Universidad de las Ciencias Informáticas

Biblioteca de Inteligencia Artificial para Sistemas de Realidad Virtual

Trabajo de Diploma para optar por el Título de Ingeniería Informática

Autores

Héctor Yunier Martínez de la Cruz

Yidier Romero Zaldivar

Tutor

Msc. Ernesto Antonio González Díaz

Ciudad de la Habana

Abril 2006

“*Mientras los filósofos discuten si es posible o no la inteligencia artificial, los investigadores la construyen”.*

C. Frabetti.

"The best way to predict the future, is to invent it"

Alan Kay. American computer scientist, researcher and visionary.

“The future belongs to those who believe in the beauty of their dreams.”

Eleanor Roosvelt.

Agradezco a Fals por su maquina de moler carne.

A Susel, a Diana, a Tamara, a la China, a Alejandro, al viejo Andrés, por su apoyo incondicional, su tertulias imprescindibles, y en especial a mi Tata, por el amor y el compromiso que me ha hecho sentir siempre. A mis padres, por el apoyo incondicional y la libertad en todas mis acciones, a mis amigos, mejor a mis hermanos, por haberme ayudado a llegar a este punto, por haber sido mi ejemplo en cada paso de mi vida. A Fabio por ser nuestro padre, el padre que muchos hubiésemos deseado tener. A todos en general, porque cada unos de ustedes ha sido cómplice de este trabajo.

Hector Y. Martinez de la Cruz.

Agradezco a Casas, a Misleidy, a Avila, a Kelvis, por apoyarme en todo momento, a Julián y a Migue por los consuelos en los momentos finales, a Dara por sus consultas de Ingeniería de Software, a Liudmila y Ailec por alegrarnos los tensos días de la confección de la tesis, a Eneybis y a Mary por recordármelo, a Andrés por los momentos de reflexión, a Héctor y a Frank por insertarme en el mundo de la realidad virtual, a nuestro profesor Fabio que tanto extrañamos cuando faltaba, a mis amigos todos, mi familia y a mis compañeros de año con que hemos compartido penas y glorias juntos.

Yidier Romero Zaldivar.

A quien dedicar el resultado de este esfuerzo, sino a mis padres y a mis amigos, que han sido, concientes o no, los fundamentales responsables de la educación recibida. A mis padres porque han sabido comprender todas mis indecisiones, mis pasos en falso, y mis aciertos, y a mis amigos porque siempre me han apoyado, sea cual sea, el camino que me haya traído hasta este punto. En especial dedicado a mi Tata, sin su amor, todo esto no hubiera tenido sentido, Te Amo.

Hector Y. Martinez de la Cruz.

A mis padres a quienes les debo todo lo que soy y se merecen lo mejor del mundo.

Yidier Romero Zaldivar.

Resumen

El presente trabajo pone en manos del usuario una alternativa para el desarrollo de aplicaciones con entornos tridimensionales y que contengan agentes que necesiten recorrer el mismo con determinado nivel de realismo. La solución propuesta consiste en un paquete de herramientas que le proporciona al usuario cierta facilidad cuando vaya a implementar aplicaciones como juegos para PC, visitas virtuales, simuladores, etc., que requieran de algoritmos para que los agentes calculen los caminos mínimos para trasladarse de un punto a otro. Nacionalmente no se han hecho investigaciones relevantes respecto al tema y este paquete de herramientas puede resultar el comienzo de solución potente para este tipo de aplicaciones. El paquete está compuesto por dos herramientas fundamentales: un importador de datos, que es el encargado de importar el mapa diseñado en el 3D max para posteriormente generar un mapa con el tipo de datos que trabaja la librería, y se tiene además, dicha librería que constituye la esencia del paquete, la misma se encarga de brindarle las funcionalidades que necesita el desarrollador para que sus agentes se trasladen de un lugar a otro simulando el comportamiento de un ser humano. El objetivo principal de esta propuesta es facilitarles a los programadores el engorroso método de implementar un módulo de navegabilidad en los entornos 3D para cada aplicación que de haga, mediante la reutilización del código y permitiéndole cierta flexibilidad al desarrollador para que pueda crear lo que desee. Podría decirse que el desarrollador se ahorra mucho tiempo después que se familiariza con el paquete. Se debe aclarar que esta propuesta no trata de ponerse al nivel de la mejor de sus homólogas internacionalmente pero estas herramientas sofisticadas tienen un costo elevado y nuestra alternativa es mucho más económica pues podría ser usada por la universidad si gastarse un céntimo.

ÍNDICE

INTRODUCCIÓN *1*

CAPÍTULO 1 Fundamentación Teórica *4*

1.1 Introducción *4*

1.2 La Inteligencia Artificial *4*

1.3 Principales tendencias de la Inteligencia Artificial (IA). *6*

1.4 Inteligencia Artificial para Juegos *9*

1.5 Generalidades de los métodos de la búsqueda de caminos (Path Finding) *12*

1.5.1 Métodos de Línea de Intersección 12

1.5.2 Métodos de Grafos Cargados 13

1.5.3 Otros métodos 13

1.6 Algoritmos de Búsqueda *14*

1.6.1 Generalidades de los Métodos 14

1.6.1.1 Búsqueda Exhaustiva 15

1.6.1.2 Métodos de Relajación 16

1.6.1.3 Programación Lineal 16

1.6.1.4 Recocido Simulado 17

1.6.2 El Algoritmo 18

1.6.2.1 Algoritmo Dijkstra's 18

*1.6.2.2 La Mejorada Heurística del A** 20

1.6.2.3 Reconstrucción del Camino 22

1.7 Representación del Grafo *23*

1.7.1 Sistema de Puntos Viables (Waypoint System) 24

1.7.2 Sistema de Reconocimiento de Áreas (Area Awareness System: AAS) 24

1.8 Lenguaje de desarrollo C++ *25*

1.9 Tendencias actuales *26*

1.9.1 El SDK PathEngine. 27
1.9.2 Incursiones en el país y en la universidad..... 27
1.10 Conclusiones 28
CAPÍTULO 2 Descripción de la Solución Propuesta 28
2.1 Introducción 28
2.2 Problema y Situación problémica 28
2.3 Objetivo de Estudio..... 29
2.4 Campo de acción..... 29
2.5 Propuesta de sistema 30
2.5.1 Representación del mundo 31
2.5.2 Importador del 3D Max (ABPathEngine Data Creator)..... 31
2.5.3 La librería ABPathEngine (Areas Based PathEngine)..... 33
2.5.4 El ABPathEngine vs. Métodos actuales 34
2.6 Modelo de Dominio..... 35
2.7 Requisitos Funcionales..... 36
2.8 Requisitos no funcionales..... 37
2.9 Definición de los casos de uso 38
2.9.1 Definición de Actores 38
2.9.2 Diagrama de Casos de Uso 41
2.9.3 Casos de Uso expandidos 41
2.9 Conclusiones 46
CAPÍTULO 3 Construcción de la Solución Propuesta..... 47
3.1 Introducción 47
3.2 Patrones de diseño..... 47
3.3 Diagramas de Clases..... 48
3.4 Diagrama de Paquetes 48
3.5.1 Paquete ABPathEngine Data Creator. 49
3.5.2 Paquete Data Creator Interface..... 52

3.5.3 Paquete ABPathEngine..... 56
3.5.3 ABPathEngine Interface..... 60
3.6 Diagramas de Secuencia..... 65
3.6.1 Flujo de Sucesos “Importar fichero .ASE”..... 65
3.6.2 Flujo de Sucesos “Generar Reachabilities”..... 66
3.6.3 Flujo de Sucesos “Exportar fichero APE”..... 66
3.6.4 Flujo de Sucesos “Importar fichero APE”..... 67
3.6.5 Flujo de Sucesos “Buscar Camino Mínimo”..... 67
3.6.6 Flujo de Sucesos “Buscar Camino Mínimo Parcial”..... 68
3.6.7 Flujo de Sucesos “Buscar Objeto Más Cercano”..... 68
3.7 Modelo de Implementación..... 69
3.8 Descripción de los Casos de Uso de pruebas..... 70
3.9 Conclusiones..... 78
***CONCLUSIONES GENERALES**..... 79*
***RECOMENDACIONES**..... 80*
***Referencias Bibliográficas**..... 81*
***Bibliografía Consultada**..... 83*
***Glosario de Términos**..... 84*
***ANEXOS**..... 86*

Figuras

Figura 1 - Un pequeño árbol de búsqueda con A*. 23
Figura 2 - Modelo de Dominio. 36
Figura 3 - Diagrama de paquetes. 49
Figura 4 - Paquete de componentes “ABPathEngine”. 69
Figura 5 - Subpaquetes de componentes “ABPathEngine”. 70
Figura 6 - Diagrama de Casos de Uso. 86
Figura 7 - Diagrama de Clases CU “Importar ASE”. 87
Figura 8 - Diagrama de Clases CU “Generar Reachabilities”. 88
Figura 9 - Diagrama de Clases CU “Exportar APE”. 89
Figura 10 - Diagrama de Clases CU “Importar APE”. 90
Figura 11 - Diagramas de Clases CU “Buscar Camino Mínimo”. 91
Figura 12 - Diagrama de Secuencia CU “Importar ASE”. 92
Figura 13 - Diagrama de Secuencia CU “Generar Reachabilities”. 92
Figura 14 - Diagrama de Secuencia CU “Exportar APE”. 93
Figura 15 – Diagrama de Secuencia CU “Importar APE”. 93
Figura 16 - Diagrama de Secuencia CU “Buscar Camino”. 94
Figura 17 - Diagrama de Secuencia CU “Buscar Camino Parcial”. 94
Figura 18 - Diagrama de Secuencia CU “Buscar Objeto más Cercano”. 95
Figura 19 - Paquete de componentes “ABPathEngine Data Creator”. 95
Figura 20 - Subpaquetes de componentes “ABPathEngine Data Creator”. 95
Figura 21 - Diagrama de componentes “ABPathEngine Data Creator”. 96
Figura 22 - Diagrama de componentes “Data Creator Interface”. 97
Figura 23 - Diagrama de componentes “ABPathEngine”. 98
Figura 24 - Diagrama de componentes “ABPathEngine Inteface”. 99

Tablas

Tabla 1 - Actor del Sistema..... 39
Tabla 2 - Listado de casos de uso. 39
Tabla 3 - Descripción de CU “Importar fichero ASE”. 41
Tabla 4 - Descripción de CU “Generar Reachabilities”. 42
Tabla 5 - Descripción de CU “Exportar fichero APE”. 42
Tabla 6 - Descripción de CU “Importar fichero APE”. 43
Tabla 7 - Descripción de CU “Buscar camino”. 44
Tabla 8 - Descripción de CU “Buscar camino parcial”. 44
Tabla 9 - Descripción de CU “Buscar objeto más cercano”. 45
Tabla 10- Clase “cWorld”. 49
Tabla 11 - Clase “cWorldImporter”. 50
Tabla 12 - Clase “cAPEExporter”. 51
Tabla 13 - Clase “cASEFileLoader”. 51
Tabla 14 - Clase “cASEObject”. 52
Tabla 15 – “Clase cArea”. 52
Tabla 16 - Clase “cBoundingObject”. 53
Tabla 17 - Clase “cBoundingHull”. 53
Tabla 18 - Clase “cDoor”..... 53
Tabla 19 - Clase “cFace”. 54
Tabla 20 – “Clase cEdge”. 54
Tabla 21 - Clase “cReachability”. 55
Tabla 22 - Clase “cReachability”. 55
Tabla 23 - Clase “peAgente”. 56
Tabla 24 - Clase “cAPEImporter”. 56
Tabla 25 - Clase “cPluginsWrapper”. 57

Tabla 26 - Clase “pePath”. 57
Tabla 27 - Clase “pePathStep”. 57
Tabla 28 - Clase “peState”. 58
Tabla 29 - Clase “peDescription”. 58
Tabla 30 - Clase “cPathEngine”. 58
Tabla 31 – “Clase cAStarSearcher”. 59
Tabla 32 - Clase “cDijkstraSearcher”. 60
Tabla 33 - Clase “cArea”. 60
Tabla 34 - Clase “cDoor”. 61
Tabla 35 - Clase “cFace”. 62
Tabla 36 - Clase “cReachability”. 62
Tabla 37 – Clase “peBoundingObject”. 63
Tabla 38 - Clase “peBoundingBox”. 63
Tabla 39 - Clase “peBoundingHull”. 63
Tabla 40 - Clase “peBoundingPlane”. 64
Tabla 41 - Clase “PathEngineInterface”. 64
Tabla 42 - Clase “Entity”. 64
Tabla 43 – Caso de prueba “Importar mal fichero”. 70
Tabla 44 – Caso de prueba “Importar fichero mal formado”. 71
Tabla 45 – Caso de prueba “Generar Reachabilities sin importar áreas del fichero”. 71
Tabla 46 – Caso de prueba “Generar Reachabilities luego de haber importado una lista de áreas del fichero”. 72
Tabla 47 – Caso de prueba “Exportar fichero APE, habiendo cargado antes un ASE mal formado”. 72
Tabla 48 – Caso de prueba “Exportar fichero APE sin haber cargado fichero alguno”. 73
Tabla 49 – Caso de prueba “Exportar fichero APE, habiendo cargado un ASE bien formado”. ... 73
Tabla 50 – Caso de prueba “Importar fichero APE no existente”. 74
Tabla 51 – Caso de prueba “Importar fichero APE existente”. 74

Tabla 52 – Caso de prueba “Buscar camino mínimo a un punto determinado no alcanzable”..... 75
Tabla 53 – Caso de prueba “Buscar camino mínimo a un punto determinado alcanzable”..... 75
Tabla 54 – Caso de prueba “Buscar camino mínimo parcial a un punto determinado no alcanzable”. 76
Tabla 55 – Caso de prueba “Buscar camino mínimo parcial a un punto determinado no alcanzable”. 76
Tabla 56 – Caso de prueba “Buscar camino a objeto, de un tipo específico no existente”..... 77
Tabla 57- Caso de prueba “Buscar camino a objeto, de tipo específico existente”. 77

INTRODUCCIÓN

En los últimos años se ha visto un gran desarrollo de las técnicas de Realidad Virtual. Muchos han sido los éxitos y los fracasos tratando de imitar nuestro entorno en la grafica computacional, y tratando también, de imitar el comportamiento de los seres vivos que en él habitan. Para lograr tales comportamientos se aplican técnicas de otras disciplinas como la inteligencia artificial que estudia los métodos y algoritmos más utilizados. En aplicaciones como los simuladores, paseo virtuales y en los conocidos juegos de video se pueden evaluar dichas técnicas donde los jugadores monitoreados por la computadora se comportan de un modo impresionante. En la industria de juegos son probadas muchas variantes de estas técnicas de inteligencia artificial y actualmente este es un medidor de la afición para catalogar la calidad del producto: un juego sin inteligencia artificial sería un reto impresionante del diseñador gráfico.

Uno de los temas siempre presentes en el desarrollo de juegos es el buscador de caminos (pathfinding), el cual tiene implementado uno o más algoritmos de búsqueda de caminos para diferentes situaciones que se presenten en la navegación dentro del entorno virtual. Este pathfinding no se implementa en los juegos de videos solamente, sino que pueden formar parte de simuladores, paseos virtuales y otras aplicaciones en la Realidad Virtual. En nuestro país no se investiga mucho en estos temas, los logros de la inteligencia artificial han sido en otros campos como la medicina, la educación, economía, etc. que han necesitado estas técnicas para automatizar algunos procesos y ganar en precisión y efectividad. Mundialmente, los grupos de desarrollo de juegos implementan un módulo de inteligencia artificial por juego, aunque se conoce que existen desarrollados varios motores de inteligencia artificial, muchos de los mismos son para resolver un problema en particular y son escogidos los que garantizan su estabilidad. También existen paquetes de herramientas de inteligencia artificial que facilitan el trabajo a los desarrolladores como el PathEngine, excelente, pero tiene el inconveniente del costo que hay que pagar para utilizarlo.

En este trabajo se propone como objetivo general desarrollar una herramienta para el desarrollador de aplicaciones virtuales: videojuegos, simuladores o paseos virtuales, que le facilite la implementación de técnicas de inteligencia artificial para la navegación de los personajes monitoreados por la computadora dentro del mundo virtual.

Los objetivos específicos que se persiguen son: lograr un método adecuado para representar el entorno virtual en un formato fácil de manipular para los desarrolladores de aplicaciones virtuales, utilizar los algoritmos indicados para que las búsquedas de caminos sean efectivas y que se adapte a los cambios con facilidad.

Para cumplir los objetivos propuestos se hizo una investigación en la literatura nacional e internacional para determinar cuáles de las soluciones actuales nos era factible. Se analizó los métodos de búsqueda que estudia la inteligencia artificial para escoger los algoritmos que se implementaría en la solución. Se analizaron las técnicas más utilizadas para representar los mapas y lograr una eficiente navegabilidad dentro del mismo. Se estudió algunos motores de búsqueda de caminos que aportaron ideas en la confección de nuestra solución entre los que se encontraba el sofisticado paquete de herramientas PathEngine.

Este trabajo cuenta de cuatro capítulos y una introducción que explica brevemente cuales son los objetivos de la investigación, la problemática y los resultados esperados. El primer capítulo consiste en la fundamentación teórica de la propuesta y abarca conceptos de la inteligencia artificial, métodos de búsqueda, algoritmos más utilizados para la búsqueda de caminos, y sus aplicaciones en los entornos 3D, además de las tendencias actuales. En el segundo capítulo se describe los motivos que conllevaron al desarrollo e investigación de nuestra propuesta, y se presenta la propuesta: se muestran los requisitos de la aplicación, las descripciones de los casos de uso y sus diagramas respectivos. En el tercer capítulo se aborda el Análisis y Diseño de la aplicación en sí, o sea, todo el funcionamiento interno de la aplicación: incluye los diagramas de clases de cada caso de uso, diagramas de interacción, y el modelo de implementación.

CAPÍTULO 1 Fundamentación Teórica.

1.1 Introducción

La Inteligencia Artificial (IA) ha devenido en los últimos años en una de las ramas de las Ciencias de la Computación más difundidas y para las que se ha dedicado un gran esfuerzo, por los beneficios que reporta en la solución de problemas donde es necesaria, principalmente, la manipulación de información simbólica, en aquellos problemas en los que aparece la subjetividad para resolverlos o en los que la información a manipular es incompleta o borrosa.

En este capítulo se presentan conceptos y aplicaciones de algunas de las disciplinas que forman parte de las múltiples ramas existentes en la Inteligencia Artificial. Además de definir qué es la Inteligencia Artificial, se hace un estudio sobre los temas referentes a la Inteligencia Artificial aplicada en entornos virtuales de videojuegos, paseos virtuales o simuladores, y se aborda las tendencias actuales.

1.2 La Inteligencia Artificial

La Inteligencia Artificial es una rama de la Ciencia de la Computación dedicada a la creación de hardware y software que imita el pensamiento humano. Su principal objetivo es llevar a la computadora las amplias capacidades del pensamiento humano y, para ello, se convierten a las computadoras en “entes inteligentes” con la creación de software que les permite imitar algunas de las funciones del cerebro humano en aplicaciones particulares. El fin no es reemplazar al hombre, sino proveerlo de una herramienta poderosa para asistirlo en su trabajo.

La Inteligencia Artificial aborda problemas poco estructurados, donde no se conoce de antemano cuál es el mejor método para resolverlo. Hay que descubrir, si acaso, alguna solución. Esta es la

razón de la palabra heurística cuyo significado se asocia a búsqueda. La esencia de la palabra heurística es contraria a la de algoritmo en el sentido de que ella es un camino para buscar lo nuevo, mientras el algoritmo es un camino para realizar lo ya muy bien conocido. Así se comprende que el paradigma primario para la resolución de problemas en Inteligencia Artificial sea la búsqueda de la solución orientada por la heurística, para tratar de reducir la explosión combinatoria que genera la búsqueda de todos los caminos posibles que se presenta en la mayoría de los problemas reales.

La Inteligencia Artificial se ocupa de la representación, adquisición y procesamiento de conocimientos de forma automatizada, de la arquitectura de los programas para estas actividades y de los lenguajes en los que se expresan tales programas. La modelación computacional de los procesos cognoscitivos es también un área de interés de la Inteligencia Artificial. Además se incluyen la percepción, la comprensión y síntesis del lenguaje natural, la robótica inteligente, la modelación del razonamiento, la programación automática y otras más, todas ellas de naturaleza no numérica y todavía del dominio de la heurística.

El desarrollo de la Inteligencia Artificial ha seguido dos líneas principales: la simbólica y la subsimbólica. La primera se caracteriza por desarrollar modelos que describen, formalizan e implementan aspectos sistematizables del conocimiento en forma explícita (Sistemas Expertos, Razonamiento basado en casos, etc.). La otra se basa en los enfoques no representacionales de la Inteligencia Artificial. (R.N.A., algoritmos genéticos y sistemas difusos). El cálculo subsimbólico se basa en el uso de representaciones analógicas, el conocimiento se reparte entre diversas componentes del sistema que están enlazadas y que pueden funcionar en paralelo.

1.3 Principales tendencias de la Inteligencia Artificial (IA).

"El campo de la Inteligencia Artificial académica consiste en una enorme variedad de ramas diferentes y subdisciplinas, algunas del todo opuestas ideológicamente con las otras"[1]. No se pretende que se entienda la IA en los juegos sin antes entender algunas cuestiones del extenso campo de la IA. A continuación se enumerará algunas de la principales tendencias de las técnicas de IA, específicamente, las que se consideran más relevantes para el presente y para el futuro de la IA en los juegos.

- *Sistemas Expertos*: intentan tomar y sacar provecho del conocimiento de un ser humano experto en un campo determinado. Un sistema experto representa la habilidad del experto dentro de una base de conocimiento, y realiza un razonamiento automatizado en la base de conocimiento en respuesta a una petición. Tal sistema puede generar respuestas similares a las que un humano experto podría proporcionar.

- *Razonamiento Basado en Casos*: técnicas que intentan analizar un conjunto de datos comparándolos con una base de datos, un historial posiblemente, de conjuntos de datos conocidos y con las respuestas más convenientes en tales situaciones. La metodología esta inspirada en la tendencia del ser humano de comprender situaciones novedosas comparándolas con situaciones similares experimentadas alguna vez en el pasado.

- *Maquinas de Estados Finitos*: son sistemas basados en reglas en las cuales un número finitos de estados están conectados en un grafo dirigido por "transiciones" entre estados. La Máquina de Estados Finitos ocupa exactamente un estado en cada momento.

- *Sistemas de Producción*: están conformados por una base de datos de reglas. Cada regla consiste en una sentencia condicional arbitrariamente compleja, más algún número de acciones que puedan ser ejecutadas si se satisface la sentencia condicional. Los Sistemas

de Reglas de Producción son esencialmente listas de sentencias “if - then”, con varios mecanismos de resolución de conflictos disponible en caso de que más de una regla estén satisfechas simultáneamente.

