Universidad de las Ciencias Informáticas

Facultad 5 "Entornos Virtuales"

Herramientas de Edición y Análisis Topográfico de Estructuras de Navegación.

Trabajo de Diploma para optar por el título de Ingeniero de Ciencias Informáticas.

Autores: Andrés Miguelevich Iparraguirre Babiy

Miguel Castro- Palomino Ruíz

Tutor: Ing. Frank Puig Placeres.

Julio

2007

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Andrés Miguelevich Iparraguirre	Miguel Castro-Palomino Ruíz	Frank Puig Placeres
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

Agradecimientos

A todos los que me apoyaron durante estos años, me guiaron y me enseñaron todo lo que aprendí y sé. A mi voluntad y mis deseos de vivir y seguir subiendo la cima de la vida.

Andrés

Mi agradecimiento a todos los que de una manera u otra me ayudaron no solo durante el desarrollo de este trabajo, sino también a lo largo de estos cinco años en la Universidad de las Ciencias Informáticas, y en especial a nuestro tutor, Frank Puig Placeres.

Miguel

Dedicatoria

A mi familia, amigos y compañeros.

Andrés

A toda mi familia, que siempre me apoyó y ayudó a llegar a donde estoy hoy. A mi madre y a mi hermano, por su incondicionalidad. A Lida, por estar ahí siempre que la necesito. A Erick, mi otro hermano. A Luis, que parece que no pero sí. Y en especial quiero dedicar este trabajo a mi abuela, Dra. Zenia Navarro Díaz, que siempre será el modelo a seguir para mí además de mi madre.

Miguel

Resumen

La Inteligencia Artificial desde su nacimiento hasta su desarrollo actual ha demostrado ser una ciencia viable en el desarrollo de la Realidad Virtual. Mediante ella las computadoras han podido mostrar entornos donde la interacción Robot – Humano alcanza un buen nivel de realismo.

La aparición de algoritmos claves para determinar características del terreno así como realizar un análisis del mismo, mediante el desarrollo de los modelos cognitivos y los grafos de búsqueda. Además de poder realizar mejor la búsqueda de caminos mediante algoritmos heurísticos.

La inexistencia de una herramienta en el proyecto de Paseos Virtuales que permitiera la creación de los grafos de búsqueda para su posterior utilización en juegos y entornos, determinó la necesidad del desarrollo de la misma.

Este trabajo propone y desarrolla una aplicación que permite crear, modificar y optimizar los grafos de búsqueda para la interacción en entornos virtuales, como juegos, simuladores u otros software de la rama de la Realidad Virtual.

PALABRAS CLAVES: Modelos Cognitivos, grafos de búsqueda.

Índice

INTRODUCCIÓN **2**

CAPÍTULO 1.FUNDAMENTACIÓN TEÓRICA **6**

1.1 MODELOS COGNITIVOS 6
1.2 EJEMPLOS DE MODELOS COGNITIVOS: 6
1.2.1 Rejilla regular: *7*
1.2.2 Mallas de navegación: *9*
1.2.3 Puntos de Visibilidad (POV) *10*
1.3 ANÁLISIS TOPOGRÁFICO DEL TERRENO 14
1.3.1 Ambush *15*
1.3.2 Stronghold *16*
1.3.3 Hide *17*
1.3.4 Sniper *17*
1.3.5 Cover *18*
1.4 METODOLOGÍAS Y HERRAMIENTAS DE DESARROLLO 19
1.4.1 Metodología *19*
1.4.2 Herramientas de desarrollo *20*
1.4.2.1 Visual Studio 2005 20
1.4.2.2 Enterprise Architect 20
1.5 LENGUAJES DE PROGRAMACIÓN 21
1.5.1 C++ *21*
1.5.2 UML *21*

CAPÍTULO 2.SOLUCIÓN PROPUESTA **23**

2.1 CICLO PROPUESTO 23
2.2 REQUERIMIENTOS DE ENTRADA 24
2.3 FUNCIONALIDADES 25
2.3.1 Procesadores *25*
2.3.1.1 Algoritmos de Procesamiento: 25

CAPÍTULO3. SISTEMA, PRODUCTO Y FUNCIONES DEL SISTEMA. **28**

3.1 REQUERIMIENTOS: 28
3.1.1 Requerimientos funcionales: *28*
3.1.2 Requerimientos no funcionales: *30*
3.2 CASOS DE USO: 31
Modelado de Casos de Uso: *32*
3.3 INTERFAZ DE USUARIO: 41
3.3.1 Interfaz cargar grafo: *41*
3.3.2 Interfaz cargar MESH: *43*
3.3.3 Interfaz aplicar procesador Flood Fill. *44*
3.3.4 Interfaz aplicar procesador Simplify *46*
3.3.5 Interfaz aplicar algoritmo FixConnections *47*
3.3.6 Interfaz exportar grafo *49*
3.4 MODELO DEL DOMINIO: 50
3.5 ARQUITECTURA DEL SISTEMA: 50
3.6 FUNCIONAMIENTO DEL FRAMEWORK: 51
3.6.1 Petición inicializar. *51*
3.6.2 Petición Cargar modelo: *52*

3.6.3 Petición Cargar grafo: *53*
3.6.4 Petición Procesar: *53*
3.6.5 Petición Exportar: *54*

CAPÍTULO 4.DISEÑO E IMPLEMENTACIÓN **55**

4.1 DISEÑO POR PAQUETES 55
4.1.1 Paquete: Manager *55*
4.1.2 Paquete: Storage *56*
4.1.3 Paquete: Procesadores *57*
4.2 DESCRIPCIÓN DE LAS CLASES: 58
4.3 DIAGRAMAS DE CLASES GENERAL 62
4.3.1 Conexiones entre paquetes: *62*
4.3.2 Diagrama de clases general: *64*
4.4 DISEÑO POR CASOS DE USO 64
4.4.1 CU Cargar grafo de .ASCII *65*
4.4.2 CU Cargar MESH .X *66*
4.4.3 CU Aplicar procesador Simplify *67*
4.4.4 CU Aplicar procesador FloodFill *68*
4.4.5 CU Aplicar procesador FixConnections *69*
4.4.6 CU Exportar grafo en .ASCII *70*
4.4.7 CU Exportar grafo en .ASE *71*
4.5 DIAGRAMAS DE COMPONENTES: 71

CAPÍTULO 5.RESULTADOS **72**

5.1 PRUEBAS A MODELOS DE MUNDOS: 72
5.2 PRUEBAS DE LA APLICACIÓN DE LOS PROCESADORES: 73
5.2.1 Flood Fill: *73*
5.2.2 Simplify: *76*
5.2.3 FixConnections: *78*
5.3 RESULTADOS DE LAS PRUEBAS: 80

CONCLUSIONES **84**

RECOMENDACIONES **85**

ANEXOS **88**

GLOSARIO DE TÉRMINOS: **91**

Introducción

Cuando el hombre empezó a comprender los números, empezó a dominar la naturaleza. La matemática se convirtió en poco tiempo en la rama fundamental y en el pilar de la ciencia y de la investigación, y cada nuevo desarrollo técnico contribuía a un pequeño aporte o grano a esta ciencia.

En 1956, cuando en la famosa conferencia de Dartmouth, se utilizó el término de Inteligencia Artificial (IA), nacía de forma oficial esta ciencia, cuya principal base se encontraría en la matemática y en la biología. Con esta rama nacía la posibilidad de que se crearan máquinas que simularan acciones humanas ante diferentes situaciones que se le plantearan.

Pocos años después, al aparecer el primer programa interactivo visual, se comenzó a pensar en como lograr que la computadora creara o simulara una inteligencia artificial lo suficientemente real para poder interactuar con el usuario y lograr con esta interacción que se sintiera la fuerza del programa o que pareciera de veras que la computadora estaba pensando sus acciones. Así nacía la Inteligencia Artificial para computadores.

En años posteriores, aparecen los primeros juegos y después los primeros videojuegos y simuladores. Estos últimos requerían lograr que el software interactuara con el usuario de una forma más realista.

Para alcanzar esta interacción, la inteligencia artificial pasó a tomar un papel importante. Esto se debía a que con ella se podían simular comportamientos de la interacción usuario–máquina tan normales como cualquier acción que se realiza en el mundo real por un ser vivo.

En el mundo actualmente se crean y aparecen nuevos algoritmos de simulación, optimización, procesamiento y cálculo de datos, que se aplican en cada nuevo entorno virtual que sale a luz. Existen aplicaciones de un alto nivel profesional que integran algoritmos para realizar algunos cálculos específicos como caminos más cortos, cálculo de datos topológicos determinados, y el trabajo de grafos para el uso de puntos de visibilidad.

Algunos de estos software son generales, usando una serie de algoritmos básicos, pero otros son más específicos y usan solo los algoritmos que necesitan para llegar a

lo que se proponen. Unos son visuales, otros no. Todas estas aplicaciones sin embargo ahorran mucho tiempo de trabajo y de cálculo de datos inteligentes que se usarían por el computador para simular la inteligencia artificial.

En Cuba, el desarrollo de videojuegos y simuladores, se ha basado más en el desarrollo de simuladores para fines de entrenamiento militar y para el desarrollo de los miembros de las FAR, aunque también se ha pasado a crear simuladores para aprender la conducción de autos con fines civiles.

Sin embargo, los videojuegos sí son un campo reciente y por tanto, conllevan una simulación inteligente mucho más real y necesitan algoritmos más fuertes y de un nivel superior, pero sucede que las herramientas que existen actualmente en el mundo son de muy difícil acceso debido a las licencias y a lo alto de su costo.

Además las aplicaciones con que se puede contar a veces no cumplen con los requerimientos necesarios, ya sea porque fueron creadas para un entorno virtual predeterminado, porque los algoritmos que usan no son los que se desean o porque el resultado obtenido no es el deseado.

Para el desarrollo de la inteligencia artificial en simulaciones virtuales se requiere la creación de modelos cognitivos que permitan a la computadora entender el entorno en que se encuentra. Actualmente en el proyecto Paseos Virtuales no existe ninguna herramienta que permita aplicar algoritmos de procesamiento y cálculo de datos en grafos.

Por lo tanto el **problema científico** es:

¿Cómo implementar una herramienta que permita cargar un grafo de búsqueda realizando el análisis topológico y minería de datos del mismo?

Objeto de Estudio:

Los modelos cognitivos para la representación de entornos virtuales.

Objetivo General de la Investigación:

Desarrollar una herramienta que permita procesar los grafos de búsqueda para un mundo virtual.

Campo de Acción:

Proceso de análisis de grafos de búsqueda para la representación de entornos virtuales en simuladores y videojuegos.

Tareas de la Investigación:

- Recopilación de información sobre los distintos algoritmos de procesamiento y cálculo de datos.
- Análisis de diferentes algoritmos existentes para el procesamiento y cálculo de datos en el grafo.
- Implementación de una herramienta que permita el procesamiento y el cálculo de datos de un grafo.

Organización del documento:

El documento se encuentra organizado del siguiente modo:

El capítulo 1. Fundamentación teórica: realiza una introducción a los conceptos básicos del análisis topológico. Además se realiza una explicación de los algoritmos de procesamiento y cálculo usados por la Inteligencia Artificial, y las herramientas y metodologías usadas.

El capítulo 2. Solución propuesta: muestra como la herramienta da solución al problema planteado. Se analiza el ciclo de trabajo, los requerimientos y funcionalidades de la aplicación.

El capítulo 3. Sistema, producto y funciones del sistema: analiza el funcionamiento externo del software, a través de los requerimientos, el modelo del dominio, la interfaz de usuario, los casos de uso y la arquitectura.

El capítulo 4. Diseño e implementación: muestra el funcionamiento interno de la herramienta, a través del diagrama de clases general, la interacción entre las clases, la explicación de los métodos y diagramas de secuencia de diseño.

El capítulo 5. Resultados: se presentan los resultados que se obtuvieron con la aplicación de los algoritmos de Flood Fill, Simplify y FixConnections para la creación,

modificación, optimización, y procesamiento de los grafos de búsqueda con la herramienta propuesta.

Capítulo 1.Fundamentación Teórica

1.1 Modelos Cognitivos

En el mundo real las personas tienen necesidad de recordar diferentes aspectos del mundo donde viven para desplazarse por este. Cada parte del entorno presenta sus propias características y brinda su propia información. Esta es analizada para llegar a un lugar específico.

Para el caso de agentes de la inteligencia artificial (BOTS), se necesita de un modelo cognitivo para visualizar el mundo y poder tener una representación de cómo está estructurado el mismo para poder moverse dentro de él. Sin el modelo cognitivo, el BOT no podría tener claros varios aspectos de la representación del entorno virtual.

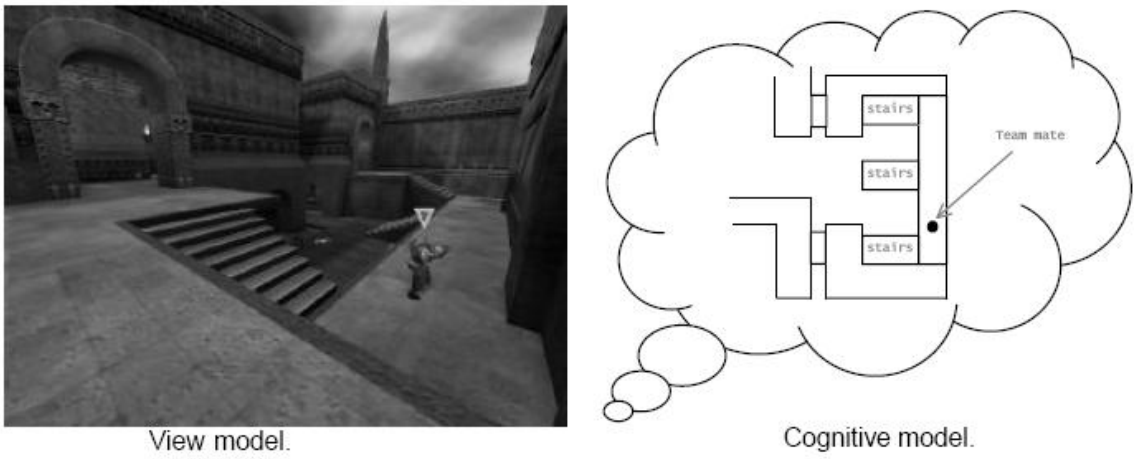


Figura 1. Representación del Modelo Cognitivo.

Usualmente el modelo cognitivo se usa para que el BOT tenga una versión simplificada del mundo virtual. No siempre un modelo cognitivo es el más óptimo a usarse en un entorno virtual, a veces es necesario realizar la representación usando varios de los modelos disponibles para lograr una representación adecuada para el BOT.

1.2 Ejemplos de Modelos Cognitivos:

Como se explicó anteriormente el modelo cognitivo se usa en la Inteligencia Artificial (IA) para representar el mundo virtual mediante la computadora. En este modelo se incluyen solamente los detalles necesarios como los obstáculos geométricos en el

entorno (puertas, ventanas, enemigos). Además almacena el estado de la información como las puertas cerradas, caminos bloqueados o el tipo de enemigo que se está moviendo.

En la práctica, existen muchos tipos de modelos cognitivos que se usan frecuentemente dándose a continuación ejemplos de algunos de los más usados:

1.2.1 Rejilla regular:

La rejilla regular se usa principalmente en la creación de entornos para juegos RTS, o juegos de estrategia en tiempo real, tales como Starcraft, Warcraft, la Era de los Imperios, entre otros.

Además la rejilla presenta un ambiente complejo basado en cuadrados o polígonos (triángulos, pentágonos o hexágonos). Esto por consiguiente es racional en el diseño de grafos de navegación aproximadamente en estas celdas. Una matriz que se puede construir para que provea información acerca de cada celda puede ser como la siguiente figura, donde cada celda es transitable o puede ser una pared.

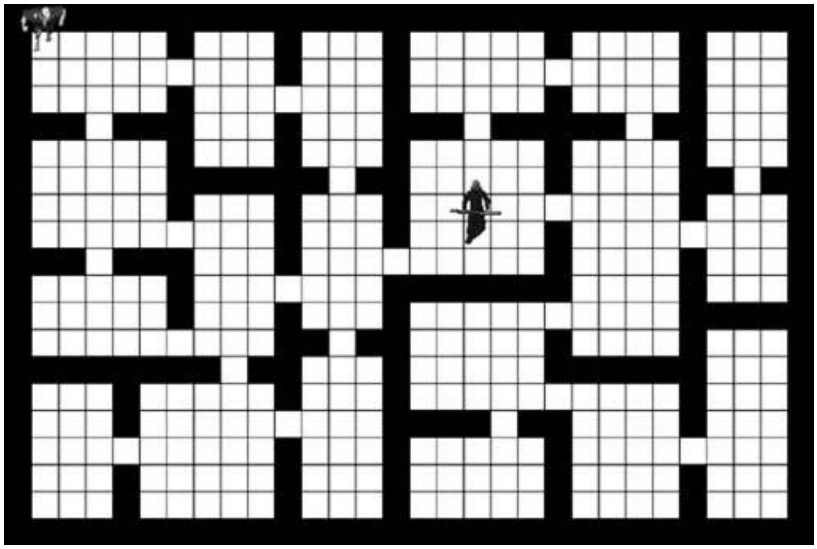


Figura 2. Representación de una rejilla regular.

La matriz además puede proporcionar información del tipo de terreno en cada celda, la siguiente figura muestra un modelo cognitivo de un mundo virtual de un juego RTS.

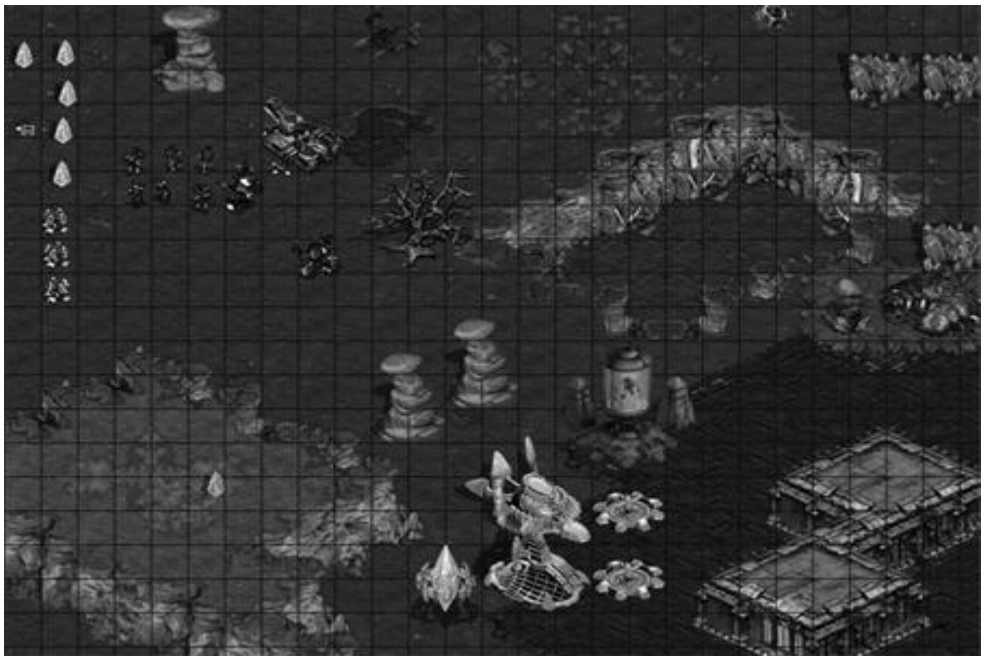


Figura 3. Uso de rejilla regular en el juego RTS: Starcraft.

Aquí el valor presente en cada celda representa árboles, murallas, enemigos, construcciones, entre otras cosas. Otros valores pueden ser asignados para representar agua, rocas, terreno, colinas, cuencas u otros.

Por estos valores la inteligencia artificial puede planear donde moverse, donde localizar enemigos, como evitar obstáculos, entre otros. La IA puede aplicar algoritmos de búsqueda para evitar los obstáculos naturales o artificiales por donde no sea permitido el movimiento.

A veces estos obstáculos, sin embargo, cumplen una función predominante al convertir ciertos lugares en posibles ubicaciones estratégicas dentro del juego como lugares de difícil acceso o de una buena defensa por lo que son tomados en consideración.

Además la IA puede encontrar el enemigo al cual atacar y localizar el lugar desde donde atacarlo. Para realizar esta búsqueda la Inteligencia Artificial necesita conocer la información que le brindan las celdas del área donde se encuentra el objetivo así como las celdas que se encuentran alrededor de esta área.

Aún cuando el modelo cognitivo de rejilla es muy simple de usar e implementar, tiene una desventaja que radica en que la búsqueda de espacios puede tender a volverse

extremadamente larga. Para un modesto mapa de celdas de 100 x 100, se necesita un grafo constituido por más de 10 000 nodos y aproximadamente 78 000 vértices.

Dado que los juegos RTS conllevan regularmente cientos y miles de unidades inteligentes activas (BOTs) cada tiempo, hace que halla una gran demanda de grafos de búsqueda cada vez que se actualiza un paso. Por cada actualización se realiza un alto procesamiento de datos sin mencionar la cantidad de memoria necesaria para realizarlo. Por fortuna existen una serie de métodos disponibles para disminuir la sobrecarga de memoria y que este procesamiento de datos sea óptimo.

1.2.2 Mallas de navegación:

Uno de los modelos más usados para representar mundos 3D son las mallas de navegación, las cuales son conocidas como NavMesh. Una malla de navegación es un grupo de polígonos convexos que describen la superficie del ambiente 3D. Este es un simple, y altamente intuitivo modelo cognitivo donde los caracteres de la inteligencia artificial realizan un plan que pueden usar para la navegación y la búsqueda de caminos en el mundo.

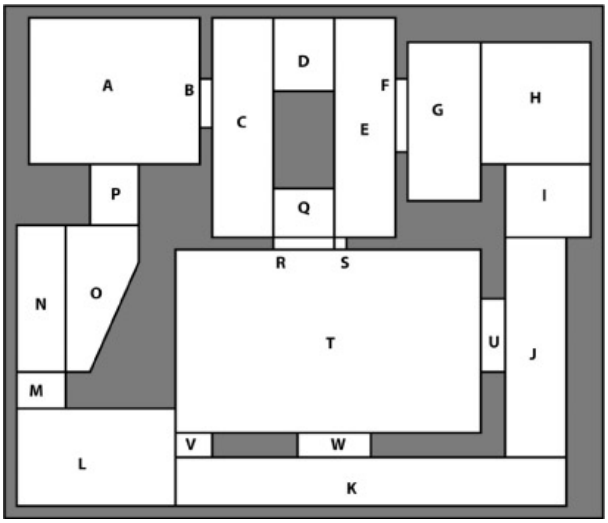


Figura 4. Malla de navegación.

Una buena idea presente en el concepto de malla de navegación es que el tamaño de la malla no depende del tamaño del mundo, pero sí de las figuras geométricas de las barreras o paredes y de la geometría del entorno. Adicionalmente como en el interior de un polígono convexo el agente puede moverse libremente, entonces es muy fácil de calcular el movimiento en batallas donde el BOT tiene necesidad de moverse

repentinamente en cualquier dirección para evitar una colisión o el fuego de un enemigo.

Sin embargo la desventaja es que es muy difícil y tedioso encontrar todos los polígonos a través de los cuales se pueda mover o andar y los caminos para alcanzarlos desde polígonos vecinos. Algunas veces este proceso no es acertado en las propiedades físicas de los modelos y esto puede marcar algunos polígonos que pueden ser alcanzados cuando en realidad las leyes físicas impuestas por la simulación no lo permitan.

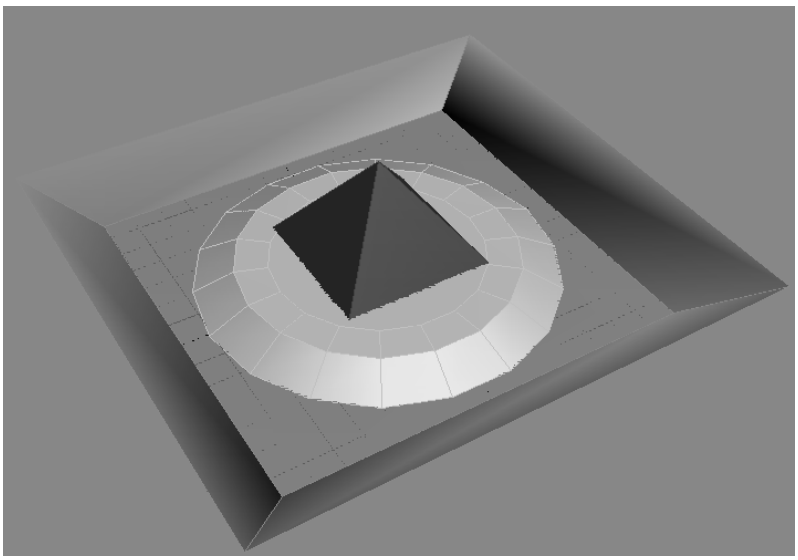


Figura 5. Representación de una malla de navegación

1.2.3 Puntos de Visibilidad (POV)

Un grafo de navegación de puntos de visibilidad (POV) o waypoints es creado por la inserción de puntos, usualmente de forma manual, en lugares importantes del ambiente, tal que cada nodo del grafo tiene una línea de observación que lo conecta con los nodos visibles.

Posicionados de forma cuidadosa es posible crear una conexión de nodos del grafo en las áreas geométricas más importantes del mundo virtual. Este posicionado tiene la importancia de lograr una representación factible hacia los actores que interactúan con el entorno y de permitir un acceso mejor al terreno.

Las características de los puntos de visibilidad es que describen el terreno accesible a los actores que interactúan con este. Describen el movimiento válido y las propiedades del entorno local como el acceso a lugares dentro del entorno.

Las razones por las cuales los puntos de visibilidad son tan atractivos dentro de la Inteligencia Artificial es que son fáciles de crear y de usar, y rápidos

Los puntos de visibilidad se usan para hacer la navegación en el grafo, para marcar el terreno accesible a la inteligencia artificial y a los jugadores, además para describir y representar las fronteras, expresar los caminos y el tiempo de viaje por estos.

Además estos se usan para predecir y soportar la búsqueda de pistas, donde a nivel de waypoints se puede encontrar los vecinos cercanos a un punto determinado, buscar el tiempo de viaje desde un punto hacia los puntos cercanos a este y qué puntos se pueden ver desde el lugar donde se encuentre el BOT.

Un grafo más denso trae como desventaja que los algoritmos para recorrerlo demoran en realizar los cálculos como por ejemplo para encontrar el camino de acceso a un área o para buscar el camino más corto entre un nodo y otro.

