La Habana, 2009
Universidad de las Ciencias Informáticas. Facultad 5

Biblioteca genérica para la Búsqueda de Caminos.

Autor: Lecnier Sanz Friman

Tutor: Ing. Frank Puig Placeres

AGRADECIMIENTOS

A todas las personas que me han ayudado a trabajar en una computadora. A mis padres por quererme tanto y la educación que me han dado. A mi tutor Frank Puig Placeres que ha sido guía de mi formación como ingeniero informático. A Yailen por ayudarme a dedicarme por completo a lo que me gusta.

DEDICATORIA

A mis padres.

RESUMEN

La popularidad de las simulaciones de entornos virtuales en los últimos años ha crecido exponencialmente. En la parte de los videos juegos a muchas personas les gusta interactuar con personas virtuales conocidos también como agentes inteligentes. Se han creado un gran número de simuladores que ayudan a la formación de militares, al estudio del comportamiento de comunidades de especies, etc. Estas simulaciones que involucran el empleo de agentes inteligentes deben contar con un módulo encargado de manejar la Inteligencia Artificial (IA).

Una de las ramas de la IA es la búsqueda de caminos. La responsabilidad de esta rama es la de mostrarle a los agentes inteligentes por donde deben desplazarse para llegar a sus destinos. El objetivo de este trabajo es la creación de una biblioteca que se encargue de la planificación y búsqueda de caminos que los agentes inteligentes puedan necesitar.

Para desarrollar esta biblioteca se estudian los temas: planificador de camino, los distintos tipos de modelos cognitivos, los algoritmos de búsqueda más usados y las heurísticas que se emplean en estos algoritmos. Se exponen además las características técnicas que presentará el sistema como solución a los problemas planteados. Se analiza con profundidad el objeto de estudio, y a partir del mismo se confeccionan los distintos diagramas que posibilitarán un mayor entendimiento y proporcionarán la base para el diseño e implementación de la biblioteca.

PALABRAS CLAVE

Inteligencia Artificial, Agentes Inteligentes, Modelo Cognitivo, Algoritmo de búsqueda, Heurística

TABLA DE CONTENIDO

INTRODUCCIÓN

Actualmente uno de los principales mercados del mundo es la venta de video juegos y simuladores virtuales. En los últimos años los componentes de hardware de las computadoras han experimentado un gran desarrollo, incrementándose también la demanda de lo que el cliente/usuario pide de los video-juegos y simuladores virtuales. Uno de los aspectos que ejerce mayor influencia en el triunfo de un video-juego o simulador virtual, es lo tan real que sea la simulación que podamos obtener de él.

Para lograr una buena simulación debemos contar con una buena biblioteca de inteligencia artificial. Entre las funciones que debe ser capaz de hacer esta biblioteca se encuentra la búsqueda y planificación de caminos. Esta función nos sirve para dar respuesta a las siguientes situaciones: un agente necesita llegar a un punto por el lado más cercano, un agente necesita encontrar un ítem, un agente necesita esconderse de otros agentes, o buscar el mejor punto para enfrentar a otros agentes.

La facultad 5 tiene un polo de realidad virtual, queriendo insertarse en este gran mercado de los video-juegos y las simulaciones de entornos virtuales. En muchos de los proyectos de este polo se necesita hacer: Búsqueda de caminos, pero cada uno de estos proyectos hacen sus propias implementaciones, que solo son válidas para las estructuras que ellos mismos han definido en sus sistemas. De ahí que las optimizaciones que han hecho no puedan utilizarse en otros proyectos y estos tengan que implementar desde cero sus algoritmos de búsqueda.

Además los proyectos actuales usan heurísticas muy restringidas que no dan libertad para especificar los caminos más convenientes, de ahí que haya que re-implementar varias veces el mismo algoritmo, solo para cambiar la heurística y las optimizaciones asociadas a ellas.

De esta forma se crea un gran problema, pues los proyectos no pueden compartir sus optimizaciones, algoritmos y diseños de espacio de búsqueda. Pero a su vez tienen que duplicar varias veces el código pues si no sus búsquedas son muy restringidas.

Por lo que nos enfrentamos al siguiente **problema científico**: ¿Cómo desarrollar una biblioteca genérica que permita hallar caminos independientemente del modelo cognitivo usado?

Nuestro **objeto de estudio** será la Búsqueda de caminos en los entornos virtuales, y como **campo de acción** el Proceso de Búsqueda de caminos en los proyectos del polo de realidad virtual de la facultad 5. Con este problema científico nos planteamos como **objetivo general**: Desarrollar una biblioteca abstracta de búsqueda de caminos e implementar un demo que visualice el proceso de integración a cualquier proyecto. Y como **objetivos secundarios:**

- Lograr que la biblioteca encuentre caminos en cualquier modelo cognitivo que se pueda necesitar en los proyectos de la facultad.
- Acoplar diversas heurísticas al proceso de búsqueda de camino.

- Estructurar la biblioteca de forma que su diseño soporte búsquedas parciales usando Time-Slicing para mejorar los tiempos de respuestas en situaciones de congestión debido a multitudes de agentes realizando búsqueda de caminos.
- Confeccionar algoritmos genéricos que realicen refinamiento de caminos así como filtros de pos procesamiento para obtener un camino más parecido al que seguiría un agente real.
- Acoplar estructuras que permitan automatizar el proceso de corrección de caminos, así como el proceso de predicción inicial del mejor camino para mejorar los tiempos de respuesta.
- Definir e incorporar como fichero de entrada un formato en modo texto entendible.

Para cumplir estos objetivos se debe dar cumplimiento a la siguientes **Tareas Investigativas:**

- Recopilar información sobre los modelos cognitivos usados actualmente en el desarrollo de video juegos y simuladores.
- Recopilar los algoritmos de búsquedas y sus variantes.
- Recopilar las optimizaciones asociadas a cada uno de estos algoritmos de búsqueda.
- Transformar los algoritmos en versiones abstractas que usen de manera virtual el modelo cognitivo y la heurística.
- Crear la arquitectura de la biblioteca.
- Definir un formato de texto para el modelo cognitivo.
- Implementar el algoritmo de búsqueda abstracto.
- Desarrollar un demo que use la biblioteca de búsqueda de caminos.

Para dar cumplimento al objetivo y dar respuesta al problema científico planteado anteriormente se aplicaron los siguientes **métodos teóricos**: analítico-sintético para procesar la información recopilada durante la investigación e histórico lógico para constatar la evolución histórica de la situación problemática planteada y la necesidad de darle solución. Como método empírico, tenemos las entrevistas que se realizaron a profesores y jefes de proyectos del polo de realidad virtual.

Con esta tesis se logrará dar a la facultad una herramienta que pueda ser acoplada con facilidad a cualquier proyecto ya que la búsqueda de caminos se hace en una capa abstracta que no accede directamente a los datos del modelo cognitivo usado. De ahí que cada proyecto puede implementar el área de búsqueda que más útil le sea.

Además, esta tesis presentará vías de cómo implementar este sistema con facilidades para búsquedas parciales, refinamiento del camino, corrección de la predicción inicial y otras técnicas que hacen que esta biblioteca no solo sea muy útil para la inteligencia artificial sino que brindará opciones para configurar el rendimiento según las situaciones de cada proyecto.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 BÚSQUEDA DE CAMINOS

En los últimos años la Inteligencia artificial se ha convertido en una de las ramas de la computación más difundidas. Esta rama se dedica a la creación de hardware y software que imita el pensamiento humano, tratando de llevar a la computadora las amplias capacidades de nuestro razonamiento. La inteligencia artificial para juegos se basa en el uso de varias técnicas como son: sistemas expertos, máquinas de estado finito, sistemas de producción, árboles de decisión, **búsqueda de caminos**, lógica difusa, comportamientos de locomoción, etc. [11]

En este capítulo se explica la técnica de búsqueda de caminos, presentando los conceptos y el estado actual de los sistemas de búsqueda de caminos en aplicaciones de entornos virtuales.

La búsqueda de caminos es probablemente el problema más popular dentro de la inteligencia artificial en la industria de los video juegos y simuladores. A pesar de que existe una gran documentación del tema, el principal problema es que cada video juego o simulador tiene una implementación muy específica. La búsqueda de caminos tiene como objetivo principal encontrar la forma de llegar desde A hasta B. Donde A y B pueden ser dos puntos en un terreno, o A ser el punto actual y B ser un punto que representa el mejor lugar para esconderse, que tenga municiones cerca y sea más cercano al punto A, o A puede ser la posición actual en un tablero de ajedrez y B la jugada recomendada. La búsqueda de caminos utiliza los modelos cognitivos para representar los entornos virtuales; dependiendo de los algoritmos y de la heurística usada, mayor eficiencia tendrá el camino obtenido.

1.2 MODELOS COGNITIVOS

En el mundo real las personas tienen necesidad de recordar diferentes aspectos del mundo donde viven para desplazarse por este. Cada parte del entorno presenta sus propias características y brinda su propia información. Esta es analizada para desplazarse en el entorno. Para un agente inteligente en un entorno virtual, funciona de la misma forma, lo que esta información la extrae de una estructura de datos nombrada: Modelo cognitivo. [2]

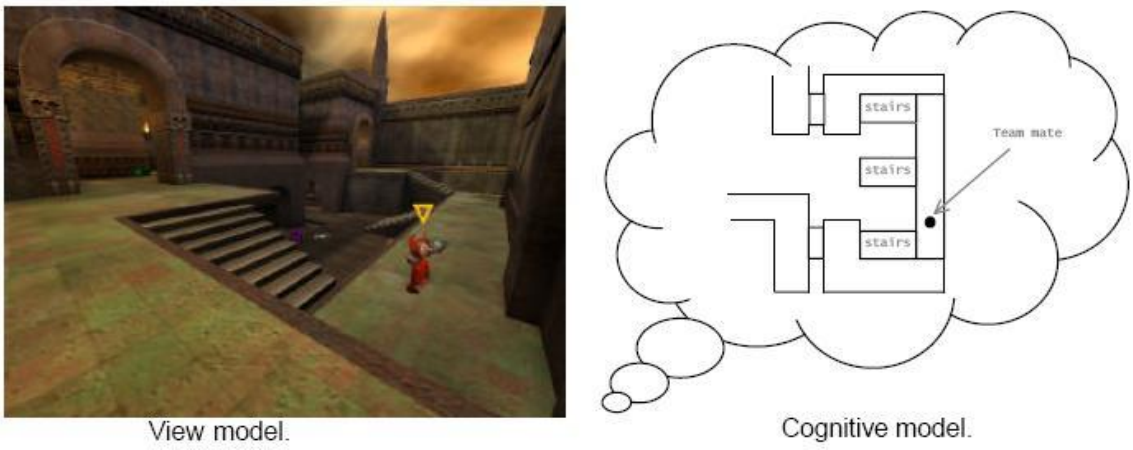

Figura 1: Representación del Modelo Cognitivo.

Los modelos cognitivos son una versión simplificada del mundo virtual, la mayoría solo incluye los detalles de la geometría (puertas, ventanas, puentes, portales, zona enemigas, etc.), además almacena el estado de la información como las puertas cerradas, caminos bloqueados o el tipo de enemigo que se está moviendo. La forma más común de representar un modelo cognitivo es en un grafo, pero en la práctica existen muchos tipos de modelos cognitivos, los más usados son:

1.2.1 REJILLA REGULAR

La rejilla regular se usa principalmente en la creación de entornos para juegos RTS (juegos de estrategia en tiempo real), tales como StarCraft, WarCraft y la Era de los Imperios, que son ambientes complejos basados en cuadrados o polígonos. Este modelo cognitivo se construye dividiendo el terreno en celdas, cada celda representa un nodo en el entorno, la celda puede ser transitable o no, no sería transitable, por ejemplo, si la celda forma parte de una pared, un árbol o algún otro obstáculo. Cada celda colinda con otras celdas, para saber cuáles celdas son las que colindan se construye la matriz de adyacencia. [2]

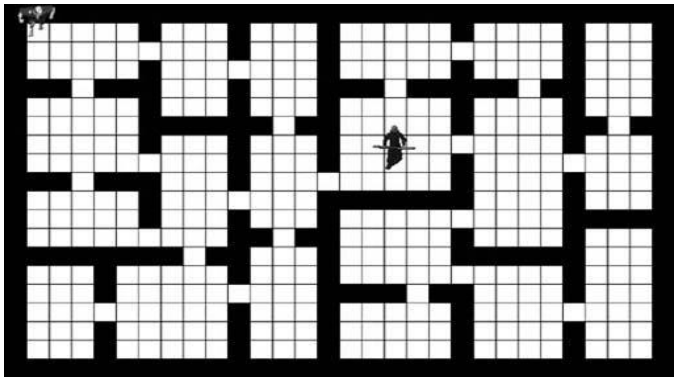

Figura 2: Representación de una rejilla regular.

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

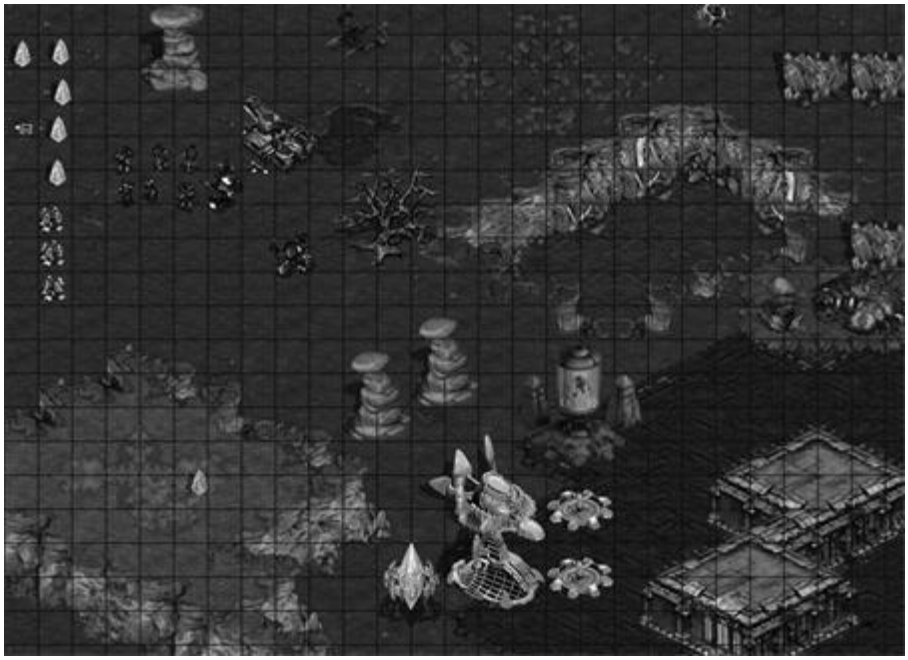

Figura 3: Uso de rejilla regular en el juego RTS: StarCraft.

En este modelo cognitivo cada celda representa árboles, murallas, enemigos, construcciones, agua, rocas, terreno, colinas, cuencas u otras geometrías que se encuentren en el entorno.

La principal ventaja de la rejilla regular es su fácil diseño e implementación, pero lo que más la desacredita es el tiempo de búsqueda, por ejemplo para un modesto mapa de celdas de 100 X 100, se necesita un grafo que contenga 10 000 nodos, aproximadamente 78 000 vértices, por eso en juegos modernos que requieren mayor nivel de computo no es aconsejable usar esta estructura de datos para representar el mundo.