• *Árboles de Decisión*: son similar a las condicionales complejas en las sentencias “if - then”. Los Árboles de Decisión toman una decisión basada en un conjunto de datos comenzando en la raíz del árbol y seleccionando, por cada nodo, un nodo hijo según el valor de entrada. Los algoritmos como el ID3 y el C4.5 pueden construir los árboles de decisión automáticamente de datos muestra.

• *Métodos de Búsqueda*: está interesado en descubrir una secuencia de acciones o estados en un grafo que satisfagan un objetivo – cualquier alcance a un “estado final” especifico o simplemente maximizando algún valor basado en los estados accesibles.

• *Sistemas de Planificación y Sistemas de Programación*: son una extensión de los métodos de búsqueda que enfatizan en el subproblema de buscar la mejor (más simple) secuencia de acciones que puedan realizarse para lograr un resultado determinado en tiempo, teniendo un estado inicial del mundo y una definición precisa de las consecuencias de cada acción.

• *Lógica de Primer Orden*: amplía la lógica preposicional con varios rasgos adicionales que le permite razonar acerca de un agente inteligente en un entorno. El mundo consiste en “objetos” con sus identidades individuales y “propiedades” que los distinguen de otros objetos, y varias “relaciones” contenidas entre esos objetos y propiedades.

• *El Cálculo Circunstancial*: emplea la lógica de primer orden para calcular cómo un agente inteligente puede actuar en una situación determinada. El cálculo circunstancial usa un

razonamiento automatizado para determinar el curso de la acción que hará los cambios deseados en la situación del mundo.

- *Sistemas Multi-agentes*: métodos centrados en cómo el comportamiento inteligente puede surgir naturalmente como una propiedad que aparece de la interacción entre múltiples agentes competidores y cooperativos.

- *Vida Artificial (o A-Life)*: se refiere a sistemas multi-agentes que intentan aplicar algunas de las propiedades universales de sistemas vivientes a agentes inteligentes en un entorno virtual.
- *Simulación de Grupos o manadas (flocking)*: es una subcategoría de la Vida Artificial que está enfocada en las técnicas para un movimiento coordinado de modo que los agentes inteligentes conduzcan lo más similar posible a los rebaños y las manadas.

- *Robótica*: trata con el problema de aceptar las maquinas que funcionen interactivamente con el mundo real. La robótica es uno de los campos más viejos, más conocido, y el más exitoso de la IA, y comenzó a experimentar un renacimiento a causa del estallido de la accesible potencia de la computación. La robótica está divida generalmente en tareas del "sistema de control" (salidas) y "sistema de sensores" (entradas).

- *Algoritmos Genéticos y Programación Genética*: son indudablemente unos de los campos más fascinantes de la IA. Estas técnicas intentan imitar directamente el proceso de evolución, realizando selecciones y entrecruzamientos mediante caminos aleatorios y operaciones de mutación en poblaciones de programas, algoritmos o conjuntos de parámetros. "Los algoritmos genéticos y la programación genética han logrado unos resultados verdaderamente notables en años resientes"[2]: es desmentido elegantemente el

falso concepto presente en todo el mundo de que una computadora “solo puede hacer lo que le programemos que haga”.

• *Redes Neuronales*: son una clase de maquina de técnicas de aprendizaje basada el arquitectura de las interconexiones neuronales del cerebro animal y el sistemas nervioso. Las redes neuronales funcionan ajustando reiteradamente los parámetros numéricos internos (o pesos) entre los componentes interconectados de la red, permitiéndoles que aprendan una respuesta óptima, o cercana a la óptima, para una extensa variedad de clases diferentes de tareas de aprendizaje.

• *Lógica Borrosa*: usa números precisos para representar el grado de asociación de un número de conjuntos – opuestos a los valores booleanos (verdadero y falso) tradicionales de la lógica. Las técnicas de la lógica borrosa permite un razonamiento más expresivo y esta calificada de más riqueza u sutileza que la lógica tradicional.

• *Redes Probabilísticas*, el campo específico de inferencias *Bayesianas:* suministran las herramientas para el modelado de las relaciones causales fundamentales entre diferentes fenómenos y utiliza la teoría de las probabilidades para tratar con incertidumbre y con un conocimiento incompleto del mundo. También suministran herramientas para hacer inferencias acerca de los estados del mundo y determina los efectos probables de varias acciones posibles.

1.4 Inteligencia Artificial para Juegos

La inteligencia artificial para juegos ha tomado ventajas de casi todas estas técnicas explicadas anteriormente de una forma u otra, con diversos grados de aciertos.

“Irónicamente, las técnicas más simples (las máquinas de estados finitos, los árboles de decisión, y los sistemas de reglas de producción) la mayoría, muy a menudo, ha probado su valor”[1]. La comunidad de inteligencia artificial para juegos enfrentada con una apretada planificación y los recursos mínimos, ha aceptado ansiosamente los sistemas basados en reglas como el tipo de inteligencia artificial más sencillo para crear, entender, y depurar.

Los sistemas expertos tienen similitudes con la inteligencia artificial para juegos en el sentido de que muchas técnicas de inteligencia artificial para juegos intentan que el personaje virtual se desempeñe en el juego como lo haría un jugador humano experto. Aunque una base de conocimiento de la inteligencia artificial para juegos no se representa tan formalmente como la de un sistema experto, el resultado final es el mismo: una imitación al estilo del jugador experto.

La inteligencia artificial de diversos juegos de tableros, como el ajedrez y el backgammon, han usado los árboles de búsqueda con formidables aciertos. La inteligencia artificial del Backgammon compite actualmente con el nivel del mejor de los jugadores humanos. La inteligencia artificial del ajedrez, genialmente probada, fue una proeza con la amarga derrota del gran maestro Garry Kasparov contra un enorme supercomputador llamado “Deep Blue”. Otros juegos como el Go, no alcanzan el nivel de maestros humanos, pero están acortando la diferencia rápidamente.

Desafortunadamente, las complejidades de los entornos de los modernos juegos de video y los mecanismos de juego hacen imposible usar los árboles de búsqueda por fuerza bruta usados por sistemas como la Deep Blue. No obstante, la inteligencia artificial para juegos usa comúnmente otras técnicas de búsqueda para la navegación y la búsqueda de caminos. El algoritmo de búsqueda A* en particular amerita una mención especial como el más estudiado de la inteligencia artificial de búsqueda de caminos (path finding) en juegos de cualquier género.[5][6]

La inteligencia artificial para juegos tiene cantidad de similitudes con la robótica. Los retos significantes que los robots enfrentaron intentando percibir y comprender el "mundo real" son dramáticamente diferentes de los entornos virtuales, accesibles con facilidad, utilizados por la inteligencia artificial para juegos, mientras la parte sensorial de los robots no es aplicable a la inteligencia artificial para juegos. No obstante, las técnicas de control de posiciones son muy usadas por los agentes que utiliza la inteligencia artificial para juegos los cuales necesitan maniobrar inteligentemente por entorno virtual e interactuar con el jugador, el mundo virtual del juego, y con otros agentes inteligentes. La búsqueda de caminos y la navegación de los agentes inteligentes de un juego tienen similitudes con los problemas de navegación enfrentados por los robots con movilidad.

Las técnicas Vida Artificial, los sistemas multi-agentes, las simulaciones del comportamiento en manadas (flocking) han encontrado su espacio en la inteligencia artificial para juegos. Los juegos como The Sims y SimCity han probado la utilidad y las atracciones significativas de las técnicas de Vida Artificial, y un número de exitosos títulos usan las técnicas de simulación del comportamiento en manadas para algunos de sus movimientos.

Las técnicas de planificación también han encontrado algunos aciertos. Los sistemas de planificación fueron diseñados para problemas de planificación mucho más complejos que las situaciones que enfrentan los agentes utilizados en la inteligencia artificial para juegos, pero esto cambiará indudablemente con los diseños de juegos modernos para lograr el desarrollo de altos niveles de complejidad.

La Lógica Difusa ha probado una técnica popular en muchos campos de inteligencia artificial para juegos. Sin embargo, la lógica de primer-orden formal y el cálculo condicional tiene que encontrar aun aceptación en los juegos. Esto es lo más probable debido a la dificultad del uso del cálculo condicional en los entornos de tiempo real y los retos de la representación adecuada del mundo del juego en el lenguaje lógico formal.

Las Redes Probabilísticas no son comúnmente usadas en juegos. No obstante, satisfacen en gran medida un número sorprendente de problemas específicos de la inteligencia artificial para juegos.[3]

1.5 Generalidades de los métodos de la búsqueda de caminos (Path Finding)

“A través de los años, unas cuantas soluciones al problema de la búsqueda de caminos en el espacio han sido propuestas dentro de varias disciplinas donde el problema ocurre con naturalidad. Gran parte de estas se encaminan en el problema de búsqueda en algún tipo grafo”[4]. Aquí se brinda una introducción muy breve de las más comunes de estas soluciones.

1.5.1 Métodos de Línea de Intersección

Esta es una solución común al problema cuando los datos consisten en objetos geométricos y se tiene un terreno de tipo binario. Esto es, que todos los objetos son obstáculos absolutos en el sentido que ellos no pueden ser atravesados y todo el terreno desocupado por un objeto es considerado despejado sin variaciones en la velocidad y otros parámetros del agente. En este caso, el camino más corto de la distancia Euclidiana es también el camino más rápido. La idea fundamental es la siguiente: primero construir un poliedro convexo (convex hull) para todos los objetos. Las esquinas de los polígonos del poliedro definen un conjunto de vértices. Conectados todos los vértices mediante aristas. Entonces hacer un filtrado de estas aristas y finalmente el camino válido más corto entre el origen y el destino a lo largo de una serie de aritas usando algoritmos para grafos estándar. La principal diferencia entre los métodos en esta clase es cómo es hecho el filtrado. El propósito de este filtrado es eliminar las líneas que no puedan quizás pertenecer a un camino corto y de tal manera que reduce significativamente el número de caminos que necesitan ser examinados. El primer paso es eliminar las aristas que (geométricamente) intersectan algún otro poliedro, esto es, aristas que podrían atravesar un obstáculo. Luego varias normas de distancias basadas en reglas pueden ser aplicadas para eliminar también los caminos extensos. Finalmente, muchos algoritmos usan algún tipo de

heurística para eliminar las aristas adicionales no deseadas, con frecuencia renunciando al requerimiento de soluciones exactas de búsqueda en cada caso y ganando en cambio la gran reducción en el número de aristas.[11]

1.5.2 Métodos de Grafos Cargados

Aunque ellos comparten la naturaleza de búsqueda en un grafo final con los métodos de línea de intersección, estos métodos difieren radicalmente, y en un sentido opuesto, para construir el grafo de búsqueda. La idea fundamental es dividir el espacio en regiones discretas, llamadas celdas, y restringir los movimientos desde una celda particular a sus vecinas. Las celdas vecinas son aquellas que pueden ser accedidas directamente desde una celda particular. Un grafo dirigido es construido tomando las celdas como vértices del grafo y los movimientos posibles a sus celdas vecinas como aristas (dirigidas) entre vértices. La función de carga es definida por la asignación de un costo a cada arista, correspondiente al costo del movimiento a lo largo de la arista en cuestión (tiempo, longitud, o cualquier función apropiada para el problema). La división de espacio, la definición de vecindad y la función de costo de la arista pueden diferir todos entre los distintos métodos en esta clase.[12]

1.5.3 Otros métodos

Varias personas han tratado unas cuantas soluciones de más índole matemática. Por ejemplo, usan funciones bivariables para rastrear la evolución de las curvas de igual distancia en una superficie y del cual se deriva una solución para encontrar los caminos más cortos en muchas superficies.

Otra idea es definir algún tipo de costo de superficie y seguir la dirección del gradiente local, o sea, un método del más abrupto descenso. Sin embargo, aunque la idea fundamental es bastante simple, el manejo del caso de los mínimos locales no es de esa manera. Considerando el ejemplo de un río que fluye naturalmente en la dirección del gradiente negativo de la altura del

terreno. En la naturaleza, estos pueden formar lagos y estanques la rededor de los mínimos locales. Los métodos para la detección y el tratado de tales casos especiales quizás puedan ser desarrollados y usados prácticos buscadores de caminos pero no es lo analizado posteriormente.

También han sido recomendadas las soluciones basadas en diferentes heurísticas. En la práctica estos métodos se comportan óptimamente en todo menos en terrenos simples, son propensos a situaciones de congestión y a menudo no está del todo garantizado que encuentre un camino. Por tanto, la heurística puede ser una muy buena idea para acelerar más los métodos exhaustivos tales como los de los próximos epígrafes, especialmente si el requerimiento de garantía óptimamente siempre puede ser sacrificado.

1.6 Algoritmos de Búsqueda

El problema se ha reducido al problema más común de búsqueda del camino de costo mínimo en un grafo dirigido y cargado. A veces es referido como ‘el camino más corto’ en lugar del de ‘mínimo costo’. Existen métodos establecidos para tales problemas y unas cuantas soluciones son presentadas brevemente próximos epígrafes. El problema es que muchos métodos tienen un bajo rendimiento para grafos grandes y complejos, o solo producen soluciones con poco grado de certeza[4]. Un algoritmo práctico es el examinado en detalle.

1.6.1 Generalidades de los Métodos

Existen un número de técnicas para resolver grafos relacionados con la minimización de problemas. Aquí, la aplicabilidad del presente problema es investigado por algunas de las más comunes soluciones.

1.6.1.1 Búsqueda Exhaustiva

La actitud ingenua es lo realizado en la búsqueda exhaustiva, enumerar todos los caminos posibles y seleccionar los de menos costo. Cualquier método usado para construir estos caminos puede ser ilustrado por un árbol de decisión. Se pudiera, por ejemplo, realizar la búsqueda exhaustiva usando un algoritmo de buscar primero el de más profundidad.[14] Brevemente, comenzar por el rastreando una decisión camino abajo hasta una hoja del árbol de decisión produciendo un primer camino. Luego se retrocede al primer punto de decisión más alto en el árbol donde no se ha examinado aún todas las decisiones posibles y se rastrea una nueva ramificación para un nuevo camino. Para limitar el número de caminos a examinar se puede descartar directamente algunos caminos que tengan ciclos. Continuar de esta manera hasta que sea recorrido el árbol entero. En otras palabras, comenzar del vértice inicial, decidir sobre una de las aristas que están orientadas a otro vértice (esto no es el camino todavía), transitándolo e iterándolo hasta que encuentre el vértice destino.

En cada punto de decisión, se tienen una opción de ocho direcciones. N será el número de vértices. Desde una celda puede ser visitada una vez como máximo (de otra manera va a ser un ciclo), se puede afirmar que cada camino posible tiene como máximo *N-1* aristas, esto es que el árbol de búsqueda tiene una profundidad de *O(N)*. Mediante el creciente número de ramificaciones con unos ocho pliegues por cada nivel se alcanzan a lo sumo unos 8^{N-1} caminos posibles, esto es, se tiene $O(8^{N-1})$ caminos por examinar en el peor de los casos. De esa manera, la actitud ingenua toma un tiempo exponencial para encontrar una solución. Claramente esto no es bueno. El algoritmo es muy estúpido pues no distingue entre la gran mayoría de caminos posibles cuales son probablemente los más óptimos y estos pocos son los probables candidatos.

1.6.1.2 Métodos de Relajación

“Una alternativa al algoritmo ‘el primero en profundidad’ para hacer un búsqueda exhaustiva es el llamado algoritmo de búsqueda ‘el primero en amplitud’”[14]. Este nombre se deriva de la manera en que se expande el ‘frente de búsqueda’ desde el punto de inicio (al árbol de decisión). No se gana mucho directamente por el uso de esta alternativa tal como está en lugar de la búsqueda del primero en profundidad. Sin embargo, existe un clase mayor de algoritmos de búsqueda basada en la llamada técnica de ‘relajación’[14], usa ideas muy similares a la búsqueda del primero en amplitud pero con mejoras importantes. Los más comunes aplicables también en el problema actual, son los clásicos de los libros de texto: los algoritmos Dijkstra’s y Bellman-Ford’s. Ambos determinan todos los caminos óptimos desde el vértice inicial hasta todos los demás vértices. El algoritmo Dijkstra’s es rápido pero requiere de que en el no haya aristas con costos negativos. El algoritmo A* es una efectiva mejora heurística del algoritmo Dijkstra’s que procura un rendimiento promedio mejor cuando se necesita solamente el camino óptimo entre dos vértices (y no de un vértice a todos los demás). Porque proporciona bien el predecible desempeño sin hacer un compromiso de optimizado, es que este algoritmo es el escogido para implementar nuestra propuesta.

1.6.1.3 Programación Lineal

La programación lineal es usada a menudo para resolver varios problemas de flujo de redes y asignación de recursos. Esto es posible para considerar nuestro grafo del problema como un problema de programación lineal de deducción de, por ejemplo, el costo mínimo de un problema de flujo o un problema equivalente de asignación.[15] En la práctica, el problema formulado de esta manera puede hacerse inmenso y el método normal de optimización usado por tales no parece muy eficiente para problemas extensos. Por ejemplo, el llamado método simple es el método clásico del libro de texto para resolver problemas de programación lineal. Sin embargo, establecer convergencias con el método simple y el tiempo que tomará puedo que no sea

establecido con prioridad. Adicionalmente, la matriz de iteración se hace, aunque muy escasamente, tremendamente inmensa (el cuadrado del número de celdas)[15]. Por esa razón esta solución no ha sido evaluada más adelante, aunque pudiera ser mejor, o al menos más efectivo en memoria, sin embargo soluciones disponibles más complicadas que los que han sido mencionados aquí.

1.6.1.4 Recocido Simulado

Este método de minimización general puede ser muy usado para resolver muchos problemas dificultosos usando una búsqueda pseudo-ramdom. Como un ejemplo conocido del problema del NPC es agente viajero puede ser resuelto efectivamente usando el recocido simulado [13]. El método encuentra su inspiración en la manera en que la naturaleza consigue minimizar. El nombre proviene del proceso donde el metal es calentado lentamente y se enfría y congela hasta la energía mínima necesaria para romper los cristales.

La función *f,* a ser minimizada, es llamada función objetivo. En nuestro caso, es el costo del camino. El método comienza con una solución inicial adivinada. Como ejemplo, se puede usar el camino en línea recta entre el origen y el destino (un camino que no sea válido). Este es mejorado de manera iterativa con la aplicación de perturbaciones aleatorias en el camino. Un camino perturbado en aceptado como una nueva solución inicial con probabilidad $p=exp(-\frac{\delta f}{T})$. De ese modo, esta es una probabilidad de aceptación para ciertas perturbaciones que aumentan la función objetivo. Esto es necesario para evitar ser atrapado por mínimos locales. El parámetro T del sistema, a menudo llamado temperatura, es utilizado para controlar esta probabilidad y es reducido hasta que haya sido encontrada una solución aceptable. La parte difícil es construir una buena función de perturbación. Pudiera tener la propiedad de introducir tanto cambios pequeños, como aleatorios, como posibles y al mismo tiempo permitir todas las soluciones posibles que fuesen alcanzadas. A menudo la T es usada también para controlar directamente la escala de las

perturbaciones. Exactamente, cómo T es controlada y cómo determinar cuando parar puede ser dificultoso. Algunos tipos de heurística o lógica de control adaptativo son empleados a menudo. Como un simple ejemplo se pudiera utilizar una T fija hasta, unas 100 iteraciones consecutivas, fallar para generar una solución mejor. Luego se divide T en dos y se continúa de la misma manera. Se detiene cuando la escala de perturbación es lo suficientemente pequeña para quedarse por debajo de la precisión de la solución deseada. De esta manera, una gran escala de optimización es realizada primero (una T grande proporciona grandes perturbaciones) y entonces la optimización se realiza continuamente a escalas reducidas hasta que se satisfaga un local óptimo adecuadamente.

La naturaleza iterativa es una gran fortaleza y una gran debilidad para este método. En el lado positivo, si se escoge un camino en momento determinado, se pudiera parar cuando se quiera. En el lado negativo, si no se tiene conocimiento de cuánto tomará alcanzar un grado de precisión, si no lo tiene nunca (depende de cuan buena es la función de perturbación).

1.6.2 El Algoritmo

El algoritmo A* adiciona una heurística para mejorar el algoritmo clásico Dijkstra's que permitirá un camino óptimo de costo mínimo en un grafo dirigido con los pesos de las aristas no negativos.

1.6.2.1 Algoritmo Dijkstra's

La entrada es un grafo $G = (V, E)$ y el vértice origen $s \in V$. Donde V es el conjunto de vértices y E es el conjunto de aristas en G. Además se tiene una función de peso de la arista $w(u,v) \geq 0$ $\forall (u,v) \in E$. El algoritmo divide los vértices, V, en dos conjuntos separados, $V = R \cup Q$. El conjunto R, contiene a aquellos vértices cuyos pesos han sido determinados el camino más corto final y el conjunto, $Q = V - R$. cuando un vértice es movido de Q a R, se dice que es retirado. Algunas descripciones del algoritmo llaman a Q el conjunto 'abierto' y R el conjunto 'cerrado'. Para cada

vértice, *v*, se almacena un valor estimado del mínimo camino más corto en ese momento, *g[v]*. Dado que no es de interés encontrar el valor del camino de mínimo costo, solamente el conjunto actual de vértices que lo componen, además se almacenará, para cada vértice de una arista predecesora, *ed[v]*, esta es la arista de la vecindad del vértice en el mejor camino actual del vértice en cuestión. Esto puede ser utilizando también para reconstruir el camino por backtraking desde el destino. En el caso de haber más de un camino óptimo, solo el descubierto más recientemente podrá ser recordado con este esquema[14].

El algoritmo, en pseudos-código, procede como se explica más adelante:

```
Dijkstra(G,w,s)
 R = Q = {}
 for all v∈V do
      g[v] = ∞
      Q.Insert(v)
 while (Q ≠ {}) do
      u = Q.ExtractMin()
      R = R ∪ {u}
      for v∈Neighbors{u} do
           if (g[v] > g[u] + w(u,v)) then
                Q.DecreaseKey(v, g[u] + w(u,v))
                ed[v] = (u,v)
```

El conjunto Q debe ser implementado como una cola con prioridad con las siguientes operaciones: *Q.Insert(u)* – que inserta un vértice en la cola, *Q.ExtractMin()* – elimina el elemento $v \in Q$ con el más pequeño de las *g[v]*, *Q.DeceaseKey(v, a)* – decrece *g[v]* hasta *a*, y actualiza la cola.