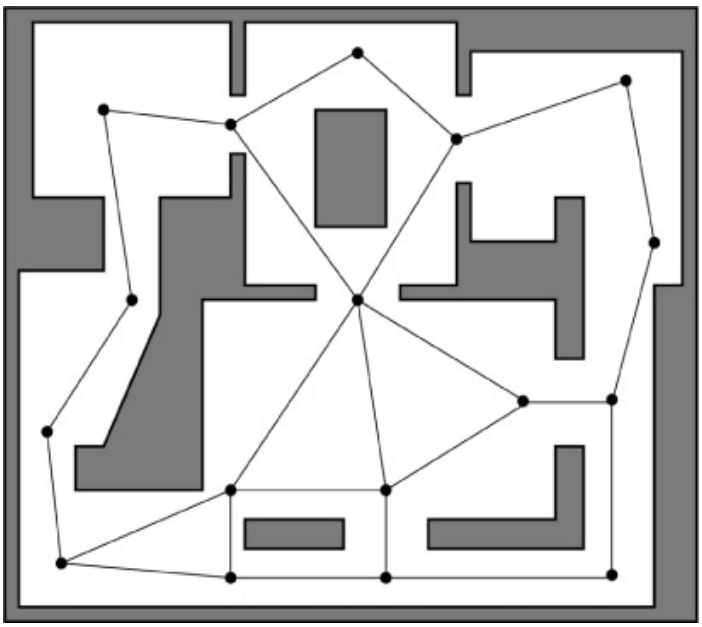


Figura 6. Puntos de Visibilidad

Cada punto en el mapa esta conectado a todos los puntos que pueden verse desde él. Estos nodos no tienen que ser definidos exclusivamente en 2D como se ve en la figura

anterior sino que es posible insertarlo en mundos tridimensionales donde estos nodos representen las posibilidades de movilidad como se muestra en la siguiente figura:

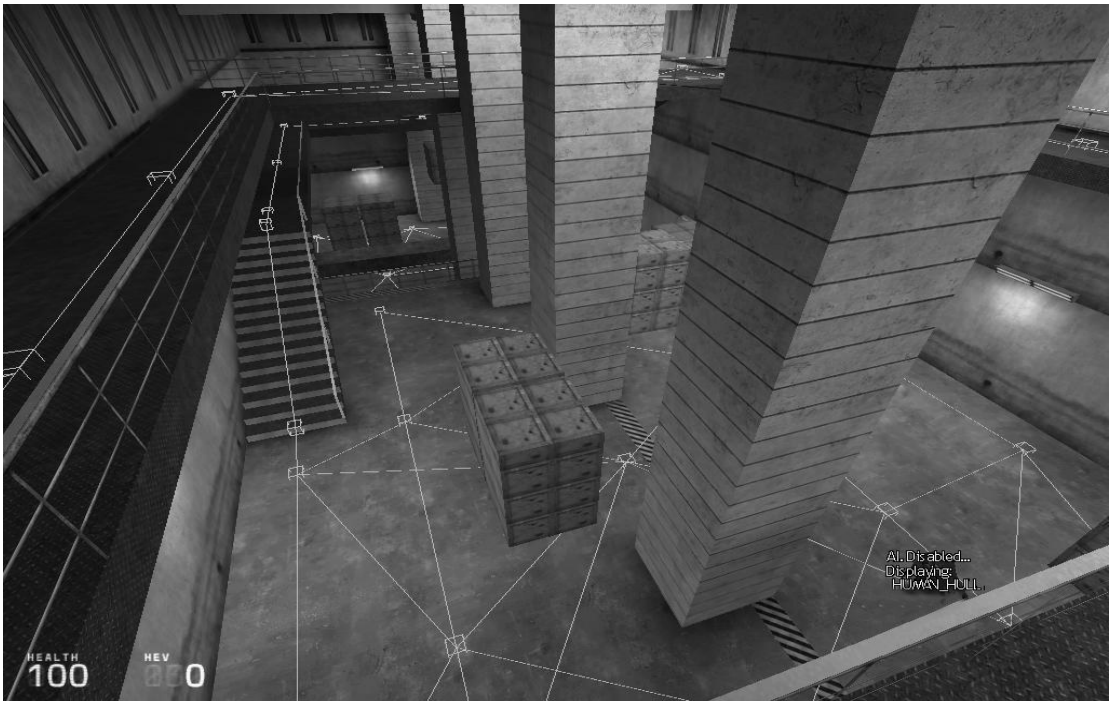


Figura 7. Representación de puntos de visibilidad en un mundo 3D

La desventaja de los puntos de visibilidad es que no entregan información acerca de los obstáculos que se encuentran a su alrededor o de la posibilidad de usar caminos alternos donde existan obstáculos que se puedan mover o desplazar. Esto ocurre, por ejemplo, en el caso de una puerta que exista en el entorno o de un foso. Por eso, un punto de visibilidad, tiene que analizar todos los puntos cercanos a él y analizar si existe alguna conexión hacia alguno de ellos.

Un grafo de puntos de visibilidad puede ser además problemático si se tiene el propósito de incluir algún tipo de características de generación de terrenos por lo que se tiene que desarrollar algún método automatizado de generación de estructuras de grafos de puntos de visibilidad. Esto es problemático porque el tiempo de generación y optimización del nuevo grafo consume tiempo.

Para moverse un agente dentro del mundo, la inteligencia artificial tiene que seguir los siguientes pasos:

1. Encontrar todos los nodos del grafo cercanos a la localización del agente actual, nodo A.

2. Encontrar todos los nodos del grafo cercanos a la localización del objetivo, nodo B.

3. Usar algoritmos de búsqueda para encontrar el camino más corto, el camino de menos costo, desde A hasta B.

4. Mover el agente del nodo A.

5. Mover el agente por el camino calculado en el paso 3.

6. Mover el agente desde B a la localización del objetivo.

Estos pasos se muestran a continuación:

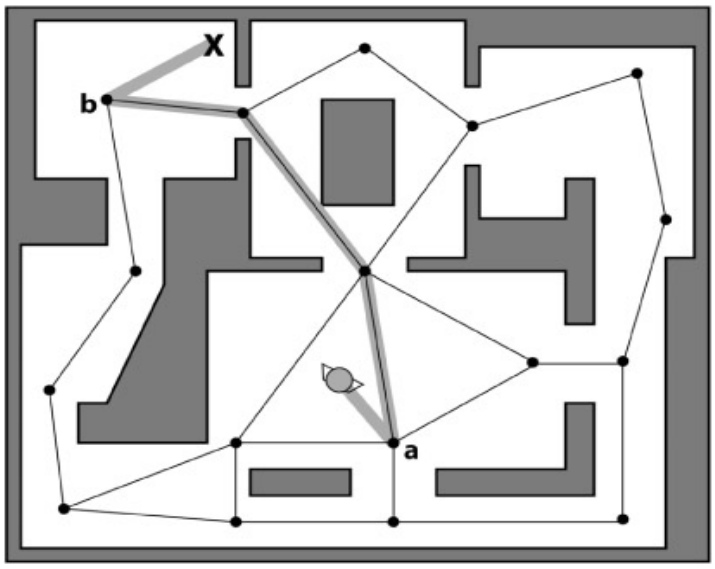


Figura 8. Búsqueda de un camino óptimo a través de puntos de visibilidad.

Cuando se esta diseñando el modelo cognitivo hay que tener mucho cuidado en la inserción de puntos dentro de este para evitar la inserción de puntos ciegos. Para determinar una posición determinada en el entorno, los algoritmos tienen que encontrar todos los puntos visibles cercanos a partir de aquella posición y el movimiento del agente dentro de la localización.

Puede ocurrir que para determinar una posición determinada no sea posible encontrar ningún punto de visibilidad que brinde esta información y a partir de esto aparecen áreas ciegas o puntos ciegos.

Las áreas ciegas no son más que áreas del entorno que quedan fuera del rango de los nodos del grafo de búsqueda. En la siguiente figura se muestran con signos de admiración las áreas ciegas:

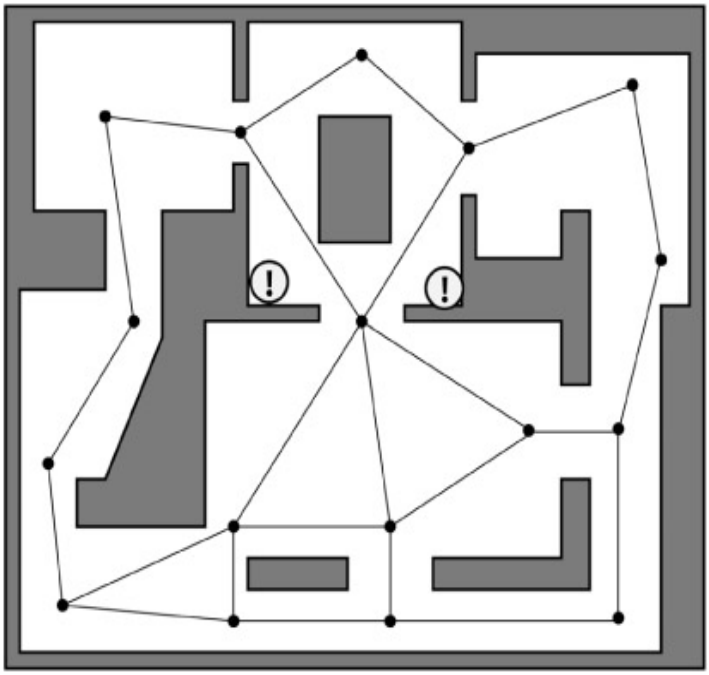


Figura 9. Representación de áreas ciegas en un grafo de búsqueda.

Aunque algunas veces cuando se insertan los puntos se dejan áreas ciegas dentro del grafo de búsqueda con el fin de lograr áreas específicas donde se pueda esconder el actor de los BOTs.

Si el jugador o actor encuentra un área de este tipo entonces se vuelve invisible para el BOT pues no puede ser visto desde ningún nodo cercano. Estas áreas se usan a veces para poder esconderse de las IA como sucede en juegos famosos como el de Need for Speed Most Wanted donde se usan para esconderse de la policía.

1.3 Análisis topográfico del terreno

Un entorno virtual es prácticamente idéntico a un mundo real, tiene casi todas las características de un terreno común. Una de las diferencias es que uno es generado por el computador con ayuda de personas y el otro por la naturaleza o el hombre.

Para lograr obtener los datos específicos del terreno se usa el análisis topográfico o topológico. Estos datos pueden ser desde la forma de un terreno hasta la presencia de

algún tipo de objeto natural o artificial sobre él además de datos estratégicos como los mejores puntos defensivos, de emboscada, entre otros.

El análisis topológico en el mundo virtual se basa fundamentalmente en el uso algoritmos muy sofisticados y eficientes que permiten generar decisiones tácticas, y esto se realiza a través de los grafos de navegación que hay en el mundo.

Como los grafos se usan principalmente para la navegación, contienen información acerca de la relación entre las diferentes posiciones en el entorno. Esta información puede ser explotada para calcular de forma eficiente el valor estratégico de una posición específica en el terreno.

Este tipo de información puede revelar la información estratégica que se tiene de un nodo determinado en la red del grafo, donde se puede, por ejemplo, analizar cual es el nodo principal que permite el acceso a una región determinada, los nodos que tienen una buena posición defensiva, o para realizar una emboscada.

A través del análisis topográfico se calculan una serie de datos tácticos que permiten a la computadora actuar y tomar decisiones contra un jugador o un grupo de jugadores.

Existen muchos tipos de análisis tácticos, por ejemplo, la emboscada (ambush), los puntos fuertes (stronghold), los lugares ocultos (hiden), los lugares buenos para colocar francotiradores (sniper), las regiones que ofrecen defensas naturales o artificiales (cover), entre otros.

1.3.1 Ambush

Los puntos de emboscada son aquellos que presentan una visibilidad amplia de un área determinada y que a su vez brindan defensas naturales o artificiales. Además estos puntos siempre se ubican en lugares por donde es obligatorio pasar para llegar a un objetivo en particular pero siempre teniendo en cuenta que sean zonas que no sean descubiertos hasta que se realice la acción planeada.

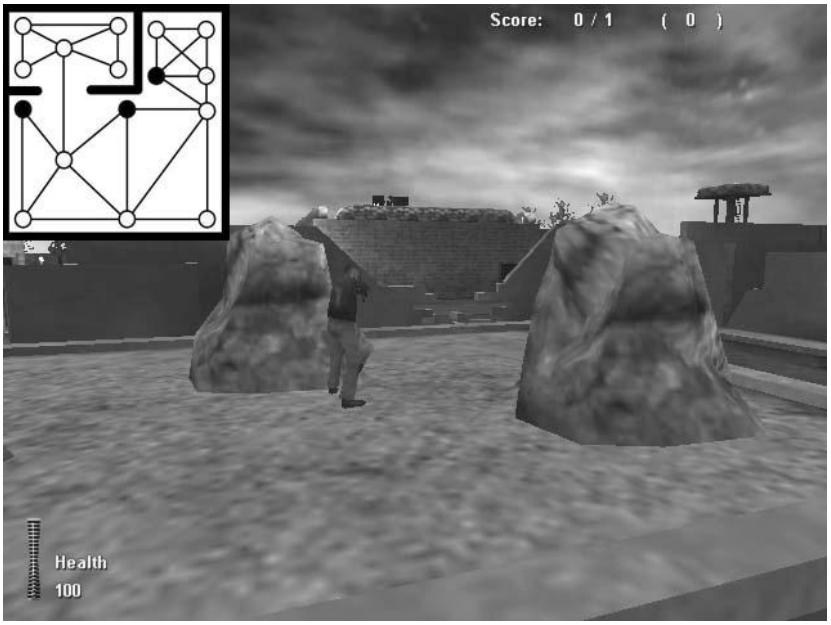


Figura 10. Representación de los puntos de Ambush.

1.3.2 Stronghold

Los puntos fuertes son aquellos en donde el jugador se puede resguardar para realizar una defensa óptima de un punto en especial. Estos puntos reúnen las características de ser puntos de difícil acceso, con una visión amplia sobre el área a defender y con los accesos cubiertos. Es una posición defensiva excelente.

Figura 11. Representación de un punto de Stronghold.

1.3.3 Hide

Los lugares ocultos son los puntos que no son visibles en un área del entorno. Son puntos que ofrecen la posibilidad en el terreno de esconderse del enemigo.

Figura 12. Representación de un punto Hide.

1.3.4 Sniper

Los puntos más óptimos para colocar francotiradores en el terreno son aquellos lugares que no se noten mucho, que pasen desapercibidos, difíciles de alcanzar por parte de los objetivos del francotirador, puntos que se encuentren en un lugar que sea poco visible, lugares donde sea difícil de sorprender al francotirador, que la visibilidad sea la mayor posible de una región determinada, que exista una libertad de movimiento, que exista un lugar donde el francotirador tenga una posición defensiva excelente y que estén lejos de la acción.

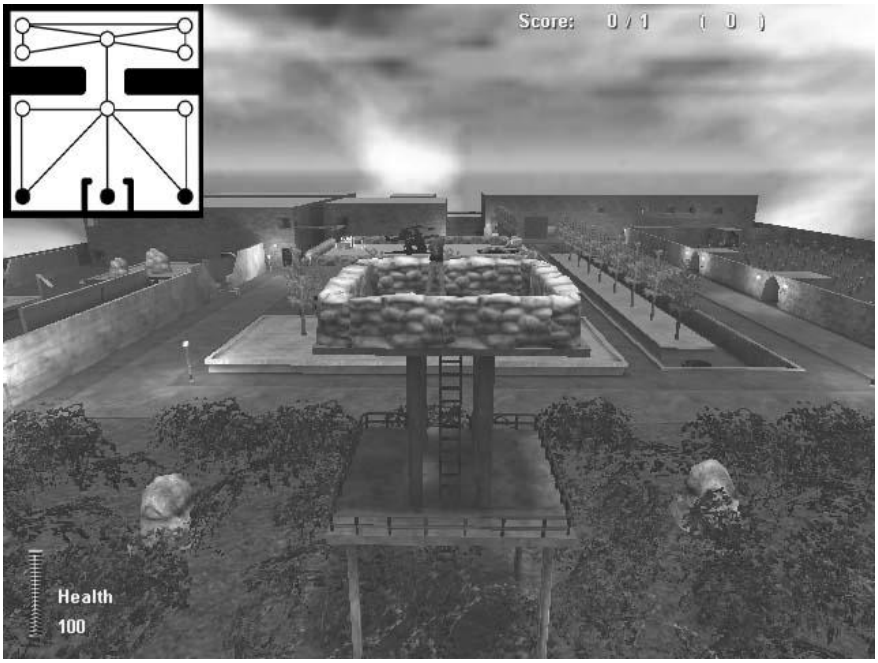


Figura 13. Representación de puntos de Sniper.

1.3.5 Cover

Los puntos de defensas son aquellos lugares que ofrecen protección durante una sorpresa y que puedan ser usados para resguardarse durante un ataque. Son puntos que presentan defensas naturales o artificiales.

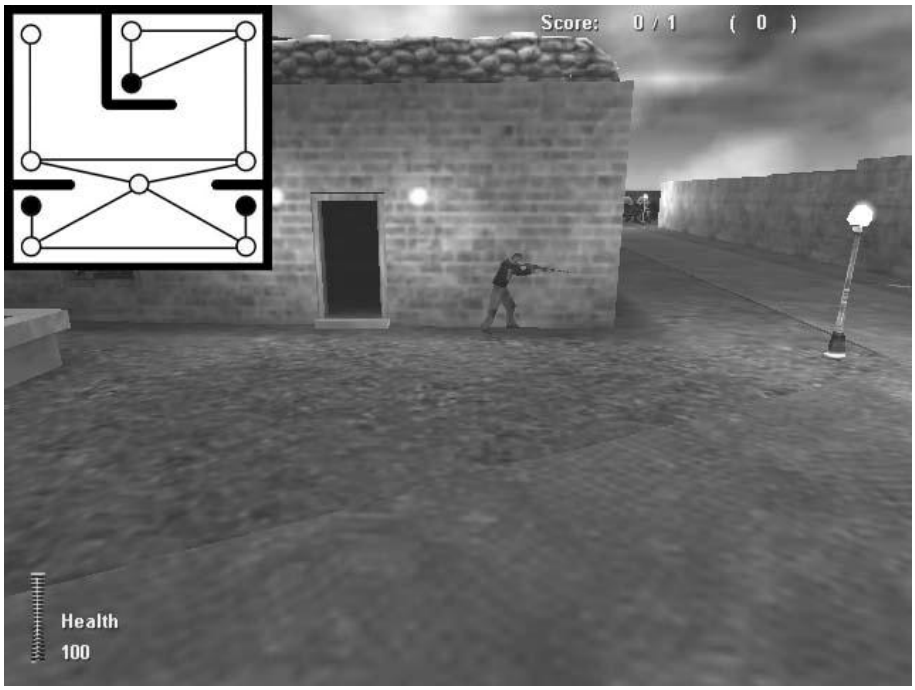


Figura 14. Representación de puntos Cover.

1.4 Metodologías y Herramientas de desarrollo

La existencia de una serie de aplicaciones posibles para llevar a cabo la realización de la herramienta propuesta hizo que fuera necesario hacer una selección de las más óptimas para ello. Para seleccionarlas se utilizaron una serie de parámetros como ventaja, tendencia actual, dominio y fortaleza.

1.4.1 Metodología

La metodología aplicada fue RUP y no solo por ser un proceso de desarrollo de software complejo, sino por ser un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software con diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.

Sin embargo, hay tres características fundamentales que lo hacen una metodología robusta y poderosa: es dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental.

Cuando se caracteriza RUP como dirigido a casos de uso se refiere a que se sigue un hilo, avanzando a través de una serie de flujos de trabajo que parten de los casos de uso. Los casos de uso se especifican, se diseñan y a partir de los casos de uso finales se obtienen los casos de uso de prueba.

El desarrollo de los casos de uso no se hace de forma aislada sino que se desarrollan con la arquitectura del sistema, por lo que tanto la arquitectura del sistema como los casos de uso maduran a medida que avanza el sistema.

Centrado en la arquitectura se refiere a que se incluye los aspectos estáticos y dinámicos más significativos del sistema. Además recoge una serie de factores como la plataforma en la cual funciona el software, los bloques de construcción reutilizables que se tienen, consideraciones de implementación, sistemas heredados y requisitos no funcionales.

Es iterativo e incremental porque el proyecto o el desarrollo de una aplicación se pueden dividir en partes y desarrollarlas de manera iterativa, incrementándose a medida que se integran unas con otras hasta llegar a formar la tarea final. Las iteraciones hacen referencia a pasos en el flujo de trabajo y los incrementos, el crecimiento del producto.

1.4.2 Herramientas de desarrollo

Las herramientas de desarrollo recogen el software usado en el desarrollo de la aplicación y en el modelado de los casos de uso. En el desarrollo de la herramienta propuesta se usó el Visual Studio 2005 y para crear la documentación de esta, el Enterprise Architect.

1.4.2.1 Visual Studio 2005

El Visual Studio 2005, es una herramienta poderosa, fuerte y voluminosa que tiene una gran integración de varios lenguajes entre ellos el C++, C#, y Asp.Net. Tiene la posibilidad de implementar aplicaciones para soluciones integrales que aprovechen de manera óptima la ventaja de cada lenguaje. Acelera de manera significativa la producción de software. Mejora en gran grado los resultados finales y optimiza el desempeño. La documentación del Visual Studio está entre las mejores.

La interfaz es altamente amigable con el usuario permitiendo que el tiempo en implementar una solución o aplicación determinada sea mucho menor. Los ejecutables desarrollados por esta herramienta son generalmente de menor tamaño lo que hace que ocupe un lugar cimero en la producción de software al lograr aplicaciones óptimas y de poco volumen.

Otro de los motivos por los cuales se usa esta herramienta para desarrollar el software propuesto es que esta es la plataforma de desarrollo usada en el proyecto Paseos Virtuales para las realizaciones de las aplicaciones tales como paseos virtuales, simulaciones y actualmente en el desarrollo de video juegos.

1.4.2.2 Enterprise Architect

El Enterprise Architect es una herramienta que trabaja con la ingeniería de software asistida por computadoras (CASE) para el diseño y construcción de software. Este soporta las especificaciones del UML 2.0 el cual describe el lenguaje visual que se usa para definir modelos o diagramas en un proyecto.

El Enterprise Architect (EA), es una herramienta avanzada que tiene soporte para todos los ciclos de desarrollo, proporcionando el desarrollo completo para la fase de diseño inicial a través del despliegue y mantenimiento. Además soporta pruebas, mantenimiento y control de cambios.

Entre las características más importantes del EA está que crea modelos UML para un amplio rango de propósitos. Además coloca esos elementos en diagramas o paquetes, crea conectores entre estos, documenta los elementos creados, genera código en un gran número de lenguajes, realiza la ingeniería inversa o hacia delante en: ActionScript, C++, C#, Delphi, Java, Python, PHP, VB. NET y Visual Basic.

Usando EA se puede sincronizar código y elementos del modelo. Así como diseñar y generar elementos de base de datos. Además permite que los modelos creados sean automáticamente documentados con una alta calidad en el formato .RTF, usado por el Microsoft Word.

Con Enterprise Architect se pueden modelar procesos de negocio, sitios Web, interfaces de usuario, redes, configuración de hardware, mensajes y más. Estima el tamaño del esfuerzo del trabajo del proyecto en horas.

1.5 Lenguajes de programación

Para el desarrollo de la herramienta se escogieron dos lenguajes fundamentales, el C++ y el UML. El primero para realizar la implementación de la aplicación y el otro crear la documentación.

1.5.1 C++

El C++ se escogió por ser un lenguaje de programación estandarizado por la norma ISO/IEC 14882:1998. Entre sus principales características está el soporte para la programación orientada a objetos y el soporte de planillas o programación genérica, además de ser un lenguaje de alto nivel que está considerado como un lenguaje potente al poder trabajar tanto en bajo como en alto nivel.

Posee dos propiedades fundamentales que son difíciles de encontrar en otros lenguajes que son la posibilidad de redefinir los operadores, comúnmente conocido como la sobrecarga de operadores, y la identificación de tipos en tiempo de ejecución.

Además presenta una biblioteca estándar muy poderosa y es muy usado tanto en el ámbito educacional como profesional.

1.5.2 UML

El UML se escogió para realizar el modelado del análisis y el diseño de la aplicación. El UML es el Lenguaje Unificado de Modelación (Unified Modeling Language).

El UML ofrece un modo estándar de visualizar, especificar, construir, documentar y comunicar los artefactos de un sistema basado en el software que debe usarse en el proceso completo del desarrollo del mismo.

Capítulo 2.Solución Propuesta

2.1 Ciclo Propuesto

Para la recogida y análisis de datos del mundo se usan los grafos de navegación, que permiten a través de los nodos la obtención de valores y datos estratégicos. Los grafos de navegación están formados por una serie de nodos que están interconectados entre si, formando una malla o red. En dependencia del entorno y del mundo pueden localizarse los nodos de forma bidimensional o tridimensional.

Para la edición de los nodos o grafos, se usan una serie de aplicaciones que permiten la manipulación y creación de estos dentro del mundo. La edición de los grafos se hace casi siempre de forma manual por los diseñadores y en algunos casos por aplicaciones especializadas en esto.

Las aplicaciones como Maya y 3D Studio MAX, dos software de diseño, pueden realizar la creación de grafos con el procesamiento y uso de otras herramientas.

Figura 15. Proceso de desarrollo.

En el software de diseño 3D Studio MAX se crea el modelo que se utiliza en la herramienta desarrollada en este trabajo para que se realice el procesamiento, creación y simplificación de grafos.

La herramienta propuesta carga un modelo del mundo. El usuario escoge los algoritmos con los cuales desea trabajar y los aplica, tales como el Flood Fill, Simplify o FixConnections.

Cuando se termina totalmente el trabajo con el grafo el usuario tiene la opción de realizar la exportación de este a un formato .ASE o ASCII. El formato ASCII es una

manera simple de representar el grafo con sus nodos y las conexiones entre los mismos. Este fue creado para su uso dentro del proyecto.

El grafo terminado se puede usar para aplicaciones como juegos, entornos virtuales y simuladores. A partir de aquí y a través de algoritmos de inteligencia artificial se realiza el análisis de los diferentes nodos para obtener datos o valores como el camino más corto a una región o nodo, o el camino más largo.

Además con este grafo se pueden encontrar los límites del terreno, los obstáculos, las puertas o entradas, los mejores puntos de emboscada, los mejores puntos para francotiradores, entre otros.

2.2 Requerimientos de Entrada

La herramienta desarrollada para realizar el procesamiento de grafos de navegación tiene una entrada principal: el modelo del mundo. Cada entrada tiene que cumplir obligatoriamente con todos los requisitos para que el software pueda dar un resultado válido. Estos requisitos son los siguientes:

- **El modelo del mundo:**
  - Debe existir.
  - Debe ser un modelo cerrado.
- **Cada polígono del modelo a cargar:**
  - Debe tener exactamente tres aristas o tres vértices.

Los requerimientos anteriores se pueden apreciar de forma gráfica en la siguiente figura:

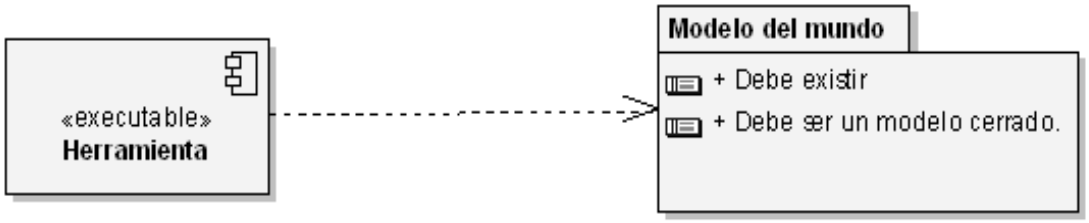


Figura 16. Requerimientos de Entrada.

2.3 Funcionalidades

2.3.1 Procesadores

Los procesadores son algoritmos que realizan un análisis y trabajo con el grafo. La herramienta tiene un grupo principal de algoritmos: los algoritmos de procesamiento.

2.3.1.1 Algoritmos de Procesamiento:

Los algoritmos de procesamiento crean, optimizan y modifican los nodos de un grafo de navegación. La herramienta aplica principalmente tres algoritmos al grafo con el cual se va a trabajar.

- **Flood Fill**

  Es un algoritmo de relleno. Su función es generar un grafo a partir de la inserción de nodos, dentro de la geometría del modelo de un mundo previamente cargado. Este comienza por un nodo inicial llamado Seed (semilla) y realiza el relleno de toda el área o región comprendida por el modelo del mundo, creando de este modo un grafo.

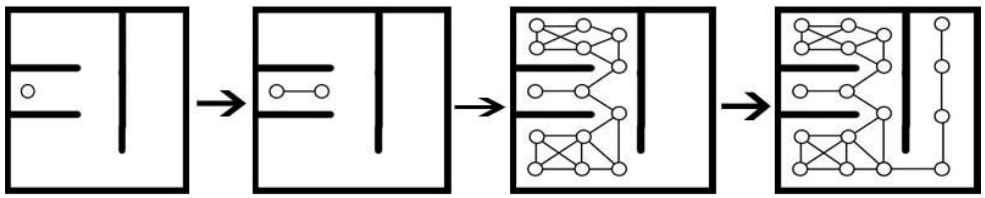


Figura 17. Representación del algoritmo Flood Fill.

- **Simplify:**

  Es un algoritmo de procesamiento que a partir de un grafo de navegación inicial, realiza la simplificación de los nodos que lo conforman. La simplificación o eliminación de los nodos se realiza suprimiendo los nodos innecesarios dentro del grafo.

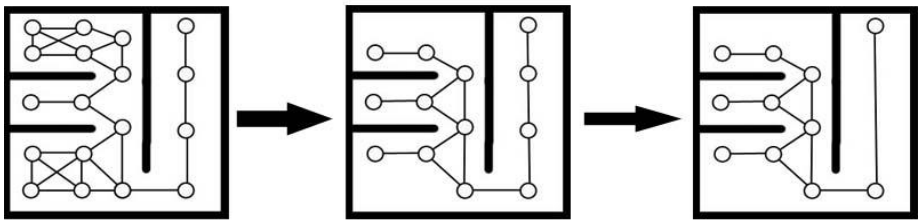


Figura 18. Representación del algoritmo Simplify.

