Universidad de las Ciencias Informáticas

Facultad 5

Título: Módulo de percepción visual para el Sistema de Percepción.

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

Autor: Arian Santiesteban Rodríguez.

Tutora: Ing. Yenifer del Valle Guevara.

Co-tutor: Ing. Alexey Broche Medina.

Ciudad de la Habana, Junio de 2010

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Arian Santiesteban Rodríguez

_____

Firma del Autor

_____ _____

Tutora: Ing. Yenifer del Valle Guevara **Co-Tutor:** Ing. Alexey Broche Medina

DATOS DE CONTACTO

Generales de la Tutora:

Nombre y apellidos: Ing. Yenifer del Valle Guevara

Especialidad: Ingeniería en Ciencias Informáticas

Años de experiencias: 4 años

Correo electrónico: ydelvalle@uci.cu

Teléfono de contacto: 837 2261

Generales del Co-Tutor:

Nombre y apellidos: Ing. Alexey Broche Medina

Especialidad: Ingeniería en Ciencias Informáticas

Años de experiencias: 2 años

Correo electrónico: abroche@uci.cu

Teléfono de contacto: 837 2261

AGRADECIMIENTOS

Agradecer fundamentalmente a todas las personas que hicieron posible la realización de este trabajo, a mis padres por siempre estar presentes cuando los necesite y siempre confiaron en mí, agradecerles además el haberme aguantado todos estos años. A mis hermanos Mary, Julio y el Chini que también me alentaron a seguir estudiando y a superarme todos los días, ellos saben que los quiero con la vida, en general agradecer a toda mi familia que es lo más grande que tengo.

Agradecer especialmente a mis tutores Yenifer y Alexey que de verdad que son los mejores, ellos siempre estuvieron ahí cuando hubo una duda o cuando algo no pinchaba en la tesis y sobre todo con este documento, ellos saben que pueden contar conmigo para lo que les haga falta. Un abrazo.

Agradecer al piquete del proyecto que si no los menciono aquí posiblemente me hagan preguntas en la exposición jajajaja, ellos han sido unos amigos excepcionales y bueno, nos hemos ayudado en todo lo que hemos podido para terminar en tiempo los trabajos. Ellos son El Ray y Saily, estos son como otros hermanitos más para mí.

Agradecer a todas las amistades de estos 5 años de la universidad que mencionarlos a todos ahora sería casi imposible, especialmente a todos aquellos que siempre me ayudaron y estuvieron presentes a lo largo de las dificultades que se presentaron en el camino. A los socios que desde los primeros días en el 75 nos estuvimos ayudando mutuamente para poder acostumbrarnos a la distancia de los hogares, al rigor de los estudios y de paso hacernos más fácil la vida aquí.

A los socios del cuarto, que más que socios son como hermanos, Capote, Alfonsito, el Hi, el DvD, bueno todos los de ese piquete. A las amistades de los primeros años que aunque nos separamos de grupos y hasta de apartamento siempre han sido mis hermanos también.

Por último y no menos importante agradecer a todos los profesores que a lo largo de mi vida como estudiante me han transmitido sus conocimientos para hacer posible que me encuentre hoy aquí, para todos ellos mi más sincero agradecimiento.

DEDICATORIA

Sobre todo este trabajo está dedicado a mis viejos que lo han dado todo en esta vida para que yo llegue hasta donde me encuentro, sin contar que me dieron la vida misma. Dedicado también a mis hermanos que siempre me han apoyado y dado ánimos para seguir adelante. Bueno, en general a todos los miembros de mi familia, que los quiero a todos.

RESUMEN

En la actualidad informática las herramientas desarrolladas sobre la base de la Inteligencia Artificial, compuestos por entornos virtuales y agentes inteligentes, son cada vez más usadas en ámbitos como el educativo, el diseño industrial, el entrenamiento militar y los paseos virtuales entre otros. Hoy día, un usuario puede interactuar con agentes inteligentes equipados con una serie de comportamientos y características similares a las que poseen los seres humanos, lo cual proporciona una elevada sensación de realismo.

Este trabajo propone desarrollar un módulo de percepción visual a partir del estudio realizado sobre las diferentes técnicas de selección de visibilidad, proponiendo niveles de complejidad mediante algoritmos híbridos basados en dichas técnicas.

En el documento se exponen los artefactos ingenieriles obtenidos al aplicar la metodología de desarrollo de *software* seleccionada; así como el diseño e implementación de un conjunto de clases que permiten el funcionamiento de los algoritmos de percepción visual. También se exponen las herramientas y lenguajes utilizados para la elaboración de la solución.

Como resultado de la investigación y el estudio de la documentación adquirida, se obtuvo un módulo que brinda a los programadores la posibilidad de escoger el nivel de precepción visual que deseen insertar a sus agentes. Cada uno de estos niveles proporciona a los agentes inteligentes la información de los elementos presentes en la escena virtual, que están dentro de su campo visual, dependiendo de la selección realizada.

Los resultados obtenidos luego del proceso de pruebas, demuestran que el módulo se encuentra completamente funcional y que no presenta problemas en la implementación de los algoritmos, cumpliendo de esta forma con el objetivo trazado. El módulo se encuentra integrado al Sistema de Percepción que se está desarrollando en el Centro de Desarrollo de Informática Industrial permitiendo a los agentes inteligentes la posibilidad de percibir visualmente su entorno. También evita la pérdida de tiempo por re-implementación de los algoritmos en el desarrollo de futuros productos.

Palabras clave: agentes, algoritmos, campo visual, entorno virtual, inteligencia artificial, percepción visual.

TABLA DE CONTENIDO

TABLA DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

Introducción

La combinación de entornos virtuales y agentes inteligentes se está convirtiendo en una herramienta cada vez más usada en ámbitos como el educativo, Los video juegos, el entrenamiento militar y los paseos virtuales. Hoy día, un usuario puede participar interactivamente con todos los elementos de un escenario virtual, teniendo la posibilidad de relacionarse con agentes inteligentes equipados con una serie de comportamientos y características similares a las que poseen los seres humanos, permitiendo crear aplicaciones con una elevada sensación de realismo.

Debido al elevado progreso que experimentan los productos de realidad virtual, el equipo de trabajo de la Línea de Navegación y Comportamiento Inteligente del Centro de Desarrollo de Informática Industrial (CEDIN), se encuentra inmerso en el desarrollo de un Sistema de Percepción (SP). Este sistema permitirá equipar a los agentes inteligentes con una percepción aceptable, permitiéndoles percibir en tiempo real el entorno que les rodea.

El CEDIN no cuenta con un sistema que le permita gestionar la percepción de los agentes inteligentes que conforman los productos de Inteligencia Artificial que se desarrollan, y por ende no cuenta con técnicas o algoritmos que le posibilite a estos agentes percibir visualmente su entorno. Es decir, se mueven a “ciegas” por la escena virtual, lo cual le resta realismo a su proceder debido a que su comportamiento viene dado por una serie de técnicas de toma de decisiones que no tienen en cuenta lo que realmente “ven” estos agentes. Además, para los programadores de video juegos del centro se hace necesario reutilizar código en el mayor número de soluciones que se desarrollan para evitar la re-implementación y con ello la pérdida de tiempo en el proceso de desarrollo.

Por tanto podemos resumir como **situación problémica** la incapacidad de los agentes inteligentes de percibir visualmente su entorno y la necesidad de reutilizar código para agilizar el proceso de desarrollo.

Ante esta situación se puede definir el siguiente **problema científico**: ¿Cómo equipar a los agentes inteligentes con la capacidad de visión?

Para dar respuesta al problema de la investigación se define como **objetivo general** desarrollar un módulo de percepción visual para el Sistema de Percepción.

Para dar solución al objetivo general antes planteado se define como **objeto de estudio** los Sistemas de Percepción en entornos virtuales. Derivándose que el **campo de acción** que abarca esta investigación sería la percepción visual en los sistemas de percepción.

Esta investigación tiene la siguiente **idea a defender**:

Al utilizar el módulo de percepción visual los agentes inteligentes se desempeñarán de una forma mucho más realista en el entorno virtual basando gran parte de su toma de decisiones en lo que realmente ven.

Conforme a lo planteado como objetivo general se derivan las siguientes **tareas investigativas**:

- Propuesta de soluciones técnicas según el estudio del estado del arte.
- Estudio de la arquitectura del Sistema de Percepción.
- Selección de la metodología de desarrollo de *software*, lenguaje y herramientas a utilizar.
- Diseño del módulo percepción visual para el Sistema de Percepción.
- Generación de los artefactos ingenieriles según los flujos de trabajo que proponga la metodología seleccionada.
- Implementación del módulo de percepción visual para el Sistema de Percepción.
- Realización de pruebas al módulo para medir su funcionalidad.

Para desarrollar las tareas investigativas se hizo uso de los siguientes métodos:

Métodos teóricos:

- Histórico – Lógico: Búsqueda de información histórica acerca de los sistemas de percepción a nivel mundial y análisis del estado del arte del tema en la actualidad.

- Analítico – Sintético: Analizar toda la bibliografía y documentación obtenida y desglosar la misma en componentes y conceptos básicos que se relacionen entre sí para poder ir llevando un resumen o control que nos sirva de guía en el proceso de desarrollo.

Métodos empíricos:

- La consulta de fuentes de información: Permite realizar un estudio actual de las distintas bibliografías que existen del tema tratado.

- Experimento: Realizar pruebas experimentales al sistema sobre demos y otros proyectos de la línea para evaluar de manera práctica su correcto funcionamiento.

- Observación científica: Permite detectar el comportamiento desempeñado por los agentes controlados por el SP para identificar errores que afecten su accionar en el entorno que los rodea.

- Pruebas: Permite validar las funcionalidades de la aplicación que se desarrollará.

El contenido de este documento está estructurado en cuatro capítulos, así como las correspondientes secciones de las recomendaciones, bibliografía, el glosario de términos y los anexos; organizados de la siguiente forma:

Capítulo 1: Fundamentación teórica

Expone los principales conceptos relacionados con los Sistemas de Percepción y con las técnicas de selección de visibilidad, partiendo del estudio del estado del arte referente al tema tratado; así como la descripción de la arquitectura base del SP.

Capítulo 2: Propuesta y descripción de la solución

Ofrece una descripción de las características que presentará el sistema como solución al problema científico planteado. También serán definidas la metodología de desarrollo de *software*, el lenguaje de modelado, el lenguaje de programación y la herramienta de modelado; que permitirán dirigir y guiar el proceso de desarrollo para poder llegar a los resultados esperados. Se realiza además la modelación del negocio y la captura de los requisitos funcionales y requisitos no funcionales con la generación de sus respectivos artefactos.

Capítulo 3: Diseño e implementación

Este capítulo incluye la generación de los artefactos según la metodología seleccionada para los flujos de trabajo diseño e implementación. Se muestran los diagramas de clases del diseño y diagramas de secuencia de cada uno de los casos de usos generados a partir de los requisitos funcionales definidos en el capítulo anterior.

Capítulo 4: Pruebas y resultados

Incluye la validación del *software* mediante diseños de casos de pruebas tanto de Caja Blanca como de Caja Negra, así como los resultados obtenidos al desarrollar la aplicación los que permiten cumplir con los objetivos antes planteados.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se realiza estudio referente a los Sistemas de Percepción y de algunas de las técnicas de selección de visibilidad dentro de un entorno virtual y sus características, así como la descripción de la arquitectura que presenta el Sistema de Percepción.

1.1 La Percepción.

"Percepción es la habilidad de detectar elementos y sucesos presentes en el medio ambiente mediante mecanismos sensibles a algún tipo de información". (1)

Muchos trabajos de Sistemas Multiagentes Virtuales Inteligentes en Entornos Virtuales (EV) tienen como objetivo proveer a los agentes de un alto grado de realismo en cuanto a su representación física o incluso su comportamiento. Sin embargo, pocos estudios se han realizado centrándose en la percepción.

Es evidente que cualquier representación es una simplificación tremenda de la realidad, que tiene un nivel de detalle potencialmente infinito. La representación se fija solo en los aspectos relevantes para el objetivo del sistema.

La percepción trabaja a partir de los estímulos, que no son más que medidas de las magnitudes físicas. Los seres vivos codifican los estímulos mediante impulsos nerviosos; en los computadores los digitalizamos y convertimos en variables numéricas. Los estímulos son información bruta, sin elaborar, contaminados con ruido, con elementos irrelevantes, etc. Recibimos un flujo de miles o incluso millones de *bits* por segundo que tenemos que reorganizar y procesar para extraer las propiedades abstractas que nos interesan en cada momento.

La percepción requiere un gran esfuerzo computacional que, curiosamente, los seres vivos realizamos de manera inconsciente y automática. En la actualidad esta capacidad es difícilmente alcanzable incluso para los más potentes supercomputadores. En contraste, ciertas tareas que son triviales para un computador resultan muy complejas para los seres vivos. (2)

1.2 Sistemas de Percepción.

Últimamente se han desarrollado varios sistemas de percepción con diferentes finalidades en dependencia de la necesidad a satisfacer. Presentan múltiples usos como en detección, monitorización y modelado de fuegos forestales, protección del medio ambiente, simulación de la percepción sensitiva del hombre, estudiar los lazos que establece un individuo con el espacio que le rodea en una situación de conducción, y es utilizado ilimitadamente en los video juegos. A continuación se muestran algunos de los sistemas de percepción desarrollados en el mundo.

- **Modelo de Percepción para Agentes Virtuales Inteligentes basado en el sistema de percepción de los seres humanos:** Sistema que permite introducir una percepción sensitiva similar a la que poseen los seres humanos que permite a los agentes virtuales inteligentes (AVIs), percibir en tiempo real el entorno que le rodea con una claridad de percepción similar a la que poseen los seres humanos. (3)

- **Sistema de Simulación para estudiar la percepción humana**: Estudia los mecanismos unidos a la percepción sensorial del movimiento y del espacio, que permitiendo principalmente una optimización de las herramientas empleadas en materia de realidad virtual. El objetivo del sistema es estudiar los lazos que establece un individuo con el espacio que le rodea en una situación de conducción. En concreto, se trata de investigar las sensaciones de desplazamiento, como la aceleración y el frenado. Los investigadores estudiarán mediante este sistema también el escenario próximo al conductor, y particularmente su percepción en el hábitat. (4)

- **Control de crecimiento y densidad de cultivos de Piscifactoría:** Se realiza la aplicación de técnicas de visión por computadora mediante procesamiento de imágenes y- sensores de proximetría para determinar el número, tamaño y forma de individuos de diferentes especies en diferentes estados de crecimiento. La adquisición de imágenes se realiza tanto desde el exterior de los tanques como en su interior. (5)

Estos son sistemas especializados en la percepción del medio ambiente detectando y captando toda la información presente en el medio. La Información captada será luego procesada en los algoritmos del Sistema de Percepción lo cual posibilitará en consecuencia la reacción apropiada del agente inteligente.

