INDEPENDENCIA: JUEGO DE ESTRATEGIA EN 3D BASADO EN HECHOS IMPORTANTES DE LA CAMPAÑA LIBERTADORA DE COLOMBIA

NORMAN HERNANDO MUÑOZ FANDIÑO
WILLIAM ALEXANDER RIVERA LÓPEZ

DIRECTOR: MSC. CARLOS ALBERTO COBOS LOZADA

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE SISTEMAS
LÍNEA DE INTERÉS EN DESARROLLO DE JUEGOS
POPAYÁN, ABRIL DE 2005

INDEPENDENCIA: JUEGO DE ESTRATEGIA EN 3D BASADO EN HECHOS IMPORTANTES DE LA CAMPAÑA LIBERTADORA DE COLOMBIA

MONOGRAFIA

NORMAN HERNANDO MUÑOZ FANDIÑO
WILLIAM ALEXANDER RIVERA LÓPEZ

DIRECTOR: MSC. CARLOS ALBERTO COBOS LOZADA

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE SISTEMAS
LÍNEA DE INTERÉS EN DESARROLLO DE JUEGOS
POPAYÁN, ABRIL DE 2005

Nota de Aceptación

_____

_____

_____

_____

_____

Firma Presidente del Jurado

_____

Firma de Jurado

_____

Firma de Jurado

Popayán 27 de Abril de 2005

Dedico este trabajo a mis padres: Eyvar Rivera y Evelia López, por brindarme su confianza y por haberme dado la mayor riqueza que un padre puede dar a sus hijos: su amor. A mi hermana Miryam Rivera, por escucharme y ofrecerme siempre su apoyo y cariño incondicionales. A mi segunda familia: Patricia Martínez, Rodrigo Rodríguez y Felipe Rodríguez, por comprenderme, tolerarme y apoyarme incluso en los momentos más difíciles. A todas las personas que han estado conmigo regalándome su afecto.
William Rivera.

Con todo mi amor a mi esposa Mónica y mi hijo Santiago, motivo de mi inspiración y constante apoyo.
Norman Muñoz

AGRADECIMIENTOS

Por el apoyo que recibimos en el desarrollo del presente trabajo de investigación, queremos expresar nuestros agradecimientos a las siguientes personas:

CARLOS ALBERTO COBOS LOZADA, Magíster en Informática. Por sus valiosos consejos y sus aportes siempre oportunos.

ERICK ORTIZ, Estudiante de diseño gráfico. Por su labor en el realce visual del juego.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

ORGANIZACIÓN DEL DOCUMENTO

En este documento se dan a conocer los conceptos teóricos y prácticos que fueron necesarios para cumplir con el propósito de esta investigación y serán abordados en cuatro (4) capítulos como se detalla a continuación:

- *Introducción:* En este capitulo se muestra el problema, se expone la pregunta objeto de la investigación, y se da a conocer la justificación de la misma. Además se plantean los objetivos que se llevaron a cabo con la investigación, y se da una descripción general de los resultados obtenidos.

- *Capítulo I – Marco teórico:* El propósito de este capítulo es el de mostrar los conceptos teóricos más importantes sobre, los juegos de computador y su taxonomía, los juegos de estrategia en tiempo real, la planeación del proyecto del juego, la implementación de juegos de estrategia, el conjunto de interfaces de programación de aplicaciones *DirectX* y el contexto histórico.

- *Capítulo II – El proceso de desarrollo:* En este capítulo se realiza una descripción general del proceso que se siguió para el desarrollo del juego, haciendo especial énfasis en los aspectos que presentaron mayores inconvenientes y en la forma en que se dio solución a estos, junto con las técnicas asociadas usadas en la construcción de los componentes principales del juego.

- *Capítulo III - Conclusiones, recomendaciones, y trabajo Futuro:* En esta capítulo se ofrecen las conclusiones relacionadas con el desarrollo del juego y el uso de metodologías y herramientas de punta, y con el trabajo de investigación en si mismo. Por último se presentan las actividades encaminadas a dar continuidad a la investigación y unas recomendaciones para otros investigadores que se interesen en la realización de trabajos de investigación en esta área.

INTRODUCCIÓN

Para el desarrollo de este trabajo de investigación se realizó un estudio de los diferentes conceptos, metodologías y herramientas que se usan actualmente para la creación de video-juegos para PC y las capacidades que cada una de estas expone. Partiendo de esta base se plantea la creación de un juego específico que hace uso de algunas de estas herramientas y metodologías y sirve como punto de partida para futuros trabajos en esta área.

En este capítulo se describe el problema y se exponen los argumentos por los cuales este trabajo se justificó como trabajo de investigación. Además se plantean los objetivos generales y específicos que se cumplieron en el desarrollo de la investigación, y se da una descripción general de los resultados obtenidos.

1. 1. DESCRIPCIÓN DEL PROBLEMA

Una de las áreas de desarrollo más frecuentes en la Ingeniería de Sistemas son los sistemas de información, hacia la cual fluyen la mayor cantidad de proyectos. Esta aceptación inherente a la profesión ha enmarcado el ejercicio de la misma a la construcción de aplicaciones software que de algún modo se podría tipificar como de carácter tradicional, en el sentido que las aplicaciones para manipulación de bases de datos objeto relacionales, se han tomado como el campo de mayor acogida dentro de la rama. Sin embargo, los conocimientos y las áreas en las que se puede desempeñar un ingeniero de sistemas no están limitados únicamente a la creación de sistemas de información, es necesario tener en cuenta otros puntos de vista y enmarcar nuestros horizontes hacia otros rumbos, tal y como puede ser el desarrollo de video juegos, manejo de seguridad computacional, procesamiento digital de imágenes, sistemas de

información geográfica, paradigmas en el desarrollo de software, informática educativa, sólo por mencionar unos cuantos.

En la mayoría de los programas de Ingeniería de Sistemas de las universidades de la región, existe una marcada tendencia hacia el desarrollo de sistemas de información tradicional, dejando de lado otro tipo de sistemas como el de los video juegos. Una de las causas de esta tendencia es la ausencia de industrias regionales enfocadas a la elaboración de juegos, lo cual lleva a creer que existe una cierta informalidad en cuanto al desarrollo de los mismos y no se le da el carácter "serio" que le corresponde. Por el contrario, en otras partes del mundo, el desarrollo de video juegos es un aspecto muy importante de la economía e incluso deja grandes dividendos, aún mayores que industrias del entretenimiento tan posicionadas en el mercado, como la del cine (en el 2002 los video juegos vendieron 10.300 millones de dólares, contra 9.300 millones producidos por Hollywood y el 27.4% de todos los juegos para PC vendidos correspondían al género estrategia [1]).

El programa de Ingeniería de Sistemas de la Universidad del Cauca no ha sido ajeno a la tendencia regional mencionada anteriormente, lo cual es originado en parte porque son pocos los espacios académicos brindados para el potenciamiento de habilidades e intereses en áreas como el desarrollo de video juegos. En tal sentido, se puede notar una ausencia de asignaturas y líneas de interés que promulguen a favor de la motivación y la exploración del área. Asimismo, es conveniente mencionar que paralelo a la anterior causa, existe muy poca orientación profesional hacia la construcción de juegos por considerar que la misma connotación de la palabra juego no involucra un desarrollo serio del mismo (en tanto que se tiende a pensar que no se sigue una metodología de desarrollo como un sistema de información habitual) y una oportunidad como campo de aplicación real de los conocimientos adquiridos.

Otro aspecto a tener en cuenta es que hasta ahora no existen personas con la experiencia suficiente en el tema de modo que se propicie un espacio para el apropiamiento de los conocimientos requeridos para la elaboración de juegos con tecnología de vanguardia. Por otro lado, existe una ausencia notable de herramientas hardware (tarjetas aceleradoras 3D, joysticks, dual shocks, equipos de alto poder de procesamiento y memoria) y software (modeladores 3D y editores de audio) adecuadas que conlleven a una motivación extrínseca y faciliten los procesos de creación de juegos 3D.

Todas las causas mencionadas anteriormente convergen en la creciente necesidad de los estudiantes del programa de Ingeniería de Sistemas de la Universidad del Cauca en cuanto a la aplicación de los conocimientos adquiridos durante el transcurso de su carrera en áreas poco tradicionales tal como es el desarrollo de juegos de gran impacto visual, comercial y/o educativo.

Mediante la implementación de un juego en 3D, se darían los primeros pasos que permitirían, en primera instancia, la apropiación del conocimiento en el área y abonarían el terreno para incentivar la creación de espacios que potencien habilidades e intereses de los estudiantes de Ingeniería de Sistemas de la Universidad del Cauca en cuanto al desarrollo de este tipo de sistemas poco tradicionales. Igualmente, se busca propiciar un cambio de pensamiento en cuanto al desarrollo de juegos, su importancia y sus campos de aplicación, de forma tal que induzca la conformación de una industria regional especializada en este sector del entretenimiento.

Como se ha mencionado antes, el desarrollo de los juegos en 3D es nuevo para la región y especialmente para el programa de Ingeniería de Sistemas de la Universidad del Cauca, de modo tal que no hay mucha información circundante al respecto, que propendan por su consideración como una posible rama de aplicación profesional. No obstante, ya se han dado los primeros pasos en áreas cercanas como la utilización de modelos 3D en óptica [2] y uso de librerías gráficas en procesamiento digital de imágenes [3].

Dentro del mundo de los video juegos es posible encontrar varios tipos [4] [5] como son los de aventura (que es un genero gráfico basado en el personaje y la historia, y donde el personaje debe resolver una serie de acertijos mientras se continúa la historia), los de rol (donde al jugador se le dan varios personajes con los que puede construir una especie de ejercito para matar monstruos y pasar niveles de dificultad), juegos en línea (en los cuales es necesario estar conectados a Internet todo el tiempo) y juegos de estrategia (usualmente basados en el control de muchas unidades en tiempo real que requiere de un plan de acción sistemático para tratar de ganar el juego y que se juega en tercera persona), entre otros.

A nivel nacional, no son marcados los trabajos que puedan dar soporte al mundo de los video-juegos, sin embargo es conveniente mencionar que ya se han llevado a feliz

término trabajos multimediales que de alguna u otra forma muestran la riqueza histórica de nuestro país, como es el caso de "Tres historias para la creación" [6], que es un cortometraje animado creado con la intención de exaltar la cultura tercermundista, mostrando mitos pre-colombinos (Aztecas, Incas, Muiscas, Mayas, etc.), pero además mostrando lo que somos capaces de hacer en cuanto a animación 3D. Para la realización de este corto, fue necesario definir un guión. Posteriormente crear el story board que buscaba definir en el papel aspectos claves que se hubieran podido quedar por fuera del guión, los colores a emplear, las zonas de pantalla, las cámaras a utilizar, los ángulos y una breve idea de cómo serían los personajes. Luego se procedió a diseñar y modelar los personajes, utilizando herramientas para la creación de gráficos 3D, para posteriormente animar a cada uno de estos según su rasgo y rol correspondiente. Por último se procedió a pintar los caracteres en la escena y darles la animación correspondiente.

Más recientemente, en el ámbito nacional, se destaca el trabajo realizado por la empresa Toonka del Parque Tecnológico de Software de Cali, la cual creó una herramienta (Ktoon 2D Animation Toolkit [7]) para la creación de animaciones en 2D y que busca impulsar el trabajo con este tipo de técnica, que tiene poco auge en nuestro medio.

A nivel mundial es posible encontrar múltiples juegos que tratan del tema en cuestión (incluyendo juegos como Age of Empires, Sim City, Age of Mitology, etc.), de los cuales se puede destacar (además de ser juegos de estrategia), que:

- Son juegos en tiempo real, es decir, son juegos en los que constantemente pasa algo y en los cuales el jugador necesita responder activamente a los cambios que se dan segundo a segundo en una progresión natural de tiempo [4].
- Manejan una interfaz gráfica con marcadas zonas como las de mini mapa, herramientas, recursos y juego en sí.
- Utilizan gráficas tridimensionales de buena calidad, pero no muy elaboradas con el fin de no hacer pesado el pintado de las escenas[1], sin embargo, la tendencia es hacia la utilización de tarjetas aceleradoras 3D.

[1] En diseño en 3D este proceso se conoce comúnmente como el *render* de los objetos (incluyendo mapa, personajes, construcciones, etc.) que hacen parte de lo que se desea mostrar.

- La música y los dispositivos de entrada, se adaptan de acuerdo a la historia y a las acciones que sean necesarias, por lo cual es común que tengan una musicalización acorde y se utilice el ratón como dispositivo de entrada principal.

De otra parte, también es posible encontrar juegos de estrategia en el ámbito académico, siendo necesario destacar el desarrollado por Jaime Sánchez y Juan Pablo Alamo del Departamento de Ciencias de la Computación de la Universidad de Chile, denominado "La tribu"[8]. Este juego educativo de tipo de estrategia en Internet, está basado en el poblamiento de Sudamérica, previo a la llegada de los españoles, siendo el objetivo del mismo construir un imperio a partir de una pequeña tribu compuesta por una población limitada; además posee una interfaz gráfica de usuario en 3D dividida en zonas, maneja un mapa con objetos sobre él (árboles, construcciones) y manejo de relieve, explora aspectos sociales, políticos y económicos propios de la época y utiliza la lúdica como herramienta para desarrollar destrezas.

Como se puede apreciar, el desarrollo de juegos en 3D es un trabajo arduo y que requiere de una gran cantidad de esfuerzo y dedicación, ya que es necesario definir algunos elementos de diseño general[9] como las gráficas, los sonidos, la interfaz, el juego en sí (cómo es y qué tan divertido es), la historia, la inteligencia artificial que usa y la habilidad para capturar la atención del jugador, además de unos elementos de estilo[10] como el movimiento y las colisiones, reglas internas consistentes, libertad de movimiento, acción en tiempo real y actividad de fondo, espacio en tres dimensiones y realidad virtual. No obstante, las consideraciones anteriores se puede afirmar que el diseño y desarrollo de un juego no se aleja demasiado de los parámetros tradicionales empleados en la Ingeniería del Software, es decir, se conservan etapas como captura de requerimientos (en algunos casos determinada por la historia sobre la cual se trabajará), diseño, implementación, pruebas e implantación. Lo anterior lleva a afirmar que el desarrollo de un video juego en 3D puede ser tan, o más, complejo, que el de un Sistema de Información tradicional, porque puede incorporar características de gran elaboración como puede ser la misma animación de los personajes, el modelado de los mismos, la implementación de rutinas para el juego, etc.; características que hasta ahora se empezaron a conocer y profundizar mediante la elaboración de este proyecto.

Así pues, se puede pensar que el desarrollo de video juegos es una de esas áreas que poco ha sido estudiada al interior de la Universidad del Cauca y como tal un proyecto que involucre un proceso que lleve a la creación de uno de ellos, permitiría abrir un

nuevo espacio de investigación al seno del Alma Mater, de tal forma que se cimienten las bases para futuros desarrollos en este campo. En tal sentido, se busca dar los primeros pasos para la creación de video juegos, de forma que puedan ser utilizados en un futuro para la creación de una industria regional enfocada en esta área. Se tiene, por tanto, una visión a mediano plazo, que involucra la creación de una empresa dedicada al desarrollo de video juegos, no sólo para la plataforma PC, sino para otras como PS One, Game Cube, etc.

Como es de esperarse, la creación de un juego, y sobre todo en esta época, requiere de una avanzada sofisticación tecnológica, porque dada la calidad de los juegos que existen en el mercado hoy en día, es pertinente utilizar técnicas de vanguardia en la creación de juegos como lo es la utilización de librerías de punta[1] que aporten no sólo en cuanto el manejo de gráficos, sino que brinden más opciones (como el manejo de sonido y dispositivos de entrada, entre otras). De este modo, se puede mencionar librerías netamente gráficas como Open Gvs, Uvision, RenderWare, Brender y Open GL, además otras con soporte para características adicionales como DirectX, que ofrece además de la API para gráficos, una para manejo de música, otra para utilización de diversos dispositivos de entrada y otra más para el desarrollo de juegos en red. Sin embargo, aunque la tecnología de punta existe, no siempre puede ser usada por los desarrolladores de juegos, debido a que sus precios son bastante elevados (como es el caso de Open Gvs, Uvision, RenderWare y Brender).

Se nota pues, que no debe buscarse únicamente una librería que sea totalmente gráfica, sino que además permita disponer de características adicionales que puedan ser explotadas en este proyecto (y en futuros), que la licencia sea económica (de ser posible freeware) y que sea de gran popularidad en el mundo de la programación de video juegos. Características estas, que se ajustan a las del SDK (Standard Development Kid) de DirectX ofrecido por Microsoft.

Se puede apreciar, entonces, que para el desarrollo del proyecto será necesario explorar tecnologías de vanguardia, de modo que no únicamente se aporta en cuanto al tema en general que será soporte del proyecto, sino que se utilizarán alternativas tecnológicas que en el momento no han sido estudiadas al interior de la Facultad de Ingeniería Electrónica y Telecomunicaciones y mucho menos puestas en práctica.

De otro lado, aunque se disponga de la tecnología para la creación de un juego, es necesaria una médula espinal o un hilo conductor para este, es decir, se requiere de una historia que sea el soporte del juego. En primera instancia se podría pensar en la creación de dicha historia (personajes, lugares, hechos, etc.), pero al analizar la historia de Colombia se puede apreciar que posee matices culturales, religiosos, políticos e ideológicos que permiten recurrir a una de las tantas "joyas" históricas de Colombia para convertirla en el eje sobre el cual se desarrolle el juego. Lo anterior, lleva a que se explore un nuevo aporte, como es el involucrar nuestra historia (la historia colombiana) para la creación del juego de forma que en un futuro se puedan hacer estudios que involucren por ejemplo análisis comparativos de aprendizaje de nuestra historia usando como variables de decisión la educación tradicional y la lúdica (en forma de video juegos educativos).

1. 2. CONTRIBUCIÓN EN LA SOLUCIÓN

Para poder llevar a cabo la idea de investigación, se propusieron un objetivo general y tres objetivos específicos, encaminados a la elaboración del guión del juego, la creación de un motor 3D orientado a objetos basado en la API de DirectX y la construcción del mundo de juego. Además se plantearon los resultados que se perseguían como logro de los objetivos planteados. Los objetivos que se presentan a continuación, están redactados en futuro, para mantener su integridad con respecto al anteproyecto de grado aprobado por el Comité de Investigaciones de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca.

1. 2. 1. OBJETIVOS

1. 2. 1. 1. GENERAL

Crear un juego de estrategia en 3D, basado en un hecho importante de la Campaña Libertadora de la República de Colombia, que estimule la exploración y el desarrollo de

los video-juegos para PC al interior del Programa de Ingeniería de Sistemas de la Universidad del Cauca.

1. 2. 1. 2. ESPECÍFICOS

- Elaborar el guión del juego basándose en un hecho de la campaña libertadora de tal forma que sirva como base para contextualizar y delimitar la historia del juego en cuanto a sus personajes, espacio geográfico y orden de los sucesos.
- Crear y usar un motor 3D orientado a objetos basado en la librería DirectX de Microsoft, que sirva como soporte para el dibujado de los objetos gráficos del juego, el control de los dispositivos de entrada y la musicalización de la historia.
- Construir el mundo[2] del juego teniendo en cuenta las siguientes especificaciones:
  - Interfaz de usuario soportada por el uso de mouse y teclado, para que un sólo jugador pueda interactuar con el juego (modo single player).
  - Un escenario o mapa de juego basado en el mapa del virreinato de Santa Fé en 1810 [11], que tenga en cuenta el manejo del terreno y la ubicación de vegetación, construcciones y minas de minerales.
  - Utilizar seis (6) modelos tridimensionales genéricos para la caracterización de los personajes que intervienen en el juego.
  - Niveles de dificultad (fácil, medio, difícil) basados en una línea de tiempo, la cual establece un plazo máximo para el cumplimiento de los objetivos y esta definida de acuerdo a los hechos tenidos en cuenta dentro del guión.

1. 2. 2. RESULTADOS OBTENIDOS

Al finalizar el proyecto y de acuerdo con los objetivos y la metodología planteada para el desarrollo del mismo, se lograron una serie de resultados tangibles y de conocimiento y experiencia en el área del desarrollo de juegos. Estos resultados son:

[2] En programación de video juegos, se conoce al mundo del juego, como el conjunto formado por el escenario de juego, el terreno, las construcciones, los personajes, la forma de interacción, la música y todo lo que hace parte, en sí, del juego como tal.

- El presente documento, como una síntesis general del desarrollo del proyecto, en el cual se describe cómo se cumplieron los objetivos y el proceso seguido para la creación del juego, los aportes significativos del trabajo, las conclusiones y las recomendaciones para trabajos futuros

- El guión del juego basado en la revolución de los comuneros de 1781 en la Republica de Colombia, donde se delimita la historia, se especifican los personajes principales y secundarios, el espacio geográfico y temporal donde se llevó a cabo y la narración de los sucesos que interesan y que acontecieron en ese trozo de la historia.

- Un motor 3D orientado a objetos basado en la librería DirectX de Microsoft, que sirve como soporte la creación de aplicaciones que requieren la utilización de objetos gráficos en tres y dos dimensiones, el control de los dispositivos de entrada (ratón y teclado) y uso de flujos de audio y video.

- Un juego de estrategia en 3D llamado Independencia que consta de una única campaña para un solo jugador (modo single player), cuya dificultad esta determinada por una línea de tiempo, el poder de ataque de los enemigos y el número de los mismos. El juego se puede controlar a través del uso del teclado y del ratón. La campaña está compuesta por un escenario basado en el mapa del virreinato de Santa Fé y cuenta con una ambientación en tercera dimensión que incluye edificaciones, vegetación y manejo de alturas del terreno. Además se incluyen personajes acordes al guión y propios de la época para el desarrollo de las acciones de juego.

  Como un valor agregado, el juego permite hacer acercamientos de primer plano que facilitan la visualización de las acciones que se dan al interior de este; hace uso de técnicas de animación avanzadas como los modelos 3D texturizados y/o animados y las secuencias de texturas con canal alfa [12], transparencias para las edificaciones, diferentes factores de niebla en las áreas del terreno, música de fondo para la partida y efectos de sonido para las acciones en el juego.

- Un Visor de modelos 3D para el formato nativo de DirectX (.x) que hace uso del Motor 3D y permite visualizar los modelos y sus animaciones en modo sólido o de

malla y haciendo uso o no de una (1) o dos (2) luces. Además, permite mover, rotar y escalar el modelo según se desee.

- Un Editor de Terreno que facilita el proceso de selección de las texturas usadas y la modificación en la ambientación y en las alturas de un terreno nuevo o previamente guardado por la aplicación.

- Una aplicación denominada Buscador de Caminos, cuyo propósito es la realización de pruebas de búsqueda de caminos para el algoritmo A estrella[3], que hace uso de un modelo que camina sobre diferentes terrenos con variados obstáculos y tamaños y que permite probar el rendimiento del algoritmo de búsqueda de caminos creado para el juego.

- Un juego de acción denominado Defensor 3D, producto de la fase exploratoria y cuyo principal objetivo era permitir a los desarrolladores apropiarse del conocimiento referente al desarrollo de juegos utilizando la tecnología de DirectX.

- Un documento de estándares de programación y documentación del código generado para el proyecto Independencia, pero que puede ser reutilizado por completo o parcialmente en la ejecución de un proyecto futuro. Para mayor información ver Anexo A.

- La acumulación de experiencia en cuanto a la planeación del proyecto, el proceso y la metodología involucrados en el diseño, desarrollo de juegos y trabajo interdisciplinario, así como la apropiación del conocimiento referente a la tecnología de vanguardia, que da soporte a las graficas en 3D, usada en la construcción de Independencia y que es ampliamente aceptada por las grandes casas de desarrollo de juegos a nivel mundial.

- La participación en la categoría de Rendering en el concurso Imagine Cup del año 2005, concurso que organiza Microsoft cada año, en la cual se premia la creatividad y uso de tecnologías de punta para la creación de proyectos informáticos. En esta categoría se logró avanzar a la final mundial y se tiene como meta quedar entre los

[3] Para mayor detalle acerca del algoritmo A estrella (A*), referirse a BÚSQUEDA DE CAMINOS en el CAPÍTULO II: MARCO TEÓRICO

tres mejores equipos. El story board que se envió para acceder a las finales se presenta en el Anexo E.

- Un artículo en evaluación y posible publicación en la revista Enlace Informático de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca, que también será presentado como ponencia en el XXXI Conferencia Latinoamericana de Informática CLEI2005. En el área de Computación Grafica.

2. CAPÍTULO II: MARCO TEÓRICO

Desde que el ser humano es un niño, una de sus formas de integrarse al mundo que lo rodea, manipularlo e incluso alterarlo, es mediante los juegos. Los juegos no solo facilitan el aprendizaje, sino que permiten que el ser humano realice actividades, que pueden ser peligrosas, sin salir lastimado (al menos físicamente). En tal sentido los juegos se convierten en parte importante de la vida de cada persona y su uso no sólo se limita a los niños, sino que puede extenderse incluso a los adultos.

Ahora bien, es sabido que con el tiempo los juegos han evolucionado, pasando de sencillos y accesibles a complicados y difíciles de jugar. Así, se pasó de juegos tan simples como el de las canicas a más complicados como los de computador, siendo el hilo conductor en esta transición, el avance tecnológico que día a día lleva a explorar nuevos horizontes.

En el siguiente capitulo, se mostrará que son los juegos de computador, que herramientas se usaron para la creación de Independencia y cual fue el contexto histórico sobre el cual se desenvolvió.

2. 1. JUEGOS DE COMPUTADOR

Los juegos de computador han evolucionado a medida que la tecnología ha avanzado y parece lejana la época de PONG® que sin ser un juego destacable por sus gráficas, si llegó a cumplir un gran requisito de los juegos: divertir. Posterior a este juego fueron apareciendo títulos que se convirtieron en clásicos y que dieron las bases para los juegos de hoy. Así por ejemplo, SPACE INVADERS, introdujo un elemento esencial en los juegos actuales: la historia. ¿Qué sería de los juegos de nuestra época sin la historia, sin un argumento sólido que motive a ganar el juego?

Pero además se debe considerar que la creciente avanzada tecnológica impulsó de gran manera el desarrollo de los juegos y la exploración de nuevos juegos, porque al mejorarse el rendimiento de los dispositivos de video, de música y de entrada, entre otros, llevó a que se mejoraran las gráficas y se pasara de planos bidimensionales a gráficos en 3D.

Es por lo anterior que el mundo de los juegos de computador ha crecido y evolucionado, pero además se le suma la industria que gira en torno a ellos y que hace que se busquen nuevas ideas, se exploren nuevas tecnologías y se intente cada día proponer juegos que sean diferentes, no solo en apariencia, sino en contenido. Esto ha llevado a que se de una gama muy amplia de juegos para computador y que dada sus características sea muy complicado a veces definir a que tipo de juego pertenece.