- **FixConnections:**

Es un algoritmo que analiza a partir de un grafo de navegación, si existe algún nodo que no contenga una conexión a un nodo dentro de su rango de visibilidad. Si existe este, se crea la conexión entre los dos.

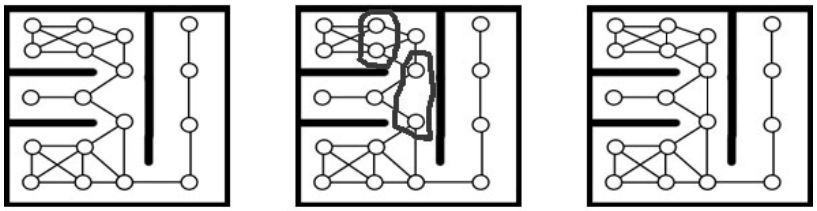


Figura 19. Representación del algoritmo FixConnections

Además la herramienta podría implementar dos algoritmos más de procesamiento para ampliar y obtener mejores resultados. Estos se definen a continuación:

- **Hierarchy:**

  Este algoritmo realiza el análisis del grafo. Después analiza cada región del grafo en el mundo. Al terminar este estudio, realiza la agrupación de los nodos de una región en común, y crea un nodo en esta área analizada. Esto es aplicado a cada área en el grafo, obteniéndose uno nuevo.

  En el grafo obtenido por el algoritmo, cada nodo guarda otro grafo, siendo este los nodos del área analizada.

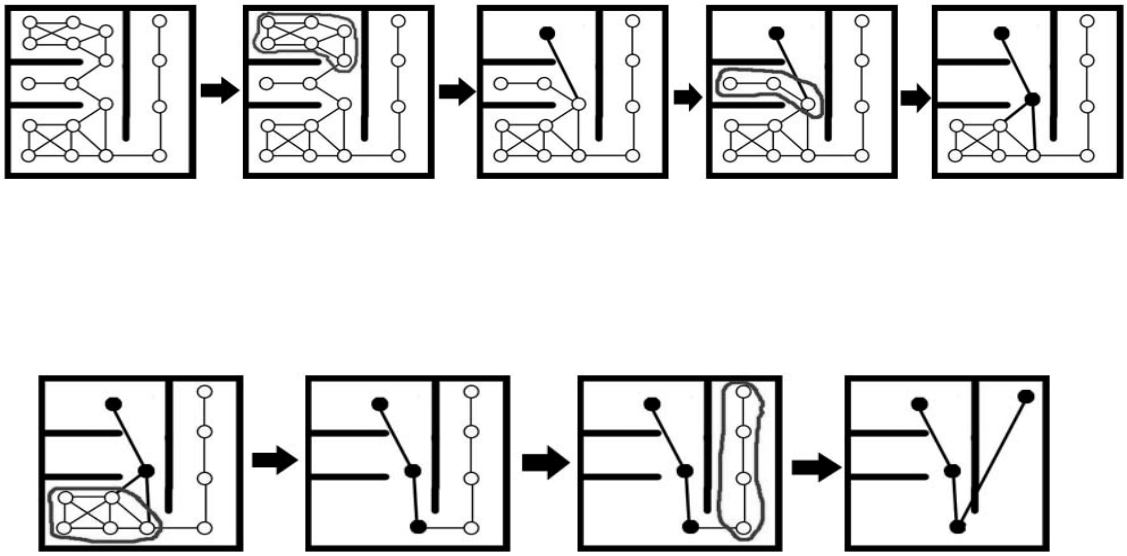


Figura 20. Representación del algoritmo Hierarchy.

- **Geometry Strode:**

Este algoritmo realiza una expansión pequeña de la geometría del mundo y a través de esta crea, donde exista un vértice en la forma expandida, un nodo, obteniéndose de este modo un grafo. Después se realiza la conexión entre cada nodo visible del grafo para así formar el grafo final.

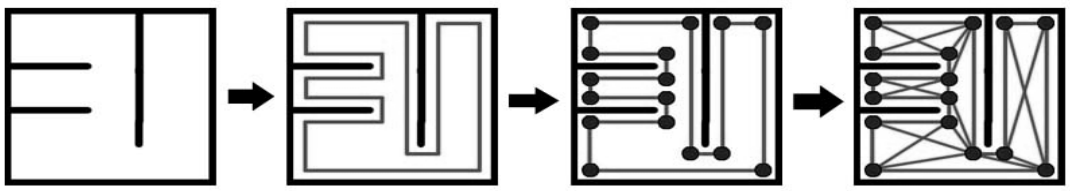


Figura 21. Representación del algoritmo Geometry Strode.

Capítulo3. Sistema, producto y funciones del sistema.

3.1 Requerimientos:

Los requerimientos son una descripción de las necesidades de un producto. Su tarea principal es identificar y documentar las necesidades reales del sistema, de una forma clara a los clientes o miembros del equipo de desarrollo. También son una condición o capacidad que debe estar presente en un sistema o componentes de este para resolver un contrato, un estándar, una especificación u otro documento formal.

Los requerimientos se dividen en requerimientos funcionales y no funcionales.

3.1.1 Requerimientos funcionales:

Estos definen las funciones que el sistema será capaz de realizar y además describe las transformaciones que el sistema realiza sobre las entradas para producir salidas. Los requerimientos funcionales del presente software se muestran a continuación:

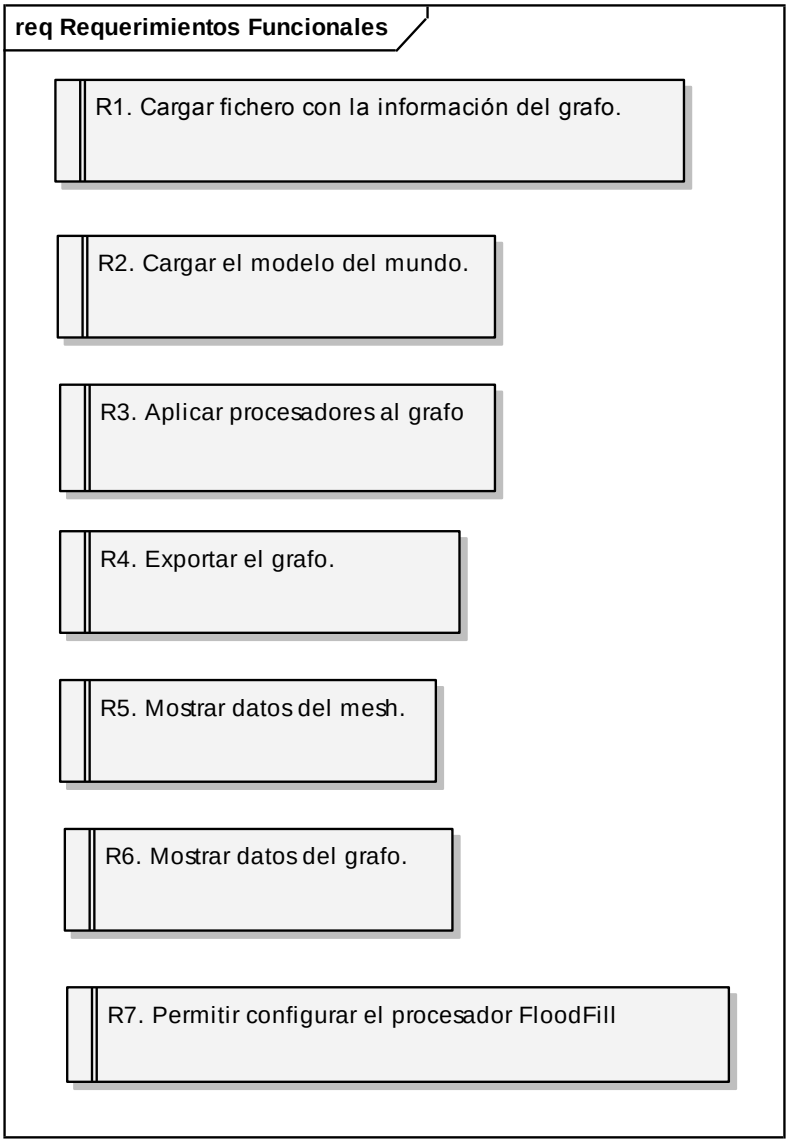


Figura 22. Requerimientos funcionales.

R1. Cargar Fichero con la información del grafo

R1.1 Cargar el fichero de formato ASCII.

R2. Cargar el modelo del mundo en un formato que presente datos de los polígonos.

R2.1 Cargar el modelo en formato .X

R3. Aplicar algoritmos de procesamiento de datos al grafo.

R3.1 Aplicar algoritmo Flood Fill.

R3.2 Aplicar algoritmo Simplify.

R3.3 Aplicar algoritmo FixConnections.

R4. Exportar el grafo a un formato determinado para su posterior uso en trabajos del proyecto.

R4.1 Exportar el grafo en formato .ASE.

R4.2 Exportar el grafo en formato .ASCII.

R5. Mostrar datos del mesh.

R6. Mostrar datos del grafo.

R7. Permitir configurar el procesador Flood Fill.

R7.1 Entrar coordenadas del nodo semilla.

R7.2 Entrar valor del offset.

3.1.2 Requerimientos no funcionales:

Son características, limitaciones, cualidades o propiedades del sistema.

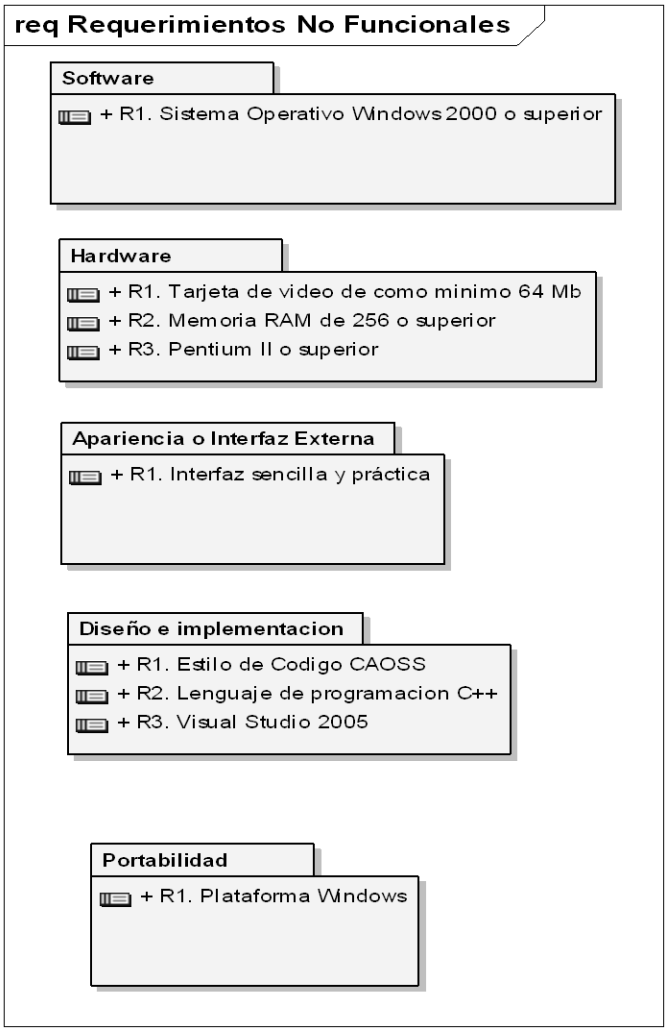


Figura 23. Requerimientos no funcionales.

3.2 Casos de Uso:

A partir de los Casos de Uso se describe como se cumplen los requerimientos dentro de la herramienta propuesta. Los Casos de Uso se refieren a descripciones narrativas de los procesos del dominio.

Los casos de uso para el desarrollo de la herramienta propuesta realizan una descripción de cómo interactuará con el trabajador y confieren la posibilidad de tener una visión de las funcionalidades que tendrá la aplicación desarrollada. Además de esto ejemplificarán el avance y desarrollo del software propuesto.

Modelado de Casos de Uso:

Actores	**Justificación**
Usuario	Es la persona que trabaja e interactúa con el software.

A continuación están definidos los casos de uso que conforman la herramienta propuesta y los requerimientos que implementa:

CU – 1	Cargar grafo de .ASCII
Actor	Usuario
Descripción	Se carga el fichero del grafo en formato .ASCII
Referencia	R1,R6

CU – 2	Cargar Mesh .X
Actor	Usuario
Descripción	Se carga el modelo del mundo en formato .X
Referencia	R2, R5

CU – 3	Aplicar procesador Simplify
Actor	Usuario
Descripción	Se aplica el algoritmo de procesamiento al grafo.
Referencia	R3, R6

CU – 4	Aplicar procesador FloodFill
Actor	Usuario
Descripción	Se aplica el algoritmo de procesamiento al grafo.
Referencia	R3, R6, R7

CU – 5	Aplicar procesador FixConnections
Actor	Usuario
Descripción	Se aplica el algoritmo de procesamiento al grafo.
Referencia	R3, R6

CU – 6	Exportar grafo en .ASCII
Actor	Usuario
Descripción	Se exporta el grafo final en formato .ASCII
Referencia	R4

CU – 7	Exportar grafo en .ASE
Actor	Usuario
Descripción	Se exporta el grafo final en formato .ASE
Referencia	R4

El siguiente diagrama representa la relación que se establece entre el actor y los Casos de Uso del sistema:

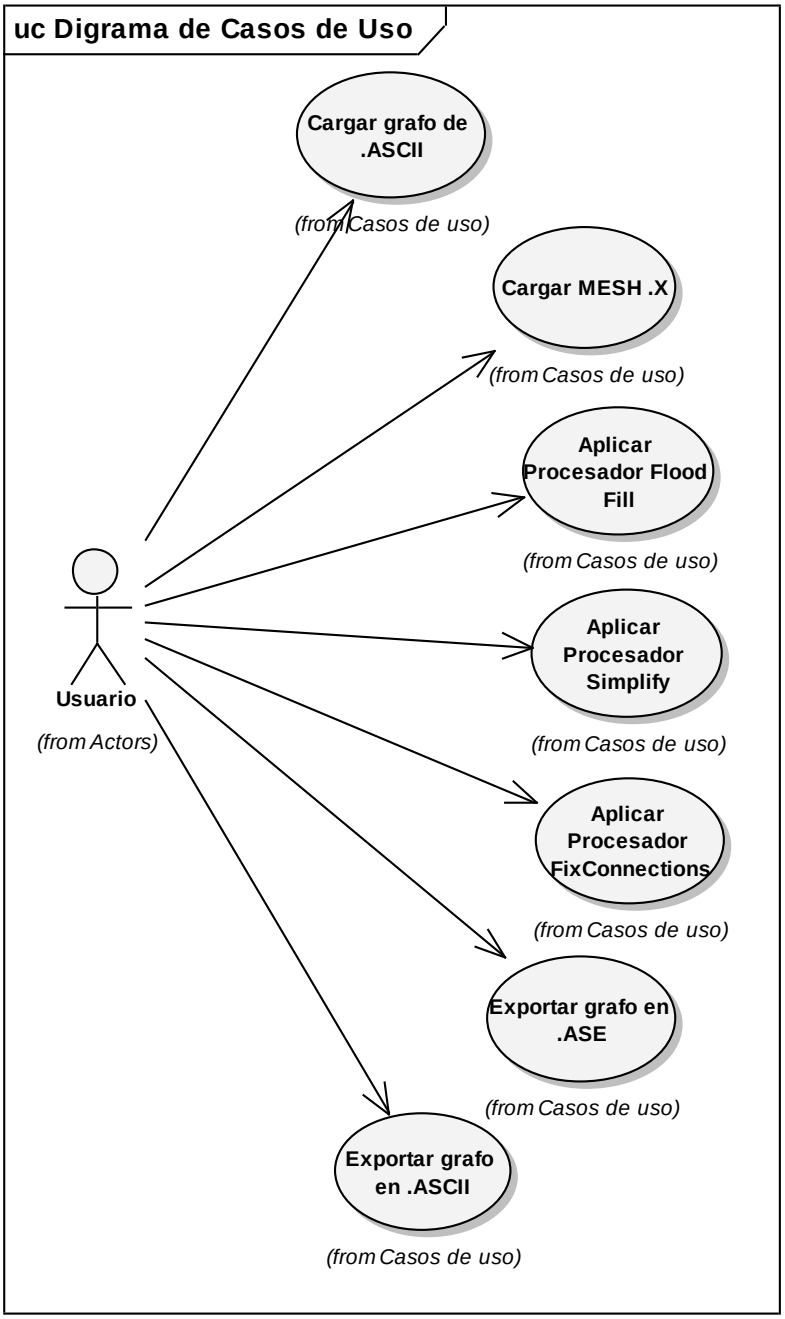


Figura 24. Diagrama de Casos de Uso.

Los Casos de Uso expandidos describen paso a paso la secuencia de eventos que los actores realizan en el sistema para completar una acción en el sistema:

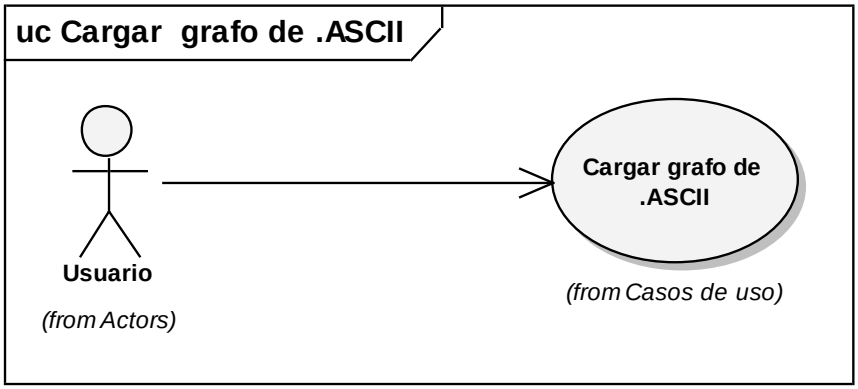


Figura 25. Caso de Uso cargar grafo de .ASCII

Caso de Uso: Cargar grafo de .ASCII		
Propósito:	Cargar el grafo del formato ASCII	
Actores: Usuario.		
Resumen: El CU se inicia cuando el usuario ejecuta la aplicación y se prepara para trabajar con ella. Se carga el fichero del grafo en formato .ASCII.		
Referencia:	R1, R6	
Precondiciones:	El usuario ejecuta la aplicación y se prepara para trabajar con ella.	
Acción del actor:		**Respuesta del Sistema:**
1. El usuario elige la opción “File” del menú principal, y ahí escoge la opción “Load”.		2. La aplicación muestra dos opciones: a) Grafo ASCII b) Mesh .X
3. El usuario escoge la opción Grafo ASCII.		4. El sistema muestra la ventana de un explorador para buscar el fichero en la PC.
5. El usuario encuentra el fichero y lo selecciona.		6. El sistema carga los datos del fichero en la aplicación.
Puntos de Extensión: R1.1		

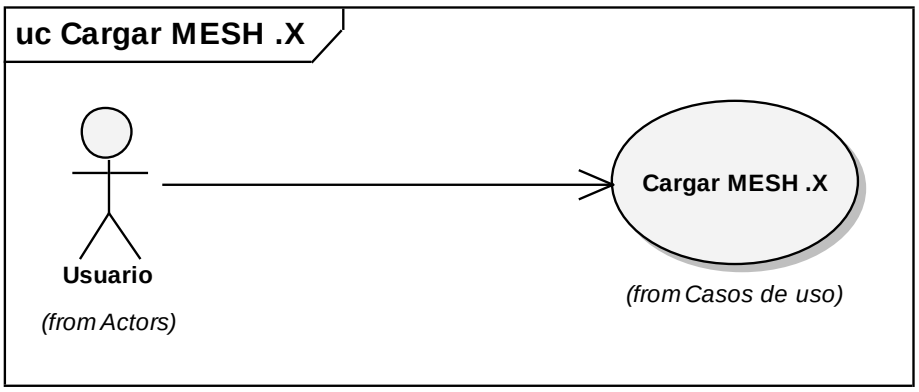


Figura 26. Caso de Uso cargar MESH .X

Caso de Uso: Cargar MESH .X		
Propósito:	Cargar el modelo del mundo en formato .X	
Actores: Usuario.		
Resumen: El CU se inicia cuando el usuario ejecuta la aplicación y se prepara para trabajar con ella. Se carga el modelo del mundo.		
Referencia:	R2, R5	
Precondiciones:	El usuario ejecuta la aplicación y se prepara para trabajar con ella.	
Acción del actor:		**Respuesta del Sistema:**
1. El usuario elige la opción “File” del menú principal, y ahí escoge la opción “Load”.		2. La aplicación muestra dos opciones: a) Grafo ASCII b) Mesh .X
3. El usuario escoge la opción Mesh .X.		4. El sistema muestra la ventana de un explorador para buscar el fichero en la PC.
5. El usuario encuentra el fichero y lo selecciona.		6. El sistema carga los datos del fichero del mesh en la aplicación.
Puntos de Extensión:		

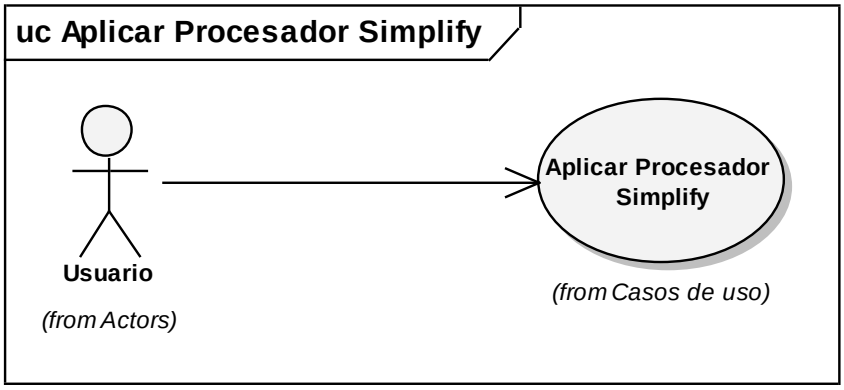


Figura 27. Caso de Uso aplicar procesador simplify.

Caso de Uso: Aplicar Procesador Simplify		
Propósito:	Aplicar el procesador Simplify.	
Actores: Usuario.		
Resumen: El CU se inicia cuando el usuario tiene los datos del grafo y el modelo del mundo. El mismo escoge la opción aplicar procesador Simplify, aplicándose éste al grafo.		
Referencia:	R3, R6	
Precondiciones:	Debe existir un grafo. Debe existir un mesh.	
Acción del actor:		**Respuesta del Sistema:**
1. El usuario escoge la opción “Processors”.		2. El sistema despliega un menú con las siguientes opciones: a) Flood Fill b) Simplify c) FixConnections
3. El usuario escoge la opción Simplify.		4. El sistema ejecuta el algoritmo predeterminado para realizar el procesamiento de los nodos. 5. El sistema guarda los datos procesados. 6. El sistema muestra los datos del grafo al finalizar la aplicación del procesador.
Puntos de Extensión: R3.2		

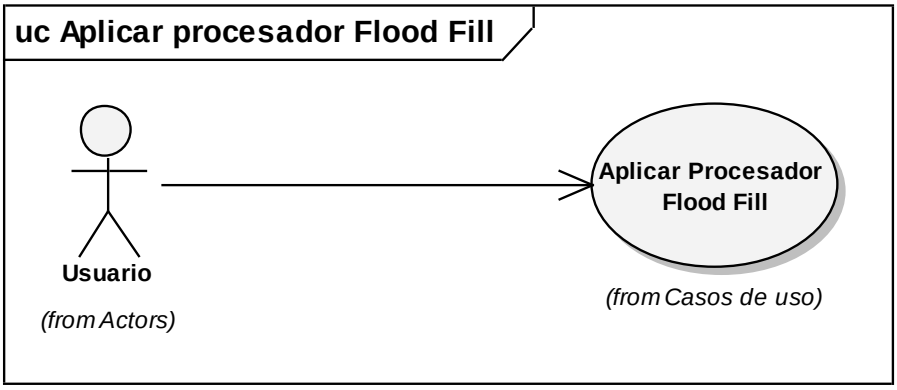


Figura 28. Caso de Uso Aplicar procesador Flood Fill

Caso de Uso: Aplicar Procesador FloodFill		
Propósito:	Aplicar el procesador Flood Fill.	
Actores: Usuario.		
Resumen: El CU se inicia cuando el usuario tiene los datos del grafo y el modelo del mundo. El mismo escoge la opción aplicar procesador FloodFill , aplicándolo al grafo.		
Referencia:	R3, R6, R7	
Precondiciones:	Tener el mesh cargado.	
Acción del actor:		**Respuesta del Sistema:**
1. El usuario escoge la opción “Processors”.		2. El sistema despliega un menú con las siguientes opciones: a) Flood Fill b) Simplify c) FixConnections
3. El usuario escoge la opción FloodFill.		4. El sistema muestra la pestaña con las coordenadas del nodo semilla el valor de la distancia entre los nodos (OffSet).
5. El usuario entra los valores y presiona el botón Aplicar.		6. El sistema ejecuta el algoritmo predeterminado para realizar el procesamiento de los nodos. 7. El Sistema guarda los datos procesados. 8. El sistema muestra los datos del grafo al finalizar la aplicación del procesador.
Puntos de Extensión: R3.1 , R7.1, R7.2		

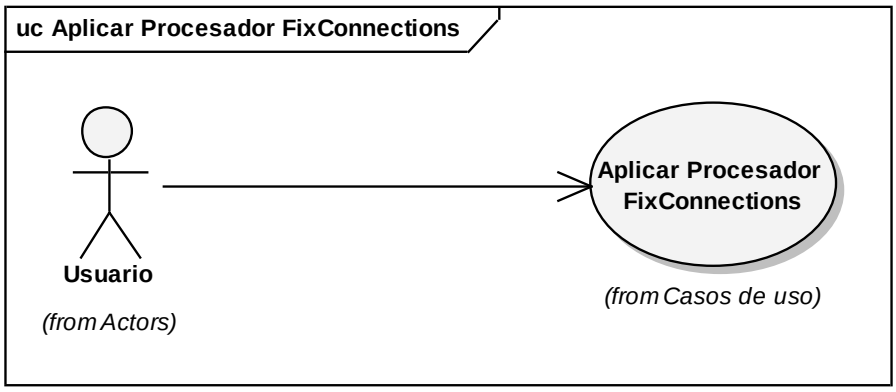


Figura 29. Caso de Uso aplicar procesador FixConnections

Caso de Uso: Aplicar Procesador FixConnections		
Propósito:	Aplicar el procesador FixConnections.	
Actores: Usuario.		
Resumen: El CU se inicia cuando el usuario tiene los datos del grafo y el modelo del mundo. El mismo escoge la opción aplicar procesador FixConnections, aplicándose al grafo.		
Referencia:	R3, R6	
Precondiciones:	Debe existir un grafo. Debe existir un mesh.	
Acción del actor:		**Respuesta del Sistema:**
1. El usuario escoge la opción “Processors”.		2. El sistema despliega un menú con las siguientes opciones: a) Flood Fill b) Simplify c) FixConnections
3. El usuario escoge la opción FixConnections.		4. La aplicación muestra la pestaña con un CheckBox con la opción de eliminar las conexiones antes de aplicar el procesador.
5. El usuario marca la opción de ser necesario y/o presiona el botón “Aplicar”.		6. El sistema ejecuta el algoritmo predeterminado para realizar el procesamiento de los nodos. 7. El Sistema guarda los datos procesados. 8. El sistema muestra el resultado al finalizar la aplicación del procesador.
Puntos de Extensión: R3.3		

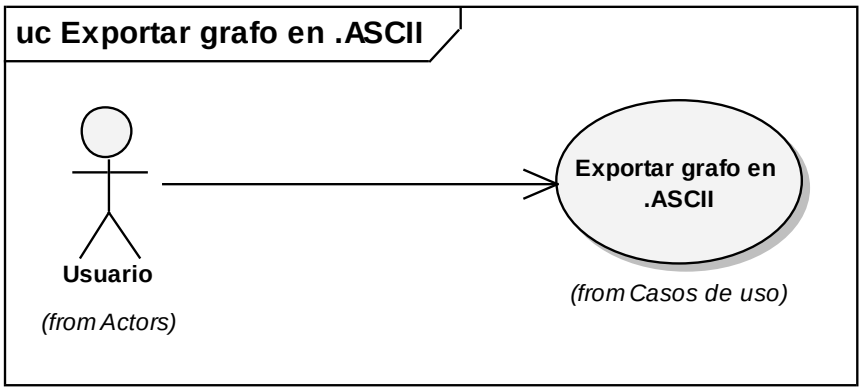


Figura 30. Caso de Uso exportar grafo en ASCII.