1.3 Percepción en Agentes Inteligentes.

La percepción se considera como uno de los temas más importantes dentro de la robótica y los videojuegos que son creados bajo la base de la Inteligencia Artificial. Cada juego dirige la

percepción de manera diferente, se conoce que la percepción más sofisticada puede imitar las limitaciones del mundo real.

La percepción provee de información a los agentes virtuales para que parezcan lo más real posible. El adelanto en las tecnologías ha permitido al hombre interactuar con mundos virtuales interactivos, esta interacción del usuario con estos entornos virtuales han llevado a la necesidad de crearlos de forma tal que recreen fielmente la realidad, y para ello se debe equipar a los Agentes Inteligentes con sistemas que permitan que su comportamiento en los entornos virtuales de forma que simulen la inteligencia a la hora de tomar decisiones. Surge entonces la necesidad de diseñar y crear sistemas de percepción capaces de recopilar la información necesaria del entorno que rodea al Agente Inteligente.

La percepción es mucho más que ver y oír, abarca todas las formas en que un **NPC** (*Non Playable Character*, un personaje del juego) obtiene la información sobre el mundo, incluyendo la topología del entorno y otros agentes. (5)

En la actualidad la mayoría de sistemas de percepciones que se desarrollan no solo recopilan esta información, se centran mayormente en la simulación y el entrenamiento de agentes que se especializan para vivir situaciones reales que pueden ser peligrosas para el usuario.

1.4 Percepción Visual.

La percepción visual (PV) permite la comprensión o el conocimiento del medio ambiente donde se encuentran. Visión es el proceso mediante el cual se descubre qué es lo está presente en el mundo y dónde está. (6)

La PV es la encargada de proporcionar la información del entorno a los agentes inteligentes mediante técnicas de selección de visibilidad. Estas técnicas permiten determinar cuáles son los objetos o elementos de la escena que se encuentran dentro del el campo visual del agente y llevándolo a un nivel superior determinar cuáles de los objetos que están en el campo visual se encuentran ocluidos por otros objetos. La información obtenida a través de la PV permitirá al agente desarrollar un comportamiento mucho más realista a la hora de interactuar con el entorno.

Los trabajos realizados por Herrero y De Antonio (7) han sido de gran ayuda para la investigación dado a que brindan una serie de conceptos claves sobre la percepción y además brinda un mecanismo de PV para dotar a los agentes de mayor realismo.

1.4.1 Técnicas de determinación de visibilidad:

El proceso de selección de visibilidad es muy complejo y costoso el cual cuenta con una serie de técnicas encargadas de enviar la menor cantidad de objetos a procesar, incrementando el

rendimiento de nuestro programa. Con el uso de las técnicas de selección de visibilidad se logra un mayor rendimiento de los recursos disponibles. Encargándose de seleccionar cuales son los objetos potencialmente visibles y descartar en la medida de lo posible aquellos que sean innecesarios. Entre las técnicas de visualización más usadas están aquellas encargadas a descartar un grupo de objetos en dependencia de las posiciones en que se encuentren, las más usadas son: *Occlusion Culling* y *View Frustum Culling.* A continuación se explicará brevemente el proceso de selección de visibilidad con las técnicas más usadas.

1.4.1.1 *Frustum Culling.*

Una de las técnicas más usadas es la del *View Frustum Culling*. Para ello cada objeto es encuestado con los planos del *Frustum* (el *Frustum* es una clase de pirámide truncada en entornos 3D que representa lo que visualiza la cámara) y el objeto que se encuentre fuera de esta es eliminado del conjunto de posibles objetos a visualizar, evitando el posterior procesado, ver Figura 1.1 (8)

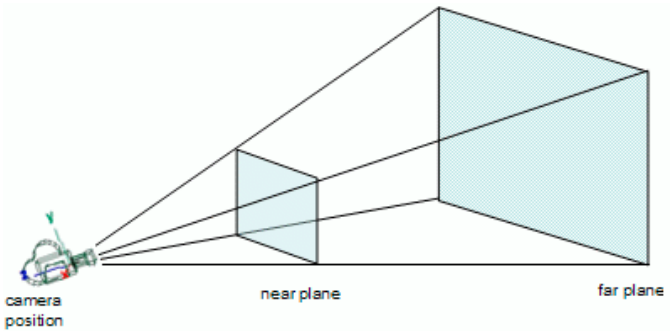

Figura 1.1 *Frustum Culling*.

El *frustum* es básicamente el volumen de visualización generado a partir de una matriz de proyección en perspectiva; la forma de este volumen es el de una pirámide con la punta recortada. El *frustum* se encuentra delimitado por 6 planos: Plano de recorte lejano, plano de recorte cercano, plano superior, plano inferior, plano izquierda y plano derecho.

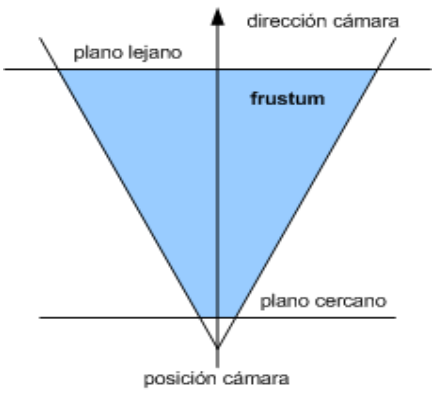

Figura 1.2 *Frustum Culling* vista superior.

La idea básica del *frustum culling* será aplicar un pequeño algoritmo para verificar si un determinado objeto se encuentra dentro o fuera del frustum y de este modo saber si debemos procesarlo o no.

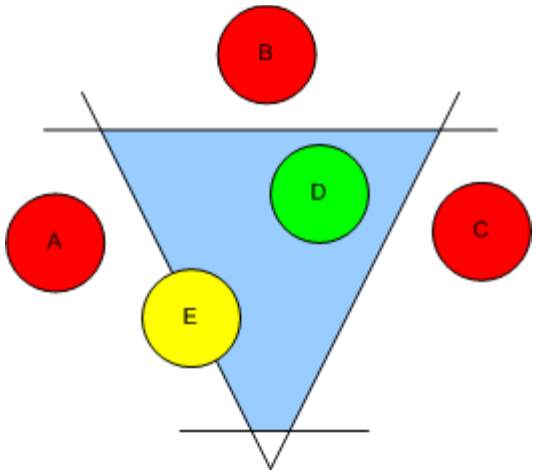

Figura 1.3 Las distintas situaciones con las cuales nos podemos encontrar.

En la figura 1.3 se especifican todos los casos que podremos encontrar cuando realicemos el frustum culling: En el caso A, B y C los objetos se encuentran totalmente fuera del frustum por lo tanto no serán vistos por el jugador y por lo tanto no deberemos procesar ninguno de sus vértices; en el caso D, el objeto se encuentra totalmente dentro del *frustum* y naturalmente debe ser visualizado; en el caso E el objeto se encuentra parte dentro y parte fuera del *frustum*, en estos casos usualmente se visualiza el objeto completo.

Como habíamos mencionado anteriormente, el *frustum* se encuentra definido por seis planos, para verificar si un objeto se encuentra dentro de él deberemos aplicar un algoritmo que tenga en cuenta estos planos. El objeto contra el cual se realiza la verificación es un volumen de encierro, usualmente, una esfera o una *bounding box* debido a que con esto simplificaremos mucho las cuentas.

1.4.1.2 *Occlusion Culling*.

Una vez realizado el *Frustum Culling* se realiza un análisis que trata de determinar cuáles son los objetos que están ocultos detrás de otros como se muestra en la Figura 1.4 y por lo tanto no contribuyen a la visión final pero si son procesados, reduciendo considerablemente el rendimiento del programa, esta técnica es conocida como *Occlusion Culling*.

Occlusion Culling es un algoritmo de determinación de la visibilidad que se utiliza para identificar a los objetos que residen en el volumen de vista, pero todavía no son visibles al agente debido a la oclusión. Eso significa que están ocultos por esos objetos que están más cerca de la cámara. (9)

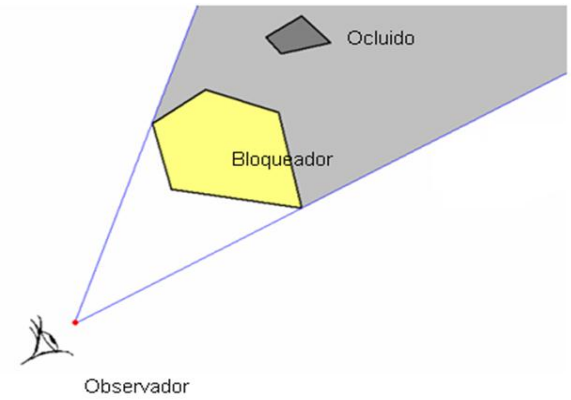

Figura 1.4 *Occlusion Culling*.

1.4.1.3 *Ray casting*.

El algoritmo *Ray casting* intenta combinar la idea de trazar rayos para determinar las superficies visibles con la de un proceso de sombreado, donde se tiene en cuenta los efectos globales producidos por las reflexiones, refracciones y sombras arrojadas por otros objetos de la escena. Para conseguir los efectos de reflexión y refracción se trazan rayos recursivamente desde el punto intersección que se está sombreando teniendo en cuenta las propiedades del material del objeto interceptado. (10)

Hoy en día la percepción de los NPC se basa casi exclusivamente en el *Ray Casting*. Esta técnica no sólo brinda una comprensión muy pobre del entorno, sino que también afecta gravemente las interpretaciones y ejecuciones. A la vez también afecta considerablemente la eficiencia del sistema. Tienes que lanzar muchos rayos para obtener una respuesta a preguntas tan sencillas como: ¿qué es lo que realmente veo?".

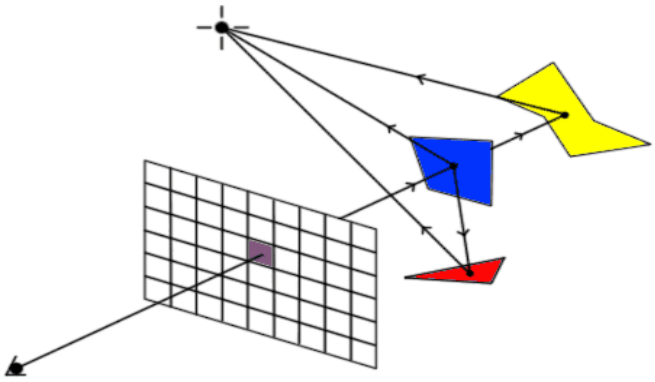

Figura 1.5 Trazado de Rayos.

1.4.2 Volúmenes de encierro.

Los volúmenes envolventes o volúmenes de encierro son objetos contenedores de los objetos de la escena, estos tienen como objetivo simplificar su geometría y así reducir el coste de operaciones como la visibilidad y la detección de colisiones. Si durante la ejecución de los algoritmos los objetos cuyo volumen envolvente se encuentran fuera del volumen del *Frustum* no son procesados. Reduciendo así el tiempo del algoritmo del *Frustum Culling*, lo cual permite un incremento del microprocesador y liberar procesos para otras tareas.

Los volúmenes envolventes los más utilizados en la creación de los grafos de escenas son los AABB (*Axis Aligned Bounding Boxes*), Esferas Frontales (*Bounding Spheres*), OBB (*Oriented Bounding Boxes*).

AABB: Son cajas alineadas por seis valores tres X, Y, Z máximos y mínimos, esto tiene como ventaja que el cálculo de la normal y la prueba de inclusión de un punto en el objeto son fáciles de obtener, el problema de este volumen recae en que no se ajusta de la mejor manera a los objetos, ver figura 1.6.

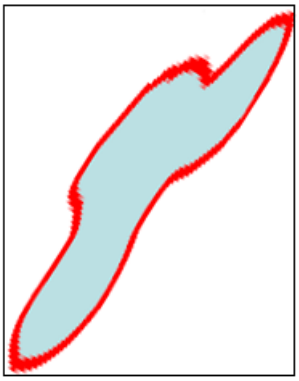

Figura 1.6 AABB (*Axis Aligned Bounding Boxes*).

Esferas Frontales: Los volúmenes envolventes de esfera están definidos por un punto y un radio, son muy rápidos a la hora de encuestar el volumen de encierro, ya que solo hay que verificar si la esfera está interceptando el *Frustum*, el problema radica que si el *Frustum* intercepta la esfera y los objetos contenidos no se ven se van a estar pintando sin estar dentro del *Frustum*, ver figura 1.7.

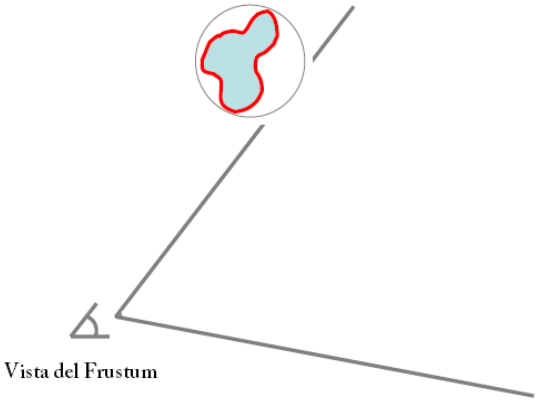

Figura 1.7 Esferas frontales (*bounding spheres*).

OBB (Oriented Bounding Box): Son cajas definidas por cuatro u ocho puntos orientados a los objetos y se adaptan más a su forma, permitiendo que la encuesta sea más exacta, pero a la vez más compleja, ver Figura 1.8).

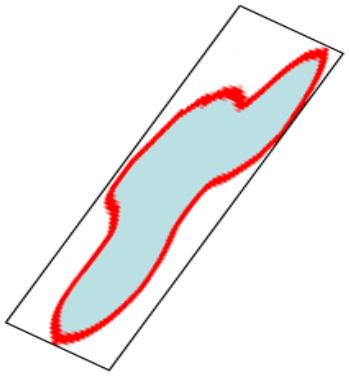

Figura 1.8 OBB (*Oriented Bounding Box*).