El algoritmo trabaja extrayendo los vértices $v \in Q$ repetidamente con el estimado del camino mínimo más corto, *g[v]*. Desde todas las aristas de costos positivos, no se puede encontrar un camino más corto prolongando un camino de otros vértices. De esa manera se ha determinado el camino más corto en *v* y los que pueden retirarse en *R*. cuando se chequea cualquier de las prolongaciones del camino óptimo para sus vecinos puede ser un camino más corto que el estimado actual de camino más corto para los respectivos vecinos. Si esto ocurre, entonces se actualiza (o relaja) el estimado de camino más corto y la arista predecesora de las vecinas correspondientes. Porque se selecciona el vértice de mínimo costo y se aflojan, el algoritmo pertenece a la clase de los ambiciosos métodos de relajación.

En una implementación práctica, *Q* se puede dividir en dos subconjuntos separados, *U* y *PQ* donde, para comenzar, *U* contiene todos los vértices excepto s y *PQ* son utilizados en lugar de *Q*. Los vértices son trasladados desde *U* hacia *PQ* cuando son visitados por primera vez. Se puede usar varias banderas para denotar si el vértice ha sido previamente visitado o no (esto es si está en *U*) y de este modo no se tiene que almacenarlo explícitamente. Manteniendo siempre la cola de prioridad *PQ* tan pequeña como sea posible puede ser beneficiosa para el rendimiento.

*1.6.2.2 La Mejorada Heurística del A**

Como solo es de interés el camino más corto entre dos vértices, parece como un derroche calcular todos los caminos desde un vértice hasta los todos los demás vértices como lo hace el algoritmo Dijkstra's. La mejora más obvia es detener directamente en el momento en que es retirado, t, el vértice de destino. El conocimiento de la posición del destino no es usado del todo durante la búsqueda así permanece quieto o enmudecido al respecto. El algoritmo A*[16] se

encarga de adicionar una heurística para alterar la búsqueda hacia el destino y así con la esperanza de terminar rápido. La única diferencia las situaciones de los algoritmos en los cuales el vértice es retirado en cada paso de iteración, esto es, como estén ordenados en la cola con prioridad. La idea fundamental es que en lugar de utilizar *g[v]* (el costo actual desde el origen hasta el vértice en que se encuentra en estos momentos) para ordenar la cola como en el algoritmo Dijkstra's, una nueva entidad, *f[v] = g[v] + h[v]* es usada donde *h[v]* es la heurística. Esto puede ser probado si *h[v]* es un despreciable (o a lo sumo, igual a) del actual camino de mínimo costo para mover desde v hasta f, entonces el A* producirá una solución óptima (tal como lo hace el Dijkstra's)[16]. Como el ordenamiento es la cola con prioridad es modificado, en el peor de los casos las características son similares al Dijkstra's pero se puede anticipar el porciento de rendimiento hasta mejorar mucho para grafos del mundo real si puede proveerse de una buena heurística, *h[v]*.

Generalmente la *'distancia desde u hasta v'* que marca el compás del *'costo mínimo del terreno'* es el mejor estimado que se hace para *h[v]* cuando debe ser garantizada una estimación baja. Cuál distancia es la óptima estimada depende de la estructura del grafo. La verdadera distancia Euclideana podría servir, pero a menudo esta será más pequeña que la longitud de la arista actual a lo largo del camino más corto. En cambio, dado que la estructura del grafo es uniforme se puede calcular la distancia exacta y óptima. Si se tiene aristas ortogonales solamente, esta podría ser la llamada distancia de 'Manhattan': $|\Delta x| + |\Delta y|$. En el caso de estudio, con la arista diagonales también, es fácilmente tratando *'la longitud de la arista diagonal'*min(*$|\Delta x| + |\Delta y|$*) + 'longitud de la arista axial'**$||\Delta x| + |\Delta y||$. Un bono adicional comparado con el uso de la distancia Euclideana es que no es muy costoso es requerido el cálculo de la raíz cuadrada.

Se puede demorar el procesamiento de *h[v]* para cada vértice hasta que se necesite realmente, esto es, cuando es registrado primero en la cola de búsqueda. De este modo la misma bandera que es usada para el seguimiento si un vértices está en *U* también dirá si *h[v]* necesita ser

inicializada. Ya que muchos vértices puede que nunca sean visitados del todo, esto puede librar un poco de trabajo.

Notar que si los costos de la arista en terrenos despejados varia extensamente (por ejemplo, las carreteras son muy baratas mientras que el tipo de terreno plano es costoso), luego adicionando una heurística basada en 'el costo mínimo del terreno' facilita generalmente una gran minimización. En esos casos A* es una mejora menor y se comportará más como el algoritmo Dijkstra's. Una alternativa pudiera ser el uso de un 'costo típico de terreno' en vez de un costo mínimo. Eso violaría la afirmación de que la heurística es una minimización del costo real y de esa manera el comprometer la garantía óptima. En algunas aplicaciones, pudiera ser cambios razonables para soluciones rápidas. En efecto, pudiera usarse $f[v] = g[v] + c*h[v]$ donde c es una constante de control de escala. Para $c = 0$ se tiene el algoritmo Dijkstra's, para $c = 1$ se tiene el algoritmo A* para $c > 1$ se tiene un 'A* aproximado' modificado. El más grande valor de c el más rápido (en porcentaje) excepto más precisa la solución es la adecuada. Con igual costo de terreno una heurística $h = 0$ aporta para una grafo especial una búsqueda frontal que se expande de forma hexagonal, mientras A* con una verdadera heurística Euclideana aporta una forma más parecida a una pera.

1.6.2.3 Reconstrucción del Camino

El resultado de una búsqueda típica es ilustrado en la Figura 1. El círculo es el final del camino. La celdas grises durante la ejecución del algoritmo nunca necesitan ser visitadas. El gris más oscuro es un obstáculo. Las flechas muestran las aristas de los mejores caminos actualmente de todas las celdas examinadas, estos son los valores de *ed[v]* para los vértices en las cabezas de flechas. Para las celdas retiradas (no marcadas) este es también el camino óptimo hasta aquellas celdas. Las flechas oscuras y gruesas muestran el camino de mínimo costo hacia el destino.

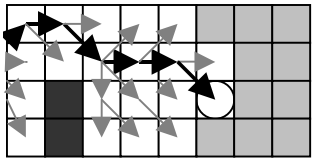

Figura 1 - Un pequeño árbol de búsqueda con A.*

Si solo interesa en la búsqueda del costo del camino mínimo entonces pudiera haber saltado *ed[v]* en el algoritmo (de este modo se ahorra un Byte por nodo). Ahora se puede usar fácilmente para retroceder el camino óptimo del vértice destino. Comenzando en la celda destino y luego caminando el sentido opuesto a la dirección almacenada de la arista en la actual celda hasta alcanzar la celda origen. Esto es simple invertir la secuencia de las celdas visitadas durante el retroceso para generar un camino hacia delante desde el origen hasta el destino.

1.7 Representación del Grafo

Los algoritmos de búsqueda de caminos son trabajados desde los libros de texto de las Ciencias de la Computación en grafos: un grafo es, matemáticamente, un conjunto de vértices conectados entre si. Un mapa cuadriculado para juegos puede ser considerado un grafo donde cada cuadricula es un vértice y las aristas dibujadas entre las cuadriculas que son adyacentes serían las interconexiones entre los vértices del grafo. En los epígrafes anteriores se eligió esta variante del mapa de cuadriculas por las ventajas que brinda al representar los métodos de búsqueda de caminos y los algoritmos, pero no es el utilizado en la aplicación. En la propuesta se aplican los principios del sistema de áreas y se adaptan a la aplicación en el momento de representar el entorno 3D, aunque se tuvo en cuenta los sistemas de puntos viables (waypoints).

1.7.1 Sistema de Puntos Viables (Waypoint System)

Comúnmente, lo usado para los propósitos de la navegación y búsqueda de caminos en los entornos 3D es un sistema de puntos viables (waypoints)[7]. Los principios de este sistema son descritos primeramente, para explicar los principios de las representaciones en 3D usadas en el AAS. El sistema de waypoints usado para la navegación y la búsqueda de caminos es una colección de nodos son interconexiones entre ellos. Para los propósitos de navegación las interconexiones entre los nodos tiene propiedades específicas. La propiedad más importante es que se puede viajar con facilidad desde un nodo a otro si están interconectados. En otras palabras, la complejidad de navegabilidad para alcanzar un waypoint desde otro mediante una interconexión es mínima, por ejemplo moverse en línea recta. Un sistema de waypoints con tales propiedades es el resultado de una solución de tipo “divide y vencerás”[7]. Transitar de un waypoints a otro es un subproblema de solución simple. Todos los lugares accesibles desde un waypoints pueden ser accedidos desde cualquier waypoints transitando a través de uno o más waypoints. La desventaja de utilizar waypoints es que el determinar correctamente si un punto es accesible desde un waypoint, o si un waypoint puede ser accedido desde otro punto, implica cálculos complejos y consumidores de memoria en tiempo de ejecución.

1.7.2 Sistema de Reconocimiento de Áreas (Area Awareness System: AAS)

El Sistema de Reconocimientos de Áreas (AAS) es el sistema perfecto usado para suministrar al bot de toda la información acerca del estado actual del mundo. Incluye información acerca de la navegación, los caminos a tomar y también otras entidades en el juego. Toda la información es formateada y preprocesada para un rápido y fácil acceso, y uso del bot. La característica principal del sistema es una representación especial en 3D del entorno del juego. Toda la información suministrada al bot está en alguna manera relacionada o vinculada con esta representación[7].

El AAS tiene propiedades similares al sistema de waypoints. El mismo usa unas casillas en forma de poliedros convexos, llamadas áreas, con una propiedad específica: la complejidad de

navegación para trasladarse de cualquier punto accesible a otro en la misma área es mínima, esto significa que un jugador puede moverse entre dos puntos cualesquiera con solo caminar o nadar en línea recta.

Solo conociendo esta propiedad de cada área no se suministra toda la información requerida para la asignación de un camino y la navegación. Sin embargo, las llamadas accesibilidades (reachabilities) entre áreas pueden ser calculadas. Tal accesibilidad es creada desde un área a otra si el jugador puede trasladarse fácilmente entre dichas áreas. Calcular esas accesibilidades no es del todo difícil porque muchas de las áreas se encuentran adyacentes a las otras. Cuando dos áreas son adyacentes es fácil verificar si realmente un jugador puede trasladarse de un área a otra. Esto no cubre todas las posibles accesibilidades entre áreas, pero calcular las demás accesibilidades es más complejo, pero definitivamente posible[7].

1.8 Lenguaje de desarrollo C++

Existen principalmente tres lenguajes que se utilizan para desarrollar aplicaciones gráficas profesionales en 3D: Lenguaje Ensamblador, C y C++, por ser los que con más velocidad ejecutan el código (menor costo de ejecución del programa). A éstos se ha unido recientemente el Java como una opción para el desarrollo de este tipo de aplicaciones.

Si se estudian las características de este lenguaje, se puede apreciar lo acertado de la elección, dado que C++ es un lenguaje de programación de propósito general, especialmente indicado para la programación de sistemas por su flexibilidad y su potencia. Es uno de los más utilizados en las comunidades de desarrolladores de *software*, incluyendo la programación gráfica.[8]

C++ es la evolución de C adaptada a la programación orientada a objetos. Tiene algunas cuestiones más pulidas como un control más estricto en el manejo de tipos de datos, y otras características que ayudan a la programación libre de errores.[8]

En general puede llegar a ser un lenguaje tan rápido como C (el más rápido después del Lenguaje Ensamblador), sin embargo, si se maneja herencia múltiple, funciones virtuales y polimorfismo en forma inadecuada, o se accede mucho en niveles de profundidad en la llamada a objetos (Objeto1.Objeto2.Objeto3.Objeto4), puede llegar a hacerse un poco más lento, lo cual no es conveniente para una aplicación en tiempo real.[8]

1.9 Tendencias actuales

En el ámbito mundial existen muchos grupos de desarrollo, principalmente en juegos, que trabajan estos métodos de búsqueda de caminos en entornos 3D pero son pocos los que han desarrollado un paquete de herramientas de calidad. Entre los mejores software de este tipo se encuentran: la herramienta AI.implant, el middleware RenderWare A.I. y el paquete de herramientas PathEngine.

El AI.implant es una herramienta de Inteligencia Artificial que le facilita a los desarrolladores de juegos profesionales con un basamento sólido para construir entornos de juegos verdaderamente realistas y dinámicos.

El RenderWare A.I. es la primera solución middleware optimizada multiplataforma de inteligencia artificial, diseñada exclusivamente para el desarrollo de juegos de la próxima generación. El RenderWare A.I. está disponible en forma única, así como también de apoyo completamente a la RenderWare Platform y al RenderWare Studio.

El paradigma escogido por el PathEngine es el resultado de un número de decisiones de diseño. Fue desarrollado por un grupo de trabajo encabezado por Thomas Young, un estudioso del tema en cuestión quien fue su fundador y tiene 10 años de experiencia en la industria de los juegos por computadoras y dado que se hizo un énfasis mayor en esta solución que en las demás se le reservó el próximo subepígrafe.

1.9.1 El SDK PathEngine.

El PathEngine es un conjunto de herramientas sofisticado diseñado para dar soporte al movimiento de los agentes inteligentes en entornos virtuales. Específicamente, se encarga del movimiento del agente sobre un grupo de áreas del terreno predefinidas. La esencia del motor de navegación (pathfinding engine) facilita el agente y al gestor de obstrucciones, la detección de colisiones, la búsqueda de caminos y de métodos que facilitan la navegación del agente a lo largo de los caminos. El conjunto también tiene incluido unas herramientas para definir un grupo de áreas del terreno por donde se haría la navegación de los agentes.[9]

El conjunto de herramientas está enfocado al componente de navegación (pathfinding component) para el movimiento de los agentes inteligentes. Porque este componente está probado históricamente que permaneció aplicable a través de un número de proyectos hasta que la implementación de un alto nivel de inteligencia artificial y el comportamiento del código cambió necesariamente, para rectificar las diferencias en los requisitos con respecto a las características de animación, la inteligencia artificial, etc.

Para una solución completa al movimiento de agentes, los usuarios implementan una capa de comportamiento sobre el PathEngine. La implementación será tan perceptible como especifiquen los requerimientos de la aplicación, pero los ejemplos del SDK incluyen el código que puede adaptarse en gran medida para simplificar la creación de esta capa de comportamiento.[9]

1.9.2 Incursiones en el país y en la universidad

Estos temas no son trabajados por los desarrolladores e investigadores en el ámbito nacional. En la parte de la inteligencia artificial se estudian los métodos de búsqueda pero se aplican en las ramas de la medicina, la economía, etc. que constituyen líneas priorizadas en el desarrollo del país. Por otra parte, la realidad virtual tampoco es muy trabajada, se conoce de empresas como Simpro que se dedica a la investigación y desarrollo de simuladores de vehículos, pero en esta

rama se ha investigado un poco más en la grafica computacional y no así, en la integración con técnicas de inteligencia artificial para lograr un comportamiento más real en los entornos virtuales.

En la universidad existen grupos de trabajo que se dedican al desarrollo de la realidad virtual pero esta integración con la inteligencia artificial se trabajo de un modo artesanal. Los algoritmos se implementan en el momento que haga falta utilizarlos y no existe una librería que facilite este trabajo quedando en manos del ingenio del programador el éxito o el fracaso de la aplicación.

1.10 Conclusiones

Basándose en la fundamentación teórica de los epígrafes anteriores surgió la propuesta del capítulo siguiente. Se abordó una serie de conceptos, métodos y tendencias que les será de utilidad para la comprensión de los problemas de búsqueda de caminos en entornos 3D. Se explicó los algoritmos de búsqueda más utilizados actualmente en el desarrollo de juegos de video, las representaciones del mapa y aplicaciones que colaboraron en alguna medida en la solución propuesta.

CAPÍTULO 2 Descripción de la Solución Propuesta

2.1 Introducción

En este capítulo se hará una descripción de la propuesta a defender, en vista a solucionar la situación actual en el campo de acción. Para entender el problema se realizó un estudio de las principales deficiencias que presentaban los métodos utilizados y posteriormente se analizó las ventajas que proporcionaría una librería de clases en el desarrollo de la navegación de agentes en entornos virtuales. Se muestran los requisitos, tanto los funcionales como los no funcionales, y los casos de uso generados a partir de dichos requisitos funcionales con sus respectivas especificaciones. Esta propuesta se elaboró con el fin de facilitar el desarrollo de la búsqueda de caminos en los entornos virtuales imprimiéndoles más realismo.

2.2 Problema y Situación problémica

Entre los grupos de trabajo existentes en la universidad se encuentra el grupo de Realidad Virtual, el cuál se dedica al desarrollo de aplicaciones con entornos 3D como los paseos virtuales y video juegos. Inicialmente se encontraban en una fase de estudio e investigación de las diferentes técnicas utilizadas en este tipo de aplicaciones visuales hasta que se le encomendó un proyecto productivo llamado ‘UCI Maqueta Digital’ el cual consistía en un paseo virtual de la UCI que serviría para mostrar el Proyecto UCI en los diferentes eventos que tuviese participación la misma. Hace dos años, fue expuesta una primera versión del paseo virtual en la Cumbre de la Información en Ginebra. Después de esta experiencia la dirección de proyectos propuso una segunda versión y se decidió por del grupo de trabajo que se debía mejorar en la dinámica del paseo virtual, que en la segunda versión aparecieran personas caminando y vehículos transitando en las calles de la maqueta virtual. Esta decisión constituiría un gran reto porque en la

universidad no existían proyectos similares, había que investigar cuáles eran las soluciones más utilizadas actualmente y a partir de estas desarrollar una solución factible que satisfaga los objetivos de la investigación. Para lograr dicho propósito se necesitaba entre otras cosas, de técnicas para la navegación y búsqueda de caminos en entornos 3D, de forma tal que simulara la trayectoria de las personas dentro de la universidad. En el mundo existen paquetes de herramientas utilizadas con ese fin, las mismas aceleran el tiempo de desarrollo de las aplicaciones virtuales y logran simular un comportamiento real. Aunque lo más frecuente es que diseña todo un módulo de pathfinding en dependencia de la aplicación, ya sea un paseo virtual, un video juego de acción, o cualquier otro tipo de aplicación virtual. También existen grupos de trabajo con una basta experiencia en el desarrollo de estas técnicas y que si les suministran el entorno en 3D ellos diseñan dichos módulos por encargo. Cualquiera de las variantes anteriores sería muy costosa para elegirla como una posible solución y en el país el tema de las técnicas de navegación en entornos 3D no es lo más trabajado por los desarrolladores de software. Dada esta situación se puede catalogar la falta de un motor de búsqueda de caminos que se encargue de gestionar la navegación de los agentes en un mundo en 3D como el problema a resolver.

2.3 Objetivo de Estudio

El objetivo de estudio en este caso sería las técnicas de inteligencia artificial aplicadas en la navegación de agentes dentro de entornos tridimensionales. Esto incluye tanto los algoritmos de búsqueda de caminos como las diferentes estructuras de datos utilizadas para representar el mundo virtual.

2.4 Campo de acción

Como campo de acción se tiene a los diferentes motores de búsqueda de caminos en entornos 3D debido a que una de sus variantes podría ser la solución al problema.

2.5 Propuesta de sistema

Dada la situación que presentaban las aplicaciones de entorno 3D con el recorrido de caminos, se eligió una solución lo más completa posible que cada vez que se utilice se tenga que modificar poco y que le proporcione al agente un comportamiento de calidad dentro del entorno virtual. Se analizaron los métodos más utilizados en el mundo y se encontró un paquete de herramientas llamado PathEngine SDK, que garantiza la búsqueda de caminos, la evasión de obstáculos, el suavizado de las trayectorias, entre muchas otras funcionalidades que lo hace un paquete de herramientas muy potente. Por otro lado, se estudió un método para la navegación en entornos 3D, muy interesante usado en los juegos para computadoras: “Quake III Arena” y “Doom 3”, dicho método tiene como característica principal que utiliza una representación especial del mundo en 3D, consiste un sistema de áreas, dichas áreas no son más que poliedros convexos, donde cada agente pueden moverse libremente dentro de dicha área, y se calculan los reachabilities que son las propiedades que permiten el acceso de un área a la otra. Mientras el PathEngine representa el mundo de otra manera, el paquete de herramientas representa el mundo mediante una maya del terreno, o sea, un conjunto de caras solapadas que definen el suelo por el que pueden transitar los agentes.

La solución propuesta se basó fundamentalmente en el PathEngine, en su estructura como librería de pathfinding para garantizar la navegabilidad en el entorno 3D. Además, se tomó en cuenta la representación del mundo en 3D que utiliza el “Quake III Arena” para crear las estructuras propias de la propuesta para representar el entorno en 3D. En fin, se quiere lograr un paquete de herramientas con funciones similares al PathEngine pero, en lugar de usar una estructura de dato tipo malla para definir las áreas del terreno despejado por donde pueden moverse los agentes, se haría una representación basada en sistema de percepción de áreas (AAS) que utiliza el “Quake III”. Esto no quiere decir que el modo de definir las áreas es idéntico al desarrollado en el “Quake III”, se tomó la idea de las áreas, aunque, se definieron de un modo

diferente y se mantuvo la idea de los reachabilities entre las áreas. También se debe tener en cuenta que el diseñador tiene que cumplir una serie de requisitos al modelar el entorno virtual para que se logren importar los datos correctamente, y que sean procesados por el motor de búsqueda de caminos. El desarrollador de aplicaciones 3D que tengan agentes monitoreados por la computadora, podrá proporcionarle funcionalidades básicas de navegabilidad mediante el paquete de herramientas, además, no tiene que rehacer un módulo de búsqueda de caminos para cada aplicación que programe, con solo modificar los parámetros claves el motor de búsqueda hace la mayor parte del trabajo.

2.5.1 Representación del mundo

Inicialmente el diseñador debe dibujar las áreas en el mundo en el 3D Max de modo que importador clasificar luego, cada detalle del entorno que necesite la librería. El PathEngine es un paquete de herramientas más elaborado y para este primer paso desarrolló un plugins para el 3D Max que tiene lo necesario para que el diseñador defina la malla del terreno por donde el agente puede moverse, entre otras estructuras de interés que contribuyen a la eficiencia del método. En la propuesta el diseñador debe definir las áreas, cada área tiene una o varias caras que conforman el piso del área, por este piso se mueven los agentes, cada área tiene una o varias caras que conforman la puerta, esta puerta define las caras de área por las cuáles el agente puede pasar a otra área y decide si un agente puede pasar desde un área a otra.