Caso de Uso: Exportar grafo en .ASCII		
Propósito:	Exportar el grafo en el formato ASCII para su posterior procesamiento y trabajo.	
Actores: Usuario.		
Resumen: El CU se inicia cuando el usuario escoge la opción “Export”. Se abre una ventana con las opciones de nombre del fichero, ruta del fichero a guardar y el tipo de formato.		
Referencia:	R4	
Precondiciones:	Debe existir un grafo.	
Acción del actor:		**Respuesta del Sistema:**
1. El usuario elige la opción “File” del menú principal, y ahí escoge la opción “Export”.		2. Se visualiza una ventana con los formatos posibles a exportar: a) Grafo ASCII. b) Grafo ASE.
3. El usuario escoge la opción grafo ASCII.		4. La aplicación visualiza una ventana con las opciones nombre del fichero, ruta y tipo de extensión para buscar la ruta a guardar.
5. El usuario escribe un nombre y ruta, y presiona el botón “Save”.		6. La aplicación guarda el fichero en la ruta establecida.
Puntos de Extensión: R4.2		

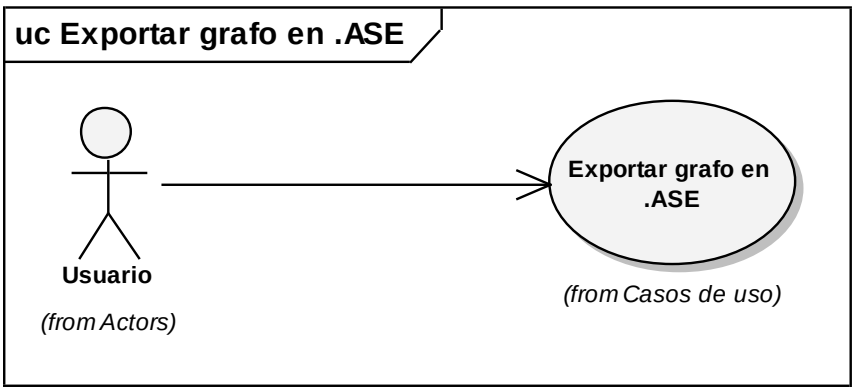


Figura 31. Caso de Uso exportar grafo .ASE

Caso de Uso: Exportar grafo en .ASE		
Propósito:	Exportar el grafo en el formato ASE para su posterior procesamiento y trabajo.	
Actores: Usuario.		
Resumen: El CU se inicia cuando el usuario escoge la opción de "Export". Se abre una ventana con las opciones de nombre del fichero, ruta del fichero a guardar y el tipo de formato.		
Referencia:	R4	
Precondiciones:	Debe existir un grafo.	
Acción del actor:		**Respuesta del Sistema:**
1. El usuario elige la opción "File" del menú principal, y ahí escoge la opción "Export".		2. Se visualiza una ventana con los formatos posibles a exportar: c) Grafo ASCII. d) Grafo ASE.
3. El usuario escoge la opción grafo ASE.		4. La aplicación visualiza una ventana con las opciones de nombre del fichero, ruta y tipo de extensión para buscar la ruta a guardar.
5. El usuario escribe un nombre y ruta, y presiona el botón "Save".		6. La aplicación guarda el fichero en la ruta establecida.
Puntos de Extensión: R4.1		

3.3 Interfaz de Usuario:

La interfaz es la encargada de permitir la interacción del usuario con el software de una forma fácil, obteniéndose resultados mejores y llegando a una solución más viable. La misma debe ser sencilla para que el usuario logre con un mínimo de conocimiento, llegar a la solución deseada.

A continuación se presentan cada una de las ventanas de la herramienta propuesta y se da una descripción de las funciones que realizan las mismas.

3.3.1 Interfaz cargar grafo:

La *figura 31 a,* representa la opción de cargar el grafo. Al ejecutar la aplicación se escoge la opción File, en el menú inicio. Aquí se despliega una ventana con tres opciones, escogiéndose la opción A, Load. Al marcar esta opción se despliegan las opciones de Grafo ASCII y Mesh X, donde se escoge la primera opción (B).

Figura 32 a. Opción cargar grafo.

Al ejecutar la opción cargar Grafo ASCII, se muestra a continuación la siguiente ventana (*figura 31 b*). Esta permite buscar el fichero que se va a cargar en la aplicación. En C se pondría el nombre del fichero a cargar y en E está la extensión, en este caso .ASCII. Además existen las opciones de buscar los ficheros en otra dirección a través de la opción D.

Al encontrar el fichero del grafo se presiona el botón de aceptar y el grafo es cargado por la aplicación.

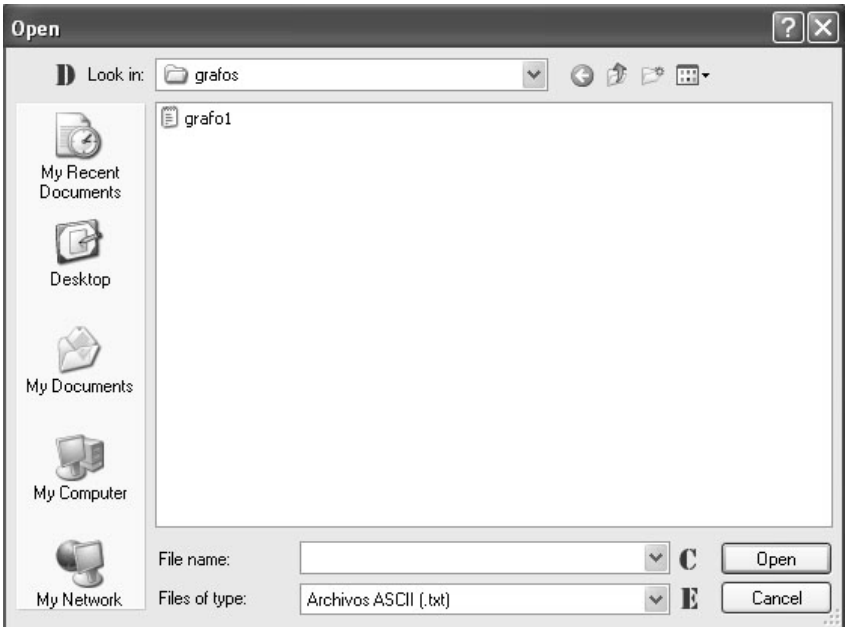


Figura 32 b. Ventana de cargar el grafo.

Al cargar el fichero, la aplicación muestra la siguiente ventana (*figura 31 c*), donde se presentan los datos del grafo cargado. En la región de Datos Generales del Grafo, se muestra la cantidad de nodos que presenta el grafo y el número de conexiones (F).

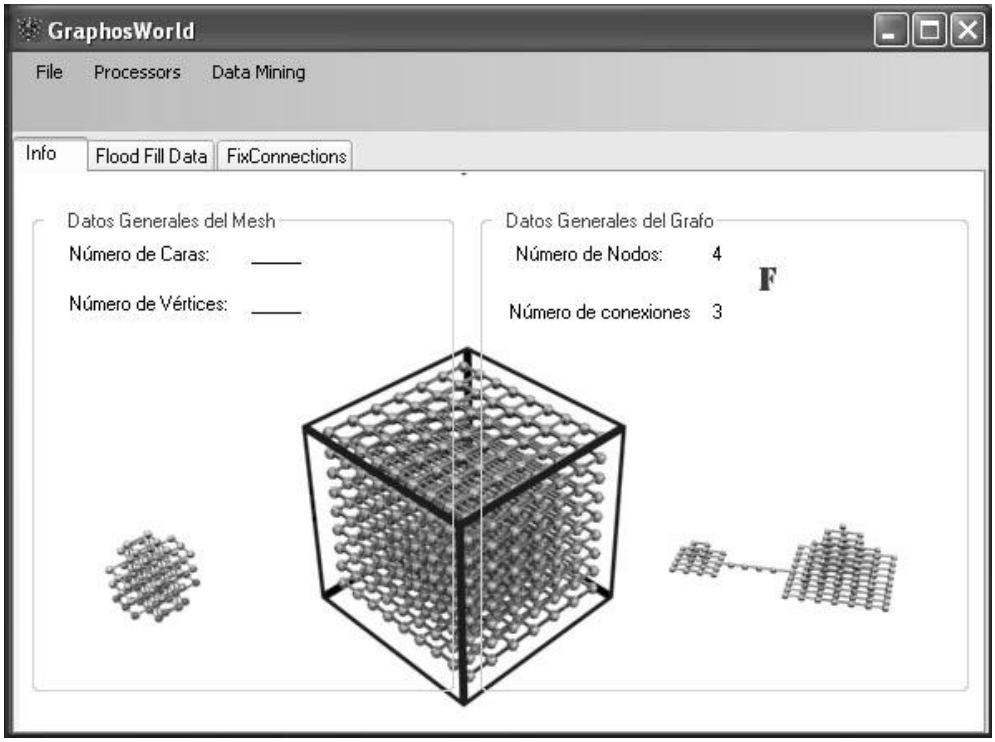


Figura 32 c. Datos del grafo al cargarlo.

3.3.2 Interfaz cargar MESH:

En la *figura 32 a* se representa la ejecución de cargar el mesh de la aplicación. Al ejecutar la aplicación se escoge la opción File, en el menú inicio. Aquí se despliega una ventana con tres opciones, escogiéndose la opción A, Load. Al marcar la opción A se despliega las opciones de Grafo ASCII y Mesh X, donde se escoge la segunda opción (B).

Figura 33 a. Opción cargar Mesh.

Al ejecutar la opción cargar Mesh X se muestra a continuación la siguiente ventana (*figura 32 b*). Esta permite buscar el fichero que se va a cargar en la aplicación. En C se pondría el nombre del fichero a cargar y en E la extensión. Además existen las opciones de buscar los ficheros a través de la opción D.

Al terminar de encontrar el modelo del mundo se presiona el botón de aceptar y es cargado por la aplicación.

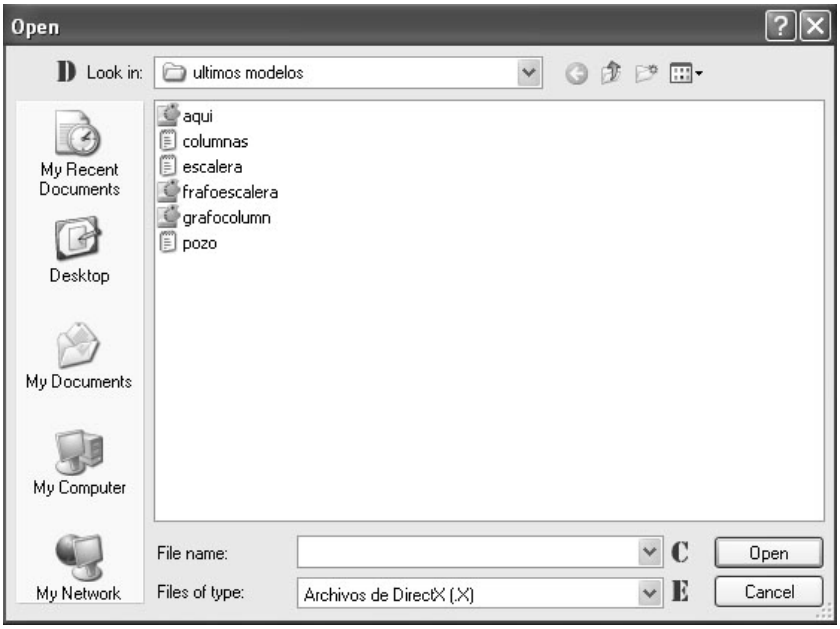


Figura 33 b. Ventana de cargar Mesh.

Al realizar la operación de cargar, la aplicación muestra en la siguiente ventana (*figura 32 c*), los datos generales del mesh. Ahí se muestra el número de caras (F) y el número de vértices (G).

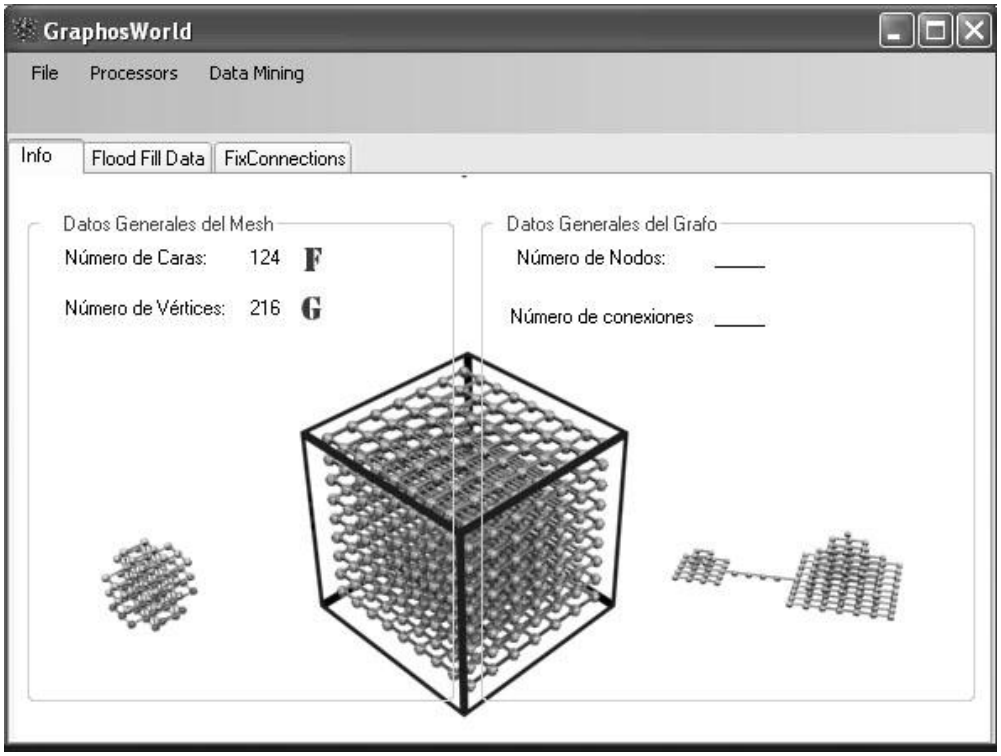


Figura 33 c. Datos generales del Mesh.

3.3.3 Interfaz aplicar procesador Flood Fill.

Para aplicar el procesador Flood Fill al grafo se siguen los siguientes pasos en la aplicación, *figura 33 a*. Se va al menú inicio y se escoge la opción Processors. Al presionarla se despliega una lista con las opciones de Flood Fill, Simplify y FixConnections. Ahí se escoge la opción Flood Fill (A).

Figura 34 a. Opción Flood Fill.

Al ejecutar esta opción se muestra la siguiente ventana, *figura 33 b*. Aquí se pasan las coordenadas del nodo Seed (semilla), que es el nodo a partir del cual se va a aplicar el algoritmo Flood Fill y crear la restante red de nodos. Este presenta las tres coordenadas del espacio X, Y y Z (B). Además existe un Offset, que es la distancia que existirá entre cada nodo (C).

Las medidas de longitud de los valores que se entran en el Offset y en el espacio de las coordenadas, son unidades genéricas.

Al terminar de entrar los parámetros, se presiona Aplicar (D). La operación demorará más o menos tiempo en dependencia de la cantidad de nodos que genere el algoritmo.

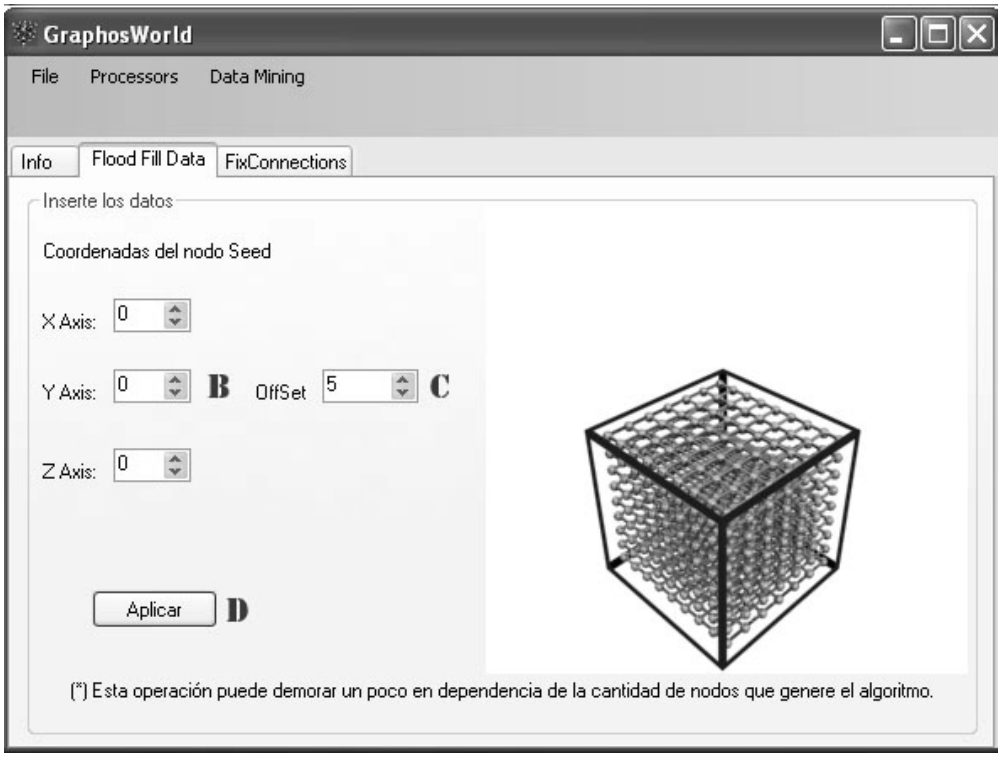


Figura 34 b. Ventana de aplicación del algoritmo Flood Fill.

Al aplicar el algoritmo se obtiene en la siguiente ventana el resultado de la aplicación del mismo, *figura 33 c*. Aquí se representan los Datos Generales del Mesh (B), número de caras y número de vértices); y los Datos Generales del Grafo (D), el número de nodos y el número de conexiones.

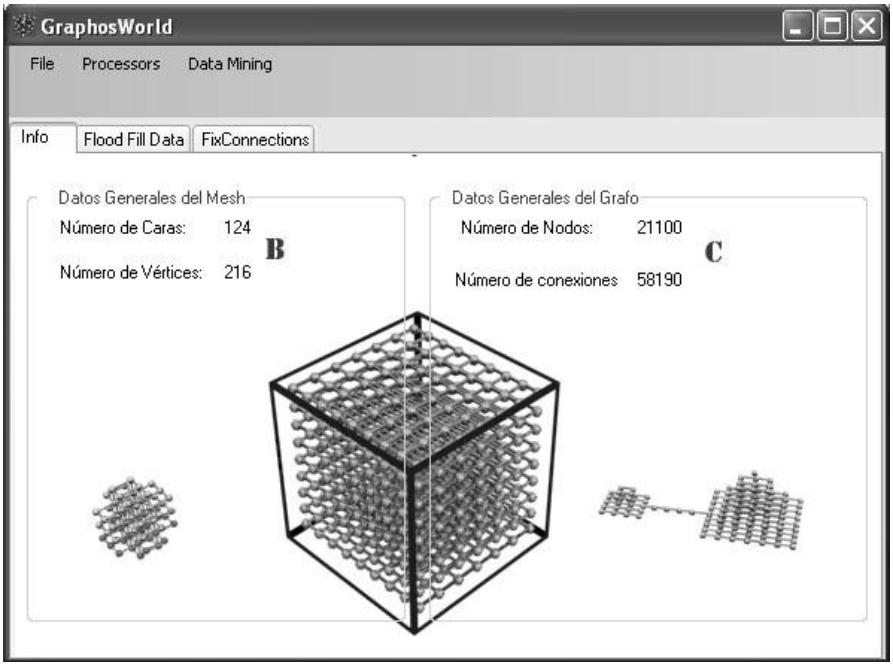


Figura 34 c. Resultados al aplicar el algoritmo Flood Fill.

3.3.4 Interfaz aplicar procesador Simplify

Para aplicar el procesador Simplify al grafo se realiza lo siguiente en la aplicación, *figura 34 a*. Se va al menú inicio y se escoge la opción Processors. Al presionarla se despliega una lista con las opciones de Flood Fill, Simplify y FixConnections. Ahí se escoge la opción Simplify (A).

Figura 35 a. Opción simplify.

Al escoger esta opción, es aplicado el algoritmo al grafo presente en ese momento en la aplicación, tanto el creado por el Flood Fill como el cargado por la aplicación. Al terminar de realizarlo, se presenta en la siguiente ventana, *figura 34 b*, los resultados finales de la aplicación del algoritmo.

En Datos Generales del Grafo (C) se presenta el número final de nodos resultantes después de la aplicación del algoritmo y el número de conexiones, y en Datos Generales del Mesh (B), los datos del modelo del mundo.

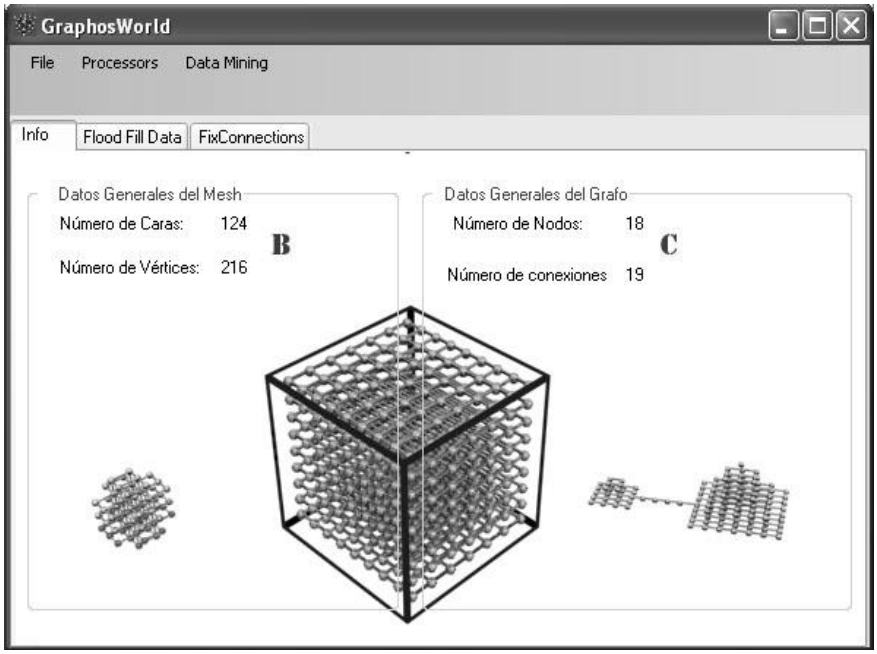


Figura 35 b. Resultados al aplicar el algoritmo Simplify.

3.3.5 Interfaz aplicar algoritmo FixConnections

Para aplicar el procesador FixConnections al grafo se realiza lo siguiente en la herramienta, *figura 35 a*. Se va al menú inicio y se escoge la opción Processors. Al presionarla se despliega una lista con las opciones de Flood Fill, Simplify y FixConnections. Ahí se escoge la opción FixConnections (A).

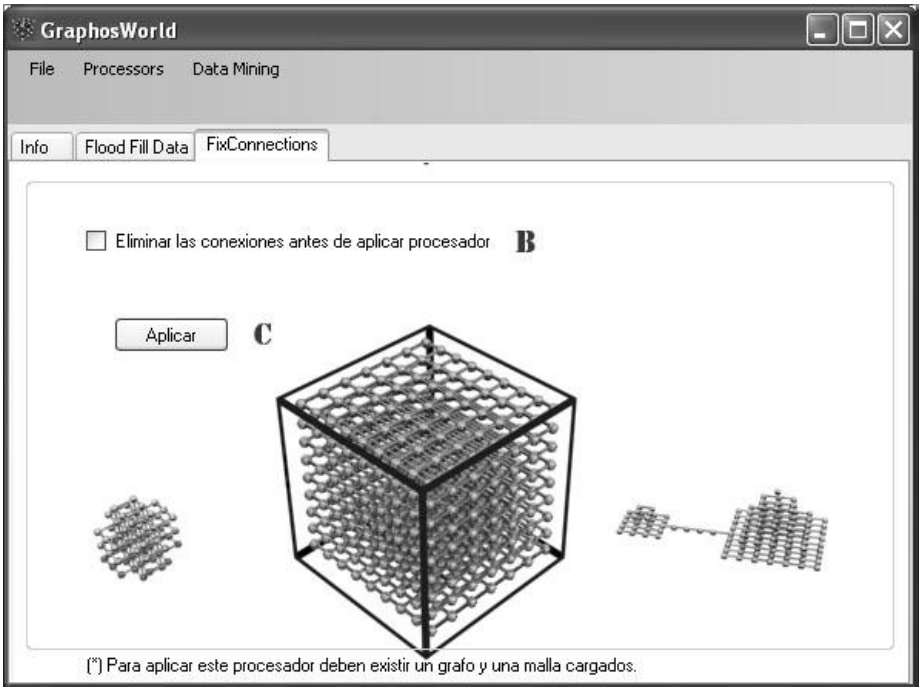


Figura 36 a. Aplicar procesador FixConnections.

Al ejecutar esta opción, se activa la siguiente ventana, *figura 35 b*. Esta presenta solamente una opción que puede ser activada o no, mediante la cual al aplicar el algoritmo este elimina todas las conexiones que existe en el grafo, (B). Al presionar el botón Aplicar (C), el algoritmo es aplicado, y se generan las conexiones entre los nodos.

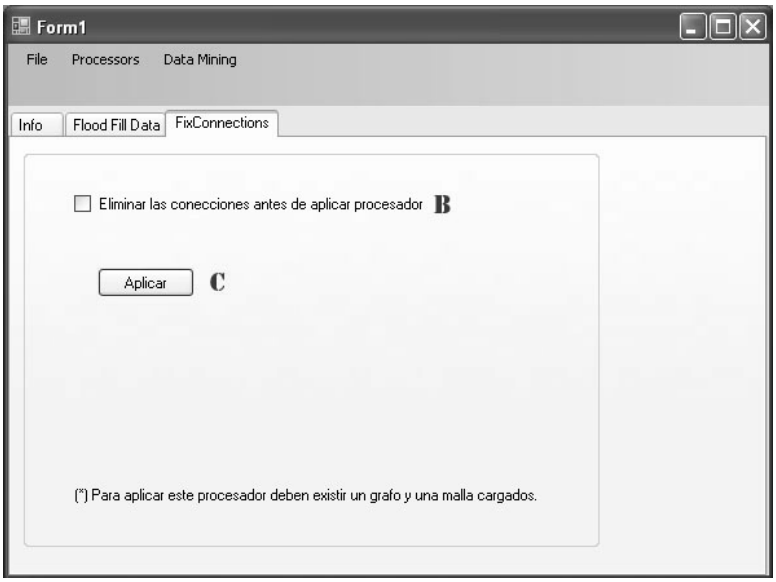


Figura 36 b. Ventana de aplicación del algoritmo FixConnections.

3.3.6 Interfaz exportar grafo

Para realizar la exportación del grafo se realiza lo mostrado en la figura 36 a. Se busca en el menú inicio la opción File. Al presionarla se despliega una lista con diferentes opciones. Se escoge la opción exportar (A), que a su vez despliega otra lista con las opciones de los formatos a exportar, ASCII o ASE, (B).

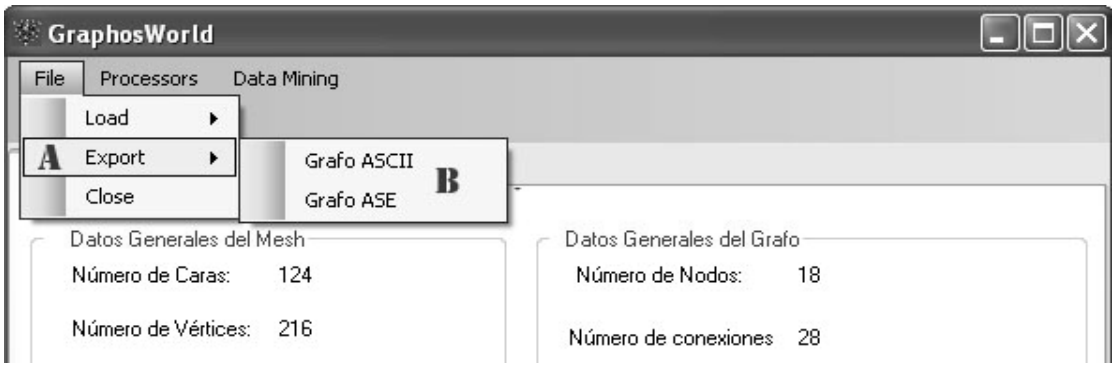


Figura 37 a. Exportar grafo.