2. 2. TAXONOMÍA DE LOS JUEGOS DE COMPUTADOR

La taxonomía de juegos de computador no es más que la clasificación sistemática que se puede hacer de estos juegos basados en características comunes y subjetivas que se tenga de ellos. Así pues, las particularidades que un individuo percibe de un juego, no necesariamente serán las mismas que otra persona pueda observar y es perfectamente posible además que un juego pueda encajar en dos o más categorías. En tal sentido, la lista que se muestra a continuación no es definitiva y pretende dar unas pautas de organización que faciliten el entendimiento de los diversos tipos de juegos para computador que pueden existir.

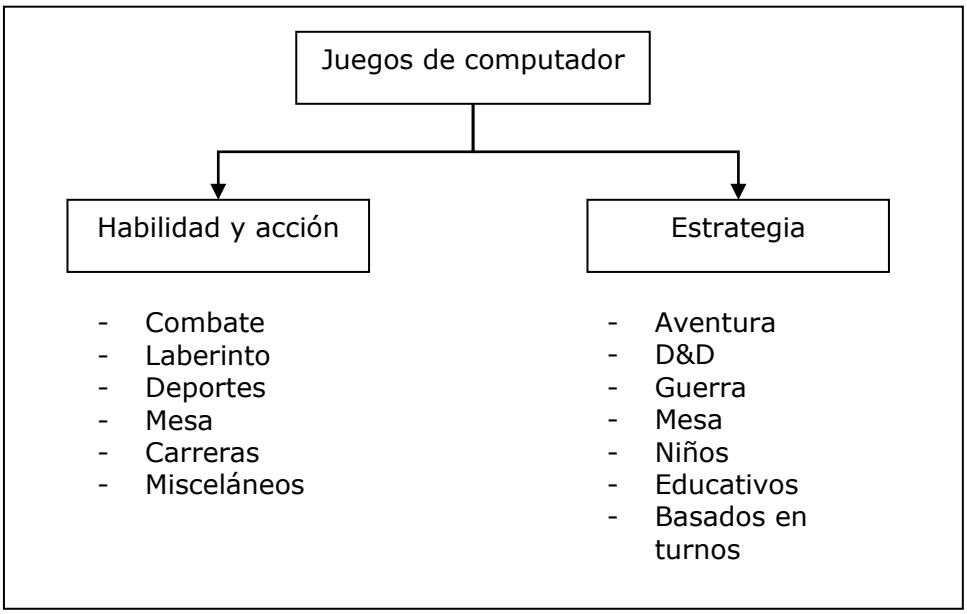

Figura 1. Taxonomía de los Juegos de computador

2. 2. 1. JUEGOS DE HABILIDAD Y ACCIÓN (S&A)

Esta es tal vez la categoría más popular dentro de los juegos para PC. Los juegos de habilidad y acción, S&A (por skill & action en inglés), "se caracterizan por ser juegos en tiempo real, con alto énfasis en gráficas y sonido, uso de joystick o ratón más que teclado. Las habilidades primarias demandadas del jugador son coordinación mano – ojo y tiempo rápido de reacción" [13]. Estos juegos son altamente difundidos y dentro de ellos se pueden encontrar seis categorías más: "juegos de combate, juegos de laberinto, juegos de deportes, juegos de mesa, juegos de carreras y juegos misceláneos" [13].

Son ejemplos de cada uno de estas sub – categorías:

- Juegos de combate: Mortal Combat, Street Fighter, Space Invaders®, Asteroids.
- Juegos de laberinto: Pac-Man®.
- Juegos de deportes: FIFA, NBA, WWF.
- Juegos de mesa: PONG, TETRIS.
- Juegos de carreras: Need for speed, Formula 1, Moto Racer, DownHIll.
- Juegos misceláneos: Donkey Kong®.

Es fácil determinar por los nombres y los ejemplos las características especiales (además de ser de habilidad y acción) de cada una de las anteriores categorías. Así, los juegos de combate se caracterizan por la confrontación entre humano y máquina, los de laberinto, por guiar al jugador por un gran número de corredores que forman un intrincado laberinto, y así sucesivamente.

Es de notar, que los juegos misceláneos, es una categoría donde podrían encajar todos aquellos juegos de acción que no caben en las otras categorías, pero también es de aclarar que hay juegos como Doom, Quake y otros similares, que presentan características tanto de juegos de combate, como de laberinto, con lo cual se ratifica que esta clasificación a pesar de ser buena, no es definitiva y por el contrario, puede ir evolucionando con el tiempo y con la aparición de nuevos títulos.

2. 2. 2. JUEGOS DE ESTRATEGIA

En este tipo de juegos es más importante el aspecto cognoscitivo que las habilidades motrices, aunque la tendencia en los últimos años ha sido la de permitir que el jugador además de tener que pensar en una estrategia para ganar el juego, deba estar pendiente de la acción y reaccione de manera casi instantánea, lo cual se constituye en una primera categoría: juegos de estrategia en tiempo real (RTS por sus siglas en inglés – Real Time Strategy). Además de esta categoría se encuentran:

- Juegos de aventura: Donde el jugador debe ir por un mundo complejo acumulando herramientas y descifrando acertijos para llegar al objetivo final. Un ejemplo clásico de esta categoría es Príncipe de Persia.
- Juegos tipo calabozos y dragones (Dungeons and Dragons - D&D): Este tipo de juegos se caracterizan por ser juegos de exploración, cooperación y conflicto en un mundo de fantasía, con castillos, dragones y hechiceros. Una versión más reciente desencadeno en los muy populares RPG (Role Playing Games), cuyo máximo exponente ha sido Super Mario Bros. ®, y donde el jugador avanza en diversos mundos descubriendo pistas.
- Juegos de guerra: Herederos de los juegos de guerra de tablero, como Batalla Naval, este tipo de juegos de estrategia se caracterizan por mostrar un campo de batalla donde el jugador piensa en una estrategia que permita ubicar

acertadamente sus unidades de combate, con el fin de ganar la partida. Ejemplos de este tipo son: Command & Conquer y Red Alert.

- Juegos de azar: Estos juegos, de fácil implementación, por que existen modelos en el mundo real, se caracterizan por estar basados en juegos de azar jugados durante mucho tiempo y lo único que ha cambiado es el medio donde se muestran. Ejemplos son: Blackjack, Poker, etc.
- Juegos de niños y educativos: Estos juegos están diseñados con propósitos netamente educativos, y sus ejemplos representativos varían desde rompecabezas (puzzles) hasta simuladores de laboratorios.
- Juegos basados en turnos: Como su nombre lo indica, durante el juego, se pasa el turno a cada jugador (humano o no) para que despliegue su jugada y mueva sus unidades o fichas de forma estratégica.

Se puede apreciar, que la lista es muy general y al igual que en los juegos de habilidad y acción, habrá lugar para que un juego pueda tener particularidades de dos o más categorías, y realmente esa es la tendencia, porque juegos de tanta elaboración como WarCraft, Rise of Nations, Age of Empires, Empire Earth, entre otros, combinan características de juegos de guerra, con algo de aventura y acción en tiempo real.

2. 3. JUEGOS DE ESTRATEGIA EN TIEMPO REAL (RTS)

Este tipo de juegos es uno de los más difundidos en la actualidad, y como se apreció anteriormente, tienen características especiales que envuelven a los jugadores y los atrapa dentro de la trama del juego, siendo las más importantes que el jugador defina una estrategia de juego que le permita ganarlo y que además pueda cambiarla si el juego así se lo exige, con lo cual se añade un nivel de habilidad y acción.

La historia de los juegos de estrategia en tiempo real puede remontarse hasta un juego desarrollado por Intellivision, llamado Utopía [12], el cual tenía como argumento el tratar de colonizar la isla de tu contendor (humano o no) o destruirlo. Años después apareció Popolous de Bullfrog, que a diferencia de todos los juegos de estrategia, donde la meta es tener más y más habitantes, en éste se incrementaba el poder si había menos personas creadas. Pero, el juego que hizo populares a los juegos RTS, sin

duda alguna, fue Command & Conquer de Wetswood; un juego de guerra que captó el interés de los jugadores por sus características gráficas, la acción en tiempo real y sobre todo su historia.

La historia de Command & Conquer, aunque es simple, es llamativa, ya que el jugador es quien controla al equipo de Iniciativa Global de Defensa de las Naciones Unidas para luchar en contra de la malvada hermandad NOD liderada por Kane. EL objetivo central del juego es competir por el control del Tiberium en el planeta. El Tiberium, en el juego, es lo que mueve al mundo y quien lo posee, domina el planeta, ya que a partir de éste puede crear unidades de combate, mejorar tecnologías y construir.

Es de notar, por lo visto hasta ahora de los juegos de estrategia en tiempo real, al igual que la mayoría de juegos actuales, que la historia es un componente esencial y le da sentido al mismo, por cuanto, establece unas metas a realizar. Pero, además es importante en los RTS otras características como las unidades de combate, los recursos y el árbol de tecnologías, entre otras, que hacen que este tipo de juegos sean atractivos, por lo cual a continuación se profundizará en los tópicos más destacables y que se deben considerar al desarrollar un juego de estrategia en tiempo real.

2. 3. 1. LA HISTORIA

¿Qué sería de un juego sin la historia de fondo? Pues, muy probablemente, serían como los primeros juegos de computador como PONG, donde el sentido es no dejar caer la bolita.

Muy temprano, en la evolución de los juegos para PC, apareció una historia de fondo para los juegos, con la aparición de Space Invaders®, en el cual se daba un motivo para jugar y se establecían objetivos claros, todos soportados por una historia que le dio un bono adicional al juego.

Lo primero que se debe hacer cuando se desarrolla un juego, es definir la historia, o por lo menos tener claro de que va a tratar el juego. Es decir, definir un tema sobre el cual se enmarca el juego, así, para un juego RTS, el tema podría ser de ciencia ficción,

de vaqueros, medieval, post – apocalíptico, etc. Una vez hecho esto, se pueden definir los elementos que contendrá la historia: el argumento y el propósito.

El argumento de la historia da profundidad y significado a la misma, además de brindar al jugador una razón para ganar. El argumento, describe el fondo de la historia, es decir, establece por qué hay que jugar, dándole al jugador un motivo extrínseco adicional al intrínseco propio de todo juego.

La historia del juego también debe dar un propósito para ganar el juego. Este propósito no sólo deben ser meras victorias militares, sino que debe involucrar al jugador de forma tal que lo haga parte del juego mismo y así se sienta tentado a continuar jugando aunque haya alcanzado el propósito inicial. Por tanto un propósito debe ser alcanzable, pero a la vez extensible en cuanto se pueda alterar de forma dinámica. Por ejemplo, si el jugador puede ganar una partida al construir una maravilla (suponiendo que ese sea el propósito del juego), que se de la posibilidad para que en una partida posterior, las condiciones con las que inició el juego no sea las mismas que la primera vez y el logro de la meta se vuelva un reto.

2. 3. 2. ELEGIR OBJETIVOS

Una vez la historia se ha definido, se deben elegir objetivos acordes al propósito. Los objetivos al igual que el propósito deben ser alcanzables, por ejemplo, conseguir 1000 unidades de alimento, 500 de oro, etc. Estos objetivos deben definirse para cada etapa del juego y es buena idea, que al ir avanzando se tengan nuevas oportunidades para mejorar lo que se tiene hasta el momento.

Un objetivo puede agrupar otros sub-objetivos, de tal forma que la unión de muchos objetivos pequeños constituya un gran objetivo. Por ejemplo, se puede tener la lista de los siguientes objetivos: construir 2 casas, un almacén, un cuartel militar y obtener 500 unidades de alimento; para poder conseguir un objetivo mayor que sería avanzar a la siguiente época.

Los objetivos establecen además metas de diseño del juego, pues cuando se esté creando el mismo, será necesario tenerlos en cuenta y se constituirán en un insumo que enriquecerá el juego.

2. 3. 3. UNIDADES DE COMBATE

No todos los juegos de estrategia en tiempo real poseen unidades de combate, algunos como SimCity, no las requieren, pues su historia no tiene esta necesidad. Pero los que entran dentro de la categoría de juegos de guerra, deben definir unidades de combate y como mínimo tener en cuenta dentro de sus características el costo, la velocidad, la armadura y el poder de ataque.

El costo de la unidad hace referencia a cuánto vale en términos de recursos. Aquí es conveniente que las unidades de combate no sean tan caras que no se puedan adquirir, o tan baratas que se pueda usar la estrategia de ataque en masa, lo cual llevaría a que el juego siempre se pudiera ganar de esa forma y se tornaría aburrido.

La velocidad de la unidad de combate se refiere a que hay que definir con que velocidad se desplaza dicha unidad a través del mapa, siendo lo usual que las unidades más poderosas sean más lentas de modo tal que se de un balance al juego y no ocurra como cuando se utilizan los populares trucos (tricks) y se saca un arma futurista que se puede desplazar a gran velocidad y causar un gran daño, llevando a que el juego se termine rápido y no sea atractivo.

La armadura de la unidad indica el valor defensivo de la misma. No todas las unidades pueden tener valor ofensivo, pero si deben tener valor defensivo, que indique como se defiende de los diferentes tipos de ataque que se hayan definido en el juego. En la Tabla 1 se muestra un ejemplo de diferentes unidades y su armadura para diferentes tipos de ataque.

	Fuego	Balas	Química
Persona	0.1	0.2	0.5
Tanque pesado	0.9	0.7	0.8
Tanque ligero	0.7	0.6	0.7

Tabla 1. Valores de armadura para tres tipos de unidades

Cuando un enemigo ataca con fuego que causa 100 puntos de daño a una persona, los puntos desviados por la armadura serán:

100 (infligido) x 0.1 (armadura) = 10 (desviados)

Entonces el daño causado sería:

100 (infligido) – 10 (desviados) = 90 (de daño causado)

Mientras que para el tanque pesado se tendría:

100 (infligido) x 0.9 (armadura) = 90 (desviados)

100 (infligido) – 90 (desviados) = 10 (de daño causado)

Con la información anterior se calculan los puntos de impacto, es decir, la vida de cada unidad, la cual podría ir de 50 a 5000, siendo 50 lo más básico (Ej. Un aldeano).

Finalmente, el poder de ataque hace referencia a cuanto daño puede causar una unidad. Sin embargo, no sólo esto se debe considerar, sino que la velocidad del disparo, los daños especiales y la velocidad del arma deben ser tenidos en cuenta, para aquellas unidades que así lo requieran. Por ejemplo, si se tiene un arquero, se debe definir la velocidad con la cual realiza un disparo además de la velocidad de la flecha al desplazarse.

2. 3. 4. JUEGO EN MODO CAMPAÑA

La mayoría de los juegos de estrategia tienen un modo de juego que es el denominado Modo Campaña, que básicamente son una serie de juegos prescritos que el jugador sigue a través de la historia. La razón por la cual se dice que son prescritos, es porque son preparados por el desarrollador y tienen diferentes propósitos como el de servir de tutor o también como soporte del juego.

Cada juego en el modo campaña, es una misión, por tanto debe existir un editor de misiones, que permita enviar al jugador a una tarea con riegos y recompensas. Cada misión a su vez, debe tener establecidos un conjunto de objetivos que el jugador debe lograr con el fin de ganar la misión. No se trata, entonces, sólo de realizar tareas, sino que estas deben estar relacionadas con uno o más objetivos.

2. 4. PLANEACIÓN DEL PROYECTO DEL JUEGO

Una vez establecida la primera parte de un juego, es decir, definida la historia y el propósito del juego, se puede pasar a la siguiente etapa en la elaboración del juego: la planeación del proyecto.

La planeación de un juego no se aleja demasiado de los parámetros establecidos para la creación de un sistema tradicional, y por el contrario, conserva muchas de las características que se establecen en estos últimos. Sin embargo, si es conveniente aclarar que al igual que en los sistemas de información tradicional, cada quien tiene su propio estilo para planear y en la elaboración de juegos ocurre algo similar y cada diseñador de juegos define sus propias fases para el logro de la meta final, el juego.

Sólo por nombrar algunos casos, Todd Barron, define las fases de invención, requerimientos, documentación técnica, desarrollo, pruebas, producción y distribución[14]; Francois Laramee, define seis pasos: tratamiento del diseño, diseño preliminar, diseño final, especificación del producto, la Biblia gráfica y el guión cinematográfico interactivo[15]; mientras que Chris Crawford define solo tres fases: definir las estructuras de entrada y salida, del juego y del programa[13].

Se puede apreciar, pues, que la planeación de un juego depende en gran parte de quien este elaborando el juego, pero a pesar de esto, existen características comunes como la documentación, la definición de características del juego, la estructura de la interfaz de entrada y salida y las relaciones existentes entre los componentes que harán parte del juego.

La metodología usada para la creación del juego motivo de esta investigación fue una adaptación propia del Proceso de Desarrollo Unificado, teniendo en cuenta que para la fase de análisis se definió el guión del juego el cual sirvió posteriormente para las etapas de diseño e implementación.

En conclusión, si se desea desarrollar un juego, antes de empezar a escribir toneladas de código, hay que sentarse a pensar en cómo se desea ver el juego, pero a partir de lo que se defina inicialmente, la historia y el propósito del juego. Con esto claro, se puede escribir las características del juego, lo que debe tener (requerimientos), las interacciones entre los componentes, la interfaz de entrada y salida y en fin todo aquello que se debería considerar al crear un sistema de información convencional.

2. 5. IMPLEMENTACIÓN DE JUEGOS DE ESTRATEGIA

Después de tener el diseño, ya se puede empezar a codificar, pero no sin antes tener unos conocimientos básicos que son útiles en el momento de elaborar un juego de estrategia en tiempo real y que incluso se utilizan en la mayoría de los juegos. Estos conceptos incluyen el conocimiento de programación basada en losas (tiles), algoritmos de búsqueda de caminos (path finding) y análisis y diseño orientado a agentes, además de algunos prerrequisitos matemáticos que se detallan en el Anexo G.

En este apartado se dará una introducción a estos temas y se buscará establecer unas bases que puedan ser utilizadas posteriormente para la creación de un motor 3D y en general para la utilización en la construcción de juegos en tres dimensiones.

2. 5. 1. MANEJO DE LOSAS

Cuando se piensa en losas, hay que hacerlo como si se estuviera viendo un tablero de ajedrez, pero no de 8 x 8, sino de n x n, donde cada casilla del tablero constituye una

losa. De tal forma, se puede afirmar que la losa es un bloque de construcción que unido a otros tiene sentido, pero, solo no.

La utilización de losas es muy popular en la programación de juegos, y se utilizan en la construcción del mapa del juego y/o de terrenos. Muchos tipos de juegos como los RPG (Role Playing Games) usan losas para crear todo el mapa del juego y además para programar la lógica del mismo. En cuanto a la creación del mapa, es claro, que cada losa constituye un pedazo que unido a los demás dan la sensación de conjunto del terreno, un caso típico es Super Mario Bros. (Figura 2), que tiene un conjunto de losas para la creación de los mundos.

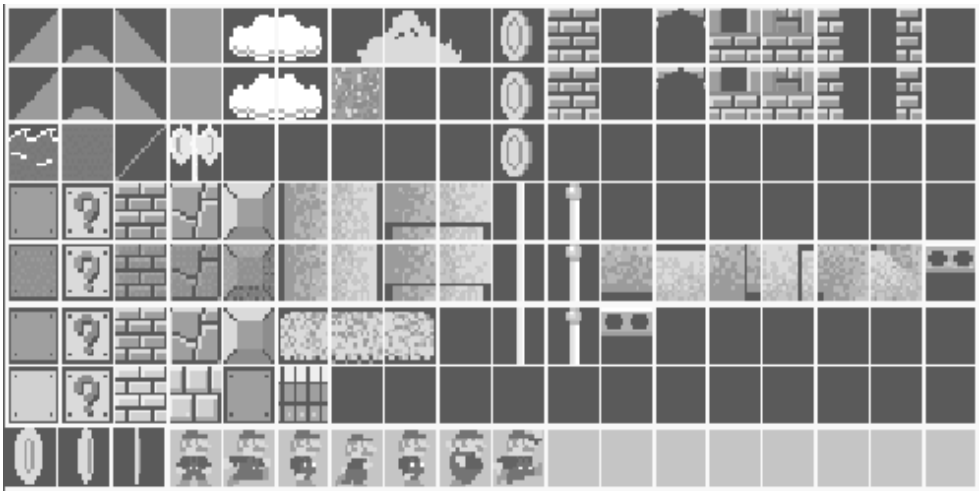

Figura 2. Losas usadas en Super Mario Bros.

En cuanto a la lógica del juego, se puede implementar gracias a que se puede establecer qué hay en cada losa en un determinado momento, de este modo se puede controlar la acción del juego.

Ahora bien, además de utilizarse como mapa, ¿por qué hay que usar losas? La respuesta contempla dos temas: memoria y reutilización. La utilización de losas ayuda a conservar la memoria, pues no es lo mismo usar 100 losas de 64 x 64 píxeles, a usar un bitmap de 100 x 100 de 64 x 64 píxeles, ya que en el primer caso sólo son necesarios aproximadamente 2MB, mientras que en el otro cerca de 163 MB de memoria.

La reutilización es un factor clave, pues el uso de losas permite que se puedan reutilizar en diferentes partes del mapa, con lo cual el diseñador gráfico sólo tendría que crear pedazos pequeños y no un gran mapa, que tuviera mayor complejidad.

Pero no solo la reutilización y conservar memoria son importantes, también es de destacar que se puede establecer si una losa está ocupada por una unidad y qué unidad es la que está ahí, además de que a las losas se les puede especificar propiedades, las cuales pueden ser utilizadas para definir características de una losa o de un conjunto de ellas. Estas propiedades incluyen:

- Obstrucción: que dice si se puede caminar o no por la losa,
- Elevación: determina si una losa es una "montaña" o no, o en su defecto, puede tener un valor que me indica que tan elevada está, y
- Brillo: que es útil para los efectos de guerra y también como propiedad para ocultar losas.

Como se ve, entonces, usar losas, es una de esas técnicas de programación que ayuda no sólo a preservar memoria, sino que facilita la construcción de mapas para la creación de terrenos y para la manipulación de la lógica del juego.

2. 5. 2. BÚSQUEDA DE CAMINOS

Teniendo definidas las losas, se debe pensar en cómo se va a hacer para desplazar a una unidad desde un punto A hasta un punto B del mapa, y más aún, gracias a las propiedades de las losas, hay que establecer si hay obstáculos que puedan alterar dicho desplazamiento. A este problema se le conoce como la búsqueda de camino. En la Figura 3 se muestran dos ejemplos de búsqueda de caminos, uno sin obstáculo y otro con un obstáculo.

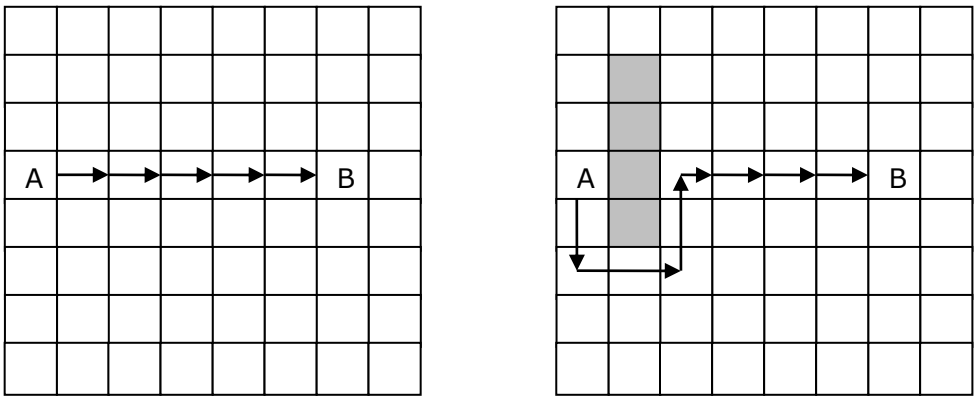

Figura 3. Ejemplos de búsqueda de caminos

El problema de encontrar caminos, no sería tan complicado si no hubiera obstáculos que sortear para ir desde un punto hasta otro, pues un algoritmo fácil sería:

```
Si la unidad está a la izquierda del objetivo, muévase a la derecha.
Si la unidad está a la derecha del objetivo, muévase a la izquierda.
Si la unidad está encima del objetivo, muévase hacia abajo.
Si la unidad está debajo del objetivo, muévase hacia arriba.
```

Pero los problemas surgen cuando hay obstáculos, pues hay que bordearlos. Una forma inicial de solucionar este inconveniente sería generando un movimiento aleatorio, pero se corre con la posibilidad de que se bloquee o se demore mucho para poder llegar al objetivo. El pseudo código sería:

```
Mientras no esté en el objetivo
      Si está a la izquierda del objetivo, muévase a la derecha.
      Si está a la derecha del objetivo, muévase a la izquierda.
      Si está encima del objetivo, muévase hacia abajo.
      Si está debajo del objetivo, muévase hacia arriba.
      Si está bloqueado, muévase aleatoriamente.
Fin mientras
```

Una forma de solucionar este problema es mediante la implementación del método de la ***A estrella*** (A*) [14]. Este método es un buen algoritmo de búsqueda de camino, puesto que bordea obstáculos y mejor aún encuentra la mejor ruta basado en un terreno cambiante.

Los fundamentos del algoritmo A* consisten en definir una lista de nodos cerrados y otra de nodos abiertos, donde cada nodo corresponde a una losa del mapa y el nodo actual se denomina nodo padre.

Lo primero que se debe hacer es definir unos costos, los cuales servirán para el cálculo de la mejor ruta. Estos costos son:

- Costo base de un nodo: Es el costo del nodo en términos de movimiento, es decir, el costo según las características de la losa. Así por ejemplo a una losa que posee césped, se le puede dar un costo de 1, a una con rocas 2, a otra con arena 3 y así sucesivamente.
- Costo del nodo de inicio: Indica cuanto cuesta para el nodo regresar al punto de inicio. Para calcular esto, se toma el costo del nodo padre y se le suma el costo base de la losa actual.
- Costo desde el nodo objetivo: Es calculado sumando el número horizontal y vertical de nodos que hay del nodo actual hasta el nodo objetivo.
- Costo total: Es la suma de los tres costos anteriores.

