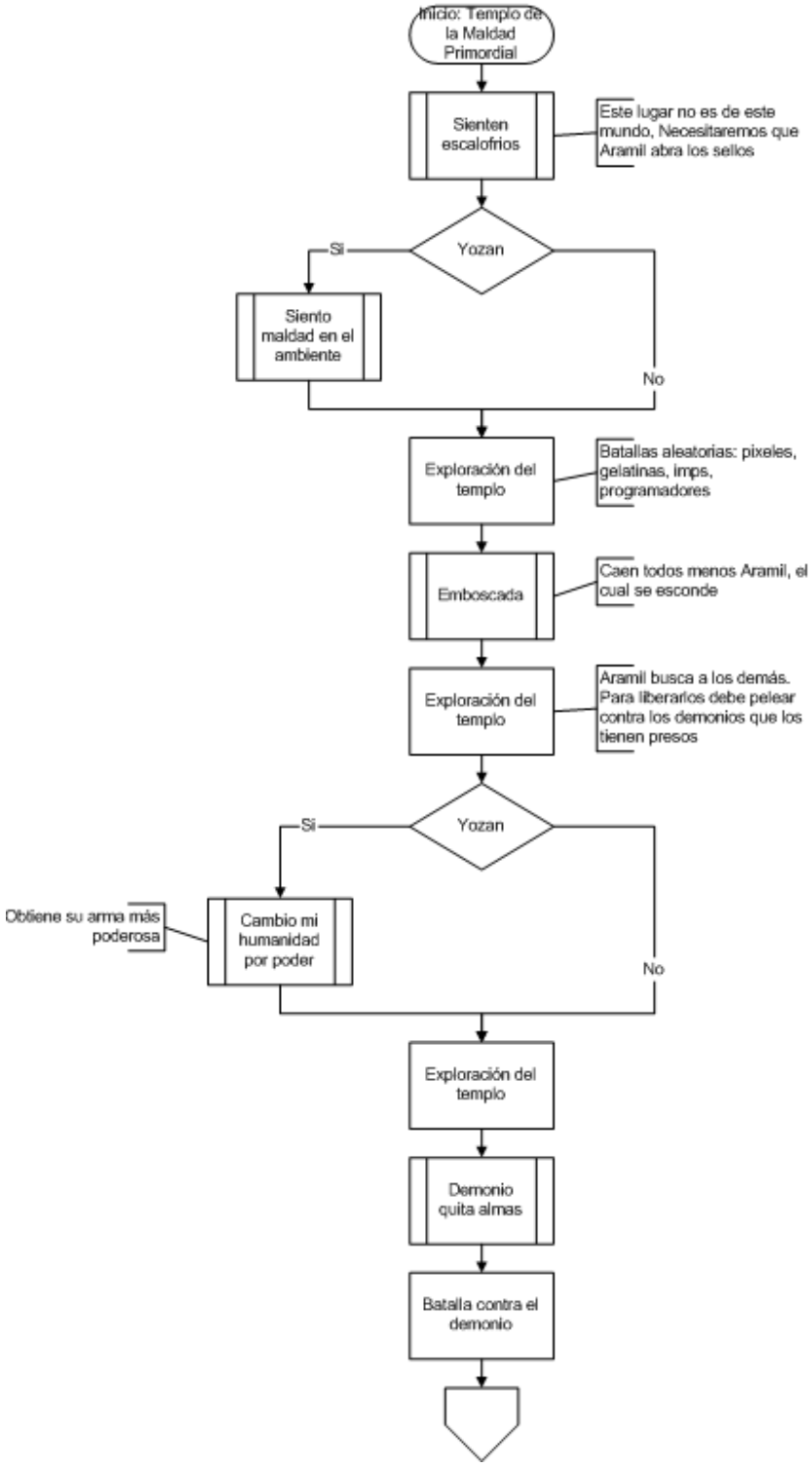

Inicio: Pueblo Namode
Exploración del pueblo
Todos los habitantes son Zombis
Campanadas, van al castillo
Exploración del pueblo
Batallas aleatorias: zombies y esqueletos
Campanadas, Van hacia la torre
Esto se ve al llegar al castillo
Exploración del castillo
Batallas aleatorias: guardias zombies y esqueletos armados y soldados fantasmas
Sala del rey, son hechos prisioneros
Transcurren dos dias, escuchan más campanadas, y llegan los otros personajes
Vlad en primer equipo
Si
Le trajeron sus bombas
Las perdió al caer
No
Vlad explota la puerta
Exploración del calabozo
Se forman dos equipos, igual que en la Caverna Fatuh Llegan hasta la sala del trono otra vez
Batalla doble contra el rey y el mago
Entran a la torre y destruyen la campana

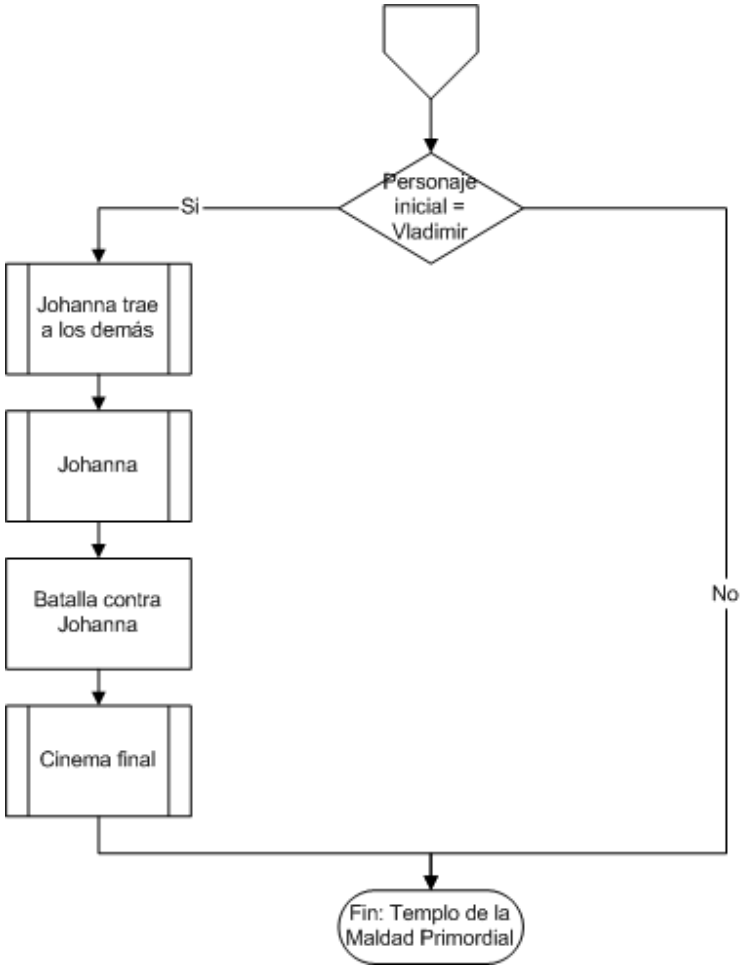

Personaje inicial = Rhiannon
Si
No
Mephisto
¡¡¡Han roto mi control sobre este pueblo!!!
¡¡Moriran!!
Batalla contra Mephisto
Elección de equipo: 3+Rhiannon
Yozan
Si
No
Por fin, mi venganza está completa
Cinema final
Fin: Pueblo Namode


Inicio: Ruinas de Denayó
Llegan y ven las ruinas
Elizabeth
Si
No
¡Vamos a explorar!
Batallas aleatorias: estatuas, armaduras y armas animadas, cristales elementales
Exploración de las ruinas
Encuentran el elevador
Vladimir
No
Si
Necesitamos un ingeniero
Necesitamos los medallones
Medallones
No
Si
Bajan al palacio
Exploración del palacio
Batallas aleatorias: hombres pez, medusas, morenas, serpientes marinas
Pueden encontrar la habilidad taladro y el arma más fuerte de Vlad
Batalla contra Calamar gigante


Personaje inicial = Aalok
Si
No
Jarein les mandó un mensaje falso
Llegan los demás corriendo
No has aprendido la lección, supongo que tendré que matar a tus amigos también
Jarein
Elección de equipo: 3+Aalok
Batalla contra Jarein
Cinema final
Fin: Ruinas de Denayó

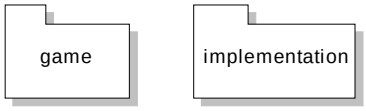

Inicio: Pico de Gauss
Necesitan a los demás. Llegan
Hay un acertijo que necesita 5 personas. Al abrir la puerta 1 o 2 deben quedarse atrás
Exploración del pico
Batallas aleatorias: salamandras, elemental de fuego, dragoncitos Pueden encontrar el arma más poderosa de Rhiannon
Batalla contra el Dragon
Personaje inicial = Aramil
Si
No
Llegan los demás corriendo
La puerta se cerró
Dieter
¡Mataste a mi mascota! ¡Prepárate para tu fin!
Batalla contra Dieter
Elección de equipo: 3+Aramil
Cinema final
Fin: Pico de Gauss


Inicio: Templo de la Maldad Primordial
Sienten escalofrios
Este lugar no es de este mundo, Necesitaremos que Aramil abra los sellos
Yozan
Si
Siento maldad en el ambiente
No
Exploración del templo
Batallas aleatorias: pixeles, gelatinas, imps, programadores
Emboscada
Caen todos menos Aramil, el cual se esconde
Exploración del templo
Aramil busca a los demás. Para liberarlos debe pelear contra los demonios que los tienen presos
Yozan
Si
Obtiene su arma más poderosa
Cambio mi humanidad por poder
No
Exploración del templo
Demonio quita almas
Batalla contra el demonio


Personaje inicial = Vladimir
Si
No
Johanna trae a los demás
Johanna
Batalla contra Johanna
Cinema final
Fin: Templo de la Maldad Primordial

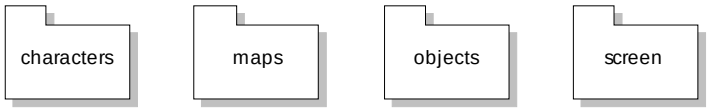
10. APÉNDICE 3: DIAGRAMAS DE CLASES.