Al escoger el formato del grafo que se va a exportar, se muestra la siguiente ventana, *figura 36 b*. En esta se pasa el nombre del fichero (C) y aparece la extensión escogida para exportar (D), así como existe la posibilidad de buscar la ruta donde se va a exportar en fichero del grafo (E), a través de diferentes vías.

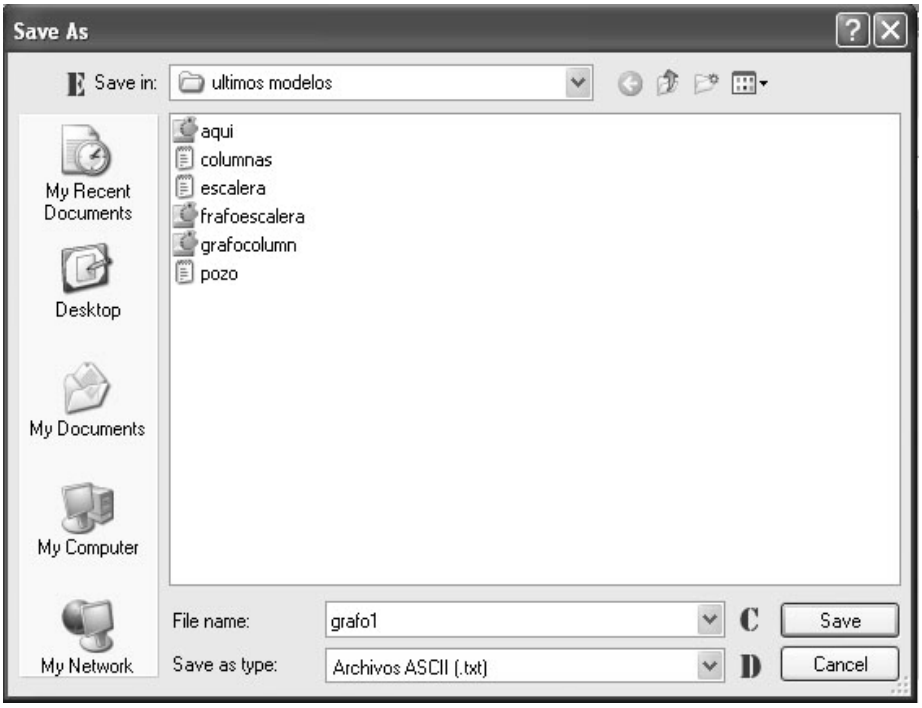


Figura 37 b. Exportar grafo.

3.4 Modelo del dominio:

Con este modelo se identifican las clases que se utilizaran en la herramienta propuesta de forma visual. Además ilustran al usuario los principales conceptos que se manejan en el dominio del sistema en desarrollo.

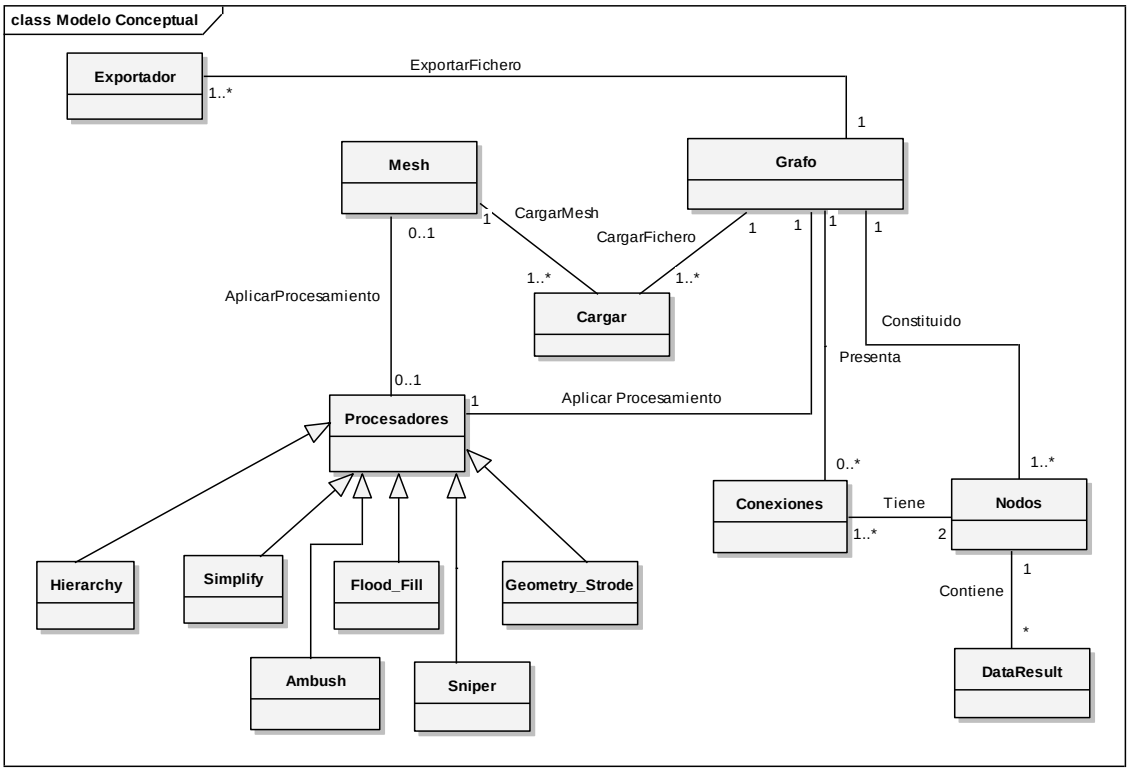


Figura 38. Modelo del dominio.

3.5 Arquitectura del Sistema:

A través de la siguiente representación se muestra las clases principales que conforman el sistema. A esto se le pasará a llamar “Framework”. Este representa una arquitectura de software que modela las relaciones generales de las entidades del dominio y provee una estructura.

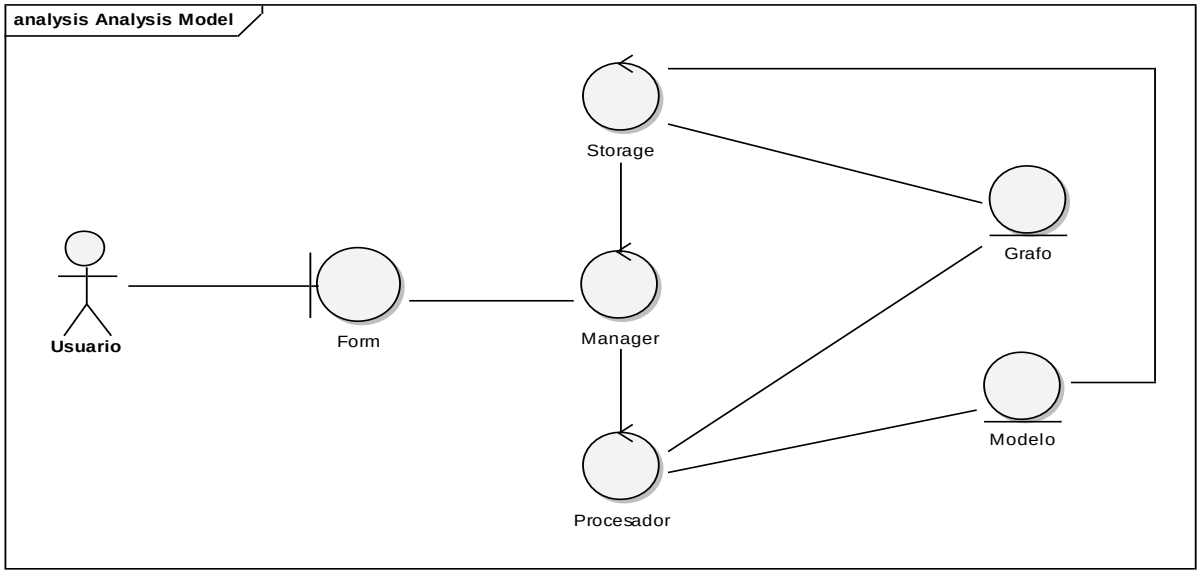


Figura 39. Modelo de Análisis.

3.6 Funcionamiento del Framework:

El usuario realiza una serie de peticiones a través de la interfaz que generan el siguiente flujo de acciones:

3.6.1 Petición inicializar.

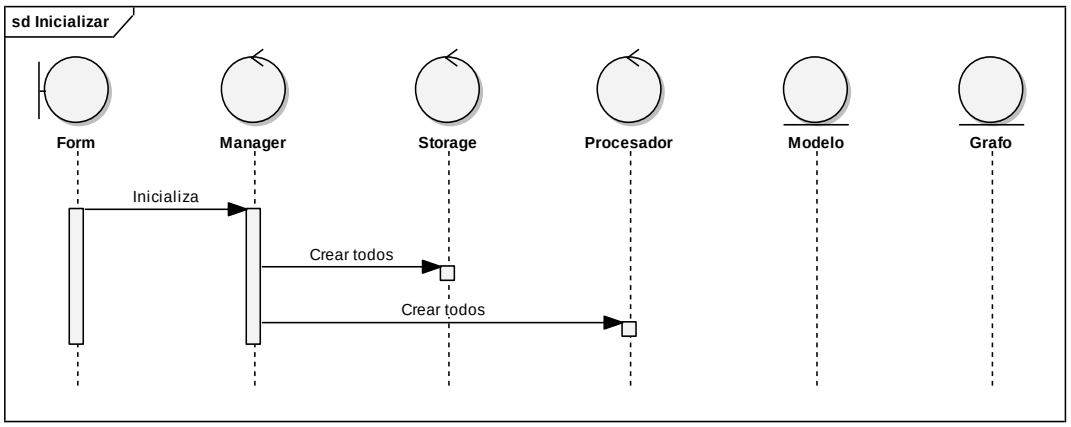


Figura 40. Petición inicializar.

El manager recibe el mensaje de Inicializar, al ejecutarse la herramienta. Al recibir este mensaje se crean todos los storages y los procesadores.

Los storages contienen los datos necesarios para cargar el grafo y el modelo del mundo dentro de la herramienta. El manager crea los storage necesarios, el storageDX y el storageASCII.

El procesador contiene los procesadores que se van a aplicar al grafo. El manager carga todos los algoritmos para el procesamiento de los datos del grafo: el Flood Fill, el Simplify y el FixConnections.

3.6.2 Petición Cargar modelo:

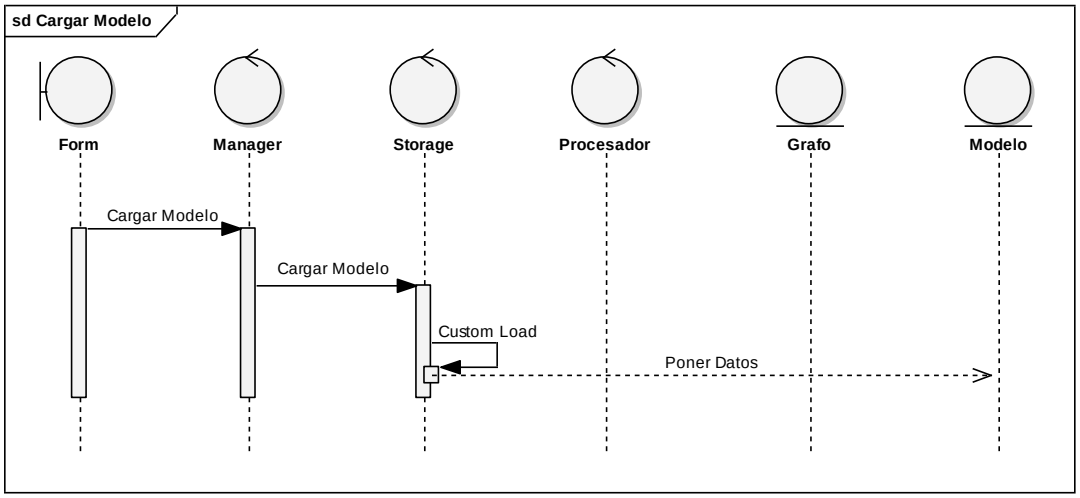


Figura 41. Petición cargar modelo.

Al enviarle al manager la petición de cargar modelo, este verifica primeramente si el modelo presenta alguna de las extensiones posibles a cargar por la herramienta, a través de los storages cargados.

Si existe algún storage que pueda cargar este formato entonces se le asigna la petición. El Storage busca los datos que necesita del fichero y los inserta en el modelo.

3.6.3 Petición Cargar grafo:

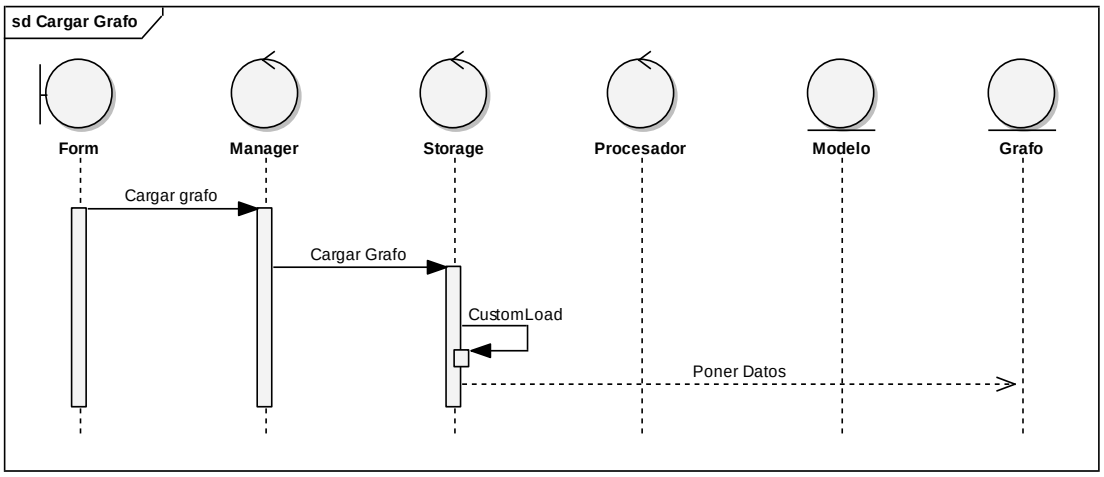


Figura 42. Petición cargar grafo.

Al enviar al manager la petición de cargar grafo, este verifica si el grafo presenta alguna de las extensiones posibles a cargar por la herramienta, a través de los Storage cargados.

Si existe algún Storage que pueda cargar este formato entonces se le asigna la petición. El Storage obtiene los datos que necesita del fichero y los inserta en el grafo.

3.6.4 Petición Procesar:

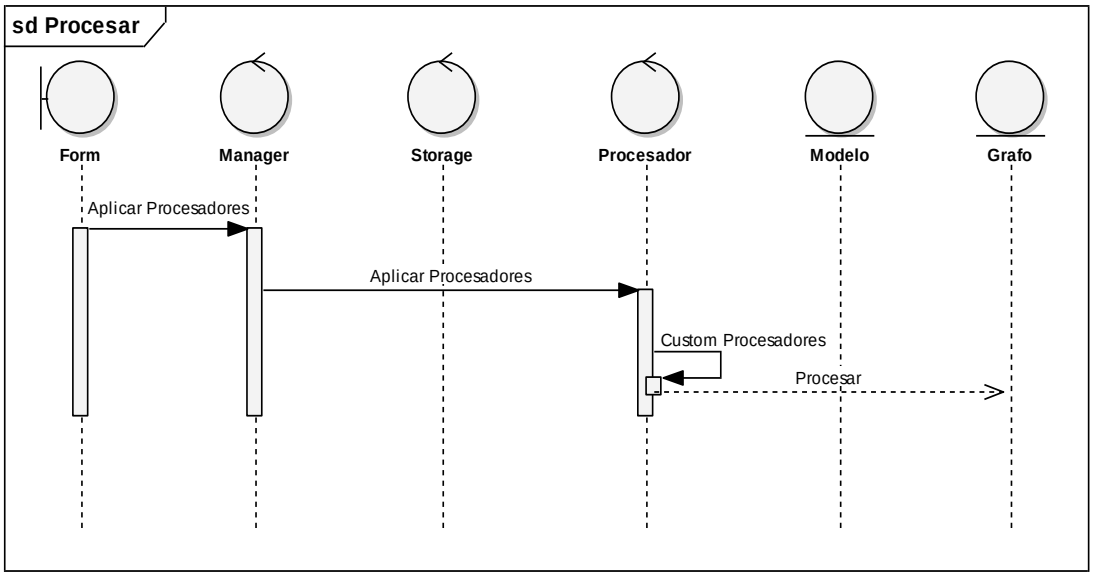


Figura 43. Petición procesar.

Al recibir el Manager una solicitud de Procesar un grafo, este lo redirecciona al procesador solicitado, el cual se ejecutara con la configuración seleccionada por el usuario.

3.6.5 Petición Exportar:

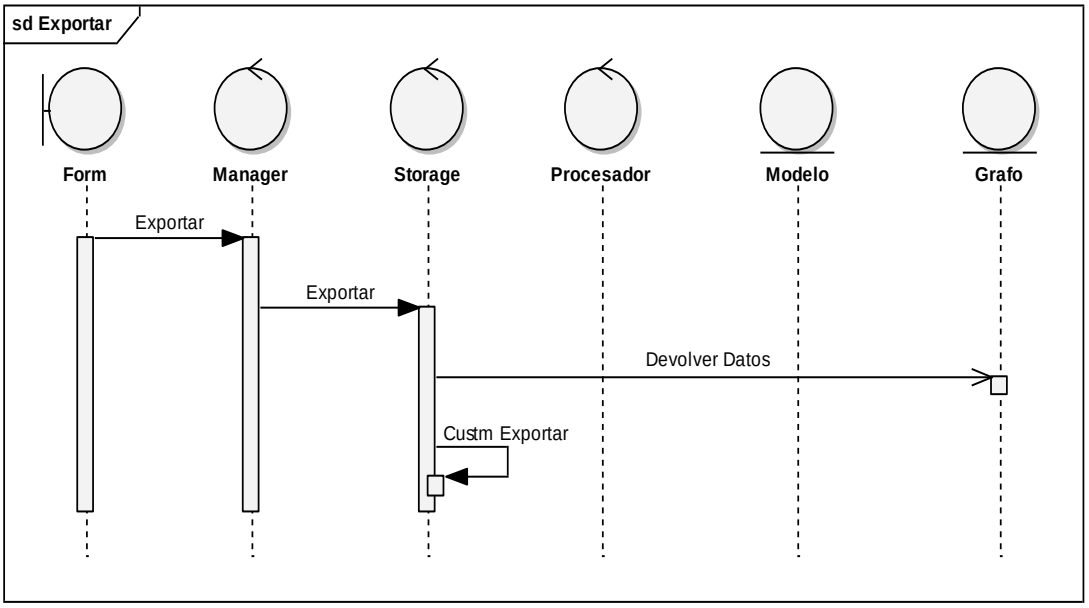


Figura 44. Petición exportar.

Al enviarle al manager el mensaje de exportar, este busca el Storage que puede exportarlo en el formato especificado y le asigna la tarea. Este Storage obtiene los datos del grafo y los guarda en el formato especificado.

Capítulo 4.Diseño e Implementación

4.1 Diseño por paquetes

4.1.1 Paquete: Manager

Este paquete recoge todas las clases que componen al Manager. Este es el más importante de todos porque a través de él se puede gestionar la carga del grafo y el modelo del mundo, aplicar los procesadores y exportar el grafo.

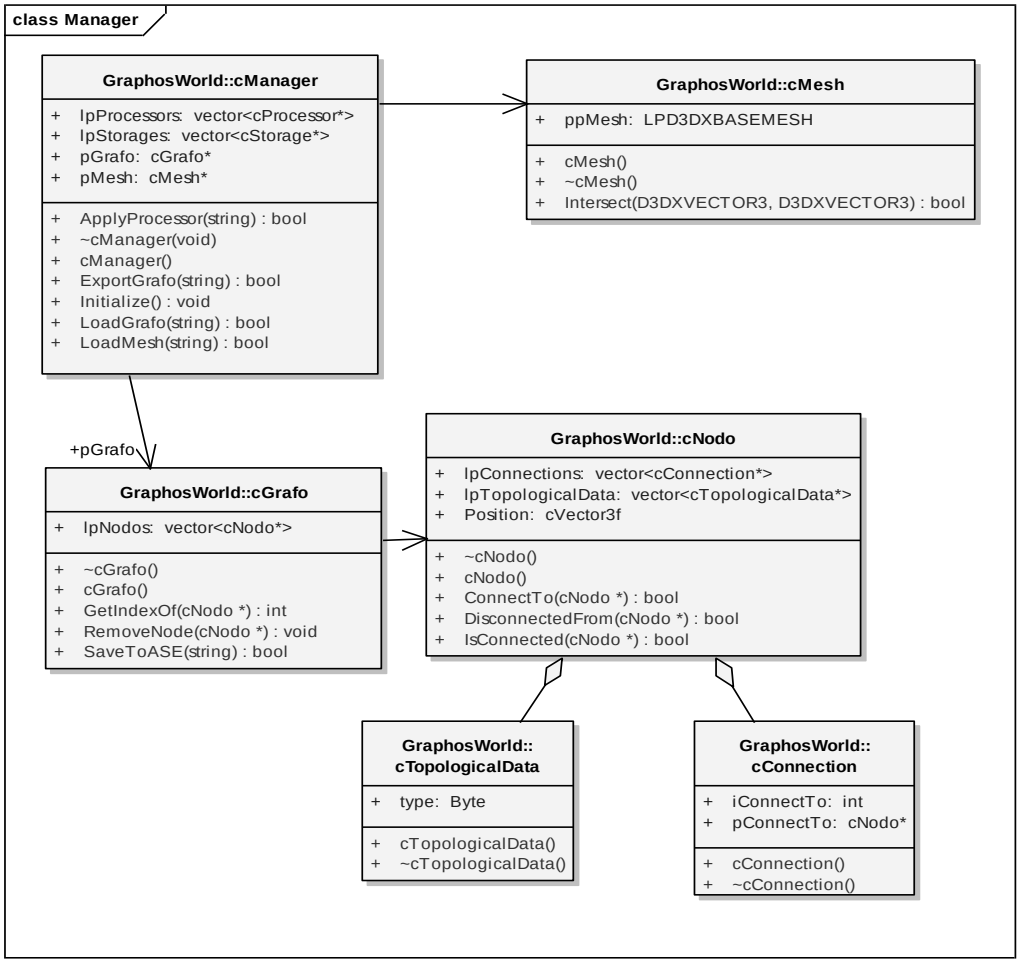


Figura 45. Paquete Manager.

La clase cManager es la clase principal y es la encargada de realizar todas las peticiones que se van a realizar dentro de la herramienta, así como inicializar los parámetros de la aplicación. La clase cMesh carga el modelo del mundo en la herramienta.La clase cGrafo permite que el grafo sea exportado en el formato .ASE por la aplicación.

La clase cNodo analiza las conexiones que presenta el nodo, así como los datos topológicos. Esta clase tiene dos agregaciones, una es la clase cTopologicalData, la cual guarda los datos topológicos de los nodos, y una segunda clase denominada cConnection que guarda todas las conexiones entre los nodos. La clase cNodo mediante estas agregaciones tendrá una lista de los datos topológicos y una lista de conexiones.

4.1.2 Paquete: Storage

El paquete Storage agrupa los métodos que dan soporte para cargar un modelo y un grafo así como exportar este último cuando se termina la aplicación de los diferentes algoritmos.

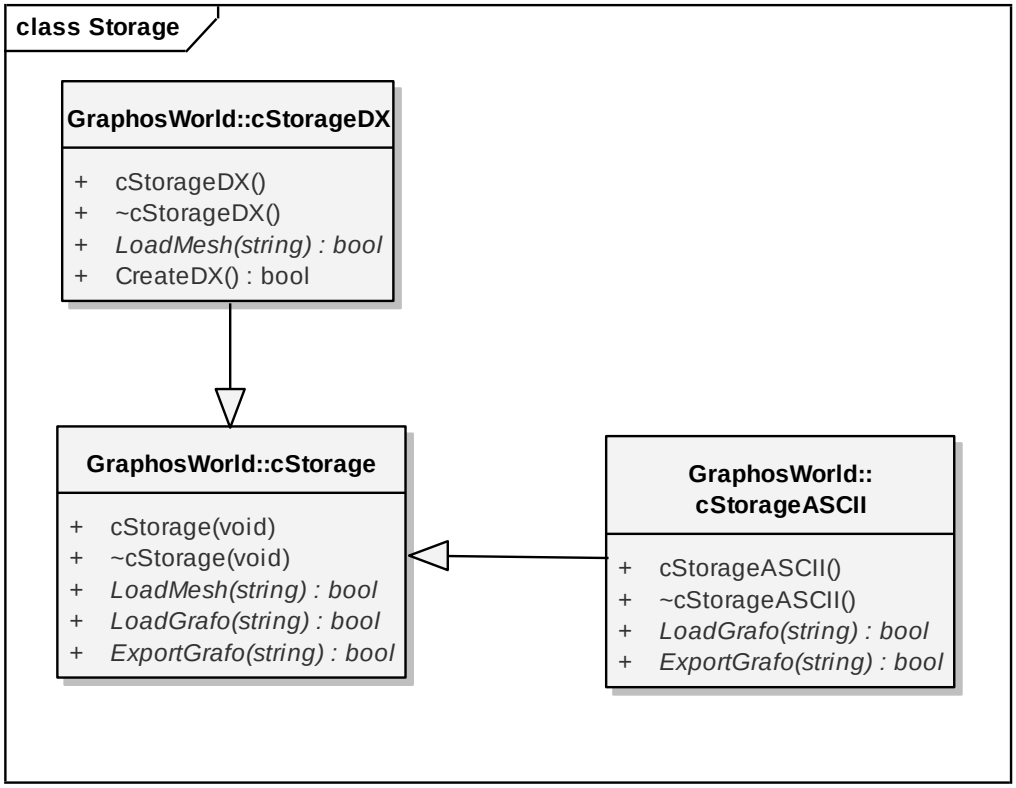


Figura 46. Paquete Storage.

Las clases cStorageASCII y cStorageDX heredan de la clase cStorage los métodos de esta. La clase cStorageASCII permite exportar y cargar un grafo en formato ASCII, mientras que por su parte la clase cStorageDX permite cargar el modelo del entorno virtual.

4.1.3 Paquete: Procesadores

Este paquete agrupa los métodos necesarios para poder aplicar los algoritmos de procesamiento Flood Fill, Simplify y FixConnections, al grafo o modelo cargados.

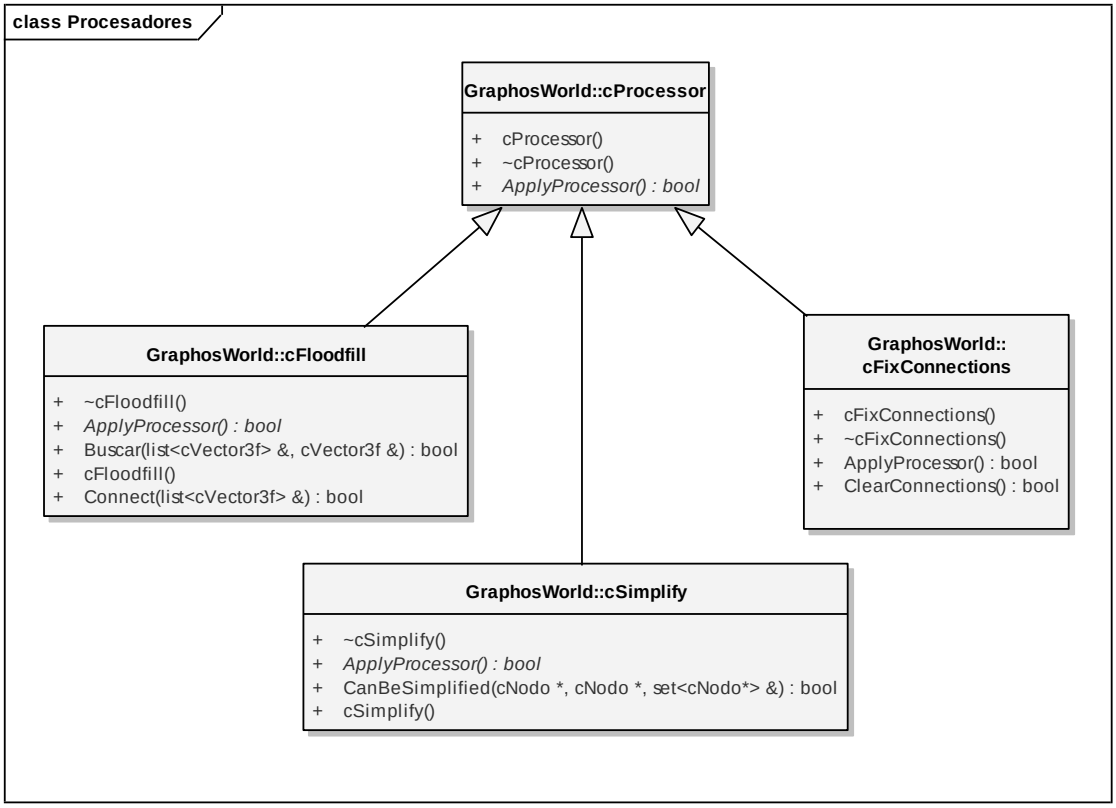


Figura 47. Paquete procesadores.