El método de búsqueda de caminos A*, trabaja de la siguiente forma:

- Se coloca el nodo de inicio en la lista de nodos cerrados.
- Se adicionan los nodos adyacentes al nodo de inicio, en la lista de los nodos abiertos.
- Se busca el nodo adyacente con el menor costo total y se adiciona a la lista de nodos cerrados.
- Se remueve el nodo con el costo total más bajo de la lista de nodos abiertos.
- Si no se ha conseguido llegar al objetivo en la lista de nodos abiertos, se continúa buscando, abriendo nodos alrededor del que se adicionó a la lista de nodos cerrados y luego colocando en esta lista al nodo con el menor costo total de la lista de nodos abiertos. El proceso sigue hasta que se encuentre el objetivo.
- Cuando se llegue al objetivo, hay que volver hacia atrás. Esto se hace utilizando los nodos padres, de tal manera que nuevamente se retorne al nodo inicial y se pueda hacer el recorrido desde el inicio hasta el final.

2. 5. 3. MODELADO DEL COMPORTAMIENTO DE LOS PERSONAJES

Existen muchas técnicas que utilizan los juegos modernos para el modelado del comportamiento de los personajes [17], pero es de destacar la utilización de las máquinas de estados finitos, los agentes y el scripting, no con esto desmeritando la importancia que tienen otras como las redes neuronales, la lógica difusa, entre otros. Sin embargo, las tres primeras son de especial atención, dado su gran utilización en el mundo de los juegos debido a su fácil codificación y a los resultados que ofrece en corto tiempo.

Para la construcción de Independencia se hizo uso de una metodología para el análisis y diseño orientado a agentes llamada GAIA, la cual permitió modelar el comportamiento de los personajes dentro del juego, haciendo uso además en la fase de implementación, de las técnicas de Maquinas de Estados Finitos y scripting. Por esto, a continuación se hace una breve introducción a las máquinas de estado finitas, los agentes y el scripting, teniendo en cuenta un ejemplo práctico de su utilización en el mundo de los juegos de computador.

2. 5. 3. 1. MÁQUINAS DE ESTADOS FINITOS

Las máquinas de estado finitos (MEF) son quizá, la técnica más ampliamente usada en los juegos de computador por sobre cualquier otra técnica. Su fortaleza radica principalmente en que son fáciles de programar y generalmente suficientes para resolver casi cualquier problema. Una MEF divide el comportamiento de los objetos en estados lógicos, de manera que cada estado identificará un tipo de comportamiento que el objeto exhibe en un momento determinado.

Otra ventaja del uso de las MEF es que facilitan el proceso de depuración y de comprensión de las mismas, además se ajustan casi a cualquier sistema que exhiba un número limitado de estados de operación y puede que no brinden la mejor solución, pero habitualmente funcionan y son mucho más simples de desarrollar que otras técnicas; e incluso pueden trabajar en conjunto con otras técnicas de IA, como por ejemplo pueden ser usadas como el módulo corazón de los agentes.

Algunos inconvenientes de las MEF es que si no se estructuran adecuadamente, presentan muy poca escalabilidad (mantenimiento muy difícil) e incrementan su tamaño con cada ciclo de desarrollo.

Son usadas en la mayoría de juegos de computador comerciales tales como: Age of Empires, Enemy Nations, Half Life, Doom y Quake. Quake, por ejemplo usa una MEF con 9 estados diferentes (parado, caminando, corriendo, eludiendo, atacando, esquivando, mirando al enemigo, desocupado y buscando) para representar el comportamiento de cada personaje. Específicamente en Independencia se utilizaron para representar la actitud que asumen los personajes en un momento determinado.

2. 5. 3. 2. AGENTES

Un agente es un sistema de computador que es capaz de realizar acciones independientes en beneficio de su propietario o usuario, es decir, son autónomos. En tal sentido, se puede afirmar que un agente es capaz de actuar independientemente, exhibiendo control sobre su estado interno, para cual hace uso de su percepción interna del ambiente en el que se desenvuelve.

Los juegos brindan ambientes realistas perfectos para los agentes, en los cuales solo una parte de la información puede estar disponible para el agente y donde las decisiones deben ser tomadas bajo restricciones de tiempo y presión. Generalmente, los agentes en juegos son conjuntos de MEF que trabajan cada uno en un problema particular y se comunican con los demás.

Cuando se considera la inclusión de agentes en juegos, es importante considerar si el agente va a ser puramente reactivo, dirigido por objetivos o una combinación de los dos. Los agentes puramente reactivos están asociados con ambientes altamente cambiantes, mientras que los agentes dirigidos por metas están asociados con ambientes estáticos, en los que es conveniente planear y considerar movimientos previos antes de emprender alguna acción.

Un buen ejemplo de la utilización de agentes en juegos es "Empire Earth", en el cual la IA se compone de varios componentes llamados administradores. Cada administrador es responsable por un área específica dentro del juego. En dicho juego hay administradores para la civilización, las construcciones, las unidades, los recursos, la investigación y el combate. El administrador de la civilización es el administrador de más alto nivel y es responsable por el desarrollo de la economía del jugador computador y la coordinación entre los otros administradores. Los otros administradores tienen deberes de más bajo nivel y envían peticiones y reportes a los demás.

2. 5. 3. 3. SCRIPTING

La técnica de scripting consiste básicamente en la creación y/o utilización de un lenguaje de programación usado para simplificar alguna tarea compleja para un programa particular. Un lenguaje de scripting es considerado un lenguaje de cuarta generación y su alcance puede variar desde un simple archivo de configuración hasta un complejo lenguaje interpretado en tiempo de ejecución, pero en todos los casos pretende controlar el comportamiento del sistema desde afuera.

Los juegos que hacen uso de scripting, tales como Quake o Unreal, permiten que el código del juego sea programado en un lenguaje de alto nivel, semejante al inglés, logrando así ocultar muchos aspectos complicados del juego y permitiendo a los usuarios no programadores modificar el comportamiento del juego de una manera más natural.

El scripting es ampliamente usado en los juegos de rol para la construcción de árboles de conversación y en general para el manejo de eventos de IA, por ejemplo para la creación de tutoriales de juego.

2. 5. 3. 4. GAIA: UNA METODOLOGÍA PARA EL ANÁLISIS Y DISEÑO ORIENTADO A AGENTES

GAIA [18], es una metodología para realizar los procesos de análisis y diseño orientados a agentes. La orientación a agentes surge como un nuevo paradigma de la ingeniería del software para la abstracción de sistemas a un nivel más alto, lo cual brinda la posibilidad de hacer una mejor comprensión del sistema en estudio mediante el uso de agentes como herramienta de entendimiento de las sociedades. La filosofía de GAIA se basa en el concepto del sistema visto como una organización en la cual existe una serie de roles y a su vez, cada rol posee un conjunto de interacciones con otros roles. Un rol es asumido por un individuo en un determinado momento.

Esta metodología hace uso de una serie de modelos de análisis y diseño. La relación entre estos, se muestra en la Figura 4.

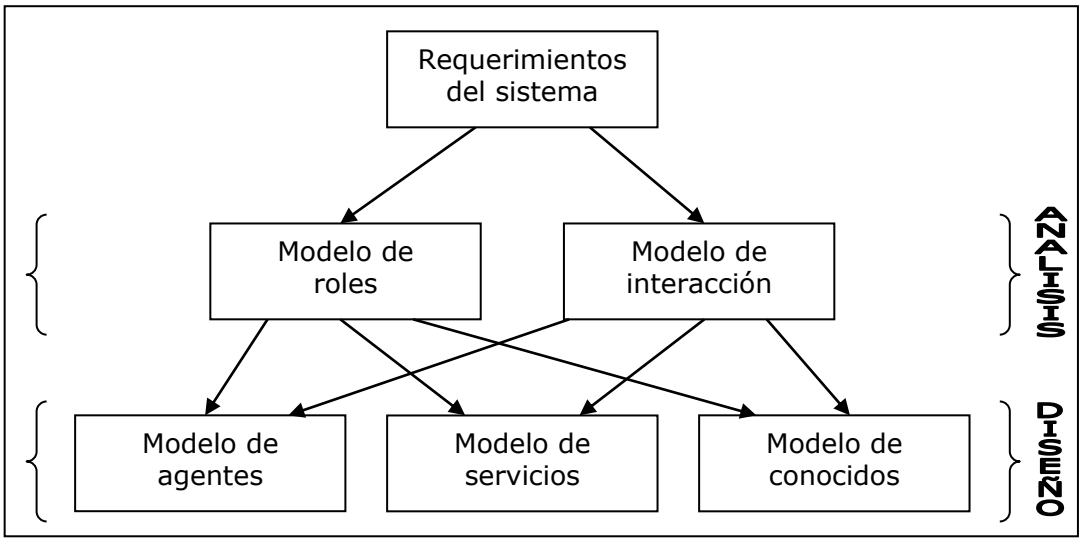

Figura 4. Relación entre los modelos en GAIA.

2. 5. 3. 4. 1. ANÁLISIS

La etapa de análisis en GAIA busca el entendimiento del sistema y su estructura como una *organización*. Una organización puede ser vista como un conjunto de roles que mantienen ciertos tipos de interacción con otros roles en el sistema. Un rol es definido

en GAIA por cuatro atributos: las responsabilidades, los permisos, las actividades y los protocolos. Las responsabilidades, a su vez, pueden fraccionarse en 2 categorías: las propiedades de vida y las propiedades de seguridad. Todos estos conceptos del sistema pueden verse en la Figura 5.

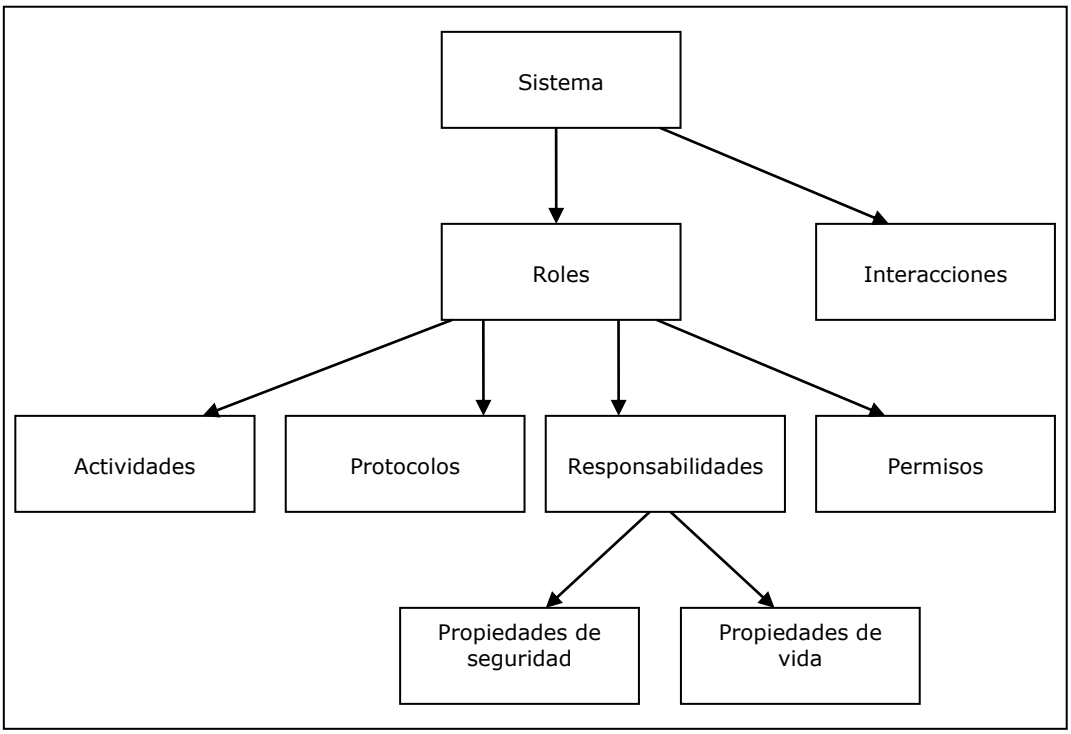

Figura 5. Conceptos de análisis en GAIA.

Las responsabilidades determinan la funcionalidad y son talvez los atributos claves asociados con un rol. Las propiedades de vida intuitivamente declaran que *"algo bueno ocurre"*. Por el contrario, las propiedades de seguridad son invariantes e intuitivamente afirman que *"nada malo pasa"*. Los permisos identifican los recursos que el rol tiene disponibles para realizar sus responsabilidades. Por lo general, estos tienden a ser recursos de información y con cada recurso habrá asociados unos derechos coligados, como por ejemplo: leer, cambiar o generar el recurso. Las actividades representan acciones privadas u operaciones que el agente puede completar sin la necesidad de interactuar con otros agentes en el sistema. Por último, los protocolos definen las formas en que un agente asumiendo dicho rol podría interactuar con otros agentes.

Durante la etapa del análisis en GAIA se generan dos tipos de modelos a saber: el modelo de roles y el modelo de interacción.

2.5.3.4.1.1 EL MODELO DE ROLES

El objetivo principal del modelo de roles es el de identificar los distintos tipos de rol que se encuentran en el sistema. Un rol puede ser visto como la representación de un tipo de función específica dentro de la organización, por ejemplo, el gerente de una compañía. Con cada rol hay asociados unos permisos y unas responsabilidades. Los permisos identifican los recursos a los cuales el rol puede acceder y los límites impuestos sobre estos recursos. Los permisos pueden ser del tipo:

```
reads      recurso       // Permite leer el recurso
changes    recurso       // Permite leer y modificar el recurso
generates  recurso       // El rol es el productor del recurso
```

La funcionalidad de un rol está determinada por sus responsabilidades, y estas pueden verse como responsabilidades de vida y responsabilidades de seguridad. Las responsabilidades de vida son llamadas así tienden a decir que *"algo se hará"* y por tanto que el agente desempeñando el rol está aún vivo. La forma general de una expresión de vida es:

```
NOMBRE_ROL = expresión
```

La expresión de vida está compuesta por actividades y/o protocolos y operadores, donde las actividades son acciones que el agente puede realizar por si mismo, mientras que los protocolos involucran la interacción con otros agentes. Los operadores se muestran en la Tabla 2.

Operador	**Interpretación**
X**.**Y	X seguido por Y.
X**|**Y	Ocurre X o Y

X*****	X ocurre 0 o más veces
X**+**	X ocurre 1 o más veces
[X**]**	X es opcional
X**||**Y	X y Y intercaladas

Tabla 2. Operadores para expresiones de vida.

Por otro lado, las responsabilidades de seguridad se relacionan usualmente con la ausencia de alguna condición no deseable y se expresan por medio de una serie de predicados, por ejemplo:

```
recurso > límite
```

Con estos conceptos claros, se puede especificar un **esquema de rol** para cada tipo de rol en la organización y conformar así el modelo de roles. Un esquema de rol presenta una descripción del rol, sus protocolos y actividades, sus permisos y las responsabilidades de vida y seguridad. Una plantilla del esquema de rol es proporcionada en la Tabla 3.

Esquema del Rol: Nombre del Rol
Descripción: Descripción del rol.
Protocolos y Actividades: **Actividades:** Lista de actividades. **Protocolos:** Lista de protocolos
Permisos: Permisos del rol.
Responsabilidades: **De vida:** Nombre del Rol = expresión **De seguridad:** Predicados de seguridad.

Tabla 3. Plantilla del esquema de rol.

2.5.3.4.1.2 EL MODELO DE INTERACCIÓN

Este modelo se conforma de una serie de **definiciones de protocolo**, una para cada tipo distinto de interacción entre los roles. Cada definición de protocolo consiste de un propósito de la interacción, los roles iniciador y receptor de la misma, las entradas y salidas y el procesamiento ejecutado. El propósito de la interacción es una descripción textual breve del objetivo de la interacción. Las entradas al protocolo son toda la información usada por el rol iniciador, mientras que las salidas son la información suministrada por/a el receptor del protocolo durante el curso de la interacción y el procesamiento es una corta descripción textual del proceso que realiza el rol iniciador. Una plantilla de la definición de protocolo es proporcionada en la Tabla 4.

Protocolo		
Rol Iniciador	Rol Receptor	Entradas al protocolo
Descripción textual de las operaciones realizadas por el rol iniciador durante el procesamiento.		Salidas del protocolo

Tabla 4. Plantilla de la definición de protocolo.

2. 5. 3. 4. 2. DISEÑO

El proceso de diseño en GAIA involucra la transformación de los modelos de análisis a un nivel de abstracción suficientemente bajo para que sea posible implementar los agentes haciendo uso de las técnicas de diseño tradicionales, tales como las que hacen uso de la orientación a objetos. Así pues, el propósito de la metodología GAIA no es el de definir como un agente va a realizar sus responsabilidades, sino el de cómo van a cooperar para cumplir con los objetivos a nivel de la organización.

En la etapa de diseño, GAIA hace uso de 3 tipos de modelos: el modelo de agentes, el modelo de servicios y el modelo de conocidos.

2.5.3.4.2.1 EL MODELO DE AGENTES

La intención de este modelo es la de definir los diferentes tipos de agente que se podrán encontrar en el sistema y el número de instancias que se tendrán de cada uno en tiempo de ejecución. Un tipo de agente, no es más que un conjunto de roles. Se puede afirmar entonces, que un tipo de agente puede asumir uno o más roles, aunque lo contrario no es cierto.

El modelo de roles es construido mediante un árbol de tipos de agentes, en el cual los nodos hoja corresponden a los roles y los restantes a los tipos de agentes. Para cada tipo de agente se debe definir un cuantificador de instancia, el cual precisa el número de instancias que se tendrán en el sistema en tiempo de ejecución. En la Figura 6 se brinda un ejemplo de un modelo de agentes y en la Tabla 5 se dan lo posibles cuantificadores de instancia para un modelo de agentes.

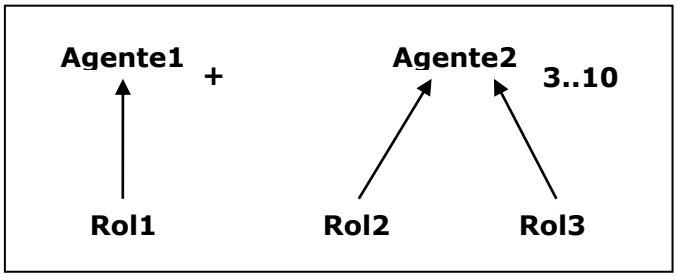

Figura 6. Ejemplo de un modelo de agentes.

Cuantificador	Significado
n	Habrá exactamente n instancias.
m..n	Habrá entre m y n instancias.
*	Habrá 0 o más instancias
+	Habrá 1 o más instancias

Tabla 5. Cuantificadores de instancia.

2.5.3.4.2.2 EL MODELO DE SERVICIOS

Este modelo expone los servicios que cada tipo de agente va a implementar, entendiendo por servicio, cierta funcionalidad del agente. Cada servicio es derivado de

las actividades y protocolos, así como de sus propiedades de vida y seguridad encontradas en la etapa de análisis.

El modelo de servicios se compone de las propiedades de cada uno de los servicios, como son: las entradas, las salidas, las precondiciones y las poscondiciones. Las entradas y salidas proceden en forma directa del modelo de protocolos. Las precondiciones y las poscondiciones constituyen limitantes en los servicios y son derivadas de las propiedades de un rol.

2.5.3.4.2.3 EL MODELO DE CONOCIDOS

La finalidad del modelo de conocidos es la de precisar los enlaces de comunicación que existen entre tipos de agentes e identificar posibles problemas de embotellamiento surgidos por el uso de estos canales de comunicación. El modelo es un grafo dirigido, en donde cada nodo corresponde a un tipo de agente y las aristas se relacionan con los caminos de comunicación. Así pues, un grafo **A -> B**, muestra que hay un camino de **A** a **B**, pero no necesariamente de **B** a **A**.

3. CAPÍTULO III: DESARROLLO DEL PROYECTO

Previo al desarrollo del proyecto, se llevó a cabo una fase exploratoria, en la cual se buscaba principalmente tener un conocimiento y dominio suficiente de la tecnología proporcionada por DirectX con el fin de que después en el juego, se pudiera realizar un diseño consistente y apropiado, teniendo en cuenta lo que podía ofrecer la tecnología elegida. Para un mejor conocimiento de la tecnología de DirectX, referirse al Anexo F.

Durante el desarrollo de este proyecto de investigación se adoptó como metodología de trabajo una adaptación del proceso unificado de desarrollo (UP), el cual está compuesto por una serie de fases (Inicio, elaboración, construcción y transición) con unas etapas bien definidas que son: Análisis, Diseño, Implementación, Pruebas y Entrega. Este proceso, se trabajó de manera iterativa e incremental. Sin embargo, la etapa de análisis no se realizó de forma tradicional, debido a que en esta se buscó, en primer lugar, contextualizar y enmarcar el juego dentro de los aspectos históricos relevantes que sirvieron como soporte para el desarrollo del mismo, y en segundo lugar, modelar el comportamiento de los personajes que intervendrían en él. Los resultados de esta fueron el guión del juego y los modelos de análisis de la metodología GAIA[4].

Ya en la etapa de diseño, y basándose en los resultados arrojados por el guión y en los objetivos propuestos al inicio de este proyecto, se hizo pertinente modelar los artefactos necesarios para la construcción de los productos que se tendrían al finalizar esta investigación. Para tal fin, se decidió hacer uso de la potencialidad del UML (Lenguaje unificado de modelado), cuyo propósito es el de apoyar a los diseñadores y desarrolladores en su trabajo de modelado y posterior paso a implementación. Cabe anotar aquí, que el UML no pretende ser camisa de fuerza para el proceso y cada desarrollador usa los artefactos que considera necesarios y/o apropiados para la buena

[4] Referirse a GAIA: UNA METODOLOGÍA PARA EL ANÁLISIS Y DISEÑO ORIENTADO A AGENTES en el CAPÍTULO II: MARCO TEÓRICO.

marcha de su proyecto. En este caso, se considero pertinente el uso de diagramas de casos de uso y diagramas de clases principalmente, además de un diagrama de estados y un diagrama de secuencia.

En la primera parte de la etapa de diseño se trabajó en la construcción de los artefactos que servirían de base para la creación de aplicaciones con capacidad para manipular objetos 3D y 2D, flujos de audio y video y entrada desde el teclado y el ratón, todo esto haciendo uso de la API de DirectX. Dichos artefactos conformaron lo que se denomina un Motor 3D, que está caracterizado por ser Orientado a Objetos e implementar las funcionalidades más comunes para soportar las capacidades arriba mencionadas, así como otras utilidades frecuentemente usadas por este tipo de aplicaciones.

A medida que los artefactos para el Motor 3D crecían y se hacían cada vez más completos, surgían nuevos inconvenientes que era imperativo solucionar. El primero de ellos fue la dificultad de sincronización y pruebas del trabajo interdisciplinario que se realizó con el personal de diseño. Más concretamente, esta dificultad se daba gracias a que para la creación de los modelos 3D de los personajes y la ambientación para el terreno, el diseñador usaba 3D Studio MAX y luego exportaba los modelos al formato nativo de DirectX (.x) mediante un plug-in (Panda DirectX) y al momento de probar dichos modelos en el contexto del proyecto, el formato, en muchas ocasiones, era incorrecto, la escala no era la adecuada, las animaciones se veían distorsionadas o simplemente el modelo no se podía visualizar. Todas estas dificultades generaban retrasos que había que eliminar del proyecto y para lograrlo se pensó en la construcción de una aplicación que permitiera visualizar los modelos en formato .x que el diseñador generaba. De esta forma, la persona encargada del modelado de los personajes podía hacer pruebas una vez realizaba la exportación del modelo y se aseguraba que éste estaba correcto. A esta aplicación se le llamo un Visor 3D y su funcionalidad general es la de permitir visualizar los modelos 3D estáticos y animados en el formato nativo del DirectX (.x) haciendo uso de las clases que fueron generadas para el Motor 3D.

Ya en el momento de trabajar con terrenos, se presentó una nueva complicación. Esta se daba básicamente por la necesidad de agilizar el proceso de edición de las texturas usadas, la ambientación y las alturas de un terreno. Para subsanar este inconveniente se consideró apropiado desarrollar una aplicación específica que diera soporte a estas

operaciones y que pudiera ser usada tanto por los desarrolladores del juego, como por el personal del área de diseño. A este aplicativo se le denomino un Editor de Terrenos, y su propósito es el de agilizar el proceso de creación y edición de terrenos, incluyendo la aplicación de texturas y la modificación de ambientación y alturas.

Superados los anteriores inconvenientes y encontrándose en la etapa de implementación, surge la necesidad de hacer pruebas de rendimiento al algoritmo usado para la búsqueda de caminos, ya que el algoritmo genérico A* funciona bien, pero puede demorarse demasiado en arrojar un resultado, y dado que el juego está clasificado como un juego en tiempo real, este comportamiento no es deseable. Por tal motivo, se empezó a trabajar también en una aplicación que permitiera a los desarrolladores, realizar pruebas con un modelo caminando sobre diferentes terrenos con variados obstáculos y tamaños y haciendo uso del algoritmo de búsqueda de caminos creado para el juego. Esta aplicación fue llamada el Buscador de Caminos.

Una vez el Motor 3D era consistente y estable (en cuanto a que proporcionaba la funcionalidad deseada y poseía una arquitectura escalable) se empezó el diseño e implementación del juego, para lo cual en primera instancia se hizo uso de la metodología GAIA [18] para el diseño orientado a agentes y posteriormente se complementó mediante la utilización de UML para el completo modelado del sistema. Acto tras del cual se pasó a la implementación.

Cabe comentar, que entre las etapas de diseño e implementación se realizaron varias iteraciones dado que en primera instancia se procedía a diseñar los artefactos que posteriormente se implementaban y de los cuales surgía un prototipo que poco a poco se iba mejorando hasta obtener el resultado deseado. De esta manera se creó el Motor 3D que a su vez sirvió como base no sólo de diseño sino de implementación para la construcción de las otras aplicaciones.

Para mostrar más en detalle el proceso de desarrollo a continuación se profundizará en la fase de exploración y en las etapas de análisis, diseño e implementación.

3. 1. FASE DE EXPLORACIÓN

En esta fase se inició un proceso de autoaprendizaje de los conceptos fundamentales referentes al soporte gráfico proporcionado por DirectX. Aquí se empezó a conocer la arquitectura de Direct3D, el manejo de luces y materiales, transformaciones de mundo, de vista y de proyección y manipulación de modelos tridimensionales. Adicionalmente, también se empezó a averiguar la forma en la que se podía trabajar la musicalización.

Para poder verificar el cumplimiento del objetivo de esta fase, es decir, lograr la apropiación del conocimiento referente a DirectX y a la construcción de juegos con esta tecnología, se propuso la creación de un juego sencillo denominado Defensor 3D, cuya historia es la de una nave espacial que debe proteger a la tierra de invasores y peligrosos asteroides. Esta aplicación tenía como requerimientos iniciales la utilización de modelos tridimensionales, música de fondo y efectos de sonido.