1.2.2 PUNTOS DE VISIBILIDAD (POV)

El modelo cognitivo: Puntos de Visibilidad (POV) está formado por puntos en zonas importantes del entorno, puntos que son accesibles para los agentes, cada uno de estos puntos está conectado a aquellos puntos con los que sean capaces de formar una línea de visión. [2]

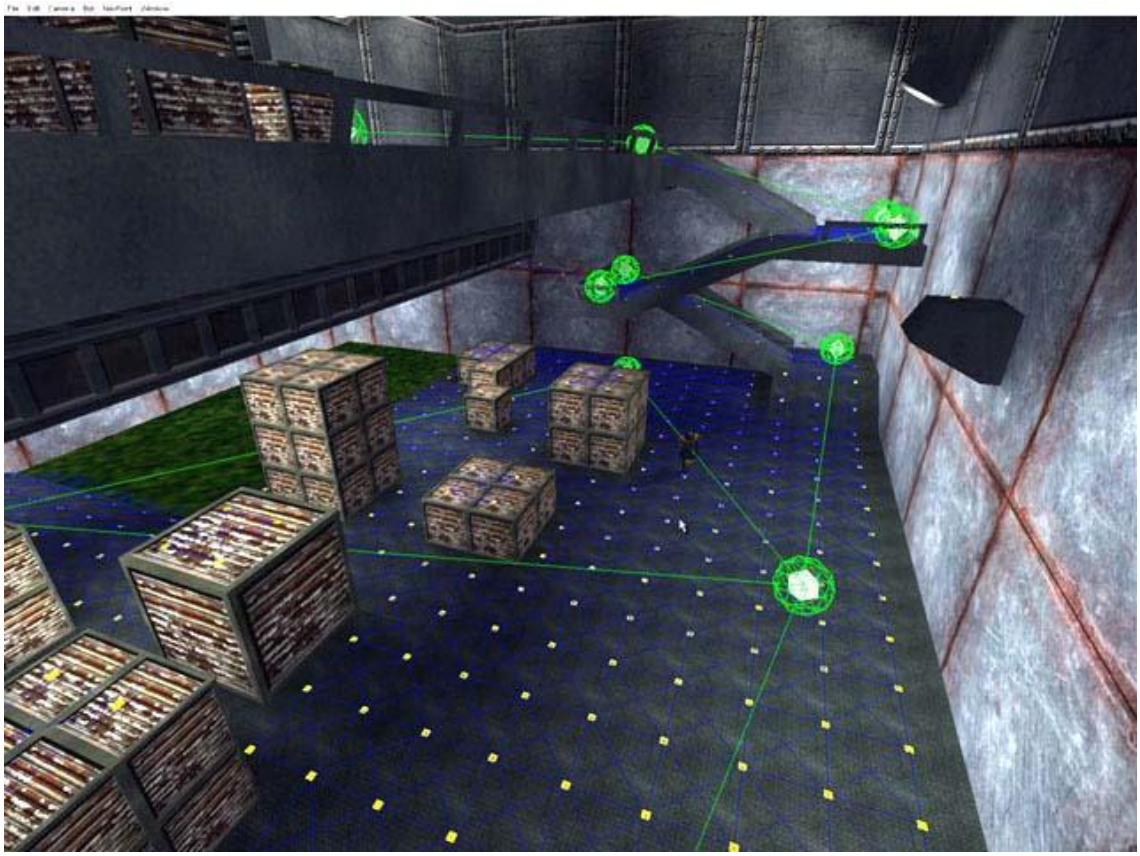

Figura 4: Puntos de Visibilidad.

Este modelo cognitivo es relativamente de fácil creación, se puede crear de forma manual o con una herramienta que lo haga de forma automática. Si el diseñador decide realizar el modelo cognitivo de forma manual puede colocar nodos en los lugares claves dentro del entorno y conectar aquellos que estén en el mismo campo de visibilidad, tiene que tener en cuenta no dejar espacios ciegos y no colocar nodos de más, pues entre más denso sea el grafo más lenta será la búsqueda. Si se decide crear una herramienta que cree el modelo de forma automática, el proceso sería el siguiente: se pone al menos un nodo en un punto clave, e intentar expandir este nodo en todas las direcciones posibles a las que se pueda ir, luego de varias iteraciones nos queda el grafo creado, este algoritmo se conoce con el nombre de **Flood Fill**.

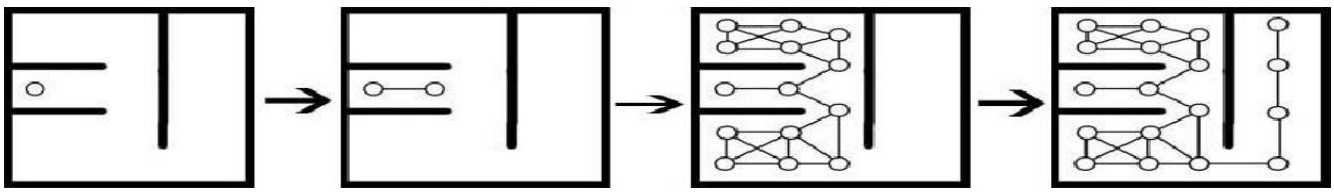

Figura 5: Creación del modelo cognitivo Puntos de Visibilidad usando el algoritmo: Flood Fill.

También se pueden crear otras herramientas que optimicen el grafo creado, por ejemplo: Siendo A un nodo y B es un nodo hijo de A, si A puede ver directamente a todos los hijos de B, no es necesario el nodo B.

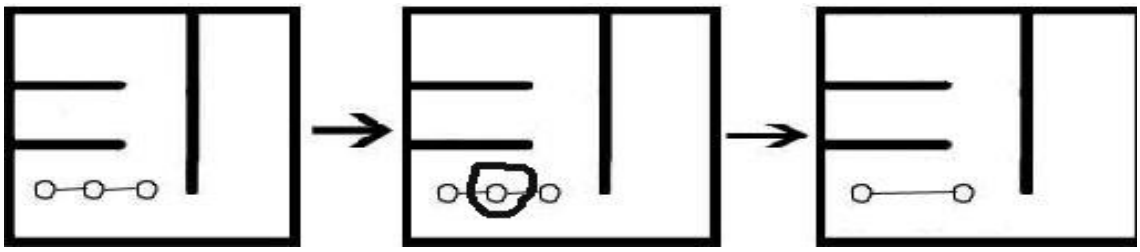

Figura 6: Optimización del grafo creado.

Las ventajas de este modelo cognitivo además de su fácil creación, radica en el bajo número de nodos que conforman este modelo, por ejemplo si se implementara este tipo de nodos en un mundo de StarCraft con 100 o 200 nodos bastaría, por lo que el tiempo de búsqueda es más rápido que otros modelos cognitivos. La mayor desventaja de este modelo es la detección de colisiones. Esto significa que cuando pedimos un camino, al transitar de un nodo a otro, no se garantiza que en esta trayectoria no pueda existir ningún obstáculo, por lo que es necesario realizar la detección de colisiones, que puede llegar a ser muy costosa en dependencia de la implementación y el tipo de entorno. Otra desventaja de este sistema son los puntos ciegos, que son aquellos puntos del terreno que debido a nuestro diseño del modelo, no se vea ningún nodo, esto suele suceder algunas veces a la hora de crear el modelo de forma manual.

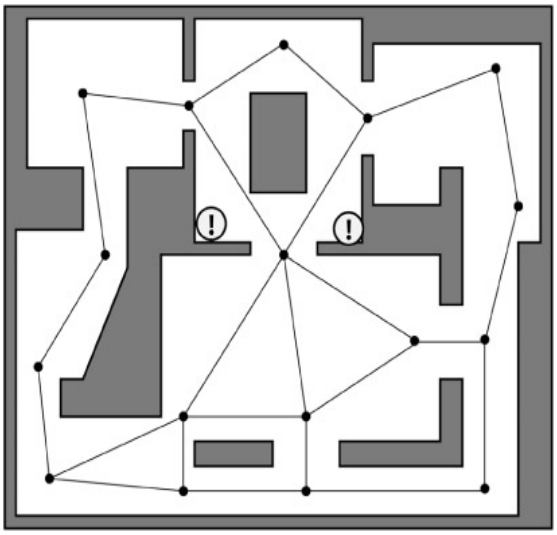

Figura 7: Representación de puntos ciegos en un grafo de búsqueda.

1.2.3 MALLAS DE NAVEGACIÓN

Uno de los modelos más usados para representar mundos 3D son las mallas de navegación, las cuales son conocidas como NavMesh. Una malla de navegación es un grupo de polígonos convexos que describen la superficie del ambiente 3D. Este es un simple, y altamente intuitivo modelo cognitivo donde los caracteres de la inteligencia artificial realizan un plan que pueden usar para la navegación y la búsqueda de caminos en el mundo. [2]

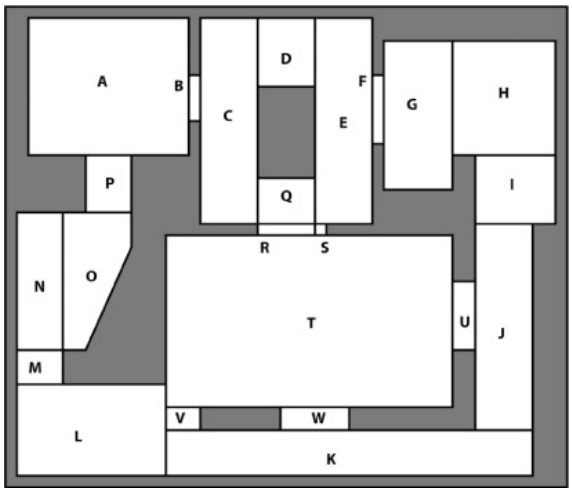

Figura 8: Malla de navegación.

Una buena idea presente en el concepto de malla de navegación es que el tamaño de la malla no depende del tamaño del mundo, pero sí de las figuras geométricas de las barreras o paredes y de la geometría del entorno. Los polígonos que conforman la malla pueden ser de varias formas: cuadrados, circulares, triangulares, etc. La técnica más usada por los diseñadores para crear este tipo de modelo cognitivo sigue los siguientes pasos: primero se diseña el entorno virtual, luego se selecciona la malla que conforma el suelo del ambiente, se crea un nuevo objeto a partir de esta y se optimiza quitándole los vértices que no afecten su forma, representando cada polígono una habitación, un corredor o simplemente una zona donde el agente pueda moverse libremente, entre menos polígonos tenga un modelo cognitivo, más sencilla y rápida será la búsqueda.

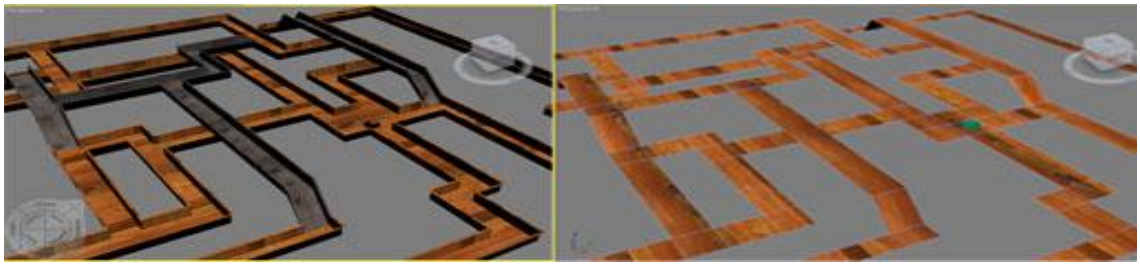

Figura 9: Ejemplo de construcción Malla de navegación.

La mayor ventaja de este modelo cognitivo radica en que como cada nodo representa un polígono convexo por donde el agente inteligente se puede mover libremente, no es necesario realizar la detección de colisiones en la trayectoria del camino obtenido. Otra ventaja es que se puede diseñar el modelo cognitivo de forma jerárquica, o sea, que esté constituido por una grafo de mallas de navegación en el cual, cada nodo representa otro grafo de mallas de navegación, por ejemplo, si quisiéramos representar el entorno virtual de La Ciudad de la Habana, sería muy engorroso y más lento la búsqueda si cada reparto fuera un nodo y la ciudad completa fuera un solo modelo cognitivo. Por el contrario, si lo diseñamos de forma tal que el modelo cognitivo raíz represente los municipios de la provincia, y que cada nodo de este modelo este constituido por otro modelo cognitivo que represente los repartos de ese municipio, y que cada nodo de ese último modelo esté formado por otro modelo cognitivo que represente las cuadras, conseguiríamos con este diseño jerárquico de 3 niveles que la búsqueda sea más rápida y eficiente.

Una desventaja puede ser al construir el modelo cognitivo para encontrar todos los polígonos por el que un agente inteligente pueda moverse.

1.3 ALGORITMOS MÁS COMUNES

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

1. Encontrar en el modelo cognitivo el nodo más cercano a la posición actual del agente, este sería el nodo A.
2. Encontrar en el modelo cognitivo el nodo más cercano a la posición del destino del agente, este sería el nodo B.
3. Usar algoritmos de búsqueda para encontrar el mejor camino entre A y B.
4. Mover el agente al nodo A.
5. Mover el agente al nodo B por el camino calculado en el paso 3.
6. Mover al agente del nodo B al destino.

En el paso 3 se hace alusión a los algoritmos de búsqueda de caminos, 2 de los algoritmos más usados son: Dijkstra y el A*

1.3.1 ALGORITMO DIJKSTRA

El **algoritmo de Dijkstra**, también llamado **algoritmo de caminos mínimos**, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo dirigido y con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. [14]

Para poder realizar este algoritmo debemos contar con lo siguiente:

- El nodo donde se comienza el camino llamado: **Origen**.
- El nodo que representa la meta del camino llamado: **Destino**.
- Una cola de prioridad de nodos organizada por la variable **CostoAcumulado** llamada: **Abierta.**

A continuación la implementación del algoritmo en seudocódigo:

```
Origen. CostoAcumulado = 0
Origen.Padre = null
Abierta.Push Origen
While Abierta is not empty do
        {
         Nodo = Abierta.Pop
         If Nodo == Origen do
                Retorna true
         For Hijo in Nodo.Conecciones do
         {
                        CostoAcumulado = Nodo.CostoAcumulado + Hijo.Costo

                        If Hijo is in Abierta and Hijo.CostoAcumulado <= CostoAcumulado do
                          Continue

                Hijo. CostoAcumulado = CostoAcumulado
                Hijo.Padre = Nodo
                If Hijo is not in Abierta do
                        Abierta.Push Hijo
         }
}
Retorna false
```

La principal desventaja de este algoritmo es que se exploran muchos caminos, gastándose tiempo y recursos porque no se cuenta con muchos datos para evaluar el costo de cada nodo.

1.3.2 ALGORITMO A*

El A* (A-estrella o A-start por los anglosajones) es el más usado y popular de los algoritmos de búsqueda en grafos por su eficiencia y eficacia, existe un gran número de literatura en internet que lo documentan. El mismo es una variación del algoritmo de Dijkstra, que inserta el uso de una lista de nodos que contiene los nodos que han sido visitados e introduce la heurística para calcular el costo acumulado de cada nodo mostrándose a continuación la nueva forma de calcularlo. [15]

- **Nodo.CostoAcumulado** = **Padre.CostoAcumulado** + **Nodo.Costo** + **Heurística**.

En la implementación estándar de este algoritmo el valor de la heurística se obtiene al calcular la distancia del nodo origen al nodo destino. Además debemos contar con los siguientes atributos para su implementación:

- El nodo donde se comienza el camino llamado: **Origen**.
- El nodo que representa la meta del camino llamado: **Destino**.
- Una cola de prioridad de nodos organizada por la variable **CostoAcumulado** llamada: **Abierta.**
- Una lista de nodos llamada: **Cerrada.**

A continuación la implementación del algoritmo en seudocódigo:

```
Heurística = DistanciaEntre Origen Destino
Origen.CostoAcumulado = Heurística
Origen.Padre = null

Abierta.Push Origen
While Abierta is not empty do
        {
                Nodo = Abierta.Pop
                If Nodo == Origen do
                        Retorna true

                For Hijo in Nodo.Conecciones where Hijo is not in Cerrada do
                {
                        Heurística = DistanciaEntre Hijo Destino
                        Hijo.CostoAcumulado = Nodo.CostoAcumulado +  Hijo.Costo + Heurística
                        Hijo.Padre = Nodo
                        Abierta.Push Hijo
                }
                If Nodo is not in Cerrada do

                    Cerrada.Add Nodo
```

}
Retorna **false**

Esta implementación estándar tiene como problema el uso de una heurística muy restringida, para calcular este valor solo se usa la distancia entre el nodo actual y el nodo destino.

1.4 HEURÍSTICA

La heurística es un nuevo factor que se incluye en este algoritmo, y es donde radica su eficiencia. Entre más específica sea la heurística, mejor y más exacta será la búsqueda a realizar, porque este factor nos ayuda a expandir menor cantidad de caminos para llegar a la solución, descartando los nodos de búsqueda de mayor coste y más alejados de la solución. Si la heurística es muy mala en el peor de los casos el A* se comporta como el algoritmo de Dijkstra. [2]

La Inteligencia Artificial aborda problemas poco estructurados, donde de antemano no se conoce 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, es un valor que se obtiene para determinada situación, y nos dice qué tan eficiente es ese estado para esa situación. [4]

1.4.1 HEURÍSTICA EUCLIDIANA

La heurística euclidiana es la distancia geométrica de un punto a otro punto, esta heurística es de suma importancia en la búsqueda de caminos porque evita que nodos que se alejan del destino sean expandidos. Esta es la heurística más usada en el algoritmo de búsqueda A* para encontrar un camino hasta un punto en específico. [2]

1.4.2 HEURÍSTICA DE LA DISTANCIA DE MANHATTAN

La distancia de Manhattan es muy popular entre los juegos basados en modelos cognitivos de rejillas regulares como StarCraf y WarCraft. Esta distancia se calcula sumando las celdas de las rejillas verticales y las horizontales entre el punto origen y el punto destino. La ventaja de esta distancia es que no es necesario realizar la costosa operación matemática de hallar la raíz cuadrada. [4]

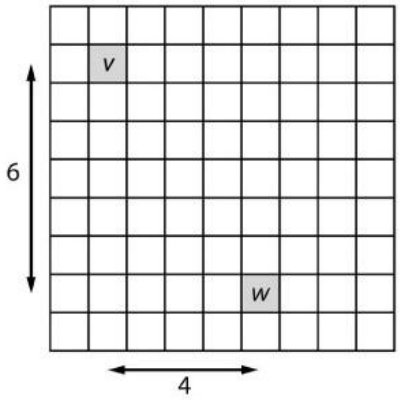

Figura 10: Muestra de cómo se calcula la distancia euclidiana.

