Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Título: Ingeniería Inversa a Biblioteca de Inteligencia Artificial para videojuegos.

Autores: Aliuchy Cangas Gómez.
Reinier Pulido Prieto.

Tutor: Ing. Yidier Romero Zaldivar.

Ciudad de la Habana, Junio de 2009

"Año del 50 Aniversario del Triunfo de la Revolución"

"El futuro de Cuba tiene que ser necesariamente un futuro de hombres de ciencia, de hombres de pensamiento".

Fidel Castro.

DATOS DE CONTACTO:

Ing. Yidier Romero Zaldivar (yromero@uci.cu).

Profesor instructor, graduado de Ingeniero en Ciencias Informáticas en el 2006, 3 años de experiencia impartiendo la asignatura de Inteligencia Artificial en la Facultad 5.

AGRADECIMIENTOS

A nuestras familias, por confiar siempre en nosotros, sin la ayuda de ellos no hubiese sido posible la realización de este trabajo.

A nuestro tutor, por dedicarnos su tiempo y ayudarnos mucho.

A nuestros compañeros de apartamento y de grupo, por su buen compañía.

A Jorgito, que ha sido un buen amigo durante estos 5 años.

A Yesnier, por su gran ayuda.

A nuestras profes Dayani y Yadira.

A Arturo, por ser un buen amigo.

A nuestros amigos que siempre han estado ahí, para apoyarnos.

Y a todas aquellas personas que de una forma u otra nos han ayudado a salir adelante.

A todos ellos muchas gracias.

Aly y Rey.

DEDICATORIA

A tata, por estar siempre a mi lado en cualquier momento de la vida, por quererme mucho, por ser mi mamá.

A mami y a papi, que han sido como mis padres, por ser mi mayor ejemplo y ser mis abuelos más queridos.

A Dayron, por ser mi primito más malcriado, por ser un hermano para mí.

A tía, por estar a mi lado cuando la necesito.

A mi novio, por estar siempre a mi lado y quererme mucho.

A la familia de Reinier, por ser tan buenos conmigo.

Aliuchy

A mami y a papi, por darme la vida.

A Laury, por ser mi hermanita mas malcriada y mi tesoro.

A mima, a papa y abuela por ser más que unos súper abuelos.

A mis tíos, por siempre estar cuando los necesito.

A mis primos, por quererme.

A mi novia, por soportarme todos estos años de universidad y quererme mucho.

A la familia de Aliuchy, por ser tan buenos conmigo y en especial a la gorda.

A Yulima, por ser como una hermana.

Y a ese sobrinito que está en camino.

Reinier

RESUMEN

Con el avance de las tecnologías se ha incrementado en gran medida el desarrollo de los juegos en el mercado, este es uno de los campos que más éxito ha tenido en la actualidad, hacerlos más atractivos e interesantes es un reto que se proponen los desarrolladores. Para esto se utilizan diferentes técnicas de Inteligencia Artificial (IA) para que de esa forma los elementos del juego tengan comportamientos inteligentes, con el objetivo de acercarse más a lo que sería un razonamiento humano.

Este trabajo tiene como objetivo realizar la Ingeniería Inversa de la Biblioteca de Inteligencia Artificial que se encuentra desarrollada en la Facultad 5 de la Universidad de las Ciencias Informáticas, con el objetivo de que pueda ser utilizada en el desarrollo de videojuegos.

Se utiliza como herramienta Case Rational Rose Enterprise Suite que utiliza el lenguaje UML como representación visual, siguiendo la metodología Rational Unified Process (RUP), además se hace uso de una herramienta llamada Doxygen para la generación de la documentación.

Como resultado final se obtuvo una descripción detallada de cada una de las clases de la Biblioteca, así como la documentación del código lo cual permite una fácil utilización y comprensión por parte de los desarrolladores de videojuegos de la Facultad 5.

Palabras Claves:

Ingeniería Inversa, Inteligencia Artificial, Biblioteca.

INDICE

INDICE DE FIGURAS

INTRODUCCION

A medida que ha ido avanzando la ciencia y la tecnología el hombre ha tenido entre sus principales objetivos, lograr que las máquinas piensen igual que el ser humano, o al menos tratar de lograr tal situación. En busca de nuevos métodos de aprendizaje para alcanzar tal comportamiento surgió una nueva rama de la Ciencia de la Computación, la Inteligencia Artificial.

Este campo de la Computación es un reto científico puesto que trata de comprender y replicar la inteligencia humana, además posee un gran interés en el campo científico ya que proporciona un conjunto de técnicas, herramientas y métodos que han demostrado su aplicabilidad.

Ha surgido con numerosas aplicaciones en muchos campos, desde áreas de propósito general como la percepción o el razonamiento, hasta áreas específicas como ingeniería del conocimiento, planificación, videojuegos, entre otras. Pero uno de los campos del desarrollo de software que más se ha beneficiado con la Inteligencia Artificial ha sido, sin duda, el desarrollo de videojuegos.

El uso de esta en videojuegos da posibilidad a un conjunto infinito de evaluaciones, cosa que no lo hacía la Investigación Operativa que era la encargada anteriormente. Un ejemplo de estos juegos es el ajedrez, típico juego de estrategia de dos contrincantes en la que no se detiene el juego hasta que se llegue a una meta final: Un jugador le dé Jaque Mate al Rey.

Enseñarle a la máquina a que aprenda a tomar una adecuada decisión en cualquier momento del juego no es tan sencillo, debido a la gran cantidad de posibilidades que el ordenador debe evaluar, no solo los suyos sino también los de su contrincante, generando un gran costo computacional. Dos de los factores limitantes al trabajar con Inteligencia Artificial en videojuegos es la capacidad de procesamiento y memoria disponibles, por lo que es importante considerar las plataformas para las cuales serán programados.

Muchas empresas de software han sacado al mercado nuevos videojuegos tanto para consolas de última generación, como para computadoras, los que a nivel gráfico son increíbles, pero el modo del juego, o sea la tecnología que rige el comportamiento de sus jugadores no ha estado sujeto a muchos cambios. La Inteligencia Artificial constituye todo un potencial por descubrir que podría revolucionar los videojuegos de cualquier género.

Hasta hace poco, en los videojuegos cualquier acción que el jugador realiza tiene que haberla pensado el desarrollador, el software no sería capaz de resolver una situación no pensada y programada previamente. Sin embargo, un videojuego basado en una Inteligencia Artificial más sofisticada podría dar soluciones para las decisiones que tome el jugador en un momento determinado.

La prioridad de los programadores en la nueva generación de videojuegos no serán los gráficos, sino la Inteligencia Artificial.

Según Ray Maguire, alto cargo de Sony Computer Entertainment en el Reino Unido. *"Ya no estamos interesados en los gráficos por sí mismos, ya que los chips gráficos pueden hacer eso por nosotros, sino que los procesadores centrales de las nuevas consolas pueden hacer los juegos más interesantes centrándose en la inteligencia artificial"* **[1]**.

En la Facultad 5 de la Universidad de la Ciencias Informáticas han surgido varios proyectos de Investigación y Desarrollo (I+D) que se dedican al desarrollo de videojuegos, simuladores o paseos virtuales. En el diseño de este tipo de aplicación se deben insertar técnicas de Inteligencia Artificial para lograr que los elementos del entorno virtual tomen sus propias decisiones y su comportamiento no sea mecánico o monótono.

Se conoce la existencia de módulos desarrollados que ayudan a implementar de alguna manera la toma de decisiones o el aprendizaje de un agente virtual, también existen otros muy completos pero están enfocados a una técnica muy específica o a resolver un problema muy particular, y se dan casos en que la documentación no es lo suficientemente explícita. Las herramientas más completas se encuentran registradas bajo licencias copyright, estas son muy profesionales pero son propietarias y en la mayoría de los casos lo que proponen es que la empresa que registra la herramienta sea contratada para realizar la parte correspondiente a la Inteligencia Artificial en el videojuego a desarrollar.

En la Facultad 5 existe una Biblioteca que se realizó con el objetivo de ser usada en el juego Laberinto del Saber **(Ver Anexo 1)**, esta soporta todo lo necesario para implementar la Inteligencia Artificial de un videojuego, pero no puede estar al alcance de desarrolladores de videojuegos ya que no cuenta con una documentación explícita para ser utilizada y entendida, los cuales se enfrentarán con esta problemática cada vez que pretendan desarrollar una aplicación de este tipo.

Esto explica el por qué la necesidad de realizar la documentación de esa Biblioteca existente para poder ser utilizada por los proyectos de videojuegos de la Facultad 5.

Por lo antes expuesto se identificó como **problema científico**: La inexistencia de la documentación de la Biblioteca de Inteligencia Artificial desarrollada en la Facultad 5 para los proyectos de videojuegos.

Se define como **objeto de estudio:** Bibliotecas de IA para el desarrollo de videojuegos.

El **campo de acción** en el cual se enmarca este trabajo es: Biblioteca de Inteligencia Artificial para videojuegos de la Facultad 5.

Como **objetivo general,** este trabajo persigue conformar la documentación de la Biblioteca de Inteligencia Artificial desarrollada en la Facultad 5 haciendo uso de la metodología más adecuada para ser utilizada en los proyectos de videojuegos.

Para darle cumplimiento a nuestro objetivo general se definieron varias **tareas investigativas:**

- Análisis de las técnicas de Inteligencia Artificial que más se utilizan en la actualidad en el desarrollo de videojuegos para una mayor comprensión de las mismas.
- Análisis de las Bibliotecas que tienen implementadas técnicas de Inteligencia Artificial para determinar elementos comunes en su diseño.
- Definición de la metodología y herramientas a utilizar para realizar la Ingeniería Inversa de la Biblioteca.
- Análisis del código fuente para generar la documentación correspondiente al mismo.
- Realización de la Ingeniería Inversa de la Biblioteca para que pueda ser utilizada en los videojuegos de la Facultad 5.

Los métodos que se utilizan en nuestro trabajo son:

Métodos teóricos de la investigación.

- ✓ **Analítico – sintético.**

En la investigación se estudia la documentación referente a herramientas de desarrollo de sistemas de Realidad Virtual, en particular se analizan las herramientas que se utilizan para el desarrollo de videojuegos, recopilándose los elementos más importantes relacionados con el campo de acción.

- ✓ **Análisis histórico lógico.**

En la investigación se hace una búsqueda del estado del arte de las herramientas de desarrollo de sistemas de Realidad Virtual en el desarrollo de videojuegos.

El trabajo está estructurado por tres capítulos de la siguiente forma:

Capitulo 1. Fundamentación teórica.

En este capítulo se explica que es la Inteligencia Artificial, se hace mención a los diferentes géneros de videojuegos que existen, se explican las técnicas utilizadas en la Biblioteca, se realiza un estudio de diferentes Bibliotecas que están publicadas las cuales implementan técnicas de Inteligencia Artificial, se hace un análisis de las metodologías de desarrollo de software más usadas en el mundo para el desarrollo de Sistemas Expertos, Sistemas Multi-Agentes y videojuegos, seleccionando así la más adecuada para realizar la Ingeniería Inversa de la Biblioteca. Se explica brevemente que es la Ingeniería Inversa y sus diferentes beneficios al aplicarla. Además, En este capítulo se exponen los conceptos y temas que ayudarán al mejor entendimiento de los capítulos posteriores.

Capitulo 2. Características del sistema.

En este capítulo se representa el diagrama del modelo del dominio explicando los principales conceptos que tienen que ver con la solución, se identifican los requisitos funcionales y no funcionales de los cuales se derivan los casos de usos, presentándose también el diagrama de casos de usos identificándose así los actores que intervienen en estos, y se hace un descripción detallada de cada caso de uso.

Capitulo 3. Diseño e Implementación del sistema.

En este capítulo se hace una explicación detallada de la estructura y el funcionamiento de la Biblioteca, se realiza un análisis de cómo el Agente se beneficia con las diferentes técnicas de Inteligencia Artificial que se encuentran implementadas en dicha Biblioteca, se desarrollan los diagramas de secuencia de acuerdo con cada caso de uso identificado en el capitulo anterior, se realizan los diagramas de paquetes donde estarán agrupadas las clase según las funcionalidades en común, se conforma el diagrama de componentes y se describe detalladamente cada una de las clases con sus atributos y métodos.