Dentro de la construcción de Defensor 3D, se presentaron algunos problemas de orden técnico que a la larga sirvieron para solidificar el conocimiento deseado y cuya solución permitió que Independencia pudiera nutrirse de toda esa experiencia. Se cuenta dentro de estos problemas, la conversión de coordenadas de dos (2) a tres (3) dimensiones, la selección con el ratón de modelos tridimensionales, la utilización de sonidos y el manejo de colisiones. Además la realización de Defensor 3D permitió vislumbrar un panorama que facilitó la estructuración del motor 3D, diseñado e implementado posteriormente.

Para poder realizar la conversión de coordenadas de dos (2) a tres (3) dimensiones, fue necesario considerar la manera en como se podía calcular el ancho y el alto de la ventana y a partir de estos, realizar los cálculos necesarios para obtener un rayo que permitiera determinar el origen y la dirección hacia la cual se tendría que expandir la transformación del punto. Adicionalmente, este problema requirió de la utilización de la transformada de vista, con el fin de hacer coincidir la ubicación espacial del objeto con la posición desde la que se observaba la escena.

La solución del problema anterior aportó al momento de poder determinar si el puntero del ratón estaba o no sobre un modelo tridimensional, porque se tuvo en cuenta la intersección entre el rayo calculado a partir de la posición del ratón y una esfera contenedora del objeto, calculada teniendo en cuenta el centro y el radio del modelo.

La esfera contenedora del modelo, no sólo sirvió para el propósito anterior, sino que fue utilizada para la detección de colisiones entre modelos, realizando los cálculos con base en el solapamiento de los centros y radios de los objetos.

El otro problema afrontado, fue el de la música, el cual se abordó desde la tecnología proporcionada por DirectMusic, que ofrece soporte para la carga y reproducción de sonidos desde archivos o recursos MIDI, WAV, o en formato de ejecución nativo de DirectMusic. Sin embargo, analizando los posibles alcances que pudiera llegar a tener el proyecto, se debió considerar la inclusión de un objeto que soportara y permitiera la reproducción de otros formatos, no sólo de audio, sino también de video. En la Figura 7 se muestra la pantalla de configuración de Defensor 3D, desde donde se puede seleccionar además de la nave para el juego, las opciones para determinar si usa o no música de fondo y efectos de sonidos controlados por la API de DirectMusic.

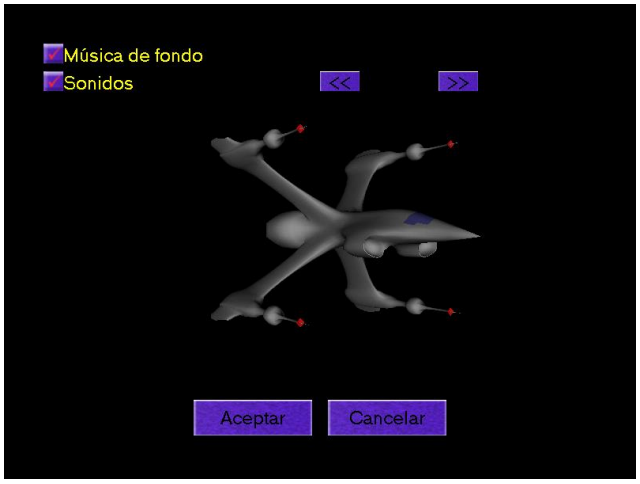

Figura 7. Pantalla de configuración de Defensor 3D.

La solución a los anteriores inconvenientes y el estudio adecuado de la tecnología ofrecida por DirectX, permitió que el diseño del motor 3D que posteriormente se abordó fuera más consistente y ofreciera la funcionalidad necesaria para crear juegos. Así por ejemplo, el requerimiento del soporte para diferentes formatos de audio y

video, que tuvo como desenlace la tecnología ofrecida por DirectShow, dio origen a una sola clase que en el motor se encarga de brindar esos servicios.

Una vez terminado el juego Defensor 3D, se llegó a la conclusión que los conocimientos apropiados hasta el momento eran los suficientemente maduros como para continuar con el proyecto y además se tuvo la satisfacción y motivación que impulsaron las siguientes etapas en la construcción de Independencia.

3. 2. ANÁLISIS

Por ser Independencia un juego de estrategia en tiempo real, el análisis que se hace en este tipo de proyecto difiere al que se podría hacer en un sistema de información tradicional, puesto que no es posible modelar la realidad del juego sin tocar aspectos de diseño, en tal sentido no se puede concebir la idea del jugador ejerciendo la acción de jugar el juego sin verlo frente a un sistema computarizado. Para hacer más claridad al respecto, téngase en cuenta el caso del análisis de un juego de ajedrez. En este tipo de juego si se puede realizar un modelado del sistema, porque es fácil visualizar al jugador ejerciendo su rol en el mundo real.

Sin embargo, era necesario crear un hilo conductor para el juego, de tal manera que definiera los hechos que se iban a tratar, en dónde se iban a realizar y que personajes iban a intervenir. Esto se logró con la creación de un guión que contenía la historia del juego, el lugar y el tiempo de la misma y los personajes que iban a intervenir.

Una vez hecho lo anterior, y antes de empezar el diseño, se inició el modelado del comportamiento de los personajes definidos previamente en el guión y para lo cual se uso la metodología GAIA.

3. 2. 1. EL GUIÓN DEL JUEGO

La tarea de escribir un guión para un juego, no es una labor para ingenieros y debería ser producto de un trabajo interdisciplinario con personas que tengan experiencia en la elaboración de libretos y sepan narrar claramente una historia apropiada para el juego. Este fue un gran inconveniente en la elaboración del presente proyecto, porque no se pudo contar con una persona idónea que proveyera este vital insumo para el juego.

Como el proyecto estaba andando y no se podía detener se optó por investigar dentro de los manuales de juego de títulos conocidos, como Warcraft y Age of mythology, la forma en la cual estructuraban la historia. Esto permitió obtener características comunes que fueron incluidas dentro del guión de Independencia, tal como la narración de la historia y la definición de los personajes y sus características. A continuación se muestra ese resultado, que es la base para el juego y que se enmarca dentro del contexto histórico que se presenta en el Anexo D.

3. 2. 1. 1. HISTORIA DEL JUEGO

El 16 de abril de 1781, un mes después del rompimiento del edicto, se reunió en el Socorro (Santander) un grupo de hombres provenientes de regiones circundantes, con el fin de protestar por el reciente incremento en los impuestos. Ese día, se nombra una junta a la cual se le denominó *Común*, siendo designado como general el señor Juan Francisco Berbeo. Se empieza, entonces, una marcha con destino hacia la Capital.

Las autoridades coloniales deciden enviar un grupo de hombres bien armados, al mando del señor Joaquín de la Barrera, para que detenga la marcha. Mientras tanto, los 'comuneros' siguen la ruta Socorro, Oiba, Velez hasta llegar a Puente Real, lugar donde se desencadena una confrontación entre los dos bandos.

La lucha era desigual, porque los comuneros tan solo estaban armados de hondas y machetes, mientras que los españoles eran superados en número, pero poseían rifles y espadas de combate. Sólo la exaltación de los ánimos mediante el continuo aliciente de su líder, llevó a los comuneros a ganar la batalla.

3. 2. 1. 2. ESPACIO GEOGRÁFICO

El espacio geográfico en el que ocurrieron los hechos abarca el espacio recorrido por el ejército de los comuneros luego del rompimiento del edicto y hasta la victoria en Puente Real, es decir, el espacio comprendido entre El Socorro y Puente Real, pasando por Oiba y Vélez.

3. 2. 1. 3. PERSONAJES

Una vez que se tiene definida la historia del juego, es necesario identificar en ésta, los personajes principales y secundarios con el fin de determinar cuales de ellos tienen real trascendencia dentro de la historia y porqué. Los personajes principales, generalmente juegan el rol de líderes, héroes o caudillos dentro de la historia, y su intervención en la misma es de vital importancia para el desarrollo de los hechos. Los personajes secundarios son los que intervienen en la historia de una forma un poco menos importante y no siempre son recordados o identificados con nombre propio en ella, por ejemplo, los soldados de un ejercito en una batalla.

En la historia del presente guión se identificaron dos personajes principales y cuatro personajes secundarios que intervienen de forma activa en la misma y que se presentan a continuación.

3. 2. 1. 3. 1. PRINCIPALES

- Sin quererlo ser, **Juan Francisco Berbeo**, se convierte en el caudillo que guía a los comuneros en su travesía hacia la capital, pero no sin antes mencionar su lealtad a la Corona Española e insinuar que su participación era obligada. Este personaje por ser el líder del ejército de los comuneros es llamado Prócer, esta armado con una espada y tiene la habilidad especial de emitir un grito de batalla denominado "El grito comunero", el cual incentiva, a los combatientes comuneros que lo escuchan, a atacar con más ahínco a su enemigo.

- En el ejercito de los españoles el personaje principal es **Joaquín de la Barrera**, quién es el dirigente de su ejército. Está armado con una gran espada, tiene un ataque fuerte y buena protección de armadura.

3. 2. 1. 3. 2. SECUNDARIOS

- **Honda**: Este integrante del ejército de los comuneros tiene como arma una honda. Tiene un ataque fuerte pero es lento entre cada ataque y está muy desprotegido de armadura.
- **Machete**: Este combatiente comunero esta equipado simplemente con un machete. Tiene un ataque débil, pero es rápido entre cada ataque y tiene protección de armadura media baja.
- **Rifle**: Este integrante del ejército español tiene como arma un rifle. Tiene un ataque fuerte pero es lento entre cada ataque y tiene armadura media.
- **Espada**: Este combatiente español esta armado con una espada. Tiene un ataque débil, pero es rápido entre cada ataque y tiene protección de armadura media.

3. 2. 1. 4. CAMPAÑA

Una vez que el usuario decida jugar la campaña del juego, debe afrontar una misión que debe ser completada, en un tiempo determinado por el nivel de dificultad, para poder obtener la victoria. Esta misión esta compuesta por unos objetivos principales o de obligatorio cumplimiento y unos objetivos secundarios u opcionales.

3. 2. 1. 4. 1. MISIÓN

Llevar al prócer desde El Socorro hasta Puente Real y ganar la batalla contra los españoles.

3. 2. 1. 4. 2. OBJETIVOS

OBJETIVOS PRINCIPALES

- El prócer debe sobrevivir.
- Armar un ejército compuesto por al menos 10 comuneros.
- Ir desde Socorro hasta Puente Real y ganar la confrontación.

OBJETIVOS SECUNDARIOS

- Desarrollar la habilidad del grito comunero.
- Buscar el centro de construcción de armas para convertir a los campesinos en guerreros.

3. 2. 2. ANÁLISIS GAIA

Cuando se habla de un juego en modo campaña contra el sistema, se debe tener en cuenta que el jugador "sistema" debe adoptar un comportamiento que dé la apariencia de inteligencia. En tal sentido, son muy variadas las técnicas con las que se puede implementar este tipo de conducta, tal como se vio en el ítem MODELADO DEL COMPORTAMIENTO DE LOS PERSONAJES en el CAPÍTULO II: MARCO TEÓRICO.

En este proyecto en particular la apariencia de inteligencia del juego estaba determinada por el comportamiento de los personajes dentro del mismo, por tal motivo, era necesario hacer uso de alguna técnica para soportar este hecho, así como también, de una metodología que permitiera trabajar en el modelado del mismo. Se pensó entonces en el uso de una metodología para realizar el proceso de análisis tal como GAIA [18]. Dicha metodología brindaba dos fortalezas: la primera de ellas es que es orientada a agentes, lo cual nos permitiría posteriormente hacer uso de los mismos como eje central de la lógica del juego, y la segunda es que no impone un núcleo para la implementación, sino que por el contrario, permanece desligada de esta etapa y brinda la posibilidad al desarrollador de escoger la estructura de implementación deseada.

La posibilidad de utilizar una metodología para el análisis y modelado orientado a agentes en Independencia, no sólo permitió el modelado del comportamiento de los personajes, sino que explora una técnica de vanguardia en el mundo de los juegos que está siendo adoptada por grandes casas de desarrollo de juegos como Sierra y su título Empire Earth.

Dado que en este proyecto el uso de GAIA es únicamente para propósitos de modelado del comportamiento de los personajes, se hace una descripción del mismo, antes de pasar al modelo de roles y posteriormente al de interacción.

3. 2. 2. 1. COMPORTAMIENTO DE LOS PERSONAJES

Todos los personajes que intervienen en la historia del juego y que fueron descritos en el guión representan integrantes dentro de un ejército u otro. Cada uno de ellos es denominado una Unidad, y en el caso especial de Juan Francisco Berbeo es denominado Prócer. El comportamiento que puede ser asumido por éstos actores es resumido en un diagrama de casos de uso que se muestra en la Figura 8, en el cual se observa que el actor Unidad puede vigilar, caminar, atacar a un enemigo, recibir un ataque, escuchar el grito comunero y morir, mientras que el actor Prócer puede asumir todo el comportamiento de una Unidad y adicionalmente puede lanzar el grito comunero.

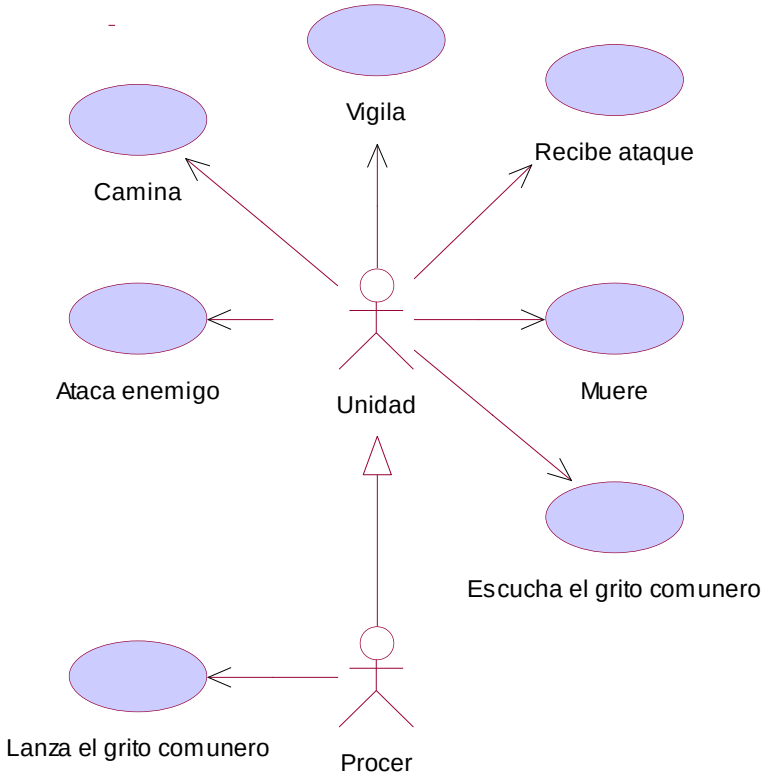

Figura 8. Casos de uso para el comportamiento de los personajes

Cuando una unidad está en estado de vigilancia, constantemente verifica si hay enemigos en el área dentro de su rango de visión y de ser así, decide atacar a la unidad enemiga visualizada. Si la unidad decide atacar, se mueve hasta que el objetivo se encuentre dentro del rango de ataque y cuando suceda lo anterior, la unidad hace que su enemigo reciba el ataque. Cuando una unidad recibe un ataque, disminuye sus puntos de vida de acuerdo al poder de ataque de la unidad que realiza el ataque, combinado con la protección o armadura de la unidad atacada. Si luego de recibir un ataque, los puntos de vida de la unidad son menores o iguales a cero (0), la unidad muere. Luego de realizar un ataque, la unidad debe esperar un tiempo dado para poder realizar el siguiente ataque, dicho tiempo está determinado por la velocidad de ataque de la unidad.

Adicionalmente, una unidad puede caminar desde su posición actual a una nueva posición deseada en el terreno, para lo cual la unidad calcula la ruta que debe seguir hasta su destino e inicia el recorrido.

El tipo especial de unidad denominada un Prócer, tiene la habilidad de lanzar un grito de batalla que incentive a los combatientes comuneros para que ataquen con más ahínco. En el momento en que el prócer decide lanzar el grito, determina si ha pasado un tiempo suficiente desde la última vez que lo hizo para poder volver a lanzarlo. De ser así, las unidades de su ejército que están al 'alcance del grito' pueden escucharlo e incrementan su poder de ataque durante un lapso de tiempo determinado por la duración del grito.

3. 2. 2. 2. EL MODELO DE ROLES

Siguiendo la metodología de análisis GAIA se empezó por identificar los roles que iban a ser considerados dentro del juego. Esto con el fin de determinar que papeles eran relevantes y aportaban dentro de la configuración de la organización del juego. Como tal, se identificó únicamente el rol ***Unidad***, el cual se detalla en la Tabla 6 mediante su esquema de rol.

Esquema del Rol: Unidad
Descripción: Este rol involucra el vigilar el área del terreno desde donde se encuentra la unidad hasta donde alcanza su rango de visión para saber si hay unidades enemigas y de ser así se acerca hasta donde su rango de ataque sea suficiente para atacar al enemigo y lo hace. En el combate la unidad puede perder todos sus puntos de vida así que puede morir. Desde el momento de su creación, la unidad sabe si puede incentivar a las unidades amigas cercanas haciendo uso del grito comunero, es decir, si la unidad se puede comportar como un prócer.
Protocolos y Actividades: **Actividades:** • **Vigilar:** Consiste en revisar permanentemente las losas que están dentro del rango de visión de la unidad para determinar si hay unidades enemigas. De ser así, la unidad puede pasar a la actividad de atacar. • **Caminar:** Esta actividad implica el desplazarse de un lugar a otro dentro del terreno teniendo en cuenta los obstáculos del terreno y la posición de las diferentes unidades dentro de él. • **Morir:** Cuando una unidad está siendo atacada, pierde progresivamente sus

puntos de vida. Si estos puntos se hacen cero (0) o menos, la unidad pasa a la actividad morir, la cual impide que la unidad se mueva, vigile, ataque o incentive a sus tropas.

Protocolos

- **Atacar:** Este protocolo se realiza cuando la unidad ha detectado una unidad enemiga y esta última se encuentra dentro del rango de ataque de la primera. Esta actividad consiste en la disminución de los puntos de vida de la unidad enemiga hasta que alguna de las dos muera.
- **GritoComunero:** Esta actividad es realizada por la unidad cuando tiene la capacidad de ser Prócer y consiste en incrementar la capacidad de ataque propia y la de las unidades amigas cercanas durante un cierto intervalo tiempo.

Permisos:

changes supplied Terreno // Terreno donde se encuentra la unidad
Losas // Información de una losa
changes supplied Mapa // Información de obstáculos en el terreno
Mapa[fila][columna] // Obstáculo en una losa
reads supplied EsProcer // Indica si la unidad es un prócer

Responsabilidades:

Liveness:

Unidad = ((Vigilar.[Caminar].[Atacar].[Morir])||[GritoComunero])^{W}

Safety:

puntosVida > 0 // Condición de vida de la unidad
EsProcer = (TRUE | FALSE) // Indica si la unidad es un prócer

Tabla 6. Modelo de roles. Esquema del rol Unidad

Del esquema anterior se puede resaltar lo siguiente:

- Terreno es un recurso que representa la información geográfica del terreno donde se encuentra la unidad. El terreno esta dividido en Losas y cada losa tiene información específica de esa parte del terreno, como por ejemplo, la altura, si la losa es visible para el jugador o no y el factor de niebla que tiene la losa. La unidad (si es una unidad del jugador) en su desplazamiento por el terreno necesita cambiar el estado de visibilidad y el factor de niebla de las losas que se encuentran dentro de su rango de visión.
- Mapa es un recurso que representa la información acerca de los obstáculos en el terreno. Esta información es utilizada por las unidades cuando desean desplazarse

de un lugar a otro dentro del terreno; por tal motivo esta información se encuentra en cambio constante.

- EsProcer: La unidad puede únicamente leer la información de este recurso, el cual le indica si puede, cada cierto intervalo de tiempo y a petición del usuario, incentivar a las unidades amigas que se encuentren cercanas a ella emitiendo el grito comunero. El tipo de unidad con esta capacidad es llamada un Prócer.

3. 2. 2. 3. EL MODELO DE INTERACCIÓN

Luego de haber hecho la identificación del rol Unidad, fue necesario definir las interacciones que podrían tener, un agente desempeñando este rol, con otros agentes en el sistema. Para lograr esto se definieron unos protocolos que podían ser usados por el rol Unidad. A continuación se muestran las definiciones de dichos protocolos.

3. 2. 2. 3. 1. EL PROTOCOLO ATACAR

En protocolo Atacar se puede visualizar desde las perspectivas de la unidad que ataca y de la unidad que recibe el ataque. Desde el punto de vista de la unidad que ataca, esté protocolo es utilizado cuando la unidad visualiza a una unidad enemiga dentro del espacio abarcado por su rango de visión. Por otro lado, la unidad que recibe el ataque hace uso de este protocolo cuando una unidad enemiga ha completado un ataque en contra de ella; el resultado de esto es la disminución de los puntos de vida de la unidad receptora en una cantidad que depende del valor del ataque, llegando posiblemente a pasar al estado de muerte si sus puntos de vida llegan a ser menores o iguales a cero. Lo anterior se representa en la Tabla 7.

Atacar		
Unidad	Unidad	UnidadEnemiga
Cuando una unidad enemiga se encuentra dentro del rango de ataque de la unidad iniciadora, esta última lanza un ataque contra la primera.		

RecibirAtaque		
Unidad	Unidad	ValorAtaque
La unidad que recibe el ataque, disminuye sus puntos de vida en un valor que depende del valor del ataque.		PuntosVida

Tabla 7. Definición del protocolo Atacar

3. 2. 2. 3. 2. EL PROTOCOLO GRITOCOMUNERO

En protocolo GritoComunero es básicamente una habilidad especial que puede tener una unidad y consiste en hacer que las unidades amigas cercanas a ella incrementen su nivel de ataque durante un tiempo determinado. Este tipo de unidad es llamada un prócer y el protocolo se describe en la Tabla 8.

LanzarGritoComunero		El usuario pide al prócer lanzar el grito.
Unidad	Unidad	
Si una unidad es un prócer, la unidad puede hacer uso del grito comunero para incentivar a las unidades amigas cercanas para que ataquen con más ahínco.		Las unidades cercanas escuchan el grito.

EscucharGritoComunero		
Unidad	Unidad	Tiempo
Cuando el prócer emite el grito comunero, las unidades amigas que estén cerca a él, pueden escucharlo, así que éstas incrementarán sus puntos de ataque durante un tiempo determinado.		PuntosAtaque

Tabla 8. Definición del protocolo GritoComunero

3. 3. DISEÑO

En esta etapa se tuvieron en cuenta dos fases: una inicial en la cual se crearon las bases para dar soporte a la parte gráfica y de musicalización del juego y otra en la cual se desarrollaron los artefactos para las aplicaciones que utilizaban los productos resultantes de la primera. A continuación se hablará en primera instancia del diseño del motor 3D para después pasar a lo relacionado con el diseño del visor de modelos, el editor de terrenos, el buscador de caminos y el juego.

3. 3. 1. MOTOR 3D

Una vez concebido el guión, se inició la tarea de construcción de la librería de clases base o el Motor 3D, el cual sirvió posteriormente para la creación del juego y quedó como un producto tangible independiente del juego y que sirve para el desarrollo de aplicaciones que necesitan hacer uso de características tales como objetos en 3D, música, sonidos, videos y control de los dispositivos de entrada (el teclado y el ratón). Para llevar a cabo esta labor, se inició en primer lugar, un trabajo de apropiación tecnológica en cuanto a la funcionalidad, interfaces y servicios expuestos por las API's de DirectX Graphics y DirectShow, las cuales se encargan de la manipulación de objetos en 3D y flujos de medios (audio y video) respectivamente, esto se logró mediante el desarrollo del juego Defensor 3D. Luego se prosiguió con el diseñó de la jerarquía de clases que conformaría el motor y posteriormente a la construcción de cada una de estas.

3. 3. 1. 1. ESTRUCTURA JERÁRQUICA

Cuando se empezó el diseño del motor 3D, se buscó que permitiera la manipulación de gráficos en tres y dos dimensiones, además de audio y video. Este requerimiento inicial, planteó una necesidad no explorada hasta el momento: derivar todas las clases a partir de un tronco en común o por el contrario usar muchas derivaciones provenientes de diferentes partes, es decir utilizar una jerarquía en árbol o por el

contrario en bosque. Al respecto Timothy Budd [19] comenta que la ventaja del enfoque de árbol es "que la funcionalidad proporcionada en la raíz del árbol es heredada por todos los objetos. En un lenguaje así, toda clase definida por el usuario debe ser una subclase de alguna clase ya existente que por lo general es la clase Objeto."

El otro enfoque sostiene que "las clases que no están lógicamente relacionadas deben ser por completo distintas. El resultado es que el usuario se enfrenta a un bosque compuesto por diversos árboles de herencia. Una ventaja de este enfoque es que la aplicación no está obligada a cargar con una gran biblioteca de clases, de las cuales podrían usarse sólo unas pocas. La desventaja es que no puede garantizarse que todos los objetos posean una funcionalidad definible por el programador."

Debido a que muchos de los objetos que se diseñaron para el motor, tenían mucho en común, se optó por una jerarquía en árbol (ver Figura 9), puesto que se consideró necesario que todo el motor fuese exportable, es decir, que se pudiera transportar y usar fácilmente. Además los objetos gráficos requerían del uso de recursos compartidos y generales a todos estos, tal como el dispositivo de video y un identificador global.

El dispositivo de video fue utilizado para el proceso de dibujado (Rendering), mientras que el identificador se empleó para ayudar al control de los eventos generados por los mensajes enviados desde la aplicación hacia los diferentes objetos. Estos mensajes son capturados por clases que implementan los servicios expuestos en las interfaces, haciendo uso del Patrón Escuchador.

Sin embargo, aunque la jerarquía utilizada fue la de árbol, no fue posible que todas las clases heredaran de la raíz en común ya que no mostraban características similares, y es por eso que algunas clases no se desprenden de la clase base (ver Figura 9).