1.4.3 HEURÍSTICA EVALUAR PUNTO EN TERRENO

En los entornos virtuales se hace necesario para los agentes inteligentes encontrar los mejores sitios para realizar una emboscada, para esconderse, el mejor lugar para un francotirador entre otros, por lo que es necesario saber cuáles nodos son los que brindan mayor ventaja estratégica para estos puntos. Esta información se suele pre-calcular y guardar en los modelos cognitivos, así la heurística solamente tiene que evaluar los valores pre-calculados. [18]

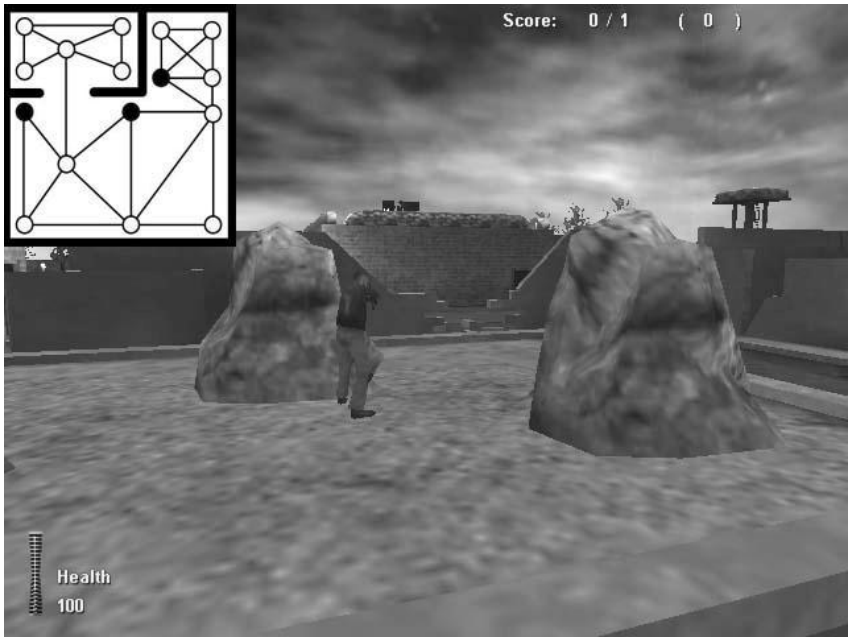

Figura 10: Representación de un punto de emboscada.

1.5 TÉCNICAS DE OPTIMIZACIÓN

Como el actual mercado de video juegos exigen que se manejen mayor volumen de recursos para que las simulaciones sean lo más reales posible, junto a los algoritmos de búsqueda Dijkstra y A* se han asociado técnicas de optimización para que la tarea de búsqueda de caminos sea más eficiente, tres de estas son: Suavizado del camino (Path Smoothing), Distribución de tiempo (Time Slicing) y Jerarquía de caminos (Hierarchical Pathfinding).

1.5.1 SUAVIZADO DEL CAMINO

Esta técnica se utiliza una vez obtenido el camino para quitar nodos innecesarios que ocupan recursos y hacen que el camino no parezca real. Los pasos de esta técnica son los siguientes:

- Primer paso: tomar el primer nodo llamado A y el nodo que le sigue llamado B
- Segundo paso: chequear si se puede ir del nodo A al nodo que le sigue a B
- Si se puede ir entonces eliminar el nodo B del camino y llamar al tercer nodo como nodo B y volver a repetir desde el segundo paso
- Si no, ir al nodo que le sigue a B, dices que el nodo B es este nodo y volver a repetir desde el paso 2
- Una vez que se chequea el nodo A con todos los nodos del camino, se corre el nodo A al nodo que le sigue a este y se repite desde el paso 2, así sucesivamente hasta que se hayan chequeado todos los nodos.

Para saber si se puede ir del nodo A al nodo que le sigue a B se lanza un rayo partiendo de A en dirección al nodo que le sigue a B y si antes de llegar a este último nodo no ha colisionado con ningún obstáculo es que del nodo A se puede ir directamente al nodo que le sigue a B. [2]

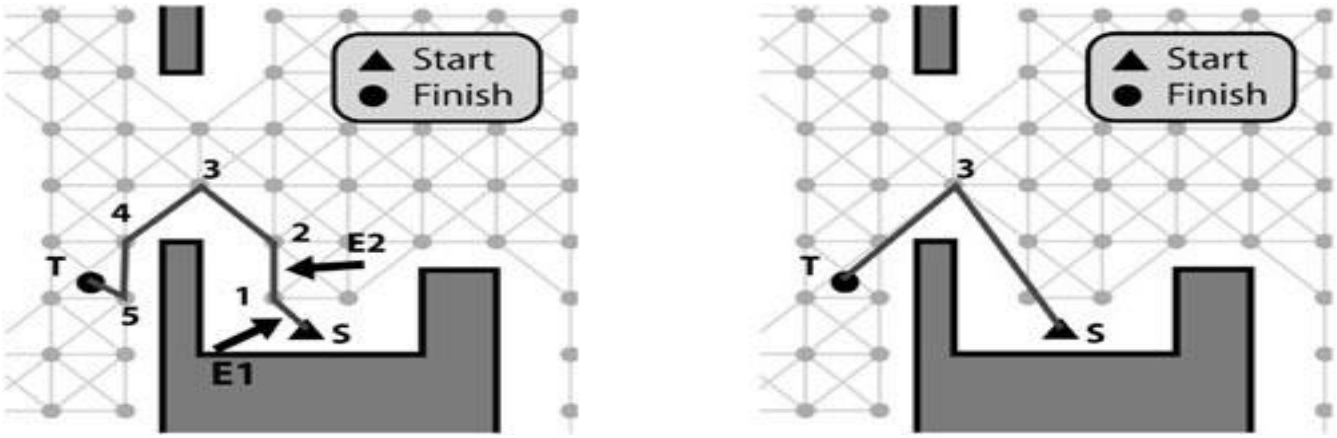

Figura 11: Ejemplo de aplicar técnica de suavizado del camino.

1.5.2 DISTRIBUCIÓN DE TIEMPO

Como actualmente hay muchos recursos que se manejan en una simulación y muchas tareas y cálculos costosos es necesario que la búsqueda de caminos no consuma recursos que son necesarios para estas otras tareas como la de pintar, atender la red o calcular la física. Esta técnica se usa para evitar que suceda esta sobrecarga y consiste en que el planificador de búsqueda de caminos reparte el número de ciclos que puede iterar entre la cantidad de solicitudes de búsqueda de caminos que tenga, de esta forma se computa todas las solicitudes de caminos y no se sobrecarga el sistema. Esta técnica es aconsejable en aquellos sistemas que tienen cientos de agente inteligentes que solicitan constantemente búsquedas de caminos, principalmente para los juegos de consolas que tienen las restricciones del hardware. [2]

1.5.3 JERARQUÍA DE CAMINOS

Cuando los entornos virtuales son muy grandes, por ejemplo el juego GTA que está ambientada en ciudades completas, sería muy costoso realizar una búsqueda en un solo modelo cognitivo, porque tendría más de 100 000 nodos. Para evitar esto se recomienda jerarquizar los modelos cognitivos, o sea, tener un modelo cognitivo para representar regiones más amplias y esas regiones dividirlas en menores regiones, cada una siendo un modelo cognitivo en sí y siendo nodos del modelo cognitivo de las regiones más amplias a la que pertenece, de esta forma las búsquedas serían locales en cada modelo cognitivo, extrayendo del árbol jerárquico el camino completo. [2]

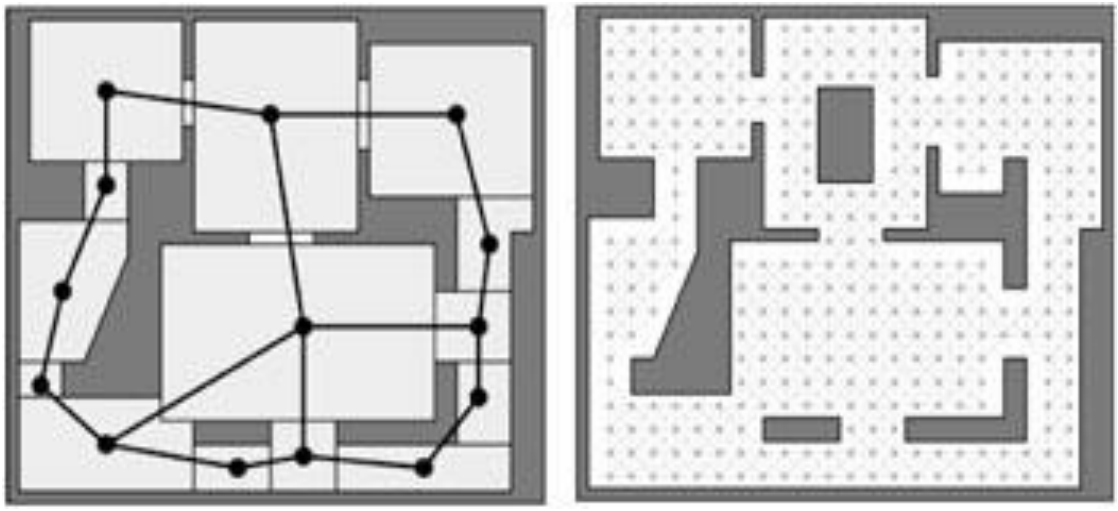

Figura 12: Jerarquía de modelos cognitivos. a) Primer nivel b) Segundo nivel

1.6 BIBLIOTECAS ACTUALES

Actualmente los sistemas profesionales que se usan para realizar video juegos y simulaciones son varios, algunos cuentan con herramientas de búsqueda de caminos, pero estas tienen el problema de no poder adaptarse a otros sistemas y su costo ser elevado. Existe una herramienta en el mercado dedicada a la búsqueda de caminos llamada: **PathEngine** la descripción que dan sus creadores es la siguiente: “es un sofisticado middleware comercial con un juego de herramientas para la implementación de movimientos de agentes inteligentes, construido alrededor de la búsqueda de caminos en superficies terrestres tridimensionales basadas en puntos de visibilidad. Posee un modelo de movimientos de agente bien definido, con búsqueda de caminos y detección de colisiones, ambos estrechamente relacionados. Este modelo permite búsquedas de caminos extremadamente rápidas en entornos complicados, con soporte para obstáculos situados dinámicamente y teniendo en cuenta la forma del agente. Viene con un poderoso procesador de contenidos (geometrías del mundo), funcionalidades de seguimiento de terrenos, una suite de prueba y herramientas asociadas. El espacio de estados manejado es continuo, permite la generación de caminos curvos y brinda soporte multi-hilos”. [12]

Esta herramienta es usada por varios proyectos en desarrollo, pero cuenta con algunos problemas como son el uso de técnicas modernas de optimización, tiene una heurística muy restringida y su licencia es propietaria oscilando en dependencia de las opciones básicas en los 4000 euros y la opción completa 13 000 euros.

1.7 UNIVERSIDAD DE CIENCIAS INFORMÁTICAS

En la Universidad de las Ciencias Informáticas se han desarrollado varios trabajos acerca de este tema, algunos títulos son: Biblioteca de inteligencia artificial para sistemas de realidad virtual, Herramienta de

creación de grafos de caminos para la biblioteca SceneToolKit y Búsqueda de caminos en entornos virtuales. Aunque todos estos trabajos constituyen aportes no se aprecia en ellos el uso de las últimas técnicas de optimización, y tampoco se cuenta con un producto real, una herramienta que sea genérica y pueda ser usada en los proyectos de la universidad actuales o futuros y además sea capaz de irse actualizando según las nuevas necesidades de estos proyectos.

CAPÍTULO 2: SOLUCIÓN PROPUESTA

En este capítulo se analizará la situación problemática y se plantearán soluciones técnicas para el diseño y funcionamiento de la biblioteca genérica de búsqueda de caminos.

2.1 ANALIZAR PROBLEMA

Dentro de una simulación de un entorno virtual donde interactúen agentes inteligentes, una de las principales tareas que deben cumplir estos agentes es la de moverse en el entorno y saber por los puntos que debe pasar para llegar a los lugares de interés, por ejemplo los lugares donde encuentre armas u objetos que incrementen sus cualidades como la rapidez, la salud o la fortaleza. Para poder implementarles esta capacidad a los agentes debemos analizar cómo funciona nuestro mundo y tratar de llevar este principio al diseño de la inteligencia artificial del sistema a implementar.

Para estudiar cómo pensamos los seres humanos analizaremos el siguiente ejemplo de la vida real en la UCI. Cuando un estudiante llega a la universidad por primera vez, este desconoce por completo el entorno en el que se encuentra, pero él necesita alguna forma de conocer donde quedan los lugares a los que debe visitar. Sería muy trabajoso para el estudiante tener que recorrer todo el entorno para poder memorizar donde queda cada cosa, esto consumiría cantidad de tiempo y esfuerzo de su parte. Por suerte existe una persona encargada de orientar a los estudiantes en el momento de su llegada. Esta persona le pregunta al estudiante donde quiere ir, (por lo general al edificio de su residencia) y le entrega un mapa del entorno, el cual analiza y le muestra por los lugares del mapa donde debe coger para llegar a su destino. Esta persona para escoger el camino tiene en cuenta en dependencia de la solicitud de donde quiera ir el estudiante la evaluación de algunos factores como por ejemplo por donde es más cerca, por donde será mejor para no perderse, a lo mejor tiene en cuenta que el camino pase por un punto de referencia que le sea fácil al estudiante, o si el estudiante desea ir a alguna cafetería, cual es la que tiene mayor oferta, o cual servicio es más rápido. Como en ocasiones a la persona encargada de orientar a los estudiantes en su llegada se le agrupan varios estudiantes a la vez pidiendo ser orientados, esta persona debe valerse de algún mecanismo para atenderlos a todos y que no se le agrupen demasiado, ni ella exceda sus límites. Por ejemplo pudiera preguntar a todos los que fueran a un mismo destino o destinos cercanos y enseñar el mismo camino a todos.

2.2 PROPUESTA DEL SISTEMA.

En el ejemplo anterior un estudiante se orienta para poder ir a los lugares de la universidad a la que acaba de llegar, si pensamos bien en una simulación de entornos virtuales los agentes inteligentes pudieran ser igual que el estudiante, que necesitan navegar en el entorno que desconocen. Por lo que pudiéramos identificar los elementos y sus responsabilidades involucrados en el ejemplo anterior, para basarnos en estos y realizar el diseño de nuestro sistema.

Tabla 1: Identificación de elementos y sus responsabilidades.

Elemento	Responsabilidad
Estudiante	Realiza peticiones de caminos hacia destinos.
Persona recibe Estudiante	Atiende peticiones de caminos.
Mapa	Representa el entorno.
Lugar del Mapa	Representa un elemento en específico del Mapa.
Analiza mapa	Busca en el mapa cual es el mejor camino.
Evaluación de Factores	Decisiones que se tienen en cuenta para escoger el mejor camino.
Camino	Conjunto de lugares del mapa por donde debe coger el estudiante para llegar al destino.
Mecanismo para atender	Técnica que sigue la persona para realizar su función de forma óptima.