CAPITULO 1: FUNDAMENTACION TEORICA.

Introducción:

En este capítulo se explica que es la Inteligencia Artificial, se hace mención a los diferentes géneros de videojuegos que existen, se explican las técnicas utilizadas en la Biblioteca, se realiza un estudio de diferentes Bibliotecas que están publicadas las cuales implementan técnicas de Inteligencia Artificial, se hace un análisis de las metodologías de desarrollo de software más usadas en el mundo para el desarrollo de Sistemas Expertos, Sistemas Multi-Agentes y videojuegos, seleccionando así la más adecuada para realizar la Ingeniería Inversa de la Biblioteca. Se explica brevemente que es la Ingeniería Inversa y sus diferentes beneficios al aplicarla. Además, En este capítulo se exponen los conceptos y temas que ayudarán al mejor entendimiento de los capítulos posteriores.

1.1 Inteligencia Artificial.

Se define la Inteligencia Artificial como aquella inteligencia exhibida por artefactos creados por humanos (es decir, artificial). A menudo se aplica hipotéticamente a los computadores. El nombre también se usa para referirse al campo de la investigación científica que intenta acercarse a la creación de tales sistemas. La Inteligencia Artificial trata de conseguir que los ordenadores simulen en cierta manera la inteligencia humana. Se acude a sus técnicas cuando es necesario incorporar en un sistema informático, conocimiento o características propias del ser humano **[2].**

La Inteligencia Artificial ha tenido gran auge en nuestros días, teniendo aplicabilidad en la informática, la ciencia, la salud y otros campos. Ha sido un gran reto tratar de acercar la mente de una máquina al pensamiento humano, y sin dudar que en un futuro no muy lejano no sepamos distinguir si estamos en presencia de una Inteligencia Artificial o una verdadera mente humana.

1.2 Géneros de Videojuegos.

Entre la gran variedad de videojuegos existentes no es fácil establecer categorías claras, que distingan unos de otros y permitan un análisis desagregado de cada una de ellas.

En la actualidad casi todos los videojuegos incorporan mezclas de distintos formatos y tipos. Hay que decir que cada vez es más habitual que un videojuego contenga elementos de diversos géneros, cosa que hace difícil su clasificación, a pesar de esto es importante tener claro el género del juego que se vaya a desarrollar para saber que técnicas de Inteligencia Artificial poder aplicar a cada uno de ellos.

Algunos de los géneros más usados son **[3]:**

- **Juegos de Aventura:** Se caracterizan por la investigación, exploración, la solución de rompecabezas, la interacción con personajes del videojuego, y un enfoque en el relato en vez de desafíos basados en reflejos.
- **Juegos de Rol (RPG):** En los videojuegos de rol cada jugador interpreta un personaje ficticio, con una serie de características propias que le definen.
- **Juegos de estrategia en tiempo real (RTS):** En este tipo de videojuego no hay turnos sino que el tiempo transcurre de forma continua. En estos suele haber más acción que en los de turnos y suelen tener mejores gráficos ya que es la tendencia actual.
- **Juegos de Disparo (FTPS):** Son un género que engloban un amplio número de subgéneros que tienen la característica común de permitir controlar un personaje que, por norma general, dispone de un arma (mayoritariamente de fuego) que puede ser disparada a voluntad.
- **Juegos de Plataforma:** Se caracterizan por tener que andar, saltar o escalar sobre una serie de plataformas, con enemigos, mientras se recogen objetos para poder completar el juego.

- **Juegos de Deporte:** Este tipo de videojuego simula el campo de deportes tradicional. Este género ha sido popular en toda la historia de los videojuegos y es sumamente competitivo, justo como los verdaderos deportes mundiales.

- **Juegos de Carreras:** Es un videojuego en el que se imitan competencias entre vehículos. Usualmente el objetivo es recorrer cierta distancia o ir de un sitio hacia otro en el menor tiempo posible.

- **Juegos de Lucha:** Se basa en manejar un personaje que pelea, ya sea dando golpes, usando poderes mágicos o aplicando llaves. Recrean combates entre personajes controlados tanto por un jugador como por la computadora.

1.3 Técnicas de Inteligencia Artificial.

A medida que los videojuegos fueron evolucionando, y fueron surgiendo nuevos géneros la necesidad de técnicas de Inteligencia Artificial fue aumentando en gran medida.

Uno de los principales objetivos de un videojuego es ofrecer retos al usuario, esto se logra introduciendo técnicas de Inteligencia Artificial, ya que ellas son las que deciden cuáles son las mejores opciones que pueden tomar los elementos del videojuego a partir de las condiciones del entorno que los rodea. Las técnicas que se utilizan para un videojuego dependen mucho del tipo de videojuego que estemos diseñando.

Existe una gran multitud de técnicas de Inteligencia Artificial que se han desarrollado paralelamente en varios campos a lo largo de los años.

Las técnicas utilizadas en la Biblioteca de Inteligencia Artificial de la Facultad 5 son:

- ✓ Steering Behaviors (SB).
- ✓ Sistema de Percepción.
- ✓ Máquina de Estados Finitos (FSM).

1.3.1 Máquinas de estados finitos (FSM).

Las Máquinas de Estados Finitos (FSM) **[4]**, también conocidas como Autómatas de Estados Finitos (FSA), son modelos de comportamiento de un sistema o un objeto complejo, con un número limitado de modos o condiciones predefinidas, donde existen transiciones de modo.

Las FSMs están compuestas por 4 elementos principales:

- ✓ Estados que definen el comportamiento y pueden producir acciones.
- ✓ Transiciones de estado que son movimientos de un estado a otro.
- ✓ Reglas o condiciones que deben cumplirse para permitir un cambio de estado.
- ✓ Eventos de entrada que son externos o generados internamente, que permiten el lanzamiento de las reglas y permiten las transiciones.

Las FSMs se usan típicamente como un tipo de sistema de control donde el conocimiento está representado en los estados, y las acciones están restringidas por las reglas.

1.3.2 Steering Behaviors.

Es un comportamiento que siguen los agentes autónomos a la hora de tomar decisiones cuando este interactúa con los demás elementos del entorno virtual. A continuación se hace un análisis más detallado de los Steering Behaviors simples **[5]**.

- ➢ **Seek (Seguir).**

Este comportamiento es útil para dirigir un agente en la dirección correcta, esto lo logra ya que retorna una fuerza que lo impulsa hacia el objetivo. Para ello lo que hace es calcular la fuerza necesaria para llegar al objetivo **(Figura1)**.

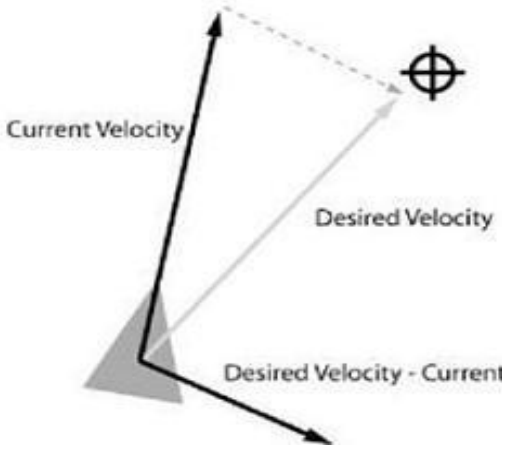


Figura 1: **Representación de fuerzas del *Seek.***

- **Flee (Huir).**

Este *Steering Behaviors* es parecido al *Seek* pero realiza la función opuesta ya que lo que se busca es alejar un agente de una posición determinada. Esto se logra generando una fuerza que dirige un agente hacia la posición contraria a la que se encuentra un objetivo específico.

- **Pursuit (Perseguir).**

Se utiliza cuando un agente busca como interceptar un objetivo en movimiento. El éxito de la función de búsqueda depende de lo bien que el demandante puede predecir la trayectoria del evasor. Esto puede ser muy complicado, por lo que debe hacerse una solución en la que se obtenga el rendimiento adecuado y sin usar demasiados ciclos de reloj **(Figura 2)**.

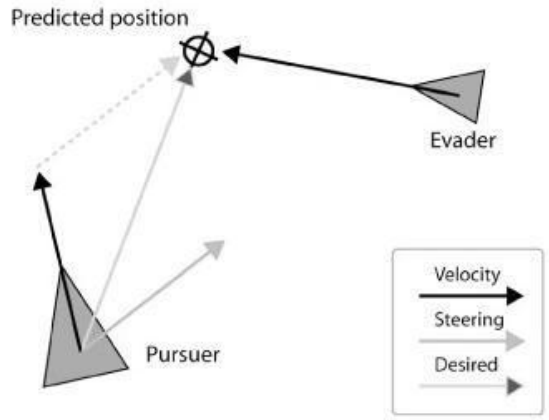


Figura 2: **Representación del comportamiento *Pursuit.***

- **Evade (Evadir).**

Al igual que el *pursuit* el Evade calcula las posiciones futuras del objetivo pero realiza la operación de alejarse de esa posición **(Figura 3)**.

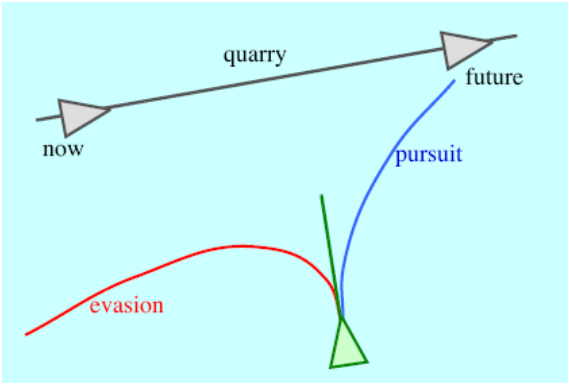


Figura 3: Representación del comportamiento *Pursuit* y *Evade.*

- **Arrive (Arribar).**

El *Seek* resulta útil para lograr que un agente avance en la dirección correcta, pero a menudo se desea que este tenga la capacidad de ir disminuyendo su velocidad según llega a su objetivo. Para ello el *Arrive* calcula el tiempo que demora el agente para llegar al objetivo, y a partir de este valor obtiene la velocidad necesaria en que debe viajar hasta el objetivo **(Figura 4)**.

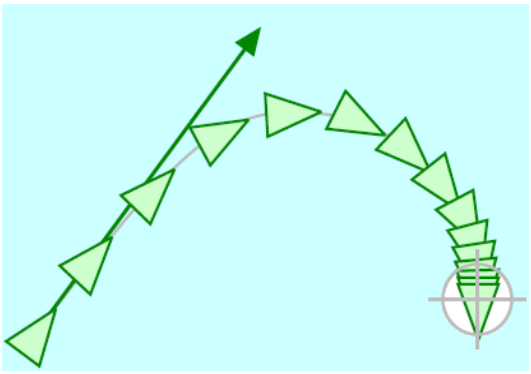


Figura 4: Representación del comportamiento *Arrive.*

- **Wander (Aleatorio).**

Su objetivo es crear una fuerza que le imprima un movimiento aleatorio al vehículo en el entorno virtual. Consiste en proyectar un círculo en el frente del vehículo y un objetivo que se mueve alrededor de su perímetro de modo que el vehículo lo siga. Este objetivo gira en función del tamaño de la circunferencia. Es capaz de imprimirle una sensación de nerviosismo al movimiento **(Figura 5)**.