1.5 Arquitectura del Sistema de Percepción.

Para lograr que los agentes se comporten de con un adecuado nivel de realismo en su comportamiento, es necesario dotarlos de la capacidad de percibir el entorno que los rodea. Para ello es necesario trabajar con la arquitectura del Sistema de Percepción, que será el encargado de otorgar a los agentes las habilidades de observar los elementos topológicos y medioambientales del entorno que los rodea.

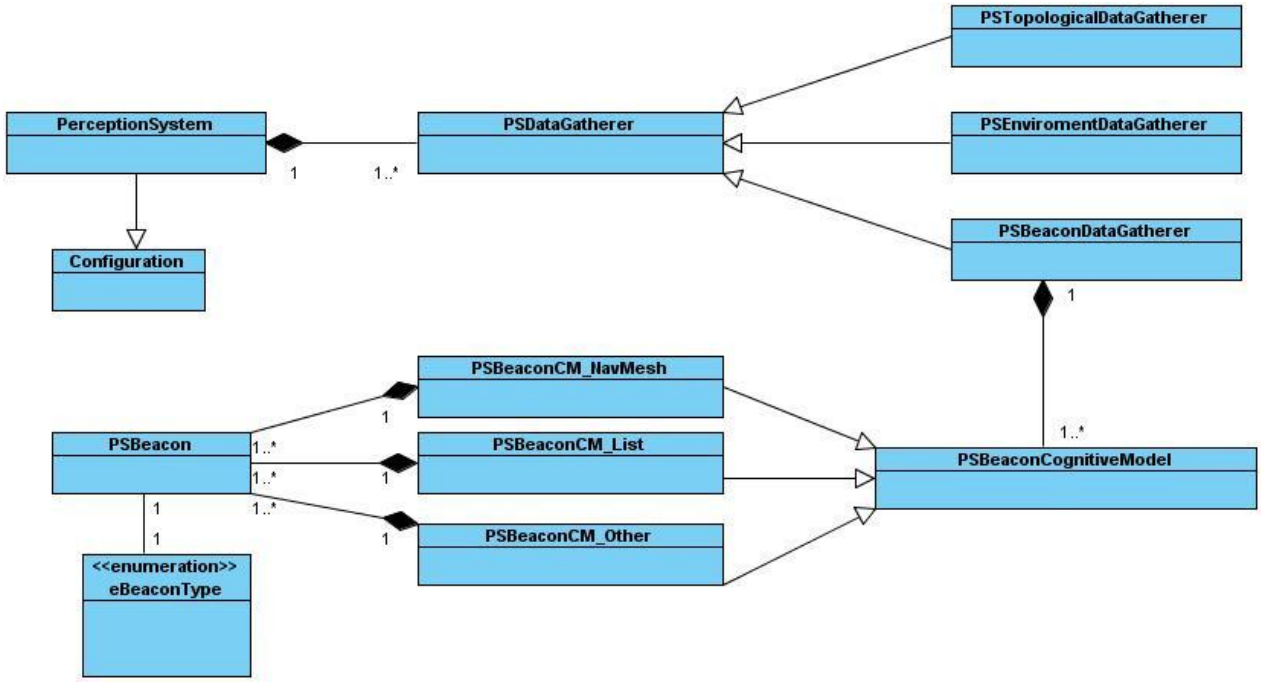

Figura 1.9 Arquitectura del Sistema de Percepción.

La arquitectura del SP (Figura 1.9), proporciona una serie de elementos a tener en cuenta para la realización del módulo:

- Mediante el objeto *PerceptionSystem* se realiza la actualización de todas las entidades *PSDataGatherer.*
- Mediante la clase *PSDataGatherer* se puede almacenar la información de percepción sobre el mundo que puede ser detectada por los agentes.
- La información detectada puede ser de cualquier tipo como: datos topológicos, datos ambientales y hasta los acontecimientos que suceden en el mundo (*PSTopologicalDataGatherer, PSEnvironmentalDataGatherer, PSBeaconDataGatherer,* respectivamente).
- El objeto *PSTopologicalDataGatherer* es el encargado de buscar la información acerca de las características tácticas del entorno que rodea al agente.
- El objeto *PSEnvironmentDataGatherer* es el encargado de recolectar una apreciación del entorno que rodea al agente mediante una representación simplificada del mismo basado en formas básicas.

- Por medio del recolector *PSBeaconDataGatherer* los agentes tienen conocimientos de los objetos tanto dinámicos como estáticos o eventos que son creados y modificados en tiempo de ejecución.

- En la clase *PSBeacon* se almacenan las características relacionadas con los eventos dinámicos y que son de interés para el agente, como pueden ser el identificador del evento, el radio del elemento, la posición, entre otras características.

1.6 Interacción del Agente con el Sistema de Percepción.

Sensores:

Como en el mundo real, el agente inteligente recibe la información del ambiente a través de los sensores. Estos sensores examinan constantemente el entorno en busca de información útil y mantienen actualizado al agente. Para realizar esto, se conectan a los distintos recolectores de datos en el Sistema de Percepción que proporcionan datos medioambientales, topológicos y dinámicos (*Beacon*) del mundo.

Cuando se crea un agente, se registran los sensores que necesitará. Para un personaje de un perro, no es necesario registrar un sensor topológico porque no necesita recibir la información sobre dónde preparar una emboscada. Sin embargo, necesita registrar un sensor medioambiental para evitar las colisiones y saber dónde va a realizar su próximo movimiento. También, necesitará un *Beacon* sensor para descubrir al jugador.

Cada uno de los sensores tiene propiedades de filtro que determinan las capacidades sensoriales del agente asociado. Esos atributos se definen según el tipo de datos que el sensor está diseñado para buscar. Por ejemplo, los sensores medioambientales y topológicos definen los atributos de visibilidad que permiten devolver los datos más cercanos y visibles cuando se realiza el escaneo. De manera similar ocurre con el *Beacon* sensor que proporciona los atributos para especificar los tipos de *Beacon* que detectará.

Cada vez que un agente se actualiza todos sus sensores se actualizan también, causando que se explore el recolector de datos a los que un agente se conecta. Este escaneo que es implementado en los recolectores de datos de manera que se optimiza según los modelos cognitivos elaborados, permitiendo búsquedas rápidas a partir de la vía de árboles de espacio particionado para grandes entornos con gran cantidad de información útil o buscar las listas simples cuando se examinan los ambientes pequeños.

1.7 Componentes de un agente.

Por su parte el agente recibe la información del entorno a través de sensores (Figura 1.10). Estos sensores escanean constantemente el entorno en busca de información y mantienen actualizado al agente. Para ello, estos sensores se conectan a los *DataGatherer* del Sistema de Percepción, los cuales le proveen de la información táctica, de *Beacon* y de estructura del entorno.

Cuando se crea un agente, a través de la interfaz *AgentOgre*, la cual conecta el Sistema de Percepción al módulo de Inteligencia Artificial (IA), se registran los sensores que va a utilizar (Figura 1.10), ya que todos los agentes no necesitan conocer todo tipo de información.

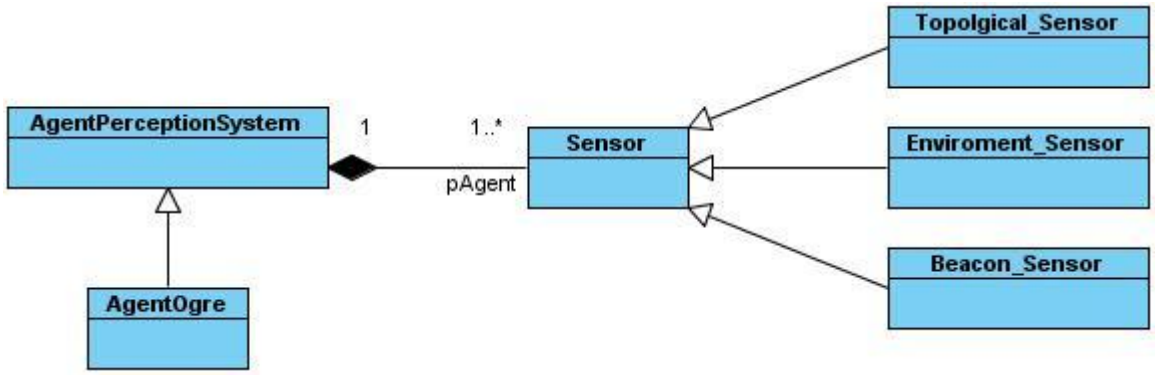

Figura 1.10 Diagrama de clases de los componentes de un agente.

Una vez que los sensores activan una búsqueda en los recolectores de datos, éstos le reportan al agente la información acerca del entorno, la cual se usa en la ejecución de los algoritmos de percepción visual.

1.8 Conclusiones del capítulo.

Una vez analizada la información sobre las técnicas de selección de visibilidad y su modo funcionamiento así como la arquitectura general del SP y la importancia de equipar a los agentes inteligentes con la capacidad de percepción visual concluimos en que:

La arquitectura presentada por el SP maneja las necesidades básicas de percepción de un agente inteligente, también puede ser ampliada para incluir algunas capacidades a los agentes como es el caso de la percepción visual del entorno. Por lo que se decide desarrollar un módulo de percepción visual para este sistema que se está implementando. Dicho módulo será el encargado de determinar todos los elementos visibles para un agente en una escena, brindando varios niveles de complejidad a la percepción del agente a través de algoritmos híbridos entre las técnicas de selección de visibilidad antes expuestas, dejando a criterio del desarrollador la opción a escoger.

CAPÍTULO 2

PROPUESTA Y DESCRIPCIÓN DE LA SOLUCIÓN

Introducción.

En el presente capítulo se ofrece una descripción de las características del módulo como solución al problema científico planteado. Para ello se define la metodología de desarrollo de *software*, las herramientas y lenguajes a utilizar para llegar a los resultados esperados. Se presentarán los artefactos obtenidos de los flujos de trabajo ingenieriles: modelo del negocio y requerimientos.

2.1 Metodología de desarrollo de *software*.

La metodología de desarrollo de un *software* es la guía para realizar un *software* con calidad; definiendo, en un proyecto de *software*, quién debe hacer qué, cuándo y cómo debe hacerlo. Una metodología es un proceso y se desarrolla con el objetivo de dar solución a los problemas que existen en la producción del *software*. Cuenta con un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo *software*. (12)

2.1.1 Proceso Unificado Racional (RUP).

El Proceso Unificado Racional es una de las metodologías estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Su meta es asegurar la producción de *software* de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible.

El Proceso Unificado se basa en componentes, lo que significa que el sistema en construcción está hecho de componentes de *software* interconectados por medio de interfaces bien definidas. Este además usa el Lenguaje Unificado de Modelado (UML) en la preparación de todos los planos del sistema. De hecho, UML es una parte integral del Proceso Unificado, fueron desarrollados a la par. (13) Los aspectos distintivos del Proceso Unificado están capturados en tres conceptos clave:

- **Dirigido por casos de uso:** los casos de uso son una herramienta para especificar los requerimientos del sistema, dirigen su diseño, implementación y pruebas, dirigen el proceso de desarrollo.

- **Centrado en la arquitectura:** la arquitectura es la vista del diseño completo con las características más importantes hechas más visibles y dejando los detalles de lado.
- **Iterativo e incremental:** es práctico dividir el trabajo en pequeños pedazos o mini-proyectos. Cada mini-proyecto es una iteración que finaliza en un incremento. (13)

2.2 Herramienta de modelado.

Visual Paradigm es una herramienta de modelado multiplataforma y muy potente. Soporta el ciclo de vida completo del desarrollo de *software*, ayuda a una rápida construcción de aplicaciones de calidad y a obtener un menor coste en las mismas. Permite dibujar todos los tipos de diagramas de clases, obtener código inverso, generar código desde diagramas así como la documentación que se necesite de cada uno de ellos.

Visual Paradigm hace uso de un lenguaje común a todo el equipo de desarrollo facilitando la comunicación entre los mismos. También presenta capacidades de ingeniería directa e inversa además de presentar disponibilidad de múltiples versiones para cada necesidad que se necesite. Es un producto de elevada calidad y altamente usado en la actualidad pues es fácil de instalar, actualizar y manipular.

2.3 Lenguaje de modelado.

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de *software*. Ofrece un estándar para describir un "plano" del sistema o modelo, incluyendo aspectos conceptuales tales como los procesos de negocios, funciones del sistema y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de *software* reutilizables. Es un lenguaje de propósito general para el modelado orientado a objetos. Este es también un lenguaje de modelado visual que permite una abstracción del sistema y sus componentes. UML ha ejercido un gran impacto en la comunidad *software*, su utilización se ha extendido en todo el mundo para construir aplicaciones en todos los dominios y de todos los tamaños. (14) (15) (16)

2.4 Lenguaje de programación.

C++ es el lenguaje multiplataforma mayormente utilizado para la realización de las aplicaciones de Realidad Virtual, pues en este tipo de aplicaciones se manejan grandes volúmenes de datos y el mismo permite un uso óptimo de la memoria y del *CPU* (Unidad Central de Procesamiento) de la computadora donde se instalará la aplicación desarrollada. Este lenguaje presenta entre sus principales características que es un lenguaje versátil, potente y de propósito general. Es rico en

operadores y expresiones y presenta programación modular, además es flexible, conciso, eficiente, portable y breve.

Se trata de un lenguaje de programación estandarizado (ISO/IEC 14882:1998), ampliamente difundido, y con una biblioteca estándar C++ que lo ha convertido en un lenguaje universal, ampliamente utilizado tanto en el ámbito profesional como en el educativo. También posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel como la posibilidad de redefinir los operadores (sobrecarga de operadores) entre otras.

2.5 Descripción de la solución.

Para el desarrollo del módulo se implementaron tres algoritmos correspondientes a cada uno de los niveles de complejidad definidos para el módulo, el nivel básico, el medio y el avanzado. Estos algoritmos son el resultado del estudio de la documentación obtenida a lo largo de la investigación y el análisis de la complejidad computacional de los mismos, pues es de vital importancia que este tipo de percepción no consuma muchos recursos debido a que se ejecuta en tiempo real cada vez que se actualiza el Sistema de Percepción general.