2.5.2 Importador del 3D Max (ABPathEngine Data Creator)

El paquete de herramientas tiene una pequeña aplicación auxiliar cuya función es importar el mundo diseñado en el 3D Studio Max, en formato ASE (3D Max ASCII), y realizar, con estos datos, una serie de operaciones, cuyos resultados serán exportados nuevamente a un fichero en formato APE, que es el formato de fichero que utiliza el motor de búsqueda de caminos. Esta herramienta importa del fichero ASE toda la información relacionada con todos los objetos

creados por el modelador en tiempo de diseño, para cumplir con este objetivo, esta aplicación tiene un cargador de ficheros ASE, que tiene una escena en la cual mantendrá una lista de todos estos objetos, cualquier tipo de objeto. La escena, luego se pasa a un importador del mundo, que es quien sabrá diferenciar cuales son los objetos de cada tipo, estos objetos pueden ser, áreas, pisos, puertas, u otros definidos por el usuario, y los agrupara según sus propiedades.

Al iniciarse la aplicación, esta carga, en tiempo de ejecución, un grupo de clases, almacenadas en “dlls”, lo que se conoce como plug-ins, dichos plug-ins contienen dos tipos de clases: los reachabilities y los reachability creators, estos plug-ins son implementados por el usuario, y es precisamente aquí donde radica, en esencia la flexibilidad del sistema, y la capacidad para adaptarse a los distintos entornos virtuales. El reachability, que no es mas que la definición de un tipo determinado de reachability, o sea, una forma especifica de trasladarse desde un área hasta otra determinada, viene unido de manera univoca a un reachability creator, el ABPathEngine Data Creator tiene definido, hasta el momento y con el objetivo fundamental de realizar las pruebas, el reachability “WalkReachability” (caminar), con su respectivo “WalkReachabilityCreator”. El reachability creator tiene dos funciones principales, determinar si desde un área A, se puede tener acceso a otra área B, trasladándose de la forma especifica, que predefine el reachability que el representa, para cumplir con esta funcionalidad, el reachability creador, tendrá acceso a los datos de todas las áreas definidas en tiempo de diseño, estos datos incluyen coordenadas de los vértices, caras del piso del área, segmentos de rectas que forman los bordes de la misma, puertas, incluyendo las dimensiones de estas, y otros objetos que el usuario puede crear, o no, libremente dentro de las áreas; si el reachability creador determina que desde A, se puede acceder a B, a través de este reachability, entonces su otra función es crear un reachablity de este tipo, y retornar, al final del proceso, un arreglo con todos los reachabilities de ese tipo que fueron creados, para que estos sean agregados a una lista de reachabilities que contiene la clase controladora de la herramienta ABPathEngine Data Creator. El desarrollador puede crear tantos plug-ins como desee, según les sean necesarios en la aplicación que desarrolla en esos momentos, de diversos tipos y estos le servirán cada vez que

haga una aplicación similar, es evidente notar que estos plug-ins pueden ser reutilizados siempre y cuando se creen aplicaciones con características similares a las ya anteriormente desarrolladas.

Una vez que han sido importados todos los objetos de la escena, y agrupados según su tipo, el ABPathEngine Data Creator, calcula, apoyándose en los reachabilities creators, cargados en tiempo de ejecución a través del sistema de plug-ins, todos los reachabilities. Todos los objetos contenidos en el mundo, tienen, entre otras cosas, la funcionalidad de exportar sus datos, aquellos que sean útiles y necesarios al motor de búsqueda de caminos (ABPathEngine), a un fichero de texto con extensión APE (Areas PathEngine), y con una estructura definida. Este tipo de fichero se creó específicamente para suministrarle la información del entorno a la librería.

2.5.3 La librería ABPathEngine (Areas Based PathEngine)

Esta librería de clases constituye la parte principal del paquete de herramientas, la misma tiene la responsabilidad de ofrecer al desarrollador, ciertas funciones que son necesarias para implementar la navegación de los Bots en cualquier entorno virtual.

Primeramente, el usuario debe crear una instancia del la librería (ABPathEngine), al inicializar esta instancia debe pasar como parámetro la dirección del fichero con extensión .APE, que debe haber sido creado, previamente, con el ABPathEngine Data Creador, y que contiene la información relativa a la estructura, diseño, dimensiones, y ubicación de los objetos que puedan resultar relevantes a la hora de correr los algoritmos de búsqueda de caminos, al inicializarse el mismo la librería se encarga de cargar una serie de plugins, que deben encontrarse en una carpeta con nombre “plugins”, en la misma ubicación que la dll (Dynamic Link Libray) que contiene la librería de clases, y que representan los reachabilities creados por el usuario. Estos reachabilities, se diferencian de los explicados en el epígrafe anterior, por el lenguaje de programación en que están escritos, la aplicación auxiliar fue implementada en C# en está primera versión, mientras que la librería está implementada en C++. Automáticamente después

de importados los datos, y cargados los plugins, se crea por cada reachability del fichero un homólogo en la librería.

La librería tiene implementados, en esta versión, dos algoritmos de búsqueda: el Dijkstra's y el A* (A Star), que son los encargados de buscar los caminos para que los agentes controlados por la computadora puedan moverse de un área a otra. El algoritmo Dijkstra se ha implementado para buscar el camino hasta un objeto de un tipo específico, debido a las características del algoritmo, y se obtendría como resultado el camino al más cercano de los objetos de ese tipo. Mientras que el A* se implemento para hacer las búsquedas que van dirigidas a encontrar el camino mas corto, o adecuado, porque no siempre es el mas corto, hasta un punto u objeto especifico (objeto, instancia, no tipo de objeto).

2.5.4 El ABPathEngine vs. Métodos actuales

El paquete de herramientas ABPathEngine, como fue nombrada la solución propuesta, puede utilizarse en las aplicaciones con el objetivo de proporcionarle la navegabilidad dentro del entorno virtual al agente monitoreado por la computadora, como una buena variante. Basado en el sistema de áreas usado en los juegos para computadoras como el "Quake III Arena" y el "Doom 3", el paquete pone en manos del desarrollador de la aplicación una herramienta flexible que le proporciona más libertad para crear nuevas formas de moverse por el mundo tridimensional. El paquete de herramientas PathEngine es muy sofisticado pero cuesta 4000 euros una versión con lo básico solamente y la versión completa cuesta 13000 euros, esta cantidad de dinero se puede ahorrar si se desarrolla una variante propia como nuestra propuesta.

Esta propuesta no se propone superar a los paquetes existentes en el mundo pero si podría ahorrarle una buena cantidad de dinero a la universidad pues esta solución sería un recurso disponible para todos. Si se sigue en el perfeccionamiento de esta herramienta podría llegar al nivel de las existentes en la actualidad, lo cual se encuentra dentro de nuestro objetivos para el futuro, estas herramientas han sido creadas, fundamentalmente, por el interés que sienten los

investigadores, por incrementar el nivel de competitividad de Cuba en el mercado de los video juegos, simuladores, y paseos virtuales, que aunque muchos vean los video juegos como algo sin mucha importancia, o de poca seriedad, no es menos cierto que en la actualidad, es la industria de desarrollo de estos, una de las que mas ganancias obtiene cada año a nivel mundial, por lo que, introducirnos en este mercado podría dar una oportunidad realmente tangible, a Cuba, para lograr un aumento substancial de la economía.

2.6 Modelo de Dominio

A continuación se presenta el Modelo de Dominio del paquete de herramientas donde se capturan los objetos más importantes en el contexto del sistema, estos objetos son eventos y entidades que describen la funcionalidad del sistema. La representación gráfica se encuentra en la Tabla.

Conceptos más importantes:

- Agente: jugador que es monitoreado por la computadora.
- Searcher: algoritmo de búsqueda de caminos.
- Área: estructura de dato definida para representar el mundo.
- Camino: camino para moverse de una posición a otra en el mundo.
- Cara: cara de una figura geométrica (puede ser de un área, de una puerta).
- Escena: mapa que se genera a partir del mapa diseñado en el 3dMax.
- Puerta: caras del área habilitadas para entrar o salir de la misma.

- Mundo: maneja todo lo relacionado con el entorno 3D por parte del Data Creator.

- Mapa ASE: mapa exportado del 3D Max.
- Mapa APE: mapa que utiliza el motor de búsqueda.
- Piso: caras de un área por donde pueden caminar los agentes.

- Reachability: modo de acceder de un área a otra.
- Path Engine: motor de búsqueda de caminos.

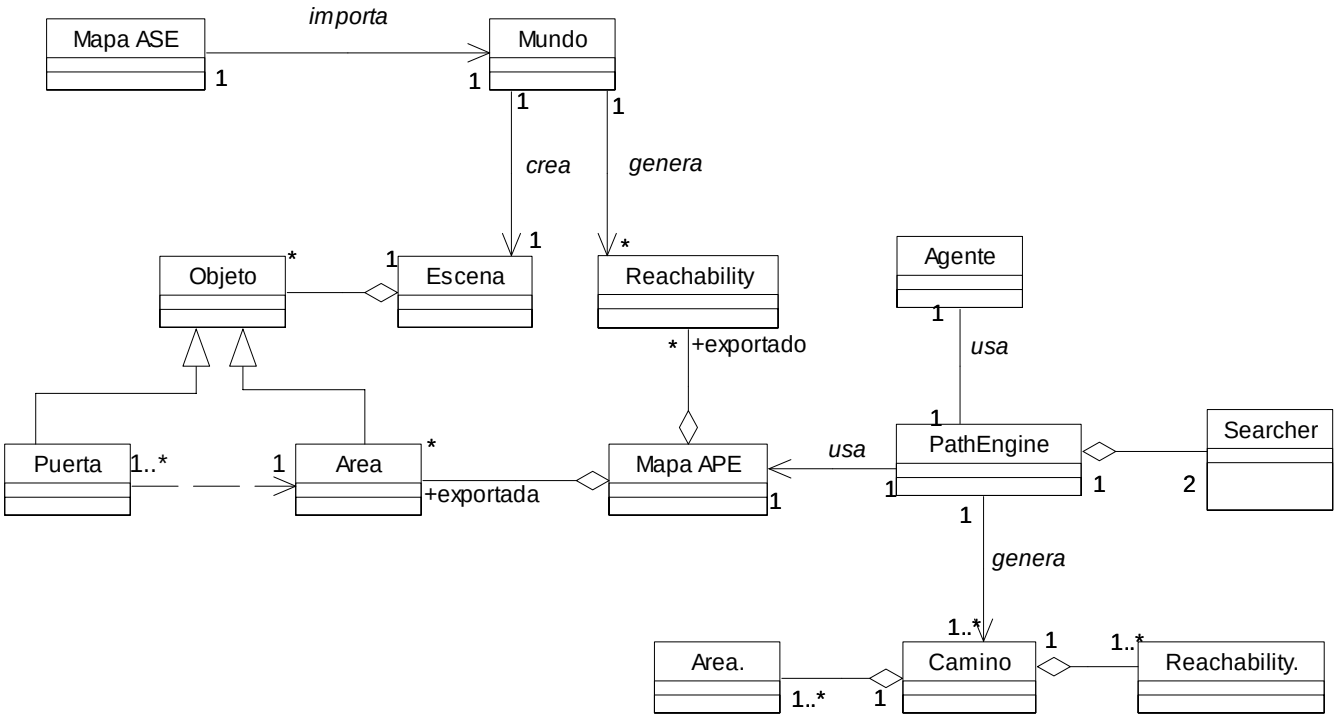

Figura 2 - Modelo de Dominio

2.7 Requisitos Funcionales

R.1 Importar fichero .ASE.

1.1 Cargar fichero .ASE.

1.2 Crear escena con todos los objetos que se definan del mundo virtual cargado.

R.2 Generar Reachabilities

2.1 Crear reachabilities

2.2 Calcular reachabilities y agregarlos a la escena.

R.3 Exportar fichero .APE.

3.1 Guardar toda la información de la escena en un fichero.

R.4 Importar fichero .APE.

4.1 Cargar la información del fichero .APE para el motor de búsqueda de caminos.

R.5 Buscar Camino.

5.1 Buscar el camino mínimo entre un punto inicial y un punto final.

5.2 Posibilitar que se recalcule el camino sin terminar el recorrido de un camino anterior.

5.3 Buscar un camino hasta un objeto de determinado tipo que más cercano esté.

R.6 Evitar Entidad.

6.1 Evitar un área determinada en el transcurso del camino.

6.2 Evitar un reachability.

6.3 Evitar una puerta.

2.8 Requisitos no funcionales

Usabilidad: Los futuros usuarios serán programadores. El producto debe estar concebido para que el usuario piense en qué desea hacer y no cómo hacerlo, por lo que éste requerimiento debe estar presente en alto grado en el producto final. Puede ser usado por varias herramientas de desarrollo como el Visual Studio.NET y el C++Builder (para

esta primera versión debe instalarse el .NET Framework 2.0 para utilizar el Data Creador, la librería no depende del mismo).

Soporte: En una versión inicial será compatible con la plataforma Windows, pero debe estar preparado para que con rápidas modificaciones pueda migrar hacia Linux.

Legales: Se regirá por las normas ISO 9000.

Software: Sistema operativo Windows 2000 o superior.

Hardware: Requerirá de computadoras Pentium 3, con 256 de memoria RAM y 1.5 GHz.

2.9 Definición de los casos de uso

En este epígrafe se realiza una explicación detallada de los principales casos de uso del sistema. Esto se realizará utilizando el UML. Se capturarán los requisitos funcionales y se mostrarán en un diagrama de casos de uso. Un caso de uso es una descripción narrativa de un proceso de dominio.

2.9.1 Definición de Actores

En esta sección se describen los posibles actores del sistema a desarrollar y se conciben, a través de la agrupación de requisitos funcionales anteriormente hallados los posibles resultados de valor que pueda brindar a sus actores, o lo que es lo mismo, los casos de uso del sistema.

Tabla 1 - Actor del Sistema

Actores	Justificación
Usuario	Es el que se beneficiará con las funcionalidades que brinda la biblioteca de clases: cargar fichero, buscar camino

Los casos de uso son la base del diseño del software, ya que a partir de ellos se derivan muchos de los posteriores procesos dentro del desarrollo del software.

Tabla 2 - Listado de casos de uso.

CU – 1	Importar fichero .ASE
Actor	Usuario
Descripción	El CU comienza cuando el usuario desea importar el mapa diseñado en el 3D Studio Max.
Referencia	R.1
CU – 2	Generar Reachabilities
Actor	Usuario
Descripción	El CU comienza cuando el usuario desea calcular todos los reachabilities posibles entre las áreas del mapa.
Referencia	R.2
CU – 3	Exportar fichero .APE
Actor	Usuario
Descripción	El CU comienza cuando el usuario exporta la escena con la trabajara la librería posteriormente.
Referencia	R.3

CU – 4	Importar fichero .APE
Actor	Usuario
Descripción	El CU comienza cuando el usuario importa el mapa en formato APE para interactuar con la librería.
Referencia	R.4
CU – 5	Buscar Camino Mínimo
Actor	Usuario
Descripción	El CU comienza cuando el usuario necesita saber el camino mínimo de un lugar inicial a un lugar destino.
Referencia	R 5.1
CU – 6	Buscar Camino Mínimo Parcial
Actor	Usuario
Descripción	El CU comienza cuando el usuario necesita obtener una parte del camino mínimo de un lugar inicial a un lugar destino.
Referencia	R 5.2, CU – 5 <<*include*>>
CU – 7	Buscar Objeto Más Cercano
Actor	Usuario
Descripción	El CU comienza cuando el usuario necesita obtener el camino al objeto más cercano, cuyo tipo se especifica con anterioridad.
Referencia	R 5.3, CU – 5 <<*include*>>

2.9.2 Diagrama de Casos de Uso

Los casos de uso son fragmentos de funcionalidad del sistema. En ellos se describe la secuencia determinada de eventos que realiza un actor en interacción con la aplicación. En el ANEXO se podrán ver el Diagrama de Casos de Uso (Anexos Figura 6).

2.9.3 Casos de Uso expandidos

En este epígrafe se describen los casos de uso del sistema.

Tabla 3 - Descripción de CU “Importar fichero ASE”.

Caso de uso		
CU – 1	Importar el fichero .ASE	
Propósito	Importar el mapa diseñado en el 3D Studio Max	
Actores: Usuario		
Resumen: La aplicación importa el fichero .ASE a un nuevo formato entendible a la misma para las posteriores operaciones.		
Referencias	R.1	
Acción del actor		Respuesta del sistema
1. El usuario indica ‘Import’ en el menú 2. El usuario elige el fichero que desea importar.		1.1 Muestra una caja de diálogo para que el usuario escoja el fichero que desea importar. 2.1 Verifica que el fichero sea del tipo correcto. 2.2 Crea una nueva escena. 2.3 Carga los objetos la escena creada.
Flujo alternativo: Fichero de tipo incorrecto		
Acción del actor		Respuesta del sistema

1. El usuario indica ‘Import’ en el menú	1.1 Muestra mensaje de advertencia y termina el caso de uso.

Tabla 4 - Descripción de CU “Generar Reachabilities”.

Caso de uso		
CU – 2	Generar Reachabilities	
Propósito	Calcular todos los posibles reachabilities en el mapa importado.	
Actores: Usuario		
Resumen: La aplicación calcula los diferentes tipos de reachabilities del mapa importado para generar el .APE		
Referencias	R.2	
Acción del actor		Respuesta del sistema
1. El usuario indica ‘Execute’ en el menú.		1.1. Verifica los ReachabilityCreator disponibles. 1.2 Calcula los reachabilities para cada tipo de ReachabilityCreator.

Tabla 5 - Descripción de CU “Exportar fichero APE”.

Caso de uso		
CU – 3	Exportar fichero .APE	
Propósito	Exportar el mapa definido por la aplicación con los reachabilities calculados	
Actores: Usuario		
Resumen: Exporta en un fichero las áreas definidas por la aplicación con los reachabilities calculados que será usado por la Librería.		
Referencias	R.3	
Acción del actor		Respuesta del sistema

1. El usuario indica ‘Export’ en el menú. 2. El usuario elige el camino dónde desea exportar el fichero.	1.1 Muestra una caja de diálogo para que el usuario escoja el fichero que desea exportar. 2.1 Crea un fichero .APE 2.2 Guarda la información de los objetos de la escena en el fichero.

Tabla 6 - Descripción de CU “Importar fichero APE”.

Caso de uso		
CU – 4	Importar fichero .APE	
Propósito	Cargar toda la información del fichero APE	
Actores: Usuario		
Resumen: Importa todas las áreas y los reachabilities del fichero APE.		
Referencias	R.4	
Acción del actor		Respuesta del sistema
1. Solicita cargar un fichero con un camino dado.		1.1 Se verifica la existencia del fichero y si este tiene una extensión es válida. 1.2 Se crea un controlador de ficheros para que haga la carga de información en un buffer. 1.3 Se carga el encabezamiento del fichero. 1.4 Se carga los datos del entorno con la información del encabezamiento del fichero. 1.5 Se crean y se adicionan a la colección todos los reachabilities.

	1.6 Se crean y se adicionan a la colección todas las áreas.
Poscondiciones	Creados los reachabilities y las de la escena.

Tabla 7 - Descripción de CU “Buscar camino”.

Caso de uso	
CU – 5	Buscar Camino
Propósito	Buscar el camino mínimo entre un lugar inicial y un lugar destino.
Actores: Usuario	
Resumen: Dado un área inicial y un área final determina el camino más corto.	
Referencias	R 5.1

Acción del actor	Respuesta del sistema
1. Se solicita buscar camino dado un destino y un agente.	1.1 Se crea un buscador AStar. 1.2 Se calcula la matriz de costo para el agente. 1.3 Se recorre las áreas hasta encontrar el punto destino. 1.4 Se crea un camino.

Tabla 8 - Descripción de CU “Buscar camino parcial”.

Caso de uso	
CU – 6	Buscar Camino Parcial
Propósito	Buscar el próximo paso que se daría para llegar hasta un lugar determinado
Actores: Usuario	

Resumen: Busca el camino mínimo para llegar hasta un lugar determinado para indicar el próximo paso que se debe dar para alcanzar dicho destino.	
Referencias	R 5.2, CU – 5 <<*include*>>
Acción del actor	Respuesta del sistema
1. Se solicita crear un camino parcial dado un punto destino y un agente.	1.1 Se crea un buscador AStar. 1.2 Se calcula la matriz de costo para el agente. 1.3 Se inicia el CU – 5 hasta encontrar el punto destino. 1.4 Se crea un camino parcial.

Tabla 9 - Descripción de CU “Buscar objeto más cercano”.

Caso de uso	
CU – 7	Buscar Objeto Más Cercano
Propósito	Buscar un objeto dado un tipo de objeto.
Actores: Usuario	
Resumen: Buscar el camino para llegar al objeto más cercano dado un determinado tipo.	
Referencias	R 5.3, CU – 5 <<*include*>>
Acción del actor	Respuesta del sistema
1. Se solicita crear un camino hasta el objeto más cercano con el agente, un punto destino y el tipo de objeto.	1.1 Se crea un buscador Dijkstra. 1.2 Se calcula la matriz de costo para el agente. 1.3 Se recorre las áreas hasta encontrar el punto destino. 1.4 Se crea un camino

2.9 Conclusiones

En el presente capítulo se hizo un esbozo detallado de la propuesta de solución que brinda este trabajo para cumplir con los objetivos planteados. Se ha analizado cada una de las partes que tiene la herramienta, los ficheros con los que trabaja, las facilidades que da. Se explicó cómo es el trabajo con ella, de qué manera maneja los datos. Se enumeraron los requisitos funcionales y los no funcionales, además de que se explicó de manera muy detallada cada uno de los casos de uso que se sacaron para satisfacer los requisitos funcionales.

CAPÍTULO 3 Construcción de la Solución Propuesta.

3.1 Introducción

En la elaboración de este software se trató de realizar un diseño lo más claro posible para su posterior programación. Era necesario que le brindara al usuario comodidades para lograr una fácil interacción con la biblioteca de clases. Además, se tuvo muy en cuenta, su posibilidad de escalabilidad puesto que esto es solo una primera versión, que estará continuamente sujeta a cambios para mejorar su eficiencia, brindando más servicios al usuario y mejorando los ya existentes.

3.2 Patrones de diseño.

“En la terminología de objetos, el patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas. Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas. (…) Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería de software. Todo lo contrario: intentan codificar el conocimiento, las expresiones y los principios *ya existentes*: cuanto más trillados y generalizados, tanto mejor.”[10]

Es el diseño de la aplicación se tuvieron en cuenta principalmente los patrones Experto, y Creador. El primero plantea que siempre se debe asignar una responsabilidad al experto en información, o sea, la clase con toda la información necesaria para llevarla a cabo. El segundo expresa que la responsabilidad de crear una instancia de una determinada clase debe asignarse a otra clase, siempre que esta agregue, contenga, registre o utilice específicamente los objetos de aquella.

3.3 Diagramas de Clases

En este epígrafe se verá todo lo referente a los diagramas de clases que componen el sistema. El diagrama de clases es uno de los elementos más importantes dentro de un proyecto de software, ya que brinda una visión bastante completa de todo el sistema, mostrando todas las clases con sus métodos y atributos, así como las relaciones entre estas. Un software se puede dividir por paquetes, ese es el caso del sistema que se propone en este trabajo.