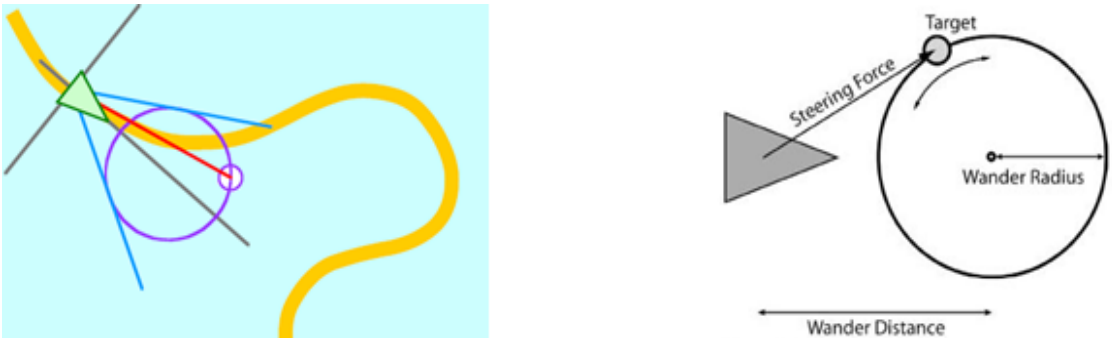


Figura 5: Representación del comportamiento *Wander.*

- **Obstacle Avoidance (Evitar obstáculos).**

Es un comportamiento que dirige un vehículo de modo que evite los obstáculos que yacen en su camino. Un obstáculo es cualquier objeto que pueda encontrarse en su camino. Esto se logra colocando un rectángulo (caja de detección) en la dirección del vehículo, que sea proporcional a la velocidad del mismo, en caso de que colisione con algún obstáculo, cambia su dirección, manteniendo a este libre de colisiones **(Figura 6).**

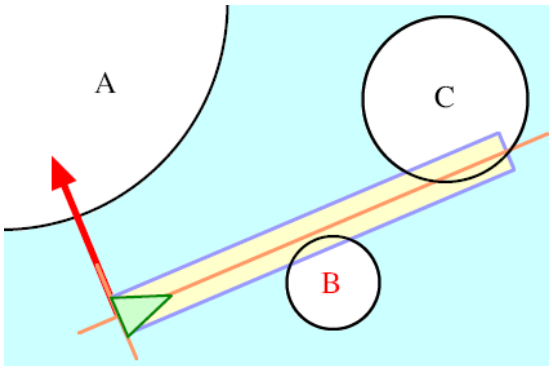


Figura 6: Representación del comportamiento *Obstacle Avoidance.*

- **Wall Avoidance (Evitar muros).**

Tiene como propósito evadir una pared (segmento de línea en 2D) que se encuentre en su camino, esto se logra proyectando tres rayos al frente del agente que son los que interceptan cualquier pared en su camino produciendo una fuerza normal en el punto donde se cruzan, que repele al vehículo logrando que esta sea evitada **(Figura 7)**. Este proceso puede ser efectivo también colocando una caja de detección al frente del vehículo, de forma tal que al colisionar con el segmento de línea cambie su curso en la dirección contraria, este proceso es similar al del *Obstacle Avoidance*.

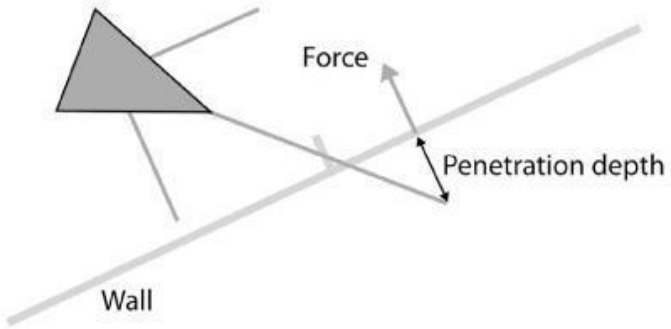


Figura 7: Representación del comportamiento *Wall Avoidance.*

- **Interpose (Interponerse).**

Genera una fuerza que dirige un vehículo al punto medio de una línea imaginaria que separa otros dos vehículos(o puntos en el espacio, o de un agente y un punto). Se comporta similar a un guardia personal que se interpone entre un atacante y su defendido.

El primer paso en este algoritmo es determinar el punto medio de una línea que conecta las posiciones de los agentes en el momento actual y se calcula la distancia. Este valor se divide por la velocidad para determinar el tiempo necesario para recorrerla **(Figura 8)**.

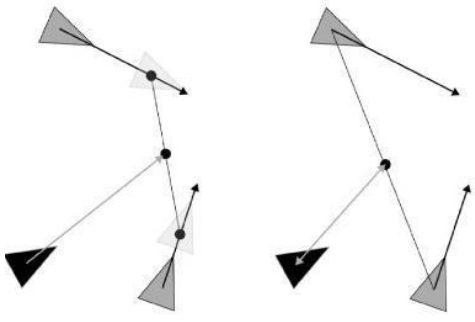


Figura 8: Representación del comportamiento *Interpose.*

- **Hide (Ocultarse).**

Este algoritmo se encarga de dirigir un agente detrás de un obstáculo con respecto a otro agente que lo persigue (cazador), de forma tal que entre ellos siempre exista un obstáculo. Además este comportamiento se puede utilizar no solo en situaciones en las que un agente está huyendo de otro, sino también cuando quiere alcanzar su objetivo (presa) escondiéndose entre los objetos sin que este note su presencia **(Figura 9)**.

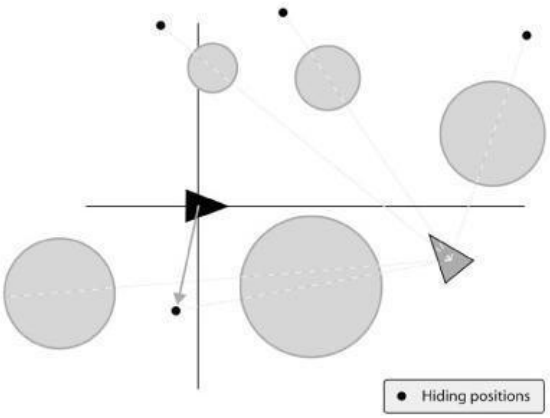


Figura 9: Representación del comportamiento *Hide.*

- **PathFollowing (Seguimiento del camino).**

Un agente con comportamiento *PathFollowing* contiene una lista de puntos que visitar, y crea una fuerza que lo dirige por el orden de la lista a cada uno de los puntos, los puntos visitados quedan descartados **(Figura10)**.

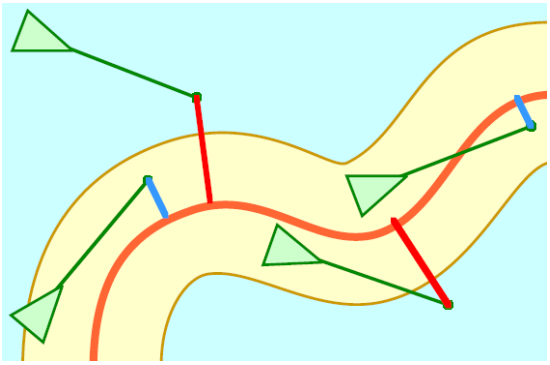


Figura 10: Representación del comportamiento *PathFollowing.*

- **Offset Pursuit (Perseguir a distancia).**

Calcula la fuerza requerida para mantener un vehículo posicionado a una distancia de un objetivo, se utiliza principalmente para construir formaciones **(Figura 11)**.

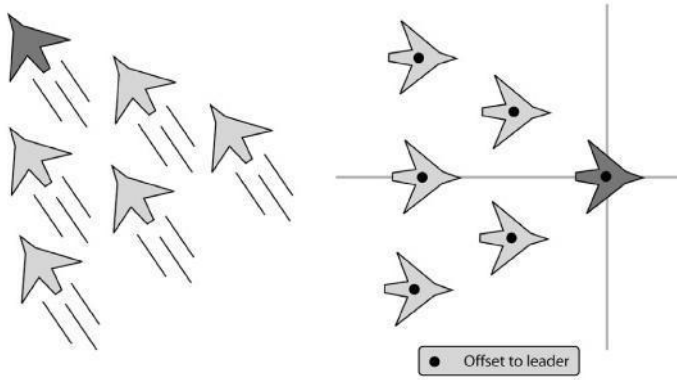


Figura 11: Representación del comportamiento Offset Pursuit.

1.3.3 Percepción.

La percepción se considera como uno de los temas más importantes de la robótica y los videojuegos que son creados bajo la base de la Inteligencia Artificial. Cada videojuego dirige la percepción de manera diferente, se conoce que la percepción más sofisticada puede imitar las limitaciones del mundo real. La percepción es mucho más que ver y oír, abarca todas las maneras en que un NPC (Non Playable Character, un personaje del juego) obtiene la información sobre el mundo, incluyendo la topología del entorno y agentes **[6]**.

La percepción brinda toda la información del entorno virtual a sus agentes, para que su comportamiento sea lo más real posible. Según sea el grado de percepción que tenga un agente virtual será el grado de comportamiento creíble que se desarrolle en un entorno virtual.

En la Biblioteca existen 2 técnicas de percepción definidas: Datos Generados Manualmente y Scripting, pero solo es utilizada la primera.

- **Datos generados manualmente.**

Los datos son generados manualmente con el objetivo de compensar la poca percepción que presentan los agentes inteligentes, en este caso son los desarrolladores los encargados de añadir los datos topológicos de manera manual en un mundo virtual. Un ejemplo simple constituye el caso de realizar una emboscada, se entrarán manualmente datos topológicos como: la posición de escondites, la activación de zonas, etc. Sin embargo estos datos no constituyen una solución aceptable para la percepción limitada de los agentes inteligentes, porque: **[7]**

- ✓ La generación de datos manual no es exhaustiva.
- ✓ La generación de datos manual consume mucho tiempo.
- ✓ Los datos generados pueden no ser compatibles con el motor de juego de colisión y los modelos de circulación.

- **Scripting**

El scripting es otra de las vías que existe en 3D para superar la limitante de percepción que presentan los agentes inteligentes. Conduce a lograr una percepción 3D detallada, teniendo en cuenta todas las posibilidades. El scripting no es el mejor enfoque para compensar la mala percepción, debido a que: **[8]**

- Es tiempo que se consume.
- Interactividad significa que usted no puede prever todo.
- Los escenarios se convierten en lineales.
- Es complejo, script ricos hacen más complejo el proceso de producción.

1.4 Ejemplos de Bibliotecas publicadas.

Cada una de las grandes empresas dedicadas al desarrollo de los videojuegos poseen sus propios módulos de IA para sus productos, estos módulos de IA no están al alcance de las demás empresas o personas interesadas en crear algún videojuego. Existen páginas Web donde se encuentran publicadas Bibliotecas de IA **[9]**, en las mismas se pueden encontrar una o varias de las limitantes mencionadas a continuación las cuales imposibilitan la utilización de ellas, una es la falta de documentación que presentan, otra puede ser que solamente tienen implementada una técnica específica, o que son propietaria que aunque se puede decir que son las más completas, habría que pagar para poder acceder a estas.

Ejemplos:

- El departamento de mecánica, Instituto de Tecnología de Massachussets tiene publicada una Biblioteca (**AGlib**), es una Biblioteca de C++ que implementa varios algoritmos genéticos, incluye herramientas para usar algoritmos genéticos para la optimización de cualquier programa de C++, usando cualquier representación y cualquier operador genético. Esta Biblioteca cuenta con una documentación la cual incluye una extensiva descripción de cómo están implementados los algoritmos genéticos, las interfaces de programación para las clases AGlib, y diferentes ejemplos, todo esto posibilita su uso.

- Free Fuzzy Logic Library (**FFLL**), es una Biblioteca la cual tiene implementada Lógica Difusa, esta no cuenta con la documentación necesaria para ser entendida y utilizada.

- La Biblioteca **GAUL**, es otra que implementa algoritmos genéticos pero para plataforma UNIX, cuenta con documentación.

- **OpenSteer** (Reynolds, 2004) es una Biblioteca para C++ creada por Craig Reinolds en 1986 para desarrollar *Steering Behaviors*, es un modelo para crear el movimiento de agentes autónomos.