Los métodos de la clase cProcessor son virtuales. Las clases cSimplify, cFloodFill y FixConnections heredan de la clase cProcessor. La clase cFloodFill aplica el algoritmo de procesamiento del mismo nombre cuando se carga el modelo del mundo. Las clases cSimplify y cFixConnections aplican los algoritmos Simplify y FixConnections respectivamente al grafo existente.

4.2 Descripción de las clases:

Se incluye la descripción de las clases que contiene la aplicación, con sus métodos y atributos, para lograr una buena comprensión de las mismas.

Nombre: cManager	
Tipo de Clase: Controladora	
Descripción: Es la clase principal y la encargada de realizar todas las peticiones de la herramienta, así como inicializar los valores de esta.	
Atributo	**Tipo**
lpStorages	vector<cStorage*>
lpProcessors	vector<cProcessor*>
*pGrafo	cGrafo
*pMesh	cMesh
Métodos:	**Descripción:**
cManager()	Es el constructor de la clase.
~cManager()	Es el destructor de la clase.
Initialize()	Inicializa todos los parámetros de la aplicación. Crea los storage que existan e inserta estos en las listas de los punteros de Storage. Además crea lo procesadores y los inserta también en una lista de procesadores. Este método es llamado en el constructor de la clase.
LoadMesh(string name)	Carga el modelo del mundo cuya dirección es especificada en la variable name.
LoadGrafo(string name)	Importa el grafo cuya dirección es especificada en la variable name.
ExportGrafo(string _Name)	Exporta el grafo usando el storage especificado por la variable _Name. Retorna verdadero si es ejecutado satisfactoriamente.
ApplyProcessor(string _Name)	Ejecuta el procesador especificado por la variable _Name. Retorna verdadero si es ejecutado satisfactoriamente.

Nombre: cMesh	
Tipo de Clase: Entidad	
Descripción: Es la clase encargada de cargar el modelo del mundo.	
Atributo	**Tipo**
ppMesh	LPD3DXMESH
Métodos:	**Descripción:**
cMesh()	Es el constructor de la clase.
~cMesh()	Es el destructor de la clase.
Intersect(D3DXVECTOR3 Pos, D3DXVECTOR3 RayDir)	Crea un rayo entre los dos vectores pasados y determina si existe entre los mismos alguna cara del mesh cargado. Retorna verdadero si esto ocurre.

Nombre: cGrafo	
Tipo de Clase: Entidad	
Descripción:	
Atributo	**Tipo**
lpNodos	vector<cNodo*>
Métodos:	**Descripción:**
cGrafo()	Es el constructor de la clase.
~cGrafo()	Es el destructor de la clase.
SaveToASE(string name)	Exporta el grafo al formato .ASE, y se guarda en una dirección especificada en la variable name.
GetIndexOf(cNodo *node)	Retorna el índice en la lista de nodos del nodo al que se le aplica el método.
RemoveNode(cNodo *node)	Elimina de la lista de nodos el nodo al cual se le aplica el método.

Nombre: cNodo	
Tipo de Clase: Entidad	
Descripción:	
Atributo	**Tipo**
Position	vector<cNodo*>
lpTopologicalData	vector<cTopologicalData*>
lpConnections	vector<cConnection*>
Métodos:	**Descripción:**
cNodo()	Es el constructor de la clase.
~cNodo()	Es el destructor de la clase.
IsConnected(cNodo *node)	Devuelve verdadero si el nodo al que se le aplica el método esta conectado al nodo especificado con el puntero node.
DisconnectFrom (cNodo *node)	Desconecta el nodo al cual se le aplica el método del nodo especificado con el puntero node. Retorna verdadero si ejecuta satisfactoriamente la acción.
ConnectTo (cNodo *node)	Desconecta el nodo al cual se le aplica el método del nodo especificado con el puntero node. Retorna verdadero si ejecuta satisfactoriamente la acción.

Nombre: cTopologicalData	
Tipo de Clase: Entidad	
Descripción: Agregación de la clase cNodo. Guarda los datos topológicos del nodo.	
Atributo	**Tipo**
Type	Byte
Métodos:	**Descripción:**
cTopologicalData()	Es el constructor de la clase.
~cTopologicalData()	Es el destructor de la clase.

Nombre: cConnection	
Tipo de Clase: Entidad	
Descripción: Agregación de la clase cNodo. Guarda las conexiones del nodo.	
Atributo	**Tipo**
iConnectTo	int
pConnectTo	cNodo*
Métodos:	**Descripción:**
cConnection()	Es el constructor de la clase.
~ cConnection()	Es el destructor de la clase.

Nombre: cStorage	
Tipo de Clase: Controladora	
Descripción: Agrupa los diferentes métodos que permiten cargar un modelo y un grafo así como exportar el grafo cuando se termine el trabajo con la aplicación.	
Atributo	**Tipo**
Métodos:	**Descripción:**
cStorage(void)	Es el constructor de la clase.
~cStorage(void)	Es el destructor de la clase.
LoadMesh(string name)	Carga el modelo del mundo, cuya dirección es especificada en la variable name.
LoadGrafo(string name)	Carga el grafo, cuya dirección es especificada en la variable name.
ExportGrafo(string name)	Permite que un grafo sea exportado utilizando el storage seleccionado por el usuario.

Nombre: cStorageASCII	
Tipo de Clase: Entidad	
Descripción: Hereda de la clase cStorage los métodos de ésta. Permite que el grafo sea cargado y exportado en formato ASCII.	
Atributo	**Tipo**
Métodos:	**Descripción:**
cStorageASCII()	Es el constructor de la clase.
~cStorageASCII()	Es el destructor de la clase.
LoadGrafo(string name)	Carga el fichero del grafo cuya dirección es especificada en la variable name.
ExportGrafo(string name)	Permite que el grafo sea exportado usando el storage seleccionado por el usuario.

Nombre: cStorageDX	
Tipo de Clase: Entidad	
Descripción: Hereda de la clase cStorage. Tiene los métodos para realizar la carga del grafo.	
Atributo	**Tipo**
Métodos:	**Descripción:**
cStorageDX()	Es el constructor de la clase.
~cStorageDX()	Es el destructor de la clase.
LoadMesh(string name)	Carga el modelo del mundo cuya dirección es especificada en la variable name.
CreateDX()	Inicializa DirectX y crea el dispositivo del mismo. Retorna falso si no se inicializa satisfactoriamente. Esto es necesario para cargar el modelo del modelo del mundo.

Nombre: cProcesadores	
Tipo de Clase: Controladora	
Descripción: Agrupa los métodos necesarios para poder aplicar los procesadores que estén implementados.	
Atributo	**Tipo**
Métodos:	**Descripción:**
cProcessor()	Es el constructor de la clase.
~cProcessor()	Es el destructor de la clase.
ApplyProcessor()	Este es un método virtual que permitirá aplicar el procesador escogido por el usuario.

Nombre: cFloodFill	
Tipo de Clase: Entidad	
Descripción: Permite que el procesador Flood Fill sea aplicado dentro del mesh cargado.	
Atributo	**Tipo**
vSeed	cVector3f
OffSet	float
Métodos:	**Descripción:**
cFloodFill ()	Es el constructor de la clase.
~ cFloodFill()	Es el destructor de la clase.
ApplyProcessor()	Este método genera un grafo dentro de la geometría del mundo cargado, manteniendo una separación entre los nodos vecinos especificada en el atributo OffSet de la clase.
Buscar(list<cVector3f> &lista, ,cVector3f &v)	Analiza si en una lista de vectores se encuentra el vector pasado. Si el vector existe en la lista se devuelve verdadero, sino falso.
Connect(list<cVector3f> &listado)	Este método conecta los nodos de la lista de nodos del grafo mientras no exista una cara entre los mismos y la distancia entre estos no sea mayor que el OffSet multiplicado por 1,4.

Nombre: cSimplify	
Tipo de Clase: Entidad	
Descripción: Permite que el algoritmo Simplify sea aplicado al grafo.	
Atributo	**Tipo**
Métodos:	**Descripción:**
cSimplify()	Es el constructor de la clase.
~ cSimplify()	Es el destructor de la clase.
ApplyProcessor()	Aplica el procesador Simplify al grafo.
CanBeSimplified(cNodo *node1, cNodo *node2, set <cNodo*>&)	Devuelve verdadero si se puede eliminar nodo especificado por el puntero node2.

Nombre: cFixConnections	
Tipo de Clase: Entidad	
Descripción: Permite que el algoritmo FixConnections sea aplicado al grafo.	
Atributo	**Tipo**
Métodos:	**Descripción:**
c FixConnections()	Es el constructor de la clase.
~ c FixConnections()	Es el destructor de la clase.
ApplyProcessor()	Aplica el procesador FixConnections al grafo.
ClearConnections()	Elimina todas las conexiones existentes entre los nodos del grafo al cual es aplicado.

4.3 Diagramas de Clases General

4.3.1 Conexiones entre paquetes:

El diagrama de conexiones entre paquetes permite que se aprecie la interacción que existe entre los paquetes principales así como también que las partes más importantes sean mostradas.

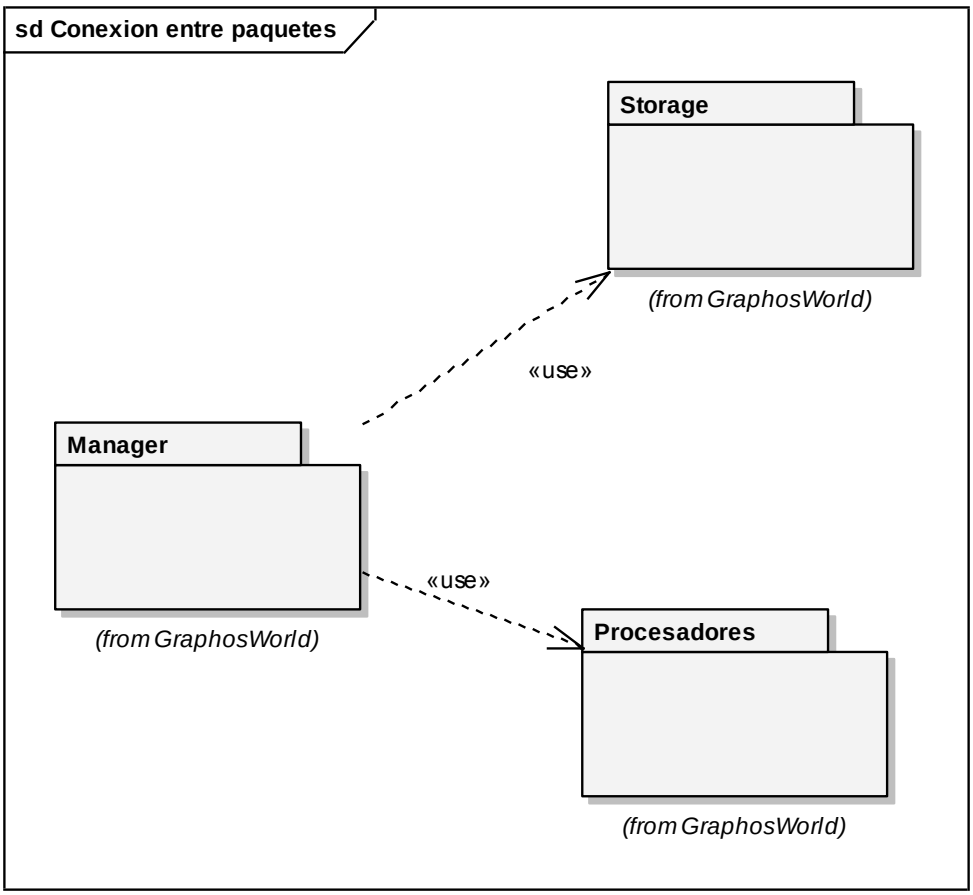


Figura 48. Diagrama de conexión entre paquetes.

Como se aprecia en la figura, la interfaz Form Interface, tiene una conexión con el paquete Manager, debido a que esta última presenta los métodos y clases que permiten que se pueda cargar el grafo y el modelo del mundo, aplicar los procesadores y exportar el grafo.

Además este paquete interacciona directamente con la interfaz gráfica de la aplicación. A su vez el paquete Manager tendrá dos conexiones de uso con los paquetes Procesadores y Storage.

La conexión que se establece entre el paquete Manager y el paquete Storage es debido a que la clase Storage tendrá los métodos necesarios para cargar el modelo del mundo y el grafo así como exportarlo.

La conexión que se establece del Manager al paquete Procesadores es debido a que en este último están presentes las clases y métodos que permitirán que el manager pueda aplicar los procesadores Flood Fill y Simplify desde la forma interfaz.

4.3.2 Diagrama de clases general:

A continuación se muestra el diagrama de clases general, con cada una de las clases que lo componen y la interacción que existe entre cada una de ellas así como la forma en que se establecen las conexiones entre cada una de las mismas.

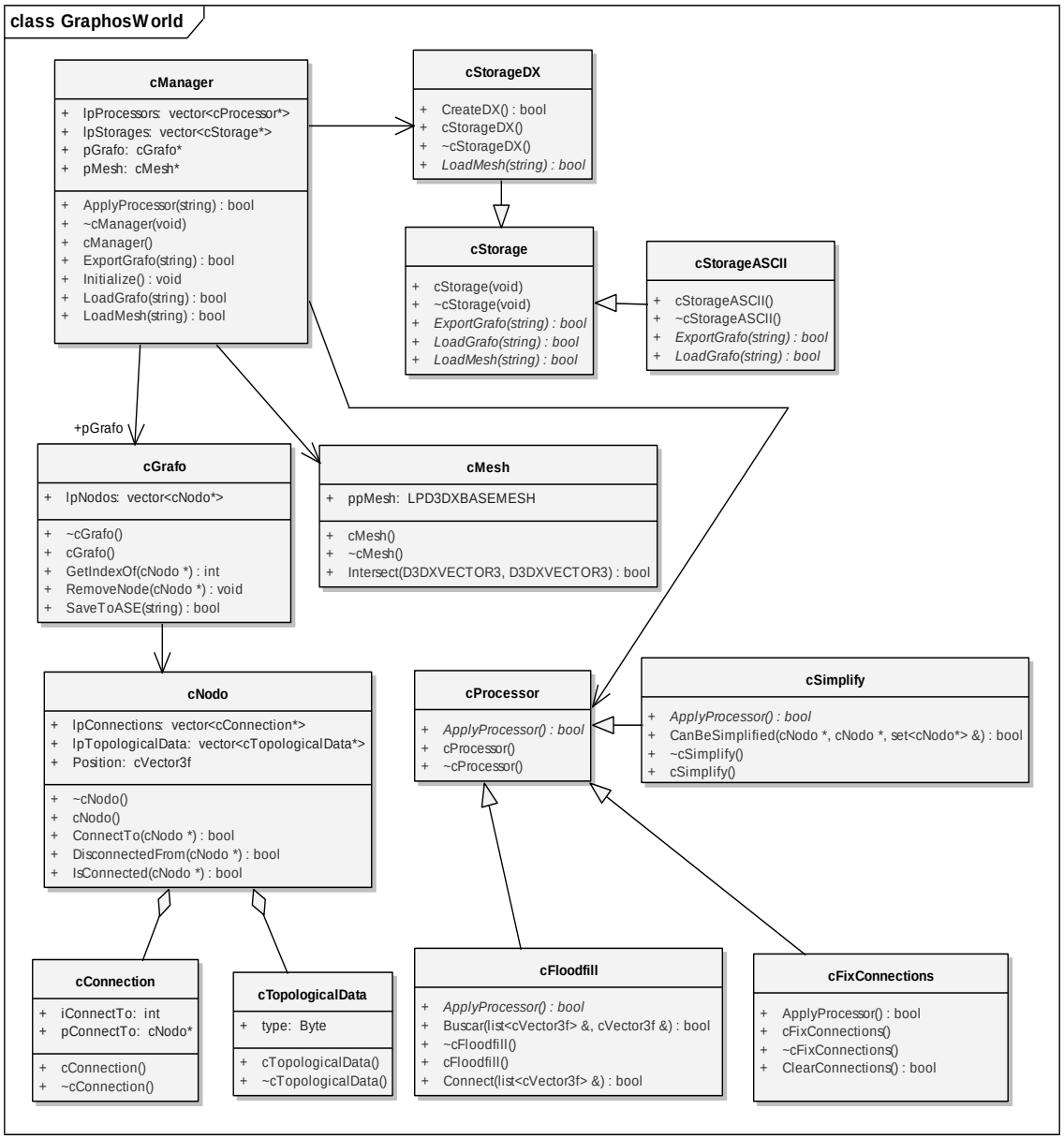


Figura 49. Diagrama de clases general

4.4 Diseño por Casos de Uso

Los diagramas de secuencia de diseño se usan para dar una idea clara de cómo interactuarán las diferentes clases entre sí para poder realizar los Casos de Uso (CU).

4.4.1 CU Cargar grafo de .ASCII

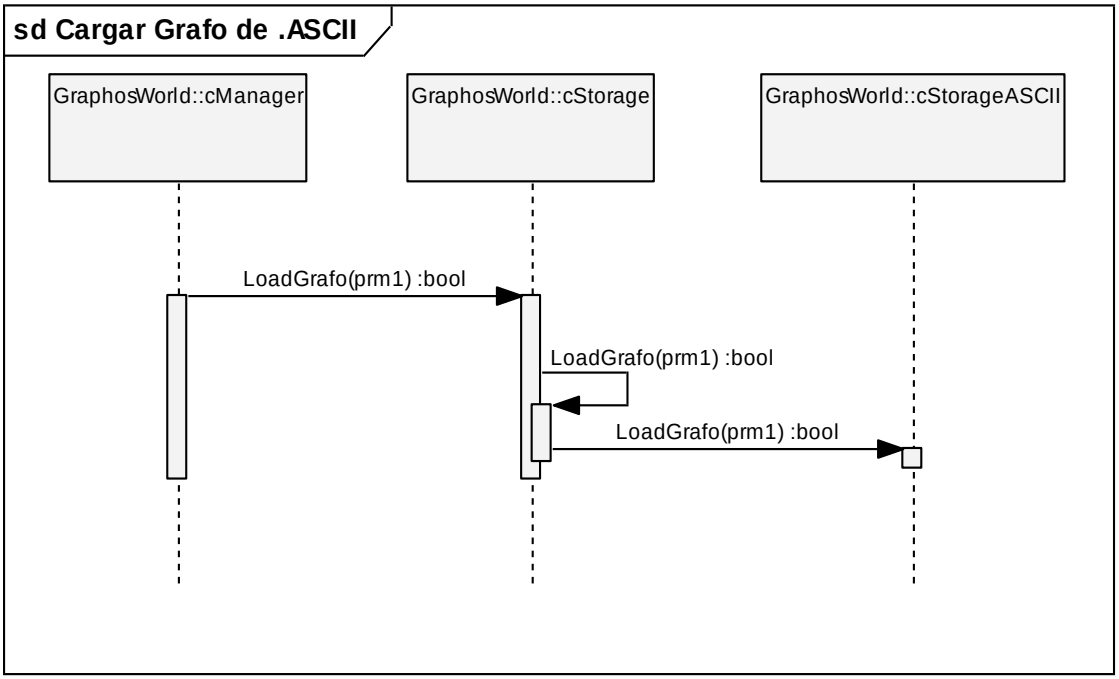


Figura 50. Diagrama de Secuencia del CU cargar grafo de .ASCII

La petición se realiza con el método *LoadGrafo* mediante la cual el Manager realiza la petición de cargar el grafo a la clase cStorage. La clase cStorage crea el storage para la extensión .ASE. Al crearlo, realiza la petición de *LoadGrafo* a la clase cStorageASCII, la cual carga el grafo.

4.4.2 CU Cargar MESH .X

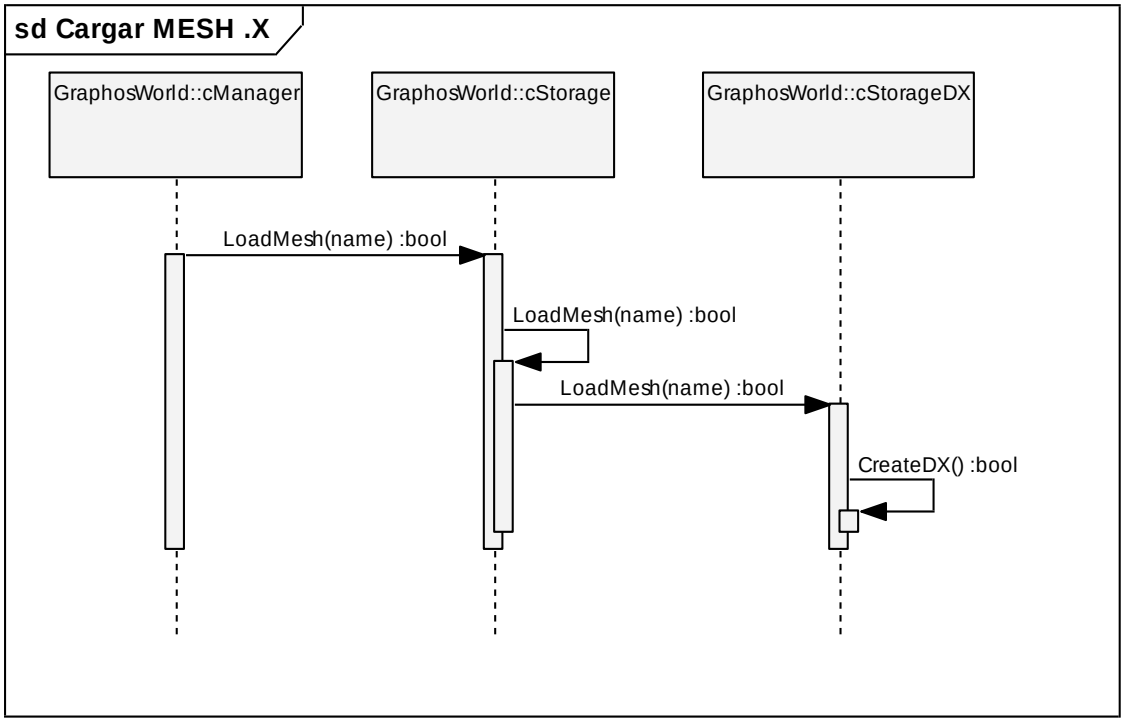


Figura 51. Diagrama de secuencia del CU cargar MESH .X

La clase cManager comienza haciendo la petición a la clase cStorage con el método *LoadMesh* para cargar el modelo del mundo. La clase cStorage por su parte crea el Storage para cargar el mesh .X.

4.4.3 CU Aplicar procesador Simplify

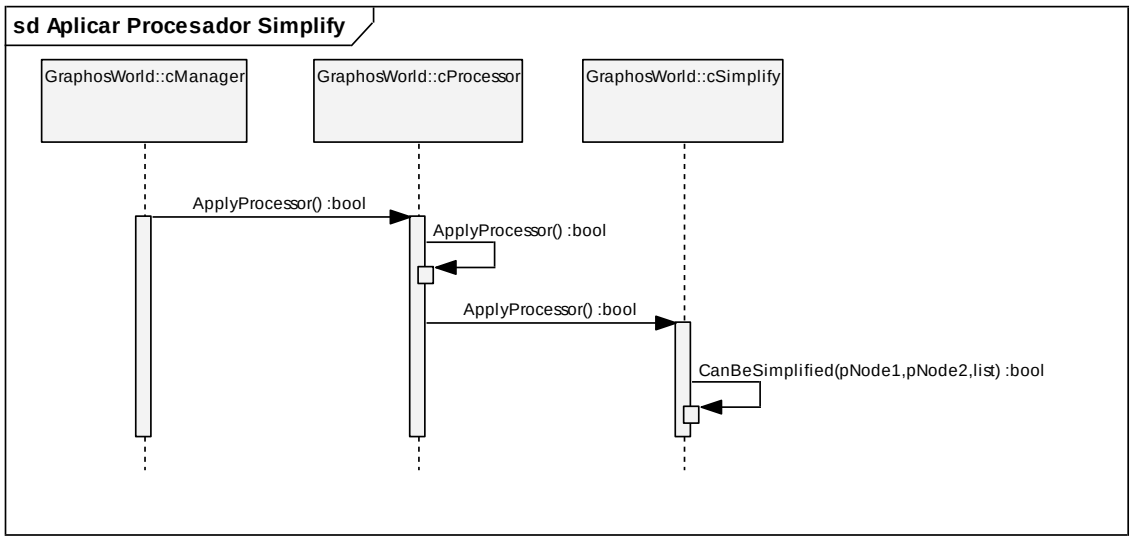


Figura 52. Diagrama de Secuencia del CU aplicar procesador Simplify

Comienza cuando la clase cManager realiza una petición a la clase cProcessor, mediante el método *ApplyProcessor* con el cual el Manager pide la aplicación del algoritmo de procesamiento a la clase cProcessor.

La clase cProcessor carga el procesador mediante el método *ApplyPocessor*. Cuando se termina esta operación, se realiza una petición a la clase cSimplify mediante el método *ApplyProcessor* para que sea aplicado el procesador.

La clase cSimplify, llama al método *CanBeSimplified* con la cual analiza los nodos que se pueden simplificar, y si estos existen entonces se realiza la simplificación del grafo.

4.4.4 CU Aplicar procesador FloodFill

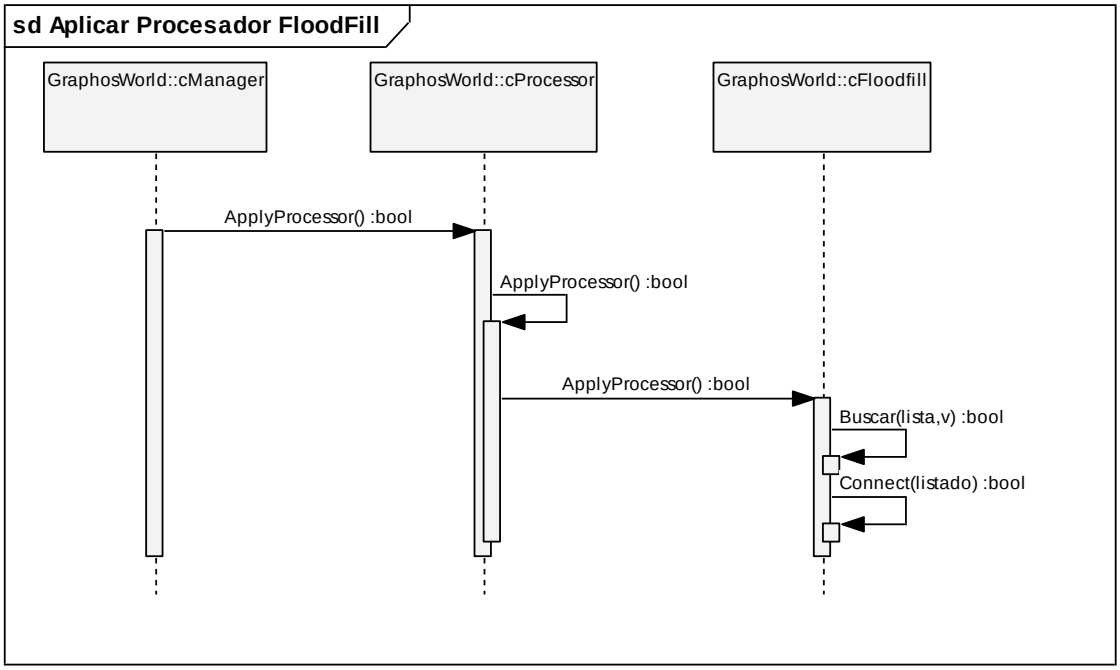


Figura 53. Diagrama de Secuencia del CU aplicar procesador Flood Fill.