10.1. Paquetes



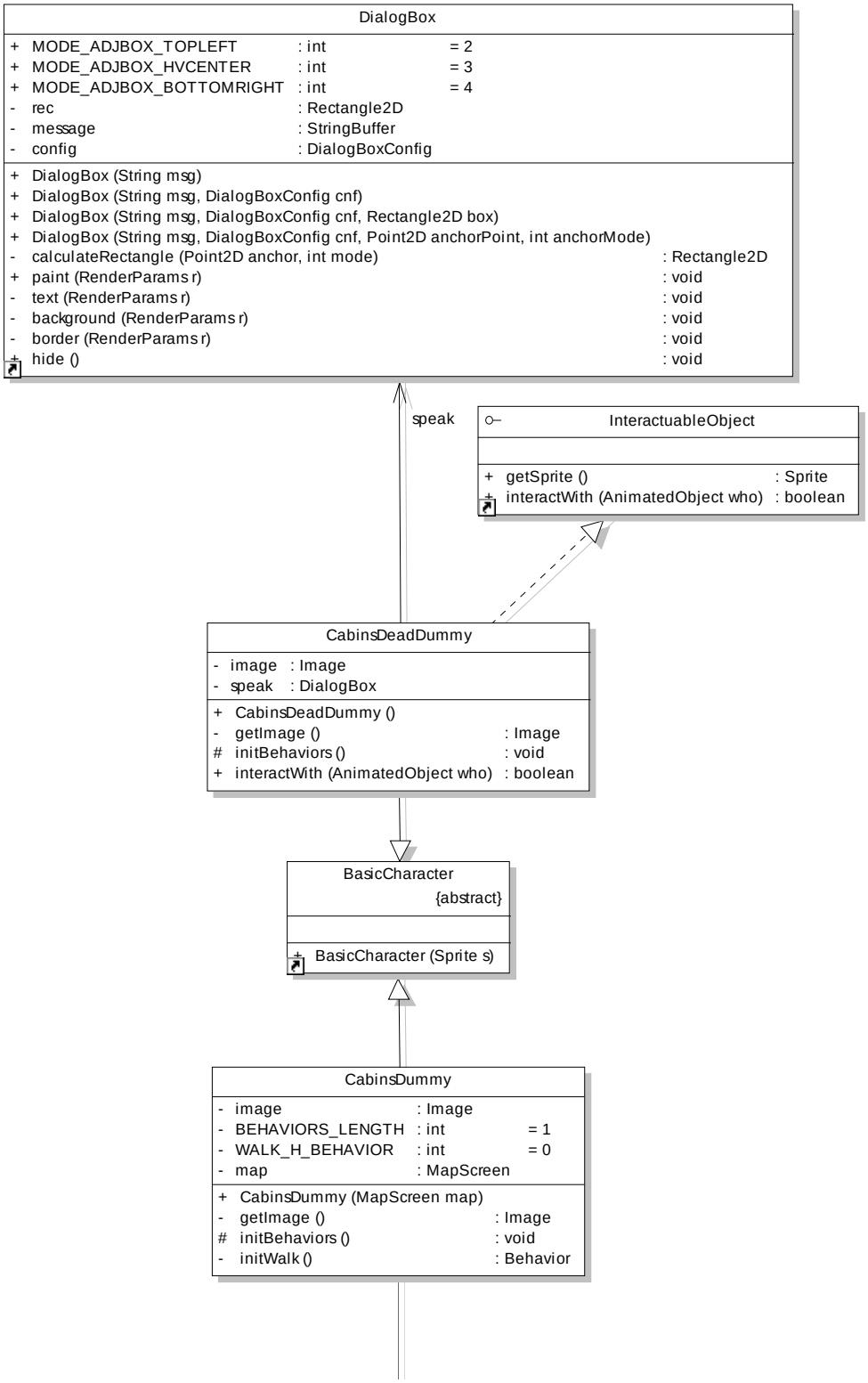
El paquete *implementation* contiene la API de desarrollo del juego, y escapa a los fines de demostración que se persiguieron durante la creación del demo.

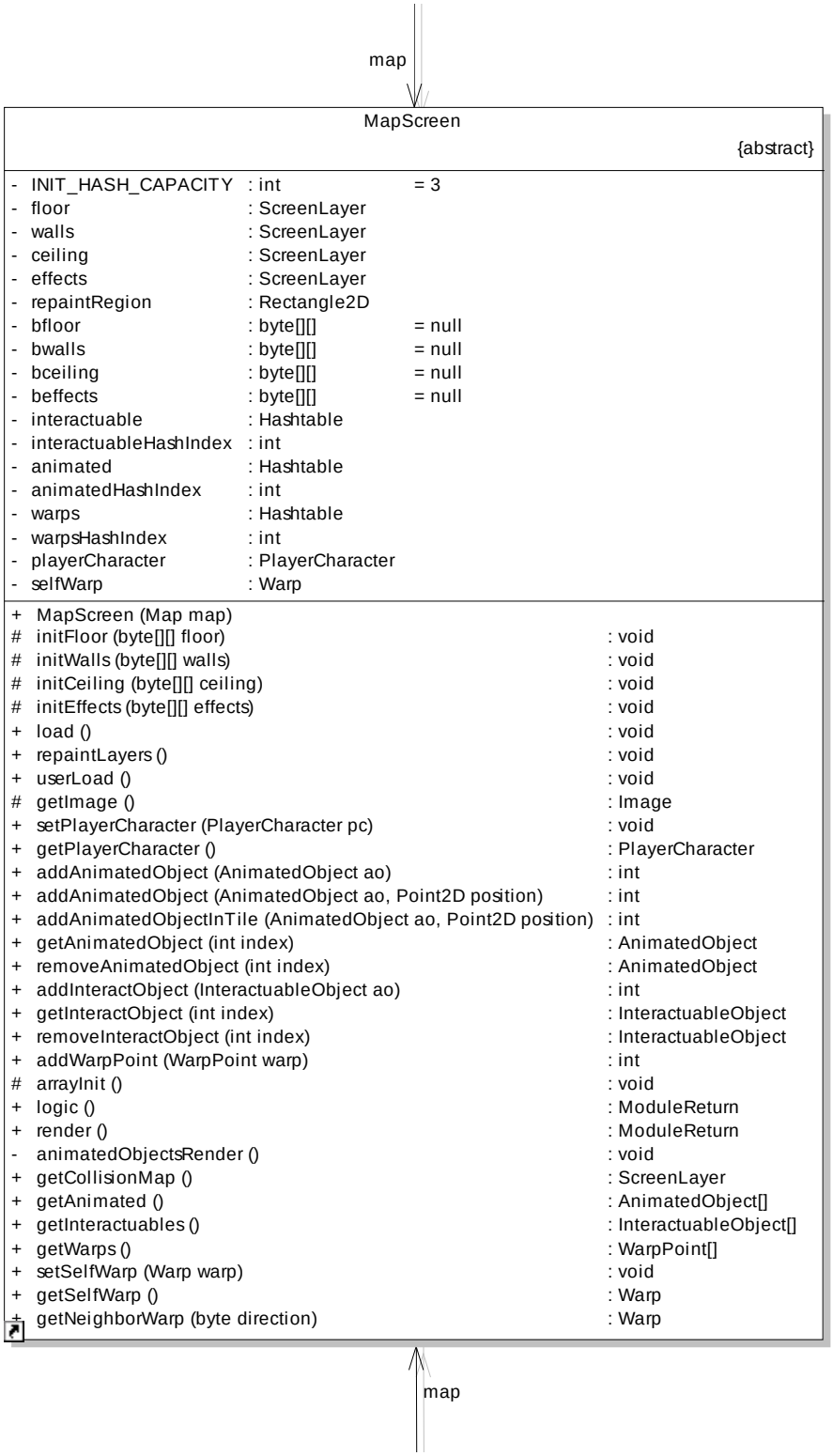
10.2. Paquete *game*

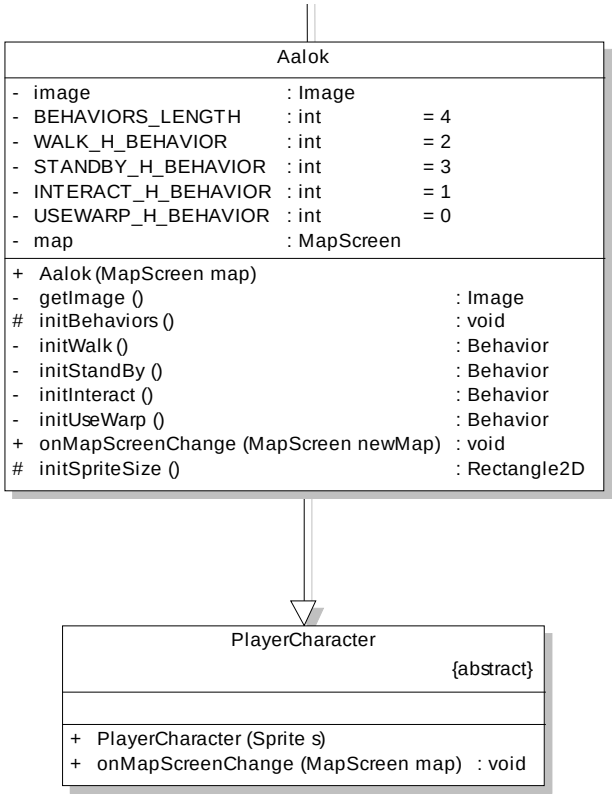


El paquete *game* contiene las clases creadas durante la programación del demo.

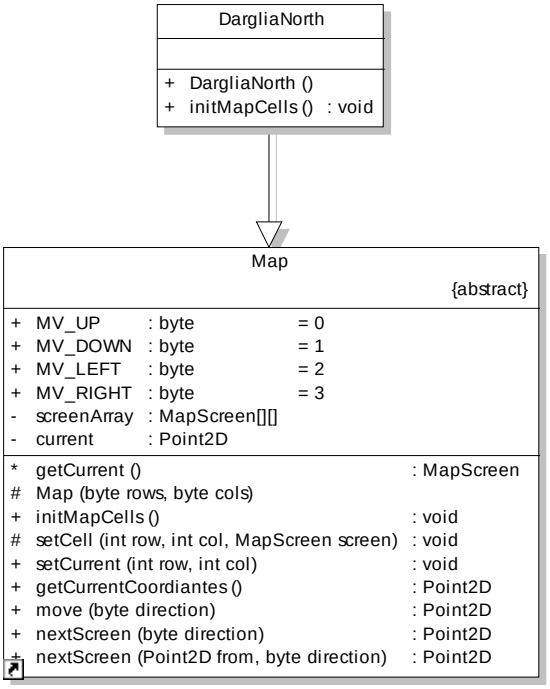
10.3. Clases del paquete *game.characters*