2.5.1 Percepción Visual Básica.

En el primer nivel o nivel básico de percepción visual, contamos con un algoritmo en el cual se le define al agente inteligente un radio de visión. Luego, teniendo en cuenta la posición en el sistema de coordenadas global del agente y de los obstáculos que están presentes en la escena, se calcula la distancia entre los mismos y se compara con el radio o capacidad de visión máxima del agente. Si esta distancia es menor que el radio de visión del agente, entonces este puede ver el obstáculo. (Figura 2.2)

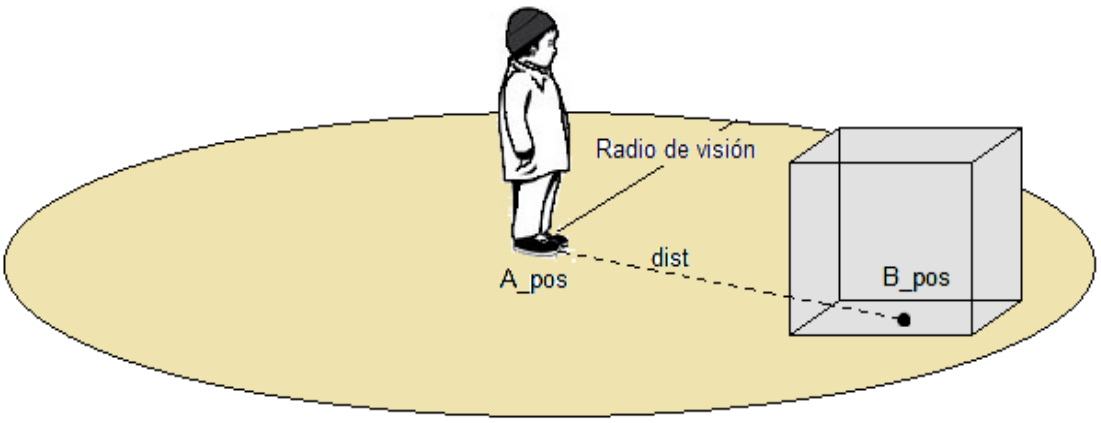

Figura 2.1 Radio de visión.

2.5.2 Percepción Visual Media.

En el segundo nivel o nivel medio de percepción visual se implementa un algoritmo en el cual primeramente mediante una instancia de la clase *PerceptionSystem* se hace una llamada al método obtainData() de la clase *BeaconDataGatherer,* que es el encargado de recolectar toda la información de tipo *Beacon* del sistema, una vez obtenida esta información se alacena la misma en la estructura de datos std::vector que brinda la biblioteca *iostream.h*.

Posteriormente se calculan las proyecciones del vector posición del agente inteligente sobre el plano X-Z llevándolo a dos dimensiones.

Luego se calcula la posición hacia la cual se encuentra mirando el agente es ese momento con una operación vectorial dada por la siguiente fórmula:

Lookpoint = P + lookvector * maxView

Lookpoint: Posición hacia la cual mira el agente.
P: Posición del agente.
maxView: Distancia máxima de visión del agente.

Una vez calculados estos dos puntos se recorre el listado de objetos y se calculan las distancias entre los tres puntos que conformarán la triangulación del algoritmo para cada objeto, acto seguido se calcula el ángulo que se forma entre el punto hacia el que se encuentra mirando el agente, la posición del agente y la posición del obstáculo encuestado en ese momento.

Si la distancia entre el agente y el obstáculo es menor que la distancia máxima de visión del agente y el ángulo comprendido entre los tres elementos es menor al ángulo de visión definido por el programador entonces el obstáculo se inserta dentro de la lista que posteriormente se devolverá con los elementos visibles en ese momento. Figura 2.3

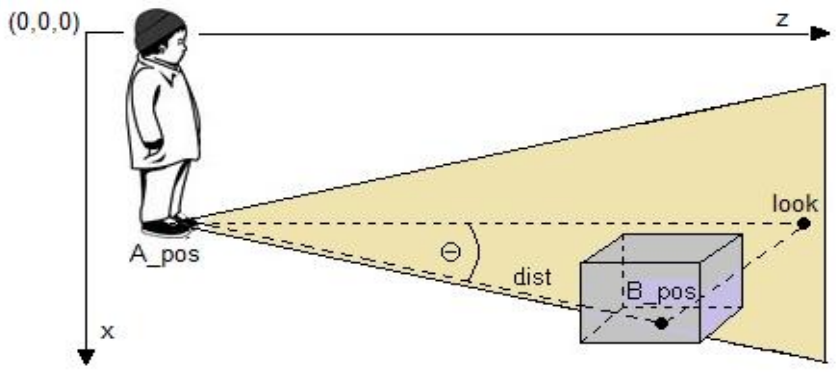

Figura 2.2 Proyección de los vectores posición sobre plano X-Z.

2.5.3 Percepción Visual Avanzada.

Este algoritmo realiza la percepción del entorno basándose en el modo de funcionamiento del algoritmo *Frustum Culling* en el cual se define un volumen de visión, con la diferencia de que en el algoritmo implementado se define como volumen de visión un cono y el volumen de visión del *Frustum Culling* lo define una pirámide truncada. Esta diferencia es debido a que el frustum culling utiliza más recursos pues necesita crear los 6 planos que conforman el volumen de la pirámide, mientras que si utilizamos un cono, este se puede definir simplemente con una recta central y un ángulo, lo cual agiliza el algoritmo a la hora de las encuestas contra todos los objetos y a la vez economiza recursos que son vitales, pues todos los algoritmos corren en tiempo real.

2.5.3.1 Niveles de visibilidad:

Para realizar este algoritmo se tuvieron en cuenta 3 niveles de visibilidad los cuales son:

- El volumen envolvente está completamente fuera del volumen de visualización. Es decir, ni el vector posición, ni ninguno de sus vértices se encuentra dentro del volumen de visión. El Beacon es descartado y no se incluye dentro de los objetos visibles.

- El volumen envolvente está completamente dentro del volumen de visualización. Esto significa que tanto su vector posición como todos los vértices del volumen envolvente se encuentran dentro del volumen de visión. El Beacon se incluye dentro de los objetos visibles.

- El volumen envolvente se intercepta con el volumen de visualización. Por lo tanto, no todos los vértices del volumen envolvente se encuentran dentro del volumen de visión del agente. En este caso el objeto es incluido dentro de los objetos visibles.

Al igual que en los algoritmos anteriores, en este algoritmo se obtiene el listado de los objetos almacenados en el modelo cognitivo usado. Luego se calcula la posición hacia la cual “mira” el agente y la posición real desde la cual se creara el origen del cono de visión, o sea, los “ojos” del agente.

Posteriormente procedemos a recorrer la lista de objetos y dado el radio y la altura de cada objeto creamos un volumen envolvente en forma de *Axis Aligned Bounding Box* el cual va a estar conformado por 8 vértices, los cuales vamos a utilizar durante la ejecución del algoritmo.

Buscando la vía de mejorar el algoritmo, debido a la importancia y complejidad computacional que tiene el mismo y teniendo en cuenta que se ejecuta constantemente en tiempo de ejecución, primeramente hacemos la triangulación entre las posiciones del punto hacia el cual “mira” le agente, su vector posición y el vector posición del Beacon, luego de calcular el ángulo comprendido determinamos si el mismo se encuentra dentro del volumen de visión del agente. Si el vector

posición del Beacon se encuentra dentro del volumen visual inmediatamente lo incluimos dentro de la lista de los elementos visibles. En caso contrario pasamos a encuestar el mismo algoritmo contra todos los vértices del volumen envolvente creado anteriormente sobre el Beacon. Esto nos garantiza que si en una primera iteración el vector posición del Beacon se encuentra dentro del volumen de visión entonces ese Beacon va a ser visible para el agente, evitando la comprobación contra todos los vértices del volumen envolvente. Posteriormente pasamos a determinar la claridad de percepción con la cual el agente es capaz de ver los elementos del escenario virtual, para realizar esta operación se definieron dos criterios para la claridad de percepción.

2.5.3.2 Claridad de Percepción.

Cuando un agente percibe un objeto en el entorno, la percepción es diferente dependiendo del área donde este objeto se encuentre situado. Tanto la posición del objeto como la orientación del ojo del agente y la región donde está situado juegan un papel muy importante a la hora de determinar la claridad de percepción del objeto en el entorno. En la Figura 2.4 el área de percepción (AP) indica si un objeto se encuentra dentro del volumen de visión y si esto ocurre, dentro de que área se encuentra situado. Teniendo en cuenta lo antes planteado se definen los siguientes criterios para la claridad de percepción:

Criterio del ángulo:

Si el ángulo conformado entre el punto hacia donde mira el agente, el vector posición del agente y la posición del Beacon es mayor de 20º entonces la claridad de percepción que va a tener el agente sobre ese Beacon es baja.

En caso contrario, si el ángulo comprendido es menor de 20º entonces la claridad de percepción sobe el Beacon es alta.

Criterio de la distancia:

En el caso donde la distancia entre el vector posición del agente y el vector posición del Beacon es mayor de 15 cm y menor que el 75% de la distancia máxima de visión del agente entonces la claridad de percepción del agente sobre el Beacon va a ser alta. En caso contrario, si la distancia es menor de 15 cm o mayor que el 75 % del al distancia máxima de visión la claridad de percepción sobre ese Beacon es baja.

Una vez implementados dichos criterios, la visión del agente queda delimitada por dos conos de visión, uno que delimita el volumen de visión clara o central del agente y otro que delimita el volumen de visión difusa o lateral. Como se muestra en la Figura 2.3.

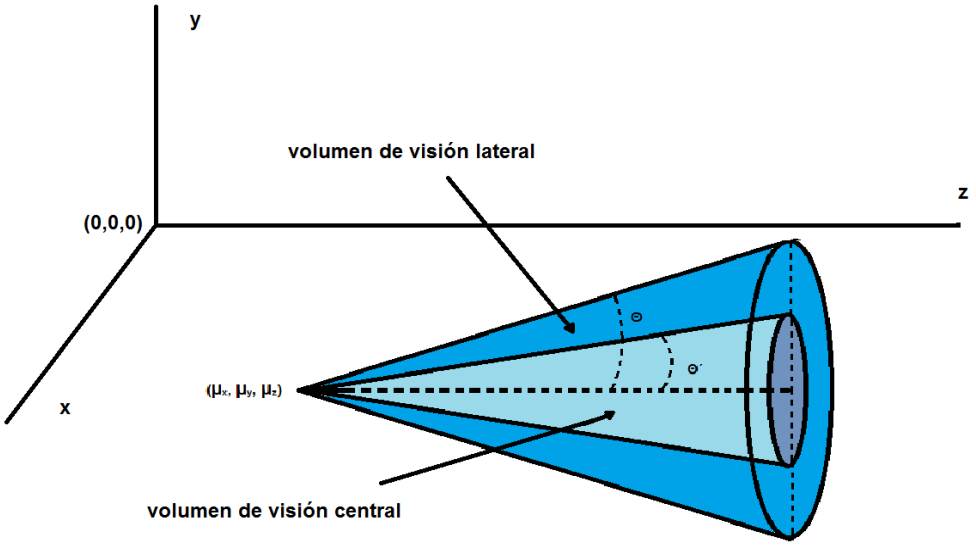

Figura 2.3 Volúmenes de visión.

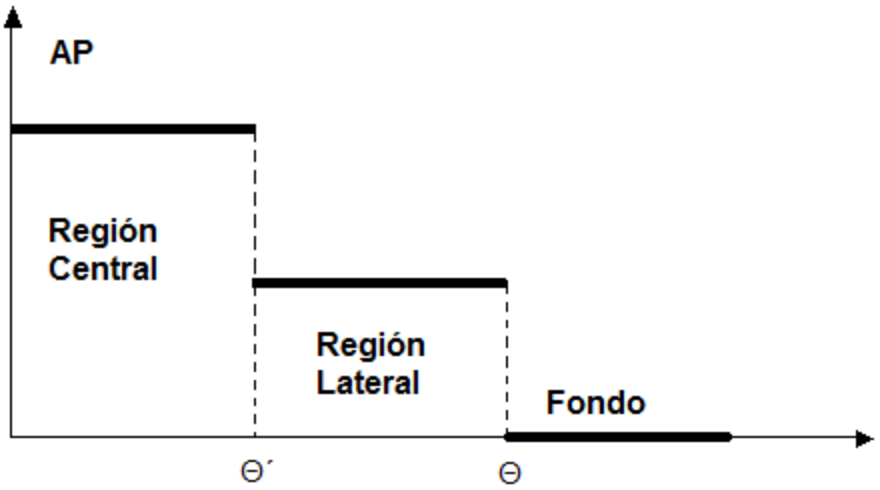

Figura 2.4 Volumen de Visión central y lateral (componente espacial).

Una vez analizadas las técnicas presentadas en el capítulo anterior y la arquitectura que presenta el Sistema de Percepción, como base para el desarrollo del módulo, se llegan a las siguientes conclusiones:

- Debido a que la arquitectura del Sistema de Percepción obtiene la información del entorno almacenada en un modelo cognitivo, a través de sensores especializados de tipo topológico, ambiental y Beacon, permitiendo a los agentes inteligentes conocer todo lo que les rodea se pueden implementar los diferentes algoritmos para determinar qué es lo que realmente ve le agente. Se puede observar entonces que la arquitectura del SP brinda soporte para la implementación los diferentes algoritmos de visibilidad con solo hacer algunas modificaciones a la misma como es el caso de inclusión de nuevas clases.

Se decide entonces realizar una serie de algoritmos basándonos en las diferentes técnicas antes expuestas en el Capítulo 1, que brinden a los desarrolladores varios niveles de percepción visual a partir de la información recolectada por los sensores del agente. Estos algoritmos deberán determinar, una vez obtenida la información, cuáles son los objetos tanto estáticos como dinámicos que son visibles para el agente inteligente.