De la tabla anterior hacemos las siguientes deducciones:

- Existe un agente inteligente (Estudiante) que es independiente al sistema de búsqueda, este la única comunicación que tiene es hacerle peticiones de caminos al sistema.
- El sistema se representa por el planificador de caminos (Persona recibe Estudiante), este es el encargado de atender las peticiones de búsqueda a cada uno de los agente inteligentes.
- El planificador de caminos tiene un modelo cognitivo (Mapa) que es un grafo que describe el entorno.
- El modelo cognitivo está formado por un conjunto de nodos (Lugar del Mapa).
- Para obtener el camino se analiza el modelo cognitivo mediante algún algoritmo de búsqueda.
- En la tarea de generar un camino se debe tener en cuenta la evaluación de la heurística que se desee (Evaluación de Factores).
- El planificador de caminos tiene implementado técnicas de optimización para evitar la sobrecarga, como por ejemplo la distribución de tiempo entre todas las peticiones de búsqueda.

2.3 OBTENCIÓN DEL MODELO COGNITIVO

En el ejemplo de la vida real que nos basamos, para la persona encargada de orientar a los estudiantes poder mostrar el camino debía contar con un mapa, este había sido generado a partir de planos de la universidad e impreso en alguna impresora de la escuela. Esto traducido al sistema de búsqueda se convierte a que el planificador de caminos debe contar con un modelo cognitivo.

El modelo cognitivo se puede obtener en el mismo software de diseño 3d que se use para obtener el entorno, por ejemplo el 3D Max Studio. El proceso de obtención sería el siguiente: una vez diseñado el entorno, seleccionamos los polígonos que conforman la base del mismo, o sea, el suelo por donde los agentes suelen transitar y estos polígonos son los que usamos para construir el modelo cognitivo. Si

usamos un modelo cognitivo de tipo puntos de visibilidad ponemos puntos encima de estos polígonos que representan cambios de dirección en la trayectoria o que unan otros puntos a los que no se tiene acceso directo producto a la existencia de obstáculos. Si el tipo de modelo cognitivo es mallas de navegación se toman los polígonos seleccionados y se optimizan de forma que cada polígono represente un cuarto o un pasillo por el que se pueda transitar libremente.

Una vez diseñado el modelo cognitivo se debe exportar los datos del software de diseño a un fichero con un formato conocido para que el sistema sea capaz de crear el modelo cognitivo a partir de este fichero. Cada modelo cognitivo exporta su propio formato.

2.3.1 FORMATO DEL FICHERO DEL MODELO COGNITIVO: PUNTOS DE VISIBILIDAD

El formato del fichero que conforma el modelo cognitivo de tipo puntos de visibilidad es el siguiente:

Descripción de la estructura :	**Ejemplo:**
Tipo de modelo cognitivo	"ModeloCognitivo_POV"
Número de puntos de visibilidad	5
Nombre de cada punto de visibilidad y posición x, y, z.	Pov00 -23.2257 0.0 -8.02948 Pov01 5.88176 0.0 -8.02948 Pov02 10.9049 0.0 -8.02948 Pov03 15.9281 0.0 -8.02948 Pov04 20.9513 0.0 -8.02948
Cantidad de conexiones que tiene cada punto de visibilidad y el índice de los puntos de visibilidad a los que se conecta.	1 1 2 0 2 2 1 3 2 2 4 1 3

2.3.2 FORMATO DEL FICHERO DEL MODELO COGNITIVO: MALLAS DE NAVEGACION

El formato del fichero que conforma el modelo cognitivo de tipo mallas de navegación es el siguiente:

Descripción de la estructura :	**Ejemplo:**
Tipo de modelo cognitivo.	"ModeloCognitivo_NavMesh"
Número de vértices que forman la malla completa.	10
Posición de cada vértice.	-23.2257 0.0 -8.02948 5.88176 0.0 -8.02948 10.9049 0.0 -8.02948 15.9281 0.0 -8.02948 20.9513 0.0 -8.02948 -23.2257 0.0 -3.05918 5.88176 0.0 -3.05918 10.9049 0.0 -3.05918 15.9281 0.0 -3.05918 20.9513 0.0 -3.05918
Cantidad de polígonos.	4

Conjunto de 4 índices que forman un polígono.	0 1 6 5 1 2 7 6 2 3 8 7 3 4 9 8
Conjunto de 2 índices de polígonos que comparten una arista y los 2 índices de los vértices que forman esta arista.	0 1 1 6 1 2 2 7 2 3 3 8

2.4 ALGORITMO DE BÚSQUEDA

En el ejemplo anterior la persona encargada de atender a los estudiantes para hallar el camino a estos debía analizar el mapa. Este proceso de análisis se traduce a la implementación de un algoritmo de búsqueda. De los algoritmos estudiados el más eficiente y popular es el A*, cuyo funcionamiento se explicó anteriormente en el epígrafe 1.3.2 Algoritmo A*. Aunque este algoritmo es muy eficiente su diseño no permite insertar en la cola de prioridad de nodos abiertos un nodo que esté en la lista de nodos cerrados, impidiendo este tipo de búsqueda: Supongamos que queremos el mejor camino para ir desde el punto A al punto B pero al mismo tiempo necesitamos en el transcurso del camino coger un ítem de comida.

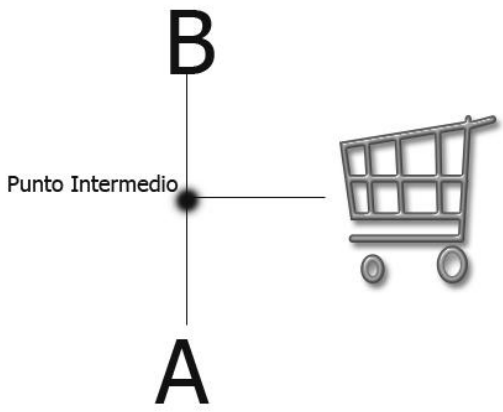

Figura 13: Camino de A hasta B pasando por el ítem de comida.

Con la actual implementación del algoritmo A* no es posible realizar una sola petición de búsqueda para el ejemplo anterior, tendríamos que hacer una petición de camino del punto A hasta el ítem de comida y otra petición del ítem de comida al punto B, consumiendo tiempo, recurso y facilidad de trabajo. Por lo que decidimos realizarle un cambio a la implementación del algoritmo de búsqueda A* para hacerlo capaz de usar múltiples heurísticas y poder hacer una sola petición de búsqueda. Para hacer la implementación debemos contar con lo siguiente:

- Lista de los nodos por donde se puede comenzar el camino llamado: **listaOrigen**.
- Lista de los nodos a los que queremos llegar llamado: **listaDestino**.
- Una cola de prioridad de nodos organizada por la variable **CostoAcumulado** llamada: **Abierta.**
- Una lista de nodos llamada: **Cerrada.**
- Una lista de la heurística a usar llamada **listaHeurística**.

A continuación se muestra el seudocódigo de la nueva implementación de este algoritmo.

```
For Nodo in listaOrigen do
   For heurística in listaHeurística do
   {
        Nodo.CostoAcumulado += heurística.DameValor()
        Nodo.Padre = null
        Abierta.Push Nodo
   }

   While Abierta is not empty do
            {
                     Nodo = Abierta.Pop
                     If Nodo == Origen do
                              Retorna true

                     If  (Nodo is not in Cerrada ) or
                     (Nodo is in Cerrada and  (Nodo in Cerrada).CostoAcumulado > Nodo.CostoAcumulado)
                     do
                     {
                              For Hijo in Nodo.Conecciones do
                              {
                                       Hijo.CostoAcumulado = Nodo.Costo +  Hijo.Costo
                                       Hijo.Padre = Nodo

                                       For heurística in listaHeurística do
                                                Hijo.CostoAcumulado +=  heurística.DameValor()

                                       Abierta.Push Hijo
                              }

                      If Nodo is in Cerrada do
                              (Nodo in Cerrada).CostoAcumulado = Nodo.CostoAcumulado
                     Else
                              Cerrada.Add Nodo
            }
   Retorna false
```

2.5 TÉCNICAS DE OPTIMIZACIÓN USADAS

En el ejemplo del estudiante la persona encargada de orientarlos tenía que tener un mecanismo para atender las peticiones de todos los estudiantes y hacerlo de la forma más óptima. En nuestro sistema es necesario optimizar el sistema de búsqueda para que aunque exista un gran número de peticiones por parte de los agentes inteligentes no se consuman recursos que son necesarios para la simulación en otras tareas de igual o más importancia. Una de las técnicas que se emplea es la distribución de tiempo, esta consiste en que el planificador de camino tiene una variable que equivale al número de iteraciones que puede realizar el proceso de búsqueda en cada llamado a actualizar el estado de las peticiones. Esta variable se reparte entre la cantidad de peticiones existentes. Por ejemplo si la variable es de 100 y existen 4 peticiones de búsqueda de caminos, en cada llamado a actualizar el proceso de búsqueda del planificador de caminos este le dirá a cada petición de camino que itere 100/4 veces.

Cuando existan un gran número de peticiones de caminos al planificador y la variable de la cantidad de iteraciones que puede hacer el proceso de búsqueda no tenga un valor muy grande, ocurrirá que el proceso de análisis del camino pedido tardará un poco, pero el agente no se puede quedar parado por lo que se usa la técnica de caminos parciales. El camino parcial consiste en que a medida que el agente le pide al camino generado el siguiente punto al que debe dirigirse, si el camino no está completo todavía le da el punto del camino que en esos momentos es el más óptimo. Esto trae consigo que una vez que se termine de calcular el camino si el agente no se encuentra en la trayectoria de este se calcula instantáneamente un nuevo camino al camino real obtenido.

Otra técnica que se usa es el suavizado del camino explicada en el epígrafe: 1.5.1 Suavizado del camino, esta técnica se usa para quitar nodos innecesarios del camino obtenido.

2.6 EL SISTEMA COMO BIBLIOTECA GENÉRICA DE BÚSQUEDA DE CAMINOS

Luego de todas las deducciones que en epígrafes anteriores se hicieron se decidió hacer un subsistema independiente del resto de la aplicación, con la única función de mostrarles a los agentes inteligentes los puntos por los que deberían transitar para alcanzar su destino. La biblioteca brinda facilidad de extensibilidad, para que en el futuro se le puedan agregar nuevas funcionalidades, así como nuevos tipos de modelos cognitivos y el proceso de búsqueda sea el mismo sin necesidad de re-implementarlo para casos específicos. Para cumplir lo antes dicho los elementos que representan a los modelos cognitivos, los nodos de los modelos cognitivos, y la heurística cuentan con interfaces que facilitan la comunicación con nuevos elementos que se agreguen en el futuro.

CAPÍTULO 3: CARÁCTERISTICAS DEL SISTEMA

En este capítulo se comienza a tener una visión más práctica del sistema a desarrollar. Se definen las reglas del negocio y el modelo de dominio. Se obtienen los requisitos funcionales (capacidades o funciones que el sistema debe cumplir) y los no funcionales (propiedades o cualidades que el producto debe tener).

3.1 Modelo del dominio.

En la siguiente figura se muestra el modelo del dominio, el cual muestra los conceptos necesarios para la solución que se pretende construir, estos conceptos se explican en el siguiente epígrafe.

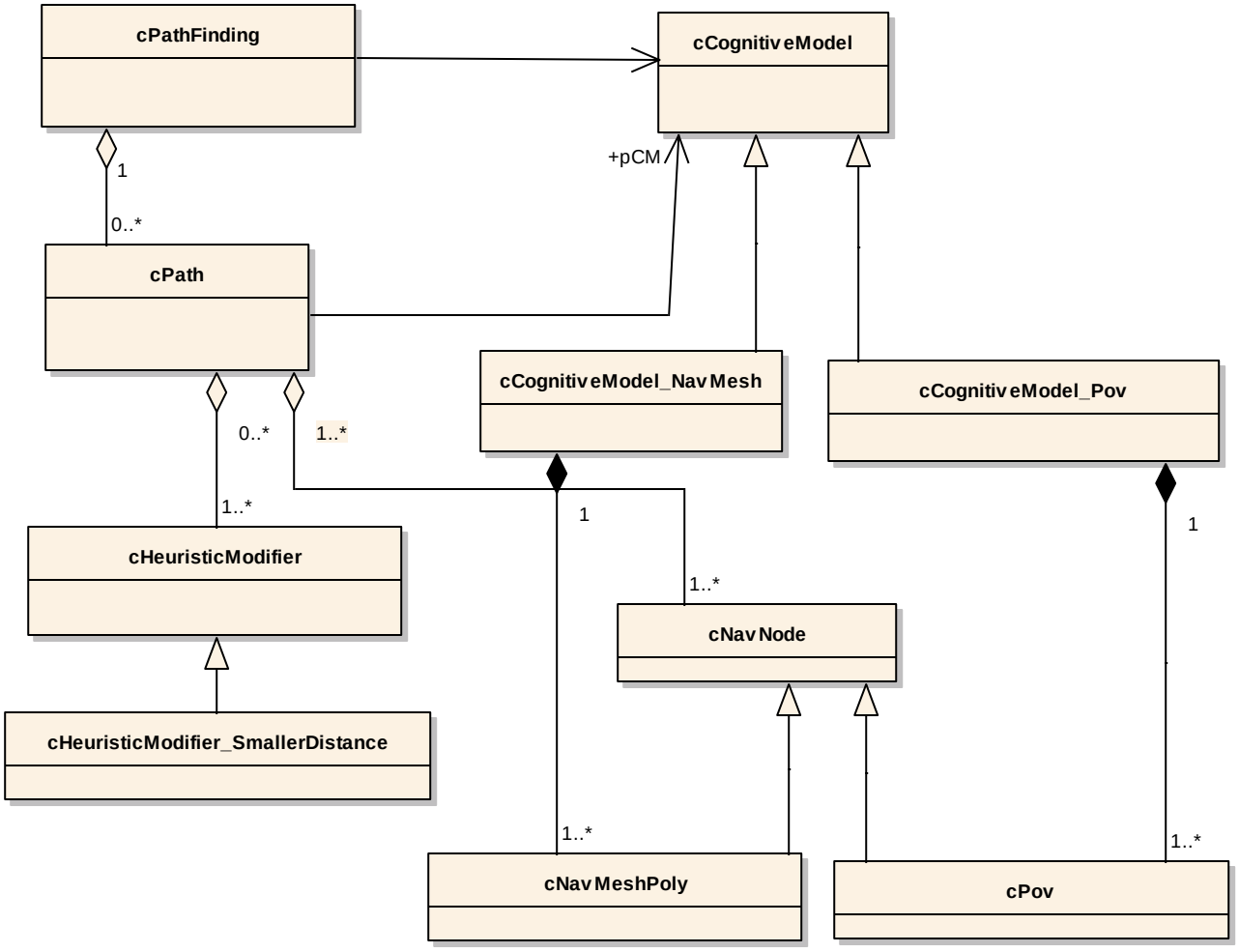

Figura 14: Modelo del dominio.

3.2 GLOSARIO DE TÉRMINOS

- **cPathFinding:** Controlador o manipulador de las peticiones de búsqueda que soliciten los agentes inteligentes.
- **cPath:** Entidad que contiene los datos del camino hacia el destino solicitado.
- **cCognitiveModel:** Grafo que contiene la descripción de las áreas o puntos que conforman el entorno.
- **cCognitiveModel_Pov:** Tipo de grafo de caminos formado por nodos que representan puntos del terreno.
- **cCognitiveModel_NavMesh:** Tipo de grafo de caminos formado por nodos que representan áreas del terreno.
- **cNavNode:** nodo del grafo del terreno, representa un área o punto de este.
- **cPov:** Tipo de nodo del grafo que representa un punto del terreno.
- **cNavMeshPoly:** Tipo de nodo del grafo que representa un área del terreno.
- **cHeurísticModifier:** Obtiene el valor de la heurística para un nodo en específico.
- **cHeurísticModifier_SmallerDistance:** Evalúa la distancia entre el nodo seleccionado y los nodos destinos.

3.3 CAPTURA DE REQUISITOS

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir y los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. A continuación se expondrán los requisitos funcionales y los no funcionales del sistema.

3.3.1 REQUISITOS FUNCIONALES

R.1 Cargar modelo cognitivo de un fichero de texto

R.2 Realizar petición de búsqueda

R.3 Obtener camino parcial

R.4 Usar técnica de distribución de tiempo

R.5 Refinar el camino

R.6 Eliminar petición de búsqueda de camino

R.7 La solución de la búsqueda debe ser la más óptima.

3.3.2 REQUISITOS NO FUNCIONALES

Usabilidad: Los futuros usuarios del sistema serán programadores con conocimientos básicos con respecto al tema de inteligencia artificial. El producto debe ser concebido para que el usuario piense qué desea hacer y no cómo hacerlo.

Rendimiento: Debe ser una aplicación en tiempo real donde tenga alta velocidad de procesamiento de los cálculos, tiempo de respuesta y recuperación del sistema.