Después de haber analizado estas Bibliotecas y establecer una comparación con la desarrollada en la Facultad 5, se puede llegar a la conclusión que esta posee un mayor nivel, no solo por la cantidad de técnicas implementadas, sino, por toda la estructura que tiene, que da la posibilidad de poder incorporar nuevas técnicas de IA, debido a que esta cuenta con un diseño flexible. Estas bibliotecas que se analizan pueden ser insertadas a la Biblioteca aumentando así las funcionalidades de la misma, además que se aprovecha la bibliografía que estas ya poseen. Si se logra fusionar estas Bibliotecas con la existente en la Facultad, y no solo las mencionadas, sino también otras que implementen mas técnicas de IA, se lograría contar con una Biblioteca de IA prácticamente con todo lo necesario para poder ser utilizada en cualquier campo que se necesite.

1.5 Metodologías de Desarrollo.

En el presente epígrafe se analizan las metodologías de desarrollo de software más usadas en el mundo para el desarrollo de los diferentes Sistemas definidos dentro del campo de la Inteligencia Artificial (Sistemas Multi-Agentes (SMA), Sistemas Expertos). Además de la metodología Scrum para el desarrollo de videojuegos.

Para el desarrollo de Sistemas Expertos se propone el análisis de 2 de las metodologías más utilizadas IDEAL y CommonKADS. Para los SMA se han seleccionado las metodologías Vowel Engineering *(*ingeniería de vocales*) y* MAS-CommonKADS según 3 criterios **[10]**.

- ✓ Utilización de diferentes vistas para la especificación del sistema.
- ✓ Incorporar la idea de proceso de desarrollo.
- ✓ Integrar técnicas de ingeniería y teoría de agentes.

Además se tiene en cuenta la metodología RUP debido al prestigio que ha alcanzado mundialmente en el desarrollo de software.

1.5.1 CommonKADS.

CommonKADS **[11]**, es un método que se aplica para el análisis y la construcción de Sistemas Basados en Conocimiento. Este método está orientado hacia la realización de actividades de modelado, donde se desarrollan un conjunto de modelos que permiten expresar diferentes perspectivas de la situación que se está analizando; actividades de administración del proyecto donde a los modelos se le asocian estados con los cuales se lleva a cabo la gestión del proyecto, y actividades de reutilización donde se pretende mejorar la productividad en el desarrollo de los Sistemas Basados en Conocimiento.

Los modelos que conforman el método son: Organización, Tarea, Agente, Conocimiento, Comunicación y Diseño. Esta metodología gira alrededor del modelo de experiencia y está pensada para desarrollar Sistemas Basados en Conocimiento que interactúen con el usuario de manera directa. Además, es posible alcanzar un alto nivel de detalle en la descripción y es consecuente con el proceso de desarrollo de software.

Requiere de mucha experiencia y conocimiento, trabaja en estadios abstractos y no se plantea el llegar a ese nivel de detalle que se requiere, está pensada para desarrollar SBC que interactúan con un usuario humano. No hay una fuente de información que contenga todo lo necesario para su aplicación. No hay un ejemplo completo de la aplicación de la metodología que pueda ser utilizado como guía.

1.5.2 IDEAL.

La metodología IDEAL **[12]** de desarrollo de Sistemas Basados en Conocimiento (SBC) utiliza el ciclo de vida en tres dimensiones conocido como espiral tronco-cónico. La base del cono representa el proceso del desarrollo de un SBCC, es decir las etapas de la metodología. La tercera dimensión (paredes del cono) representa la etapa de mantenimiento. Es importante resaltar la importancia del mantenimiento perfectivo, que representa la adición de nuevos conocimientos durante la vida del SBC.

El eje del cono representa la calidad de nuevos conocimientos, la espiral va de mayor diámetro (más conocimientos) a menor y de abajo a arriba (menor calidad a mayor calidad). Propone el desarrollo de un SBC en 5 fases.

1.5.3 MAS-CommonKADS.

Esta metodología **[13]** es una extensión de la ya conocida CommonKADS, cuyo enfoque principal es la construcción de sistemas expertos. Su extensión orientada a agentes trata de aprovechar las ideas orientadas a objetos que ya tienen una buena base en la experiencia diseñando con su progenitor. Además, esta metodología ha sido la primera en plantear la integración del SMA con un modelo de ciclo de vida de software, más concretamente el espiral dirigido por riesgos. Divide su planteamiento en la definición del sistema empleando 7 modelos diferentes, cada uno de los cuales está basado en una teoría distinta. Estos modelos son: agente, tareas, experiencia, coordinación, comunicación, organización y diseño.

El principal inconveniente de MAS-CommonKADS es que el nivel de detalle alcanzado en la descripción no es realizable sin el apoyo de herramientas de soporte, además de que estas herramientas son propietarias.

1.5.4 Vowel Engineering.

El curioso nombre de esta metodología (Ingeniería de vocales) procede de su propio proceso, que se basa en la conjunción de 4 unidades denominadas mediante vocales: A (de agentes), E (de entorno), I (de interacciones) y O (de organización). Bajo esta subdivisión inicial, se aplican técnicas de modelado diferentes para cada uno de los 4 subgrupos.

Así pues, los agentes pueden ser representados por simples autómatas o por complejos sistemas de conocimientos; las interacciones van desde modelos físicos, propagación de ondas, hasta sistemas basados en actos del habla.

En este modelo, lo que se pretende principalmente, es el desarrollo de componentes individuales en cada uno de sus 4 grupos, de modo que un diseñador pudiera escoger un componente ya implementado de cada uno de los grupos, y así construir un nuevo sistema reutilizando componentes. A la hora de desarrollar un sistema siguiendo las pautas de Vowel Engineering, de lo que se trata es de ir realizando las etapas de cada una de las vocales en un orden determinado por la importancia que tendrá cada una de ellas en un sistema **[14]**.

Aunque Vowel Engineering ha sido una de las primeras metodologías en modelar sistemas utilizando diferentes vistas. El trabajo con esta metodología está incompleto ya que no termina de estabilizarse con herramientas de soporte. El proceso de desarrollo es el punto débil de esta metodología ya que solo proporciona las vocales como resumen de elementos a considerar en el desarrollo y un conjunto de tecnologías aplicadas.

1.5.5 Scrum.

Scrum **[15]** (metodología ágil) es un modelo de referencia que define un conjunto de prácticas y roles, puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. Los roles principales en Scrum son el *ScrumMaster*, que mantiene los procesos y trabaja de forma similar al director de proyecto, el *Product Owner*, que representa a los *stakeholders* (clientes externos o internos), y el *Team* que incluye a los desarrolladores.

Durante cada *sprint*, un periodo entre 15 y 30 días (la magnitud es definida por el equipo), el equipo crea un incremento de software potencialmente entregable (utilizable). El conjunto de características que forma parte de cada sprint viene del *Product Backlog*, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar. Los elementos del *Product Backlog* que forman parte del sprint se determinan durante la reunión de *Sprint Planning*.

Durante esta reunión, el *Product Owner* identifica los elementos del *Product Backlog* que quiere ver completados y los hace del conocimiento del equipo. Entonces, el equipo determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint. Durante el sprint, nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el sprint.

Esta metodología no genera toda la evidencia o documentación de otras metodologías, no es apta para todos los proyectos, sólo habla de cómo despachar tareas, centrándose en la importancia de sacar trabajo adelante en cada uno de los Sprint, y en la importancia de avanzar en la línea de lo que el cliente quiere.

1.5.6 RUP (Rational Unified Process).

El Proceso Unificado de Desarrollo de Software (Rational Unified Process) constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos basados íntegramente en Lenguaje Unificado de Modelado UML como soporte a la metodología.

RUP es un producto de Rational. Se caracteriza por ser **[16]**:

- Iterativo e incremental: La alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos o versiones del producto donde a cada uno de estos se le denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo.
- Centrado en la arquitectura: La arquitectura representa la forma del sistema, la cual va madurando en su interacción con los casos de uso hasta llegar a un equilibrio entre funcionalidad y características técnicas.

- Guiado por los casos de uso: RUP utiliza los casos de uso tanto para especificar los requisitos funcionales del sistema, como para guiar todos los demás pasos de su desarrollo, dígase diseño, implementación y prueba.

Fases e Iteraciones de la Metodología RUP.

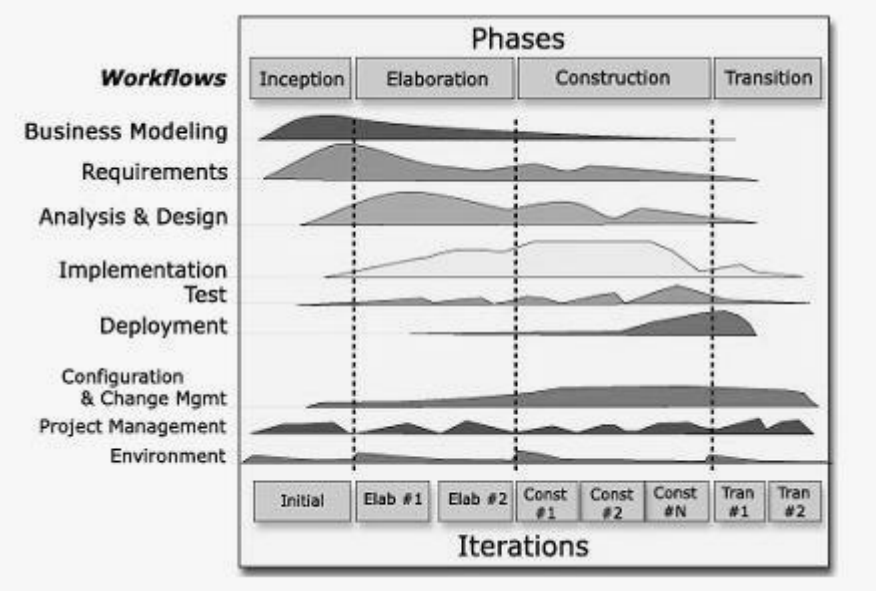


Figura 12: Fases e Iteraciones de la Metodología RUP.

RUP es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, y no tan solo de software.

El proceso puede describirse en dos dimensiones, o a lo largo de dos ejes:

- ✓ El eje horizontal representa tiempo y muestra el aspecto dinámico del proceso, expresado en términos de ciclos, fases, iteraciones, y metas.

- ✓ El eje vertical representa el aspecto estático del proceso; como está descrito en términos de actividades, artefactos, trabajadores y flujos de trabajo.

Define cuatro fases: Inicio, Elaboración, Construcción y Transición. Y nueve flujos de trabajo, seis de Ingeniería (Modelado del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba y Despliegue) y tres de apoyo (Gestión de la Configuración, Gestión de Proyecto y Ambiente).

RUP pretende implementar las mejores prácticas actuales en Ingeniería de Software:

- ✓ Desarrollo iterativo del Software.
- ✓ Administración de requerimientos.
- ✓ Uso de arquitecturas basadas en componentes.
- ✓ Modelación visual del software.
- ✓ Verificación de la calidad del software.
- ✓ Control de cambios.

1.5.7 Justificación de la propuesta seleccionada.

El éxito de un proyecto está estrechamente ligado a la metodología que se adopte para su realización, pero resulta difícil en ocasiones determinar cuál es la idónea para el desarrollo exitoso del proyecto en cuestión, por tanto la elección depende mucho de las características en que se debe desarrollar el mismo. En la realización de la Ingeniería Inversa de cualquier producto no se tienen en cuenta los pasos ni las fases de las metodologías debido a que ya el producto está hecho, solo se toman las partes de interés que el desarrollador considere necesario de acuerdo con la documentación que se necesite.

Después de hacer un análisis de varias de las metodologías utilizadas dentro del campo de la IA y en el desarrollo de videojuegos se decide utilizar RUP, que aunque no se utiliza en los entornos citados, goza de excelente prestigio dentro del mundo informático por los éxitos alcanzados, además el objetivo que

persigue este trabajo es conformar la documentación de la Biblioteca, de todas las metodologías analizadas, RUP es la que más se centra en la documentación del producto, le da énfasis a los requisitos y al diseño del producto, las demás metodologías tienen como objetivo principal el producto final, prestándole menos atención a la documentación que lleva el mismo. Aunque han progresado en la integración con la ingeniería del software clásica, aún no muestran la madurez que se puede encontrar en metodologías convencionales como el Unified Process. El motivo principal es que siguen faltando herramientas de soporte y un lenguaje para la especificación del Sistema Multi-Agente que permitan trabajar de forma similar a como se trabaja en Rational Rose o Visual Paradigm.