El sistema propuesto consta de cinco paquetes, el ABPathEngine Data Creator, el Data Creator Interface, el ABPathEngine y el ABPathEngine Interface, aunque se utilizó otro paquete que es una librería externa, el Math, para las operaciones con vectores. En el paquete ABPathEngine se encuentran todas las clases de la biblioteca, es decir, las que ejecutan todas las operaciones que hacen que el sistema cumpla con los objetivos planteados. En el paquete ABPathEngine Data Creator se tiene las clases relacionadas con la aplicación auxiliar que importa los datos del 3D Max y crea las estructuras de datos que utiliza la biblioteca de clases posteriormente. Los paquetes Data Creador Interface y ABPathEngine Interface están constituidos por clases que utilizan los paquetes ABPathEngine Data Creator y ABPathEngine respectivamente, las cuáles se agruparon para aplicar la reutilización del código. En el ANEXO se encuentran los diagramas de clases de cada caso de uso.

3.4 Diagrama de Paquetes

El diagrama de paquetes lo que muestra es la relación entre todos los paquetes antes expuestos. Comprender esta relación es muy importante ya que es una forma de ver cómo funciona todo el sistema de una manera más general.

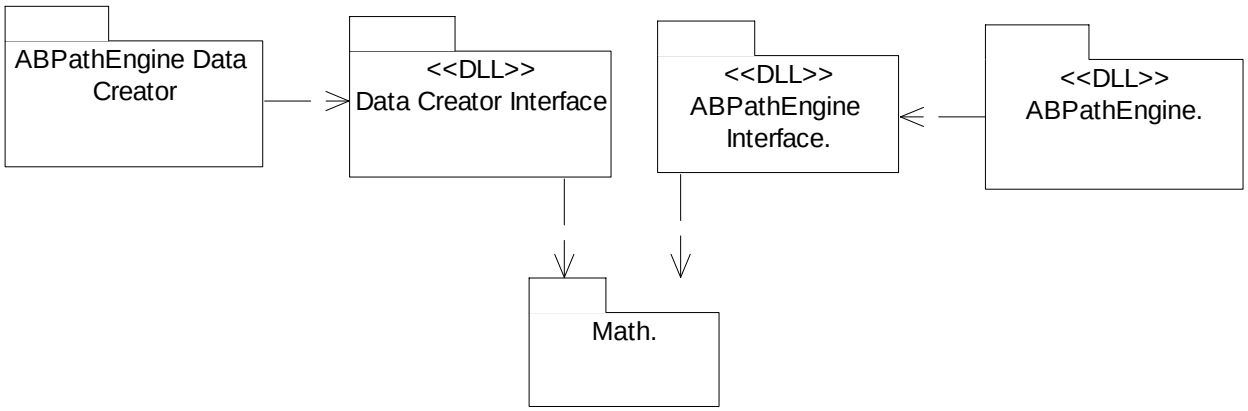

Figura 3 - Diagrama de paquetes.

3.5 Descripción de las Clases

En este epígrafe se realiza una descripción detallada de todas las clases del sistema, se dividieron por paquetes para ser más específicos.

3.5.1 Paquete ABPathEngine Data Creator.

Tabla 10- Clase “cWorld”.

Nombre: cWorld	
Tipo de clase	
Atributo	Tipo
Areas	ArrayList
Reachabilities	ArrayList
ReachabilityCreator	ArrayList

Importer	cWorldImporter
Exporter	cAPEExporter
Para cada responsabilidad:	
Nombre:	Import(string)
Descripción:	Importa las áreas y las adiciona en la colección de áreas.
Nombre:	Export(string)
Descripción:	Exporta las áreas y los reachabilities a un fichero.
Nombre:	AddReachabilityCreator(object)
Descripción:	Adiciona un objeto ReachabilityCreator a la colección de ReachabilityCreators
Nombre:	CreateReachabilities()
Descripción:	Recorre todas las áreas para cada ReachabilityCreator, crea los reachabilities y los adiciona a una colección de Reachabilities.

Tabla 11 - Clase “cWorldImporter”.

Nombre: cWorldImporter	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	Import(string, ArrayList)
Descripción:	Crea una escena e importa una colección de áreas a dicha escena.
Nombre:	NextTokens(string)
Descrpción:	Devuelve los tokens dado un número de línea del fichero.

Tabla 12 - Clase “cAPEExporter”.

Nombre: cAPEExporter	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	Export(string, ArrayList)
Descripción:	Exporta todas las areas y los reachabilities a un fichero.

Tabla 13 - Clase “cASEFileLoader”.

Nombre: cASEFileLoader	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	Import(string)
Descripción:	Llama al método GetASEEscene(string) pasándole como parámetro el nombre del fichero.
Nombre:	GetASEEscene(string)
Descripción:	Crea una escena y carga los datos de un fichero ASE en dicha escena.
Nombre:	NextTokens(string)
Descripción:	Devuelve el próximo tokens, se utiliza para leer del fichero.

Tabla 14 - Clase “cASEObject”.

Nombre: cASEObject	
Tipo de clase	
Atributo	Tipo
aVertices	cVector3<float>[ ]
aFaces	cFace [ ]
aFloorFaces	cFace [ ]
Name	string
NumberOfFloorFace	byte
Para cada responsabilidad:	
Nombre:	
Descripción:	

3.5.2 Paquete Data Creator Interface.

Tabla 15 – “Clase cArea”.

Nombre: cArea	
Tipo de clase	
Atributo	Tipo
Name	string
Type	string
BObject	cBoundingObject
Faces	cFaces [ ]
FloorFaces	cFaces [ ]
Doors	cDoor [ ]
areaEdges	cEdge [ ]
Para cada responsabilidad:	
Nombre:	Export(StreamWriter)
Descripción:	Exporta los datos de esta área a un fichero.

Tabla 16 - Clase “cBoundingObject”.

Nombre: cBoundingObject	
Tipo de clase	
Atributo	Tipo
aVertices	cVector3<float> [ ]
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 17 - Clase “cBoundingHull”.

Nombre: cBoundingHull	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 18 - Clase “cDoor”.

Nombre: cDoor	
Tipo de clase	
Atributo	Tipo
Name	string
DoorFaces	cFace [ ]
DoorVertices	cVector3<float> [ ]
Para cada responsabilidad:	

Nombre:	Export(StreamWriter)
Descripción:	Exporta los datos de esta puerta a un fichero.

Tabla 19 - Clase “cFace”.

Nombre: cFace	
Tipo de clase	
Atributo	Tipo
Vertices	cVector3<float> [ ]
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 20 – “Clase cEdge”.

Nombre: cEdge	
Tipo de clase	
Atributo	Tipo
iVertices	Int [ ]
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 21 - Clase “cReachability”.

Nombre: cReachability	
Tipo de clase	
Atributo	Tipo
Type	string
Cost	int
AreaFromName	string
AreaToName	string
PointFrom	cVector3<float>
PointTo	cVector3<float>
Para cada responsabilidad:	
Nombre:	Export(StreamWriter)
Descripción:	Exporta los datos del reachability en un fichero

Tabla 22 - Clase “cReachability”.

Nombre: cReachabilityCreator	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	CreateReachabilities(cArea [ ])
Descripción:	Crea todos los reachabilities de este tipo y los adiciona a una colección de reachabilities.

3.5.3 Paquete ABPathEngine.

Tabla 23 - Clase “peAgente”.

Nombre: peAgente	
Tipo de clase	
Atributo	Tipo
State	peState*
PosibleState	peState [ ]
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 24 - Clase “cAPEImporter”.

Nombre: cAPEImporter	
Tipo de clase	
Atributo	Tipo
lpAreas	VECTOR<cArea*>*
lpReachabilities	VECTOR<cReachability*>*
Para cada responsabilidad:	
Nombre:	Import(string, VECTOR<cArea*>*, VECTOR<cReachability*>*, cPluginsWrapper*)
Descripción:	Importa los datos de un fichero APE.

Tabla 25 - Clase “cPluginsWrapper”.

Nombre: cPluginsWrapper	
Tipo de clase	
Atributo	Tipo
Plugins	VECTOR<Plugin>
Para cada responsabilidad:	
Nombre:	CreateInstance(char*)
Descripción:	Crea una instancia de plugins cargado.

Tabla 26 - Clase “pePath”.

Nombre: pePath	
Tipo de clase	
Atributo	Tipo
PathSteps	pePathStep [ ]
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 27 - Clase “pePathStep”.

Nombre: pePathStep	
Tipo de clase	
Atributo	Tipo
NextPoint	cVector3f
Action	Int
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 28 - Clase “peState”.

Nombre: peState	
Tipo de clase	
Atributo	Tipo
BoundingObject	peBoundingObject*
Description	peDescription*
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 29 - Clase “peDescription”.

Nombre: peDescription	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 30 - Clase “cPathEngine”.

Nombre: cPathEngine	
Tipo de clase	
Atributo	Tipo
ApeImporter	cApeImporter*

Plugins	cPluginsWrapper*
AStarSearcher	cAStarSearcher*
DijkstraSearcher	cDijkstraSearcher*
plAreas	VECTOR<cArea*>*
plReachabilities	VECTOR<cReachability*>*
plEntities	VECTOR<peEntity>*
Para cada responsabilidad:	
Nombre:	Import(char*)
Descripción:	Importa los datos del fichero APE.
Nombre:	InsertEntity(peEntity*)
Descripción:	Inserta una entidad en el entorno virtual.
Nombre:	CreatePath(peAgent*, cVector3f)
Descripción:	Busca un camino dado un punto destino.
Nombre:	CreatePartialPath(peAgent*, cVector3f)
Descripción:	Busca un camino dado un punto destino y retorna el primer paso del recorrido.
Nombre:	CreatePathToClosestItem(peAgent*, cVector3f)
Descripción:	Busca el camino hasta el objeto más cercano dado un tipo de objeto
Nombre:	LoadPlugins()
Descripción:	Carga los plugins de los reachabilities.
Nombre:	GetPluginPath()
Descripción:	Devuelve la dirección del plugins.
Nombre:	EntityCanBeThere(peEntity*)
Descripción:	Devuelve si una entidad puede insertarse en un lugar determinado.

Tabla 31 – “Clase cAStarSearcher”.

Nombre: cAStarSearcher	
Tipo de clase	
Atributo	Tipo

Para cada responsabilidad:	
Nombre:	CalculateCostMatrix()
Descripción:	Calcula la matriz de costo para los datos de un agente específico.
Nombre:	CreatePath()
Descripción:	Encuentra un camino del lugar donde esta situado el agente hasta un punto que se le pasa como parámetro
Nombre:	CreatePartialPath()
Description:	Encuentra un camino como el método anterior pero con la diferencia que devuelve el próximo paso que hay que dar si elegimos ese camino.

Tabla 32 - Clase “cDijkstraSearcher”.

Nombre: cDijkstraSearcher	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	CalculateCostMatrix()
Descripción:	Calcula la matriz de costo para los datos de un agente específico.
Nombre:	CreatePathToClosestItem()
Descripción:	Encuentra el camino hasta el objeto más cercano dado un tipo de objeto

3.5.3 ABPathEngine Interface.

Tabla 33 - Clase “cArea”.

Nombre: cArea	
Tipo de clase	
Atributo	Tipo

Name	const char*
VerticesNumber	int
FacesNumber	int
aFaces	VECTOR<cFace>
aDoors	VECTOR<cDoor>
Para cada responsabilidad:	
Nombre:	SetVerticesNumber(int)
Descripción:	Actualiza el valor del número de vértices.
Nombre:	SetFaceNumber(int)
Descripción:	Actualiza el valor del número de faces (caras).
Nombre:	AddVertex(int, float, float, float)
Descripción:	Adiciona vértices.
Nombre:	isBetween(cVector3f)
Descripción:	Verifica si el punto que se pasó como parámetro está dentro del área definida.
Nombre:	Destroy()
Descripción:	Destruye la instancia de la clase y libera memoria.

Tabla 34 - Clase “cDoor”.

Nombre: cDoor	
Tipo de clase	
Atributo	Tipo
aVertices	VECTOR<cVector3f>
aFaces	VECTOR<cFace>
Para cada responsabilidad:	
Nombre:	Destroy()
Descripción:	Destruye el la instancia y libera memoria.

Tabla 35 - Clase “cFace”.

Nombre: cFace	
Tipo de clase	
Atributo	Tipo
iVertices	Int [3]
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 36 - Clase “cReachability”.

Nombre: cReachability	
Tipo de clase	
Atributo	Tipo
Cost	int
SourceAreaName	char*
TargetAreaName	char*
StartPoit	cVector3f
TargePoint	cVector3f
Para cada responsabilidad:	
Nombre:	Import(char*, long&)
Descripción:	Importa los datos de un fichero.
Nombre:	Destroy()
Descripción:	Destruye la instancia y libera memoria.

Tabla 37 – Clase “peBoundingObject”.

Nombre: peBoundingObject	
Tipo de clase	
Atributo	Tipo
aVertices	VECTOR<cVector3f>
Para cada responsabilidad:	
Nombre:	isBetween(cVector3f)
Descripción:	Verifica que el punto que se pasó como parámetro se encuentra dentro del Bounding Object.

Tabla 38 - Clase “peBoundingBox”.

Nombre: peBoundingBox	
Tipo de clase	
Atributo	Tipo
MaxVector	cVector3f
MinVector	cVectro3f
Para cada responsabilidad:	
Nombre:	isBetween(cVector3f)
Descripción:	Verifica que el punto que se pasó como parámetro se encuentra dentro del Bounding Box.

Tabla 39 - Clase “peBoundingHull”.

Nombre: peBoundingHull	
Tipo de clase	
Atributo	Tipo
aNormals	VECTOR<cVector3f>

Para cada responsabilidad:	
Nombre:	isBetween(cVectro3f)
Descripción:	Verifica que el punto que se pasó como parámetro se encuentra dentro del Bounding Hull.

Tabla 40 - Clase “peBoundingPlane”.

Nombre: peBoundingPlane	
Tipo de clase	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	isBetween(cVector3f)
Descripción:	Verifica que el punto que se pasó como parámetro se encuentra dentro del Bounding Plane.

Tabla 41 - Clase “PathEngineInterface”.

Nombre: cPathEngineInterface	
Tipo de clase	
Atributo	Tipo
nPathEngineInterface	int
Para cada responsabilidad:	
Nombre:	
Descripción:	

Tabla 42 - Clase “Entity”.

Nombre: peEntity	
Tipo de clase	
Atributo	Tipo

Position	cVector3f
Type	char*
pBoundinObject	peBoundingObject*
Para cada responsabilidad:	
Nombre:	SetBoundingObject(peBoundingObject*)
Descripción:	Actualiza el “Bounding Object” al que pertenece.
Nombre:	SetType(char*)
Descripción:	Actualiza el tipo del agente.
Nombre:	SetPosition(float, float, float)
Descripción:	Actualiza la posición del agente.

3.6 Diagramas de Secuencia.

Los diagramas de secuencia brindan un modelo de cómo interactúa el usuario con el sistema y de cómo interactúan los objetos del sistema entre sí. A continuación explicará el flujo de sucesos de cada uno de los diagramas de secuencia, los mismos se pueden ver en el ANEXO.

3.6.1 Flujo de Sucesos “Importar fichero .ASE”.

El caso de uso “Importar fichero ASE” comienza de la siguiente manera. Cuando el usuario desea cargar el fichero ASE, que no es más que el mapa diseñado en el 3d Max, selecciona un fichero del tipo correcto y manda a la aplicación auxiliar (PathEngine Data Creator) a que lo importe, este es el primer mensaje y se inicia del usuario hacia a la aplicación. La aplicación tiene un *World* (mundo), la interfaz le ordena al World, que es quien controla la mayoría de los sucesos en el PathEngine Data Creador, que importe el fichero y le proporciona el nombre del fichero a importar. Entonces, este crea una lista de Areas y un Importer (importador), que es el responsable de importar los ficheros, y le facilita el nombre del fichero a importar y el arreglo de áreas creado. El Importer a su vez, crea un una Escene (escena), crea un fLoader (cargador de

fichero) y le ordena a este último que importe el fichero y le pasa el nombre del fichero. Por último el fLoader se encarga de cargar la Escene, en este proceso se comienza la lectura del fichero, primeramente se crea una lista de pObject y a medida que se lee el fichero se llenan los datos del objeto y se adiciona a la lista de pObject hasta que queda conformada la *Escene*. Esta escena se le envía al Importer para que clasifique cada pObject y los adicione en las listas de *areas* y *doors* (puertas) según corresponda.

3.6.2 Flujo de Sucesos “Generar Reachabilities”.

En el caso de uso “Generar Reachabilities” el usuario indica en la interfaz de la aplicación que desea crear los reachabilities y seguidamente se envía un mensaje al World. El World tiene una lista de Reachabilities, una lista de ReachabilityCreators y una lista de Areas inicialmente vacías, antes de comenzar a crear reachabilities el World carga los plugins de los reachabilities definidos en otro momento y son adicionados a la lista de ReachabilityCreators. Luego comienza por cada ReachabilityCreator a verificar entre las áreas si se cumple las condiciones particulares respectivas para crear el rechability, de ser posible se crea y los adiciona a la lista de Reachabilities.

3.6.3 Flujo de Sucesos “Exportar fichero APE”.

En el caso de uso “Exportar APE” el usuario indica que desea exportar la escena y especifica la dirección. La interfaz de la aplicación manda a exportar al World y este a su vez crea un Exporter, que es el responsable de exportar todos los datos de la escena a un fichero. El Exporter recibe del World una lista de Areas y una de Reachabilities del World, posteriormente crea un fichero, y a medida que recorre las listas de Areas y Reachabilities le manda a exporten sus datos al fichero.

3.6.4 Flujo de Sucesos “Importar fichero APE”.

El caso de uso “Importar fichero APE” se inicia cuando el usuario crea una instancia de la clase PathEngine y llama a su constructor, pasándole como parámetro la dirección del fichero .APE con el cual va a trabajar su aplicación. Dicho constructor crea una instancia de la clase APEImporter, quien se encarga de crear las áreas y cargar del fichero sus datos, y crear además los reachabilities, y hacer la llamada al método Import, que tiene todo Reachability, pasándole las dirección del fichero, y la posición del fichero a partir de la cual debe comenzar a leer sus datos.

3.6.5 Flujo de Sucesos “Buscar Camino Mínimo”.

Para el usuario usar la librería debe incluir las dll y los .lib, luego crea una instancia de las clase PathEngine que esta dentro de la librería ABPathEngine, para inicializar este objeto se llama al constructor, pasándole como parámetro, la dirección del fichero .APE. Después de tener ese objeto de tipo PathEngine inicializado, el puede llamar a las funciones de este, en este caso sería:

```
pePath CreatePath(peAgent* Agent, cVector3f TargetPoint)
```

Debe pasar como dato, un Agente (un agente se crea para representar un objeto que debe saber navegar por el entorno, y que es controlado por la computadora) y un Punto, en caso de que sea un entorno en 2D se deja los valores de z en 0. La función crea un buscador, AStarSearcher y le pide buscar el camino, para esto se le pasa como parámetros el agente y el punto, pero además, la lista de areas y los reachabilities, ya hecho esto, el searcher crea una matriz de costo, basándose en las areas y los reachabilities, y apoyándose en esa matriz de costo busca el camino. Ese camino es retornado a la aplicación del usuario, para que este siga los pasos indicados por el camino.

3.6.6 Flujo de Sucesos “Buscar Camino Mínimo Parcial”.

Similar al anterior flujo de sucesos, el usuario una instancia de la clase PathEngine, después de tener ese objeto de tipo PathEngine inicializado, el puede llamar a las funciones de este, en este caso se llamaría a la siguiente función:

```
pePath CreatePartialPath(peAgent* Agent, cVector3f TargetPoint)
```

Debe pasar como dato, un Agente (un agente se crea para representar un objeto que debe saber navegar por el entorno, y que es controlado por la computadora) y un Punto, al igual que en el caso anterior, la función crea un buscador AStarSearcher y le pide buscar el camino. La característica de esta función que la difiere de la función anterior es que no retorna el camino completo, si no que retorna el primer paso que habría que dar si escogiéramos ese camino. Esto tiene como objetivo que el agente pueda cambiar de destino sin tener que recorrer un camino entero.

3.6.7 Flujo de Sucesos “Buscar Objeto Más Cercano”.

En este caso, el usuario crea una instancia de la clase PathEngine que esta dentro de la librería ABPathEngine, después de tener ese objeto de tipo PathEngine inicializado, llama a la función:

```
pePath CreatePathToClosestItem(peAgent* Agent, char* Type)
```

Debe pasar como dato, un Agente y un Punto, la función crea un buscador, pero en este caso sería un DijkstraSearcher (es usado por la estrategia de búsqueda de este algoritmo). Luego le pide al buscador encontrar el camino, de manera similar a la función CreatePath(), y basándose en las areas y los reachabilities, calcula la matriz de costo, y comienza a buscar hasta encontrar el elemento, del tipo Type, mas cercano, y crea el camino hacia este. Ese camino es retornado a la aplicación del usuario, para que este siga los pasos indicados por el mismo.

3.7 Modelo de Implementación

El modelo de implementación describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describe también como se organizan y se relacionan unos con otros, definiendo un componente como el empaquetamiento físico de los elementos de un modelo, como es el caso de las clases del modelo de diseño.

A continuación se muestran los paquetes definidos y los componentes que pertenecen a cada uno de estos, así como la relación de dependencia que existe entre componentes de distintos paquetes, representando los paquetes según su definición por un color determinando. Además, se describe cada uno de estos componentes de acuerdo a su propósito y clases que contiene. Los diagramas se muestran en el ANEXO.

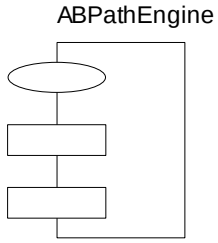

Figura 4 - Paquete de componentes “ABPathEngine”.

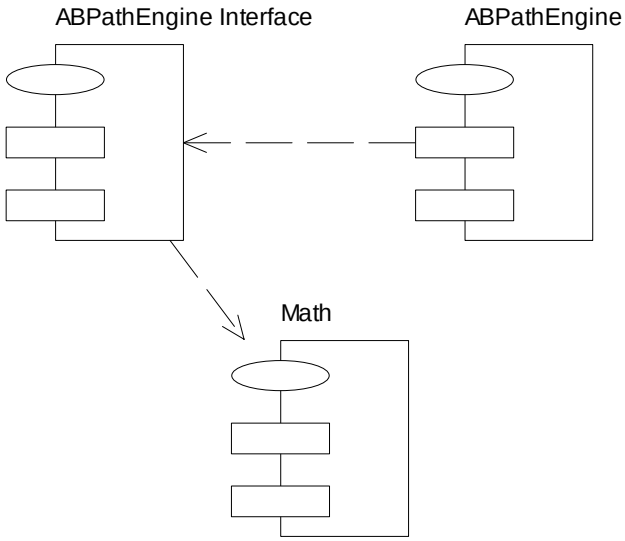

Figura 5 - Subpaquetes de componentes “ABPathEngine”.