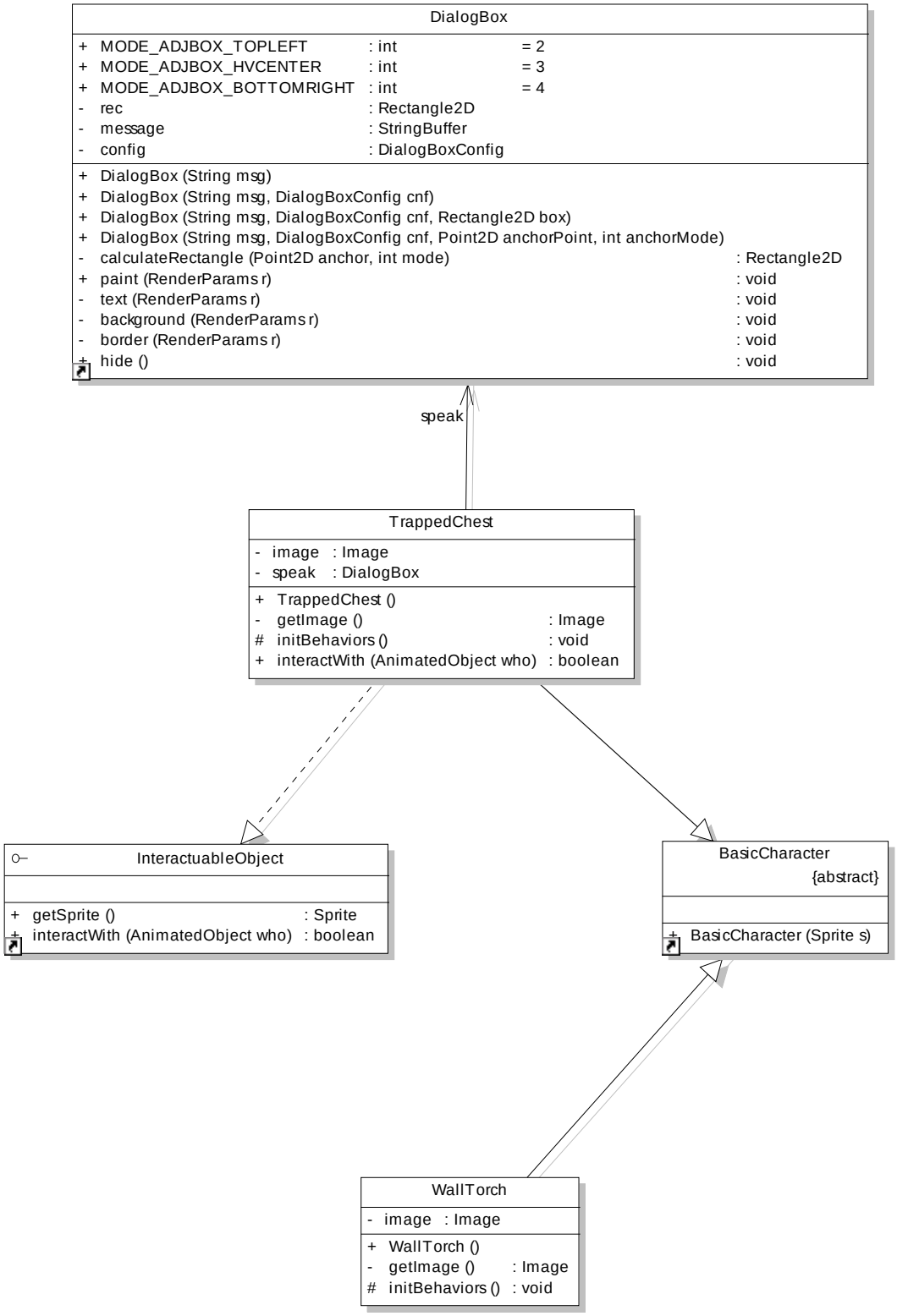

map
MapScreen
{abstract}
- INIT_HASH_CAPACITY : int = 3
- floor : ScreenLayer
- walls : ScreenLayer
- ceiling : ScreenLayer
- effects : ScreenLayer
- repaintRegion : Rectangle2D
- bfloor : byte[][] = null
- bwalls : byte[][] = null
- bceiling : byte[][] = null
- beffects : byte[][] = null
- interactuable : Hashtable
- interactuableHashIndex : int
- animated : Hashtable
- animatedHashIndex : int
- warps : Hashtable
- warpsHashIndex : int
- playerCharacter : PlayerCharacter
- selfWarp : Warp
+ MapScreen (Map map)
# initFloor (byte[][] floor) : void
# initWalls (byte[][] walls) : void
# initCeiling (byte[][] ceiling) : void
# initEffects (byte[][] effects) : void
+ load () : void
+ repaintLayers () : void
+ userLoad () : void
# getImage () : Image
+ setPlayerCharacter (PlayerCharacter pc) : void
+ getPlayerCharacter () : PlayerCharacter
+ addAnimatedObject (AnimatedObject ao) : int
+ addAnimatedObject (AnimatedObject ao, Point2D position) : int
+ addAnimatedObjectInTile (AnimatedObject ao, Point2D position) : int
+ getAnimatedObject (int index) : AnimatedObject
+ removeAnimatedObject (int index) : AnimatedObject
+ addInteractObject (InteractuableObject ao) : int
+ getInteractObject (int index) : InteractuableObject
+ removeInteractObject (int index) : InteractuableObject
+ addWarpPoint (WarpPoint warp) : int
# arrayInit () : void
+ logic () : ModuleReturn
+ render () : ModuleReturn
- animatedObjectsRender () : void
+ getCollisionMap () : ScreenLayer
+ getAnimated () : AnimatedObject[]
+ getInteractuables () : InteractuableObject[]
+ getWarps () : WarpPoint[]
+ setSelfWarp (Warp warp) : void
+ getSelfWarp () : Warp
+ getNeighborWarp (byte direction) : Warp
map



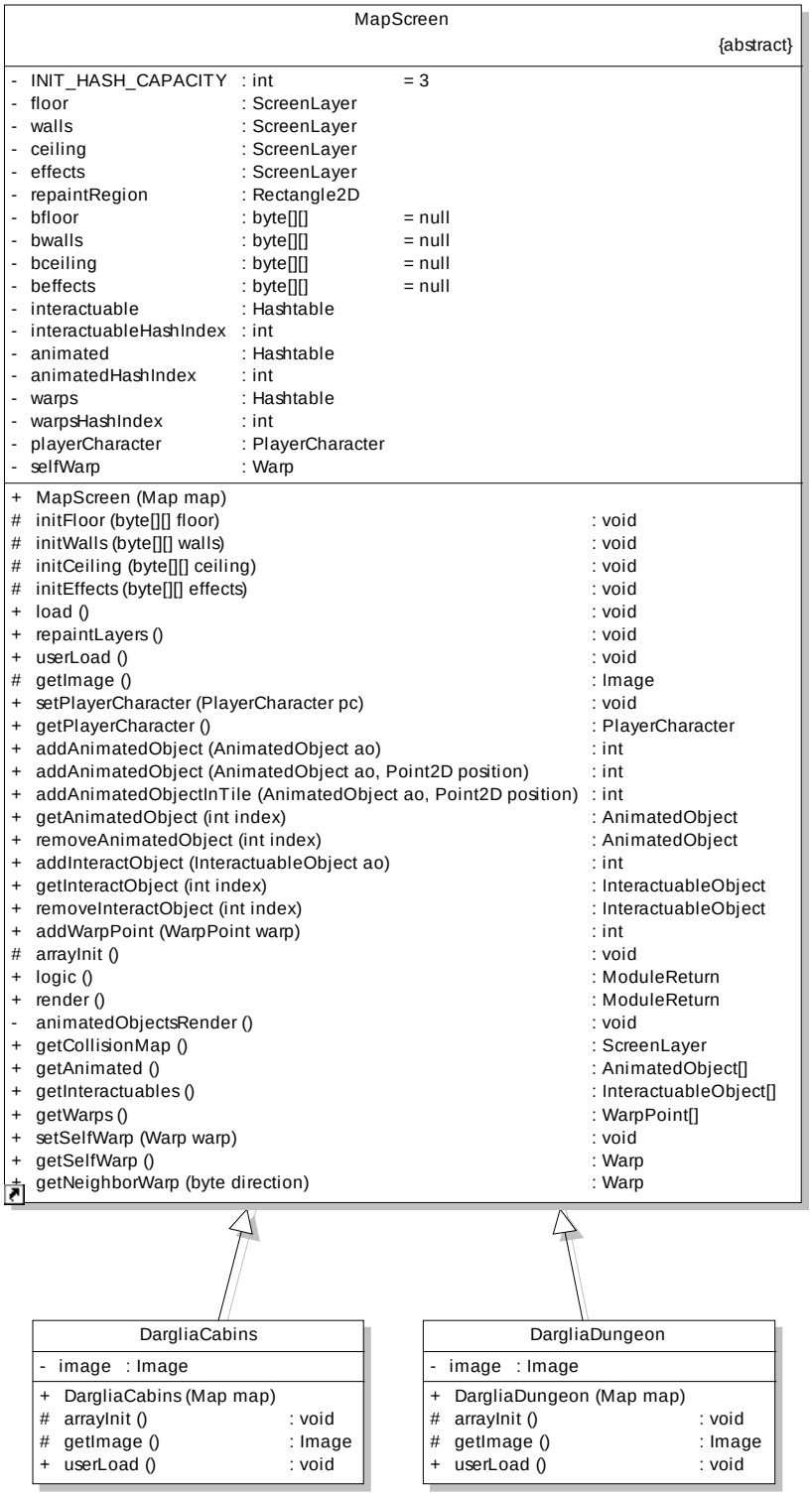
10.4. Clases del paquete *game.maps*



10.5. Clases del paquete *game.objects*



10.6. Clases del paquete game.screen



11. APÉNDICE 4: CÓDIGO DEL DEMO.

11.1. Aalok.java

```
package ldc.game.characters;

import ldc.implementation.AnimatedObject;
import ldc.implementation.Behavior;
import ldc.implementation.screen.MapScreen;
import ldc.implementation.utils.Constants;
import ldc.implementation.utils.geometry.Rectangle2D;

import ldc.implementation.characters.PlayerCharacter;
import ldc.implementation.behaviors.Interact;
import ldc.implementation.behaviors.UseWarps;
import ldc.implementation.behaviors.Walk;
import ldc.implementation.behaviors.StandBy;

import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class Aalok extends PlayerCharacter
{
    private static Image image;
    private final int BEHAVIORS_LENGTH=4;
    private final int WALK_H_BEHAVIOR=2;
    private final int STANDBY_H_BEHAVIOR=3;
    private final int INTERACT_H_BEHAVIOR=1;
    private final int USEWARP_H_BEHAVIOR=0;

    private MapScreen map;

    public Aalok(MapScreen map)
    {
          super(new Sprite(getImage(),Constants.CHAR_FRAME_WIDTH_PX,
```

```
                    Constants.CHAR_FRAME_HEIGHT_PX)
                    );
        this.map=map;
        int stop[]={0}; //Parado
        this.setAnimationSequence(stop);
    }

    private static Image getImage()
    {
        try{
            if(image==null)
                image=Image.createImage("/mc01.png");
            return image;
        }catch (java.io.IOException e)
            {
                System.out.println("No image");
                e.printStackTrace();
            }
        return null;
    }

    protected void initBehaviors()
    {
        Behavior[] hierarchy= new Behavior[BEHAVIORS_LENGTH];
        hierarchy[WALK_H_BEHAVIOR]=initWalk();
        hierarchy[STANDBY_H_BEHAVIOR]=initStandBy();
        hierarchy[INTERACT_H_BEHAVIOR]=initInteract();
        hierarchy[USEWARP_H_BEHAVIOR]=initUseWarp();
        this.behaviors.setHierarchy(hierarchy);
    }

    private Behavior initWalk()
    {
        int[][] walkSequences={
             {0,1}            //WALK_FRONT
            ,{2,3}            //WALK_LEFT
```

```
                ,{4,5}            //WALK_BACK
                ,{6,7}            //WALK_RIGHT
        };
        Walk walk= new Walk(walkSequences,(AnimatedObject)this);
        walk.setMap(map);
        return walk;
    }
    private Behavior initStandBy()
    {
        int[] standBySequence={0,0,9,10,13,14,13,14};
        return new StandBy(standBySequence,this);
    }

    private Behavior initInteract()
    {
        Interact inter=new Interact(this);
        inter.setMap(map);
        return inter;
    }

    private Behavior initUseWarp()
    {
        UseWarps uw= new UseWarps(this);
        uw.setMap(map);
        return uw;
    }

    public void onMapScreenChange(MapScreen newMap) {
        this.map=newMap;
        Walk walk =

    (Walk)behaviors.getHeriarchyBehavior(WALK_H_BEHAVIOR);
        walk.setMap(this.map);
        Interact inter =

    (Interact)behaviors.getHeriarchyBehavior(INTERACT_H_BEHAVIOR);
```

```
            inter.setMap(this.map);
            UseWarps uw =

    (UseWarps)behaviors.getHeriarchyBehavior(USEWARP_H_BEHAVIOR);
            uw.setMap(this.map);
            walk.reset();
    }
    protected Rectangle2D initSpriteSize() {
            return new
Rectangle2D(0,0,(int)Constants.CHAR_FRAME_WIDTH_PX
                        ,(int)Constants.CHAR_FRAME_HEIGHT_PX);
    }
}
```