1.6 ¿Que es la Ingeniería Inversa?

La Ingeniería Inversa **[17]** se ha definido como el proceso de construir especificaciones de un mayor nivel de abstracción partiendo del código fuente de un sistema de software o cualquier otro producto (se puede utilizar como punto de partida cualquier otro elemento de diseño.).

Estas especificaciones pueden volver a ser utilizadas para construir una nueva implementación del sistema, utilizando por ejemplo, técnicas de Ingeniería Directa. La finalidad de la Ingeniería Inversa es la de desentrañar los misterios y secretos de los sistemas en uso a partir del código.

Se deben afrontar 3 enfoques de Ingeniería Inversa: nivel de abstracción, completitud y direccionalidad. Antes de comenzar a realizarle la Ingeniería Inversa a un sistema (código fuente o ejecutable) se debe entender a la perfección su funcionamiento para poder tener un mayor nivel de abstracción.

En otras palabras, la Ingeniería Inversa es un proceso que recorre hacia atrás el ciclo de desarrollo de software. Se inicia a partir del código fuente o un ejecutable hasta llegar a la fase de análisis. La Ingeniería Inversa realizada a esta Biblioteca de IA fue a partir de un código fuente con el objetivo de generar la documentación necesaria para su reutilización.

1.6.1 Beneficios de Ingeniería Inversa.

- ✓ Reducir la complejidad del sistema: al intentar comprender el software se facilita su mantenimiento y la complejidad existente disminuye.
- ✓ Generar diferentes alternativas: del punto de partida del proceso, principalmente código fuente, se generan representaciones gráficas lo que facilita su comprensión.
- ✓ Detectar efectos laterales: los cambios que se puedan realizar en un sistema puede conducirnos a que surjan efectos no deseados, esta serie de anomalías pueden ser detectadas por la Ingeniería Inversa.
- ✓ Facilitar la reutilización: por medio de la Ingeniería Inversa se pueden detectar componentes de posible reutilización de sistemas existentes, pudiendo aumentar la productividad, reducir los costes y los riesgos de mantenimiento.

1.7 Lenguaje UML.

Figura 13: Logo del lenguaje modelado UML.

Es un lenguaje de modelado visual (UML) **[18]** que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo.

El lenguaje de modelado pretende unificar la experiencia basada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar.

UML no es un lenguaje de programación. Las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguaje de programación, así como construir modelos por Ingeniería Inversa a partir de programas existentes.

Es un lenguaje de propósito general para el modelado orientado a objetos. UML es también un lenguaje de modelamiento visual que permite una abstracción del sistema y sus componentes.

Existían diversos métodos y técnicas Orientadas a Objetos (OO), con muchos aspectos en común pero utilizando distintas notaciones, se presentaban inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc., además de pugnas entre enfoques, lo que generó la creación del UML como estándar para el modelamiento de sistemas de software principalmente, pero con posibilidades de ser aplicado a todo tipo de proyectos.

1.8 Herramienta CASE.

Las Herramientas CASE **[19]** (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática y documentación o detección de errores.

1.8.1 Rational Rose Enterprise Suite.

El Rational es una herramienta CASE desarrollada por Rational Corporation, basada en UML, permite crear los diferentes diagramas que se generan en el proceso de Ingeniería durante el desarrollo del software. Presenta un gran número de estereotipos que permiten el proceso de modelación del software.

Esta herramienta es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño, pero tiene la limitación de que aún hay varios lenguajes de programación que no lo soporta o que sólo lo hace a medias.

1.9 Herramientas generadoras de documentación.

La documentación del código de un software suele ser el último paso en el ciclo habitual de un proyecto. Mientras que la importancia de este paso ha sido expuesta en disímiles oportunidades, los desarrolladores dedican la mayor parte del tiempo a la terminación del software y a la hora de hacer la documentación solo logran un trabajo mediocre, por eso es recomendable documentar el código a medida que se programe. De la documentación del código de un software se pueden generar documentación de este producto en diferentes formatos (*.html, *.pdf, *.xml), para realizar este tipo de trabajo existen las herramientas generadoras de documentación.

1.9.1 Doxygen.

Doxygen **[20]** es un programa de Dimitri van Heesch con licencia GLP que te permite la creación de documentación a partir del código fuente de un programa. Para esto, Doxygen utiliza la gramática del lenguaje en que esté escrito el código, así como los comentarios en un formato particular. Los formatos que soporta son: C, C++, Java, Objective C, Python, IDL y, en algunos casos, PHP, C# y D. Por otro lado, la documentación puede generarse en HTML, LaTeX, RTF, PDF y XML.

El principal interés de Doxygen es permitir la integración de la documentación directamente en el código fuente, favoreciendo así la coherencia entre ambos. Este software también posibilita extraer documentación a partir de un código fuente no documentado de antemano, lo que facilitará la compresión de un programa generado por este código.

Parte de la información que puede extraerse del código fuente es:

- ✓ Prototipo y documentación de las funciones, ya sean locales, privadas o públicas.
- ✓ Lista de archivos incluidos.
- ✓ Documentación de las estructuras de datos.
- ✓ Prototipo y documentación de las clases y su jerarquía.
- ✓ Diferentes tipos de gráficas: diagramas de clase, de colaboración, de llamadas, etc.
- ✓ El índice de todos los identificadores.

CAPITULO 2: CARACTERISTICAS DEL SISTEMA.

Introducción:

En este capítulo se representa el diagrama del modelo del dominio explicando los principales conceptos que tienen que ver con la solución, se identifican los requisitos funcionales y no funcionales de los cuales se derivan los casos de uso, presentándose también el diagrama de casos de uso identificándose así los actores que intervienen en los casos de uso y se hace un descripción detallada de cada caso de uso.

2.1 Modelo del Dominio.

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.

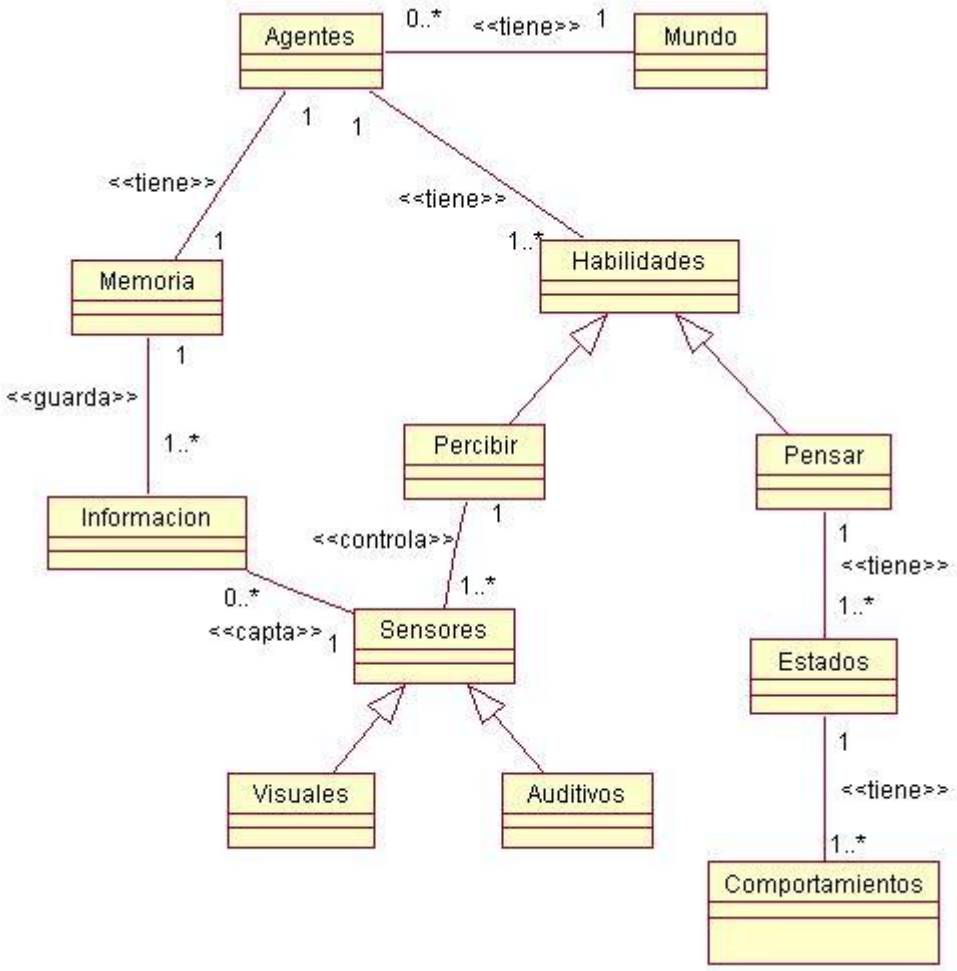
En este trabajo se realiza modelo de dominio ya que no se logra determinar el proceso del negocio con fronteras bien establecidas, no se logra ver claramente quienes son las personas que lo inician, los beneficiados con cada uno de estos procesos, pero además quienes son las personas que desarrollan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

El Modelo de Dominio (o Modelo Conceptual) 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. Representa 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.

Estos objetos del dominio representan “cosas” que existen o los eventos que acontecen en el medio en el que se desenvuelve la aplicación **[21]**.

2.1.1 Diagrama del Modelo del Dominio.



2.1.2 Glosario de Términos.

- ✓ **Mundo:** Entorno Virtual en el que se desarrolla el juego.
- ✓ **Agente:** Personaje del Entorno Virtual que posee inteligencia.
- ✓ **Memoria:** Lugar donde se guardan los datos captados por el Agente en el Entorno Virtual.

- ✓ **Habilidades**: Cualidades que posee cada Agente para actuar de forma inteligente.
- ✓ **Pensar:** Habilidad que tiene cada Agente que le permite saber qué hacer en las disímiles situaciones en las que se puede encontrar.
- ✓ **Percepción:** Habilidad que poseen los Agentes para captar información del entorno en el que se encuentran.
- ✓ **Información:** Datos receptados por el Agente del mundo donde se encuentra.
- ✓ **Estados:** Son los estados por los que el Agente puede pasar durante el transcurso del juego.
- ✓ **Comportamiento:** Es el comportamiento que debe seguir un Agente según el estado en el que se encuentre.
- ✓ **Sensores:** Atributos o accesorios que le permiten al Agente obtener información del Entorno Virtual.
- ✓ **Visuales:** Sensores para captar la información visual.
- ✓ **Auditivos:** Sensores para captar la información auditiva.

2.2 Requisitos.

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.

2.2.1 Requisitos funcionales

RF 1-Gestionar Agente.

RF 1.1-Registrar Agente.

RF 1.2-Eliminar Agente.

RF 2-Gestionar Servicios.

RF 2.1- Registrar Servicios.

RF 2.2- Eliminar Servicios.

RF 2.3-Devolver información de un Servicio.

RF 3- Gestionar Información del Mundo.

RF 3.1-Registrar información obtenida del ambiente.

RF 3.2-Eliminar información obtenida del ambiente.

RF 3.3-Buscar información obtenida del ambiente.

RF 4-Gestionar Servicios de Actualizar.

RF 4.1-Registrar actualización.

RF 4.2-Eliminar actualización.

RF 4.3-Eliminar todas las actualizaciones.

RF 5-Gestionar Componentes.

RF 5.1-Insertar componentes.

RF 5.2-Eliminar todos los componentes de un Agente.

RF 5.3-Eliminar componente.

RF 5.4-Obtener información de un componente.

RF 6-Actualizar Mundo.

RF 6.1-Actualizar información obtenida.

RF 6.2-Actualizar Servicios de Pensar.

2.2.2 Requisitos no funcionales.

- Requerimientos de Software.

Sistema Operativo Windows, Visual Studio.Net.