3.8 Descripción de los Casos de Uso de pruebas.

Tabla 43 – Caso de prueba “Importar mal fichero”.

ABPathEngine Data Creator	Caso de Uso	Importar Fichero .ASE
	Caso de prueba	Importar Fichero Mal Formado
	Entrada	Se manda a importar un Fichero
	Resultado	No se importa ningún dato, por lo que el fichero que se exporte no contendrá datos.
	Condiciones	El fichero, aunque el formato no sea correcto debe tener la extensión .ASE

Tabla 44 – Caso de prueba “Importar fichero mal formado”.

ABPathEngine Data Creator	Caso de Uso	Importar Fichero .ASE
	Caso de prueba	Importar Fichero Bien Formado
	Entrada	Se manda a importar un Fichero
	Resultado	Son importadas de manera adecuada, todas las áreas, creadas por el diseñador en el 3D Studio Max, junto con todos los datos necesarios para el correcto funcionamiento de la aplicación.
	Condiciones	El fichero debe tener la extensión .ASE

Tabla 45 – Caso de prueba “Generar Reachabilities sin importar áreas del fichero”.

ABPathEngine Data Creator	Caso de Uso	Generar Reachabilities
	Caso de prueba	Generar Reachabilities sin haber importado área alguna del fichero .ASE.
	Entrada	Se ejecuta la aplicación, para calcular lo reachabilities.
	Resultado	Al no existir áreas, lo rechabilities no pueden ser creados y se mantiene la lista de reachabilities vacía.
	Condiciones	Se deben haber cargado los plug-ins de los reachabilities, e importado un fichero .ASE.

Tabla 46 – Caso de prueba “Generar Reachabilities luego de haber importado una lista de áreas del fichero”.

ABPathEngine Data Creator	Caso de Uso	Generar Reachabilities
	Caso de prueba	Generar Reachabilities luego de haber importado una lista de áreas del fichero.
	Entrada	Se ejecuta la aplicación, para calcular lo reachabilities.
	Resultado	Los reachabilities creators, calculan e instancian a sus reachabilities.
	Condiciones	Se deben haber cargado los plug-ins de los reachabilities, e importado un fichero .ASE.

Tabla 47 – Caso de prueba “Exportar fichero APE, habiendo cargado antes un ASE mal formado”.

ABPathEngine Data Creator	Caso de Uso	Exportar Fichero .APE
	Caso de prueba	Exportar Fichero .APE, habiendo cargado un .ASE, bien formado
	Entrada	Se selecciona desde el menú, la opción de exportar.
	Resultado	Se crea un fichero con extensión .APE, y son agregados a este todos los datos de las áreas, y los reachabilities creados.
	Condiciones	Se deben haber cargado los plug-ins de los reachabilities, e importado un fichero .ASE.

Tabla 48 – Caso de prueba “Exportar fichero APE sin haber cargado fichero alguno”.

ABPathEngine Data Creator	Caso de Uso	Exportar Fichero .APE
	Caso de prueba	Exportar Fichero .APE, sin haber cargado fichero alguno.
	Entrada	Se selecciona desde el menú, la opción de exportar.
	Resultado	Se crea un fichero con extensión .APE, pero sin datos algunos.
	Condiciones	

Tabla 49 – Caso de prueba “Exportar fichero APE, habiendo cargado un ASE bien formado”.

ABPathEngine Data Creator	Caso de Uso	Exportar Fichero .APE
	Caso de prueba	Exportar Fichero .APE, habiendo cargado un .ASE, bien formado
	Entrada	Se selecciona desde el menú, la opción de exportar.
	Resultado	Se crea un fichero con extensión .APE, y son agregados a este todos los datos de las áreas, y los reachabilities creados.
	Condiciones	Se deben haber cargado los plug-ins de los reachabilities, e importado un fichero .ASE.

Tabla 50 – Caso de prueba “Importar fichero APE no existente”.

ABPathEngine	Caso de Uso	Importar Fichero .APE
	Caso de prueba	Importar Fichero .APE, no existente.
	Entrada	El usuario instancia la aplicación y para esto pasa como parámetro un fichero no existente.
	Resultado	La aplicación muestra en consola un mensaje mostrando que no se ha encontrado el fichero.
	Condiciones	

Tabla 51 – Caso de prueba “Importar fichero APE existente”.

ABPathEngine	Caso de Uso	Importar Fichero .APE
	Caso de prueba	Importar Fichero .APE, existente.
	Entrada	El usuario instancia la aplicación y para esto pasa como parámetro un fichero existente.
	Resultado	La aplicación muestra en consola un mensaje mostrando que se ha importado el fichero adecuadamente.
	Condiciones	Se deben haber importado, mínimo, un plug-in de reachability.

Tabla 52 – Caso de prueba “Buscar camino mínimo a un punto determinado no alcanzable”.

ABPathEngine	Caso de Uso	Buscar Camino Mínimo
	Caso de prueba	Buscar camino mínimo a un punto determinado no alcanzable.
	Entrada	Se solicita buscar el camino hasta un punto desde la aplicación cliente.
	Resultado	Retorna un camino vacío, porque el punto no es alcanzable.
	Condiciones	Se deben haber importado, mínimo, un plug-in de reachability, y un fichero .APE

Tabla 53 – Caso de prueba “Buscar camino mínimo a un punto determinado alcanzable”.

ABPathEngine	Caso de Uso	Buscar Camino Mínimo
	Caso de prueba	Buscar camino mínimo a un punto determinado alcanzable.
	Entrada	Se solicita buscar el camino hasta un punto desde la aplicación cliente.
	Resultado	Retorna un camino mas corto hasta el punto.
	Condiciones	Se deben haber importado, mínimo, un plug-in de reachability, y un fichero .APE

Tabla 54 – Caso de prueba “Buscar camino mínimo parcial a un punto determinado no alcanzable”.

ABPathEngine	Caso de Uso	Buscar Camino Mínimo Parcial
	Caso de prueba	Buscar camino mínimo parcial a un punto determinado no alcanzable.
	Entrada	Se solicita buscar el camino hasta un punto desde la aplicación cliente.
	Resultado	Retorna un camino vacío, porque el punto no es alcanzable.
	Condiciones	Se deben haber importado, mínimo, un plug-in de reachability, y un fichero .APE

Tabla 55 – Caso de prueba “Buscar camino mínimo parcial a un punto determinado no alcanzable”.

ABPathEngine	Caso de Uso	Buscar Camino Mínimo Parcial
	Caso de prueba	Buscar camino mínimo parcial a un punto determinado alcanzable.
	Entrada	Se solicita buscar el camino hasta un punto desde la aplicación cliente.
	Resultado	Retorna un camino hasta la próxima área que se debe acceder, para llegar hasta el punto, con un menor costo.
	Condiciones	Se deben haber importado, mínimo, un plug-in de reachability, y un fichero .APE

Tabla 56 – Caso de prueba “Buscar camino a objeto, de un tipo específico no existente”.

ABPathEngine	Caso de Uso	Buscar Objeto Mas Cercano
	Caso de prueba	Buscar camino a objeto, de un tipo especifico, no existente.
	Entrada	Se solicita buscar el camino hasta el objeto, de un tipo determinado, mas cercano.
	Resultado	Retorna un camino vacío, porque no existe objeto alguno de dicha clase.
	Condiciones	Se deben haber importado, mínimo, un plug-in de reachability, y un fichero .APE

Tabla 57- Caso de prueba “Buscar camino a objeto, de tipo específico existente”.

ABPathEngine	Caso de Uso	Buscar Objeto Mas Cercano
	Caso de prueba	Buscar camino a objeto, de tipo específico, existente.
	Entrada	Se solicita buscar el camino hasta el objeto, de un tipo determinado, más cercano.
	Resultado	Retorna un camino hasta el objeto, de dicho tipo, más cercano.
	Condiciones	Se deben haber importado, mínimo, un plug-in de reachability, y un fichero .APE

3.9 Conclusiones

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del sistema y la secuenciación de pasos traducida a mensajes entre clases de los primeros casos de usos a desarrollar, con lo que se puede pasar a la etapa de implementación del proyecto.

CONCLUSIONES GENERALES

En este trabajo se cumplió con los objetivos planteados, debido al estudio detallado de las herramientas similares. De manera que cumpliera los requisitos funcionales establecidos para lograr las metas trazadas se realizó un paquete de herramientas que mostró su funcionalidad con los resultados obtenidos.

El aporte principal de este trabajo ha sido el desarrollo de un paquete de herramientas que brinda al desarrollador de aplicaciones virtuales una serie de facilidades para implementar el módulo de inteligencia artificial, específicamente, la navegación y búsqueda de caminos de los personajes virtuales monitoreados por la computadora. En estos momentos solo se ha implementado una primera versión por la poca experiencia de los desarrolladores en este campo de realidad virtual para utilizarla en el proyecto de realidad virtual dada la falta realismo de los personajes.

Debido al diseño del paquete de herramientas es adaptable a los posibles cambios, lo que posibilita que se puedan agregar nuevas funcionalidades en versiones posteriores sin hacer grandes modificaciones. Esto aumentaría la utilidad del paquete de herramientas en vistas a confeccionar un producto cada vez más completo y eficaz.

RECOMENDACIONES

En Cuba no se conocen grandes logros en el campo de la realidad virtual y, respecto a la inteligencia artificial, se han experimentado en varios campos como la medina y la educación con muy buenos resultados pero no se le ha prestado la misma atención a la aplicación de las técnicas de AI a la realidad virtual. Esta investigación se realizó con el objetivo de insertarse en el mundo de la inteligencia artificial en los entornos tridimensionales, y se recomienda continuar profundizando en esta rama dado que le proporciona más dinámica al producto y un comportamiento más real. Los productos en la rama de la informática siempre se pueden ir mejorando y actualizando y la solución propuesta no está exenta de esto, por lo que a continuación se muestran algunas de las posibles mejoras que se recomiendan.

- Agregarle más funciones que le posibiliten al desarrollador trabajar más detalladamente con el comportamiento de los personajes en la navegación dentro del entorno virtual.
- Agregarle otros tipos de algoritmos para aumentar las variantes de búsqueda.
- Mejorar el importador de datos o ABPathEngine Data Creator para que logre importar otros tipos de ficheros de herramientas de diseño.
- Agregarle más plugins con las definiciones de los reachabilities para que el desarrollador tenga varias opciones a escoger por defecto (en esta versión solo se implementó el de caminar).
- Implementar una herramienta visual que facilite el uso de las funcionalidades de la biblioteca de inteligencia artificial.

Referencias Bibliográficas

[1] Recopilación de Autores. *AI Game Programing Wisdom*. Charles River Media, 2002.

[2] Koza, John R.; Bennett, Forrest H. III; Keane, Martin; Andre, David. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, 1999.

[3] Touzour, Paul. *An Introducing to Bayesian Network and Rasoning Undercertain*, de *AI Game Programming Wisdom*. Charles River Media, 2002

[4] F. Markus Jönsson. *An optimal pathfinder for vehicle in real-world digital terrain map*. 1997. disponible en: http://www.student.nada.kth.se/~f93-maj/pathfinder/index.html

[5] Stout, Bryan. *The Basics of A* of Path Planning* de Ed. DeLoura, Mark. *Game Programming Gems*. Charles River Media, 2000.

[6] Rabin, Steve. *A* Aestheric Optimizations* y *A* Speed Optimizations* de Ed. DeLoura, Mark. *Game Programming Gems*. Charles River Media, 2000.

[7] van Waveren, J.M.P. *Quake III Arena Bot*. 2001. disponible en: http://www.cs.rochester.edu/research/quagents/QuakeIII.pdf

[8] Análisis de Lenguajes de Programación. 2003 disponible en http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=4

[9] Documentación del PathEngine disponible en: http://www.pathengine.com/Contents/page.php

[10] Larman, Craig. UML y Patrones. *Introducción al análisis y diseño orientado a objetos.* Editorial Félix Varela. La Habana, 2004.

[11] P. D. Holmes, E.R.A. Jungert. *Symbolic and geometric connectivity graph methods for route planning in digitized maps*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 5, 1992.

[12] A. Patel (editor), *Game Programming*, 1997 disponible en: http://www-cs-students.stanford.edu/~amitp/gameprog.html

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C, 2nd ed.*, Cambridge University Press, New York, 1992.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 1990.

[15] D. G. Luenberger, *Linear and Nonlinear programming*, 2nd ed, Addison Wesley, 1984.

[16] N J Nilsson, *Principles of Artificial Intelligence*, Springer Verlag. Berlin, 1982.

Bibliografía Consultada

- Recopilación de Autores. *AI Game Programing Wisdom 2*. Charles River Media, 2003.
- Buckland, Mat. *Programming Game AI by Example*. WordWare Publishing. Inc, 2005.
- Buckland, Mat. *AI Techniques for Game Programming*. Premier Press, 2002.
- Schwab, Brian. *AI Game Engine Programming*. Charles River Media, 2000.
- Andrew Rollings, Ernest Adams. *Andrew Rollings and Ernest Adams on Game Design*. New Riders Publishing, 2003.
- John P. Flynt Ph. D with Omar Salem. *Software Engineering for Game Developers*. Thomson Course Technology, 2005.
- Alexander Nareyek. *AI in Computers Games*. Febrero 2004. disponible en: http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=117
- David M. Bourg, Glenn Seemann. *AI for Game Developers*. O'Rielly Media, 2004.
- Craig W. Reynolds. *Steering Behaviours for Autonomous Characters*. 1999. disponible en: http://www.red3d.com/cwr/papers/1999/gdc99steer.pdf
- M. Lozano, F. Barber, L. Vera, J. Carrasco, M. Fernández. *Navegación autónoma de actores virtuales en entornos dinámicos*. 2001. http://www.uv.es/mlazano/publicaciones/Ceig2001.doc

Glosario de Términos

3D Max Studio: es un programa de creación de gráficos y animaciones 3D desarrollado por Autodesk Media & Entrtaiment (formalmente conocido como Discreet y Kinetix).

Simulador: es un aparato capaz de reproducir un sistema, los simuladores nos hacen vivir sensaciones que en realidad no están sucediendo.

Realidad Virtual: es un sistema o interfaz informático que genera entornos sintéticos en tiempo real, representación de las cosas a través de medios electrónicos o representaciones de la realidad, una realidad ilusoria, pues se trata de una realidad perceptiva sin soporte objetivo, sin res extensa, ya que existe sólo dentro del ordenador.

Visita virtual o paseo virtual: son una forma de publicidad altamente efectiva y atractiva para los usuarios. El uso de esta tecnología interactiva aumenta notablemente el número de visitas a una página. En el caso de una visita virtual por una vivienda, mediante este sistema los usuarios tendrán la sensación de pasear por dentro del inmueble como si estuviesen allí.

Engine: programa orientado a la creación de otros softwares.

Entornos virtuales o mundos virtuales: se trata de la simulación de mundos o entornos, denominados virtuales, en los que el hombre interacciona con la máquina en entornos artificiales semejantes a la vida real.

Middleware: es un software que permite ofrecer un conjunto de servicios que hacen posible el funcionamiento de aplicaciones sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red).

Pathfinding: conjunto de rutinas de programación que ayudan a un personaje controlado por el ordenador a encontrar el camino más corto entre dos puntos del mapa, así como a actualizar la ruta ante la aparición de obstáculos inesperados.

R.N.A.: Redes Neuronales Artificiales.

Bot: procedente robot, es el personaje virtual que es manipulado por la computadora.

Backtraking: (o vuelta atrás) es una técnica de resolución general de problemas mediante una búsqueda sistemática de soluciones. El procedimiento general se basa en la descomposición del proceso de búsqueda en tareas parciales de tanteo de soluciones (*trial and error*). La solución de una tarea parcial permite construir la solución del problema de partida. Las tareas parciales son similares a la de partida y se pueden resolver directamente o volver a descomponer en otras tareas parciales. Las tareas parciales se plantean de forma recursiva al construir gradualmente las soluciones.

Reachability: accesibilidad o método de acceder de un área a otra.

Waypoint: son coordenadas de puntos de referencia utilizados en la navegación de un entorno virtual.

ANEXOS

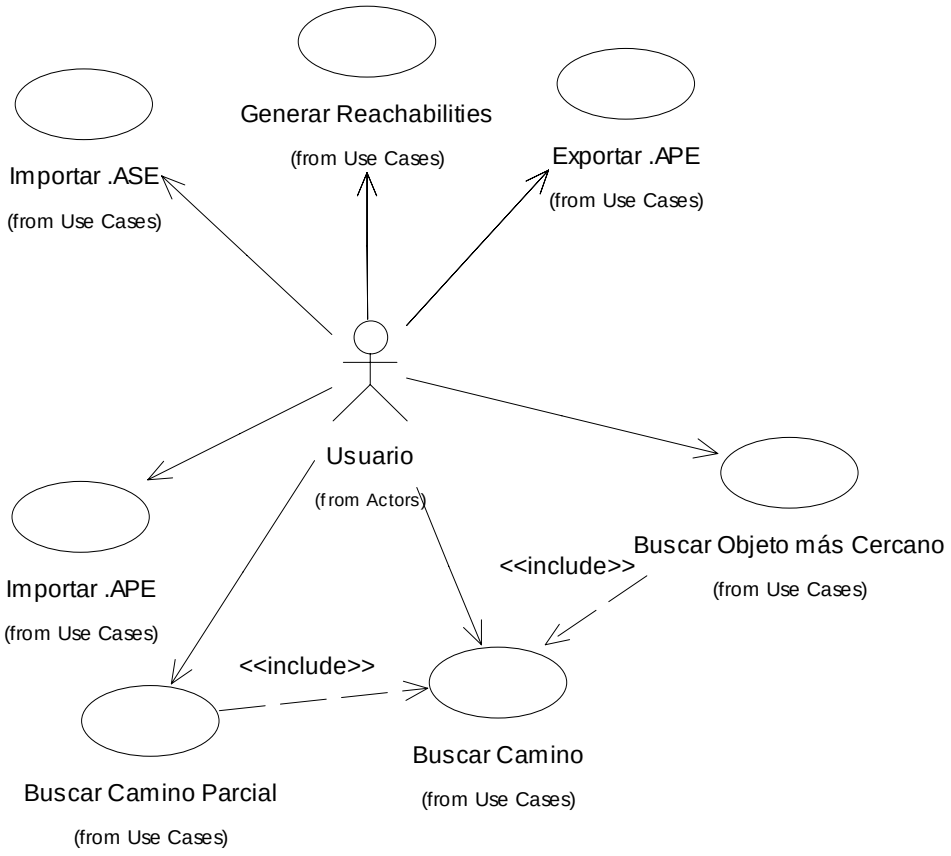

Figura 6 - Diagrama de Casos de Uso.

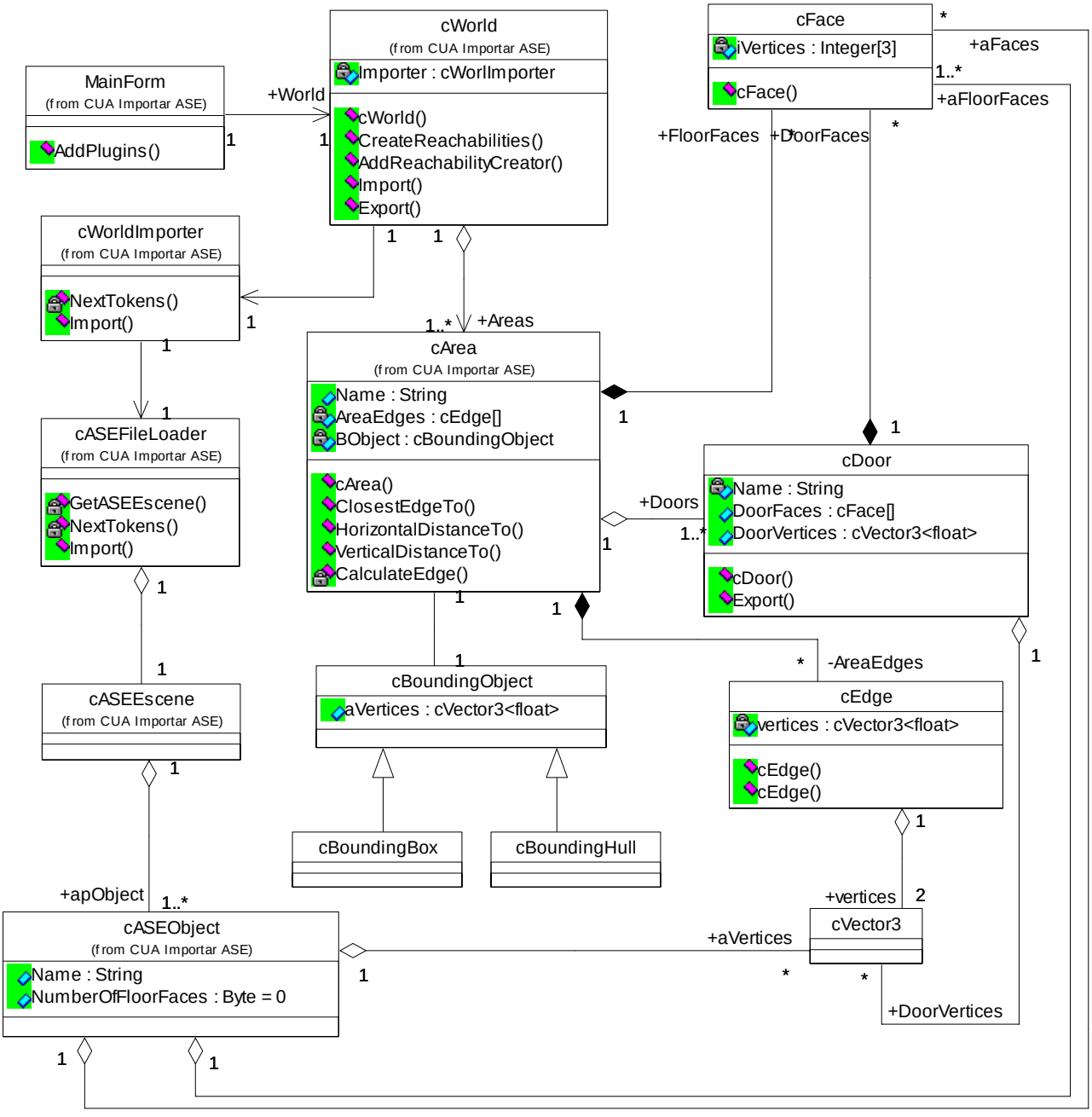

Figura 7 - Diagrama de Clases CU “Importar ASE”.