Soporte: Debe ser compatible con la plataforma Windows, pero con pequeñas modificaciones puede migrar hacia Linux.

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

Diseño e implementación: Se regirá por la filosofía de Programación Orientada a Objetos. Se hará uso de la STL de C++ para representar las estructuras de datos.

Extensibilidad: El sistema es muy extensible dando la posibilidad a los programadores que hagan uso de la biblioteca de incluir nuevos modelos cognitivos y evaluadores de la heurística.

3.4 MODELO DE CASOS DE USOS DEL SISTEMA.

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

3.4.1 ACTOR DEL SISTEMA

Tabla 2: Justificación de los actores.

Actores	Justificación
Aplicación	Es la que hará uso de la biblioteca, eliminando y haciendo, peticiones de búsqueda.

3.4.2 CASOS DE USO DEL SISTEMA

- Cargar modelo cognitivo.

- Realizar búsqueda.
- Escoger heurística.
- Buscar nodos origen y destino.
- Actualizar búsqueda.
- Iterar.
- Ejecutar algoritmo de búsqueda.
- Reencontrar camino.
- Refinar camino.
- Eliminar camino.

3.4.3 DIAGRAMA DE CASOS DE USO.

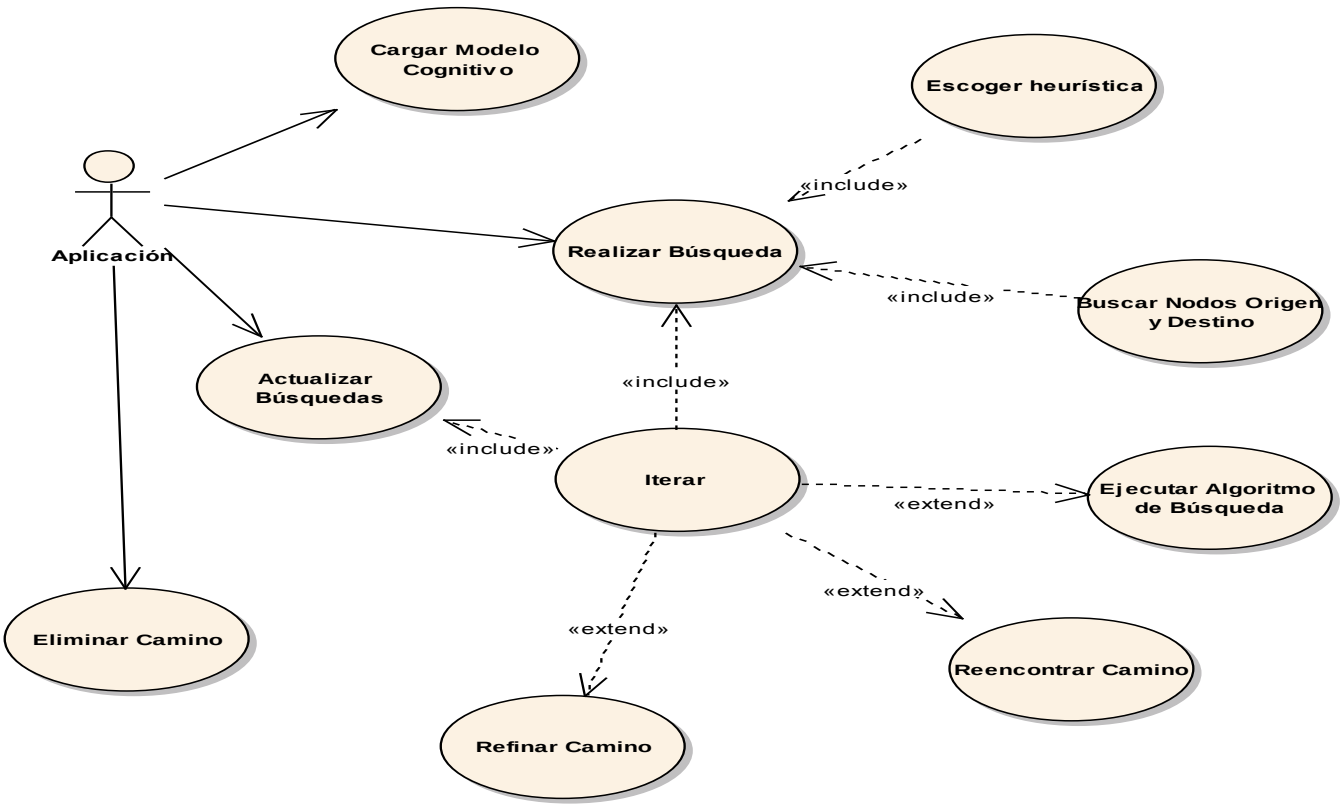

Figura 15: Diagrama de caso de uso

3.4.4 DESCRIPCIÓN DE LOS CASOS DE USO EN FORMATO EXPANDIDO.

Tabla 3: CU "Cargar modelo cognitivo".

Nombre del Caso de Uso	Cargar modelo cognitivo.
Actores	Aplicación
Propósito	Crear el grafo del entorno.
Resumen: Se inicia cuando el actor solicita cargar un grafo de navegación especificando la dirección del fichero. En caso de no encontrarse el fichero se devuelve el valor de falso. El CU finaliza cuando se ha leído del fichero la información y se han creado todos los nodos del grafo.	
Referencias	**R1**
Precondición	-
Pos condición	El grafo de navegación ha sido creado.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1 - Solicita cargar modelo cognitivo pasando la dirección del fichero.	1.1- Verifica la existencia del fichero. 1.2- Si no existe el fichero ir a curso alterno1. 1.3- Se verifica la estructura de los datos del fichero. 1.4- Se lee el tipo de modelo cognitivo, y se crea. 1.5- Por cada estructura de nodo encontrada, se lee y se crea el nodo. 1.6- Por cada estructura de conexión entre nodos se lee y se crea la conexión del grafo.

	1.7- Termina caso de uso.
Curso Alterno1:	
Acción del actor	**Respuesta del sistema**
	1.1 - Si el fichero no existe se informa al actor. 1.2 - Si los datos están mal estructurados se cierra el fichero y se le informa al actor.
Prioridad: Crítico.	

Tabla 4: CU "Realizar búsqueda".

Nombre del Caso de Uso	Realizar búsqueda.
Actores	Aplicación
Propósito	Crear petición de búsqueda.
Resumen: Se inicia cuando el actor solicita una búsqueda especificando los puntos de origen y de destino, y la heurística con la que se evaluarán los nodos en la búsqueda. El CU finaliza cuando se ha creado la instancia de la clase cPath, que será la que contendrá los datos del camino solicitado.	
Referencias	**R2,R3,R4, CU1,CU3,CU4**
Precondición	El CU1 haya ocurrido.
Pos condición	Se obtiene entidad cPath que contiene los datos del camino a obtener.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**

1 – Solicita camino dado la lista de orígenes, la lista de destino y la lista de heurística a usarse.	1.1- Se inicia CU3 1.2- Se crea instancia cPath 1.3- Se inicia CU4 1.4- Se inicia CU7 1.5- Se inserta la instancia de cPath en la clase controladora cPathFinding 1.6- Se retorna la instancia de cPath terminando el caso de uso
Prioridad: Secundario.	

Tabla 5: CU3 "Escoger heurística".

Nombre del Caso de Uso	Escoger heurística.
Actores	Aplicación
Propósito	Escoger los términos que evaluaran la búsqueda.
Resumen: Se inicia cuando el actor solicita una búsqueda debe escoger qué términos evaluarán cada nodo para saber si es más óptimo el camino transitado por esta hasta la solución. El CU finaliza cuando se ha creado la lista de heurísticas que se utilizarán en el CU2.	
Referencias	**R7,**
Precondición	-
Pos condición	Se obtiene lista de heurísticas para la búsqueda
Curso normal de los eventos:	

Acción del actor	**Respuesta del sistema**
1 – Selecciona heurística a usar en la búsqueda.	1.1- Chequea los tipos de heurísticas que están registradas. 1.2- Crea instancias de entre las heurísticas que están creadas las que desea usar en la búsqueda 1.3- Inserta las instancias en una lista 1.4- Devuelve la lista de heurística terminado el caso de uso.
Prioridad: Crítico.	

Tabla 6: CU4 “Buscar nodos origen y destino”.

Nombre del Caso de Uso	Buscar nodos origen y destino
Actores	Aplicación
Propósito	Escoger los términos que evaluarán la búsqueda.
Resumen: Se inicia cuando el actor solicita una búsqueda debe escoger que términos evaluaran cada nodo para saber si es más óptimo el camino transitado por esta hasta la solución. El CU finaliza cuando se ha creado la lista de heurísticas que se utilizarán en el CU2.	
Referencias	**CU1**
Precondición	El CU1 haya ocurrido.
Pos condición	Se obtienen los nodos origen y destino
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**

1 – Solicita nodo más cercano al punto origen.	1.1- El cPathFinding le dice al modelo cognitivo que busque el nodo más cercano al punto origen. 1.2- Si el modelo cognitivo es del tipo cCognitiveModel_Pov ir a curso alterno 1. 1.3- Si el modelo cognitivo es del tipo cCognitiveModel_NavMesh ir a curso alterno 2. 1.4- El modelo cognitivo devuelve este nodo al cPathFinding.
2 – Solicita nodo más cercano al punto origen.	2.1- El cPathFinding le dice al modelo cognitivo que busque el nodo más cercano al punto destino. 2.2- Si el modelo cognitivo es del tipo cCognitiveModel_Pov ir a curso alterno 3 2.3- Si el modelo cognitivo es del tipo cCognitiveModel_NavMesh ir a curso alterno 4 2.4- El modelo cognitivo devuelve este nodo al cPathFinding.
Curso Alterno1:	
Acción del actor	**Respuesta del sistema**
	1.1- El modelo cognitivo le pregunta a cada nodo cual es su distancia al punto origen. 1.2- El modelo cognitivo escoge de todos los nodos el que menor distancia al punto origen tenga.

	1.3- Volver al curso normal de los eventos 1.4
Curso Alterno2:	
Acción del actor	**Respuesta del sistema**
	1.1- El modelo cognitivo le pregunta a cada nodo si el punto origen se encuentra dentro. 1.2- Si se encuentra un nodo que cumpla con la condición anterior se escoge este nodo 1.3- Volver al curso normal de los eventos 1.4
Curso Alterno3:	
Acción del actor	**Respuesta del sistema**
	1.1- El modelo cognitivo le pregunta a cada nodo cual es su distancia al punto destino. 1.2- El modelo cognitivo escoge de todos los nodos el que menor distancia al punto destino tenga. 1.3- Volver al curso normal de los eventos 2.4
Curso Alterno4:	
Acción del actor	**Respuesta del sistema**
	1.1- El modelo cognitivo le pregunta a cada nodo si el punto destino se encuentra dentro. 1.2- Si se encuentra un nodo que cumpla con la

	condición anterior se escoge este nodo 1.3- Volver al curso normal de los eventos 2.4
Prioridad: Crítico.	

Tabla 7: CU5 "Actualizar búsqueda".

Nombre del Caso de Uso	Actualizar búsqueda.
Actores	Aplicación
Propósito	Actualizar las peticiones de búsqueda hechas por los agentes.
Resumen: Se debe iniciar con regularidad durante la aplicación para actualizar el estado de las peticiones de búsquedas hechas por los agentes. El CU finaliza cuando se libere la biblioteca.	
Referencias	**R4**
Precondición	-
Pos condición	Se actualiza el estado de búsqueda de las peticiones de caminos.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1- Actualizar búsquedas.	1.1- El cPathFinding calcula el número de iteraciones que puede realizar en cada petición de búsqueda entre el número de ciclos que puede realizar en total. 1.2- El cPathFinding le dice a cada petición que realice el CU6 el número de veces calculado en el paso anterior.
Curso Alterno:	

1.1 Si no existen peticiones de búsqueda en estos momentos no se hace nada.
Prioridad: Crítico.

Tabla 8: CU6 "Iterar".

Nombre del Caso de Uso		Actualizar búsqueda.
Actores	Aplicación	
Propósito	Iterar proceso de búsqueda.	
Resumen: Se inicia cuando se actualiza el cPathFinding este le dice a cada cPath que itere. Este caso de uso termina el estado del cPath es: Done que quiere decir que se tiene el camino óptimo.		
Referencias	**R4,R5,R7**	
Precondición	-	
Pos condición	Se actualiza el estado de búsqueda de esta petición.	
Curso normal de los eventos:		
Acción del actor		**Respuesta del sistema**
		1.1 Se chequea en qué estado esta la búsqueda. 1.2 Si el estado es buscando se realiza CU7. 1.3 Si el estado es reencontrar camino se realiza el CU8. 1.4 Si el estado es refinar camino se realiza el CU9. 1.5 Termina el caso de uso.
Curso Alterno:		

1.1 Si la búsqueda está en estado terminada no se hace nada.
Prioridad: Crítico.

Tabla 9: CU7 "Ejecutar algoritmo de búsqueda".

Nombre del Caso de Uso	Ejecutar algoritmo de búsqueda.
Actores	Aplicación
Propósito	Usar algoritmo A* para encontrar la solución.
Resumen: Se expanden los nodos del grafo hasta encontrar la solución.	
Referencias	**R7**
Precondición	-
Pos condición	Se actualiza el estado de búsqueda de esta petición.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
	1.1 Se coge el primer nodo de la cola con prioridad. 1.2 Se chequea si este nodo es la solución. 1.3 Si es la solución se pasa al estado de reencontrar el camino. 1.4 Si no es, se solicitan los hijos de este nodo. 1.5 Se insertan en la cola con prioridad estos nodos. 1.6 Se pone el nodo en la lista de nodos cerrados. 1.7 Termina caso de uso.

Prioridad: Crítico.

Tabla 10: CU8 "Reencontrar camino".

Nombre del Caso de Uso	Reencontrar camino.
Actores	Aplicación
Propósito	Encontrar el camino a la solución obtenida.
Resumen: Se realiza una búsqueda para encontrar el camino entre la posición actual y el camino óptimo obtenido.	
Referencias	**R3,R7**
precondición	El estado anterior era "Buscando"
Pos condición	Se encuentra camino a la solución
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
	1.1- Se obtiene la posición actual. 1.2- Se iguala la lista de nodos destinos a la lista de nodos del camino obtenido. 1.3- Se pone como nodo origen la posición actual. 1.4- Se itera el CU7 hasta encontrar la solución del nuevo camino solicitado. 1.5- Se termina el caso de uso.
Curso Alterno:	

Prioridad: Crítico.

Tabla 11: CU9 “Refinar camino”.

Nombre del Caso de Uso	Refinar camino.
Actores	Aplicación
Propósito	Quitar nodos innecesarios del camino.
Resumen: Se recorren los nodos del camino chequeando aquellos a los que se puede ir directamente desde cada nodo, quitando los nodos innecesarios.	
Referencias	R5**,R7**
Precondición	El estado anterior era “Reencontrando”
Pos condición	Se encuentra camino óptimo.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
	1.1- Se coge el primer nodo del camino llamándola A. 1.2- Se coge el tercer nodo del camino llamándolo B. 1.3- Si se puede ir del nodo A al nodo B sin colisionar, se elimina el padre del nodo B. 1.4- Se corre el nodo B al que le sigue a este mismo. 1.5- Se repite desde el paso 1.3 hasta llegar al final de la lista. 1.6- Si se llega al final de la lista se corre el nodo

	A al nodo que le sigue y el B al que le sigue a este ultimo. 1.7- Se repite desde el paso 1.3 hasta que el nodo A sea el penúltimo de la lista. 1.8- Termina caso de uso.
Curso Alterno:	
Prioridad: Crítico.	

Tabla 12: CU10 "Eliminar camino".

Nombre del Caso de Uso	Reencontrar camino.
Actores	Aplicación
Propósito	Eliminar una petición de camino.
Resumen: Se inicia cuando el actor solicita al cPathFinding que elimine el cPath de los caminos que tiene registrados.	
Referencias	**R6**
Precondición	Existe el camino
Pos condición	Se elimina el camino
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1- Eliminar camino pasándole el puntero al cPath creado.	1.1- Quita de la lista de caminos para actualizar. 1.2- Destruye el camino liberando los recursos.

	1.3- Termina caso de uso.
Curso Alterno:	
Prioridad: Crítico.	

CAPÍTULO 4: DISEÑO DEL SISTEMA

A continuación se presentarán los diagramas de clases del diseño el cual tendrá un paquete "cPathFinding" con las relaciones existente que pueden ser de agregación, generalización/especialización y composición. Además se presentan los diagramas de secuencia de los casos de usos que intervendrán en el primer ciclo de desarrollo de software.