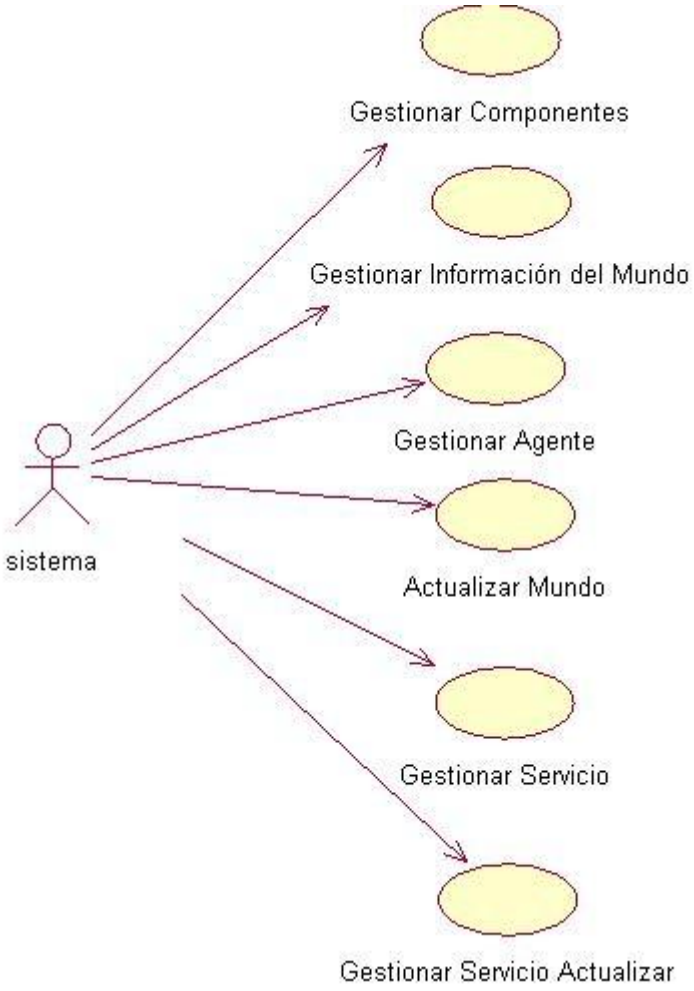
2.3 Casos de Usos del Sistema.

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 y posibilidades (requisitos) que debe cumplir el sistema.

2.3.1 Definición de los actores del Sistema.

Actor	**Justificación**
Sistema	Es el que interactúa con la Biblioteca. Se encarga de ejecutar las diferentes funcionalidades de la Biblioteca.

2.3.2 Diagrama de Casos de Usos del Sistema.



2.3.3 Descripción detallada de los Casos de Uso 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. Aquí se presenta la descripción detallada del Caso de Uso Gestionar Agente, las demás descripciones referentes a los demás casos de uso se encuentran en el **(Anexo 2)**.

Caso de Uso:	Gestionar Agente.
Actores:	Sistema
Resumen:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita adicionar o eliminar Agentes.

Precondiciones:	Se debe haber creado un mundo con anterioridad.
Referencias	RF 1, RF 1.1, RF 1.2
Flujo Normal de Eventos	
Sección "Registrar Agente"	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema le da al Mundo el Agente que quiere registrar.	1.1- Verifica que este Agente no se encuentre registrado. 1.2– Guarda la información correspondiente al Agente en el Servicio Pensar.
Poscondiciones	Queda registrado un Agente en el Mundo.
Sección "Eliminar Agente"	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a eliminar un Agente.	1.1– Verifica que existe el Agente que se quiere eliminar. 1.2– Elimina el Agente deseado del Mundo.
Poscondiciones	Queda eliminado un Agente del Mundo.

CAPITULO 3: DISEÑO E IMPLEMENTACION.

Introducción:

En este capítulo se realiza una explicación detallada de la estructura y el funcionamiento de la Biblioteca, se analiza cómo el Agente se beneficia con las diferentes técnicas de Inteligencia Artificial que se encuentran implementadas en la Biblioteca, se desarrollan los diagramas de secuencia de acuerdo con cada caso de uso identificado en el capítulo anterior, se realizan los diagramas de paquetes donde estarán agrupadas las clases según las funcionalidades en común, se conforma el diagrama de componentes y se describe detalladamente cada una de las clases con sus atributos y métodos.

3.1 Funcionamiento de la Biblioteca.

El Actor que se encuentra implementado en la Biblioteca, representa el Agente que se menciona en capítulos anteriores, se realiza este cambio para que no cause confusión con el Actor del Sistema, ya en este capítulo el término que se utiliza es el de Actor, que es el que se encuentra en la Biblioteca

La Biblioteca tiene como clase rectora el Mundo, esta brinda una serie de Servicios a los elementos inteligentes que se encuentran dentro de ella, escenario donde transcurre el juego.

- **Percepción**, este servicio controla todos los eventos que ocurren en el Mundo que puedan ser de interés para los Actores y que además puedan ser percibidos por sus sensores. Cada cierto tiempo el Mundo manda a los sensores del Actor a tomar datos del entorno en el que se encuentran.

- **Pensar**, este servicio tiene un control sobre los Cerebros y sus Actores, como bien dice su nombre su función fundamental es poner a pensar a los Cerebros para que cada uno realice la función que le corresponde.

- **Actualizar,** este servicio mantiene un control estricto sobre todas las actualizaciones que ocurren en el juego en diferentes instantes de tiempo.

En este Mundo va a existir una determinada cantidad de Actores. Estos Actores van a estar compuesto por 3 componentes fundamentales:

- **Memoria**, esta va a estar conectada directamente con los dos tipos de Sensores con los que va a contar el Actor para poder obtener información del Mundo y así poder procesarla, estos Sensores son de audio y visibilidad. Por tanto todos los datos de interés para el Actor van a ser guardados en este Componente.

- **Localización**, este componente es el que maneja toda la información y el desarrollo del movimiento del Actor dentro del espacio. Contiene los métodos capaces de mover, rotar y cualquier otro movimiento del Actor dentro del Mundo.

- **Cerebro**, componente fundamental del Actor, dirige todas las tareas que ejecuta el Actor. Como el Actor tiene que ejecutar disímiles tareas, este componente se divide en varios SubCerebros según las tareas que tiene que realizar el Actor.

    - ✓ **Cambio de estados**, durante el transcurso del juego el Actor transitará por diferentes estados que, para llegar a cada uno de estos estados y cambiar de un estado a otro deben ocurrir una serie de eventos (transiciones). En cada uno de estos estados el Actor debe realizar un conjunto de tareas al igual que cuando sale de los mismos. Algunos de los eventos que dan lugar a los estados no están relacionados directamente a estos sino a otros eventos, estos eventos que preceden a otros eventos ejecutan tareas sobre los mismos. Todo este proceso da respuesta a la Técnica de IA FSM.

- ✓ **Control de Script,** una vez terminado el juego se le pueden hacer diferentes modificaciones a los estados y transiciones por las que pasa el Actor mediantes los Script sin necesidad de recompilar todo el código.
- ✓ **Comportamientos (Steering Behaviors),** durante el juego el Actor debe tener cierto comportamiento, estos comportamientos están ligados a los estados por los que pasa el Actor y a la información que obtienen del Mundo. La ejecución correcta de estos comportamientos pueden lograr que el Actor tenga una actitud razonable antes las situaciones que se le presenten en el entorno.
- ✓ **Cerebro rector,** como existen diferentes tareas las cuales las ejecutan los subcerebros, se necesita de uno que dirija esta acción, este, según la prioridad de estas tareas decide en qué orden se van a ejecutar.

3.1.1 Relación de la Biblioteca con el juego Laberinto del Saber.

La Biblioteca fue diseñada para ser utilizada en el juego Laberinto del Saber (Proyecto de la Facultad 5), este juego necesitaba de otros módulos además del de IA para su funcionamiento. Esta cuenta con un total de 41 clases de las cuales 14 dependen de alguna manera de 12 clases del juego (cVariableHolder, cUpdater, cSimulationLayer, cMatrix4f, cQuaternionf, cVector3f, AI_constants, cScriptPack, cScriptRunnerBehavior, cEventCatcher, cTrigger, cFrustum) definidas dentro de estos módulos, lo que representa un 44 % del total. Esto imposibilita que la Biblioteca pueda ser usada de forma independiente en otros proyectos en los que se necesite.

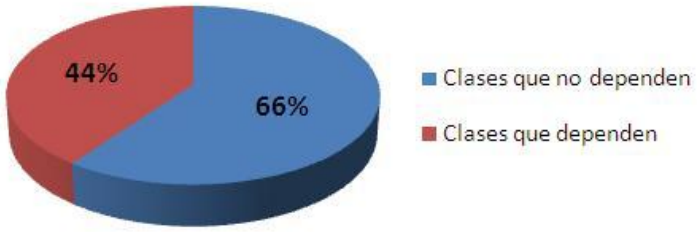


Figura 14: Relación de las clases de la Bibliotecas con otras clases.

3.1.2 Criterio sobre la Biblioteca.

Como ya se dijo la Biblioteca tiene un 44% de dependencia con el juego Laberinto del Saber, en este juego existen vario tipos de datos (cQuaternionf, cVector3f&, cMatrix4f) los cuales utiliza la Biblioteca, estos tipos de datos pueden ser convertidos a otros tipos de datos similares y así poder ir independizando la Biblioteca del Juego. Si se hace la conversión de datos entonces del juego solo dependerían 6 clases (que representa un 14.6 %) de las 14 iniciales, como se puede apreciar la diferencia es considerable.

El manejo de Script está definido dentro de la Biblioteca, pero al igual que otras funcionalidades, no está implementado, la clase encargada de manejar estos Script depende de 3 de las 12 clases que utiliza la Biblioteca del juego Laberinto del Saber, su posibilidad de independizarla resulta muy trabajosa.

La Biblioteca cuenta con un módulo de Steering Behaviors simples donde están definidos varios comportamientos, todas las funciones están implementadas, por lo que se puede aislar este módulo si se necesita usar de forma independiente.

Todas las FSM en cada estado arrojan uno o más comportamientos y según las circunstancias se decide qué comportamiento es el más adecuado, esto demuestra la relación entre los Steering Behaviors y las FSM, sin embargo esta relación directa, no se pone de manifiesto en la implementación de la

Biblioteca. La FSM que se encuentra implementada no está completa, los estados para poder funcionar dependen de sus transiciones, y estas no están implementadas por tanto tienen un 0 % de funcionalidad. Dado esto, podemos decir que para que la técnica de FSM funcione en su totalidad se deben implementar las transiciones y asociar estas con los SB.

Después de todo este estudio, se puede decir que existen algunas clases en las cuales todas sus funcionalidades no se encuentran implementadas (cpMovil, CBrain_FSM_Transition, cMemorySlot), además existen otras que tiene funcionalidades implementadas, pero son insuficientes para su completo funcionamiento (CBrain_FSM_State).

La Biblioteca tiene presente un patrón de diseño el cual está hecho para diseñar FSM (Patrón State). Este patrón propone un diagrama de clases (**Ver figura 15**), además establece, que como pueden existir varios comportamientos en un solo estado, se divida cada comportamiento en sub-Estados para aminorar la complejidad a la hora de programar.