Para obtener lo antes planteado es necesario hacer una modificación en la arquitectura del Sistema de Percepción. Esta modificación consiste en agregar a la arquitectura existente, las siguientes clases:

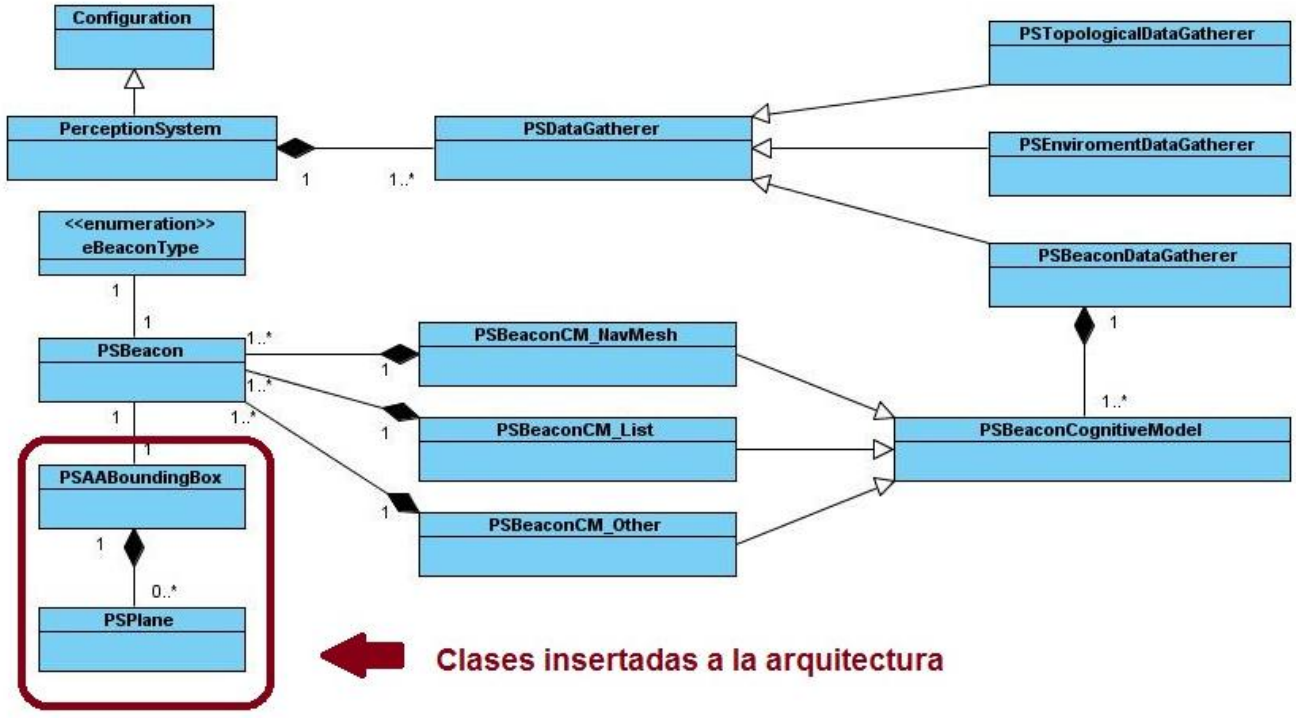

Figura 2.5 Clases agregadas a la arquitectura del Sistema de Percepción.

- ***PSAABoundingBox:*** Esta clase se encarga de construir un volumen de encierro que rodea a los Beacons en forma de Axis Aligned Bounding Box (AABB) el cual va a estar alineado a los ejes de coordenadas teniendo en cuenta su posición, radio y altura. Este volumen de encierro es utilizado posteriormente para la realización de los algoritmos.

- ***PSPlane:*** Esta clase es la encargada de crear los planos laterales del volumen de encierro guardando los valores mínimos y máximos del sub-plano definido por el volumen. Todos estos valores juntos a la ecuación del plano son utilizados posteriormente en los algoritmos.

2.6 Modelamiento del negocio.

Unas de las aproximaciones que existen en la Ingeniería de *Software* para expresar el contexto de un sistema utilizable son el modelo del negocio y el modelo de dominio.

En este trabajo se realiza modelo de dominio ya que no se logra determinar el proceso del negocio con fronteras bien establecidas, no se logra ver claramente quienes son las personas que lo inician, los beneficiados con cada uno de estos procesos, pero además quienes son las personas que desarrollan las actividades en cada uno de estos procesos.

2.6.1 Modelo del dominio.

El Modelo de Dominio es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema e incorpora conceptos del mundo real, no de los componentes de *software*, además es importante señalar que se le considera en RUP un subconjunto del llamado modelo de objetos del negocio.

Este modelo se realiza a través de un diagrama de clases de UML simplificado, en el cual se representan las clases conceptuales que pueden intervenir en el sistema y sus asociaciones preliminares, así como los objetos más importantes en el mismo.

El modelo del dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema. Por ello, a continuación se representa un acercamiento a la solución propuesta, donde se modelan los principales conceptos con los que se trabajarán en el desarrollo de la solución, así como las relaciones existentes entre ellos.

2.6.1.1 Diagrama del Modelo de Dominio:

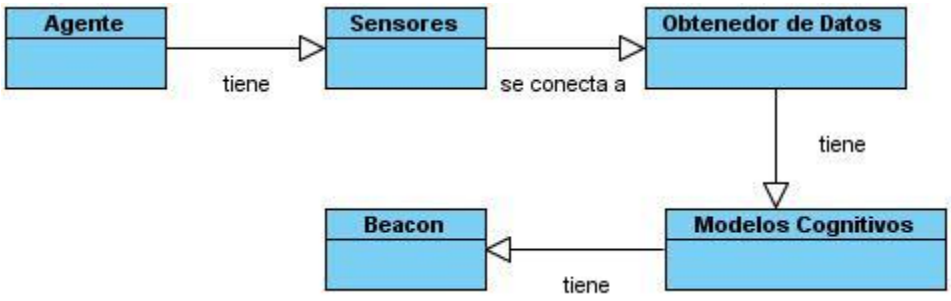

Figura 2.6 Modelo del dominio.

2.6.1.2 Glosario de Términos del Modelo de Dominio.

Agente: es la entidad inteligente por la cual estará conformado el SP, siendo capaz de percibir el entorno que lo rodea, interactuar con él y con otros agentes; y tomar las decisiones que sean necesarias para darle solución a los problemas u objetivos que se le puedan presentar.

Sensores: son los encargados de entregarles a los agentes la información del entorno que éstos necesitan para resolver sus problemas.

Obtenedor de datos: es el encargado de recolectar la información almacenada en los modelos cognitivos para su posterior procesamiento.

Modelos Cognitivos: contienen toda la información del entorno virtual.

Beacon: Es la unidad básica que representa los elementos presentes en la escena con los cuales interactúa el agente inteligente.

2.7 Requerimientos del sistema.

Los requerimientos de un sistema definen qué es lo que este debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen. Un requerimiento es una característica de diseño, una propiedad o un comportamiento de un sistema. Estos constituyen la descripción de los deseos o de las necesidades de un producto. Se pueden clasificar en requerimientos funcionales y no funcionales.

- Requerimientos funcionales: son capacidades o condiciones que el sistema debe cumplir.
- Requerimientos no funcionales: son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

2.7.1 Requisitos funcionales.

A continuación se presentan los requisitos funcionales del sistema y una breve descripción de los mismos para su mejor comprensión:

RF1. **Configurar percepción visual:** define la posibilidad de configurar los niveles de percepción visual implementados.

RF2. **Actualizar percepción visual:** define la posibilidad de mantener actualizada la información que reciben los algoritmos.

RF3. **Obtener datos del modelo cognitivo:** se encarga de capturar la información guardada en el modelo cognitivo.

RF4. **Brindar un nivel básico de percepción visual:** se realiza la percepción del entorno basándose en un radio a partir del agente.

RF5. **Brindar un nivel medio de percepción visual:** define un área de visión de dos dimensiones (2D) sobre los planos X-Z del sistema de coordenadas globales.

RF6. **Brindar un nivel avanzado de percepción visual:** se establece un cono de visión al agente en tres dimensiones (3D).

2.7.2 Requisitos no funcionales.

Restricciones en el diseño y la implementación:

Se utilizará el lenguaje de programación C++ estándar bajo el paradigma de programación Orientado a Objetos.

Requerimientos de Usabilidad:

Para los niveles apropiados de usabilidad se requiere de un personal familiarizado con el lenguaje de programación C++ y con conocimiento básico sobre el Sistema de Percepción. Todo será implementado con terminologías del idioma Inglés.

Requerimientos de soporte:

El sistema que se desarrollará deberá tener compatibilidad con el sistema operativo Windows.

Rendimiento:

Como aplicación de tiempo real, debe tener alto grado de velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación, y disponibilidad.

2.8 Modelo de Casos de Uso del Sistema.

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Por lo tanto, establece un acuerdo entre clientes y desarrolladores sobre las condiciones (requisitos) que debe cumplir el sistema.

2.8.1 Actores del sistema.

Un actor es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Un actor caracteriza las interacciones que los usuarios

exteriores pueden tener con el sistema, puede representar el rol que juega una o varias personas, un equipo o un sistema automatizado.

En el caso particular de esta investigación quien realizará las invocaciones a los casos de uso del sistema será la misma aplicación, por lo tanto el actor del sistema será llamado Aplicación.

Actor	Descripción
Aplicación	Es el encargado de inicializar el sistema permitiendo que sean ejecutadas las funcionalidades: Configurar percepción visual, Actualizar percepción visual y Obtener los datos.

Tabla 2.1 Actor del sistema.

2.8.2 Diagrama de CU del sistema.

El diagrama de CU del sistema define las relaciones entre los actores y los diferentes CU existentes. Para la realización de esta investigación fueron definidas un conjunto de acciones que deben ser ejecutadas por el actor del sistema y que desencadenan un conjunto de operaciones. Las interrelaciones entre las acciones y el actor del sistema son agrupadas en el diagrama de CU del sistema que se muestra a continuación.

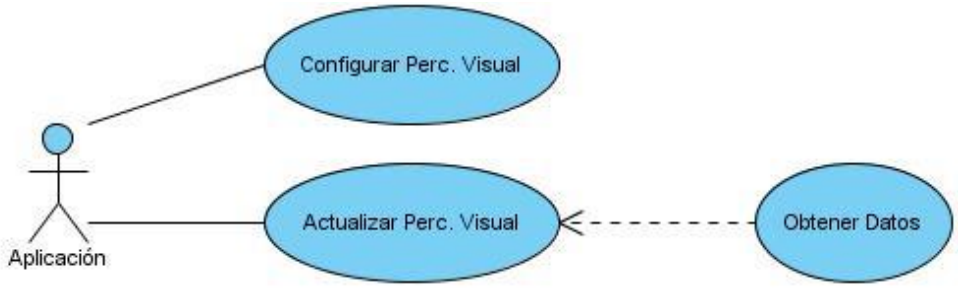

Figura 2.7 Diagrama de CU del sistema.

2.8.3 Descripción de los CU del sistema.

Mediante la descripción detallada de los casos de uso se describe paso a paso la secuencia de eventos que los actores utilizan para completar un proceso usando el sistema. Cada caso de uso tiene una descripción de la funcionalidad que se construirá en el sistema propuesto, las tablas presentadas a continuación forman parte de esta descripción detallada, donde se argumentan con mayor profundidad los flujos operacionales de cada caso de uso.

Caso de Uso: Configurar percepción visual.

CU 1	Configurar percepción visual.
Actores	Aplicación
Resumen	Se inicia cuando el programador selecciona cual nivel de percepción visual va a utilizar en su aplicación.
Precondiciones	
Referencias	RF1
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- Selecciona uno de los niveles de percepción visual que brinda el sistema.	2- Manda a ejecutar el algoritmo acorde a la selección hecha ya sea básico, medio o avanzado.
Poscondiciones	Queda configurado el sistema para que utilice la selección que hizo el programador.

Tabla 2.2 CU expandidos: Configurar percepción visual.

Caso de Uso: Actualizar percepción visual.

CU 2	Actualizar percepción visual.
Actores	Aplicación
Resumen	Se ejecuta constantemente mientas este activa la aplicación y se encarga de mantener actualizada la información que reciben los algoritmos de percepción visual.
Precondiciones	Que el modelo cognitivo que se esté utilizando contenga la información correcta de los elementos presentes en la escena.
Referencias	RF2
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1- Manda a actualizar los agentes	2 - Manda a actualizar los sensores 2.1 - El sensor de Beacon inicializa el caso de uso Obtener Datos. 2.2 - Una vez obtenidos los datos ejecuta el algoritmo de percepción visual anteriormente seleccionado. 2.3 - Pasa los datos percibidos al evento *onVisualPerceptionRecieved*().
Poscondiciones	Los datos percibidos quedan a disposición del agente.

Tabla 2.3 CU extendido: Actualizar percepción visual.

Caso de Uso: Obtener Datos.

CU 3	Obtener Datos.
Actores	Aplicación
Resumen	Se inicia cuando el CU Actualizar percepción visual mando a obtener los datos almacenados en los modelos cognitivos.
Precondiciones	El CU Actualizar percepción visual esté inicializado.
Referencias	RF2, RF3
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El sensor le pide al Sistema de Percepción los datos del modelo cognitivo.	2 -Ordena al *BeaconDataGatherer* a obtener los datos. 2.1- El *BeaconDataGatherer* le pide los datos al modelo cognitivo que se esté usando.

	2.2 - El modelo cognitivo devuelve los datos sensor.
Poscondiciones	El sensor tiene conocimiento de los datos almacenados en el modelo cognitivo.

Tabla 2.4 CU extendido: Obtener Datos.

2.9 Conclusiones del capítulo.

En el capítulo que aquí concluye se expuso la solución propuesta, sentando las bases sobre las cuales se podrá iniciar la implementación del sistema haciendo uso de la metodología y las herramientas seleccionadas. Se escogió como metodología *RUP* debido a que es aplicable a proyectos a largo plazo y es capaz de adaptarse a las características y complejidad de cualquier proyecto de *software*. Se decide usar *Visual Paradigm* como herramienta de modelado porque es multiplataforma y es considerado un producto de calidad. Se utiliza UML como lenguaje de modelado y como lenguaje de programación C++ estándar ya que es el que se utiliza en las aplicaciones de Realidad Virtual.

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN

Introducción.

La primera parte del capítulo encierra el diagrama de clase del sistema propuesto como resultado del refinamiento de las etapas anteriores. Posteriormente se muestran los diagramas de secuencia elaborados a partir de los casos de uso que intervienen en el desarrollo del módulo. Se presentan además los componentes físicos, que se traducen en los ficheros .h y .cpp correspondientes a la implementación en C++. Además se elabora el diagrama de despliegue del sistema.

3.1 Modelo del diseño.

En este epígrafe se mostrarán los diagramas de clases de diseño del sistema propuesto como resultado del refinamiento de etapas anteriores. Se presentan los diagramas de secuencia a partir de la descripción de los casos de uso del sistema que intervendrán en el primer ciclo de desarrollo del proyecto. En el diseño se modela el sistema y se confecciona su estructura (arquitectura), que sirve de soporte para la realización de todos los requisitos. La realización de los casos de usos del diseño contiene una descripción de los flujos de eventos textuales, diagramas de clases y diagramas de interacción. Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. (13)

3.1.1 Diagrama de clases del diseño.

Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como las relaciones existentes entre las mismas. Una clase del diseño es una abstracción sin costuras con una clase o construcción similar en la implementación del sistema donde se le atribuyen visibilidad a sus atributos y operaciones. (13)

A continuación se muestran los diagramas de clases del diseño del Sistema de Percepción (Figura 3.2) y de las interfaces y sensores del sistema (Figura 3.1) con sus relaciones, métodos y atributos utilizados en la implementación del módulo.