11.2. CabinsDeadDummy.java

```
package ldc.game.characters;

import ldc.implementation.AnimatedObject;
import ldc.implementation.InteractuableObject;
import ldc.implementation.characters.BasicCharacter;
import ldc.implementation.screen.gui.DialogBox;
import
ldc.implementation.screen.gui.profiles.standard.STDDialogBoxConfig;
import ldc.implementation.utils.Constants;
import ldc.implementation.utils.SharedResources;
import ldc.implementation.utils.geometry.Point2D;
import ldc.implementation.utils.geometry.Rectangle2D;

import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class CabinsDeadDummy extends BasicCharacter implements
          InteractuableObject {
    private static Image image;
    private DialogBox speak;

    public CabinsDeadDummy(){
          super(new Sprite(getImage(),Constants.CHAR_FRAME_WIDTH_PX,
                      Constants.CHAR_FRAME_HEIGHT_PX));
          int deadAnimation[]={11};
          this.setAnimationSequence(deadAnimation);
    }

    private static Image getImage()
    {
          try{
                if(image==null)
                      image=Image.createImage("/mc09.png");
                return image;
          }catch (java.io.IOException e)
```

```
                {
                      System.out.println("No image");
                      e.printStackTrace();
                }
          return null;
    }

    protected void initBehaviors() {
    }

    public boolean interactWith(AnimatedObject who)
    {
          if(speak==null)
          {
          speak=new DialogBox("¡Black Mage! ¡Esta muerto!"
                      ,new STDDialogBoxConfig()
                      ,new Rectangle2D(80,0,160,48));
          }

          speak.paint(SharedResources.getRenderParams());

          if(SharedResources.getLogicParams().getInput().B())
          {
                speak.hide();
                return true;
          }
          return false;
    }
}
```

11.3. CabinsDummy.java

```
package ldc.game.characters;

import ldc.implementation.AnimatedObject;
import ldc.implementation.Behavior;
import ldc.implementation.behaviors.Walk;
import ldc.implementation.characters.BasicCharacter;
import ldc.implementation.screen.MapScreen;
import ldc.implementation.utils.Constants;
import ldc.implementation.utils.Path;

import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class CabinsDummy extends BasicCharacter {
    private static Image image;

    private final int BEHAVIORS_LENGTH=1;
    private final int WALK_H_BEHAVIOR=0;

    private MapScreen map;

    public CabinsDummy(MapScreen map){
          super(new Sprite(getImage(),Constants.CHAR_FRAME_WIDTH_PX,
                      Constants.CHAR_FRAME_HEIGHT_PX)
                      );
          this.map=map;
          initBehaviors();
    }

    private static Image getImage()
    {
          try{
                if(image==null)
                      image=Image.createImage("/fc01.png");
                return image;
```

```
            }catch (java.io.IOException e)
                  {
                        System.out.println("No image");
                        e.printStackTrace();
                  }
            return null;
      }

      protected void initBehaviors() {
            Behavior[] hierarchy= new Behavior[BEHAVIORS_LENGTH];
            hierarchy[WALK_H_BEHAVIOR]=initWalk();
            this.behaviors.setHierarchy(hierarchy);
      }

      private Behavior initWalk()
      {
            int[][] walkSequences={
                   {0,1}            //WALK_FRONT
                  ,{2,3}            //WALK_LEFT
                  ,{4,5}            //WALK_BACK
                  ,{6,7}            //WALK_RIGHT
            };
            Walk walk= new Walk(walkSequences,(AnimatedObject)this);
            walk.setMap(map);
            byte[] bpath={
                  Walk.RIGHT, Walk.RIGHT, Walk.RIGHT, Walk.RIGHT,
                  Walk.RIGHT, Walk.RIGHT, Walk.LEFT, Walk.LEFT,
                  Walk.LEFT, Walk.LEFT, Walk.LEFT, Walk.LEFT};
            Path path= new Path(bpath,true);
            walk.setPath(path);
            walk.usePath(true);
            return walk;
      }
}
```

11.4. DargliaNorth.java

```
package ldc.game.maps;

import ldc.game.screen.DargliaCabins;
import ldc.game.screen.DargliaDungeon;
import ldc.implementation.Map;

public class DargliaNorth extends Map {

    public DargliaNorth()
    {
          super((byte)2,(byte)1);
    }

    public void initMapCells() {
          setCell(0,0,new DargliaCabins(this));
          setCell(1,0,new DargliaDungeon(this));
          setCurrent(0,0);
    }
}
```

11.5. TrappedChest.java

```
package ldc.game.objects;

import ldc.implementation.AnimatedObject;
import ldc.implementation.InteractuableObject;
import ldc.implementation.characters.BasicCharacter;
import ldc.implementation.screen.gui.DialogBox;
import
ldc.implementation.screen.gui.profiles.standard.STDDialogBoxConfig;
import ldc.implementation.utils.Constants;
import ldc.implementation.utils.SharedResources;
import ldc.implementation.utils.geometry.Point2D;
import ldc.implementation.utils.geometry.Rectangle2D;

import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class TrappedChest extends BasicCharacter implements
          InteractuableObject {
    private static Image image;
    private DialogBox speak;
    public TrappedChest(){
          super(new Sprite(getImage(),Constants.CHAR_FRAME_WIDTH_PX,
                      Constants.CHAR_FRAME_HEIGHT_PX)
                      );
          int closed[]={60};
          this.setAnimationSequence(closed);
    }

    private static Image getImage()
    {
          try{
                if(image==null)
                      image=Image.createImage("/castle1.png");
                return image;
          }catch (java.io.IOException e)
```

```
                {
                      System.out.println("No image");
                      e.printStackTrace();
                }
          return null;
    }

    protected void initBehaviors() {
    }

    public boolean interactWith(AnimatedObject who) {
          if(speak==null)
          {
          speak=new DialogBox("Mejor no lo abro, puede tener
trampas..."
                      ,new STDDialogBoxConfig()
                      ,new Rectangle2D(0,0,240,48));
          }
          speak.paint(SharedResources.getRenderParams());
          if(SharedResources.getLogicParams().getInput().B())
          {
                speak.hide();
                return true;
          }
          return false;
    }
}
```

11.6. WallTorch.java

```
package ldc.game.objects;

import ldc.implementation.AnimatedObject;
import ldc.implementation.characters.BasicCharacter;
import ldc.implementation.utils.Constants;
import ldc.implementation.utils.SharedResources;

import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class WallTorch extends BasicCharacter {
    private static Image image;

    public WallTorch(){
          super(new Sprite(getImage(),Constants.CHAR_FRAME_WIDTH_PX,
                      Constants.CHAR_FRAME_HEIGHT_PX)
                      );
          int torchAnimation[]={50,51,52,53,52,51};
          this.setAnimationSequence(torchAnimation);
    }

    private static Image getImage()
    {
          try{
                if(image==null)
                      image=Image.createImage("/castle1.png");
                return image;
          }catch (java.io.IOException e)
                {
                      System.out.println("No image");
                      e.printStackTrace();
                }
          return null;
    }
```

```
    protected void initBehaviors() {
    }
}
```

11.7. DargliaCabins.java

```
package ldc.game.screen;

import ldc.implementation.Map;
import ldc.implementation.Warp;
import ldc.implementation.WarpPoint;
import ldc.implementation.screen.MapScreen;
import ldc.implementation.utils.Constants;
import ldc.implementation.utils.geometry.Point2D;

import ldc.game.characters.Aalok;
import ldc.game.characters.CabinsDeadDummy;
import ldc.game.characters.CabinsDummy;
import ldc.game.objects.TrappedChest;
import ldc.game.objects.WallTorch;

import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class DargliaCabins extends MapScreen
{
    private Image image;
    public DargliaCabins(Map map)
    {super(map);}
    protected void arrayInit()
    {
    /**Inicializacion de los arreglos de las capas*/
    byte[][] tmpfloor = {
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 3, 0, 4, 7, 4, 0, 3, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0,10,10,11,10,11,10,10, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0,10,10,10,10,10,10,10, 0,10,10,10, 0, 0},
    { 0, 0, 0, 0, 0, 0,10,10,10,10,10,10,10, 0,10,10,10, 0, 0},
    { 0, 0, 0, 0, 0, 0,10,10,10,10,10,10,10, 0,10,16,10, 0, 0},
    { 0, 0, 0, 0, 0, 0,10,10,10,10,10,10,10, 0, 2, 2, 2, 0, 0},
```

```
{ 0, 0, 0, 0, 0, 0,10,10,10,10,10,10, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 2,10,10,10, 2, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0,10,10,10, 0, 0, 0,10,10,10, 0, 0, 0,10,16,10, 0, 0, 0},
{ 0, 0,10,10,10, 0, 0, 0,10,10,10, 0, 0, 0,10,10,10, 0, 0, 0},
{ 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0},
{ 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0},
{ 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0},
{ 0, 2, 2, 2, 2, 2, 2, 2,10,10,10, 2, 2, 2, 2, 2, 2, 2, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0,10,10,10, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
byte[][] tmpwalls = {
{ 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,13, 3, 5,38, 5, 3,13, 2, 3,51, 3, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,14, 3,46, 0,46, 3,14, 2, 3, 3, 3, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2,15, 0, 0, 0, 0, 0,15, 2,61,62,61, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 1, 1},
{ 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 1, 1},
{ 1, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 1, 1},
{ 1, 2, 3,51, 3, 2,13, 0, 0, 0, 0, 0,13, 2, 3,51, 3, 2, 1, 1},
{ 2, 2, 3, 3, 3, 2, 2, 0, 0, 0, 0, 0, 2, 2, 3, 3, 3, 2, 2, 1},
{ 2, 3, 0, 0, 0, 3,50, 3, 0, 0, 0, 3,50, 3, 0, 0, 0, 3, 2, 1},
{ 2, 3, 0, 0, 0, 3, 3, 3, 0, 0, 0, 3, 3, 3, 0, 0, 0, 3, 2, 1},
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1},
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1},
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1},
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1},
{ 3, 3, 3, 3, 3, 3, 3, 2, 0, 0, 0, 2, 3, 3, 3, 3, 3, 3, 3, 1},
{ 3, 3, 3, 3, 3, 3, 3, 2, 0, 0, 0, 2, 3, 3, 3, 3, 3, 3, 3, 1},
{ 1, 1, 1, 1, 1, 1, 1, 3, 3, 0, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 3, 3,38, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1}
};
byte[][] tmpceiling = {
```

```
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    };
    byte[][] tmpeffects=new
byte[Constants.MAP_TILE_ROWS][Constants.MAP_TILE_COLS];

        this.initFloor(tmpfloor);
        this.initWalls(tmpwalls);
        this.initCeiling(tmpceiling);
        this.initEffects(tmpeffects);

        /*Agregamos los objetos animados*/
        Aalok aalok=new Aalok(this);
        aalok.setTilePosition(9,2);

        CabinsDeadDummy cdd=new CabinsDeadDummy();
        cdd.setTilePosition(15,4);
        this.addAnimatedObject(cdd);
```

```
          TrappedChest tc1=new TrappedChest();
          tc1.setTilePosition(14,3);
          this.addAnimatedObject(tc1);

          TrappedChest tc2=new TrappedChest();
          tc2.setTilePosition(16,3);
          this.addAnimatedObject(tc2);

          this.addAnimatedObjectInTile(new CabinsDummy(this),new
Point2D(6,12));

          this.addAnimatedObjectInTile(new WallTorch(),new
Point2D(15,1));
          this.addAnimatedObjectInTile(new WallTorch(),new
Point2D(3,8));
          this.addAnimatedObjectInTile(new WallTorch(),new
Point2D(15,8));

          /*Agregamos los objetos interactuables*/
          this.addInteractObject(cdd);
          this.addInteractObject(tc1);
          this.addInteractObject(tc2);

          /*Establecemos cual sera el personaje que el usuario
controlara*/
          this.setPlayerCharacter(aalok);

          /*Agregamos los objetos warp*/
          //Warp(map, screen(renglon,columna), position.x,
position.y)
          Warp warp1 = new Warp(getSelfWarp().getMap(), new
Point2D(0,1),6,10);
          Warp warp2 = new Warp(getSelfWarp().getMap(), new
Point2D(0,1),11,10);
          //Creamos el elemnto grafico del warp
```

```
            Sprite warpSprite1 = new
  Sprite(getImage(),Constants.MAP_TILE_WIDTH_PX

      ,Constants.MAP_TILE_HEIGHT_PX);
            Sprite warpSprite2 = new
  Sprite(getImage(),Constants.MAP_TILE_WIDTH_PX

      ,Constants.MAP_TILE_HEIGHT_PX);
            WarpPoint wp1= new WarpPoint(warp1, warpSprite1);
            WarpPoint wp2= new WarpPoint(warp2, warpSprite2);
            //Establecemos donde quedara el warp dentro del mapa
            wp1.setTilePosition(15,10);
            wp2.setTilePosition(15,6);
            wp1.setActive(true);
            wp2.setActive(true);
            // Definimos la secuencia de animación que utilizara
            int[] animation={15};
            wp1.setAnimationSequence(animation);
            wp2.setAnimationSequence(animation);
            //Agregamos el warp
            this.addWarpPoint(wp1);
            this.addWarpPoint(wp2);
      }

      protected Image getImage()
      {
            try
            {
                  if(this.image==null)
                        this.image=Image.createImage("/castle1.png");
                  return image;

            } catch (java.io.IOException e)
            {
                  System.out.println("No image");
                  e.printStackTrace();
```

```
            }
            return null;
      }

      public void userLoad() {
      }
  }
```