4.1 DIAGRAMA DE PAQUETES DE CLASES DEL DISEÑO

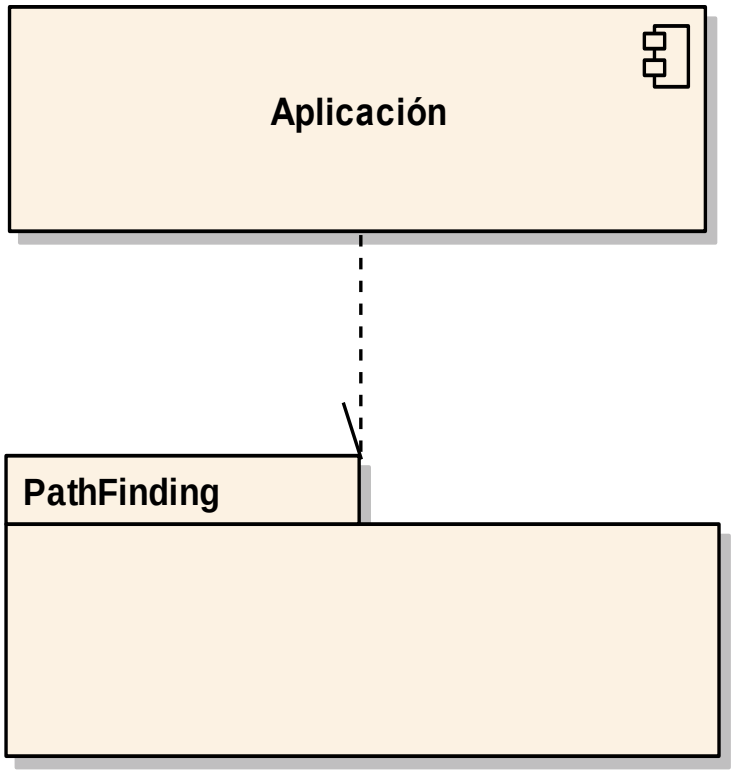

Figura 16: Diagrama de paquetes.

4.2 DIAGRAMA DE CLASES DEL PAQUETE “PATHFINDING”

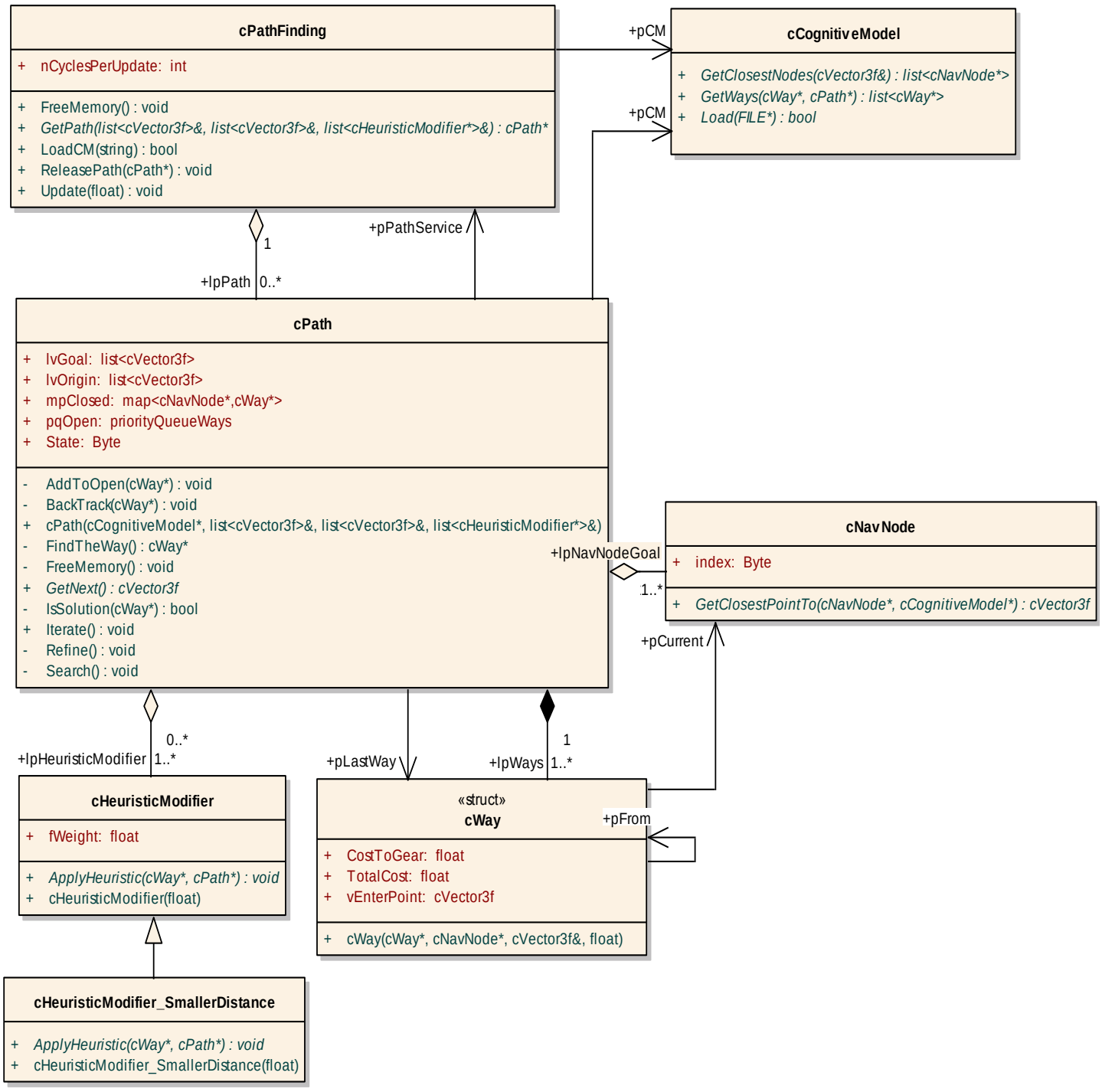

Figura 17: Diagrama de clases.

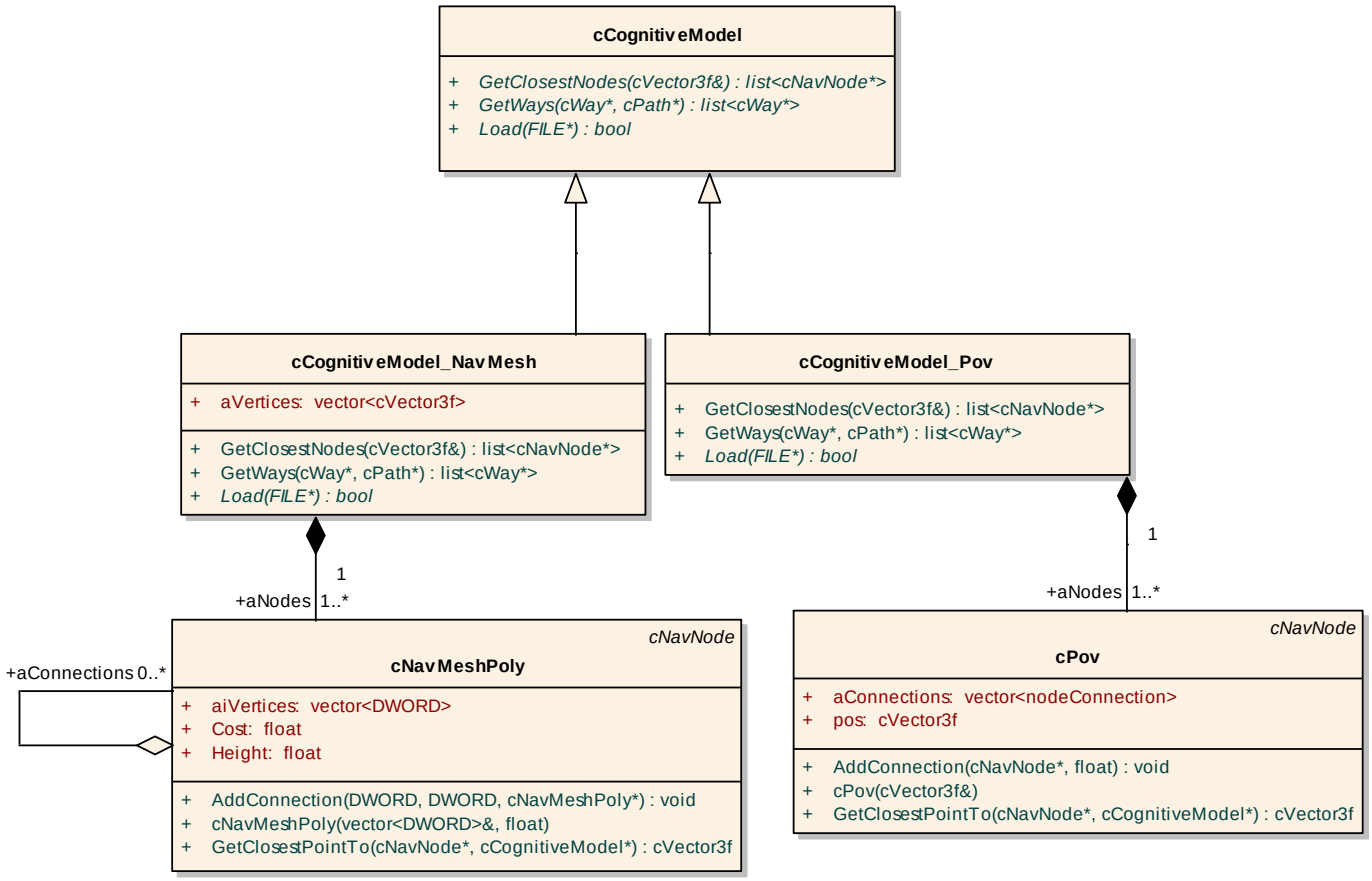

Figura 18: Diagrama de clases Herencia Modelos Cognitivos.

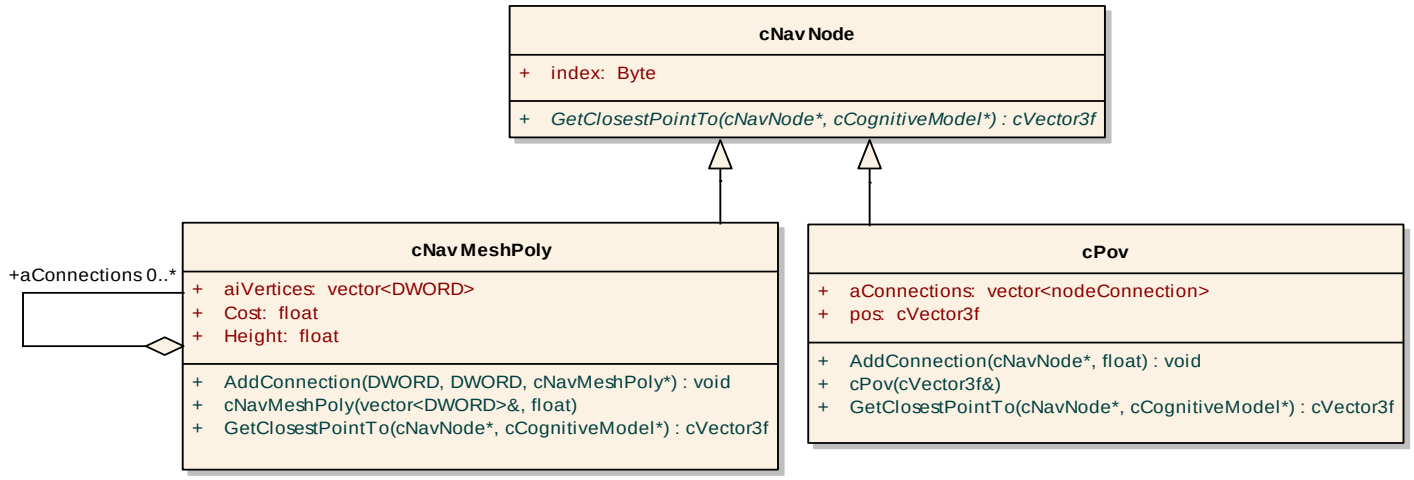

Figura 19: Diagrama de clases Herencia nodos del Modelo Cognitivo.

4.3 DIAGRAMAS DE INTERACCIÓN DE DISEÑO.

En este epígrafe se muestran las realizaciones de los caso de usos.

4.3.1 REALIZACIÓN DE CASO DE USO "CARGAR MODELO COGNITIVO"

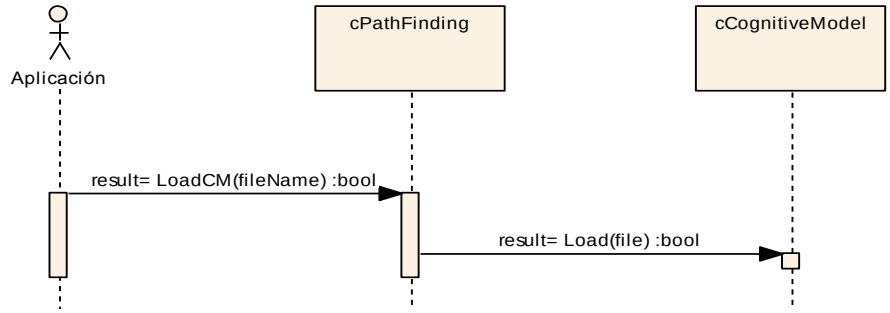

Figura 20: Realización de caso de uso: Cargar Modelo Cognitivo

4.3.2 REALIZACIÓN DE CASO DE USO "REALIZAR BÚSQUEDA"

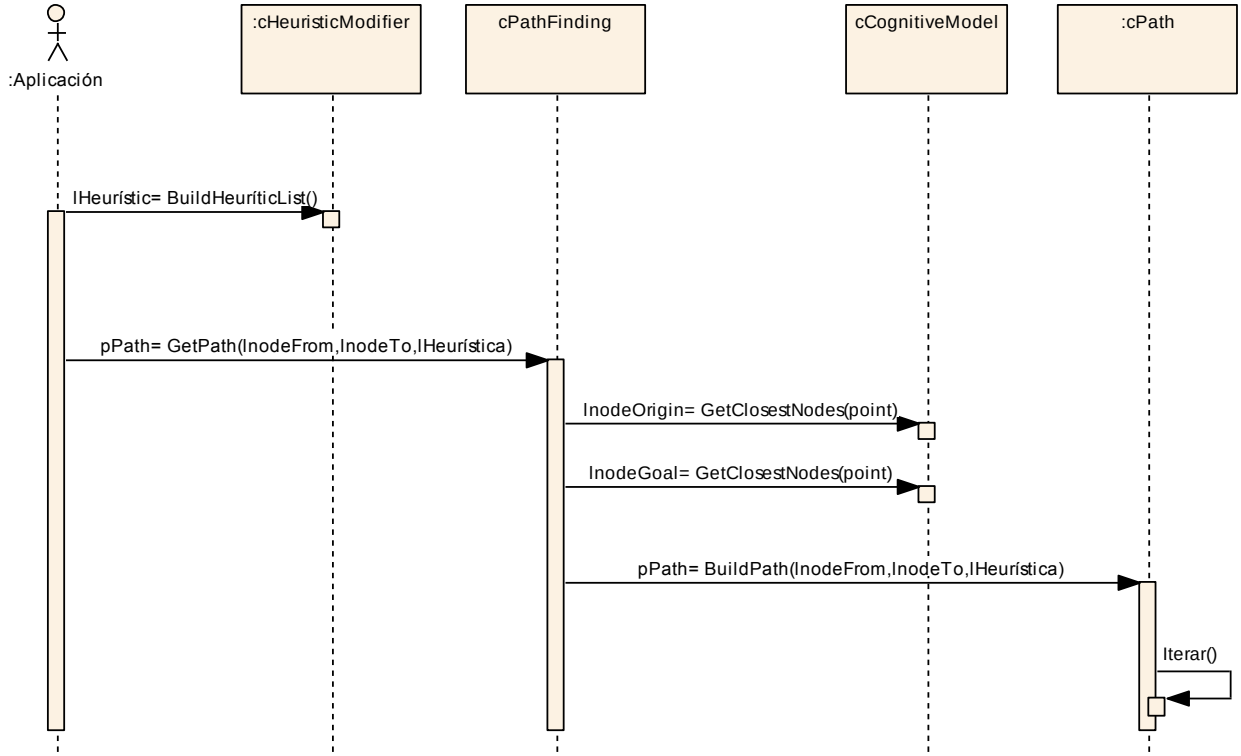

Figura 21: Realización de caso de uso: Realizar búsqueda.