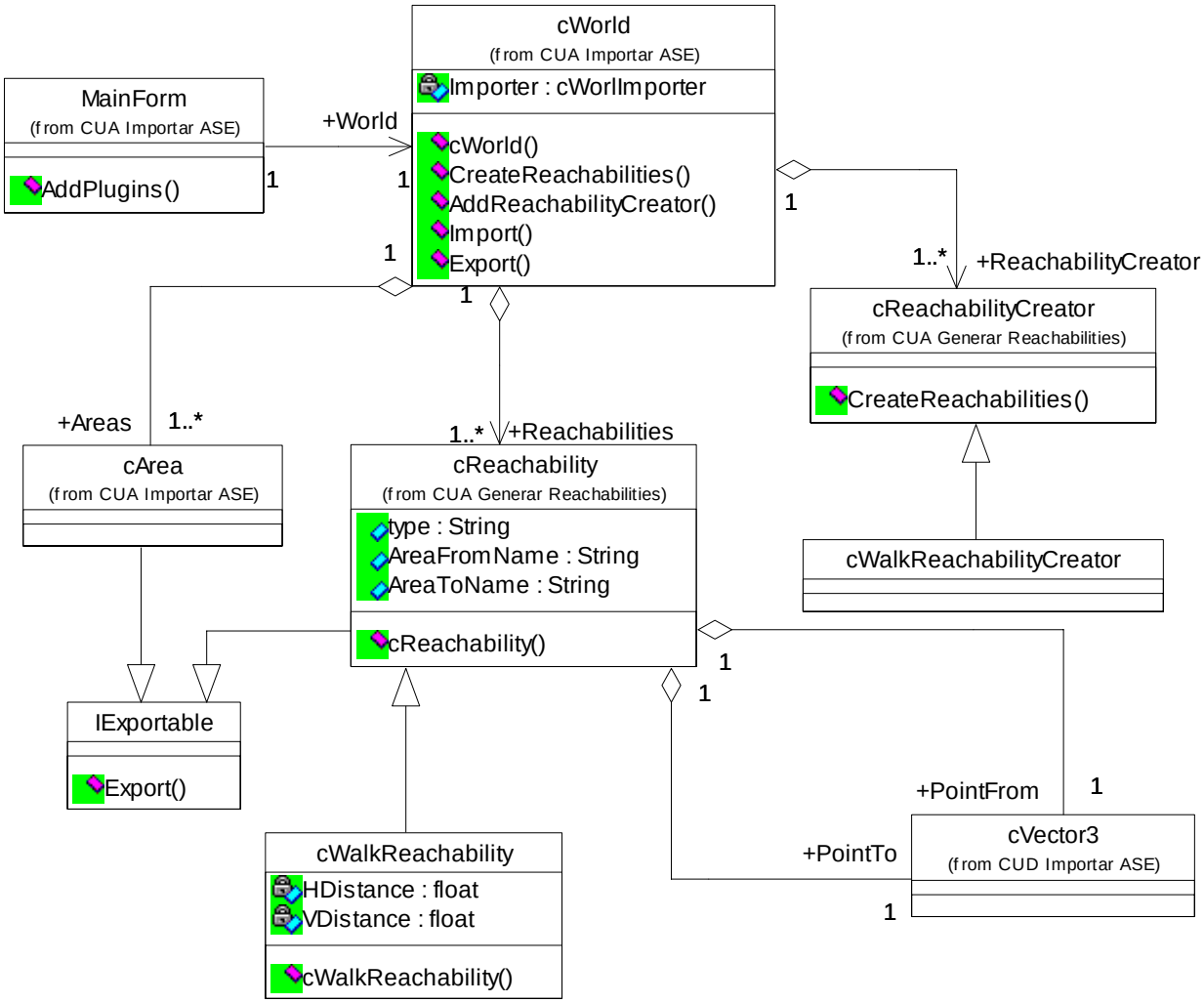

Figura 8 - Diagrama de Clases CU “Generar Reachabilities”.

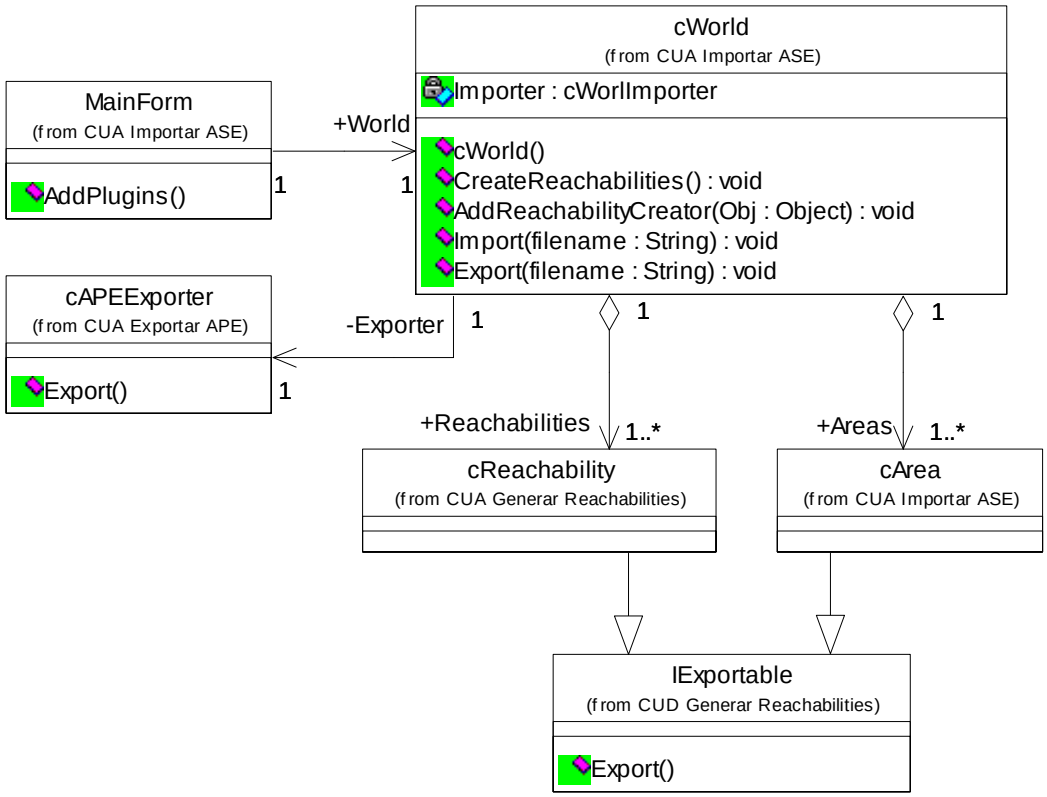

Figura 9 - Diagrama de Clases CU “Exportar APE”.

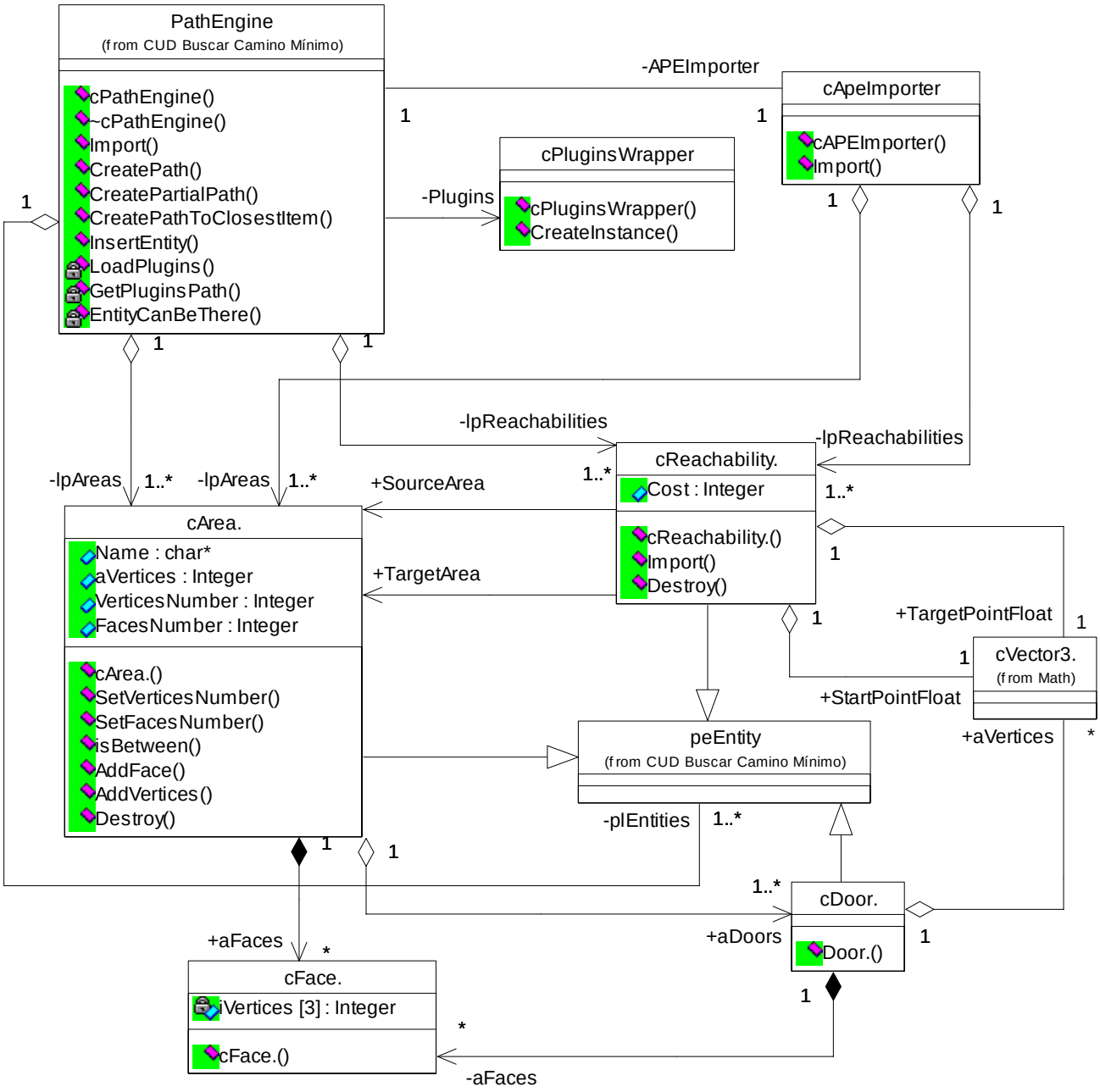

Figura 10 - Diagrama de Clases CU “Importar APE”.

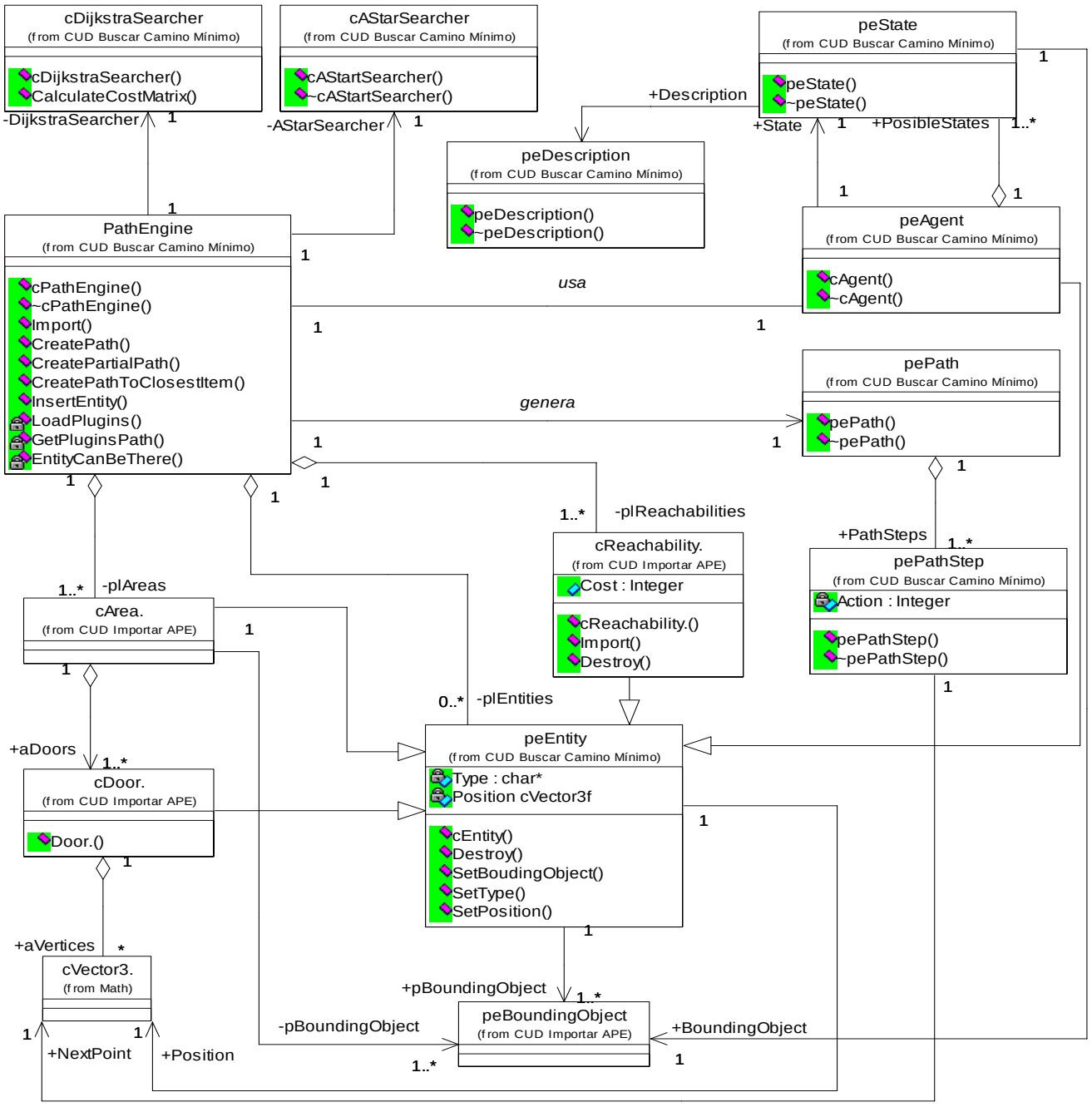

Figura 11 - Diagramas de Clases CU “Buscar Camino Mínimo”.

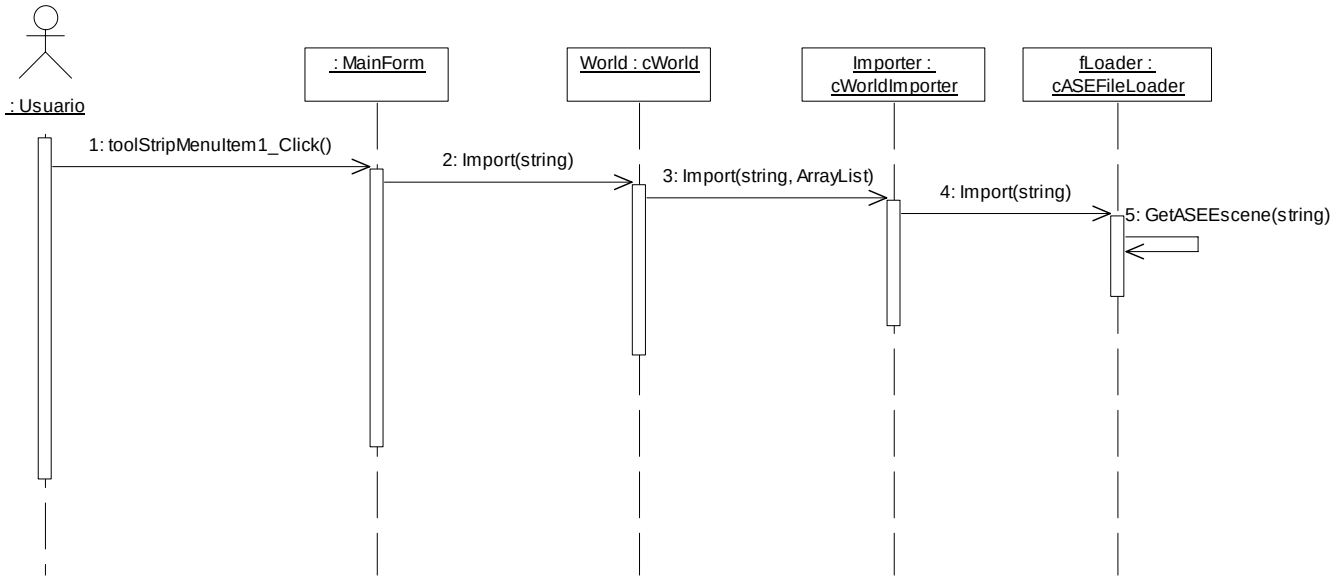

Figura 12 - Diagrama de Secuencia CU “Importar ASE”.

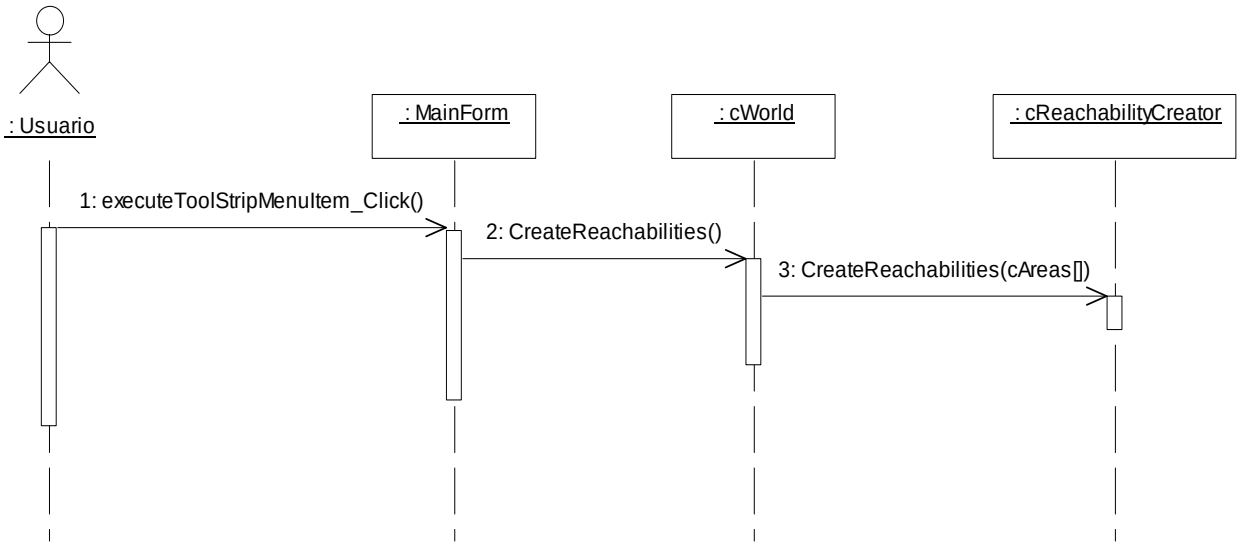

Figura 13 - Diagrama de Secuencia CU “Generar Reachabilities”.

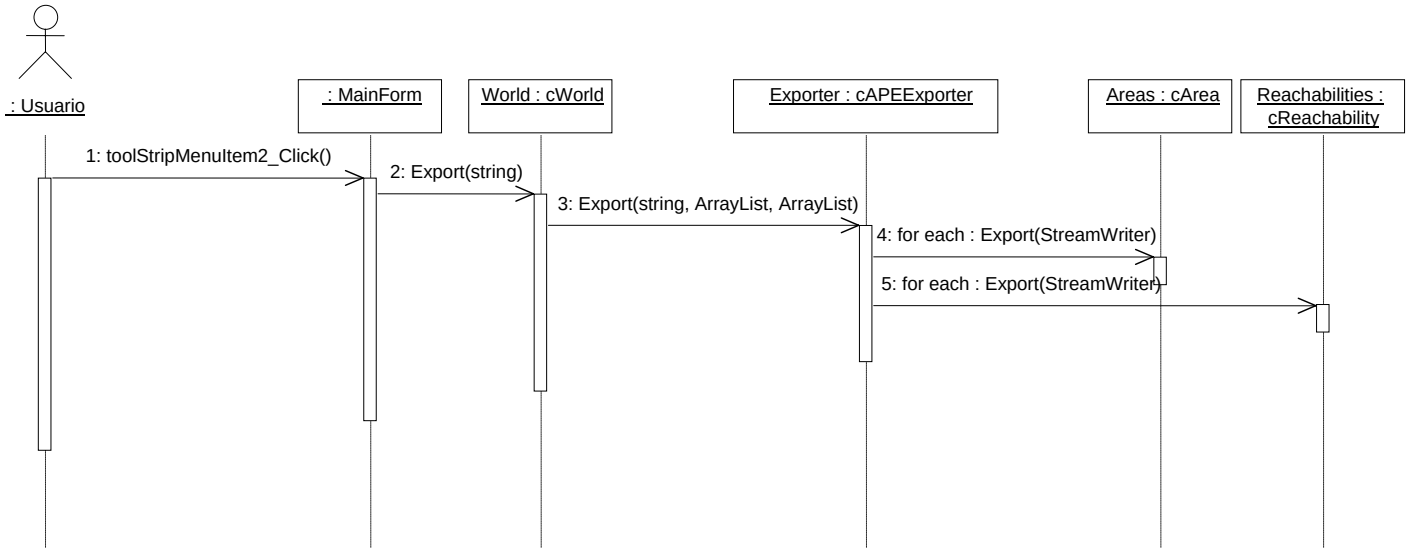

Figura 14 - Diagrama de Secuencia CU “Exportar APE”.

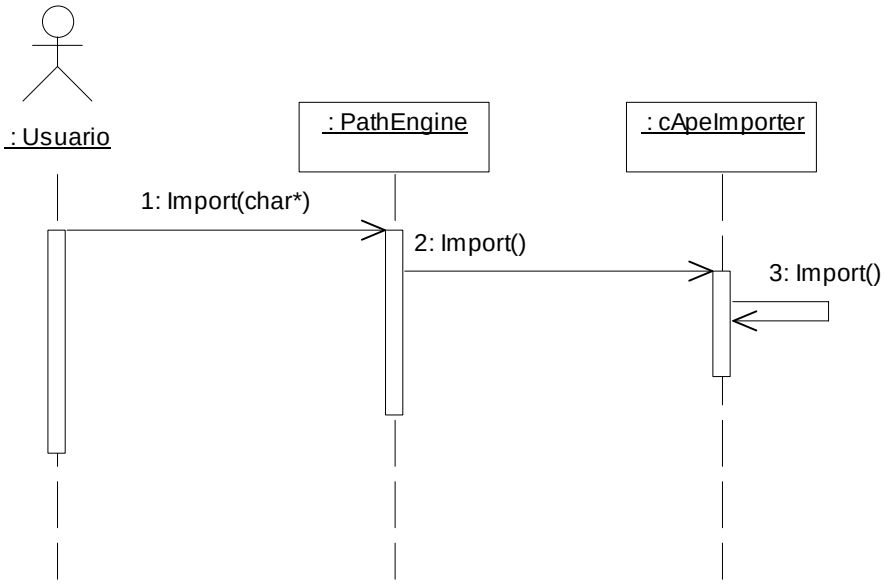

Figura 15 – Diagrama de Secuencia CU “Importar APE”.