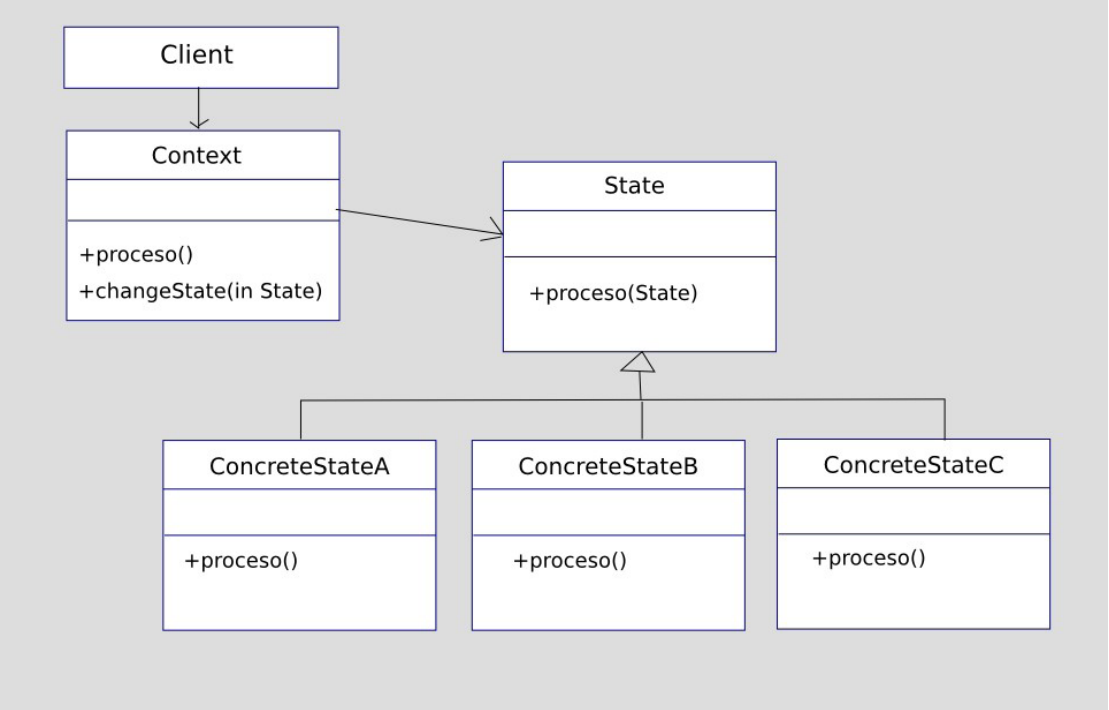


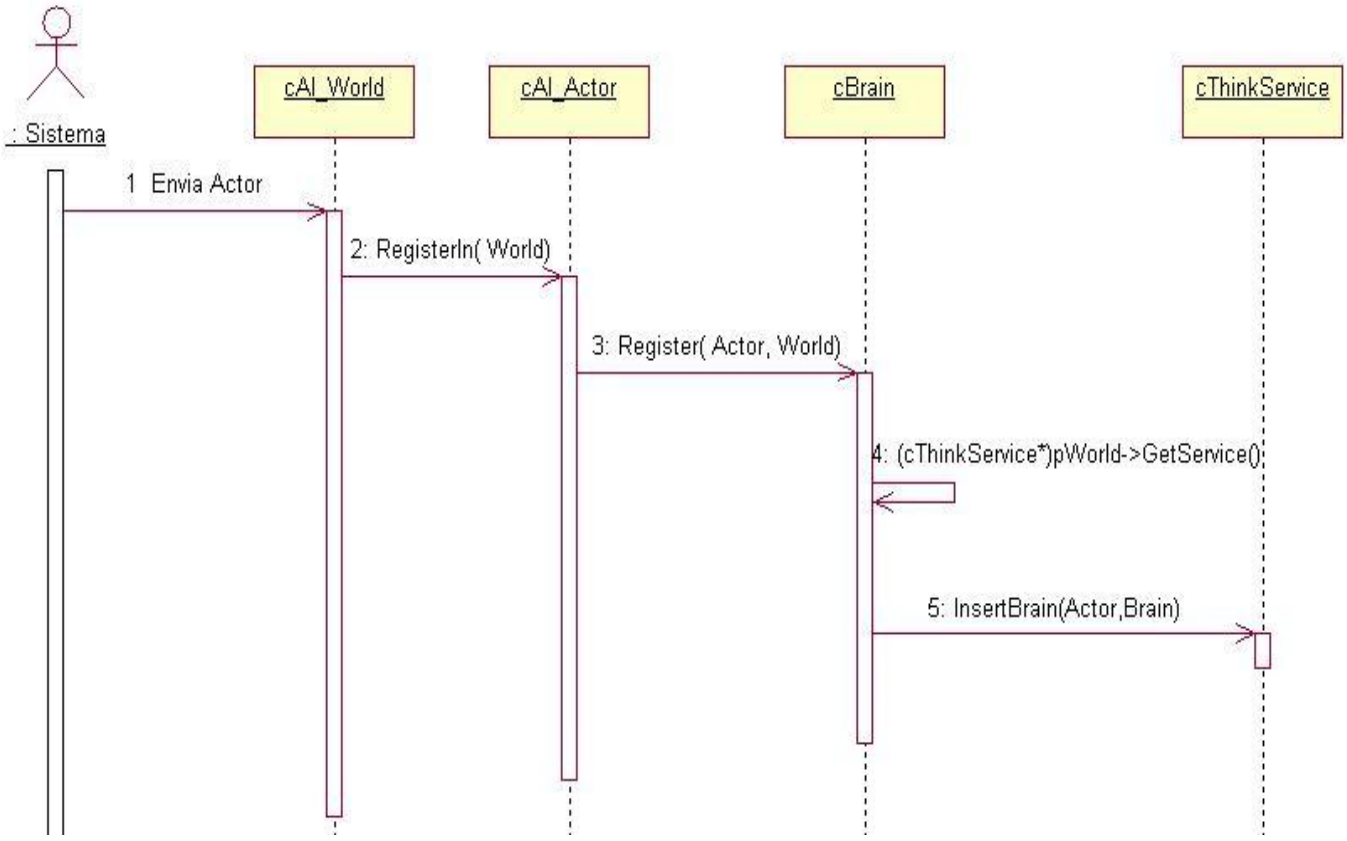
Figura 15: Diagrama de Clase propuesto por el patrón State para FSM.

En la Biblioteca, según este patrón, el Client sería el Actor, el Context sería cBrain_FSM y el State está representado por la clase cBrain_FSM_State.

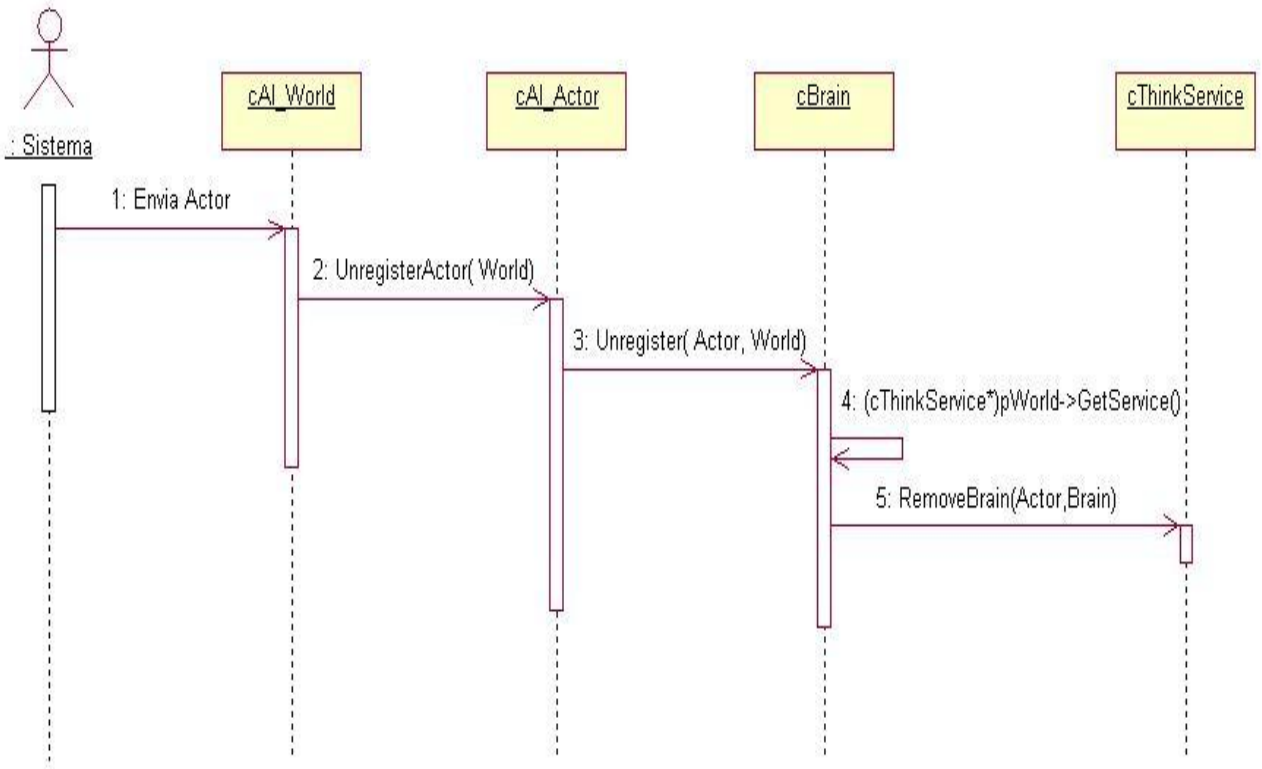
3.2 Diagrama de Secuencia.

Los Diagramas de Secuencia son uno de los diagramas más efectivos para modelar la interacción entre objetos en un sistema. Un diagrama de secuencia se modela para cada caso de uso. Mientras que el diagrama de caso de uso permite el modelado de una vista del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos **[22]**. A continuación se muestran los Diagramas de Secuencia más importantes, el resto se encuentran en **(Anexo 3)**.

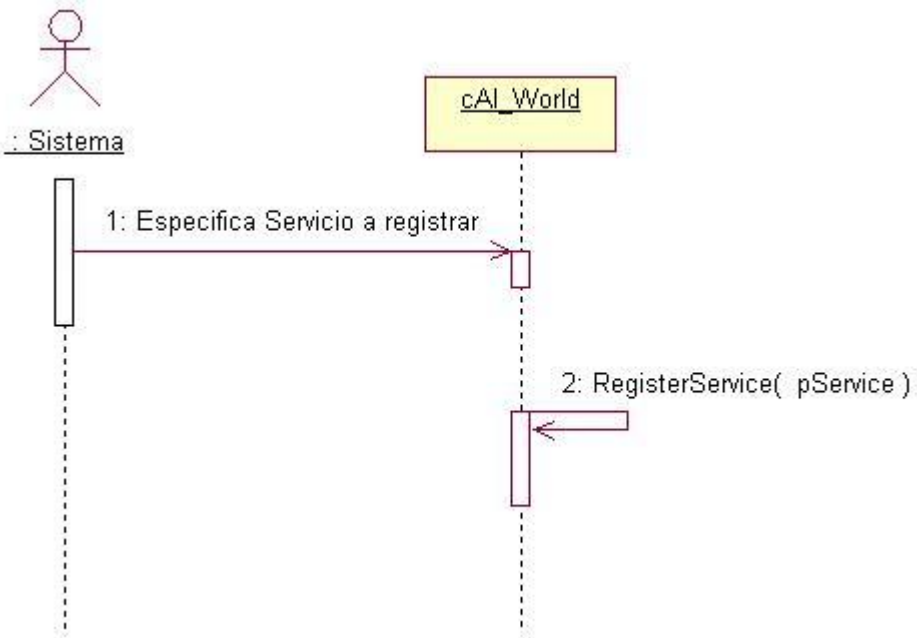
- **Diagrama de Secuencia: "Registrar Actor"**