4.3.3 REALIZACIÓN DE CASO DE USO “ESCOGER HEURÍSTICA”

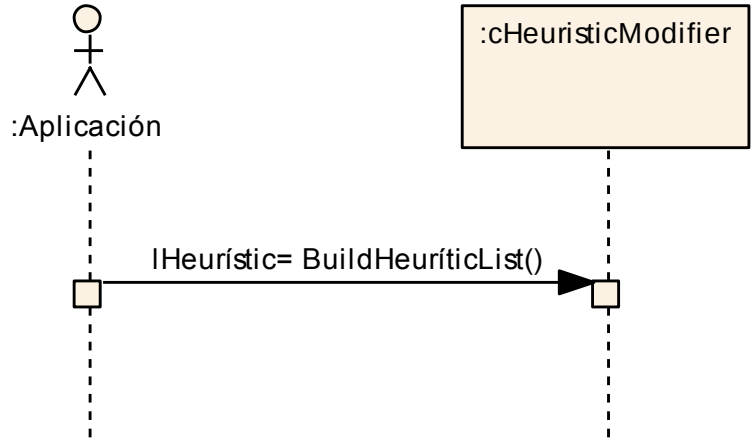

Figura 22: Realización de caso de uso: Escoger Heurística.

4.3.4 REALIZACIÓN DE CASO DE USO “BUSCAR NODOS ORIGEN Y DESTINO”

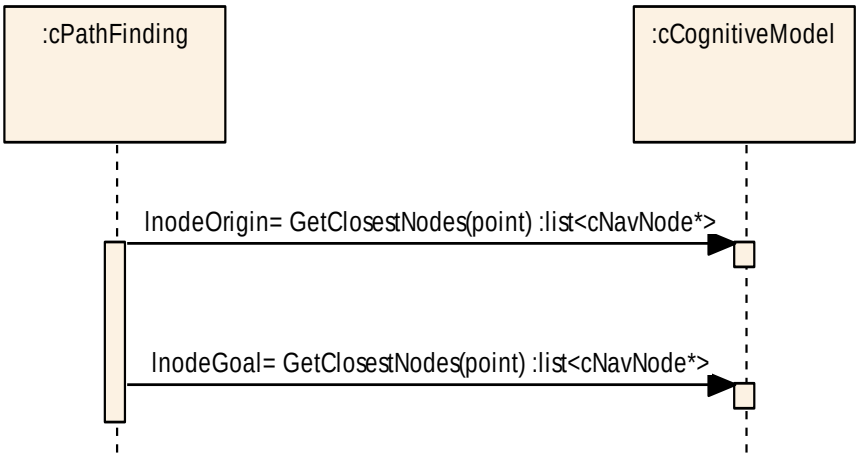

Figura 23: Realización de caso de uso: Buscar nodos origen y destino.

4.3.5 REALIZACIÓN DE CASO DE USO “ACTUALIZAR BÚSQUEDA”

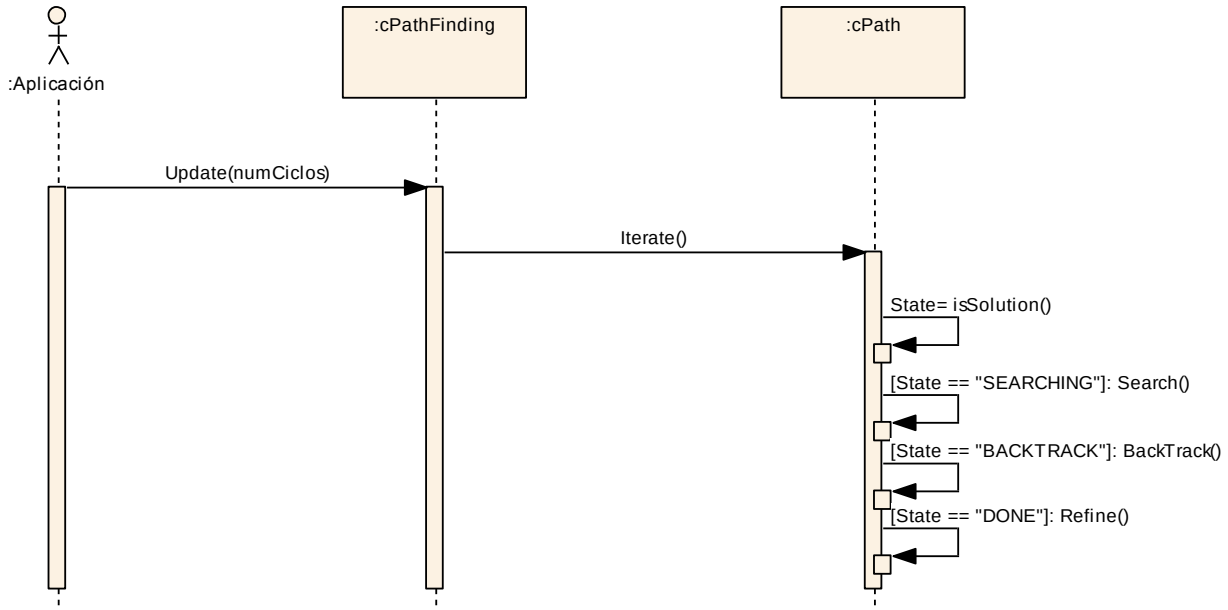

Figura 24: Realización de caso de uso: Actualizar búsqueda.

4.3.6 REALIZACIÓN DE CASO DE USO “ITERAR”

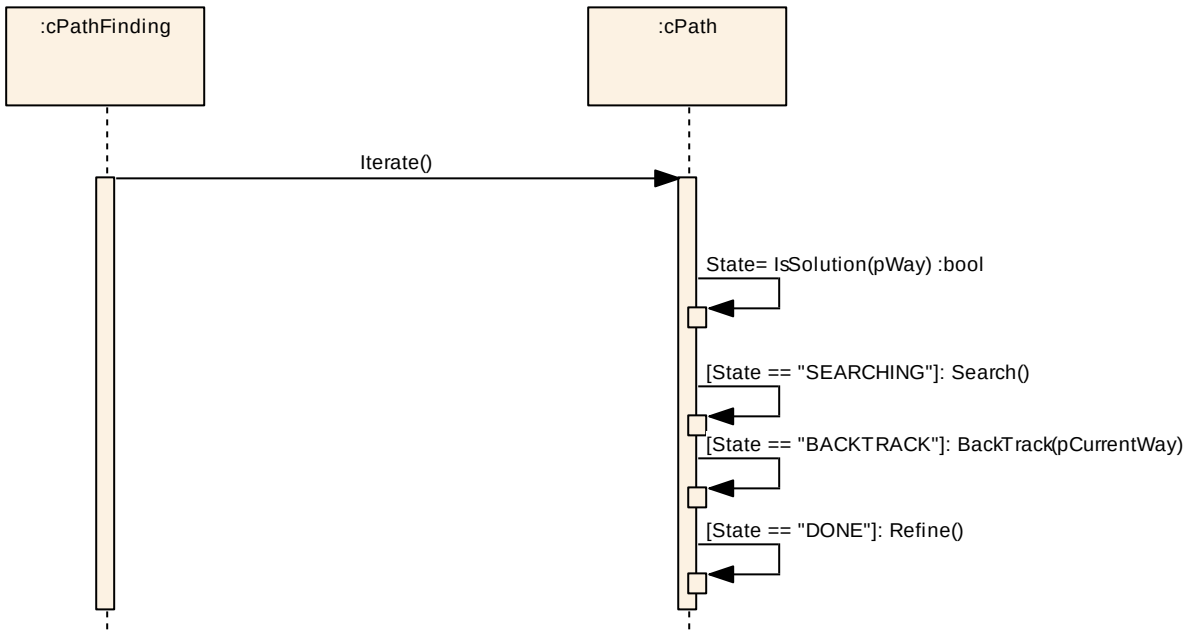

Figura 25: Realización de caso de uso: Iterar.

4.3.7 REALIZACIÓN DE CASO DE USO “EJECUTAR ALGORITMO DE BÚSQUEDA”

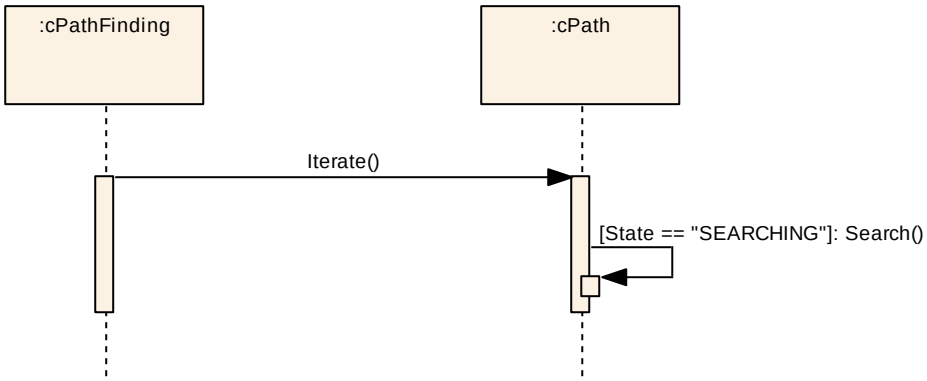

Figura 26: Realización de caso de uso: Ejecutar algoritmo de búsqueda.

4.3.8 REALIZACIÓN DE CASO DE USO “REENCONTRAR CAMINO”

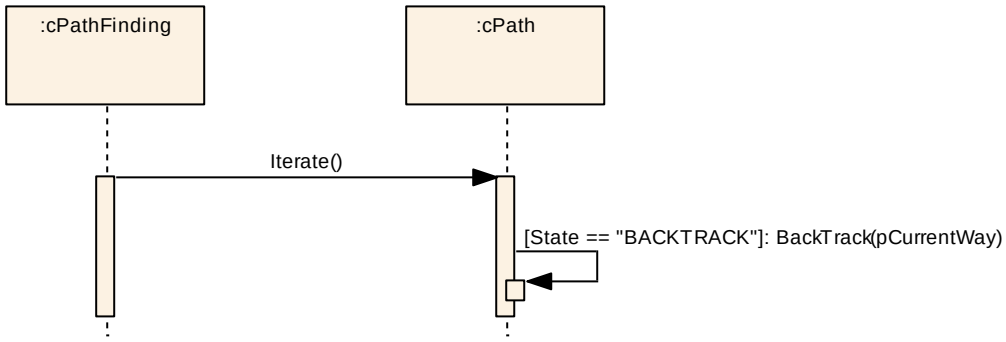

Figura 27: Realización de caso de uso: Reencontrar camino.

4.3.9 REALIZACIÓN DE CASO DE USO “REFINAR CAMINO”

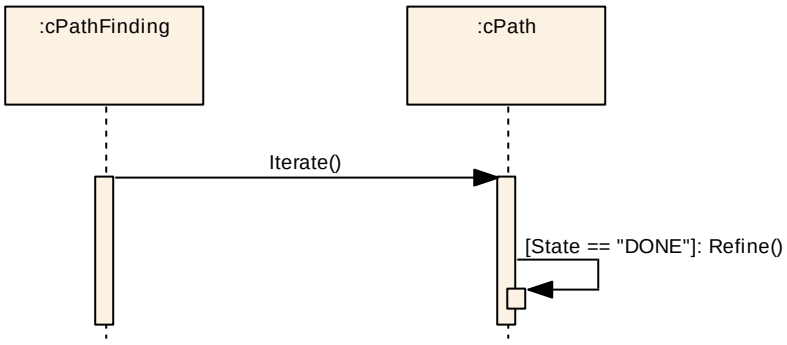

Figura 28: Realización de caso de uso: Refinar camino.

4.3.10 REALIZACIÓN DE CASO DE USO “ELIMINAR CAMINO”

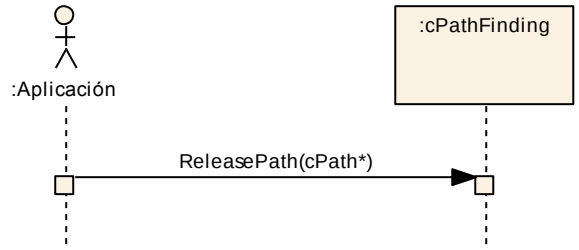

Figura 29: Realización de caso de uso: Eliminar camino.

4.4 DESCRIPCIÓN DE LAS CLASES

Tabla 13: Clase “cPathFinding”.

Nombre: cPathFinding	
Tipo de clase (interfaz)	
Descripción: Clase interfaz de la biblioteca, brinda la funcionalidad de la misma y controla las peticiones de búsqueda y los modelos cognitivos.	
Atributo	**Tipo**
lpPath	list<cPath*>
nCyclesPerUpdate	int
pCM	cCognitiveModel*
Para cada responsabilidad	
Nombre:	**Descripción:**
`cPathFinding()`	Constructor de la clase.
`GetPath(list<cVector3f>&, list<cVector3f>&,`	Devuelve el puntero al camino que se ha solicitado pasándole la lista de puntos origen, la lista destino y la

`list<cHeuristicModifier*>&):cPath*`	lista de heurística a usar.
`LoadC``M(string):bool`	Carga el modelo del nombre de la dirección especificada.
`Update(float)`	Actualiza el estado de las búsquedas de caminos.

Tabla 14: Clase "cCognitiveModel".

Nombre: cCognitiveModel	
Tipo de clase (interfaz)	
Descripción: Clase interfaz para los tipos de grafos de caminos.	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre:	**Descripción:**
`Load(FILE*):bool`	Lee de un fichero el grafo de camino.
`GetClosestNodes(cVector3f&):list<cNavNode*>`	Dado un punto devuelve el nodo del grafo más cercano a este.
`GetWays(cWay*, cPath*):list<cWay*>`	Devuelve todos los caminos del nodo padre a sus hijos.

Tabla 15: Clase "cCognitiveModel_Pov".

Nombre: cCognitiveModel_Pov
Tipo de clase (controladora)
Descripción: Clase controladora del tipo de dato cPov, representa modelo cognitivo puntos de

visibilidad.	
Atributo	**Tipo**
aNavNodes	vector<cNavNode*>
Para cada responsabilidad	
Nombre:	**Descripción:**
`cCognitiveModel_Pov()`	Constructor de la clase.
`Load(FILE*):bool`	Lee de un fichero el grafo de camino.
`GetClosestNodes(cVector3f&): list<cNavNode*>`	Dado un punto devuelve el nodo del grafo más cercano a este.
`GetWays(cWay*, cPath*):list<cWay*>`	Devuelve todos los caminos del nodo padre a sus hijos.

Tabla 16: Clase "cCognitiveModel_NavMesh".

Nombre: cCognitiveModel_NavMesh	
Tipo de clase (controladora)	
Descripción: Clase controladora del tipo de dato cNavMeshPoly, representa el modelo cognitivo Mallas de navegación.	
Atributo	**Tipo**
aNodes	vector<cNavMeshPoly>
aVertices	vector<cVector3f>
Para cada responsabilidad	

Nombre:	**Descripción:**
`cCognitiveModel_NavMesh ()`	Constructor de la clase.
`Load(FILE*):bool`	Lee de un fichero el grafo de camino.
`GetClosestNodes(cVector3f&): list<cNavNode*>`	Devuelve el nodo del grafo que representa el área donde se encuentra un punto dado.
`GetWays(cWay*, cPath*):list<cWay*>`	Devuelve todos los caminos del nodo padre a sus hijos.

Tabla 17: Clase "cNavNode".

Nombre: cNavNode	
Tipo de clase (interfaz)	
Descripción: Clase interfaz para los nodos del grafo de camino.	
Atributo	**Tipo**
index	Unsigned int
Para cada responsabilidad	
Nombre:	**Descripción:**
`GetClosestPointTo(cNavNode*, cCognitiveModel*): cVector3f`	Devuelve el punto más cercano del nodo actual a un nodo dado.

Tabla 18: Clase "cPov".

Nombre: cPov

Tipo de clase (Entidad)	
Descripción: Clase entidad que contiene los datos de un punto de visibilidad.	
Atributo	**Tipo**
aConnections	vector<nodeConnection>
pos	cVector3f
Para cada responsabilidad	
Nombre:	**Descripción:**
`cPov ()`	Constructor de la clase.
`AddConnection(cNavNode*,float)`	Conecta el nodo con otro nodo.
`GetClosestPointTo(cNavNode*,cCognitiveModel*) : cVector3f`	Devuelve el punto más cercano del nodo actual a un nodo dado.

Tabla 19: Clase "cNavMeshPoly".