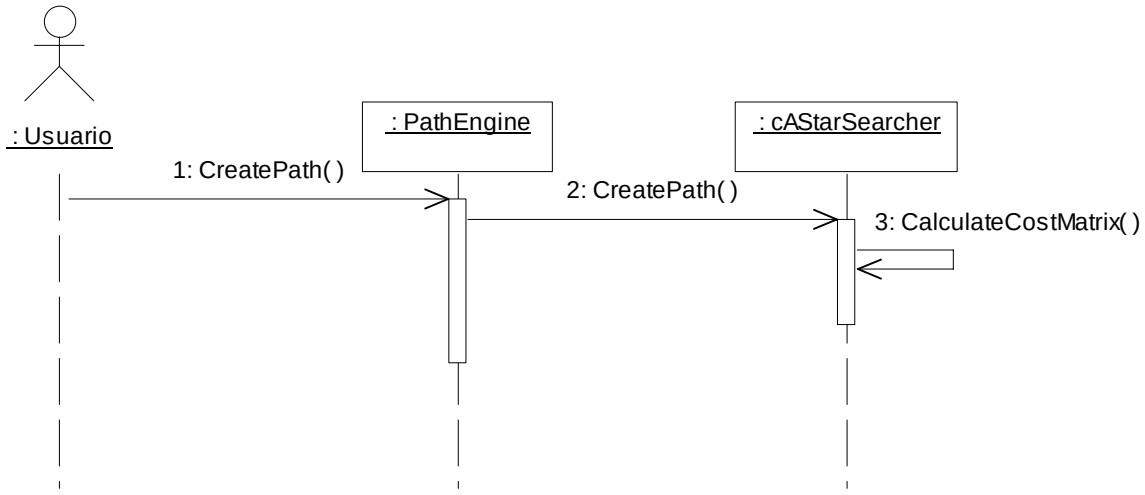

Figura 16 - Diagrama de Secuencia CU “Buscar Camino”.

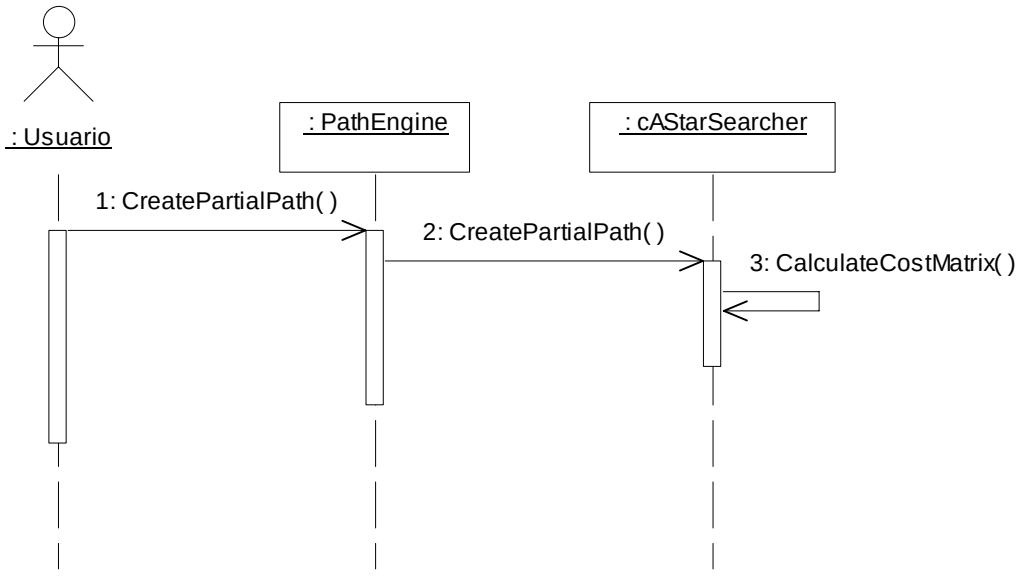

Figura 17 - Diagrama de Secuencia CU “Buscar Camino Parcial”.

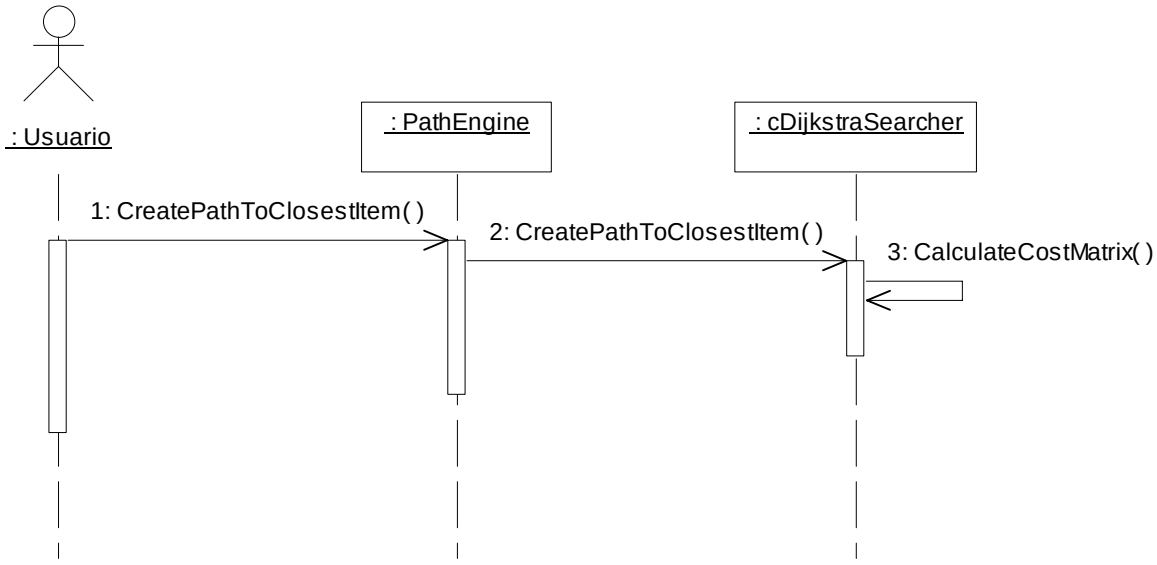

Figura 18 - Diagrama de Secuencia CU “Buscar Objeto más Cercano”.

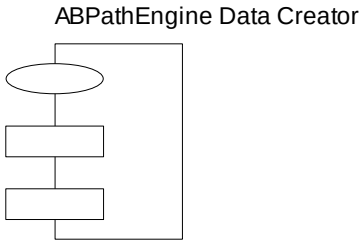

Figura 19 - Paquete de componentes “ABPathEngine Data Creator”.

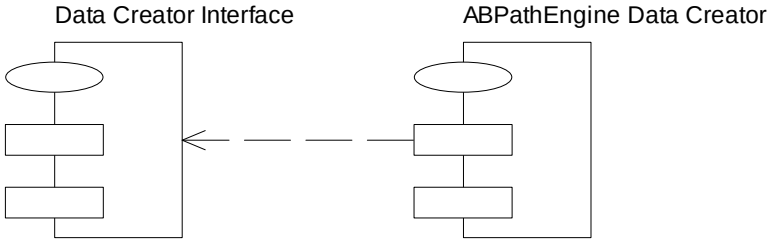

Figura 20 - Subpaquetes de componentes “ABPathEngine Data Creator”.

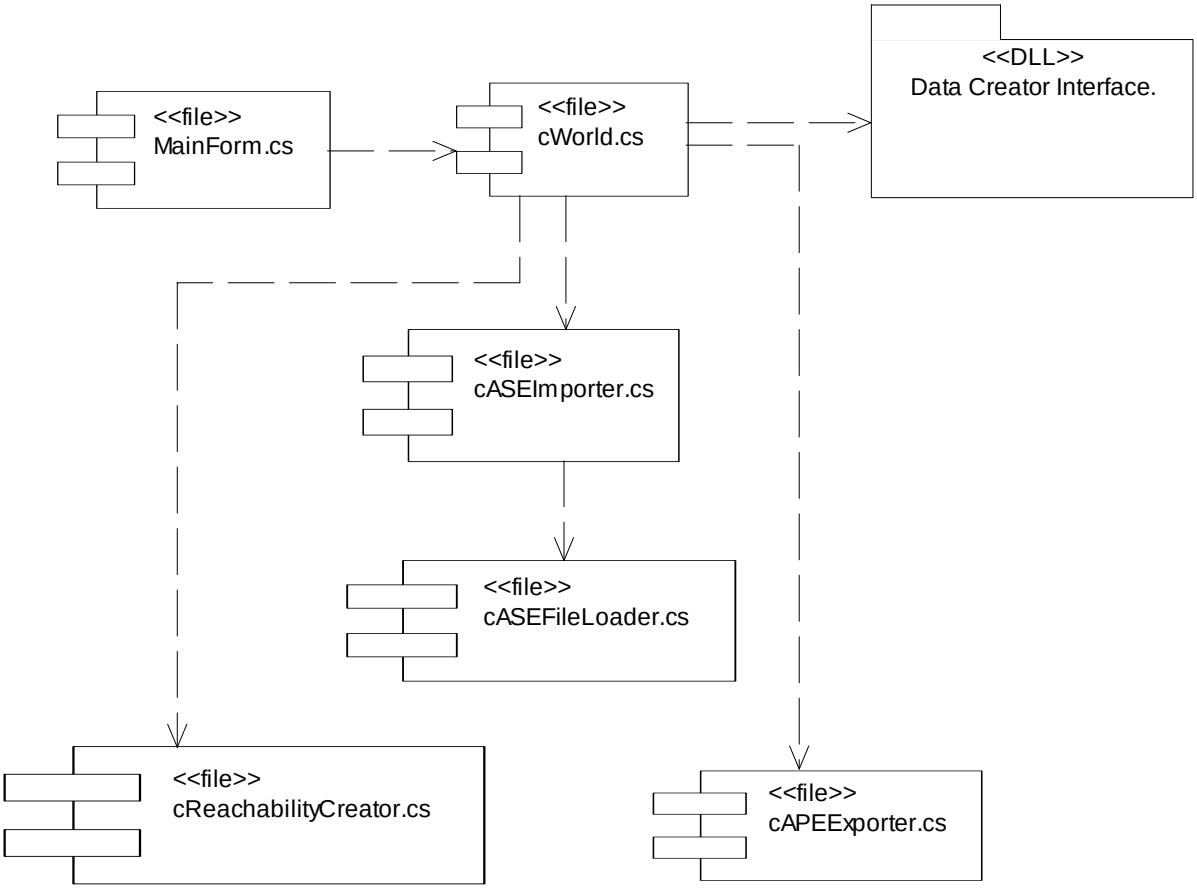

Figura 21 - Diagrama de componentes “ABPathEngine Data Creator”.

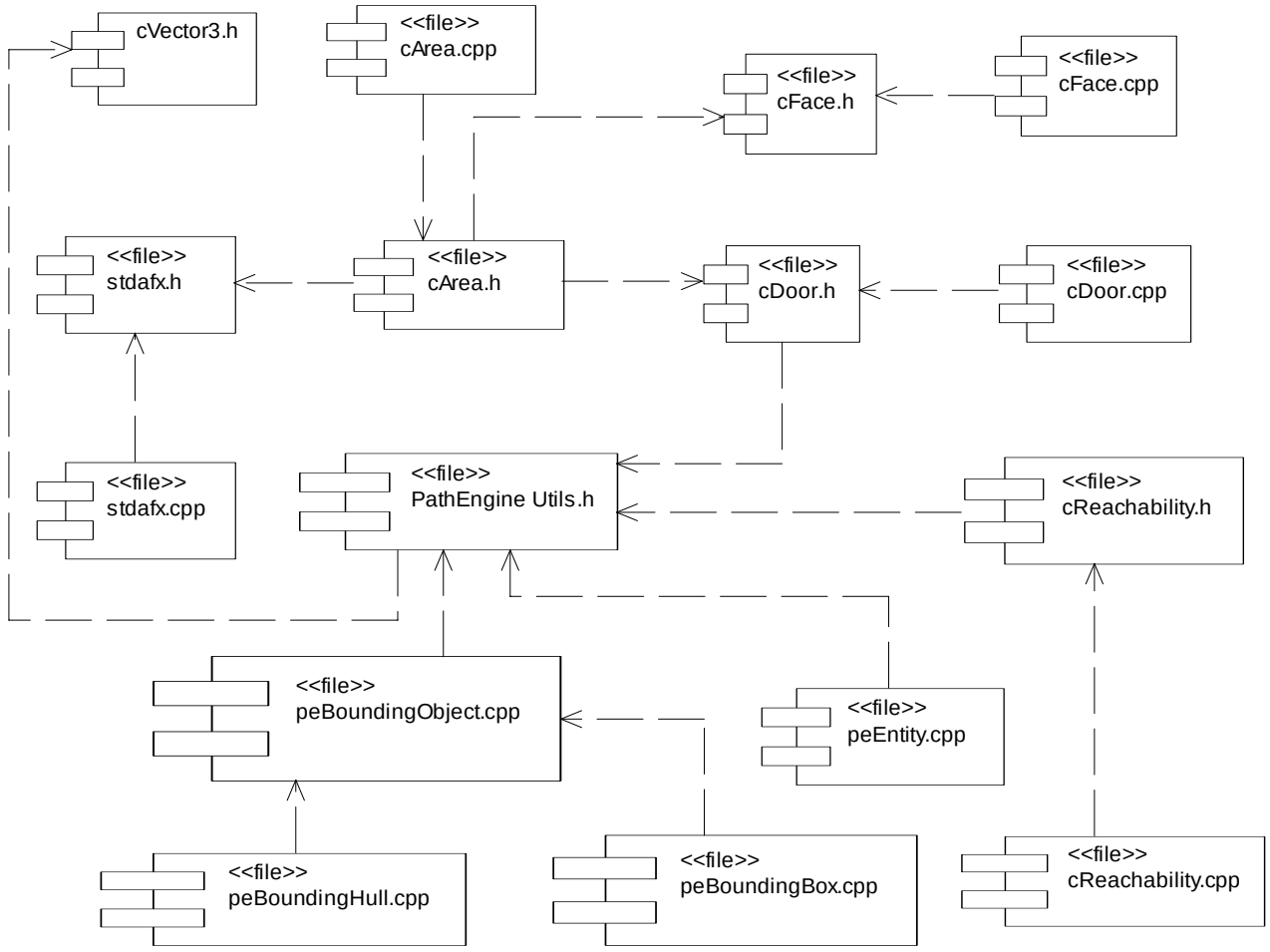

Figura 22 - Diagrama de componentes “Data Creator Interface”.

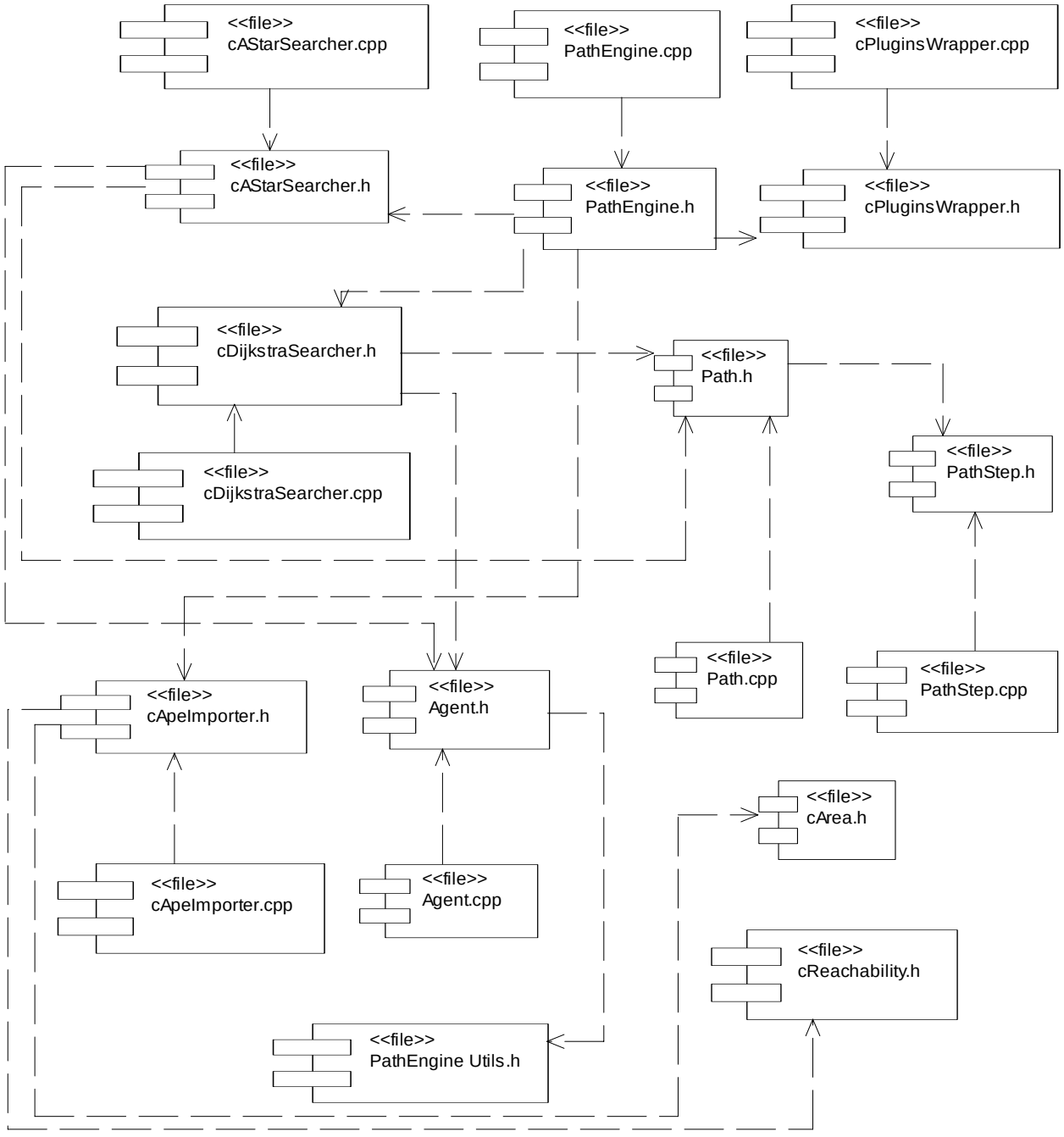

Figura 23 - Diagrama de componentes “ABPathEngine”.

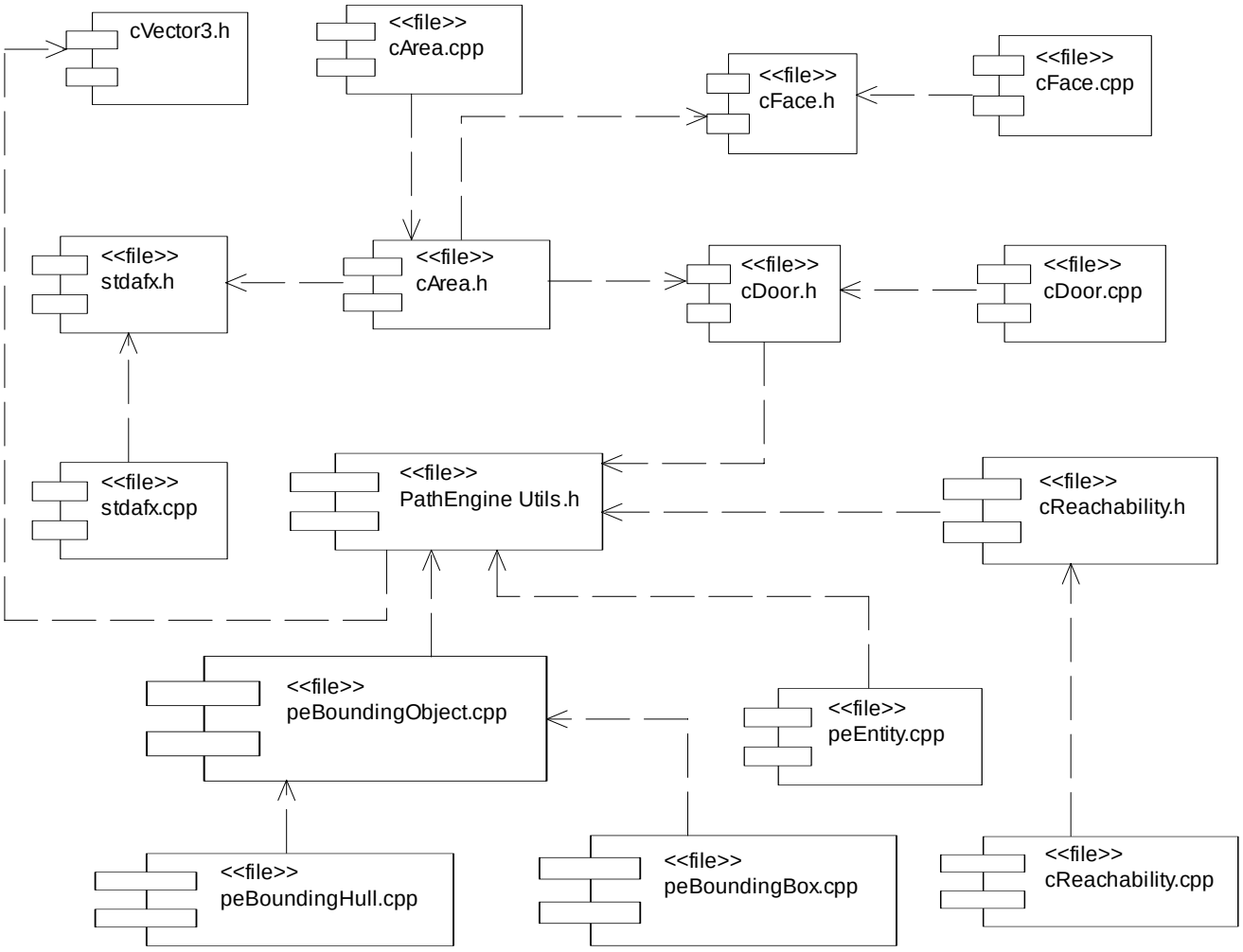

Figura 24 - Diagrama de componentes “ABPathEngine Inteface”.