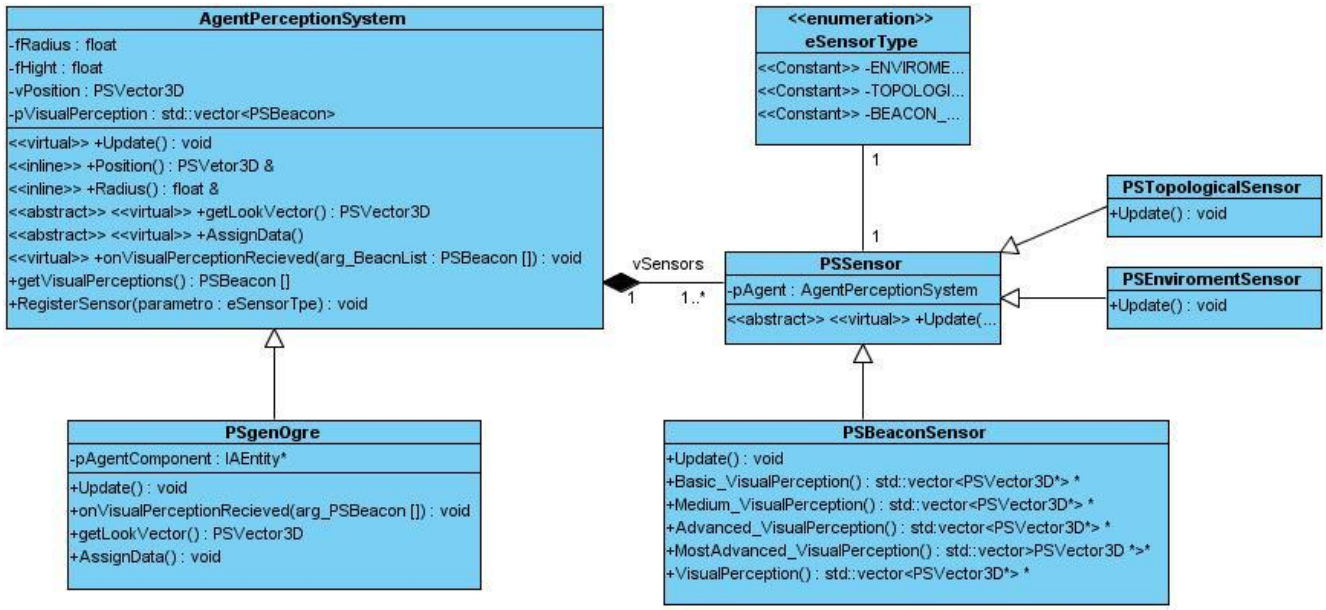

Figura 3.1 Diagrama de clases del diseño de Interfaces y sensores.

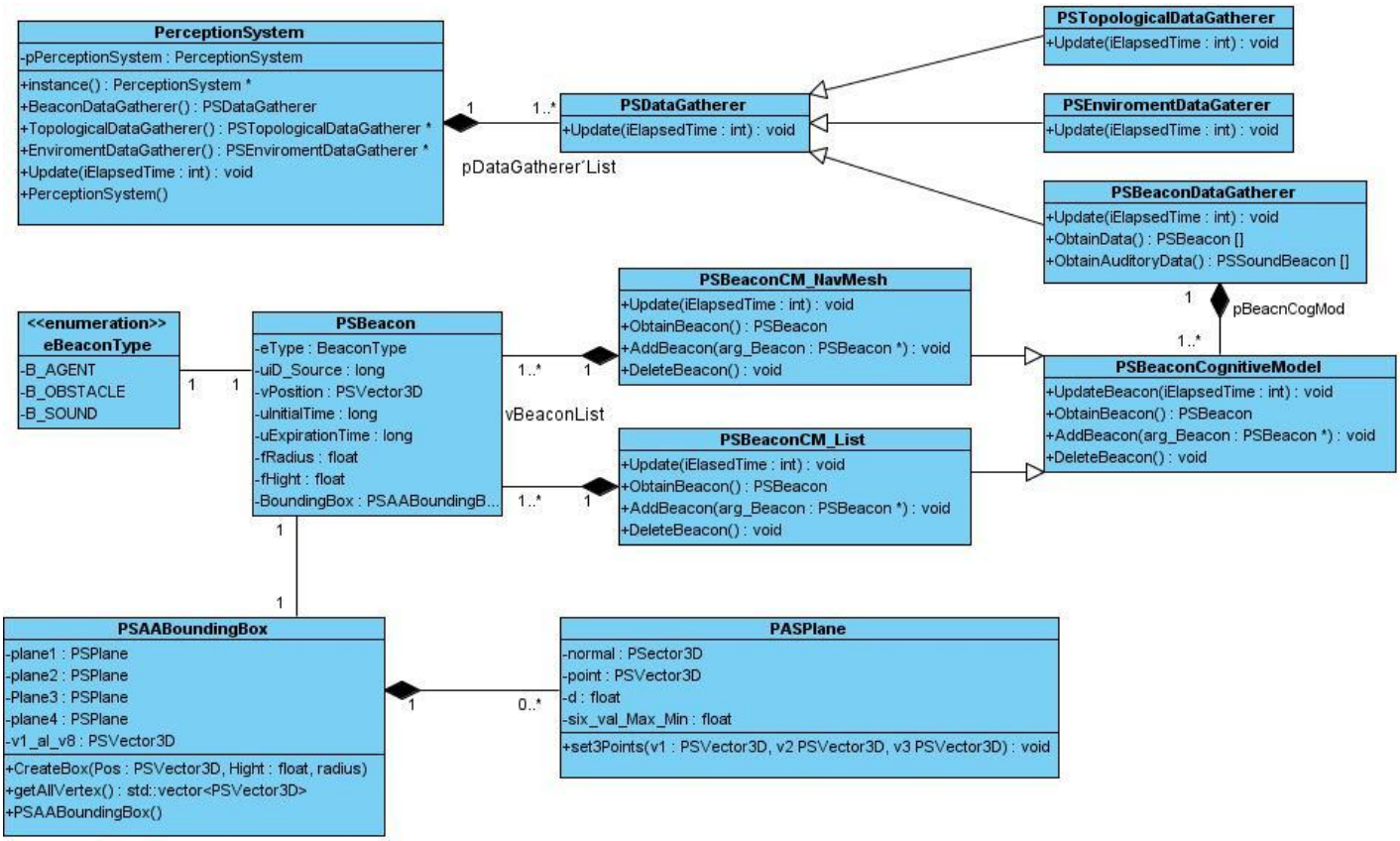

Figura 3.2 Diagrama de clases del diseño del Sistema de Percepción.

3.1.2 Diagramas de Secuencia del diseño.

A continuación se encuentran los diagramas de secuencia del diseño para cada uno de los distintos CU y clases existentes en dicho sistema; de forma tal que se facilite la comprensión de las relaciones entre los mismos. (Figuras 3.3, 3.4 y 3.5)

CU Configurar percepción visual.

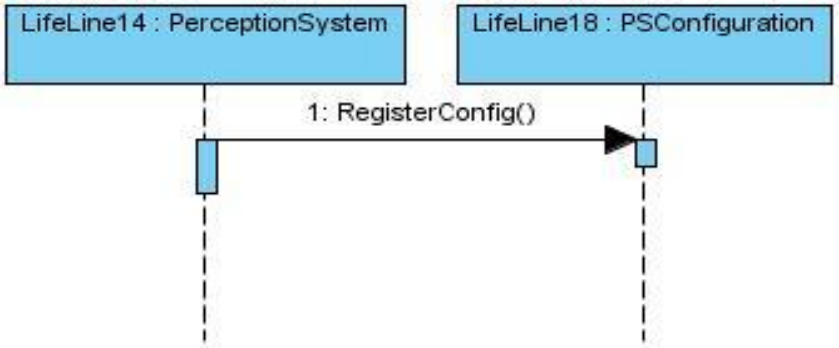

Figura 3.3 Diagrama de secuencia CU Configurar percepción visual.

CU Actualizar percepción visual.

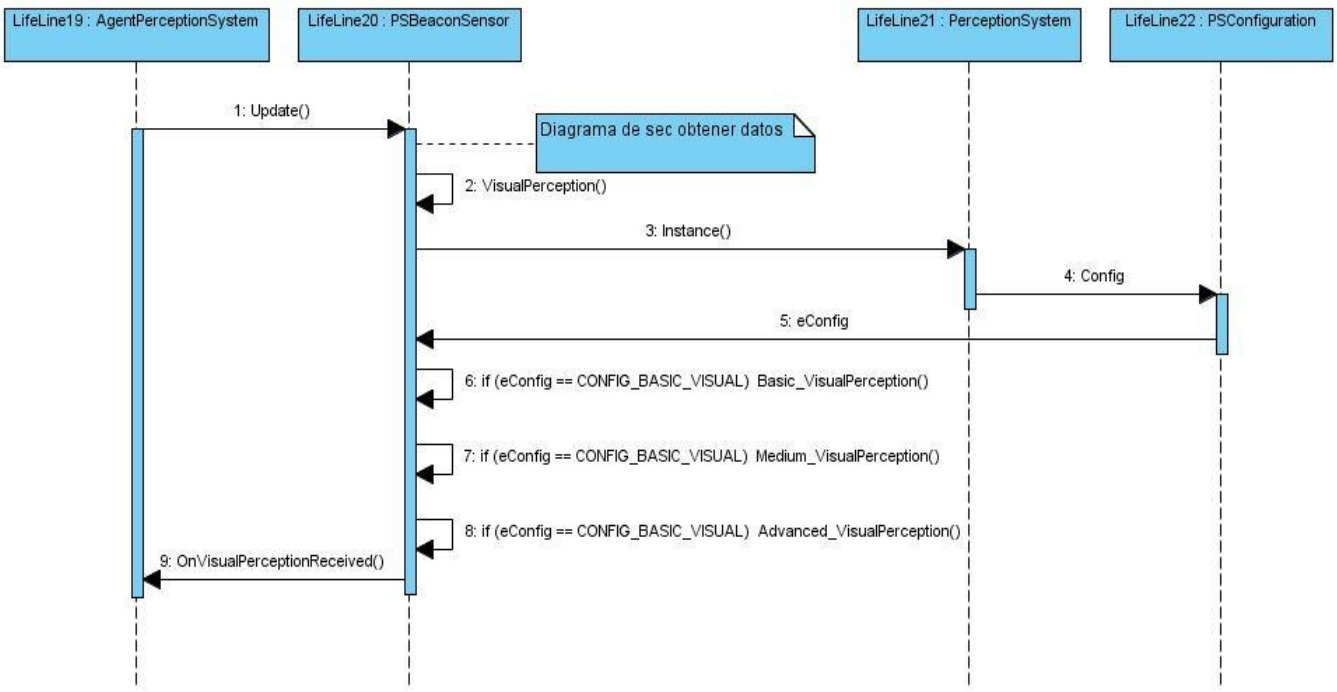

Figura 3.4 Diagrama de secuencia Actualizar percepción visual.

CU Obtener Datos.

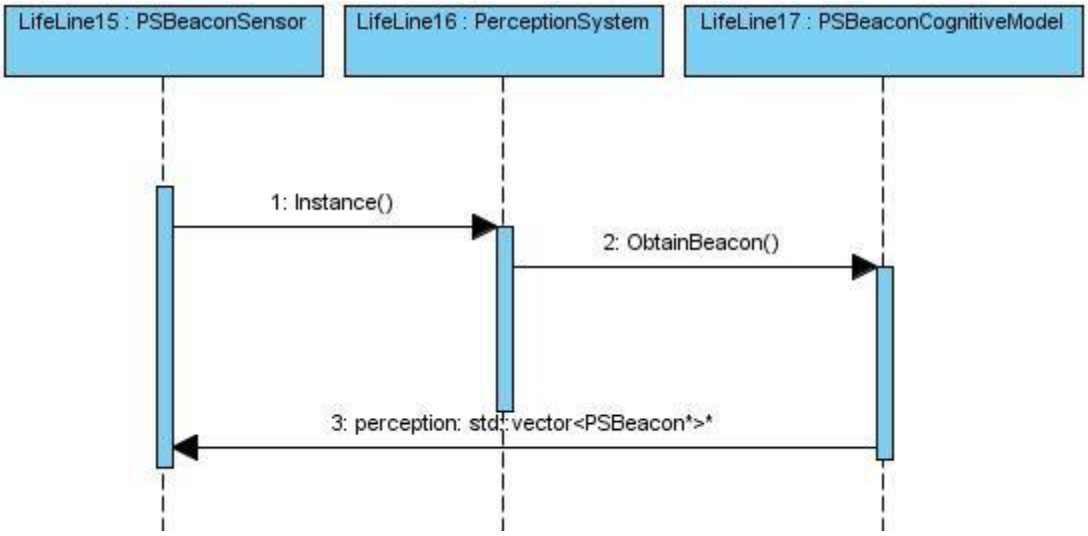

Figura 3.5 Diagrama de secuencia CU Obtener Datos.

3.2 Modelo de implementación.

El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

Los diagramas de componentes y de despliegue, que son artefactos generados en este flujo de trabajo, conforman lo que se conoce como un modelo de implementación, al describir los componentes a construir y su organización; así como su dependencia entre nodos físicos en los que funcionará la aplicación, respectivamente. (13)

3.2.1 Diagrama de componentes.

El diseño de las clases del sistema constituye el paso fundamental para crear los componentes físicos del sistema. Los cuales se traducen en el Sistema de Percepción a desarrollar en ficheros .h y .cpp correspondientes a la implementación que se desarrolla en C++.