11.8. DargliaDungeon.java

```
package ldc.game.screen;

import ldc.implementation.Map;
import ldc.implementation.Warp;
import ldc.implementation.WarpPoint;
import ldc.implementation.screen.MapScreen;
import ldc.implementation.utils.Constants;
import ldc.implementation.utils.geometry.Point2D;

import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class DargliaDungeon extends MapScreen
{
    private Image image;
    public DargliaDungeon(Map map)
    {super(map);}
    protected void arrayInit() {
    /**Inicializacion de los arreglos de las capas*/
    byte[][] tmpfloor = {
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0},
    { 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0},
    { 0, 0, 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0, 0},
    { 0, 0, 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0, 0},
    { 0, 0, 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0, 0},
    { 0, 0, 0,10,10,10,10,17,10,10,10,10,17,10,10,10,10, 0, 0, 0},
    { 0, 0, 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0, 0},
    { 0, 0, 0,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0, 0},
    { 0, 0, 0,10, 0,10,10,10,10,10,10,10,10,10,10, 0,10, 0, 0, 0},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
byte[][] tmpwalls = {
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1},
{ 1, 1, 2, 3,13, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,13, 3, 2, 1, 1},
{ 1, 1, 2, 3,14, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,14, 3, 2, 1, 1},
{ 1, 1, 2,60,15,60, 0, 0, 0, 0, 0, 0, 0, 0,58,15,58, 2, 1, 1},
{ 1, 1, 2,60,60,60,60, 0, 0, 0, 0, 0, 0, 0,58,58,58, 2, 1, 1},
{ 1, 1, 2,60,60,60, 0, 0, 0, 0, 0, 0, 0, 0, 0,58,58, 2, 1, 1},
{ 1, 1, 2, 0,60, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,58, 2, 1, 1},
{ 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1},
{ 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1},
{ 1, 1, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 1, 1},
{ 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1},
{ 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
};
byte[][] tmpceiling = {
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0,13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,13, 0, 0, 0, 0},
{ 0, 0, 0, 0,14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,14, 0, 0, 0, 0},
{ 0, 0, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
byte[][] tmpeffects = {
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

```
            this.initFloor(tmpfloor);
            this.initWalls(tmpwalls);
            this.initCeiling(tmpceiling);
            this.initEffects(tmpeffects);

            /*Agregamos los objetos warp*/
            //Warp(map, screen(renglon,columna), position.x,
position.y)
            Warp warp1 = new Warp(getSelfWarp().getMap(), new
Point2D(0,0),14,10);
            Warp warp2 = new Warp(getSelfWarp().getMap(), new
Point2D(0,0),14,6);
            //Creamos el elemnto grafico del warp
            Sprite warpSprite1= new
Sprite(getImage(),Constants.MAP_TILE_WIDTH_PX
      ,Constants.MAP_TILE_HEIGHT_PX);
            Sprite warpSprite2= new
Sprite(getImage(),Constants.MAP_TILE_WIDTH_PX

      ,Constants.MAP_TILE_HEIGHT_PX);
            WarpPoint wp1= new WarpPoint(warp1, warpSprite1);
            WarpPoint wp2= new WarpPoint(warp2, warpSprite2);
            //Establecemos donde quedara el warp dentro del mapa
            wp1.setTilePosition(7,10);
            wp2.setTilePosition(12,10);
            wp1.setActive(true);
            wp2.setActive(true);
            // Definimos la secuencia de animación que utilizara
            int[] animation={16};
            wp1.setAnimationSequence(animation);
            wp2.setAnimationSequence(animation);
            //Agregamos el warp
            this.addWarpPoint(wp1);
            this.addWarpPoint(wp2);
      }
```

```
    protected Image getImage()
    {
          try
          {
                if(this.image==null)
                      this.image=Image.createImage("/castle1.png");
                return image;

          } catch (java.io.IOException e)
          {
                System.out.println("No image");
                e.printStackTrace();
          }
          return null;
    }