Comienza cuando la clase cManager realiza una petición a la clase cProcessor, mediante el método *ApplyProcessor* con el cual el Manager pide la aplicación del algoritmo de procesamiento a la clase cProcessor.

La clase cProcessor carga el procesador mediante el método *ApplyPocessor*. Cuando se termina esta operación, se realiza una petición a la clase cSimplify mediante el método *ApplyProcessor* para que sea aplicado el procesador.

La clase cFloodFill, realiza la aplicación del algoritmo de análisis del grafo y después realiza la llamada a dos métodos diferentes. Con el primer método, *Buscar*, se analiza si un vector pasado esta en la lista de los elementos que se encuentra creada y con el segundo método, *Connect* ,se analiza las conexiones de un vector dado.

4.4.5 CU Aplicar procesador FixConnections

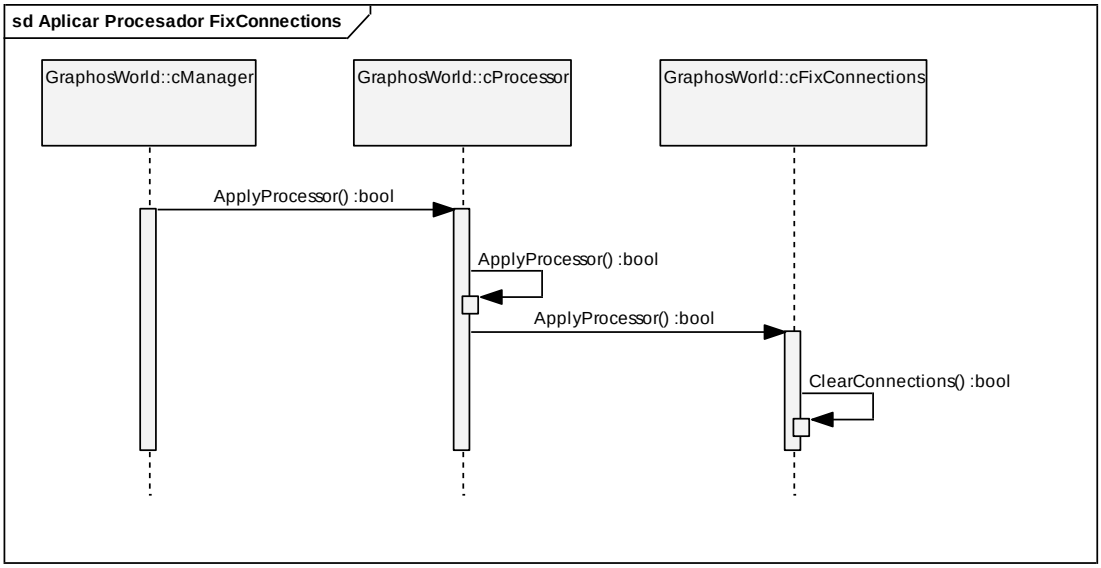


Figura 54. Caso de uso aplicar procesador FixConnections.

Comienza cuando la clase cManager realiza una petición a la clase cProcessor, mediante el método *ApplyProcessor* con el cual el Manager pide la aplicación del algoritmo de procesamiento a la clase cProcessor.

La clase cProcessor carga el procesador mediante el método *ApplyPocessor*. Cuando se termina esta operación, se realiza una petición a la clase cFixConnections mediante el método *ApplyProcessor* para que sea aplicado el procesador.

La clase cFixConnections, llama al método *ClearConnections* con el cual elimina todas las conexiones existentes en el grafo de búsqueda al cual es aplicado.

4.4.6 CU Exportar grafo en .ASCII

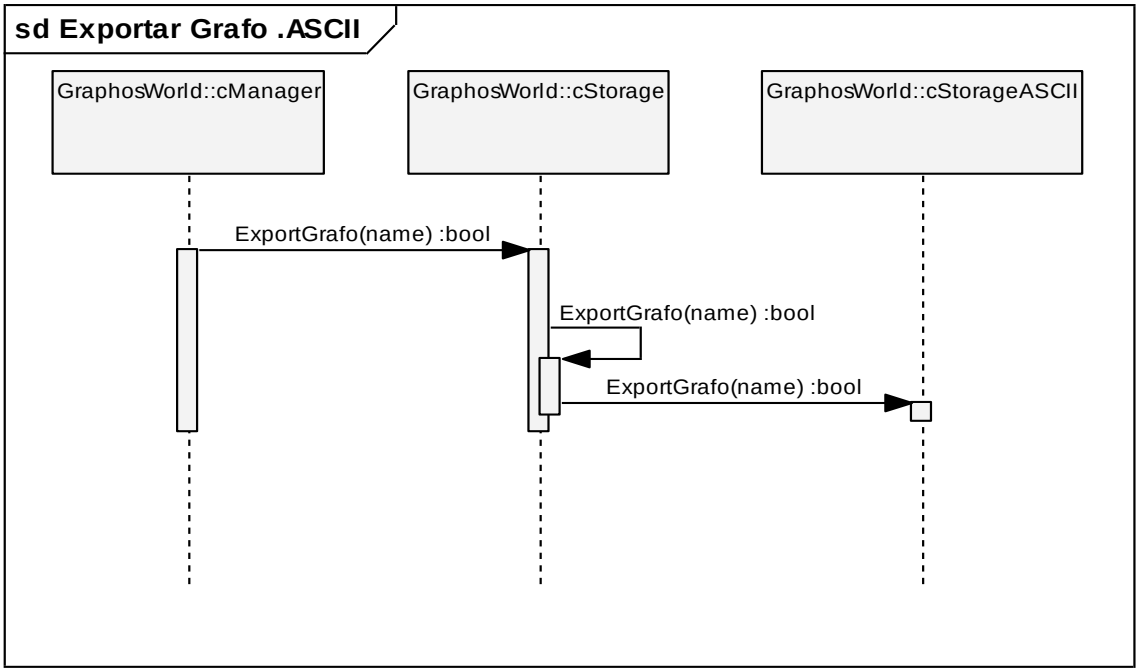


Figura 55. Diagrama de Secuencia del CU Exportar grafo .ASCII

Comienza cuando la clase cManager realiza la petición de exportar a la clase cStorage. La clase cManager realiza la petición mediante el método *ExportGrafo* a la clase cStorage para realizar la exportación del grafo al formato .ASCII.

La clase cStorage crea el storage para realizar la exportación del grafo y después realiza una petición mediante el método *ExportGrafo* a la clase cStorageASCII.

La clase cStorageASCII recibe la petición de la clase cStorage y realiza la exportación del grafo en el formato .ASCII por esta clase.

4.4.7 CU Exportar grafo en .ASE

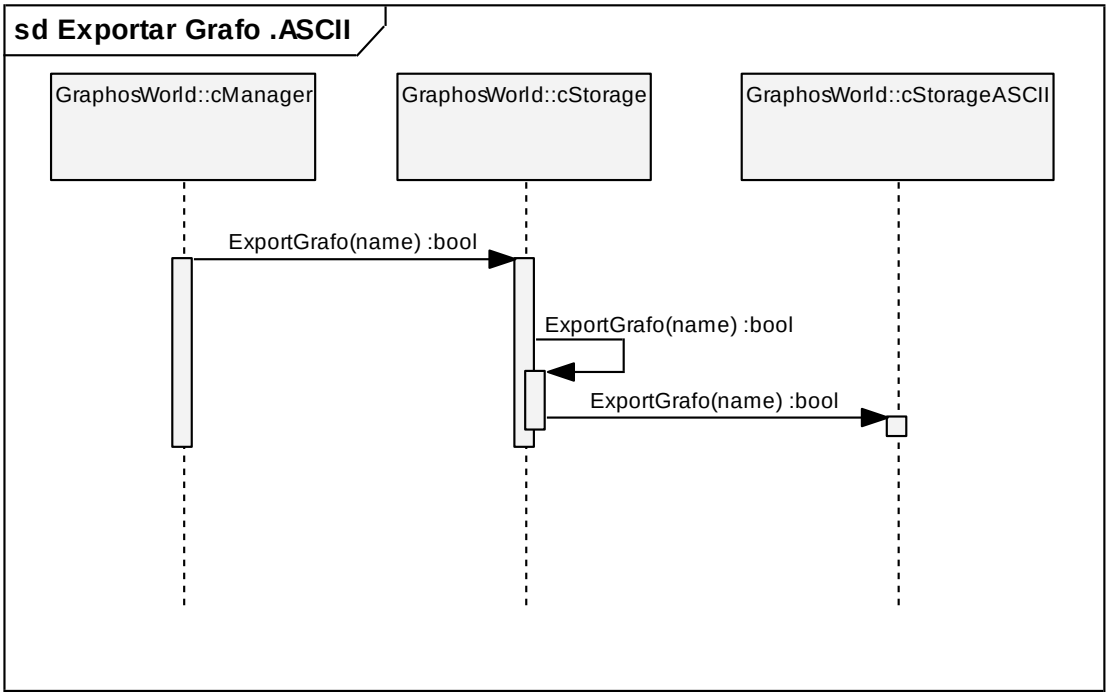


Figura 56. Diagrama de Secuencia del CU Exportar Grafo en .ASE

Comienza cuando la clase cManager realiza una petición a la clase cStorage. La clase cManager realiza la petición mediante el método *ExportGrafo* a la clase cStorage para realizar la exportación del grafo al formato .ASE.

La clase cStorage crea el storage para realizar la exportación del grafo y después realiza una petición mediante el método *ExportGrafo* a la clase cStorageASCII.

La clase cStorageASCII recibe la petición de la clase cStorage y realiza la exportación del grafo en el formato .ASE por esta clase.

4.5 Diagramas de Componentes:

En los anexos 1 – 3, se presentan los diagramas de componentes de la aplicación desarrollada por el presente trabajo. Estos muestran la relación que existe entre los diferentes ficheros de la herramienta, así como con las librerías que se utilizaron para desarrollarla.

Capítulo 5.Resultados

5.1 Pruebas a modelos de mundos:

Para realizar las pruebas de la aplicación de los algoritmos de procesamiento se escogieron cuatro modelos que se muestran en las figuras 56 y 57. Estos modelos se escogieron por el nivel de su complejidad, que va desde un modelo simple hasta un modelo con un alto nivel de complejidad.

El último de estos modelos se encuentra en desarrollo en el proyecto Paseos Virtuales. En la siguiente figura se muestran los modelos en modo wireframe:

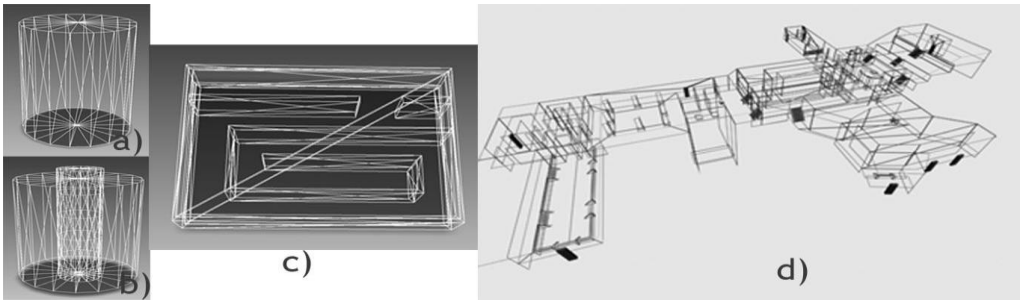


Figura 56. Vista wireframe de los modelos.

Los modelos escogidos por orden son:

a) Cilindro

b) Cilindro dentro de otro cilindro.

c) Laberinto

d) Sector 1 del laberinto del saber 2.

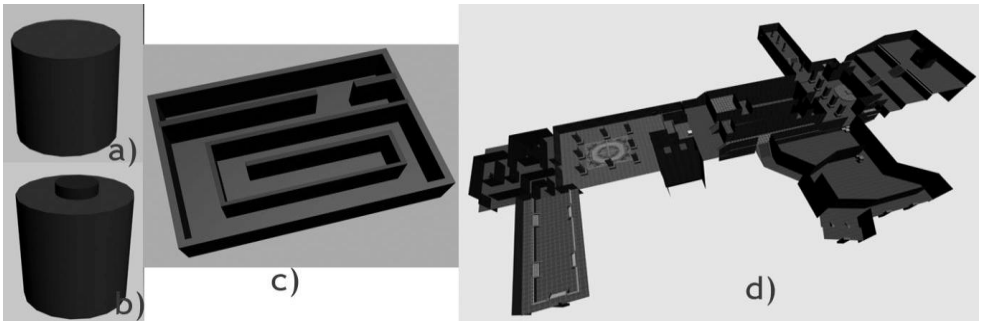


Figura 57. Vista volumétrica de los modelos.

5.2 Pruebas de la aplicación de los procesadores:

A continuación se presentan los resultados que se obtienen a través de aplicación de los diferentes procesadores a cada uno de los modelos propuestos en el tópico anterior.

5.2.1 Flood Fill:

El procesador Flood Fill crea una red de nodos dentro del modelo que se carga. En las siguientes tablas se presentan los resultados obtenidos al aplicarse el procesador Flood Fill a los cuatro modelos propuestos. Cada tabla viene acompañada de la representación del grafo generado.

Algoritmo: FloodFill	
Datos del modelo a): Cilindro	
Número de caras	72
Número de vértices	82
Parámetros de entrada del Procesador:	
X Axis	0
Y Axis	0
Z Axis	0
OffSet	5
Datos del grafo resultante:	
Número de nodos	460
Número de conexiones	1184

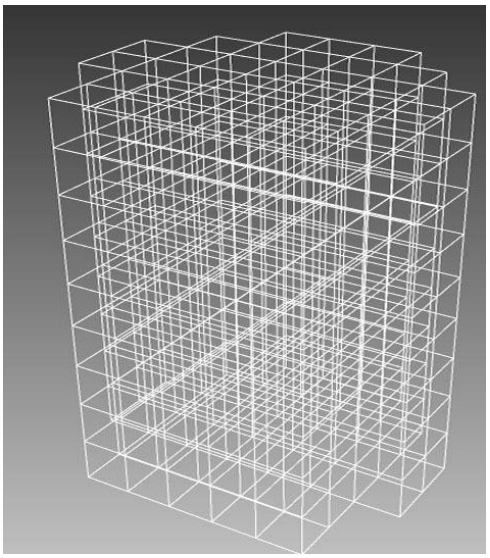


Figura 58. Flood Fill al modelo a).

Algoritmo: FloodFill	
Datos del modelo b): Cilindro dentro de otro cilindro	
Número de caras	288
Número de vértices	238
Parámetros de entrada del Procesador:	
X Axis	7
Y Axis	9
Z Axis	2
OffSet	5
Datos del grafo resultante:	
Número de nodos	351
Número de conexiones	834

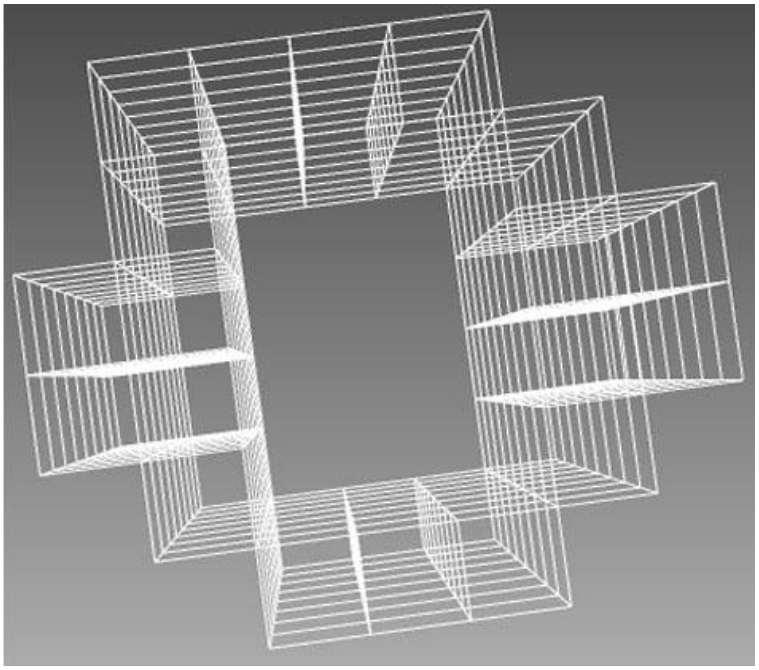


Figura 59. Flood Fill al modelo b).

Algoritmo: FloodFill	
Datos del modelo c): Laberinto	
Número de caras	124
Número de vértices	216
Parámetros de entrada del Procesador:	
X Axis	0
Y Axis	0
Z Axis	0
OffSet	10
Datos del grafo resultante:	
Número de nodos	2665
Número de conexiones	6707

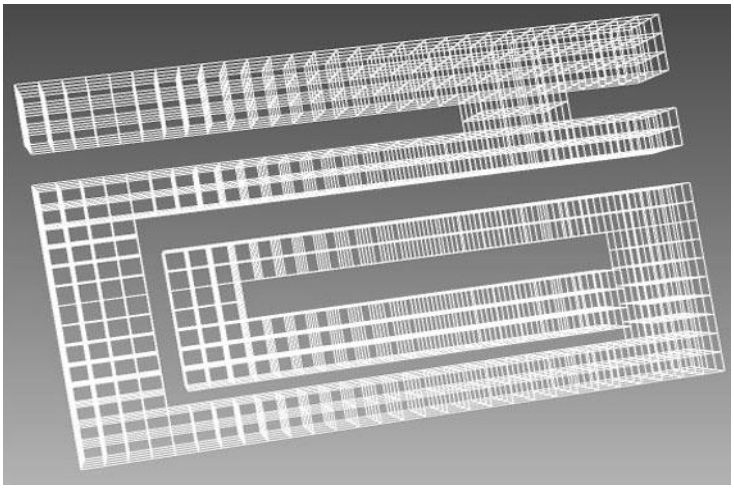


Figura 60. Aplicación del algoritmo Flood Fill al modelo c).

Algoritmo: FloodFill	
Datos del modelo d): Sector 1 Laberinto del Saber	
Número de caras	2428
Número de vértices	5781
Parámetros de entrada del Procesador:	
X Axis	0
Y Axis	0
Z Axis	0
OffSet	50
Datos del grafo resultante:	
Número de nodos	5762
Número de conexiones	13527

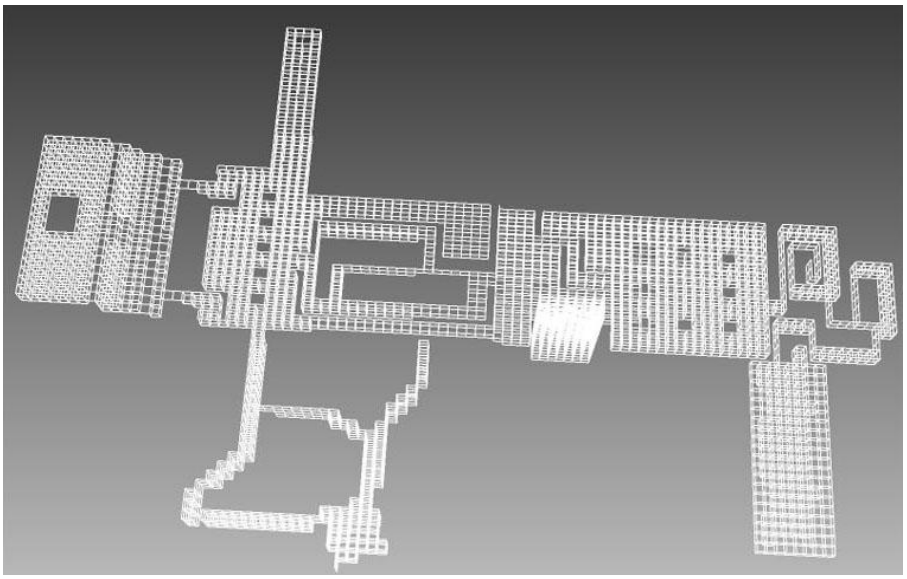


Figura 61. Aplicación del algoritmo Flood Fill al modelo d).

5.2.2 Simplify:

El algoritmo de procesamiento Simplify realiza la simplificación o eliminación de nodos del grafo de búsqueda. En las siguientes tablas se presentan los resultados obtenidos al aplicarse el procesador Simplify a los cuatro grafos obtenidos a partir de los cuatro modelos propuestos justo después de habér sido generados con el procesador Flood Fill. Cada tabla viene acompañada de la representación del grafo generado.

Algoritmo: Simplify	
Datos del modelo a): Cilindro	
Número de caras	2428
Número de vértices	5781
Datos del Grafo generado por el Flood Fill	
Número de nodos	460
Número de conexiones	1184
Datos del grafo resultante:	
Número de nodos	1
Número de conexiones	0

En la figura 62, se representa la aplicación del algoritmo al grafo obtenido del modelo a). Se obtiene como resultado, un nodo sin conexiones. Esto sucede dado que el nodo tiene toda la visibilidad dentro del modelo.

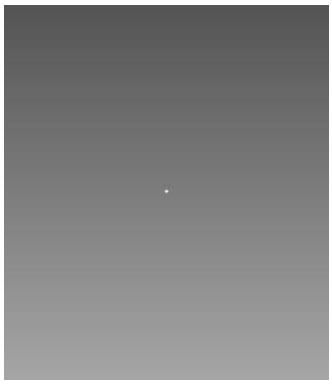


Figura 62. Aplicación del algoritmo Simplify al grafo obtenido del modelo a).

Algoritmo: Simplify	
Datos del modelo b): Cilindro dentro de otro cilindro	
Número de caras	288
Número de vértices	238
Datos del Grafo generado por el Flood Fill	
Número de nodos	351
Número de conexiones	834
Datos del grafo resultante:	
Número de nodos	9
Número de conexiones	11

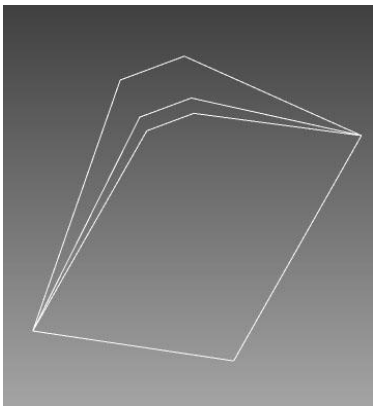


Figura 63. Aplicación del algoritmo Simplify al grafo del modelo b).

En la siguiente figura se representa la aplicación al grafo del modelo c). El resultado obtenido es de 12 nodos y 11 conexiones.

Algoritmo: Simplify	
Datos del modelo c): Laberinto	
Número de caras	124
Número de vértices	216
Datos del Grafo generado por el Flood Fill	
Número de nodos	2665
Número de conexiones	6707
Datos del grafo resultante:	
Número de nodos	12
Número de conexiones	11

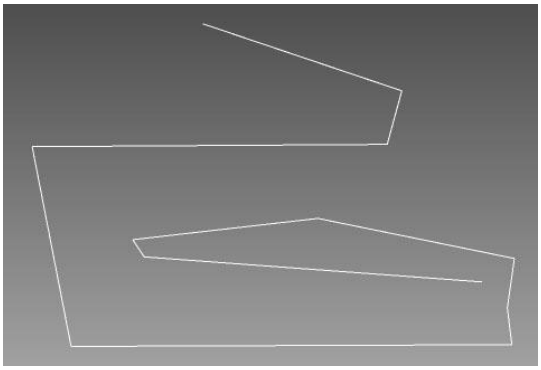


Figura 64. Aplicación del algoritmo Simplify al grafo del modelo c).

Algoritmo: Simplify	
Datos del modelo d): Sector 1 Laberinto del Saber	
Número de caras	2428
Número de vértices	5781
Datos del Grafo generado por el Flood Fill	
Número de nodos	5762
Número de conexiones	13527
Datos del grafo resultante:	
Número de nodos	148
Número de conexiones	190

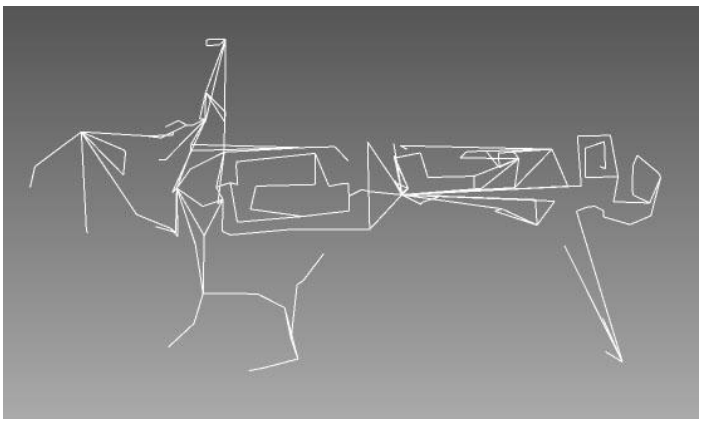


Figura 65. Aplicación del algoritmo simplify al grafo del modelo d).

5.2.3 FixConnections:

Este procesador conecta todos los nodos que no estén conectados mientras entre ellos no exista ninguna cara del modelo cargado. Este procesador solo modifica el número de conexiones.

Al grafo generado por el procesador simplify a partir del modelo a), no se le aplicó el procesador FixConnections ya que el mismo presenta un solo nodo y no presenta conexiones.

En las siguientes tablas se presentan los resultados obtenidos al aplicarse el procesador FixConnections a los últimos tres grafos obtenidos a partir de los cuatro modelos propuestos justo después de habérsele aplicado el procesador Simplify. Cada tabla viene acompañada de la representación del grafo generado.

Algoritmo: FixConnections	
Datos del modelo b): Cilindro dentro de otro cilindro	
Datos del Grafo generado por el procesador Simplify	
Número de nodos	9
Número de conexiones	11
Datos del grafo resultante:	
Número de nodos	9
Número de conexiones	23

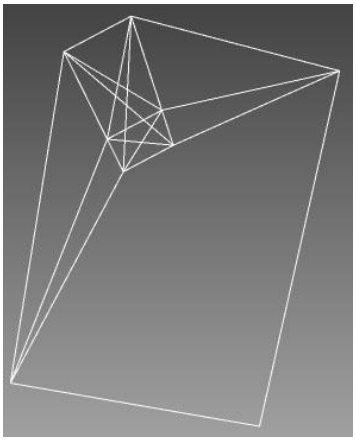


Figura 66. Aplicación del procesador al grafo del modelo b).

Algoritmo: FixConnections	
Datos del modelo c): Laberinto	
Datos del Grafo generado por el procesador Simplify	
Número de nodos	12
Número de conexiones	11
Datos del grafo resultante:	
Número de nodos	12
Número de conexiones	12

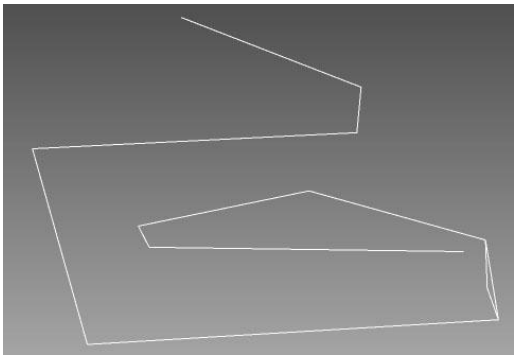


Figura 67. Aplicación del procesador al grafo del modelo c).

En la figura 68, se presenta el grafo obtenido al aplicar el procesador al modelo d). Se obtuvieron 684 conexiones con la misma cantidad de nodos.

Algoritmo: FixConnections	
Datos del modelo c): Sector 1 Laberinto del Saber	
Datos del Grafo generado por el procesador Simplify	
Número de nodos	148
Número de conexiones	190
Datos del grafo resultante:	
Número de nodos	148
Número de conexiones	684

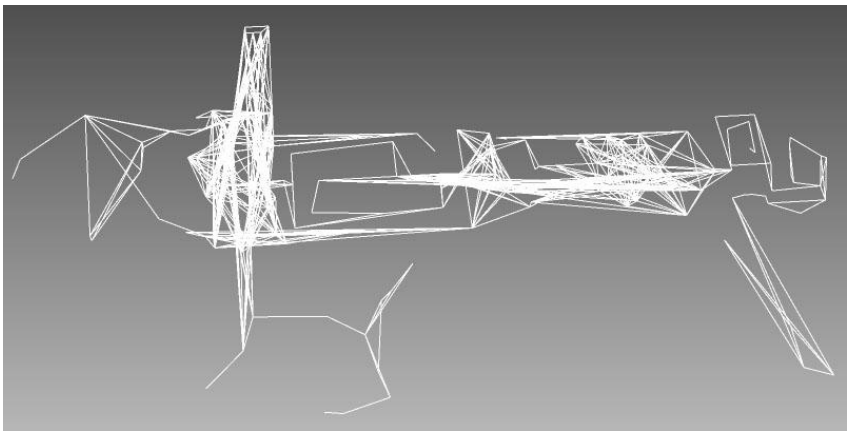


Figura 68. Aplicación del procesador al grafo del modelo d).