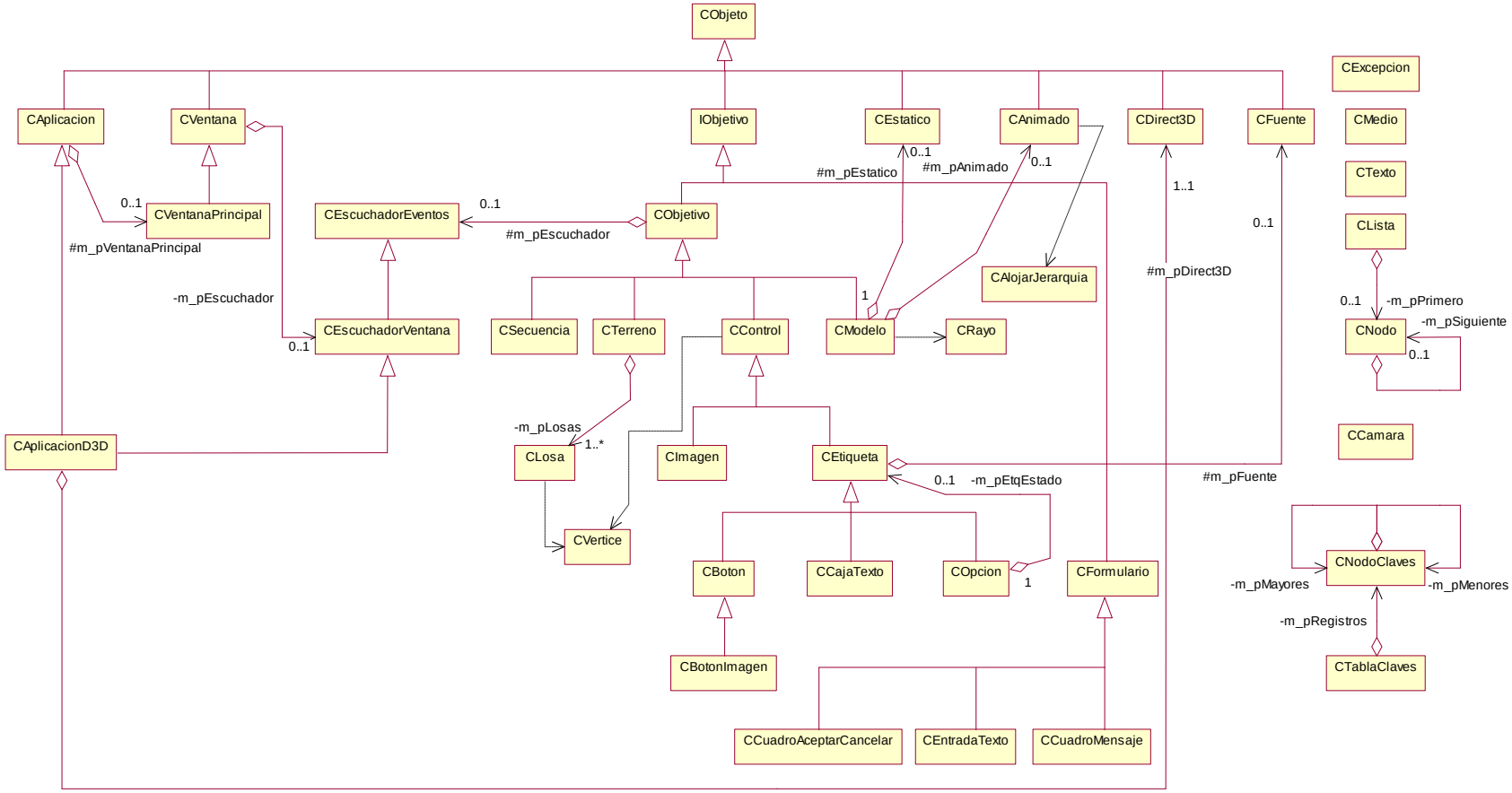

Figura 9. Estructura jerárquica del motor 3D.

3. 3. 1. 2. DIAGRAMA DE CLASES

Para entender el diagrama de clases del motor 3D que se muestra en la Figura 9, es necesario dividirlo de acuerdo a la función de sus clases, examinando en primera instancia la clase base de la jerarquía, seguido por las clases para la creación de aplicaciones, las clases para el manejo de eventos, las de manipulación de objetos 3D, controles, medios y finalmente detallar las correspondientes a utilidades.

3. 3. 1. 2. 1. CLASE BASE

Con el fin de que todas las clases gráficas pudieran tener acceso al dispositivo de video para su correspondiente dibujado y tuvieran un identificador único dentro de la aplicación que ayudara con el manejo de eventos y mensajes, se creó la clase CObjeto, la cual es la base de la jerarquía del motor 3D.

3. 3. 1. 2. 2. CLASES PARA LA CREACION DE APLICACIONES

Durante la creación de una aplicación que hace uso de las características de Direct3D, es necesario realizar una serie de pasos encaminados a registrar la clase de Windows, definir un procedimiento para el procesamiento de mensajes de Windows, crear la ventana e inicializar un dispositivo Direct3D. En el motor 3D, todas estas operaciones son efectuadas por medio de un conjunto de clases dedicadas a tal labor, las cuales se describen a continuación:

- CAplicacion: El propósito de esta clase es el de manejar la instancia de la aplicación, para lo cual se encarga de registrar la clase de Windows y finalizarla cuando termina, así como también de recibir los mensajes de Windows y enviarlos a la ventana principal.

- CVentana: Esta clase permite la creación de ventanas de Windows, tales como formularios, botones, etiquetas, etc., además define un procedimiento para el procesamiento de los mensajes, los cuales son manejados por el escuchador de la ventana.
- CVentanaPrincipal: Esta clase hereda de CVentana y representa la ventana principal para la aplicación de Windows.
- CDirect3D: La función de esta clase es la de interactuar con la interfaz IDirect3D con el fin de inicializarla, verificar las capacidades de(l) dispositivo(s) de video, obtener los modos de pantalla y las tasas de refresco soportadas por el mismo y finalmente crear el dispositivo Direct3D obteniendo un puntero a la interfaz IDirect3DDevice9.
- CAplicacionD3D: El objetivo de esta clase es el de facilitar la creación de aplicaciones con soporte para Direct3D, para lo cual hereda de las clases CAplicacion y CEscuchadorVentana y crea un objeto de la clase CDirect3D con el fin de inicializar el dispositivo. Adicionalmente, ofrece funcionalidades típicamente requeridas por este tipo de aplicaciones tales como llevar el tiempo transcurrido entre actualizaciones de la misma, calcular los ticks por segundo y permitir la inicialización de un material o una luz.

3. 3. 1. 2. 3. CLASES PARA EL MANEJO DE EVENTOS

La comunicación entre el sistema operativo Windows y sus aplicaciones se hace por medio de mensajes de diferentes tipos que son enviados a un procedimiento especial que debe definir la aplicación en el momento de registrar la clase Windows.

Para realizar el control de los mensajes de eventos que Windows pasa a la aplicación y más precisamente de los eventos generados por los objetos gráficos de la misma, se diseñó un conjunto de clases que permitiera realizar dicha tarea dentro del motor 3D. A continuación se describen las clases involucradas en dicho proceso y en la sección DIAGRAMAS DE SECUENCIA se presenta con más detalle el control de los mensajes de eventos dentro del motor 3D.

- CEscuchadorEventos: Interfaz que define un conjunto de métodos específicos para el control de los eventos relacionados con el ratón (mover, presionar un botón, liberar un botón, dar clic, girar la rueda) y el teclado (presionar o liberar una tecla).
- CEscuchadorVentana: En esta interfaz se definen los métodos para el control de los eventos propios de la ventana de la aplicación, como son crearse, activarse, desactivarse, cerrarse o ejecutar un comando.
- IObjetivo: Esta interfaz define los métodos por los cuales un objeto es informado del acontecimiento de un evento de ratón o teclado.
- CObjetivo: Clase base para los objetos que responden a eventos de ratón y teclado.

3. 3. 1. 2. 4. SECCIÓN DE OBJETOS 3D

El motor 3D fue concebido principalmente para facilitar la creación de aplicaciones con soporte para Direct3D, razón por la cual fue necesario diseñar una colección de clases que permitieran la creación y manipulación de objetos tridimensionales, tal como modelos animados y estáticos, rayos, secuencias de imágenes, terrenos y losas. En seguida se muestra una descripción de la funcionalidad de cada una de estas clases:

- CEstatico: Esta clase permite cargar y pintar una malla (mesh) estática (sin animación).
- CAnimado: El objetivo de esta clase es permitir cargar, manipular y pintar una malla con animación incluida.
- CAlojarJerarquia: Esta clase permite alojar toda la jerarquía con la que se construye un modelo animado y su importancia radica en las operaciones que debe realizar para crear o liberar un frame o un contenedor en el momento de animar el modelo.
- CModelo: Esta clase representa un modelo animado o estático en el espacio tridimensional y hereda de la clase CObjetivo, por lo cual los objetos de este tipo puede responder a eventos de ratón y teclado.
- CRayo: Esta clase se encarga de crear un 'rayo', para que pueda ser utilizado en el momento en que se desee calcular si se hace clic sobre un objeto 3D.

- CSecuencia: Esta clase permite la creación de un objeto que hace uso de una secuencia de texturas para crear la sensación de estar visualizando una textura animada en el espacio 3D.
- CTerreno: La función de esta clase es la de facilitar la construcción de un terreno en 3D que hace uso de losas. Un ejemplo de aplicación de este tipo de objeto es los juegos de estrategia.
- CLosa: Esta clase representa una losa dentro del terreno 3D y guarda información relacionada con la altura y posición del vértice.

3. 3. 1. 2. 5. SECCIÓN DE CONTROLES

Generalmente, una aplicación que usa objetos 3D, necesita también hacer uso de objetos en dos dimensiones para la construcción de interfaces de usuario. Estos objetos pueden ser botones, imágenes, etiquetas, diálogos y otros.

En el motor 3D, se diseñaron varias clases con las cuales, un desarrollador puede construir una interfaz de usuario enriquecida con controles en dos dimensiones. Estas clases se presentan a continuación.

- CControl: Clase base que permite la creación de objetos que se muestran en dos dimensiones haciendo uso de Direct3D. Cada control posee tres estados: libre (el control se encuentra libre), abajo (el control esta presionado), arriba (el control no está presionado, pero el ratón está sobre él) y click (se hizo click sobre el control).
- CImagen: Esta clase es un control que permite la visualización de imágenes.
- CEtiqueta: Esta clase permite mostrar un texto en la pantalla.
- CBoton: Este control permite la creación y manipulación de botones.
- CBotonImagen: Esta clase es un botón que permite mostrar una imagen diferente en cada uno de los tres estados gráficos del control: libre, abajo o click, arriba.
- CCajaTexto: Mediante esta clase se puede crear una caja que permite el ingreso de texto por parte del usuario.
- COpcion: La función de esta clase es la de permitir la construcción de un objeto que tiene un cuadro de chequeo y un texto.

- CFormulario: Esta clase es un contenedor de controles y su principal función es la de facilitar el paso de mensajes a los objetos que contiene.
- CCuadroMensaje: Mediante esta clase se puede mostrar un dialogo que contiene un mensaje y un botón Aceptar.
- CEntradaTexto: Este es un dialogo para solicitar al usuario la entrada de un texto.
- CCuadroAceptarCancelar: Este dialogo permite mostrar un mensaje al usuario con las opciones Aceptar y Cancelar.

3. 3. 1. 2. 6. SECCIÓN DE MEDIOS

Cuando se construye una aplicación que hace uso de características tales como el 3D, puede resultar útil tener la posibilidad de incluir audio o video a la misma. Por tal motivo, dentro del motor 3D se incluye una clase que se encarga de crear y controlar lo referente a flujos multimedia, tanto de audio, como de video.

Dicha clase se denomina CMedio y sirve para la manipulación de contenido multimedia, tal como videos y música en formatos avi, mpeg, wav, mp3, etc. Para lograr esto, la clase hace uso de la API DirectShow, la cual se describe con más detalle en el Anexo F.

3. 3. 1. 2. 7. SECCIÓN DE UTILIDADES

Esta sección comprende todas las clases que brindan utilidades complementarias a la funcionalidad de las clases antes mencionadas, y que igualmente son importantes dentro de la construcción de aplicaciones Direct3D. A continuación se detallan las clases.

- CExcepcion: Clase genérica para el manejo de excepciones.
- CFuente: Esta clase permite definir un tipo de fuente para utilizar posteriormente en etiquetas y demás objetos 2D.
- CTexto: Esta clase sirve para manipular cadenas de texto.

- CCamara: Permite la creación de un objeto que guarda información de una cámara en el espacio 3D.
- CLista: Mediante esta clase se puede manejar una lista enlazada de punteros a la información de otras clases.
- CNodo: Define un nodo dentro de una lista enlazada dinámica.
- CTablaClaves: Esta clase permite construir una tabla de claves, asociando un objeto de cualquier tipo con una clave de búsqueda.
- CNodoClaves: Esta clase guarda la información de un registro dentro de la tabla de claves.

3. 3. 1. 3. DIAGRAMAS DE SECUENCIA

Durante la etapa de diseño del motor, no se estimó necesaria la realización de diagramas de secuencia para todas las clases. Sin embargo, fue conveniente crear un diagrama de secuencia para poder tener una mejor visualización de cómo se lleva a cabo el proceso de control de los mensajes de eventos generados dentro del motor 3D.

En el motor 3D, los mensajes son recibidos por el procedimiento de ventana de la clase CAplicación y desde allí enviados a la ventana principal, la cual determina el tipo del mensaje e informa a su escuchador de ventana. El escuchador de ventana, generalmente debe informar a los controles apropiados del evento. Cuando un control es informado de un evento, éste realiza algunas comprobaciones, como por ejemplo si el ratón se encuentra sobre el objeto (cuando el evento es de ratón). De ser así informa a su respectivo escuchador para que este controle apropiadamente el evento generado. Todo lo anterior puede visualizarse en el diagrama mostrado en la Figura 10 en el cual se supone que el evento generado es un botón del ratón que se presionó.

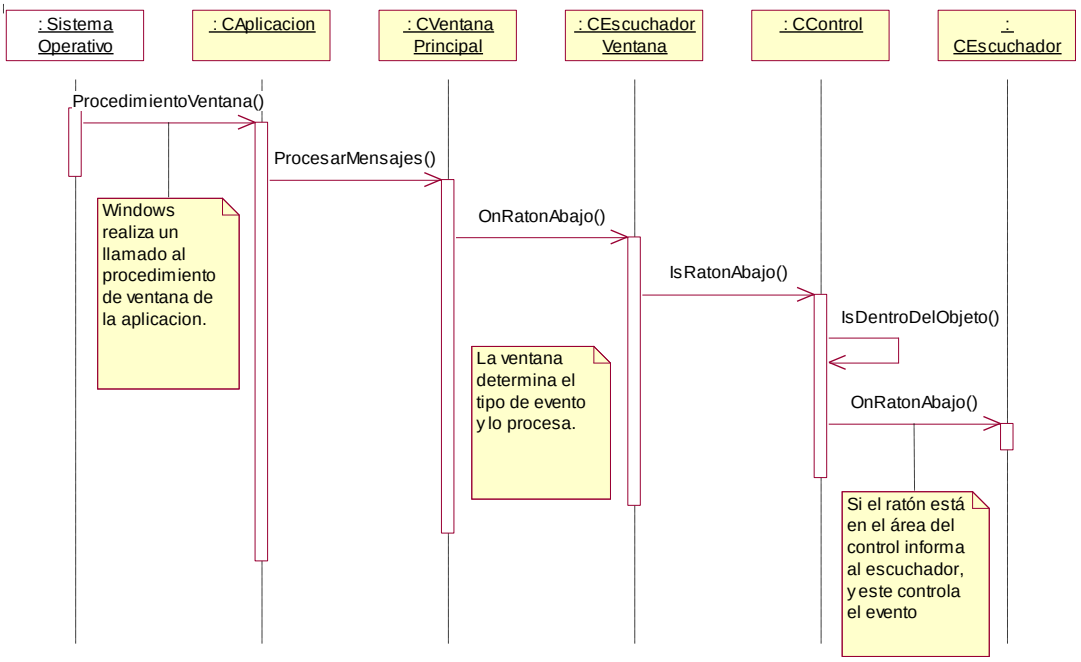

Figura 10. Diagrama de secuencia del manejo de mensajes.

3. 3. 2. VISOR DE MODELOS

Después de determinar que el Motor 3D cumplía con la funcionalidad necesaria para la creación del juego surgieron dos inconvenientes:

1. Era necesario tener un punto en común con el diseñador gráfico, de tal manera que los personajes que modelara, pudieran ser cargados y mostrados adecuadamente por el motor, ya que cuando se realizaron los primeros modelos se encontraron problemas como intercambio de ejes, animaciones mal hechas y problemas en el manejo de luces. Estos problemas se originaron principalmente porque para poder exportar los modelos desde 3D Studio Max (que fue la herramienta utilizada por el diseñador gráfico) al formato de DirectX, era necesaria la utilización de un plug – in (Panda DirectX) que en principio fue difícil de configurar adecuadamente.
2. Hacía falta un módulo que permitiera probar el adaptador de video, puesto que no en todos los equipos era posible iniciar Direct3D, que es la base para el motor.

La solución planteada fue la de crear una aplicación en la cual se pudiera probar los modelos desarrollados por el diseñador gráfico y que además permitiera configurar el adaptador de video.

Esta aplicación por supuesto, daría las bases para entender aún más los conceptos relacionados con matrices de transformación, luces, control de animaciones y por supuesto configuración del adaptador de video. En la Figura 11 se muestra el diagrama de casos de uso para el visor de modelos, donde se establece la funcionalidad básica de la aplicación.

En primera instancia se puede apreciar que el usuario puede cargar un modelo 3D, esto es, buscar un archivo en formato de DirectX (*.x), leerlo y mostrarlo (pintarlo). Después de esto se puede manipular el modelo 3D, en cuando a realizarle transformaciones (como mover, rotar o escalar), iluminarlo o no, cambiar la forma de presentación (en malla, sólido o con una esfera contenedora), o controlar las animaciones si ese era el caso. Adicionalmente, permitía configurar el adaptador de video para que trabajara en distintos modos de pantalla.

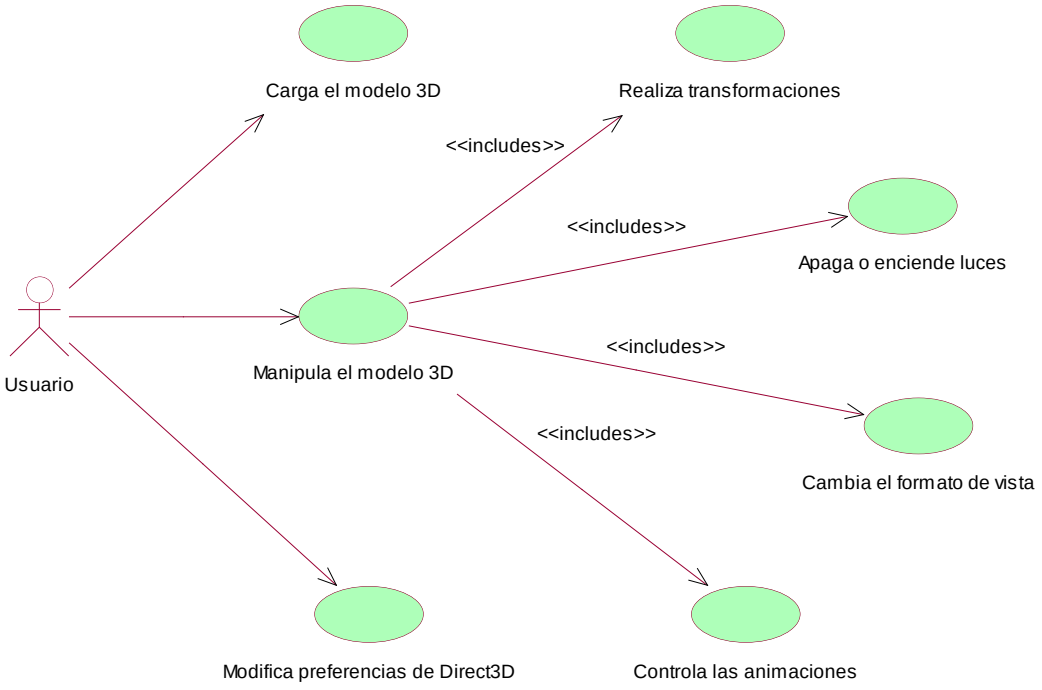

Figura 11. Diagrama de casos de uso para el visor de modelos

Para la implementación del visor de modelos se tuvo en cuenta que el motor 3D ya soportaba y proporcionaba la funcionalidad necesaria para su creación (del visor), por lo que se utilizaron las clases que se detallan en el diagrama mostrado en la Figura 12.

Como se aprecia el modelo es bastante simple y solamente consideró tres clases externas al motor, pero que de alguna manera se relacionaban con este mediante la especificación de clases que soportaban toda la funcionalidad requerida.

El diagrama de clases muestra como se creó una clase llamada CVisor3D que heredaba de CAplicacionD3D y que contenía una clase de tipo CModelo y otra de tipo CAnimado. Aquí se encuentra la funcionalidad principal del visor, es decir, la clase CVisor3D permitiría crear la aplicación, mientras que CModelo y CAnimado, estarían encargadas de facilitar la manipulación de los modelos tridimensionales hechos por el diseñador gráfico. Además se diseñó otra clase que se encargará de mostrar un cuadro de diálogo para definir las opciones de creación del adaptador de video. Esta clase se encargaría de cargar las opciones desde un archivo de configuración, facilitaría la configuración y también guardaría los cambios realizados.

Pensando en el futuro, se decidió que la clase CDlgPreferencias, quedara por fuera del visor, porque sería necesaria para la configuración del adaptador de video de otras aplicaciones como el mismo juego.

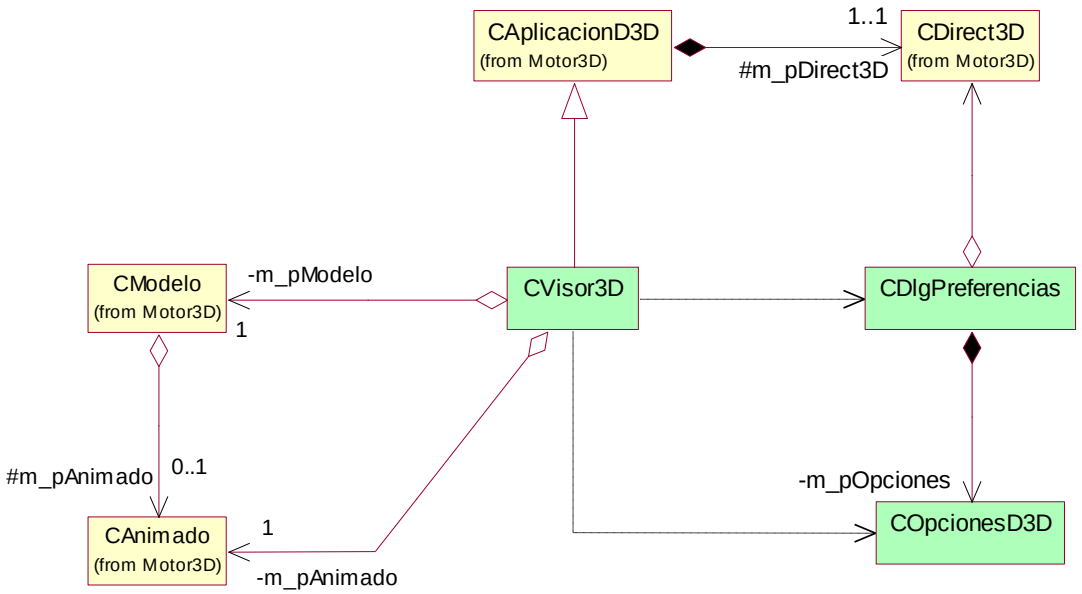

Figura 12. Diagrama de clases del visor de modelos

3. 3. 3. EDITOR DE TERRENOS

Cuando se comenzó a trabajar con el motor, se descubrió que el proceso de edición de un terreno es un trabajo que resultaría engorroso y difícil de realizar si no se cuenta con una herramienta que lo facilite, puesto que es necesario determinar la altura de cada vértice de las losas, la textura que usa cada una de ellas y la ubicación de los modelos de ambientación. Además, esta dificultad es directamente proporcional al tamaño del terreno debido a que con un terreno más grande se debe usar un número mayor de losas.

Se plantea entonces, la creación del editor de terrenos como una aplicación que ayuda a agilizar el proceso de ubicar texturas y ambientación y editar las alturas de un terreno. Para lograr esto, en la etapa de diseño se delimitó la funcionalidad esperada de la aplicación mediante un diagrama de casos de uso, el cual se muestra en la Figura 13. De dicho diagrama cabe resaltar que el actor de todos los casos de uso es llamado Usuario, el cual puede ser el desarrollador del juego u otro integrante del proyecto de

desarrollo, por ejemplo las personas encargadas de la labor de modelado de los personajes y la ambientación. En este sentido, es de resaltar, que el editor de terrenos fue de gran utilidad para el diseñador gráfico, porque le permitió escalar adecuadamente los modelos de los personajes y de la ambientación de acuerdo al tamaño del terreno y además le ayudo a visualizar el aspecto más acertado para las texturas que utilizarían las losas del terreno.

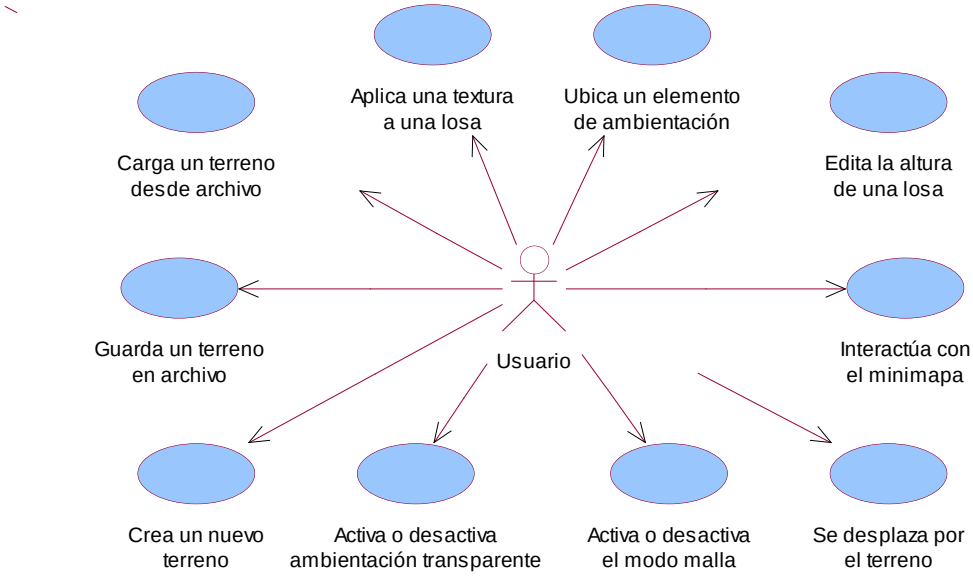

Figura 13. Diagrama de casos de uso del editor de terreno.

La aplicación permite, en primer lugar, que el usuario cargue un terreno ya existente o guarde el terreno que se encuentra editando, desde o hacia un archivo, respectivamente. Además, el usuario tiene la posibilidad de crear un nuevo terreno pudiendo escoger el tamaño deseado (pequeño, mediano, grande o enorme) y también si desea usar un archivo con información para las alturas del terreno o simplemente desea que éste comience plano.