    public void userLoad() {
    }
}
```

12. APÉNDICE 5: ARTE CONCEPTUAL E IMPLEMENTACIÓN.

Mapa de Zar'an

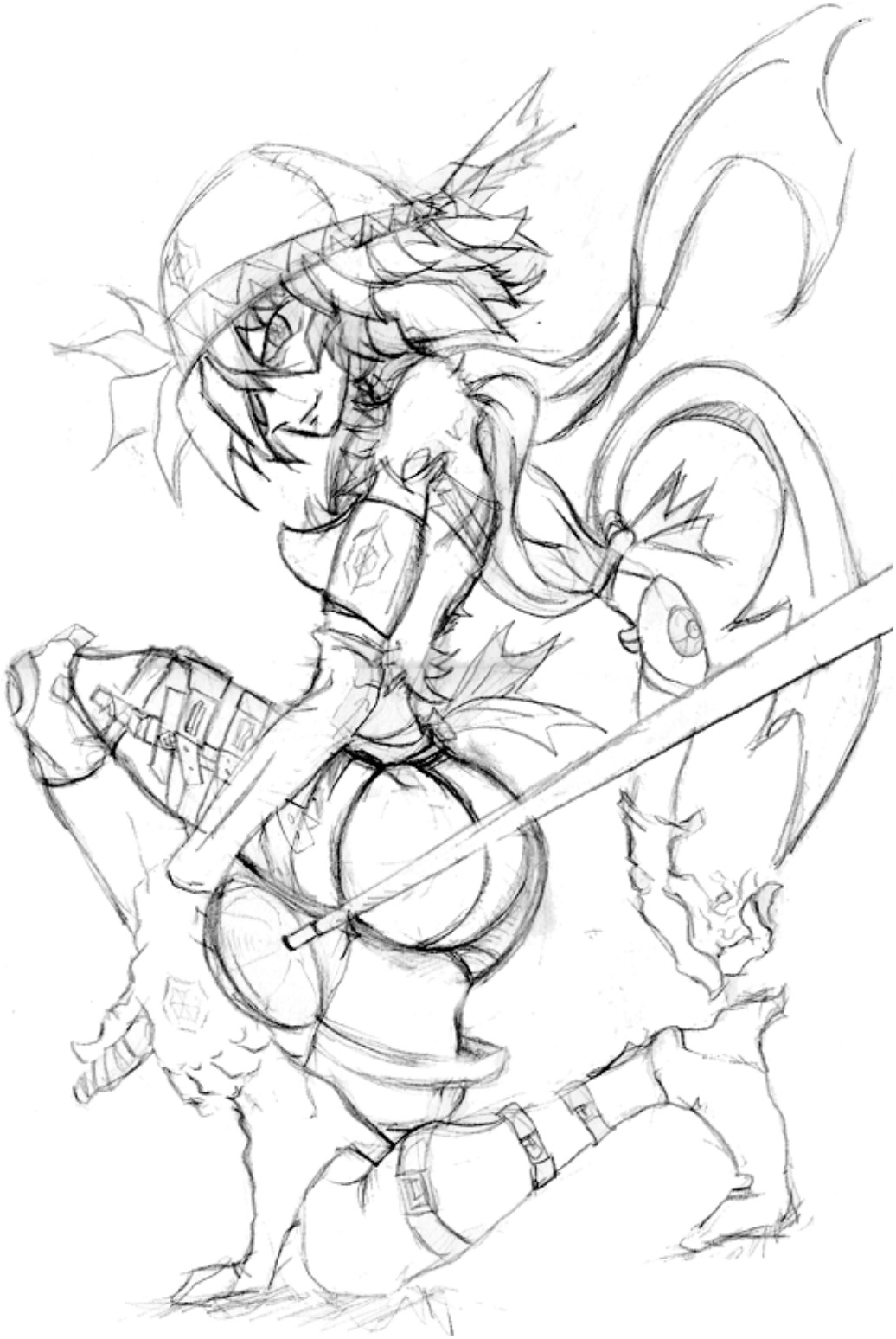
Rhiannon

Yozan

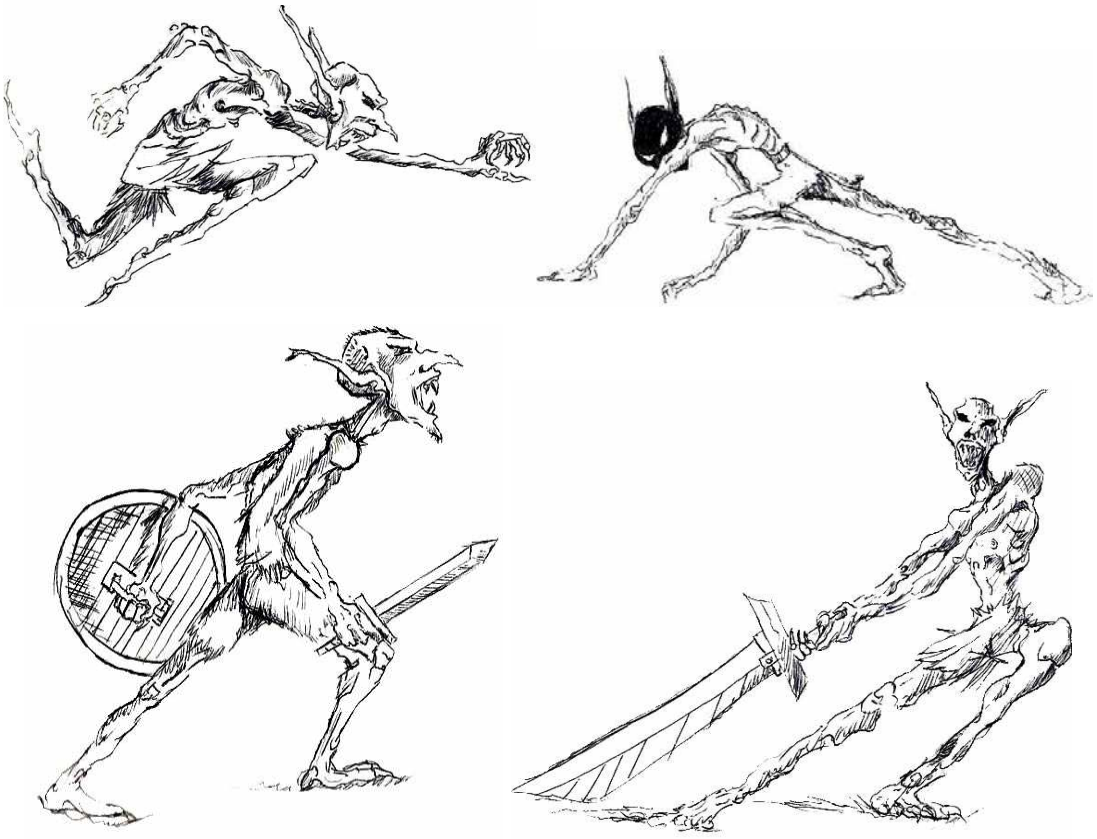
Sprites de Rhiannon y Yozan

Vladimir

Aalok



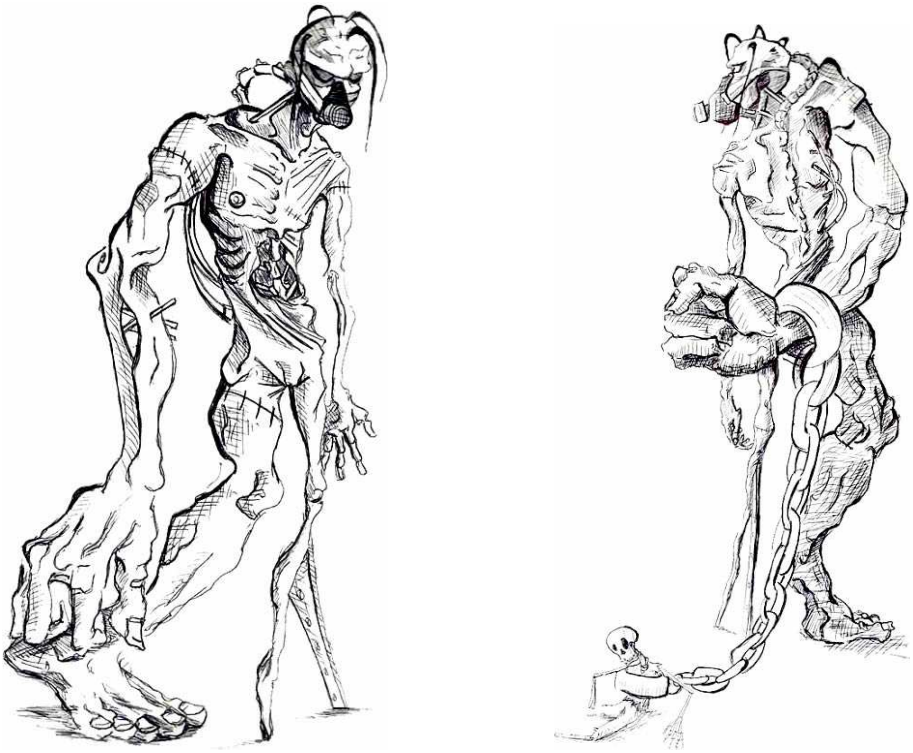
Elizabeth



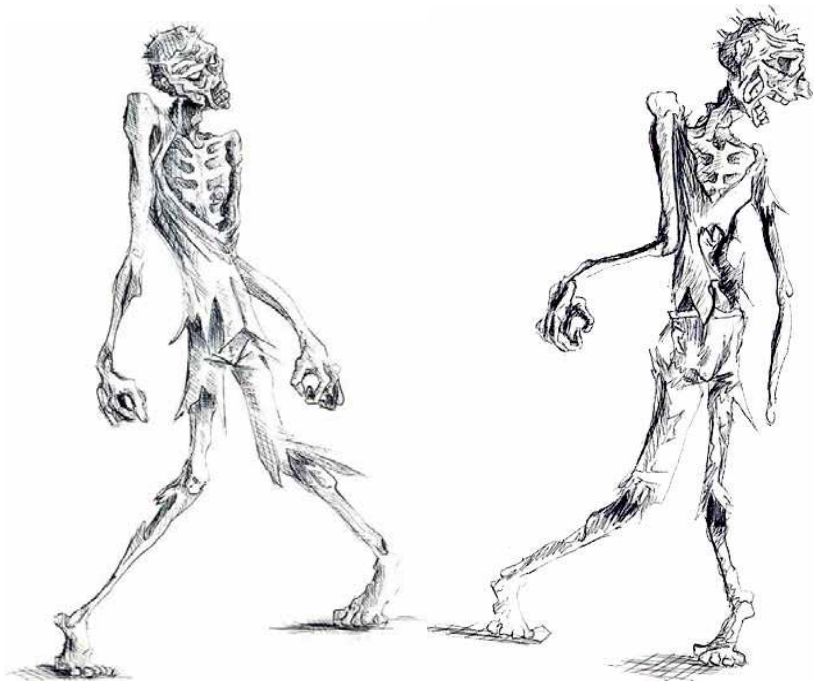
Goblin

Goblin mago

Goblin guerrero

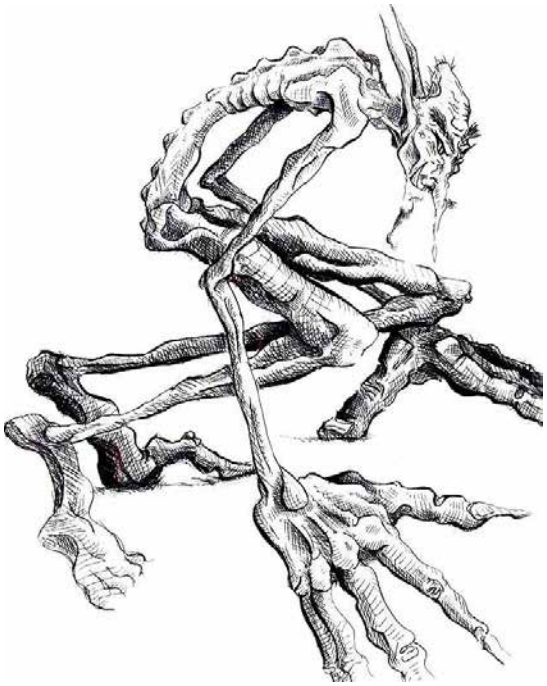


Gólem

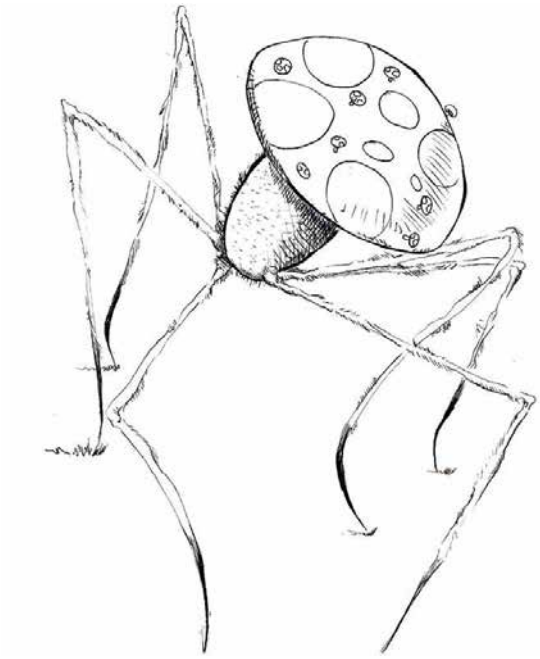


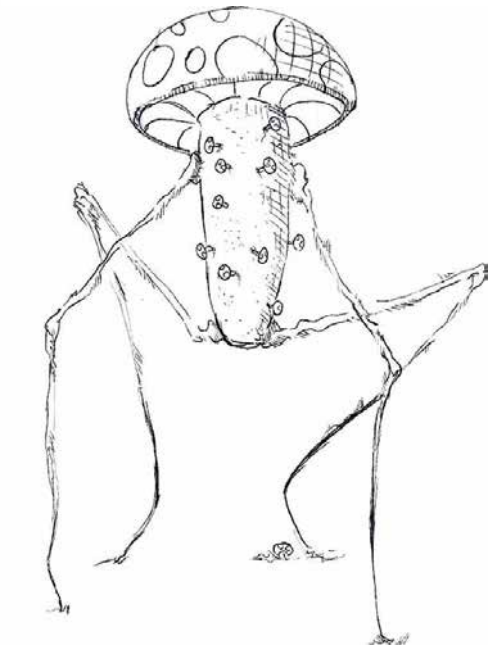
Zombi

Elemental de fuego

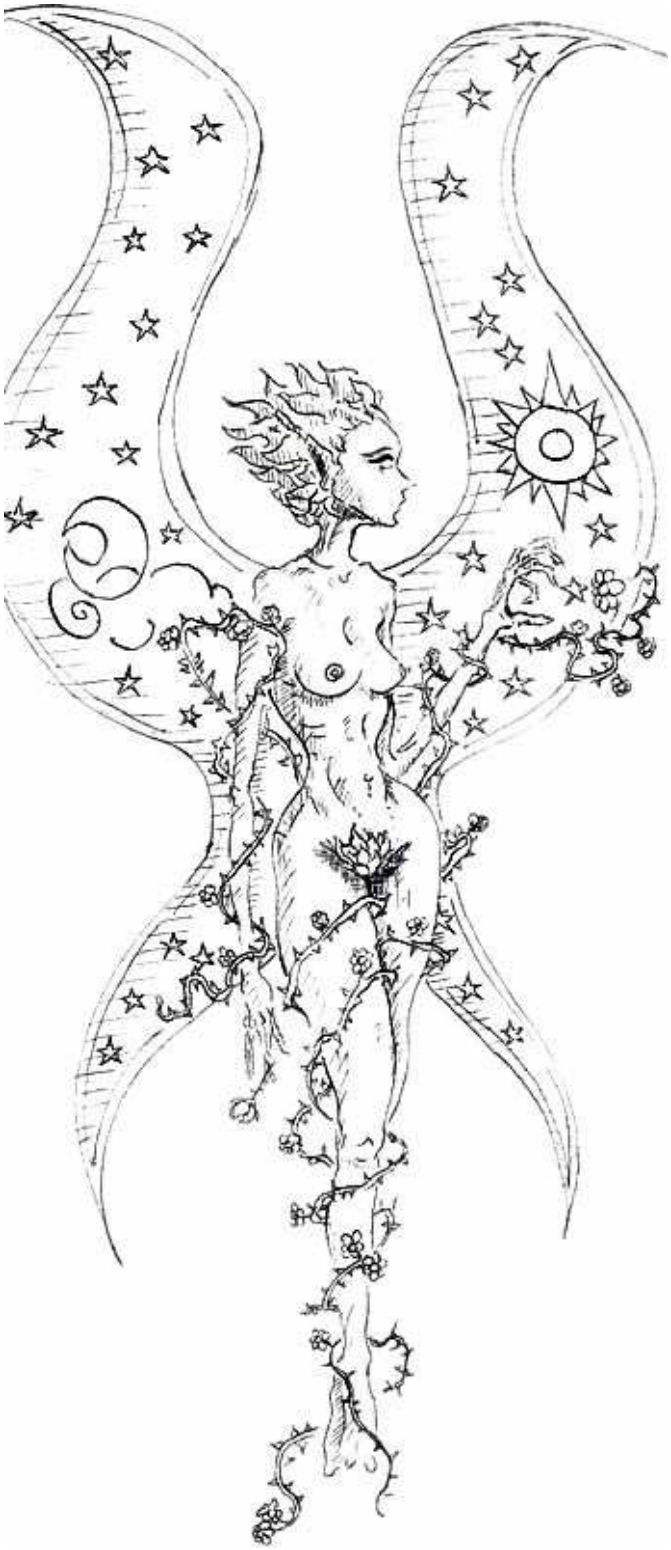


Troll





Hongo



Hada oscura

Hada

Esqueleto

Rostro de Rhiannon

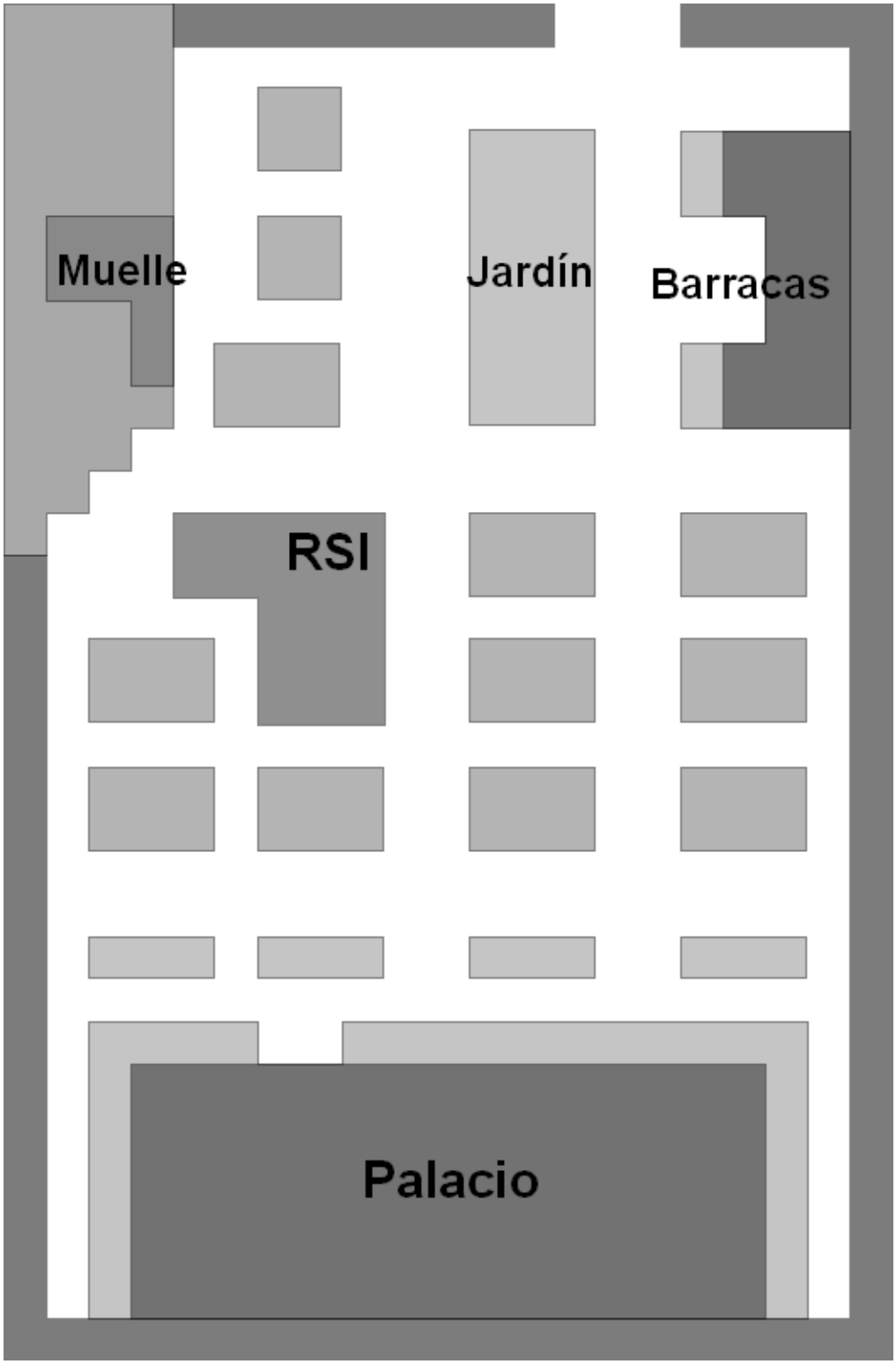
Créditos

Iniciar juego

Elección de personaje principal

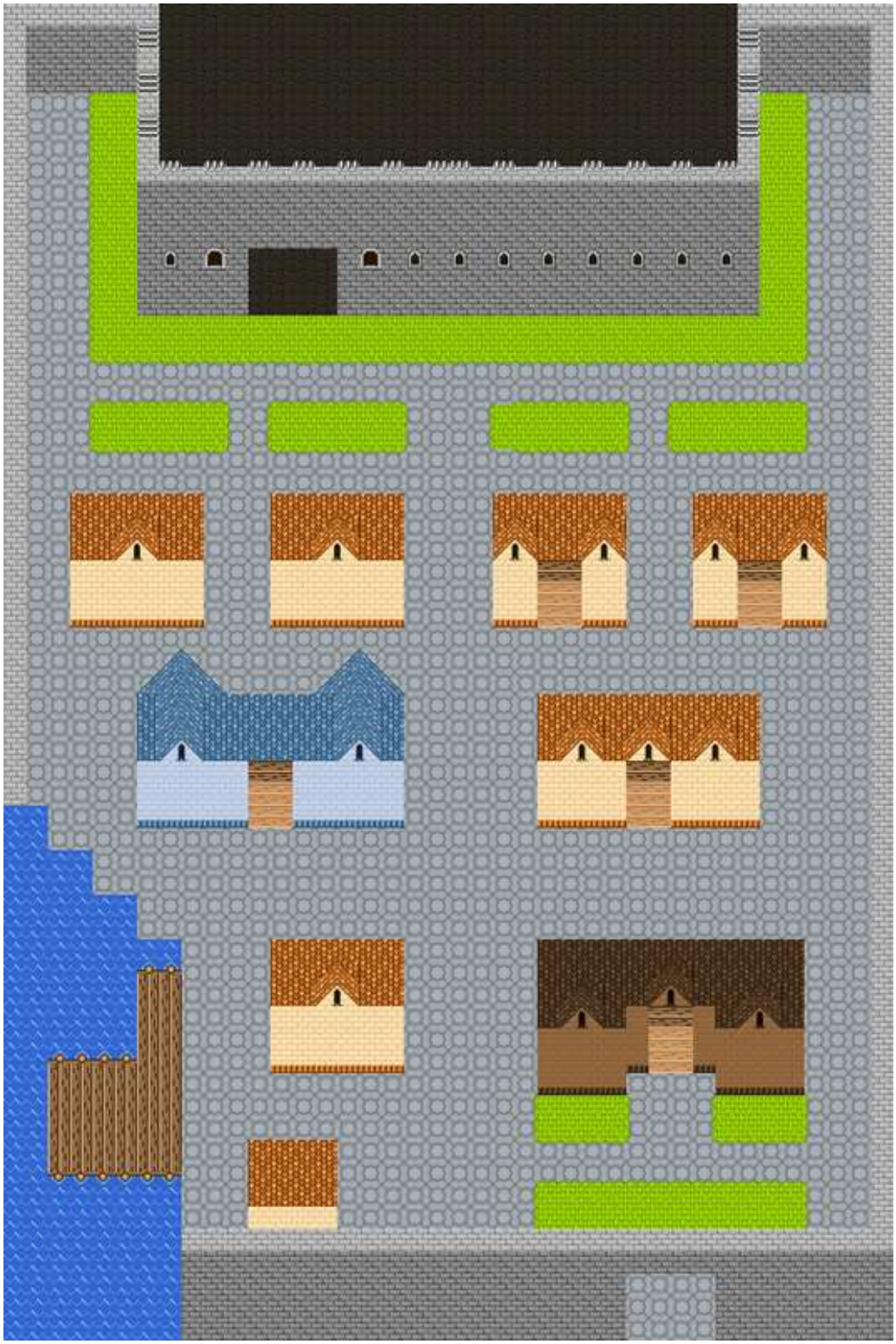
Batalla

Propuestas de pantallas

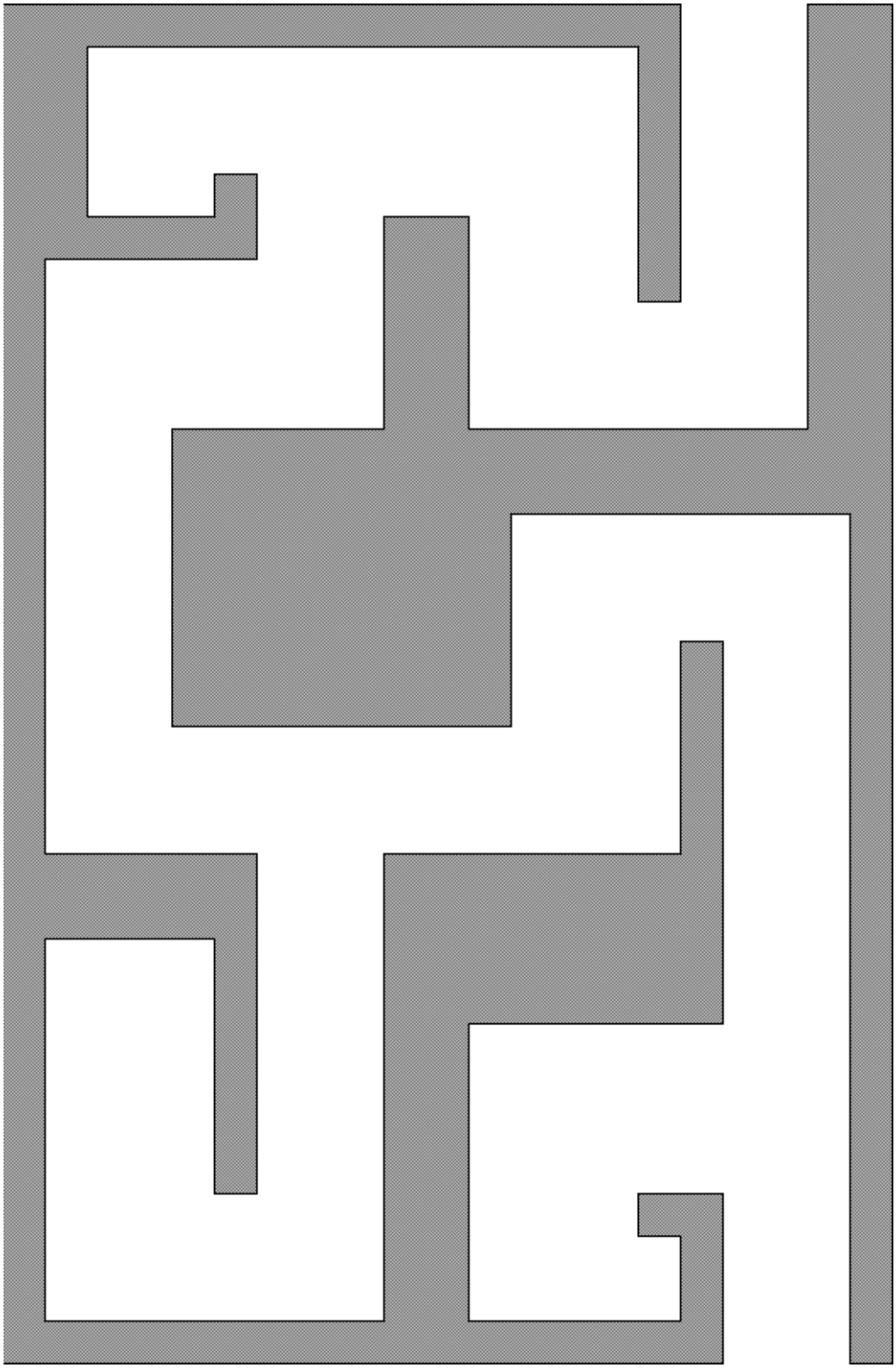


Mapa de Darglia.

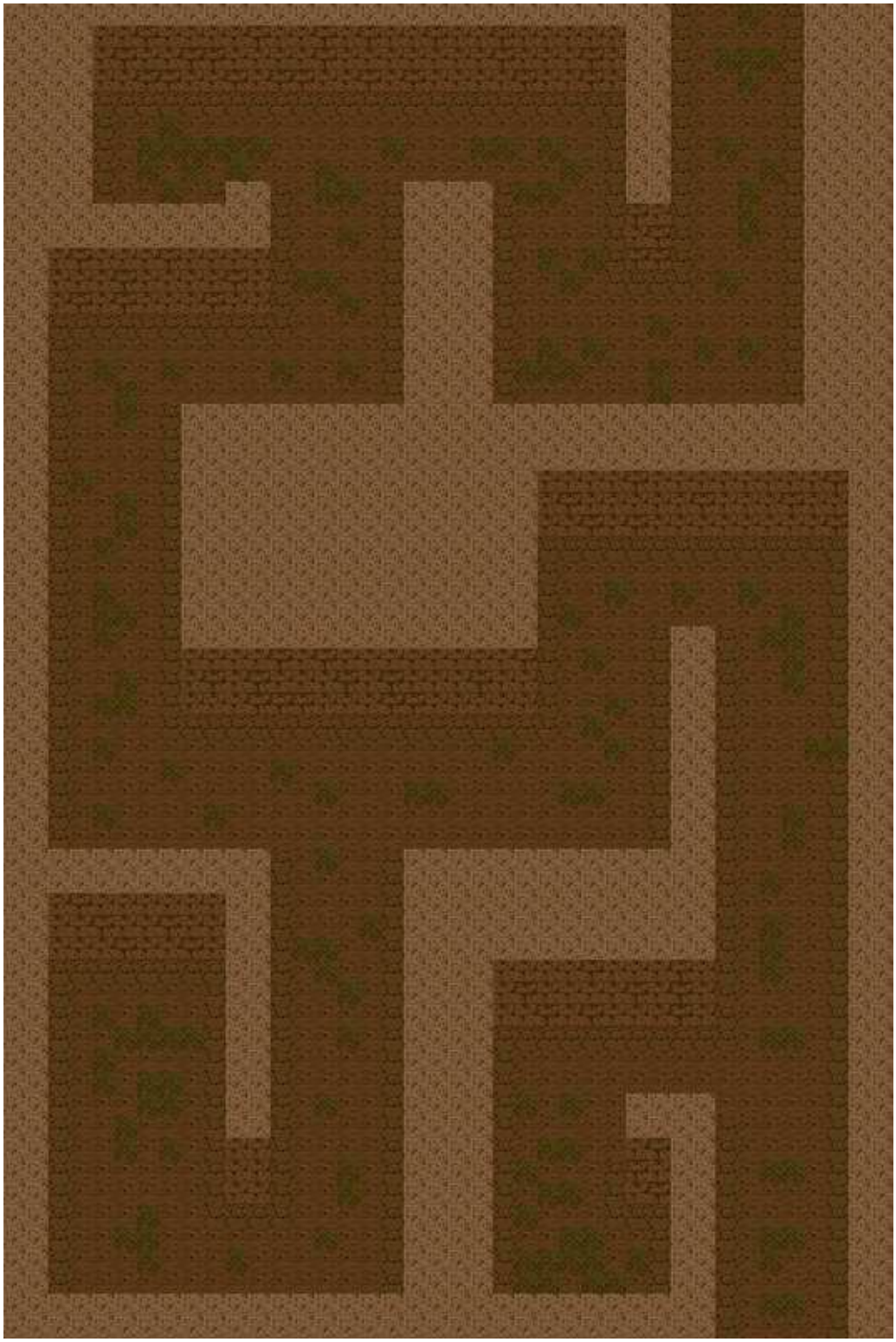
Mapa de Darglia (implementación).



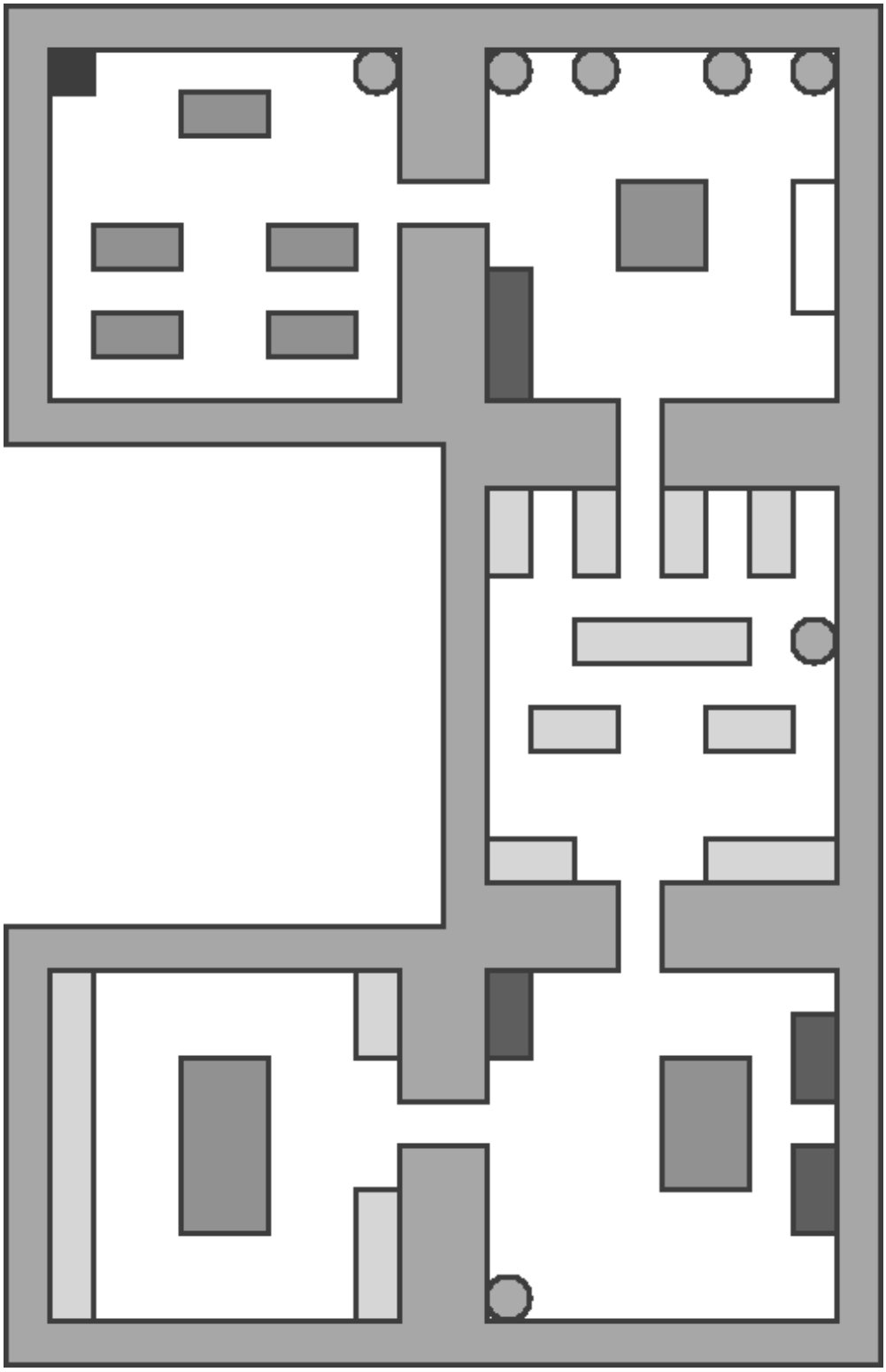
Mapa de Darglia (adaptación).



Mapa de las Cloacas.



Mapa de las Cloacas (implementación).