5.3 Resultados de las pruebas:

Para concluir se muestra a partir de las siguientes figuras los resultados obtenidos a partir de las pruebas realizadas con cada uno de los procesadores. En estas se obtuvieron grafos de búsqueda lo suficientemente optimizados y aceptables para interactuar con los entornos virtuales para los cuales se generaron.

Además en las pruebas hechas a los modelos se llegó a la conclusión de que es recomendable aplicar el método Simplify como máximo dos veces al grafo que se obtiene del Flood Fill. Si se pasara de esta cantidad de veces, el grafo obtenido cambiaría y no lograría obtener el resultado requerido.

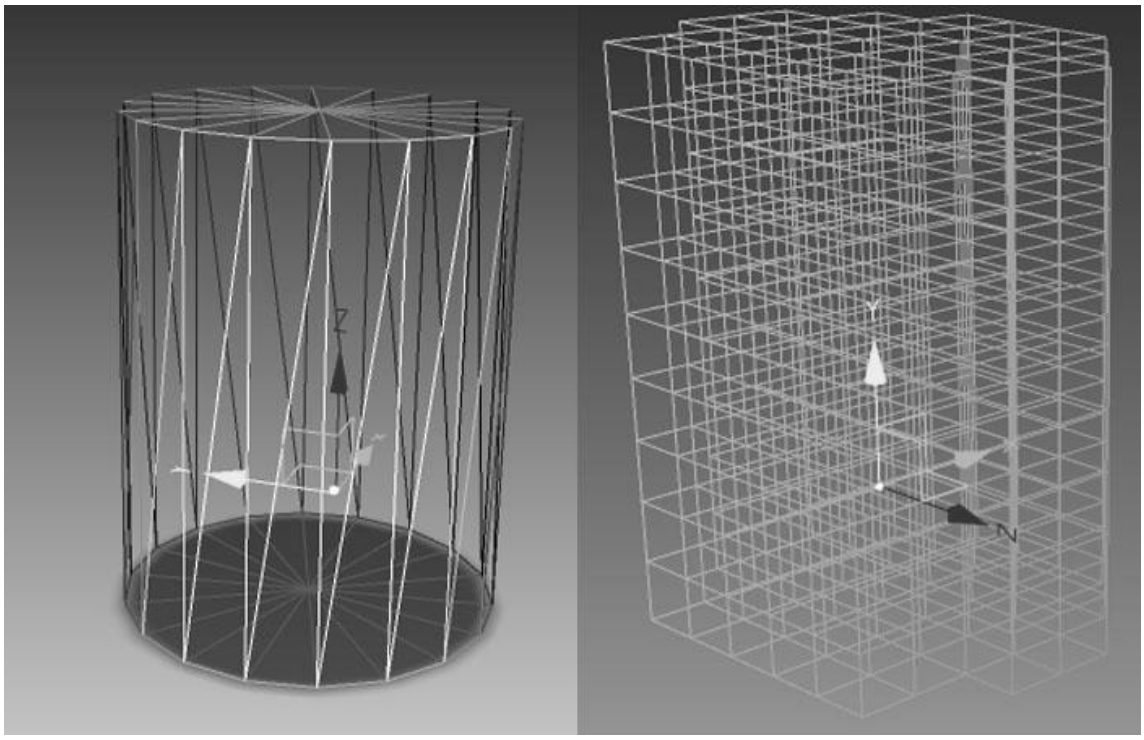


Figura 69. Resultado obtenido en el modelo del cilindro.

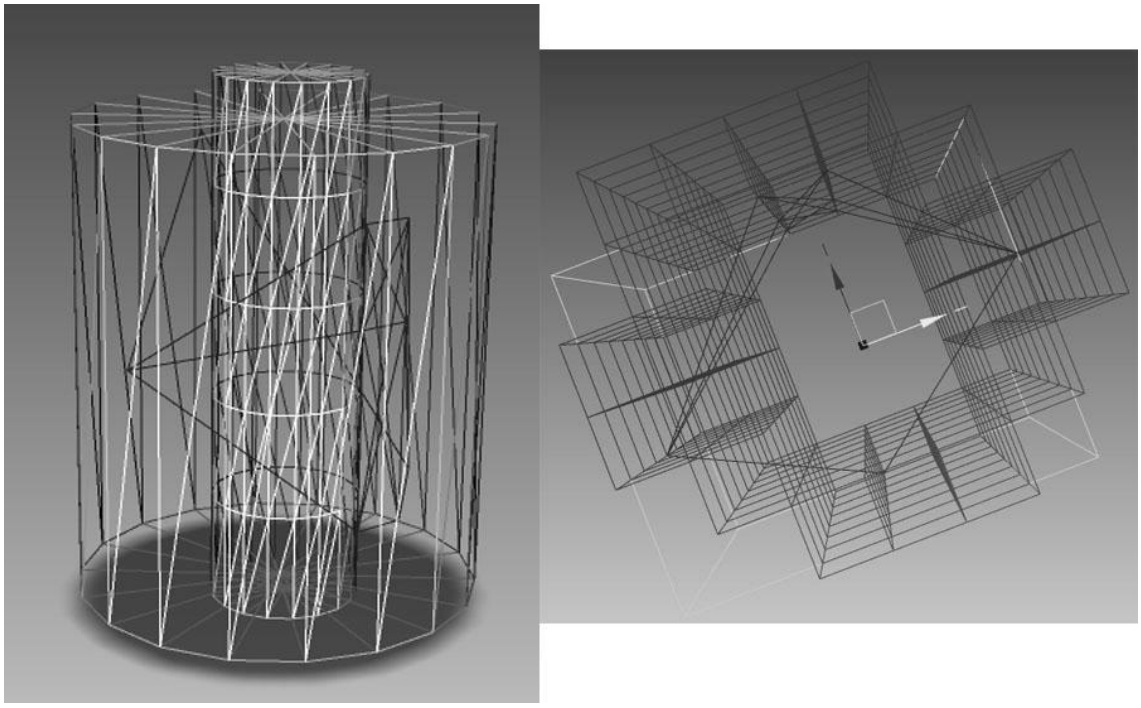


Figura 70. Resultados en el modelo de un cilindro dentro de otro.

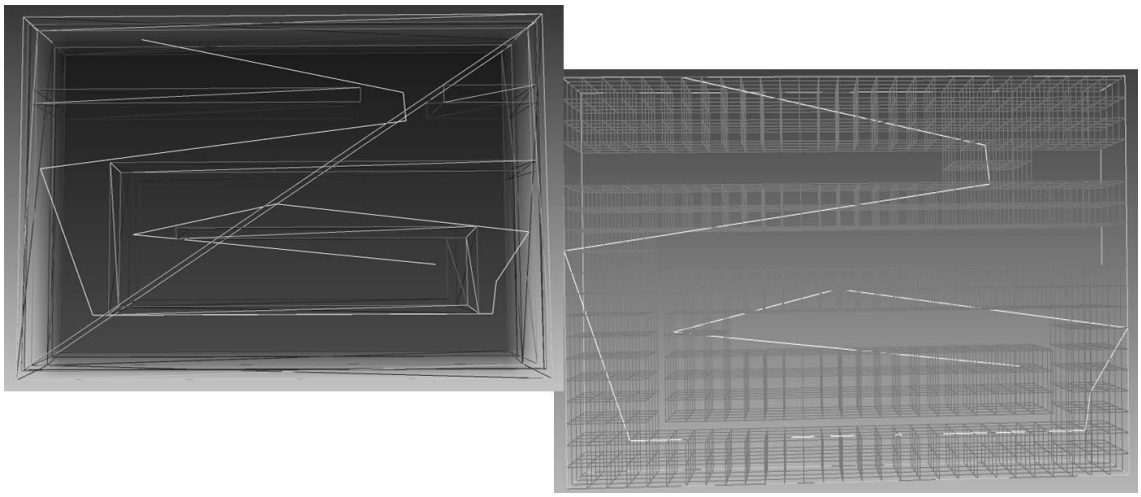


Figura 71. Resultados en el modelo laberinto.

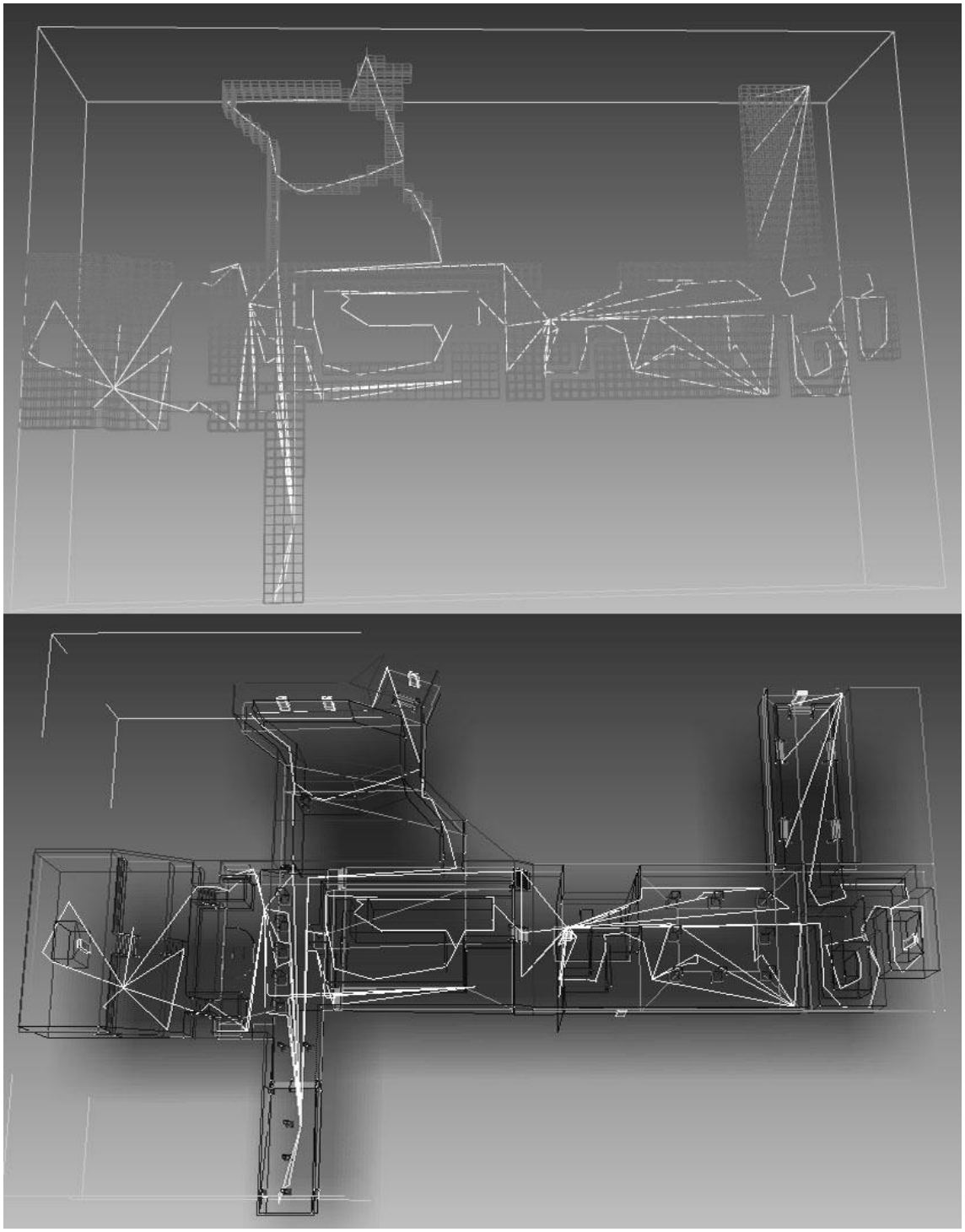


Figura 72. Resultados en el modelo del sector 1 del juego en desarrollo.

Conclusiones

Para dar solución al objetivo principal de este trabajo se necesitó realizar un estudio a profundidad de los modelos cognitivos y de los grafos de búsqueda. En el estudio se tuvo que realizar una recopilación de información acerca de los distintos algoritmos de procesamiento y cálculo, así como realizar el análisis de estos para comprender su interacción con los grafos de búsqueda.

Al terminar este estudio, se pasó a implementar los algoritmos a ser usados por la herramienta propuesta en el trabajo. Se realizaron pruebas de su funcionamiento y aplicación.

Al ser terminados, se unieron y pasaron a formar parte de la aplicación propuesta en el diploma. Demostrando buenos resultados en la creación, modificación y optimización de grafos de búsqueda.

Recomendaciones

- Continuar el desarrollo del software con la incorporación de otros procesadores que puedan crear, modificar y optimizar grafos de búsqueda así como brindar datos topológicos del mismo.

- Implementar uno o varios plugins que puedan interactuar de forma visual con el 3D Studio Max, para realizar ajuste de los grafos creados por la aplicación.

- Ampliar los formatos de la aplicación para cargar y/o exportar los grafos.

- Crear algoritmos más eficientes que demoren menor tiempo en la ejecución de los procesadores.

1. Boden, M. A. (1996). Artificial Genius. Artificial Genius. 17: 3.

2. Boden, M. A. (1998). Creativity and Artificial Intelligence. Artificial Intelligence. 103: 9.

3. Buckland, M. (2005). Programming Game AI by Example. W. Publishing.

4. Ivar Jacobson, G. B., James Rumbaugh, Ed. (2004). El Proceso Unificado de Desarrollo de Software Ciudad de La Habana, Editorial Felix Varela.

5. Larman, C., Ed. (2004). UML y Patrones. Ciudad de la Habana, Editorial Felix Varela.

6. Lidén, L. (2002). Strategic and Tactical Reasoning with Waypoints. AI Programming Winsdom. 1: 10.

7. Lidén, L. (2002). "The Use of Artificial Intelligence in the Computer Game Industry. [Disponible en: http://ai.eecs.umich.edu/people/laird/game-seminar/Liden.ppt]

8. Remco Strattman, A. B., William van der Sterren (2006). Dynamic Procedural Combat Tactics. AI Game Programming Wisdom. 3: 15.

9. Snook, G. (2000). 3D Movement and Pathfinding Using Navigation Meshes. Game Programming Gems. 1: 16.

10. Sterren, W. v. d. (2000) AI for Tactical Grenade Handling. Volume, [Disponible en: http://www.cgf-ai.com/docs/grenadehandling.pdf]

11. Sterren, W. v. d. (2001). Terrain Reasoning for 3D Action Game. GameDevelopers Conference. San Jose Convention Center.

12. Sterren, W. v. d. (2001). Terrain Reasoning for 3D Action Games. Game Programming Gems. 2: 10.

13. T. Conde, D. T. (2004). An Artificial Life Environment for Autonomous Virtual Agents with multi-sensorial and multi-perceptive features. . Computer Animation

and Virtual Worlds. 15: 8.

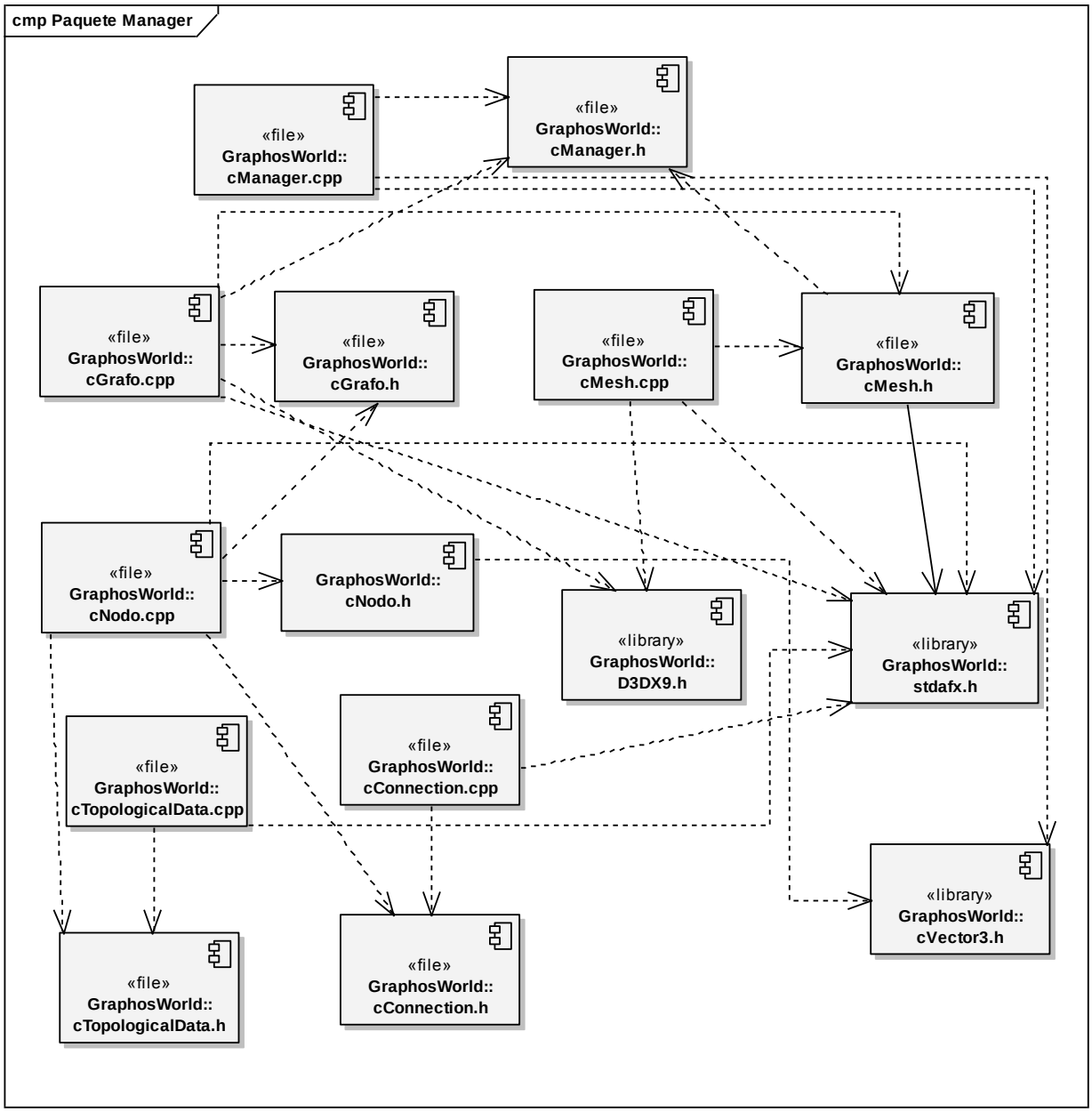
14. Tozour, P. (2001). Influence Mapping. Game Programming Gems. 2: 11.

15. Tozour, P. (2001). Strategic Assessment Techniques. Game Programming Gems. 2: 9.

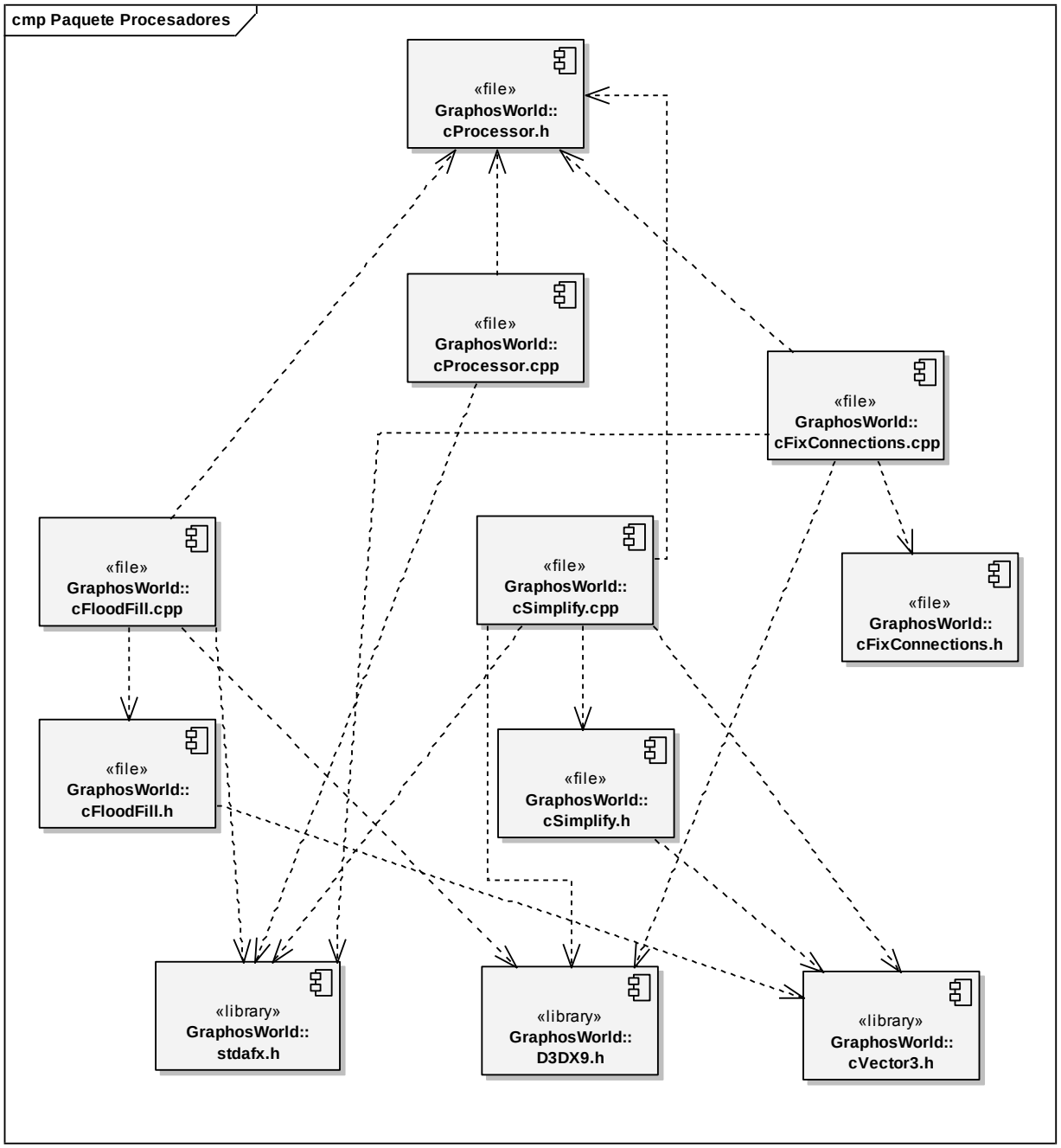
16. Watson, M. (1996). AI Agents in Virtual Reality Worlds. W. Sons. New York: 309.

17. Young, T. (2001). Expand Geometry for Points of Visibility Pathfinding. Game Programming Gems. 2: 7.

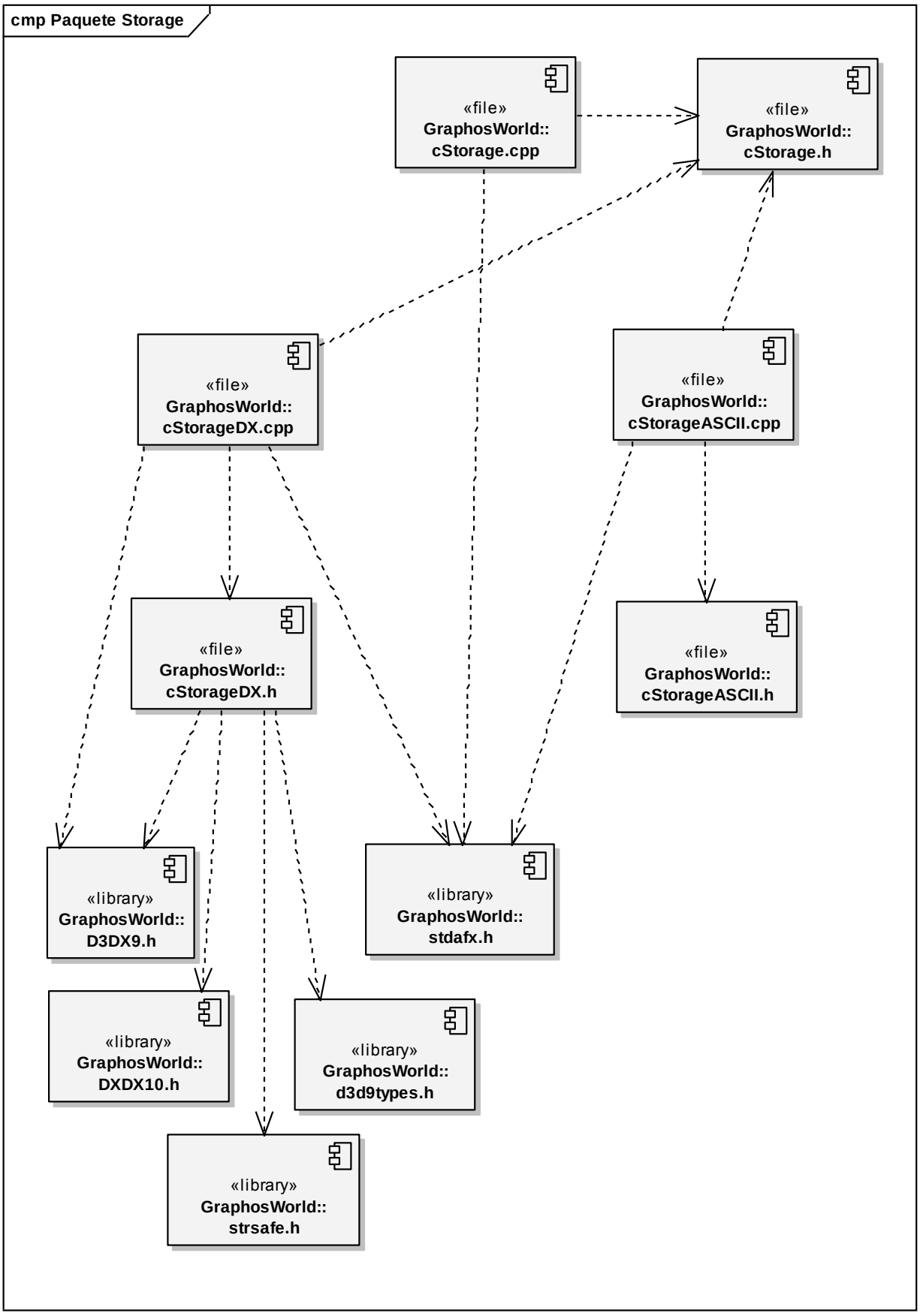
Anexos



Anexo 1. Diagrama de componentes del paquete Manager.



Anexo 2. Diagrama de Componentes del paquete Procesadores.



Anexo 3. Diagrama de Componentes del paquete Storage.

Glosario de términos:

BOT: Diminutivo de Robot. Programa informático que realiza diversas funciones imitando el comportamiento humano.

Grafo: Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas.

Grafos de búsqueda: Grafo que se usa en entornos virtuales para brindar información acerca de la estructura del entorno y para la aplicación de algoritmos de inteligencia artificial como la búsqueda de caminos.

Inteligencia artificial: Se denomina a la ciencia que intenta la creación de programas para máquinas que imiten el comportamiento y la comprensión humana.

Interfaz de usuario: Es la parte del programa informático que permite el flujo de información entre varias aplicaciones o entre el propio programa y el usuario.

Mesh: Objeto tridimensional compuesto por un conjunto de triángulos que a su vez están compuestos por vértices y aristas.

Modelo Cognitivo: Modelo que recoge las principales características del mundo que ha de ser analizado por el BOT.

Nodos: Estructuras que representan la posición en el mapa o grafo.

Plugin: aplicación que interactúa con otra para aportarle una función o utilidad específica. Se utilizan como una vía de expandir programas de forma modular, de manera que se puedan añadir nuevas funcionalidades sin afectar a las ya existentes, ni complicar el desarrollo del programa principal.

RTS: Real Time Strategy (Estrategía en Tiempo Real)

RAM: Random Access Memory (memoria de acceso aleatorio ó memoria de acceso directo).

UML: Unified Languajes Process (Lenguaje Unificado de Procesos)

Waypoints: Puntos de visibilidad en un grafo que son usados para la navegación por el mismo.

Wireframe: Vista alámbrica. Vista que muestra solamente las aristas de los objetos.