Ya en el campo de la edición del terreno, el usuario tiene la posibilidad de aplicar la textura que desee a una losa seleccionada, ubicar un elemento o modelo de ambientación y también cambiar la altura de los vértices de la misma con el fin de generar formaciones montañosas o variaciones en la altura del terreno.

Además, el usuario puede interactuar con el mini mapa dando clic sobre él, de tal forma que la cámara de la aplicación se ubique en la parte del terreno que se selecciona. Otra forma en la que el usuario puede cambiar la parte del terreno que se encuentra visualizando es a través del uso del teclado, de modo tal que la cámara se desplace en una dirección deseada (adelante, atrás, hacia la izquierda o hacia la derecha).

Finalmente, si el usuario lo desea, puede hacer que la aplicación muestre el terreno y los objetos de ambientación en modo de malla, o que la ambientación se pinte de modo transparente. El modo malla hace que se visualicen simplemente el conjunto de líneas que unen los vértices del modelo tridimensional, mientras que el modo transparente hace que los objetos de ambientación se pinten de tal forma que permitan ver los objetos que hay detrás.

Con el fin de implementar la funcionalidad del editor de terreno descrita anteriormente, se diseñó un conjunto de clases que hacen uso de los artefactos generados para el motor 3D, las cuales se muestran en el diagrama de clases de la Figura 14.

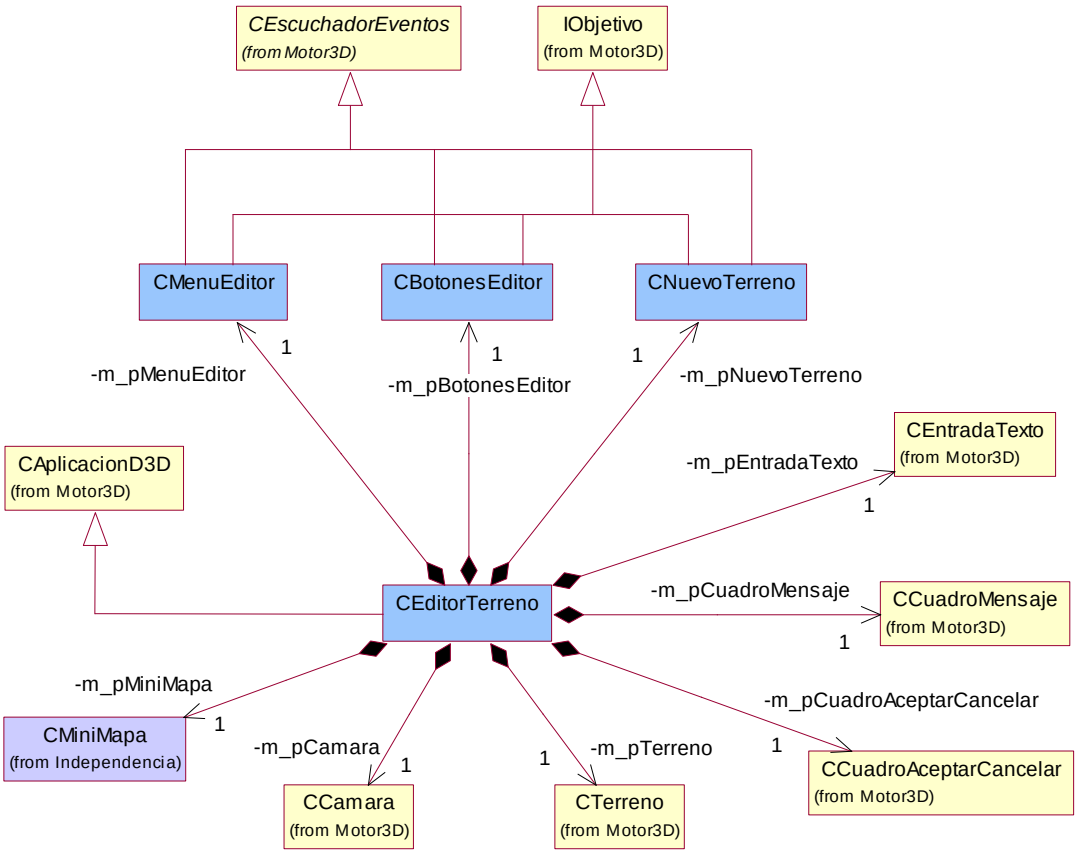

Figura 14. Diagrama de clases del editor de terreno.

En dicho diagrama se puede destacar que las clases que se encuentran en color amarillo, son clases derivadas del motor 3D y la clase de color violeta (CMiniMapa) es necesaria para el juego Independencia, mientras que las clases de color azul claro son específicas al editor de terreno. De estas últimas, la clase CEditorTerreno es la clase responsable de la aplicación ya que hereda de CAplicacionD3D y por tanto es el escuchador de eventos de los objetos gráficos de la misma. Esta clase es la responsable de crear el mini mapa, la cámara y el terreno que el usuario edita. Hace uso también, de un objeto CEntradaTexto para pedir el nombre del archivo de terreno, cuando el usuario desea cargar o guardar un terreno, de un objeto CCuadroMensaje para mostrar mensajes de información al usuario, y de un objeto CCuadroAceptarCancelar para preguntar al usuario si desea sobrescribir un archivo de

terreno cuando este ya existe. Finalmente, esta clase crea los objetos de tipo CMenuEditor, CBotonesEditor y CNuevoTerreno.

La clase CMenuEditor es la encargada de mostrar un menú al usuario con las opciones para crear un nuevo terreno, cargar uno ya existente, guardar el actual en un archivo o salir de la aplicación. Las opciones para seleccionar una textura o un modelo de ambientación o editar la altura de una losa, son presentadas al usuario a través de la clase CBotonesEditor. Finalmente, cuando el usuario decide crear un nuevo terreno, se le muestra un diálogo en el que puede seleccionar el tamaño e ingresar el nombre de un archivo con información para las alturas del terreno, si así lo desea. La clase responsable de mostrar este dialogo es la clase CNuevoTerreno.

3. 3. 4. BUSCADOR DE CAMINOS

A medida que iba creciendo el proyecto e iba tomando forma, apareció un nuevo reto: lograr que los personajes que iban a intervenir dentro del juego pudieran desplazarse dentro del terreno definido para éste. La solución a este dilema fue el estudio y adopción del algoritmo A estrella para la búsqueda de caminos, puesto que combina lo mejor de otros algoritmos como es: una búsqueda rápida y cálculo de la mejor ruta. Se procedió entonces a diseñar una aplicación que permitiera hacer pruebas para la búsqueda de caminos y evaluar el rendimiento de la implementación que se haría.

Una vez realizada la primera fase de implementación y hechas las pruebas de rendimiento, se determinó que el algoritmo presentaba algunas falencias que debían corregirse, por lo que en otra iteración dentro del proceso de desarrollo, se buscaron soluciones para estos problemas. Específicamente se encontró que el algoritmo en determinadas situaciones llegaba a visitar un número muy elevado de nodos, razón por la cual, su tiempo de respuesta se incrementaba considerablemente (haciendo muy lento el juego). La solución para esto fue limitar el número de nodos visitados de acuerdo con el tamaño del terreno.

En este proceso de prueba, aprendizaje, experiencia y mejora, fue de vital importancia la aplicación para la búsqueda de caminos, ya que permitió encontrar un algoritmo más óptimo, que posteriormente se utilizó dentro del juego. La funcionalidad de esta aplicación se muestra en el diagrama de casos de uso detallado más adelante y posteriormente se exhibe el diagrama de clases en el que se presentan las diferentes relaciones entre las clases utilizadas para la creación de la misma.

Para la aplicación de búsqueda de caminos, se consideró que debía permitir cargar distintos terrenos con el fin probar diferentes escenarios a los cuales se podría enfrentar el algoritmo. Una vez cargado el terreno, el desarrollador debería tener la posibilidad de desplazarse por el terreno usando el teclado para la navegación o interactuando con un mini mapa. Finalmente, era necesario que la aplicación permitiera al desarrollador ordenarle al modelo moverse hasta una determinada posición, utilizando por supuesto, el algoritmo A estrella. En este último paso el sistema debería mostrar las estadísticas relacionadas con la búsqueda del camino, tales como el número de nodos abiertos y cerrados usados por el algoritmo, el número de nodos de la ruta y el tiempo empleado para realizar el cálculo. Toda esta funcionalidad se expone en el diagrama de casos de uso mostrado en la Figura 15.

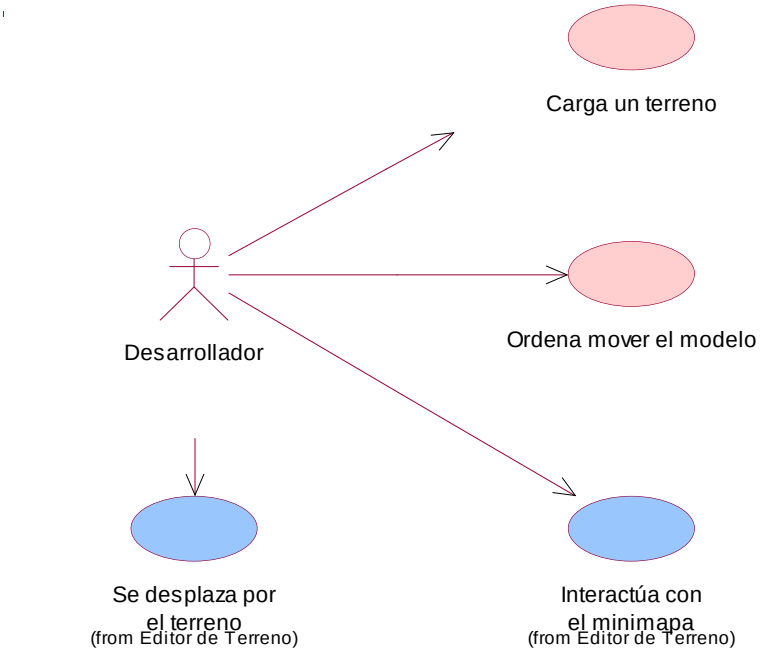

Figura 15. Diagrama de casos de uso del buscador de caminos

La construcción de la aplicación para la búsqueda de caminos, continuó con la creación de un diagrama de clases que permitiría establecer las clases usadas para la misma y las relaciones entre ellas. Este diagrama se muestra en la Figura 16, en la cual se observa que la clase CAplBuscadorCamino es la clase que hereda de CAplicacionD3D y por tanto es la encargada del control de la aplicación. Además, la clase crea el terreno, la cámara, el mini mapa, el modelo y su animado, un dialogo CEntradaTexto para leer el nombre del archivo de terreno que se desee cargar, un dialogo CCuadroMensaje para mostrar un mensaje cuando no se puede cargar el archivo y una etiqueta para mostrar las estadísticas de los cálculos de camino.

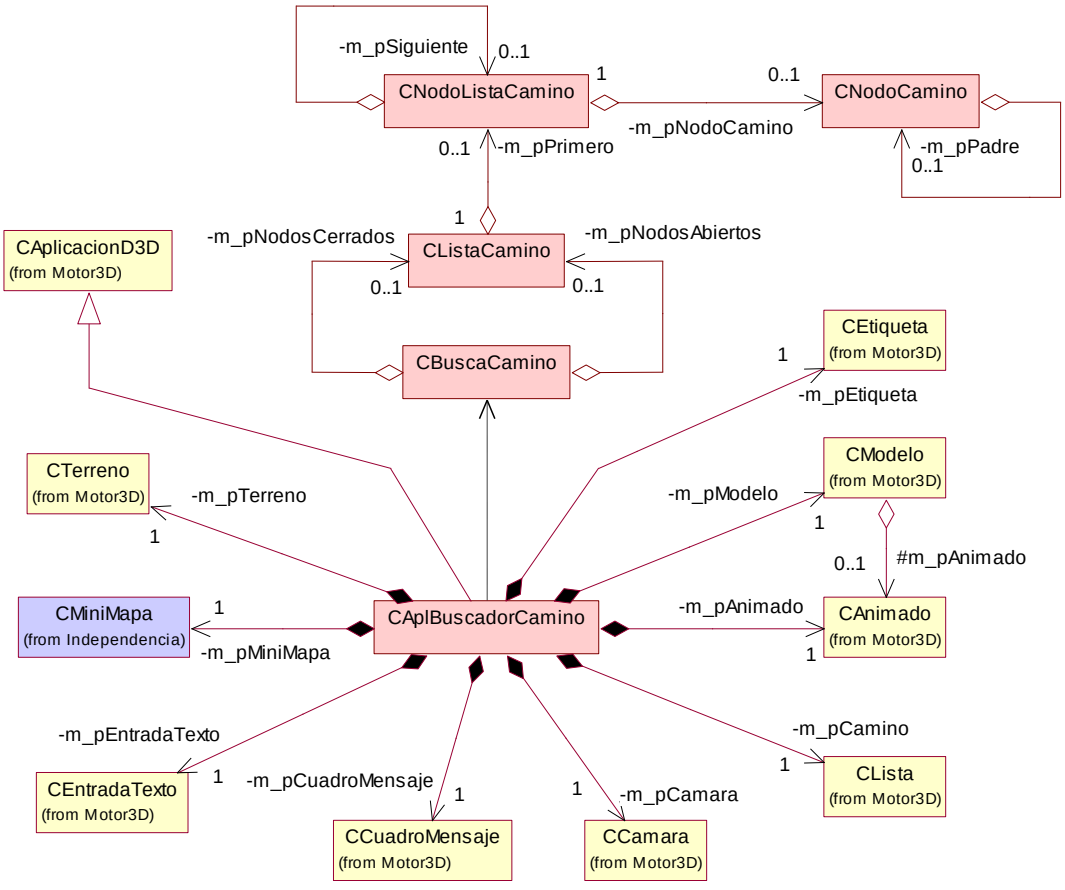

Figura 16. Diagrama de clases del buscador de caminos

Cuando el usuario ordena a la aplicación que mueva el modelo desde un punto a otro dentro del terreno, la clase CAplBuscadorCamino hace uso de un objeto del tipo CBuscaCamino, el cual es el encargado de realizar el cálculo de la ruta entre los puntos. Para realizar esa labor, la clase hace uso de dos listas del tipo CListaCamino con el fin de mantener los nodos cerrados y los nodos abiertos por el algoritmo durante el proceso. Cada nodo dentro de estas listas es almacenado por un objeto del tipo CNodoListaCamino y la información relacionada con el nodo de la ruta, como su posición en el terreno, su antecesor y su costo, son guardadas en un objeto del tipo CNodoCamino, el cual sirve, posteriormente, para generar una lista del tipo CLista con los nodos de la ruta encontrada. Esta última lista es almacenada por la aplicación para que posteriormente el modelo pueda seguirla.

3. 3. 5. INDEPENDENCIA

Llegada la hora de abordar el diseño de Independencia, se tenía la experiencia y el conocimiento fruto del esfuerzo realizado en los anteriores desarrollos, por lo que se nutrió de lo explorado en Defensor 3D, y los diseños realizados para el visor de modelos, el editor de terrenos y el buscador de caminos.

La experiencia del desarrollo de Defensor 3D fortaleció el diseño del Motor 3D, puesto que no solo se obtuvieron las bases que permitieron la definición jerárquica del motor, sino que se obtuvo la suficiente madurez y experiencia para definir las clases apropiadas para las diferentes tareas que debía soportar, como manejo de gráficas en dos y tres dimensiones, carga y reproducción de flujos de medios, control de aplicaciones y mensajes de eventos.

En cuanto a los diseños previos, se puede resaltar que del visor de modelos se consideró lo concerniente al manejo de modelos que separaban la animación de la lógica del mismo (tal como la posición, rotación, escala, animación actual, tiempo de animación, etc.), del editor de terreno se rescató lo referente al manejo del terreno, el movimiento de la cámara, y el encapsulamiento de la funcionalidad (considerando una clase que se encargara de funciones comunes, por ejemplo una clase para el manejo

del menú que controlaba los eventos para los objetos que hacían parte de éste). Del buscador de caminos se utilizó todo el bagaje y la experiencia adquiridos tratando de robustecer el algoritmo A estrella.

Aunque se contaba con diseños que se podían reutilizar en parte, hacia falta abordar un problema que ya se había considerado en el análisis, como era el del comportamiento de las unidades. En esa etapa se había propuesto la utilización de la metodología de GAIA para la definición de agentes que modelaran el comportamiento de los personajes en el juego y en este diseño se identificaron los distintos tipos de agentes que intervendrían en el sistema y los servicios que deberían implementar. Posteriormente, el diseño de GAIA se modeló de tal forma que se pudiera integrar al diseño realizado con UML y de esta manera facilitar la tarea de implementación.

3. 3. 5. 1. DISEÑO GAIA

El proceso de diseño en la metodología GAIA involucra la generación de 3 modelos: El Modelo de Agentes, El Modelo de Servicios y El Modelo de Conocidos.

3. 3. 5. 1. 1. EL MODELO DE AGENTES

El modelo de agentes en la Figura 17 indica que el sistema únicamente contendrá el tipo de agente *AgenteUnidad*, el cual asumirá el rol *Unidad*. Además, en el modelo se puede ver que habrá una o más instancias de este tipo de agente en el sistema en tiempo de ejecución.

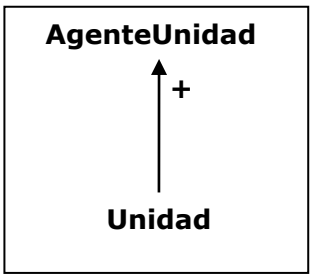

Figura 17. Modelo de agentes

3. 3. 5. 1. 2. EL MODELO DE SERVICIOS

La Tabla 9 muestra el modelo de servicios implementados por el tipo de agente AgenteUnidad.

Servicio	**Entradas**	**Salidas**	**Precondiciones**	**Post-Condiciones**
Vigilar	Mapa	Actividad o Protocolo	PuntosVida > 0	La actividad de la unidad es Vigilar, Atacar o Morir
Caminar	Destino	Posición actualizada	PuntosVida > 0 Λ Camino ≠ nil	La posición final esté dentro del terreno.
Morir	-	-	PuntosVida <= 0	-
Atacar	UnidadEnemiga	La unidad enemiga recibe el ataque.	PuntosVida > 0 Λ UnidadEnemiga ≠ nil Λ TiempoUltimoAtaque > T	**-**
Recibir Ataque	ValorAtaque	PuntosVida	PuntosVida > 0	Puntos de vida disminuidos
EmitirGritoComunero	-	Notificación a las unidades amigas cercanas.	PuntosVida > 0 Λ TiempoUltimoGrito > t	No se puede emitir el grito comunero nuevamente hasta que transcurra un tiempo t.
EscucharGritoComunero	Tiempo	PuntosAtaque	PuntosVida > 0	Puntos de ataque incrementados

Tabla 9. Modelo de servicios.

3. 3. 5. 1. 3. EL MODELO DE CONOCIDOS

El modelo de la Figura 18 muestra que existe un enlace de comunicación entre el tipo de agente AgenteUnidad y otro(s) agente(s) del mismo tipo.

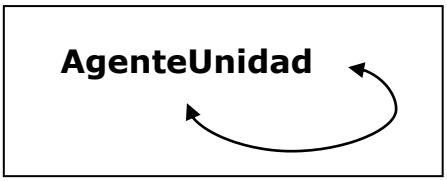

Figura 18. Modelo de conocidos

3. 3. 5. 2. DISEÑO UML

Luego de haber trabajado en la definición de los tipos de agentes que implementarían el comportamiento de los personajes en el juego mediante la metodología GAIA, fue necesario abordar la funcionalidad que expondría la aplicación hacia el usuario final o jugador. Se definieron, entonces, casos de uso de diseño y diagramas de clases de manera que se lograra modelar los requerimientos básicos que debe contener todo juego, así como los propios de Independencia y que se expresaron mediante la definición del guión.

El primer diagrama de casos de uso que se muestra en la Figura 19 detalla la funcionalidad en general del juego. Ahí se puede apreciar como el jugador puede en primera instancia, configurar el adaptador de video, esto es, determinar cuál resolución de pantalla y formato de dibujado es el apropiado para la tarjeta de video que posee. Este caso de uso inicia cuando el sistema determina que no hay una configuración para el adaptador de video y en ese momento se muestra un cuadro de diálogo que permite realizar la configuración.

Posteriormente, el jugador también puede configurar las opciones dentro del juego, tal y como son el sonido y la música, así como las propias del juego. Dentro de la configuración del sonido y la música se estableció que el jugador pudiera al menos

determinar el volumen tanto de la música como de los efectos de sonido empleados en el juego y mediante las opciones propias del juego se buscó que el jugador pudiera establecer el nivel de dificultad, pudiendo elegir entre fácil, medio o difícil.

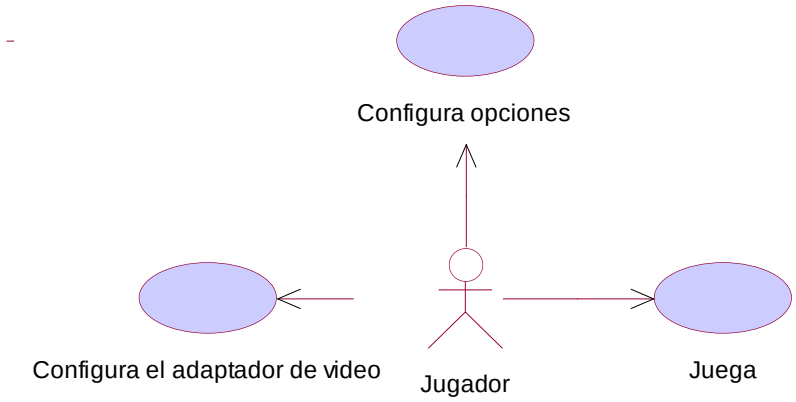

Figura 19. Diagrama de casos de uso general de Independencia.

La otra acción que puede realizar el usuario es jugar en modo campaña la misión implementada para el juego, para lo cual el jugador en primer lugar puede visualizar la historia y los objetivos de la misión y luego pasar a ejecutarla. Todas las acciones que puede emprender el jugador en esta parte de la aplicación, se resumen en el diagrama de casos de uso mostrado en la Figura 20. En él se afirma que el jugador puede seleccionar una unidad, lo cual implica desplazar el puntero del ratón hasta que se encuentre sobre la unidad y luego hacer clic. Una vez seleccionada la unidad, el jugador puede moverla, esto se logra haciendo clic derecho sobre la parte del terreno a la cual desea mover la unidad. El jugador puede también, ordenar a la unidad seleccionada que ataque, para lo cual debe hacer clic derecho sobre la unidad enemiga que desea atacar. Adicionalmente, Si la unidad seleccionada es el prócer, el usuario puede ordenarle que emita el grito comunero, esto se hace dando clic sobre el botón para lanzar el poder especial del prócer. Este botón estará activo siempre y cuando el prócer pueda emitir el grito comunero, ya que cada vez que lo haga deberá esperar unos segundos hasta que pueda volverlo a hacer.

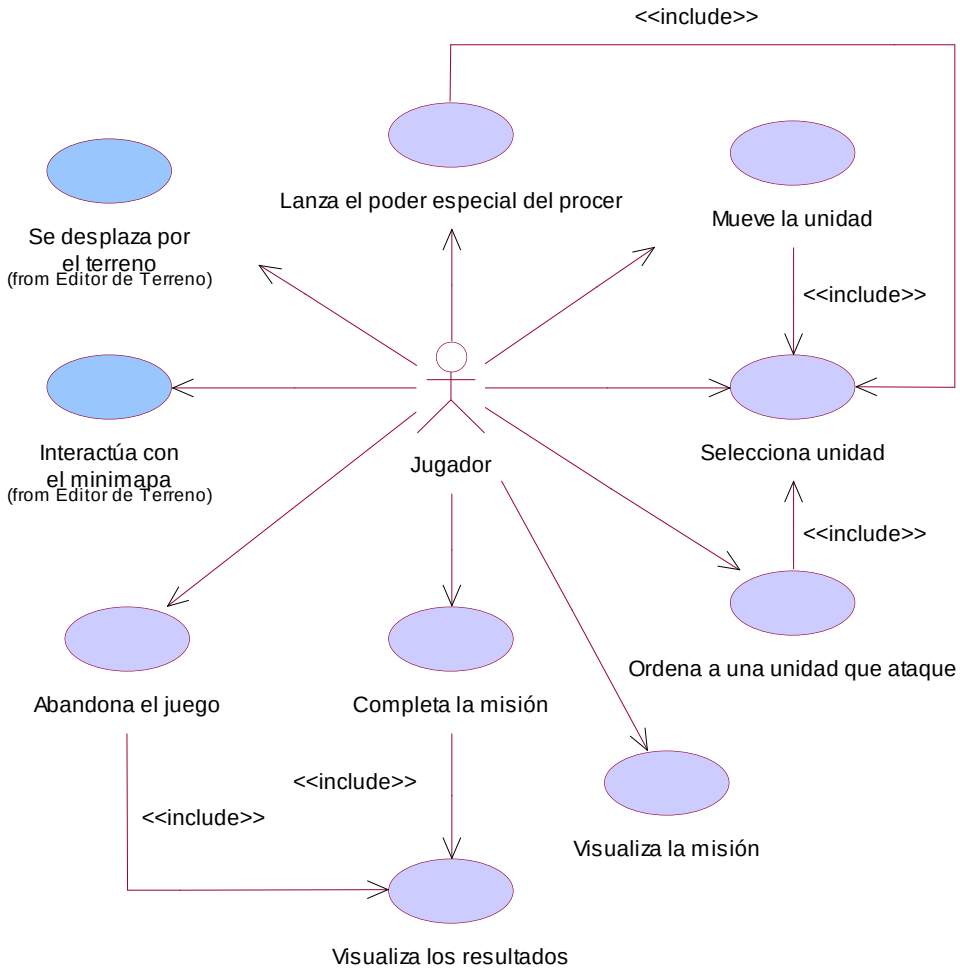

Figura 20. Diagrama de casos de uso para la etapa de Juego de Independencia.

Una vez que el sistema valida que el jugador ha cumplido con los objetivos de la misión, se dice que ésta se completó, por lo cual se le presenta al usuario una pantalla en la cual puede visualizar los resultados de la misión. Estos resultados también son mostrados al jugador cuando éste decide abandonar el juego sin que haya terminado.

En la etapa del juego, el jugador también puede interactuar con el mini mapa y desplazarse por el terreno, tal como lo hace en el editor de terrenos, además cuando lo desee, puede visualizar la misión para luego continuar jugando.