Edificio Real de Ingeniería de Ric'hem (Propuesta planta baja).

13. APÉNDICE 6: SPRITES UTILIZADOS EN EL DEMO.[5]

Aalok.

CabinsDeadDummy.

CabinsDummy.

Mapas, *WallTorch*, *TrappedChest*.

Efectos.

[5] Propiedad de Square Enix, utilizados con fines demostrativos únicamente.

14. BIBLIOGRAFÍA Y FUENTES DE INFORMACIÓN

- Wilson, Grez, Ostrem, Jean, Bey, Christopher. *PalmOS Programer's Companion Volumen 1*. PalmSource Inc., 2004.
- Wilson, Grez, Ostrem, Jean, Bey, Christopher. *Palm OS Programmer's API Reference*. PalmSource Inc., 2004.
- Maas, Brian. *Using Palm OS Emulator*. PalmSource Inc., 2004.
- Eckel, Bruce. *Thinking in Java*. Prentice Hall, New Jersey, 2003.
- Llopis, Noel. *C++ for game programmers*. Charles River Media, Massachusetss, 2003.
- Rollings, Andrew, Morris, Dave. *Game architecture and design*. Coriolis, Arizona, 2000.
- Ceballos, Francisco Javier. *Curso de programación de Visual Basic 6*. Alfaomega, Madrid, 2000.
- Alonso, Abraham, *Locos por los videojuegos*, Muy Interesante, México, año XXI, núm. 4, abril del 2004, pp. 4-18
- García de Jalón, Javier, Rodríguez, José Ignacio, et al. *Aprenda Visual Basic 6.0 como si estuviera en primero*. Universidad de Navarra, España, 1998.