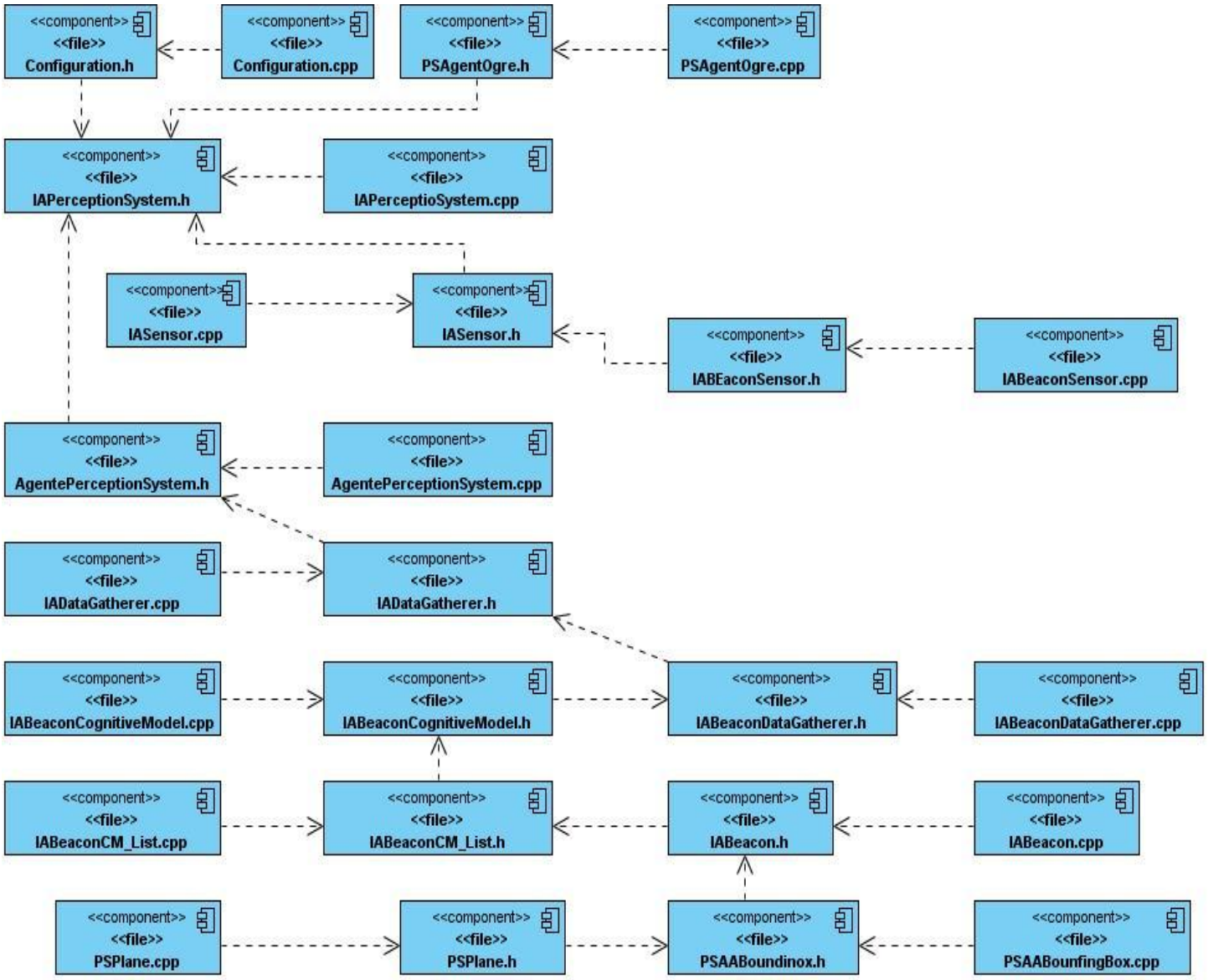

Figura 3.6 Diagrama de componentes.

El componente *IAPerceptionSystem* usa la clase componente *Configuration* para configurar el sistema completo en cuanto al nivel de percepción visual que el desarrollador desee utilizar. Posteriormente los agentes representados por el componente *AgentPerceptionSystem* a través del sensor especializado *PSBeaconSensor* realizan la percepción visual con los datos obtenidos de por los obtenedores de datos *(IADataGatherer*) de los modelos cognitivos.

3.2.2 Diagrama de despliegue.

En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir, se sitúa el *software* en el hardware que lo contiene. Cada hardware se representa como un nodo. Un nodo se representa como un cubo y es un elemento donde se ejecutan los componentes. (13)

Atendiendo a las características del sistema antes planteadas, se realizó el diagrama de despliegue que se muestra a continuación, el cual satisface las necesidades y exigencias de la aplicación.

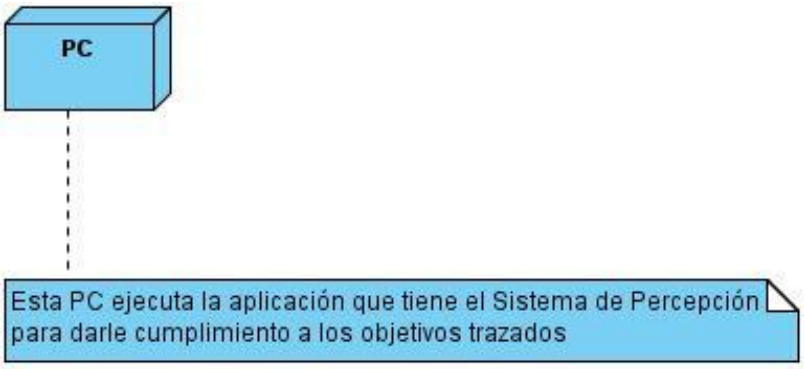

Figura 3.7 Diagrama de despliegue.

3.3 Conclusiones del capítulo.

En el capítulo que concluye se muestran los flujos de trabajo correspondientes al diseño y a la implementación, generando los diagramas de secuencia para cada caso de uso correspondientes a cada uno de los requisitos funcionales que debe cumplir el módulo. También se presenta el diagrama de componentes del sistema y el diagrama de despliegue del mismo.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

Introducción.

En este capítulo se describen las técnicas de pruebas utilizadas para diseñar los casos de pruebas (CP) que se utilizaron en la validación de la solución propuesta. Estos CP se aplican al módulo desarrollado con el fin de descubrir los errores que pueda presentar, con el objetivo de darles solución antes de darlo por terminado, garantizando su correcto funcionamiento.

4.1 Ejecución de las pruebas.

Las pruebas constituyen una etapa imprescindible durante el proceso de desarrollo del *software*, pues permiten detectar y corregir el máximo de errores posibles antes de la entrega al cliente, por lo que el éxito de las mismas puede mejorar la percepción de calidad del usuario final y lograr su satisfacción.

Es necesario trabajar con un procedimiento basado en CU, principal artefacto de la metodología RUP, para llevar a cabo el proceso de pruebas y poder explorar todos los caminos implementados en la aplicación usando un orden lógico; logrando de esta forma, ampliar las probabilidades de detectar errores antes que el sistema se despliegue.

4.2 Técnicas de Pruebas.

Las técnicas de evaluación dinámica o prueba proporcionan distintos criterios para generar casos de prueba que provoquen fallos en los programas. Estas técnicas se agrupan en:

- ***Técnicas de caja blanca o estructurales***: que se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.
- ***Técnicas de caja negra o funcionales***: que realizan pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar.

A primera vista parecería que una prueba de caja blanca completa nos llevaría a disponer de un código perfectamente correcto. De hecho esto ocurriría si se han probado todos los posibles caminos por los que puede pasar el flujo de control de un programa. Sin embargo, para programas

de cierta envergadura, el número de casos de prueba que habría que generar sería excesivo, nótese que el número de caminos incrementa exponencialmente a medida que el número de sentencias condicionales y bucles aumenta. Sin embargo, este tipo de prueba no se desecha como impracticable. Se pueden elegir y ejercitar ciertos caminos representativos de un programa. (17)

Por su parte, tampoco sería factible en una prueba de caja negra probar todas y cada una de las posibles entradas a un programa, por lo que análogamente a como ocurría con las técnicas de caja blanca, se seleccionan un conjunto representativo de entradas y se generan los correspondientes casos de prueba, con el fin de provocar fallos en los programas.

4.2.1 Pruebas de Caja Blanca o Estructurales.

A este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. Este método se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento. (17)

El objetivo de la técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa.

Como se ha indicado ya, puede ser impracticable realizar una prueba exhaustiva de todos los caminos de un programa. Por ello se han definido distintos criterios de cobertura lógica, que permiten decidir qué sentencias o caminos se deben examinar con los casos de prueba. Estos criterios son:

- ***Cobertura de Sentencias*:** Se escriben casos de prueba suficientes para que cada sentencia en el programa se ejecute, al menos, una vez.
- ***Cobertura de Decisión*:** Se escriben casos de prueba suficientes para que cada decisión en el programa se ejecute una vez con resultado verdadero y otra con el falso.
- ***Cobertura de Condiciones*:** Se escriben casos de prueba suficientes para que cada condición en una decisión tenga una vez resultado verdadero y otra falso.
- ***Cobertura Decisión/Condición*:** Se escriben casos de prueba suficientes para que cada condición en una decisión tome todas las posibles salidas, al menos una vez, y cada decisión tome todas las posibles salidas, al menos una vez.
- ***Cobertura de Condición Múltiple*:** Se escriben casos de prueba suficientes para que todas las combinaciones posibles de resultados de cada condición se invoquen al menos una vez.
- ***Cobertura de Caminos*:** Se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Entendiendo camino como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

4.2.1.1 Prueba #1: Algoritmo percepción visual básica.

```
vector <PSBeacon> obstaculos = PerceptionSystem->Lista de obstaculos;          (1)
PSVector3D pos = posición del agente;
PSVector3D look = lookat() del agente;
inicialización de variables;

FOR (obstaculos.length())                                                      (2)
{
        Determinar distancia;                                                  (3)
        IF(dist < maxView)                                                     (4)
                THEN insertar en lista de visibles;                            (5)
        ELSE ignore                                                            (6)
        END IF
}

Retornar lista de visibles;                                                    (7)
```

Figura 4.1 Pseudocódigo del algoritmo de percepción visual básica.

Una vez analizado el código y definidas los nodos se obtiene el siguiente grafo de flujo:

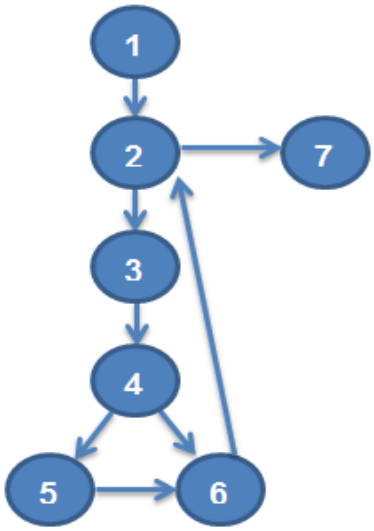

Figura 4.2 Grafo de flujo del algoritmo de percepción visual básica.

La complejidad ciclomática, V(G), de un grafo de flujo G se define como:
V(G) = Aristas - Nodos + 2

V(G) = 8 - 7 + 2 = 3

Por lo tanto el grafo cuenta con tres caminos, los cuales son:

Camino #1: 1, 2, 7

Camino #2: 1, 2, 3, 4, 5, 6, 2, 7

Camino #3: 1, 2, 3, 4, 6, 2, 7

A continuación se realizan los casos de pruebas para cada uno de los caminos definidos anteriormente, teniendo en cuenta las variables involucradas en el proceso.

Número de Camino	Caso de Prueba	Objetivo	Resultado obtenido	Descripción
Camino #1	kLength = 0	Probar el algoritmo cuando la lista de percepciones está vacía.	*VisibleObjects.length*()= 0	Se devuelve una lista de elementos de tipo PS*Beacon* vacía, esto no afecta el funcionamiento del algoritmo.
Camino #2	kLength > 0 distaux = 3050.94 maxView = 1000	Probar el algoritmo de percepción visual básico cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 0 (en la primera iteración)	El elemento no se inserta en la lista de elementos visibles pues su distancia es mayor al rango de visión del agente. Luego se pasa a comprobar el siguiente elemento.
Camino #3	kLength > 0 distaux = 675.28 maxView = 1000	Probar el algoritmo de percepción visual básico cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 1 (en la primera iteración)	El elemento se inserta dentro de la lista de los elementos visibles al agente pues se encuentra dentro de su rango de visión y se pasa a comprobar el siguiente elemento.

Tabla 4.1 Casos de Pruebas del algoritmo de percepción visual básico.

4.2.1.2 Prueba #2: Algoritmo percepción visual media.

```
Vector <PSBEacon> obstaculos = PerceptionSystem-> Lista de objetos;          (1)
PSVector3D pos = posición del agente;
PSVector3D look = lookat() del agente;
inicializacion de variables y proyecciones sobre el plano X-Z

FOR(perceptions.length())                                                    (2)
{
        Determinar distancias
        Determinar ángulos                                                   (3)

        IF(dist<maxView && angulo < 45)                                      (4)

                THEN insertar en lista de visibles                           (5)

                ELSE ignore                                                  (6)
        END IF
}

retornar lista                                                               (7)
```

Figura 4.3 Pseudocódigo del algoritmo de percepción visual media.

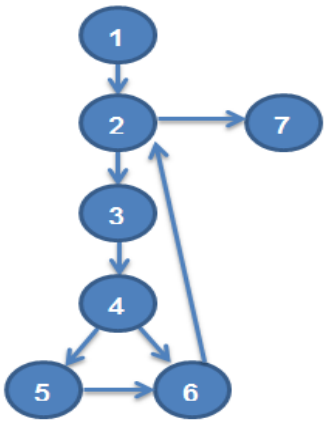

Figura 4.4 Grafo de flujo del algoritmo de percepción visual media.

La complejidad ciclomática, V(G), de un grafo de flujo G se define como:

V(G) = Aristas - Nodos + 2

V(G) = 8 - 7 + 2 = 3

Se pueden identificar tres caminos posibles para el algoritmo de visión media los cuales son:

Camino #1: 1, 2, 7

Camino #2: 1, 2, 3, 4, 5, 6, 2, 7

Camino #3: 1, 2, 3, 4, 6, 2, 7

A continuación se realizan los casos de pruebas para cada uno de los caminos definidos anteriormente, teniendo en cuenta las variables involucradas en el proceso.

Número de Camino	Caso de Prueba	Objetivo	Resultado obtenido	Descripción
Camino #1	kLength = 0	Probar el algoritmo cuando la lista de percepciones está vacía.	*VisibleObjects.length*()= 0	Se devuelve una lista de elementos de tipo PS*Beacon* vacía, esto no afecta el funcionamiento del algoritmo.
Camino #2	kLength = 26 distaux = 658.92 maxView = 1000 angP = 37.11 inside = True	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 1 (en la primera iteración del algoritmo)	El elemento se inserta a la lista de elementos visibles pues tanto su ángulo respecto al agente como su distancia cumplen con los parámetros establecidos para estar dentro del campo visual del agente.
Camino #3	kLength = 26 distaux = 1721.20 maxView = 1000 angP = 37.11 inside = True	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo)	El elemento no se inserta en la lista de elementos visibles aunque su ángulo respecto a su vector posición cumple con los parámetros pues su distancia es mayor al rango de visión del agente. Continúa luego con el segundo elemento de la lista.
Camino #3	kLength = 26 distaux = 765.20 maxView = 1000 angP = 69.23 inside = True	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo)	El elemento no se inserta en la lista de elementos visibles aunque su distancia a la que se encuentra del agente es menor que el rango visón del mismo, su ángulo no cumple con los parámetros establecidos para visión. Continúa luego con el segundo elemento de la lista.
Camino #3	kLength = 26 distaux = 3274.42 maxView = 1000 angP = 52.73 inside = False	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo)	El elemento no se inserta en la lista de elementos visibles pues no se cumple ninguno de los parámetros necesarios para que este dentro del campo visual del agente.