Modelados los casos de uso, tanto de la aplicación en general, como el juego en sí, se procedió a crear los diagramas de clases necesarios para definir la estructura de la aplicación. Estos se dividen en dos: en el primero de ellos (Figura 21) se muestran a

modo general, las clases de cada uno de las diferentes pantallas de la aplicación y en el segundo (Figura 22), se detallan las clases involucradas en la campaña del juego. En ambos diagramas se reutilizaron clases que ya se habían diseñado en aplicaciones anteriores, tal y como es el caso de la clase CDlgPreferencias creada en el visor de modelos y CBuscaCamino diseñada en el buscador de caminos.

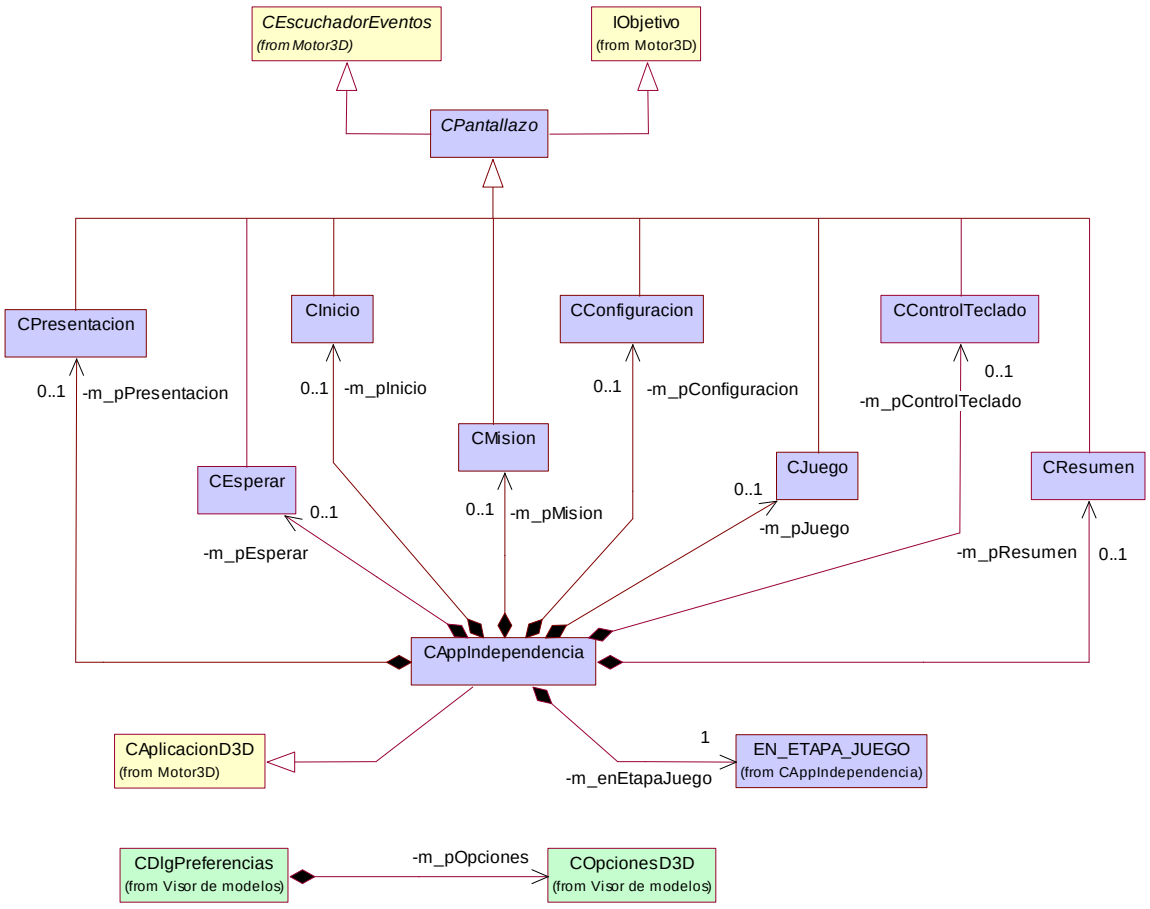

Figura 21. Diagrama de clases de aplicación de Independencia.

El diagrama de clases mostrado en la Figura 21 es fruto de la experiencia y el conocimiento adquiridos en las aplicaciones anteriores donde se determinó que la funcionalidad ofrecida al usuario por una pantalla de la aplicación debería estar encapsulada en una clase que a su vez controlara la lógica necesaria para brindar el servicio. En este caso, mediante la clase CPantallazo se obtiene la funcionalidad necesaria para recibir mensajes desde la aplicación (CAppIndependencia) y pasarlos a cada uno de objetos que se hayan definido en una clase descendiente. Estas clases que

heredan de CPantallazo, a su vez sirven como escuchadores de eventos de los objetos que contienen y se encargan del control de la lógica.

Cabe anotar que CAppIndependencia hace uso de una máquina de estados finitos para definir las diferentes etapas de la aplicación, donde cada uno de los estados de la máquina se representa por un valor definido dentro de la enumeración EN_ETAPA_JUEGO. En cada etapa se identifica la pantalla que se debe mostrar al jugador y los cambios entre estados se realizan por medio de un mensaje enviado desde la clase que muestra la pantalla actual hacia la aplicación.

Cuando la aplicación inicia se muestra la primera etapa que es la presentación (CPresentacion), seguida por un menú de inicio (CInicio) desde donde se puede configurar las opciones del juego (CConfiguracion) o iniciar el juego en modo campaña, para lo cual se muestra la misión (CMision) y se procede luego al desarrollo del juego como tal (CJuego). Una vez terminado el juego se muestra el resumen del mismo (CResumen).

Respecto al diseño de la etapa de juego, se debe apreciar que se utilizó el modelado hecho con GAIA para definir las clases mediante las cuales se crearían los agentes que modelan el comportamiento de los personajes. Estas clases, posteriormente se integrarían con el resto del diseño del juego y se verían reflejadas en la clase CUnidad y sus respectivas especializaciones, las cuales deberían implementar los servicios definidos para el tipo de agente AgenteUnidad.

Por tratarse de un agente, la clase CUnidad debía tener capacidad de procesamiento autónomo, por lo cual se le definió el método Ejecutar( ), mediante el cual la aplicación le puede ceder el control al agente para que éste realice sus procesos.

En el diagrama de la Figura 22 se muestran las clases de la etapa de juego de Independencia, incluyendo la clase CUnidad y sus descendientes. La campaña del juego se diseñó mediante la clase CJuego, que contiene información del terreno, la cámara, el mini mapa, la lista de unidades comuneras y españolas, la música y los sonidos, y la ayuda.

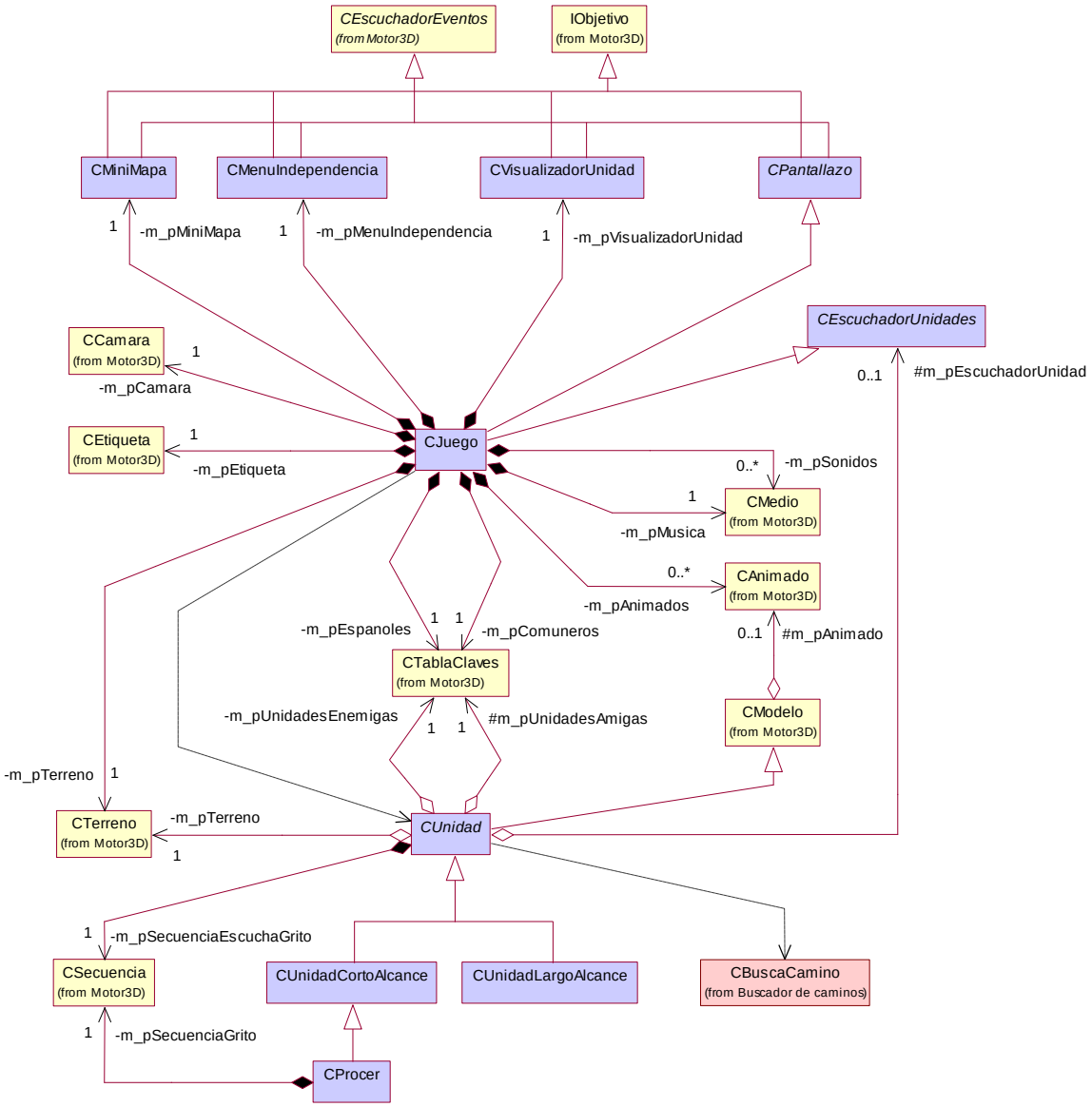

Figura 22. Diagrama de clases de juego de Independencia

3. 4. IMPLEMENTACIÓN

Al enfrentar la etapa de implementación la primera tarea que se realizó fue la de definir claramente un documento de estándares de programación y documentación del código fuente generado para el proyecto (Ver ANEXO A), esto con el fin de tener las reglas claras para el nombrado de clases y variables, la documentación y el control de

cambios entre las modificaciones hechas por parte de los desarrolladores. Gracias al seguimiento de lo estipulado en dicho documento y al uso de Visual Source Safe para comparar los archivos de código, se facilitó la tarea de unificar el trabajo realizado individualmente.

En esta sección se hará énfasis principalmente en los problemas de implementación que se presentaron en las diferentes aplicaciones que se desarrollaron y las soluciones que se dieron, antes que hacer un resumen de clases y mostrar su codificación. De esta manera, se abordarán en primera instancia los problemas que surgieron en la creación del Motor 3D, para luego continuar con los presentados en el visor de modelos, el editor de terrenos, la aplicación para búsqueda de caminos y finalizar con los que se dieron en la creación de Independencia.

3. 4. 1. CREACIÓN DEL MOTOR 3D

En el diseño del Motor 3D, había un requerimiento que especificaba que este debía ser independiente del juego, lo cual suponía una librería aparte que soportara la funcionalidad necesaria para la creación de juegos. La solución a este requerimiento fue la creación del Motor 3D como una librería de enlace dinámico (DLL), que permitiera exportar los objetos que se definieran y mantuviera ocultos los que no se requerían.

Pero debido a que las DLL no soportan herencia para exportar, es decir, si se define una clase base que sea exportable, las descendientes no heredarán esta cualidad. Este inconveniente llevó a pensar en replantear la jerarquía de clases o a una solución menos drástica. Afortunadamente, la solución estuvo al alcance y lo que se hizo fue definirle específicamente el atributo de exportable a cada clase que lo necesitara, sin modificar la jerarquía ya definida.

Posteriormente surgió la necesidad de encapsular el funcionamiento de la aplicación con el fin de dejar las tareas de registro de la clase de Windows, creación e inicialización de la ventana de la aplicación y posterior liberación de estos recursos en una o varias clases del motor. Todas estas actividades se dejaron como responsabilidad de la clase CAplicacion, de la cual heredó la clase CAplicacionD3D. Esta

última se encarga de crear e inicializar el dispositivo Direct3D a través de un objeto del tipo CDirect3D para que posteriormente todos los objetos de la aplicación puedan usarlo en las tareas de dibujado.

Un nuevo inconveniente se presentaría al intentar controlar los mensajes provenientes del sistema operativo, determinar qué objeto de los que conforman la aplicación disparó el evento y luego manejarlo adecuadamente. Se pensó entonces, en el uso de un patrón que es usado en algunos lenguajes de programación tales como Java y que consiste en asignar un escuchador para el control de los eventos de cada uno de los objetos. Para lograr implementar este patrón se crearon las interfaces CEscuchadorEventos, CEscuchadorVentana e IObjetivo. El escuchador de eventos sería implementado por las clases que iban a controlar los eventos de ratón y teclado de algunos objetos, la interfaz para escucha de una ventana se usaría para poder controlar los eventos generados los objetos que heredan de CVentana (tal como la ventana principal de la aplicación), y la interfaz IObjetivo serviría para poder implementar objetivos que responden a eventos de ratón y teclado.

Aunque las clases con las que se implementó el patrón escuchador responden bien ante el propósito deseado, su forma de uso en una aplicación puede resultar un poco confusa y difícil de entender para los desarrolladores que por primera vez entren en contacto con el Motor 3D, por lo cual se recomienda revisar la forma en la que se realiza el control de los mensajes de eventos en las aplicaciones creadas durante el desarrollo de este proyecto. De igual manera se recomienda revisar el ANEXO B, en el cual se muestra un ejemplo de cómo utilizar el Motor 3D.

Teniendo ya las bases para la creación de aplicaciones con soporte para Direct3D, se inició la tarea de implementar los objetos con características tridimensionales y en este proceso se tuvo que encontrar, en primer lugar, la forma en la cual se podía realizar la carga, manipulación y dibujado de mallas con animación incluida. Esto se convirtió en una dificultad que generó un retraso en el proyecto, puesto que es de vital importancia para un juego en 3D poder mostrar modelos con diferentes animaciones en el espacio tridimensional.

Luego de investigar la forma en la que se podía realizar la carga y dibujado de los modelos animados se llegó a la implementación de la técnica de mallas con piel (SkinMesh), la cual consiste en cargar el modelo como una jerarquía, donde existe una

parte del modelo que es la 'raíz' y a esta se le adjuntan las demás partes. Cada parte se toma como un hueso, al cual se le 'pegan' los vértices que representan la piel. Además, las animaciones se dan como transformaciones de mundo que en cada parte del modelo dependen de la parte que está arriba en la jerarquía. Así por ejemplo cuando se construye el modelo de una persona, la posición de un brazo, una pierna o la cabeza, dependerá de la posición del tronco en el momento de la animación.

Una vez que se logra cargar y pintar adecuadamente un modelo con información de animación, incluyendo animaciones diversas (por ejemplo caminar, atacar, morir), surgió la necesidad de verificar si el puntero del ratón se encontraba sobre la imagen del modelo animado proyectada en la pantalla. Para dar solución a este problema existen dos técnicas: la primera de ellas consiste en calcular una esfera que pueda contener al modelo a partir de su centro y radio, y luego verificar matemáticamente si el rayo generado por la posición del ratón en la pantalla se intersecta con la esfera, y la segunda técnica usa el mismo rayo para determinar si intersecta alguno de los triángulos que componen la malla del modelo.

De estas dos técnicas, se optó por el uso de la esfera contenedora, ya que los cálculos que se realizan para determinar si el rayo la intersecta son relativamente sencillos y no consumen recursos abundantes en cuanto a procesamiento, mientras que la otra técnica, debe verificar triángulo por triángulo si el rayo intersecta a alguno de ellos, lo cual consume una cantidad considerable de tiempo en procesamiento. Pero es preciso anotar, que aunque la técnica de la esfera funciona, no siempre es la mejor ya que brinda un margen de error considerable debido a que dentro de la esfera que contiene al modelo queda un espacio 'libre' que el modelo no ocupa. Esto produce por ejemplo, que cuando haya varios modelos juntos, no responda al evento clic el modelo deseado. Se recomienda entonces hacer una revisión de los requerimientos en cuanto a tiempo de procesamiento y precisión de los cálculos cuando se desee implementar una de estas técnicas, e incluso contemplar la posibilidad de implementar un hibrido de ambas.

El trabajo que se había realizado hasta el momento con los modelos animados era muy bueno, pero aún era necesario tener en cuenta un aspecto que es de gran impacto en el rendimiento y uso de la memoria de un juego y es que, generalmente, se necesita cargar varios modelos del mismo tipo con el fin de mostrarlos dentro de la misma escena pero en una posición diferente y con animaciones heterogéneas. Para lograr

reducir la cantidad de memoria ocupada por los diversos modelos de un mismo tipo, se separó la parte lógica del modelo, como su posición, escala y rotación, de la parte que contiene la información de los vértices del modelo y la animación. De este modo, si en un juego se deben mostrar varios personajes con el mismo modelo (por ejemplo, los ciudadanos en EmpireEarth) basta simplemente con crear un objeto del tipo CAnimado en el cual se carga la información de la malla y la animación, y crear tantos objetos CModelo como personajes se necesiten y a cada uno asignarle el animado anterior. Cada objeto CModelo crea un objeto para el control de la animación mediante el uso de la interfaz ID3DXAnimationController del DirectX.

Habiendo finalizado la implementación de las clases para los modelos, se presentó un nuevo reto y consistía en implementar una clase para el terreno. En principio se intentó manejar una malla completa para representar todo el objeto terreno, pero esto se dificultó dado que en un juego de estrategia normalmente el terreno debe ocultarse en las partes en las que no se ha explorado y con una malla esto resulta demasiado complicado de realizar. Además, el objeto para el terreno debe permitir posteriormente 'navegar' por el mismo o desplazarse por su superficie, para lo que se debe conocer la información exacta de la altura de un punto en la malla y esto implicaría hacer un recorrido por todos (o la mayoría) de sus vértices. Estas razones motivaron el uso de un terreno compuesto por losas, lo cual permitiría: ocultar las losas que el jugador no ha explorado, aplicar un factor de niebla a las losas que las unidades del jugador no pueden visualizar, obtener la altura exacta de un punto dentro del terreno con el fin de facilitar el desplazamiento sobre su superficie, y posteriormente usar las losas como unidades lógicas para el algoritmo de búsqueda de caminos asumiendo el terreno como una cuadrícula en la cual cada losa puede representar un espacio libre o un obstáculo.

Otro aspecto de gran importancia durante el proceso de implementación del terreno y las losas fue el uso de los conceptos matemáticos de 3D para generar los vectores normales adecuados para cada vértice de las losas de tal forma que al pintar el terreno no se notaran las divisiones o bordes entre losas consecutivas. Adicionalmente, se trabajo con los rayos con el fin de determinar la losa sobre la que se encuentra el puntero del ratón en un determinado momento, lo cual es fundamental para el editor de terrenos y para Independencia.

En la siguiente fase de implementación del Motor 3D se abordaría la construcción de los objetos que se presentan en dos dimensiones y que son usados en la mayoría de

juegos, para lo cual se decidió construir los más comunes y que resultan más útiles para crear interfaces de usuario, tales como botones, etiquetas, cajas de texto, cajas de chequeo e imágenes. Estos objetos se construyen como objetos en 3D, pero se muestran en el espacio 2D de la pantalla, por lo que son denominados controles e implementados heredando de la clase CControl. La implementación de esta clase estuvo ligada al inconveniente de hacer que estos objetos siempre se muestren en la pantalla de acuerdo con su posición en 2D y para superarlo se trabajó en el método para pintar el control. En este método se tomó la inversa de la transformada de vista y se fijó como la transformada de mundo, lo que en otras palabras implica que cada vez que se pinta un control se toma éste y se lleva hasta en frente de la posición desde la cual se observa la escena. De este modo los controles siempre se pintan en la misma posición de la pantalla, no importando que la posición u orientación de la vista cambien.

Con los objetos antes mencionados se abarcaban los más importantes y necesarios para la creación de juegos en cuanto a la parte gráfica usando Direct3D, pero es preciso señalar que los juegos recientes hacen uso de las características multimedia que los equipos de esta era poseen y que ayudan en la tarea de atraer y divertir al jugador. Buscando que las aplicaciones creadas haciendo uso del Motor 3D puedan fácilmente implementar este tipo de características, se creó la clase CMedio, la cual se responsabiliza de la carga y reproducción de flujos de audio y video. Para la construcción de esta clase se tomó la experiencia vivida en la creación de Defensor 3D, en la cual se buscó que el juego presentara música de fondo y sonidos de ambiente. En Defensor 3D se usó DirectMusic, pero se pudo apreciar que esta API solo permite la reproducción de archivos de sonido en los formatos WAV, MIDI y los nativos de DirectX. Por esta razón se exploró en la API DirectShow y se encontró que haciendo uso de ella se puede realizar la carga de flujos de audio y video en diversos formatos tales como MP3, WAV, MIDI, MPEG, AVI, DIVX y otros. La clase CMedio hace uso entonces, de esta API para lograr encapsular dicha funcionalidad dispersa en las interfaces que expone DirectShow.

Adicionalmente a las clases que controlan la aplicación, los eventos, la parte gráfica en dos y tres dimensiones y los flujos de audio y video, era necesario incluir en el Motor un conjunto de clases que resultaran útiles para la creación de juegos. Una de ellas es la clase CExcepcion, la cual se construyó con el propósito de que el desarrollador pueda realizar el manejo de las excepciones producidas por el motor. Estas

excepciones, regularmente se originan cuando el motor detecta que algún tipo de inicialización que intentaba realizar no fue exitosa.

Otra situación que se presenta muy frecuentemente en los juegos, es el uso de listas para mantener el estado de objetos de algún tipo. Una lista es una colección enlazada de nodos, la cual, dependiendo de la implementación, puede generar retrasos en un juego debido a que se puede necesitar recorrerla muchas veces. Para ayudar con esta situación y dependiendo de las necesidades específicas del juego, el motor brinda dos tipos de estructuras para el manejo de listas: la clase CLista representa una lista lineal enlazada y la clase CTablaClaves es una implementación a modo de árbol binario de una lista que en cada nodo asocia una clave de búsqueda con la referencia al objeto que contiene.

La solución a todos los problemas mencionados en está sección arrojó como resultado el Motor 3D orientado a objetos (para mayor detalle de las clases ver Anexo C), con el que luego se implementaría el visor de modelos, el editor de terrenos, el buscador de caminos y el juego Independencia.

3. 4. 2. VISOR 3D

Después de obtenido un primer prototipo del Motor 3D y que brindaba la funcionalidad necesaria ya expuesta se procedió a la creación del visor de modelos, que usa la librería de clases, proporcionada por dicho motor, para implementar el diseño definido para el visor.

Se identificó para entonces, que era necesario poder configurar el adaptador de video y de esta manera ofrecer una mayor versatilidad para las aplicaciones que se desarrollaran de ahí en adelante. Para la implementación de esta utilidad se empleó un cuadro de diálogo de Windows que se definió mediante un archivo de recursos y se programó en la clase CDlgPreferencias, garantizando con esto que podía ser reutilizada en las demás aplicaciones.

Un punto a resaltar de la implementación de la clase anterior fue que se empezó a utilizar archivos de configuración, donde se guardaban y podían recuperar las opciones

elegidas, con lo cual se empezó a entender las bases de la técnica de scripting, muy utilizada en la implementación de juegos.

Al continuar con la codificación del visor se descubrieron algunas falencias que se presentaban en la clase CAnimado, dentro de las que se contaba el no poder fijar una animación a voluntad ni poder detenerla, por lo que se debió realizar una extensa consulta que proveyera las respectivas soluciones, consistiendo éstas en la utilización de las interfaces ID3DXAnimationSet e ID3DXAnimationController, que ofrecen los servicios para fijar una animación y determinar cuando ha finalizado un periodo de ejecución.

Adicionalmente y gracias al conocimiento adquirido en cuanto al manejo de animaciones se logró profundizar en cuanto a la forma en como están definidos los archivos .x, es decir, se conoció la sintaxis de las plantillas empleadas para la creación de dichos archivos. Esto permitió que se pudiera saber donde encontrar la información referente a las animaciones lo cual fue de mucha utilidad porque después se necesito integrar de forma manual los diferentes archivos de animación de los modelos, ya que el diseñador gráfico (debido a los problemas de exportación del plug – in) solo podía entregar una animación por archivo.

Al finalizar el desarrollo de esta aplicación, se logró el objetivo deseado: el diseñador podía contar con una herramienta que le facilitaba probar los modelos que generaba y los desarrolladores tenían la certeza de contar con modelos que iban a funcionar perfectamente en Independencia. Una imagen de la ventana de la aplicación se muestra en la Figura 23.

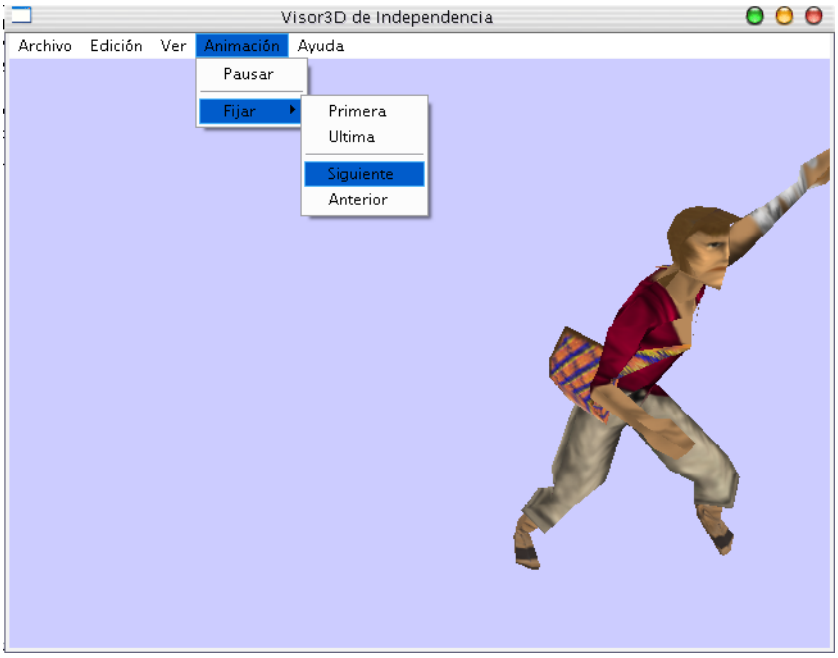

Figura 23. Imagen del Visor de modelos.