Nombre: cNavMeshPoly	
Tipo de clase (Entidad)	
Descripción: Clase entidad que contiene los datos de una malla de navegación.	
Atributo	**Tipo**
aConnections	vector<cNavMeshPoly*>
aiVertices	vector<DWORD>

Cost	float
Height	float
Para cada responsabilidad	
Nombre:	**Descripción:**
`cNavMeshPoly()`	Constructor de la clase.
`AddConnection(DWORD,DWORD,cNavMeshPoly*)`	Conecta el nodo con otro nodo.
`GetClosestPointTo(cNavNode*,cCognitiveModel*) : cVector3f`	Devuelve el punto más cercano del nodo actual a un nodo dado.

Tabla 20: Clase "cHeuristicModifier".

Nombre: cHeuristicModifier	
Tipo de clase (interfaz)	
Descripción: Clase interfaz para los tipos de heurísticas.	
Atributo	**Tipo**
fWeight	float
Para cada responsabilidad	
Nombre:	**Descripción:**
cHeuristicModifier `(float)`	Constructor de la clase.
`ApplyHeuristic(cWay*,cPath*)`	Calcula la heurística para un nodo dado.

Tabla 21: Clase "cHeuristicModifier_SmallerDistance".

Nombre: cHeuristicModifier_SmallerDistance	
Tipo de clase (Entidad)	
Descripción: Clase entidad calcula la heurística de la distancia entre dos puntos.	
Atributo	**Tipo**
fWeight	float
Para cada responsabilidad	
Nombre:	**Descripción:**
`cHeuristicModifier_SmallerDistance (float)`	Constructor de la clase.
`ApplyHeuristic(cWay*,cPath*)`	Calcula la heurística para un nodo dado.

Tabla 22: Clase "cPath".

Nombre: cPath	
Tipo de clase (Entidad)	
Descripción: Clase entidad representa el camino obtenido de una petición de búsqueda.	
Atributo	**Tipo**
lpHeuristicModifier	list<cHeuristicModifier*>
lpNavNodeGoal	list<cNavNode*>
lpWays	list<cWay*>
lvGoal	list<cVector3f>

lvOrigin	list<cVector3f>
mpClosed	map<cNavNode*,cWay*>
pCM	cCognitiveModel*
pLastWay	cWay*
pPathService	cPathFinding*
pqOpen	priorityQueueWays
State	Byte
Para cada responsabilidad	
Nombre:	**Descripción:**
cPath `(cCognitiveModel*, list<cVector3f>&, list<cVector3f>&, list<cHeuristicModifier*>&)`	Constructor de la clase.
`AddToOpen(cWay*)`	Adiciona los hijos de un nodo a la cola de abiertos.
`Iterate()`	Ocurre una iteración del proceso de búsqueda.
`Search()`	Ocurre una iteración del proceso de búsqueda.
`IsSolution(cWay*)`	Chequea si el nodo actual es la solución.
`BackTrack(cWay*)`	Se genera la lista de nodos que conforman el camino y se encuentra un nuevo camino desde el punto actual del agente hasta

	los nodos de la lista generada.
`Refine()`	Se refina el camino obtenido quitando nodos innecesarios.
`GetNext(): cVector3f`	Se obtiene el punto del camino al que debemos movernos.

Tabla 23: Clase "cWay".

Nombre: cWay	
Tipo de clase (entidad)	
Descripción: Clase entidad representa un camino expandido.	
Atributo	**Tipo**
CostToGear	float
TotalCost	float
pCurrent	cNavNode*
pFrom	cWay*
vEnterPoint	cVector3f
Para cada responsabilidad	
Nombre:	**Descripción:**
`cWay(cWay*,cNavNode*,cVector3f&,float)`	Constructor de la clase.

CAPÍTULO 5: RESULTADOS

En este capítulo se muestra el resultado de algunos casos de pruebas hechos a la biblioteca y pruebas de rendimiento realizados a la aplicación que se hizo para mostrar la integración de la biblioteca a cualquier sistema.

5.1 CASOS DE PRUEBA A LOS CASOS DE USO DEL SISTEMA.

Tabla 24: Caso de prueba "Intentar cargar MC poniendo la dirección del fichero incorrecta".

Caso de uso	Cargar modelo cognitivo
Caso de prueba	Intentar cargar modelo cognitivo poniendo la dirección del fichero incorrecta
Entrada	Se intenta leer el fichero especificado
Resultado	Se retorna valor falso y no se crea modelo cognitivo.
Condiciones	

Tabla 25: Caso de prueba "Encontrar nodo en el que se encuentra un punto".

Caso de uso	Buscar nodos origen y destino.
Caso de prueba	Encontrar nodo en el que se encuentra un punto.
Entrada	Buscar nodo más cercano a un punto
Resultado	Se retorna el valor NULL
Condiciones	Modelo cognitivo cargado

Tabla 26: Caso de prueba "Cola de prioridad de nodos abiertos vacía".

Caso de uso	Ejecutar algoritmo de búsqueda.
Caso de prueba	Cola de prioridad de nodos abiertos vacía.
Entrada	Se manda a ejecutar el algoritmo de búsqueda.
Resultado	Se cambia el estado del camino a: “Invalid”
Condiciones	

Tabla 27: Caso de prueba "Intentar cargar MC con formato incorrecto".

Caso de uso	Cargar modelo cognitivo
Caso de prueba	Intentar cargar modelo cognitivo con formato incorrecto.
Entrada	Se intenta leer el fichero del modelo cognitivo
Resultado	Se retorna el valor: false y no se crea el modelo cognitivo.
Condiciones	

5.2 PRUEBAS DE RENDIMIENTO

Para demostrar el proceso de integración de la biblioteca de Búsqueda de caminos a cualquier sistema se creó un Demo usando el SDK de Irlich para pintar y manejar los objetos, se diseñó un nivel basado en áreas unidas por corredores, del cual se exportó el modelo cognitivo de tipo mallas de navegación. El MC tiene 184 nodos. En este epígrafe mostraremos algunos resultados de rendimiento obtenidos de la búsqueda de caminos en esa aplicación.

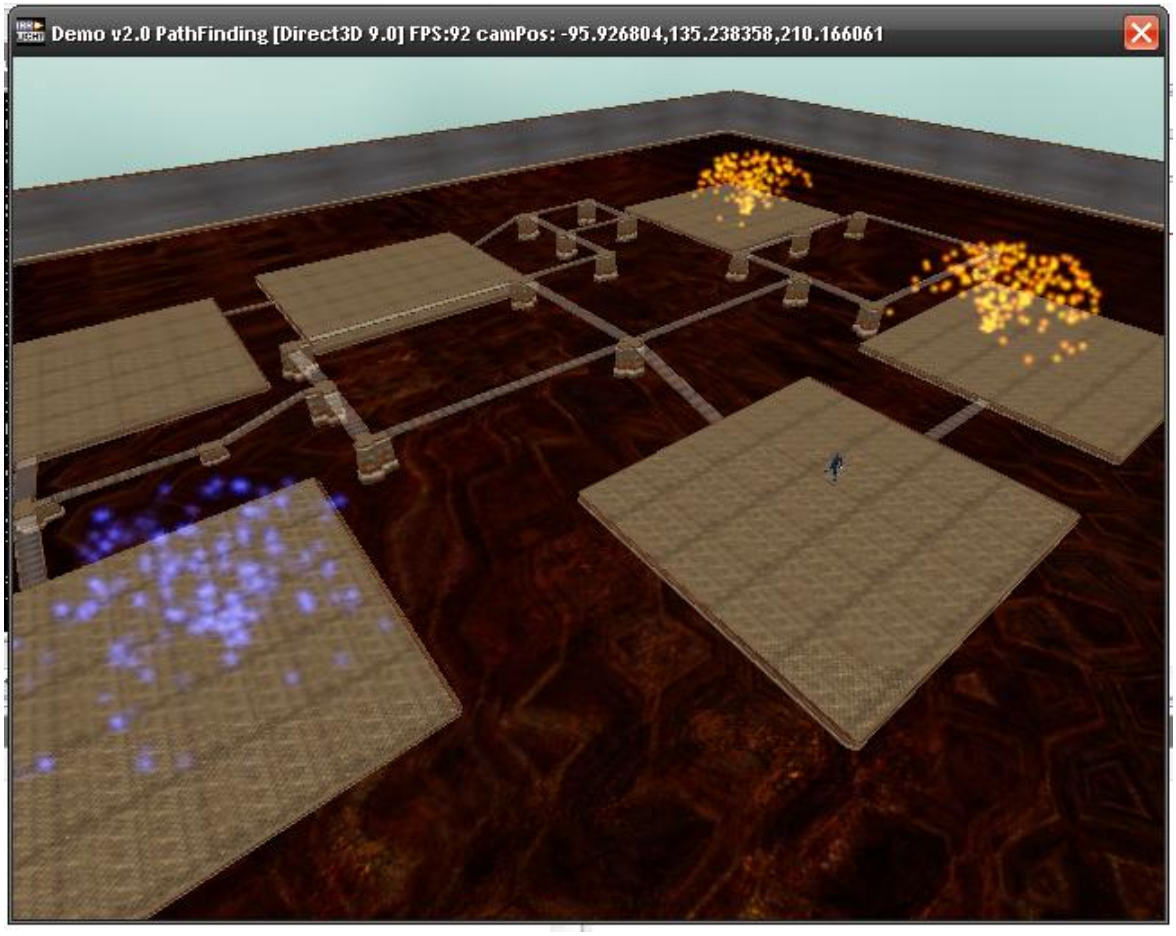

Figura 30: Imagen del Demo

Como el modelo cognitivo “Mallas de navegación” consiste en que cada cuarto o corredor sea un nodo del grafo, al hacer una petición de camino por parte de un agente a un punto dentro del mismo cuarto la respuesta del camino es instantánea, el camino obtenido es del punto en que se encuentra al agente inteligente al punto de destino pedido.

Tabla 28: Prueba rendimiento "Petición de camino a un punto dentro del mismo cuarto".

Variable	Resultado
Nodos expandidos	1
Iteraciones	1
Tiempo respuesta	1 ms
Número de nodos del camino sin refinar	1
Número de nodos del camino refinado	1

El agente inteligente puede hacer una petición de camino a un determinado ítem, estas son los resultados de un ejemplo.

Tabla 29: Prueba de rendimiento "Petición de camino a un 'ítem".

Variable	Resultado
Nodos expandidos	63
Iteraciones	55
Tiempo respuesta	15 ms
Número de nodos del camino sin refinar	22
Número de nodos del camino refinado	8

La biblioteca también es capaz de recibir una petición de un camino hacia varios destinos posibles, calculando y obteniendo el camino al destino que sea más óptimo, a continuación los resultados obtenidos de un tipo de esta petición.

Tabla 30: Prueba de rendimiento "Petición de camino a varios destinos".

Variable	Resultado
Nodos expandidos	7
Iteraciones	4
Tiempo respuesta	10 ms
Número de nodos del camino sin refinar	5
Número de nodos del camino refinado	3

CONCLUSIONES

Para dar solución al objetivo principal de este trabajo se necesitó realizar un estudio a profundidad de las técnicas de búsqueda de caminos, y la serie de conceptos que la conforman como son los modelos cognitivos, los algoritmos de búsqueda, las heurísticas asociadas a estos algoritmos, y las técnicas de optimización más usadas por su eficiencia. Por lo que se realizó una gran recopilación de información de estos temas y se diseñó la biblioteca de forma que fuera extensible y óptima para el incremento de sus funcionalidades en el futuro.

Al terminar este estudio, se pasó a implementar la biblioteca de acuerdo al diseño previo, creando una aplicación que muestra el funcionamiento de la misma. Se realizaron pruebas de peticiones de caminos por parte de los agentes del entorno virtual en diferentes escenarios demostrando la eficiencia de la biblioteca en la tarea de una de las principales ramas de la inteligencia artificial la búsqueda de caminos y el uso de las técnicas de optimización refinado del camino, búsquedas parciales y distribución de tiempo.

RECOMENDACIONES

Este trabajo se realizó con el objetivo de crear una biblioteca que facilite todo lo relacionado con el tema de búsqueda de caminos por lo que se recomienda en el futuro:

- Continuar el desarrollo de la biblioteca agregándole otros evaluadores de la heurística como pueden ser evaluar un punto en el terreno para realizar emboscadas, o para ocultarse.
- Incluirle otros modelos cognitivos de diferente escenario como por ejemplo puede ser el modelo de un tablero de ajedrez.
- Incluirle más datos a los modelos cognitivos como puede ser el tipo de animación que debe seguir el agente inteligente al recorrer determinada arista.
- Agregarle herramienta al software de diseño 3D Blender para que pueda crear y exportar modelos cognitivos.

BIBLIOGRAFÍA

1. **Pinter, Marco.** *Toward More Realistic Pathfinding.*

2. **Buckland, Mat.** *Programming Game AI by Example.* Texas : Wordware Publishing, 2005. ISBN 1-55622-078-2.

3. **Higgins, Daniel.** How to achieve Lightning-Fast A*. *AI Game Programming Wisdom .* 2002.

4. —. Pathfinding Design Architecture. *AI Game Programming Wisdom.*

5. **Rabin, Steve.** *A* Speed Optimizations.* 2002.

6. **Stroustrup, Bjarne.** *The C++ Programming Language.* s.l. : Addison-Wesley, 1997.

7. **Matthews, James.** *Basic A* Pathfinding Made Simple.* s.l. : Charles River Media, 2000.

8. **Stout, Bryan.** *The Basics of A* for Path Panning.* s.l. : Charles River Media, 2000.

9. **Wesson, Richard.** *Path Finding Via Ant Races (a flood-fill algorithm).* 2001.

10. **Aho, Alfred V, Hopcroft, John E y Ullman, Jeffrey, D.** *Data Stuctures and Algorithms.* s.l. : Addison-Wesley, 1983.

11. **Zambrano Rodriguez, Douglas Francisco.** Monografias.com. [En línea] [Citado el: 10 de 02 de 2009.] http://www.monografias.com/trabajos10/intelart/intelart.shtml.

12. PathEngine. [En línea] 2008. [Citado el: 10 de 02 de 2009.] www.pathengine.com/overview.

13. **Golden, Bruce L.** Shortest Path Algorithms: A Comparison. [En línea] http://dspace.mit.edu/handle/1721.1/5263.

14. **leGrey, Lucien.** Algoritmo de Dijkstra. [En línea] 17 de 10 de 2008. [Citado el: 22 de 01 de 2009.] http://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra.

15. **Serruya, Juan Manuel.** Articulo: Introducción al Pathfinding, Teoría e Implementación del Algoritmo A* y Algoritmo HPA*. [En línea] 07 de 2007. [Citado el: 22 de 02 de 2009.] http://foros.3dgames.com.ar/programacion.97/509364.articulo-introduccion-al-pathfinding-algoritmos-y-hpa.html.

16. **Stout, W. Bryan.** The Star of the Search Algorithms (A* Search). [En línea] 12 de 02 de 1999. [Citado el: 22 de 02 de 2009.] http://www.gamasutra.com/features/19990212/sm_04.htm.

17. **Lester, Patrick.** A* Pathfinding for Beginners. [En línea] 18 de 07 de 2005. [Citado el: 22 de 02 de 2009.] http://www.gamedev.net/reference/programming/features/astar/.

18. **Iparraguirre, Andres Miguelevich y Palomino Ruiz, Miguel Castro.** *Herramientas de Edición y Análisis Topográfico de Estructuras de Navegación.* La Habana : Universidad de las Ciencias Informáticas, 2007.

GLOSARIO DE ABREVIATURAS:

3D: Tercera Dimensión.

CPU: Unidad Central de Procesamiento.

CU: Caso de Uso.

IA: Inteligencia Artificial

MC: Modelo Cognitivo

RUP: Rational Unified Process, (Proceso Unificado de Desarrollo).

RV: Realidad Virtual.

RTS: Real Time Strategy (Estrategia en Tiempo Real)

UCI: Universidad de la Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado.

GLOSARIO DE TÉRMINOS

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

B:

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

Blender: programa multiplataforma, dedicado especialmente al modelado y creación de gráficos tridimensionales.

D:

Demo: Aplicación para mostrar una o varias funcionalidades del sistema.

E:

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.

G:

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

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

I:

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

M:

Mesh: Objeto tridimensional compuesto por un conjunto de triángulos los cuales están compuestos por vértices y aristas.

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

N:

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

P:

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.

Path: camino, curso, ruta, senda, sendero, trayecto, vereda; (informática: trayecto de búsqueda).

S:

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

W:

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

ÍNDICE DE FIGURAS Y TABLAS