- http://java.sun.com
- http://www.palm.com
- http://www.palmsource.com

- http://www.soyentrepreneur.com/pagina.hts?N=13375
- http://www.todito.com/paginas/noticias/167100.html
- http://www.emuunlim.com/doteaters/play1sta1.htm
- http://www.scotsmist.co.uk/glossary_p.html
- http://www.vjuegos.org
- http://www.um.es/docencia/barzana/IAGP/Iagp3.html#BM3
- http://lenguajes-de-programacion.com
- http://www.mtas.es/injuve/biblio/estudio_injuve/estudiospdf/videojuegos/introduccion1.2.pdf
- http://www.angelfire.com/goth/bacata/rpg.htm
- http://www.zonapda.com/noticia.asp?CODIGO=929
- http://vx.myftp.org/rs/home.htm
- http://www.programacion.com/tutorial/uml/
- http://www.dcc.uchile.cl/~psalinas/uml/modelo.html
- http://perso.wanadoo.es/fby/rol2.html
- http://sun.waterfaerie.org/faq.html
- http://www.angelfire.com/goth/bacata/rpg.htm
- http://www.idc.com/getdoc.jsp?containerId=pr2003_12_08_102033
- http://www.monografias.com/

- http://www.clikear.com/manuales/uml/diagramascasouso.asp
- http://www.tevasoft.com/UMSA_ANT/ADOO/CasosdeUso.pdf
- http://www.wikipedia.com
- http://www.gamasutra.com