3. 4. 3. EDITOR DE TERRENO

Al iniciar la implementación del diseño del editor de terrenos se tuvo un gran tropiezo: sólo los objetos definidos a partir de la clase CObjetivo podían procesar los eventos de ratón y teclado de modo que la aplicación debía conocerlos a todos, y por tanto debía realizar muchas tareas que podía delegar. Esta dificultad llevó a la creación de la interfaz IObjetivo que define los métodos que deben implementar las clases que quieran responder a eventos de ratón y teclado y que no sean necesariamente objetos de la clase CObjetivo. Es así, como a partir de la interfaz se implementaron las clases que controlarían la lógica del menú, de los botones y del mini mapa.

Superada esta situación era momento de abordar la implementación del mini mapa que suponía una instantánea del terreno pero en miniatura y cuyo principal inconveniente radicó en poder modificar la imagen que se iba a mostrar para que reflejara lo que en realidad se observa en el terreno. Fue necesario hacer uso de una superficie (algo así como un lienzo de dibujo) en memoria del sistema para poder modificarla y luego actualizar una textura (una imagen) en memoria del dispositivo que finalmente se dibuja en la pantalla.

Se habían terminado los objetos de la interfaz de usuario del editor de terreno y se había implementado la funcionalidad deseada, obteniéndose un producto robusto que adaptó algunas características de los editores de juegos comerciales como Age of Empires y Rise of Nations. Era hora de empezar a generar terrenos. Pero se presentó un problema porque no se poseían cuadros de diálogo que permitieran pedir un texto al usuario para abrir o guardar los archivos de terreno al igual que cuadros de diálogo que mostraran mensajes.

Se inició la labor de diseñar e implementar cuadros de diálogo para las tareas mencionadas y se consideró su conveniencia dentro de la jerarquía de objetos del Motor 3D, siendo necesaria su inclusión y finalizando con esto la codificación del editor de terrenos. En la Figura 24 se muestra una imagen de la creación de un terreno, donde se detalla un cuadro de dialogo implementado para la selección de opciones para el nuevo terreno.

Figura 24. Imagen del Editor de Terrenos.

3. 4. 4. BUSCADOR DE CAMINOS

Con el fin de permitir que los personajes del juego pudieran desplazarse por el terreno, se inició la construcción de una aplicación que implementara un algoritmo de búsqueda de camino y permitiera probar los escenarios más comunes que se pudieran presentar. El algoritmo elegido fue el A estrella (A*) y a pesar que existe muy buena documentación acerca de su funcionamiento, cada desarrollador debe adecuarlo a sus necesidades.

Se emprendió el proceso de adaptación del algoritmo a los requerimientos de Independencia y se obtuvo una primera versión que al ser probada arrojó unos tiempos de respuesta demasiado grandes en determinadas situaciones, como por ejemplo cuando se intentaba ir desde una zona libre hasta otra que estuviera encerrada por obstáculos, tal como se muestra en la Figura 25. En estos casos el número de nodos visitados por el algoritmo crecía demasiado dado que una vez llegaba al lado de la frontera que encerraba el destino, empezaba a expandirse hacia los lados, pudiendo visitar toda el área por fuera de la región limitada. Después de hacer el análisis respectivo, se optó por limitar el número de nodos cerrados que el algoritmo podía crear ya que estos nodos determinan cuanto se expande la búsqueda.

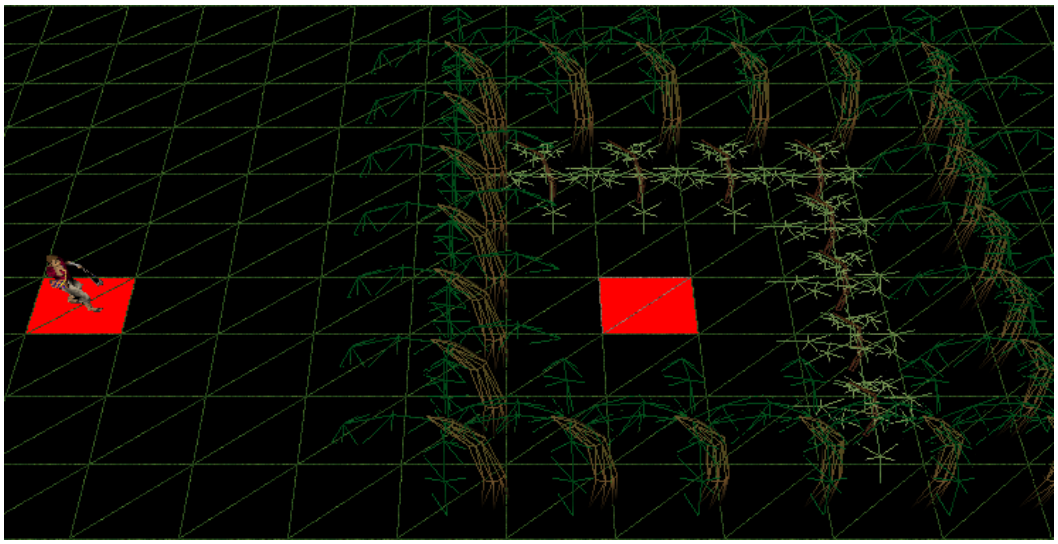

Figura 25. Ejemplo de un escenario de prueba para búsqueda de caminos

Fue de gran ayuda también para mejorar el rendimiento del algoritmo, el hecho de haber creado listas que permitieran ordenar los nodos, puesto que la lista que se definió inicialmente (CLista) no soportaba esta funcionalidad. Esta mejora implementada ayudo a que no se tuviera que recorrer toda la lista de nodos abiertos cada vez que deseaba encontrar el de menor costo.

Ya obtenido el algoritmo de búsqueda deseado y estando nuevamente en una fase de pruebas en diferentes escenarios, se notó que los personajes del juego podían pasar por en medio de los modelos de ambientación cuando estos ocupaban más de una losa. La solución a este problema suponía el replanteamiento del diseño del terreno manejado con losas modificándolo de tal manera que hubiera sido necesario cambiar toda la lógica trabajada hasta el momento. Esto no era conveniente en este momento para los intereses del proyecto y había que buscar otra solución menos drástica. Se pensó entonces en el uso de modelos de ambientación planos que estuvieran ubicados por debajo de los modelos más grandes.

Esta última solución no sólo fue creativa, sino que sirvió para añadir otra característica especial al juego como la de permitir que los personajes dieran la sensación de poder caminar por los andenes e incluso se podría pensar en la posibilidad de que ingresaran a las edificaciones.

3. 4. 5. INDEPENDENCIA

En el momento de abordar la codificación de Independencia, se contaba con un cúmulo de experiencias y conocimiento fruto de los anteriores desarrollos, de tal modo que se sabía como manipular modelos con animaciones, trabajar con terrenos, usar el algoritmo para búsqueda de caminos, e incluso la manera de implementar las diferentes pantallas del juego. Solo hacia falta unir todos los componentes y dar forma a Independencia.

De acuerdo al diseño definido, se debía empezar la creación del tipo de agente AgenteUnidad para que llevara a cabo las labores propias de los personajes dentro del juego, que incluían vigilar, caminar, atacar, recibir ataque, emitir y escuchar el grito comunero y morir. Estas acciones fueron mapeadas a métodos, los cuales eran

llamados desde el procedimiento de ejecución del agente de acuerdo con la actividad que se encontraba realizando la unidad en ese momento.

El implementar el método para caminar y atacar supuso una mejora en cómo la unidad debía moverse, porque hasta ese momento no se había considerado que si se elegía ir a una losa que ya estuviera ocupada, la unidad debía desplazarse hasta el punto más cercano posible. Fue necesario entonces, realizar un cambio al algoritmo de búsqueda de caminos de tal forma que si la losa de destino se encuentra ocupada se prueba con las losas adyacentes pero siempre teniendo en cuenta en buscar por el lado más cercano posible.

Pero para que el jugador pueda ordenar a sus unidades que realicen una actividad como caminar o atacar, es necesario antes seleccionarlas y esto debe reflejarse visualmente. Para lograr esto se tomó como referencia la forma como lo realizan juegos de similar clasificación y es mediante el dibujado de una barra sobre la unidad. Se exploraron diferentes alternativas de solución como son:

- Dibujar una línea recta sobre el modelo en el espacio 3D. Esta solución presentó el inconveniente de que la línea se mostraba en perspectiva y por lo tanto se deformaba.
- Dibujar la misma línea pero en el plano más cercano de la cámara, lo cual no gustó porque visualmente no era atractivo.
- Hacer uso de una malla para pintar un cilindro encima del modelo. Esta fue la opción seleccionada debido a que es visualmente muy llamativa y por ser un objeto 3D no presenta el problema de la perspectiva, es decir no se deforma.

Con esto se obtenía la funcionalidad y la apariencia gráfica de las unidades, sin embargo, se detectó que gráficamente el juego en general era muy plano, en el sentido de que a pesar de ser tridimensional, el terreno y la ambientación se veían un poco bidimensionales. Se realizó entonces una revisión de juegos de estrategia en la que se vio que juegos como Age of Empires y Rise of Nations usan una vista isométrica, es decir rotan el terreno o la cámara en un ángulo determinado, que por lo general es de 45º. Cabe resaltar que Age of Empires a pesar de ser un juego en 2D, gracias a este cambio de perspectiva adquiere el aspecto de ser un juego en 3D. Esto motivó a que en Independencia se trabajara con una vista isométrica para lo cual se realizó una rotación de 45º a la cámara logrando con esto mejorar su apariencia.

Junto con la rotación de la cámara era necesario rotar también el mini mapa pues se debía mantener la concordancia con el terreno. Aunque el trabajo de rotar la cámara resultó sencillo, no ocurrió igual con el mini mapa, dado que éste se implementó con objetos de tipo control (CImagen), los cuales no permitían mostrarse con algún tipo de rotación. Este inconveniente fue resuelto modificando la clase CControl del Motor 3D, para lo cual se hizo uso de los conceptos matemáticos de rotación de ejes [25]. De igual manera se requirió de los mismos conceptos de rotación de ejes para poder hacer selección múltiple de unidades utilizando el ratón mediante el método de mover el puntero mientras se tiene presionado el botón izquierdo.

Hasta ese momento ya se habían implementado muchos de los requerimientos del juego, pero faltaban varios detalles que complementan y enriquecen la interfaz de usuario y la funcionalidad de Independencia, tales como animación del grito comunero, zoom sobre el terreno, despliegue de botones de funciones para cada unidad y la musicalización y efectos de sonido. En cada uno de ellos se precisó de un trabajo específico y aunque los requerimientos eran pequeños, no por eso debían ser menospreciados, pues proporcionan el realce audiovisual necesario para lograr la motivación del jugador.

Un ejemplo de cómo se fue mejorando la apariencia del juego fue la animación del grito comunero, que es el poder especial que puede usar el prócer y por tanto debería destacarse gráficamente (como se muestra en la Figura 26). Por sugerencia del diseñador gráfico se optó por el uso de una secuencia de imágenes para crear una animación, razón por la cual se incluyó dentro del motor la clase CSecuencia, que cumplía con el fin propuesto.

Figura 26. Animación del grito comunero en Independencia.

Finalmente se pudo obtener el juego deseado, Independencia contaba con las características de un juego de estrategia en modo campaña, pero además:

- Permite hacer acercamientos de primer plano que facilitan la visualización de las acciones.
- Hace uso de técnicas de animación avanzadas como los modelos 3D texturizados y/o animados y las secuencias de texturas con canal alfa.
- Utiliza transparencias para las edificaciones y diferentes factores de niebla en las áreas del terreno.
- Emplea música de fondo para la partida y efectos de sonido para las acciones en el juego.

4. CAPÍTULO IV: CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

El último capítulo de este documento tiene por finalidad presentar las conclusiones a las que se llegó una vez terminado este trabajo de investigación, así como también plantear una serie de recomendaciones encaminadas a ayudar los futuros trabajos que se realicen a partir de los resultados de este proyecto o que estén relacionados con el área del desarrollo de video juegos. Finalmente, se presentan las actividades futuras a llevar a cabo por parte de los investigadores y que buscan dar continuidad a la labor iniciada con la presente investigación.

4. 1. CONCLUSIONES

Cuando se optó por llevar a cabo un proyecto que tendría que ver con la implementación de un juego de estrategia en 3D, fueron muchos los escépticos que pensaron que era una labor muy complicada e incluso difícil de lograr, debido en gran parte a que proyectos de este tipo no son frecuentes en nuestro medio y por tanto no se cuenta con la experiencia suficiente para desarrollos así.

Se logró pues implementar, no sólo un juego de estrategia en 3D y varios productos más, sino que se ganó experiencia y conocimiento en la creación de esta clase de proyectos en los cuales hasta ahora no se había incursionado, por lo menos, en el ámbito regional. En este sentido es de destacar que, a pesar de haber empezado casi desde cero, se logro un dominio deseable no sólo de la tecnología utilizada, sino del proceso de desarrollo en sí, en el cual se empezó por una fase exploratoria, para después ir construyendo poco a poco el producto final mediante acercamientos a la solución de problemas que tenían que ver con el juego.

Fue de gran ayuda para iniciar este proceso de construcción del conocimiento el estudio y apropiación de la tecnología ofrecida por Microsoft DirectX, pero además fue un gran aliciente la creación de Defensor 3D, porque no solamente brindó los primeros conceptos en cuanto a la creación de juegos usando Direct3D, sino que sirvió como motivación para el equipo de desarrollo, pues por ser un proyecto pequeño, los resultados se obtuvieron en un corto plazo y la aplicación de lo estudiado empezó a rendir los frutos deseados.

Sirvió esta primera aproximación, también, para empezar a dar forma al Motor 3D, ya que se inició con la determinación de las clases que serían necesarias para llevar a cabo los desarrollos subsiguientes. Pero fue mediante la definición de la jerarquía de objetos y su posterior implementación que se logró cumplir con uno de los objetivos propuestos: crear y usar un Motor 3D para el dibujado, musicalización y manejo de dispositivos de entrada y salida.

No obstante, aunque el motor tenía un propósito claro que era ayudar en la construcción del juego, su utilidad se extendió más allá, hasta el punto de servir como soporte para otras aplicaciones como el visor de modelos, el editor de terrenos y el buscador de caminos.

Se emprendió pues un proceso segmentado, en cuanto que se abordó el proyecto dividiéndolo en proyectos más pequeños, pero con objetivos claros, porque el objetivo central seguía siendo el mismo: crear Independencia. En este sentido es de destacar que cada aplicación 'intermedia' tuvo un propósito definido, siendo el del visor de modelos poder servir de puente de comunicación entre el diseñador gráfico y el equipo de desarrollo, el del editor de terrenos facilitar la creación y edición de terrenos para ser usados en Independencia y el del buscador de caminos facilitar la realización de pruebas y permitir la optimización del algoritmo A estrella.

Hoy por hoy es claro que el trabajo realizado con el diseñador gráfico se aprovechó y simplificó más gracias a la creación del visor de modelos, pero el valor de esta aplicación se extendió aún más, debido a que además se implementó una utilidad para la configuración del adaptador de video, con lo cual se permitió que el motor se pudiera instalar en diferentes máquinas y determinar si era capaz de ejecutarse en cada una de ellas o no.

Respeto al editor de terrenos es de destacar que, al igual que el visor de modelos, sirvió para trabajar en conjunto con el diseñador gráfico, con el fin de definir el tamaño de los modelos de ambientación, las imágenes de las losas y el terreno en general, demostrando con esto que cuando se realiza trabajo interdisciplinario es necesario definir canales adecuados de comunicación, siendo en este caso unas aplicaciones que facilitaran la labor de empalme de información necesaria tanto para el diseño gráfico, como para la implementación del juego. Es de resaltar que la experiencia adquirida mediante este trabajo interdisciplinario fortaleció el proceso de desarrollo, pues gracias a ello se aprendió a trabajar con otras personas que tienen un dominio en un área de aplicación diferente y resultó interesante poder llegar a un punto de entendimiento tal entre las partes de manera que si una de ellas hacía una propuesta en su "propio lenguaje", la otra captaba la esencia del mensaje.

Referente al buscador de caminos es de destacar que su creación fue vital para el mejoramiento del algoritmo A estrella, porque aunque la teoría acerca de la utilización de dicho algoritmo es muy difundida, la implementación es difícil de conseguir, por lo que las mejoras hay que encontrarlas sobre la marcha y esto se logra probando diversos escenarios posibles. Se logró de esta manera conseguir un algoritmo eficiente y robusto que posteriormente fue utilizado por las unidades modeladas en Independencia para realizar los desplazamientos dentro del terreno.

Ya entrando en detalle en cuanto a la creación de Independencia en sí, se debe resaltar que la metodología utilizada sirvió en pro de la solución buscada. Se nota en primera instancia, que siguiendo los consejos de gente con experiencia en el diseño de juegos, el punto de partida de un juego debe ser el guión, pues este servirá como eje conductor del mismo y será el que al final determine si se cumplieron las metas respecto a lo que se esperaba en un principio. Después se debe seguir un proceso de desarrollo al igual que se haría para cualquier otro proyecto de software, pero se resalta en Independencia la utilización de otras metodologías como GAIA, que facilitaron su creación.

En relación a este último aporte se debe considerar que GAIA ayudó a definir los agentes que se utilizarían dentro del juego sin necesidad de entrar en detalles de implementación y que los productos obtenidos gracias a esta metodología se pudieron diseñar muy fácilmente usando UML. Existió, entonces, una complementariedad entre ambos permitiendo con esto que el trabajo para la definición e implementación del

comportamiento de las unidades del juego estuviera a la altura y vanguardia de grandes títulos de juegos en esta misma categoría.

Pero no sólo se destacó la implementación de dicho comportamiento en Independencia, sino que el juego como tal aportó muchas características que no estaban definidas desde el principio y que fortalecieron la apariencia visual y auditiva del mismo. Dentro de estas características se cuenta con poder hacer zoom sobre las unidades, animaciones, uso de transparencia y niebla y musicalización, siendo el soporte para todo esto la tecnología de DirectX.

DirectX ofrece un gran y completo conjunto de librerías que permiten la creación de proyectos multimedia de diversa índole, pero además brinda la posibilidad de que cada desarrollador utilice las características que necesita de la API. De las diversas API's que conforman a DirectX, se destaca la versatilidad de DirectShow, que tan solo a partir de una clase (CMedio) dio soporte para audio y video de una manera transparente para el desarrollador. Las otras API's como DirectGraphics, DirectPlay, DirectInput, DirectMusic, DirectSound, DirectSetup y DirectX Media Objects, se destacan por su funcionalidad y cobertura de las necesidades para la creación de juegos de excelente calidad.

Al crear Independencia se aprovecharon las ventajas tecnológicas de DirectX, pero también se benefició de la historia patria, buscando con esto rescatar la tradición y darle a esta tecnología la posibilidad de servir como instrumento motivador para el redescubrimiento y reconocimiento de esa historia olvidada en los libros de educación media.

Realizado todo el plan de creación de Independencia, se puede afirmar que es posible llevar a cabo en nuestro medio un proyecto relacionado con la creación de juegos, no obstante aún no existan todos los espacios que favorezcan el desarrollo de proyectos de este tipo. Es así como a la luz de los hechos se puede ver fácilmente que la apropiación de la tecnología no es una gran limitante, ni tampoco la escasez de recursos o el tiempo empleado y que lo que hace falta realmente es un poco de apoyo para poder sacar a flote estas ideas salidas de lo convencional. Se han dado, entonces, los primeros pasos en el programa de Ingeniería de Sistemas de la Universidad del Cauca, para que se considere ofrecer nuevos espacios para la creación de proyectos fuera de lo convencional, en cuanto que no se debe brindar únicamente la posibilidad

de incursionar en proyectos que se puedan denominar tradicionales (como Sistemas de información para gestión de datos o aplicaciones para la Web), sino que se requiere un esfuerzo para proponer nuevas alternativas que permitan que los egresados puedan desempeñarse en un mercado que hoy por hoy está falto y ávido de ingenieros motivados y con ganas de emprender proyectos diferentes.

4. 2. RECOMENDACIONES

A los investigadores y personas interesadas en dar continuidad al trabajo iniciado con este proyecto se les recomienda en primer lugar, realizar una optimización de los algoritmos de rendering de los diversos objetos del Motor 3D para luego continuar con la implementación de características que aún no están presentes en el juego como jugar en modo aleatorio y modo multi jugador. Además sería muy enriquecedor el iniciar el desarrollo de otro tipo de juegos diferentes a los de estrategia, tal como los juegos de habilidad y acción.

Por todo el trabajo que involucró la construcción de Independencia y por todos los conocimientos que se tuvieron que aplicar, se puede afirmar que en el proceso de desarrollo de un juego se deben complementar muchos de los conceptos tratados en las diferentes asignaturas del programa de Ingeniería de Sistemas de la Universidad del Cauca. Por tal motivo se recomienda integrar las actividades realizadas en asignaturas complementarias, como es el caso de la electiva de desarrollo de software orientado a agentes y las asignaturas en las que se tratan tópicos relacionados con matemáticas en 3D, ingeniería de software, teoría de la computación y otros.

Sería acertado, también que en un futuro los estudiantes del programa de ingeniería de sistemas puedan tener la posibilidad de contar con un pensum más flexible en el que se brinden más opciones en cuanto a las diferentes áreas en la que se pueden desenvolver y aplicar sus conocimientos, habilidades y fortalezas, una vez que terminen sus estudios de pregrado.

De igual manera se recomienda que para proyectos de este tipo se pueda contar, no solo con el apoyo de diseñadores gráficos, sino con muchas más personas en diversas

áreas, como por ejemplo literatos e historiadores que den forma a la historia y en general al guión del juego y matemáticos que ayuden en la tarea de reforzar los conceptos necesarios para un mejor trabajo con objetos en tres dimensiones.

Así mismo se recomienda la creación de una electiva al interior del programa de Ingeniería de Sistemas de la Universidad del Cauca en la que se ofrezca la posibilidad de conocer y desarrollar proyectos en el área de creación de juegos y que además se realice un trabajo interdisciplinario con docentes y estudiantes del programa de Diseño Gráfico, de manera que se puedan complementar las actividades propias del diseño de juegos.

4. 3. TRABAJO FUTURO

La labor de la creación del juego no debe quedar sólo con lo hecho hasta ahora en Independencia, sino que se debe complementar el juego de manera tal que se brinde mayor funcionalidad y más características que no se pudieron llevar a cabo en este proyecto debido a limitantes de tiempo y recursos. Para lograr este objetivo, será necesario buscar instituciones que apoyen la investigación y el trabajo innovador en nuestro medio, tal y como puede ser el caso de COLCIENCIAS.

Con esto se podría avanzar mucho más en cuanto al proceso de desarrollo de juegos y se podría explorar temas que no fueron incluidos dentro de Independencia, tal y como es el caso del framework de efectos o la utilización de shaders en Direct3D. Se lograría una investigación y aprehensión de conocimientos que redundaría en beneficios personales y en mejora del producto con posibilidades reales de comercialización.

BIBLIOGRAFÍA

[1] Revista ENTER. Edición Nº 57: Información tomada de http://www.theesa.com/pressroom.html. Mayo 2003.

[2] PENQUÉ, EDGAR. Implementación de un sistema de visión artificial para la clasificación y el control de calidad de frutas. Trabajo de grado para optar al título de Ingeniero Físico Universidad del Cauca, 2003.

[3] GARRETA, LUIS. Presentación Electiva de Procesamiento Digital de Imágenes. Universidad del Cauca, Departamento de Sistemas. 2003.

[4] HAWKINS, KEVIN. Game Dictionary. http://www.gamedev.net/dict/

[5] SAWYER BENJAMIN. The Getting Started Guide to Game Development FAQ: What Types of Games Do Well?. http://www.gamedev.net/reference/articles/article261.asp. 1995

[6] GÓMEZ CLAROS, RODRIGO. Tres historias para la creación y la animación como profesión. Video Conferencia dictada en el marco de Imagen Invisible 2002 en la Cámara de Comercio de Cali, sede principal. 2002. http://www.imageninvisible.com/espa%F1ol/3d.html

[7] TOONKA Films. K-TOON 2D Animation Toolkit. http://ktoon.toonka.com/.

[8] ALAMO, JUAN PABLO y SÁNCHEZ, JAIME. La tribu. Software presentado en TISE 99. Departamento de Ciencias de la Computación, Universidad de Chile; 1999.

[9] HOWLAND, GEOFF. Game Design: The Essence of Computer Games. http://www.gamedev.net

[10] BELDING, TONY. Elements Of VideoGame Style. http://www.gamedev.net

[11] HNO. JUSTO RAMÓN. Historia de Colombia, significado de la obra Colonial, Independencia y Republica. Bogota: Librería Stella, 1965.

[12] LUNA, FRANK D. Introduction to 3D game programming with DirectX 9.0. Wordware Publishing Inc.: 2003

[13] CRAWFORD, CHRIS. The art of computer game design. http://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html. 1982

[14] BARRON, TODD. Strategy game programming with DirectX 9.0. Wordware Publishing Inc.: 2003

[15] LARAMEE, FRANCOIS DOMINIC. The game design process. http://www.gamedev.net/reference/list.asp?categoryid=23#32. 1999

[16] COOK, DANIEL. Evolutionary Design: A practical process for creating great game designs. http://www.lostgarden.com. 2002

[17] SWEETSER, PENELOPE. Current AI in Games: A review. School of ITEE, University of Queensland. AUSTRALIA http://www.itee.uq.edu.au/~penny/Game%20AI%20Review.pdf

[18] WOOLDRIDGE, MICHAEL, JENNINGS, NICHOLAS and KINNY, DAVID. The Gaia Methodology for Agent-Oriented Analysis and Design. Kluwer Academic Publishers, Boston.

[19] BUDD, TIMOTHY. Introducción a la programación orientada a objetos. Addison Wesley Iberoamericana: Wilmington, Delaware (USA), 1994.

[20] GRANADOS, RAFAEL. Historia de Colombia, la Independencia y la República. Bogotá: Bibliográfica Colombiana Ltda., 1964.

[21] MORRIS, DAVE y ROLLING ANDREW. Game architecture and design: Chapter 8, the future of game design. 1999.

[22] PATEL, AMIT J. Amit's Game Programming Site. http://www-cs-students.stanford.edu/~amitp/gameprog.html. 2003.

[23] SNOOK, GREGORY. Real-Time 3D Terrain Engines Using C++ and DirectX 9 (Game Development Series).

[24] TAYLOR, GREG. Tile-Based Games FAQ version 1.2. x2ftp.oulu.fi: pub/msdos/programming/docs. 1995

[25] PURCELL, EDWIN y VARBERG, DALE. Cálculo con geometría analítica. Prentice Hall. México. 1993.