Tabla 4.2 Casos de Pruebas del algoritmo de percepción visual medio.

4.2.1.3 Prueba #3: Algoritmo percepción visual avanzada.

```
Vector <PSBeacon> obstaculos = PerceptionSystem-> lista de objetos;          (1)
PSVector3D pos = posición del agente;
PSVector3D look = lookat() del agente;
FOR(obstaculos.length())                                                     (2)
{
    Creo el Volumen Envolvente;
    Determinar distancias a pos;
    Determinar ángulos a pos;                                                (3)
    bool inside = false;
    IF (dist<maxView && ángulo < 45)                                         (4)
        THEN Insertar en lista de visibles;                                  (5)
             inside == True;
    ELSE IF (inside == false)                                                (6)
        THEN int vertexInside;                                               (7)
             FOR(obstacle.BOX.cantidadVetex())                               (8)
             {
                 Determinar distancias a vertex;
                 Determinar ángulos a vertex;                                (9)
                 IF (dist<maxView && ángulo < 45)                            (10)
                     THEN vertexInside++;                                    (11)
                     ELSE ignorar
                     END IF                                                  (12)
             }
             IF(verrtexInside > 3)                                           (13)
                 THEN Insertar en lista de visibles                          (14)
    END IF                                                                   (15)
}
Retornar lista de visibles                                                   (16)
```

Figura 4.5 Pseudocódigo del algoritmo de percepción visual avanzada.

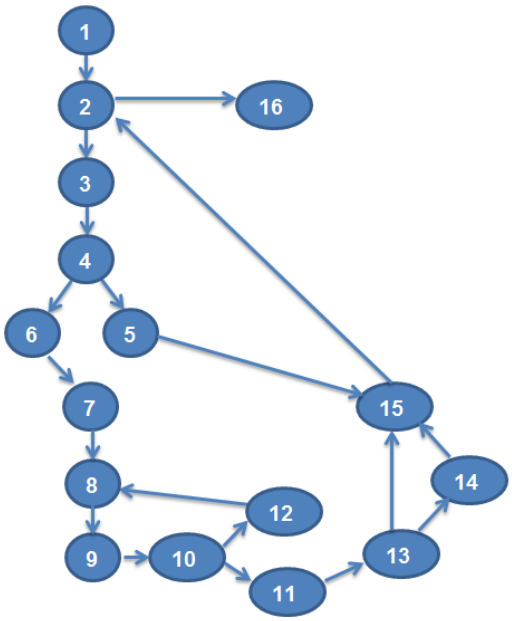

Figura 4.6 Grafo de flujo del algoritmo de percepción visual avanzada.

La complejidad ciclomática, V(G), de un grafo de flujo G se define como:
V(G) = Nodos Predicado + 1

V(G) = 3 + 1 = 4

Derivándose los siguientes caminos:

Camino #1: 1, 2, 16

Camino #2: 1, 2, 3, 4, 5, 15, 2, 16

Camino #3: 1, 2, 3, 4, 6, 7, (8, 9, 10, 12 o 11) x8, 13, 14, 15, 2, 16

Camino #4: 1, 2, 3, 4, 6, 7, (8, 9, 10, 11 o 12) x8, 13, 15, 2, 16

A continuación se realizan los casos de pruebas para cada uno de los caminos definidos anteriormente, teniendo en cuenta las variables involucradas en el proceso.

Número de Camino	Caso de Prueba	Objetivo	Resultado obtenido	Descripción
Camino #1	kLength = 0	Probar el algoritmo cuando la lista de percepciones está vacía.	*VisibleObjects.length*()= 0	Se devuelve una lista de elementos de tipo PSBeacon vacía, esto no afecta el funcionamiento del algoritmo.
Camino #2	kLength = 26 distaux = 658.92 maxView = 1000	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista	*VisibleObjects.length*()= 1 (en la primera iteración del algoritmo)	El elemento se inserta a la lista de elementos visibles pues tanto su ángulo respecto al agente como su distancia cumplen con los

	angP = 37.11 inside = True	de percepciones.		parámetros establecidos para estar dentro del campo visual del agente.
Camino #2	kLength = 26 distaux = 1721.20 maxView = 1000 angP = 37.11 inside = False	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo)	El elemento no se inserta en la lista de elementos visibles aunque su ángulo respecto a su vector posición cumple con los parámetros pues su distancia es mayor al rango de visión del agente. Continúa luego con el segundo elemento de la lista.
Camino #2	kLength = 26 distaux = 765.20 maxView = 1000 angP = 69.23 inside = False	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo).	El elemento no se inserta en la lista de elementos visibles aunque su distancia a la que se encuentra del agente es menor que el rango visón del mismo, su ángulo no cumple con los parámetros establecidos para visión. Continúa luego con el segundo elemento de la lista.
Camino #2	kLength = 26 distaux = 3274.42 maxView = 1000 angP = 52.73 inside = False	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones.	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo).	El elemento no se inserta en la lista de elementos visibles pues no se cumple ninguno de los parámetros necesarios para que este dentro del campo visual del agente.
Camino #3	kLength = 26 distaux = 981.39 maxView = 1000 angP = 49.11 inside = False distV = 932.22 angV = 43.56 vertexInside = 5 (después de las 8 iteraciones)	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones. (comprobaciones de vértices)	*VisibleObjects.length*()= 1 (en la primera iteración del algoritmo)	El elemento se inserta dentro de la lista de los elementos visibles pues por lo menos 3 de los vértices de su volumen envolvente están dentro del campo de visión del agente y se pasa a comprobar el siguiente elemento.
Camino #4	kLength = 26 distaux = 873.15 maxView = 1000 angP = 49.11 inside = False distV = 1108.48 angV = 43.56 vertexInside = 2	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones. (comprobaciones de	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo)	El elemento no se inserta dentro de la lista de los elementos visibles pues el número de vértices de su volumen envolvente que están dentro del campo de visión del agente es menor de 3.

	(después de las 8 iteraciones)	vértices)		Luego se pasa a comprobar el siguiente elemento.
Camino #4	kLength = 26 distaux = 873.15 maxView = 1000 angP = 72.81 inside = False distV = 1108.48 angV = 62.19 vertexInside = 0 (después de las 8 iteraciones)	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones. (comprobaciones de vértices)	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo)	El elemento no se inserta dentro de la lista de los elementos visibles pues el número de vértices de su volumen envolvente que están dentro del campo de visión del agente es menor de 3. Luego se pasa a comprobar el siguiente elemento.
Camino #4	kLength = 26 distaux = 873.15 maxView = 1000 angP = 49.11 inside = False distV = 1108.48 angV = 50.23 vertexInside = 0 (después de las 8 iteraciones)	Probar el algoritmo de percepción visual avanzada cuando existe al menos un elemento en la lista de percepciones. (comprobaciones de vértices)	*VisibleObjects.length*()= 0 (en la primera iteración del algoritmo)	El elemento no se inserta dentro de la lista de los elementos visibles pues el número de vértices de su volumen envolvente que están dentro del campo de visión del agente es menor de 3. Luego se pasa a comprobar el siguiente elemento.

Tabla 4.3 Casos de Pruebas del algoritmo de percepción visual avanzada.

4.2.2 Pruebas de Caja Negra o Funcionales.

También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una “Caja Negra” cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. No obstante, como el estudio de todas las posibles entradas y salidas de un programa sería impracticable se selecciona un conjunto de ellas sobre las que se realizan las pruebas. (17)

4.3 Resultados.

Luego de realizar una exhaustiva búsqueda de la información relacionada con la evolución de los sistemas de percepción, un análisis y desglose de toda la documentación obtenida en componentes y conceptos básicos se obtuvo un resumen que se convertiría en una guía en el proceso de desarrollo. Posteriormente, luego de realizar el diseño y la implementación se obtuvo un módulo de percepción visual que permite a los AI conocer cuáles son los objetos visibles en tiempo real. (Ver Anexos 1 y 2)

En estos momentos este módulo se encuentra integrado al SP que se está desarrollando, cumpliendo así con los objetivos propuestos al principio de la investigación. Este módulo es de fácil configuración para los programadores, los cuales solo deben seleccionar uno de los tres niveles que actualmente se proporcionan y de esta forma equipar a los AI de una percepción de acuerdo a lo que se desee desarrollar.

CONCLUSIONES GENERALES

Luego de valorar cada uno de los resultados obtenidos en los capítulos anteriores, podemos concluir que se obtuvo un módulo de percepción visual, el cual se encuentra ya incorporado al SP ofreciendo funcionalidades de visión a los agentes inteligentes, lo cual era una carencia del sistema. El módulo ofrece los niveles básico, medio y avanzado de percepción, dejando a criterio del programador la selección de uno de ellos, acorde a la aplicación que desee desarrollar. Esto posibilita que los programadores que usen este SP puedan equipar a sus agentes inteligentes con capacidades que les permitan actuar de una forma más realista en la escena virtual. El SP con la nueva arquitectura definida para dar solución al problema planteado, se probó en el proyecto Aduana que se está desarrollando en el CEDIN comprobándose la completa funcionalidad de los requisitos definidos, quedando disponible para su utilización en cualquier proyecto futuro.

RECOMENDACIONES

- Implementar nuevos niveles de complejidad al módulo de percepción visual pero siempre teniendo en cuenta el consumo de recursos de los mismos ya que es un factor vital para el Sistema de Percepción.

BIBLIOGRAFÍA

1. **Herrera, Pedro J., de Blas, Pablo M. y Rubio, Ignacio.** *Sistema Multiagente con capacidad de Percepción Visual y toma de decisiones en un entorno 3D.* 2008.

2. **Ruiz, Alberto.** *Sistemas de Percepción, Visión por Computador.* 2007.

3. http://en.scientificcommons.org/22171835. [En línea]

4. http://www.lukor.com/not-neg/empresas/0505/24121830.htm. [En línea]

5. http://grvc.us.es/rar/mainFrame/percepcion/percepcion.html. [En línea]

6. http://www.visualtraining.com/esp/lavision.htm. [En línea]

7. **Herrero, Pilar y de Antonio, Angélica.** *A human based perception model for cooperative intelligent virtual agents.* 2003.

8. http://www.fuzzygamedev.com/2007/04/frustum-culling/3/. [En línea]

9. http://rastergrid.com/blog/tag/occlusion-culling/. [En línea]

10. *Curso Técnicas de aceleración de Raytracing.*

11. *Proceso de Desarrollo y Gestión de Proyectos de Software (1ra versión).* Ciudad Habana : s.n., 2009.

12. **Jacobson, I. y Booch, G.** *El proceso unificado de desarrollo de software.* 2000.

13. **Enriquez, Barrientos.** *El proceso Unificado de Modelado (RUP).* 2005.

14. **Rumbaugh, J. y Jacobson, I.** *El Lenguaje Unificado de Modelado. Manual de Referencia. Addison Wesley.* 2000.

15. **Rossi, G. y Moreira, A.** *UML: el lenguaje estándar para el modelado del software.*

16. **Juristo, Natalia, Moreno, Ana M. y Vegas, Sira.** *TÉCNICAS DE EVALUACIÓN DE SOFTWARE.*

17. **Pontevia, Pierre.** *Open de Eyes of your Non Player Characters.*

18. http://grvc.us.es/rar/mainFrame/percepcion/percepcion.html. [En línea]

19. **Funge, Jhon David.** *Artificia Intelligence for Computer Games.* 2004.

20. **J., Esmitt Ramírez.** *Frustum Culling fundamentos en computación gráfica.* 2009.

21. **Ulf Assarsson, Tomas Akenine-Möller.** *Occlusion Culling and Z-Fail for Soft Shadow Volume Algorithms.*

22. http://www.gamedev.net/reference/programming/features/occlusionculling/page1.asp. [En línea]

23. *Game Programming Gems 1.*

24. **Placeres, Frank Puig.** *ADVANCED PERCEPTION SYSTEM FOR GAMES.*

25. **Pedro Javier Herrera Caro, Pablo Martín de Blas Sánchez, Ignacio Rubio Díaz.** *Sistema multiagente con capacidad de percepción visual y toma de decisiones en un entorno 3D.* 2008.

26. **Panadero, M. Carmen Fernández y Román, Julio Villena.** *Pruebas de Programas.*

27. http://dotests.blogspot.com/2010/06/pruebas-caja-blancatecnica-coberturas.html. [En línea]

28. **Presman, Roger S.** *Ingeniería de software un enfoque practico.* 2002.

GLOSARIO DE TÉRMINOS

Entorno virtual: Está formado por la representación de todos los elementos de la escena con los que interactúan los agentes inteligentes, estos elementos pueden ser de diferentes tipos como obstáculos, paredes, otros agentes, etc.

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

Agente inteligente: es una entidad capaz de percibir su entorno, procesar tales percepciones y responder o actuar en su entorno de manera racional, es decir, de manera correcta y tendiendo a maximizar un resultado esperado.

Percepción: Proceso nervioso superior que permite al organismo, a través de los sentidos, recibir, elaborar e interpretar la información proveniente de su entorno.

Realidad Virtual: Es una representación de las cosas a través de medios electrónicos, que nos da la sensación de estar en una situación real en la que podemos interactuar con lo que nos rodea.

Beacon: Puede ser cualquier estímulo, en un juego, que requiera de un agente para responder. En un entorno determinado los Beacon podrían ser cualquier cosa audible o visible, que afecta el comportamiento de los agentes, tales como: disparos de armas de fuego, explosiones, enemigos cercanos, ruidos de las puertas al cerrarse o los cadáveres, etc.

Campo Visual: es el área total en la cual un objeto puede ser visto en la visión periférica mientras el ojo está enfocado en un punto central.

Sensor: es el dispositivo encargado de captar la información del entorno virtual, esta información puede ser de tipo topológica, ambiental o de tipo Beacon.

ANEXOS

Anexo 1: Proyecto Entrenador Aduanero.

Figura A.1 Entrenador Aduanero.

Figura A.2 Entrenador Aduanero.