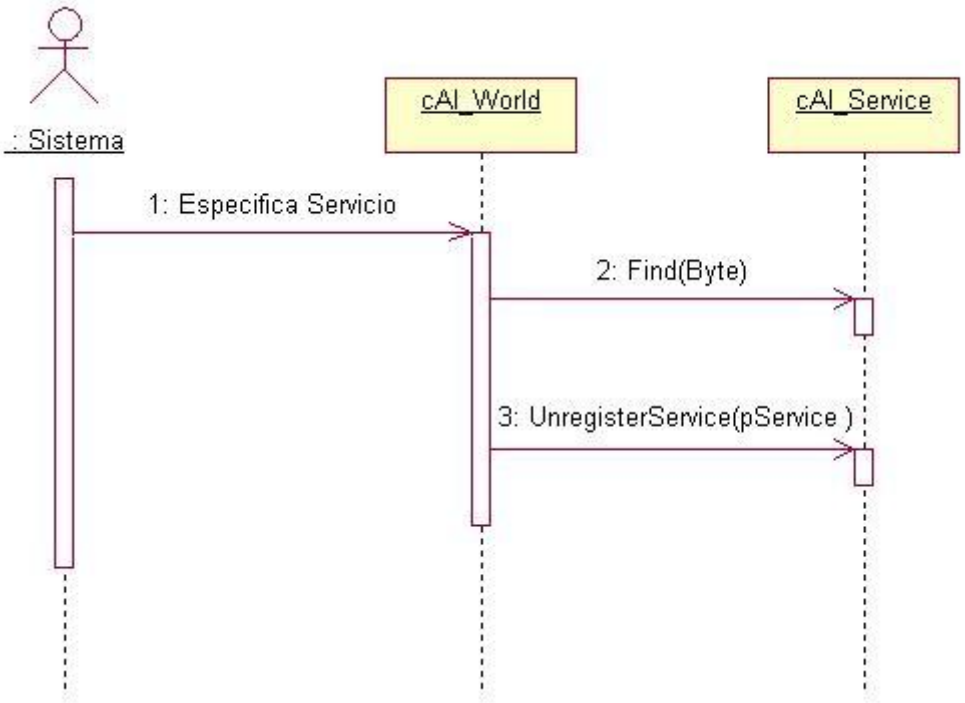
- **Diagrama de Secuencia: "Eliminar Actor"**



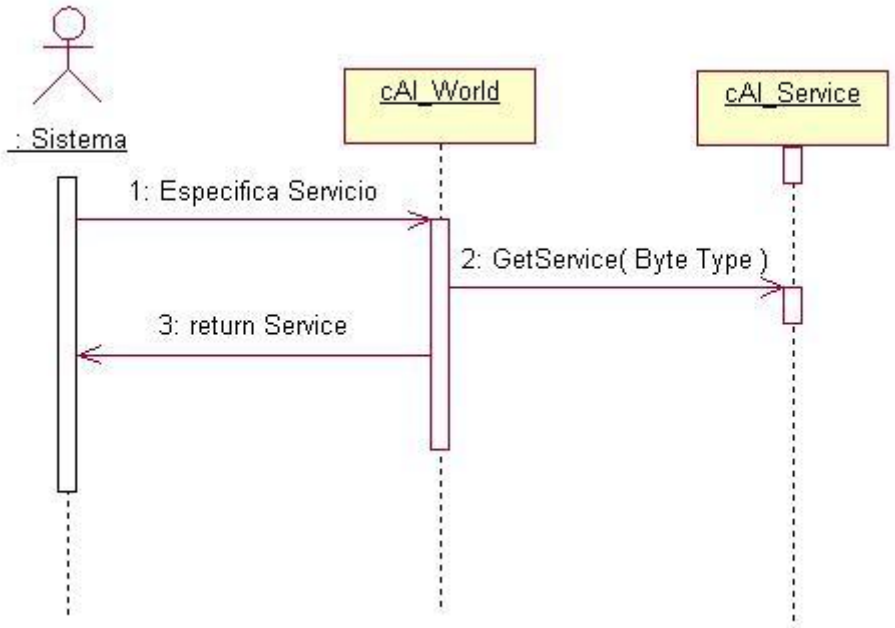
- **Diagrama de Secuencia: "Registrar Servicio"**



- **Diagrama de Secuencia: "Eliminar Servicio"**

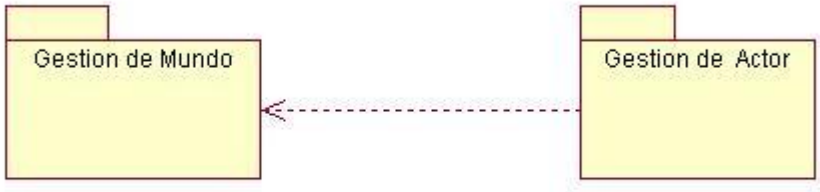


- **Diagrama de Secuencia: "Dar información de un Servicio"**



3.3 Diagrama de paquetes.

Un diagrama de paquetes muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema. Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido **[23]**.



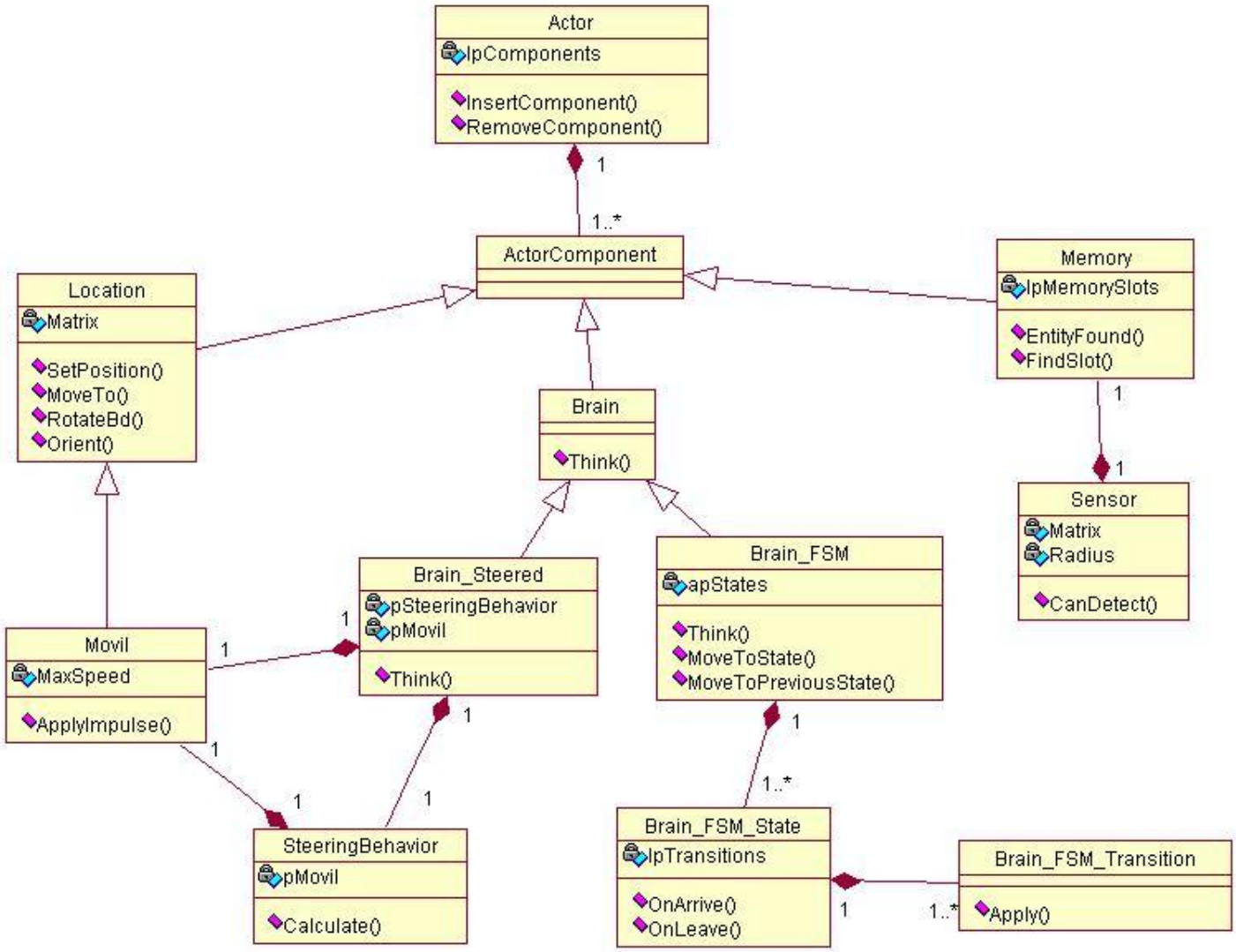
3.4 Diagramas de clases del diseño por paquetes.

En este epígrafe se muestran los diagramas de clases por cada uno de los paquetes identificados anteriormente. Estos diagramas de clases se realizaron por paquetes para así poder hacer el modelo del diseño en partes más pequeñas y para una mejor comprensión de estos diagramas.

3.4.1 Diagrama de clases del paquete Gestión de Actor:

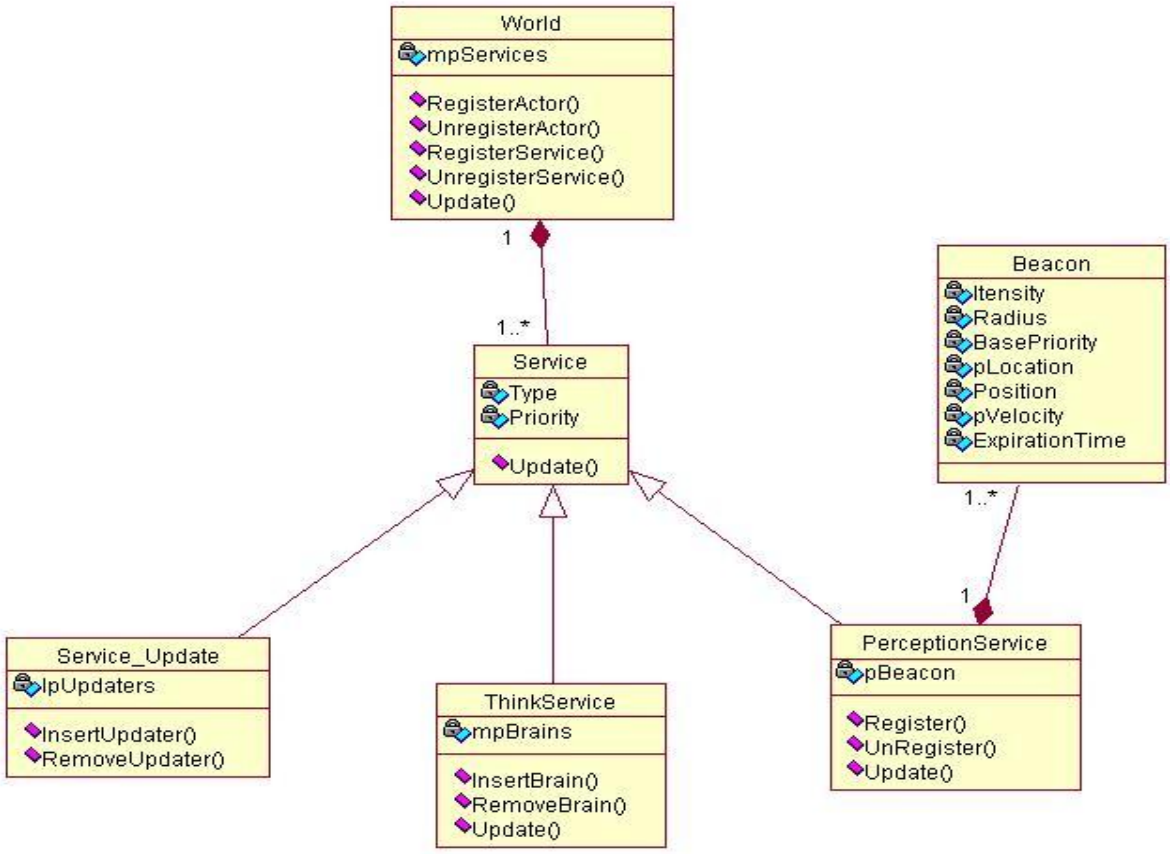
En este paquete se agruparon los siguientes casos de uso, Gestionar Actor y Gestionar Componente, en ambos casos la clase que dirige todo el proceso de desarrollo de los casos de uso es la clase Actor. Todas las clases que conforman este diagrama son elementos que componen al Actor,

y cada una de ellas influyen de alguna manera en el desenvolvimiento del Actor en el desarrollo del juego.



3.4.2 Diagrama de clases del paquete Gestión de Mundo.

En este paquete se agruparon los siguientes casos de uso, Gestionar Servicios, Gestionar Información del Mundo, Gestionar Servicios de Actualizar y Actualizar Mundo. En estos casos de uso el Mundo es la clase que dirige todo el proceso de ejecución de estos casos de usos, las clases que se ven implicadas son las clases que responden a los Servicios que presta el Mundo a los Actores que lo componen, así como a los sucesos que ocurren en él.



3.5 Diagrama de Componentes por paquetes.

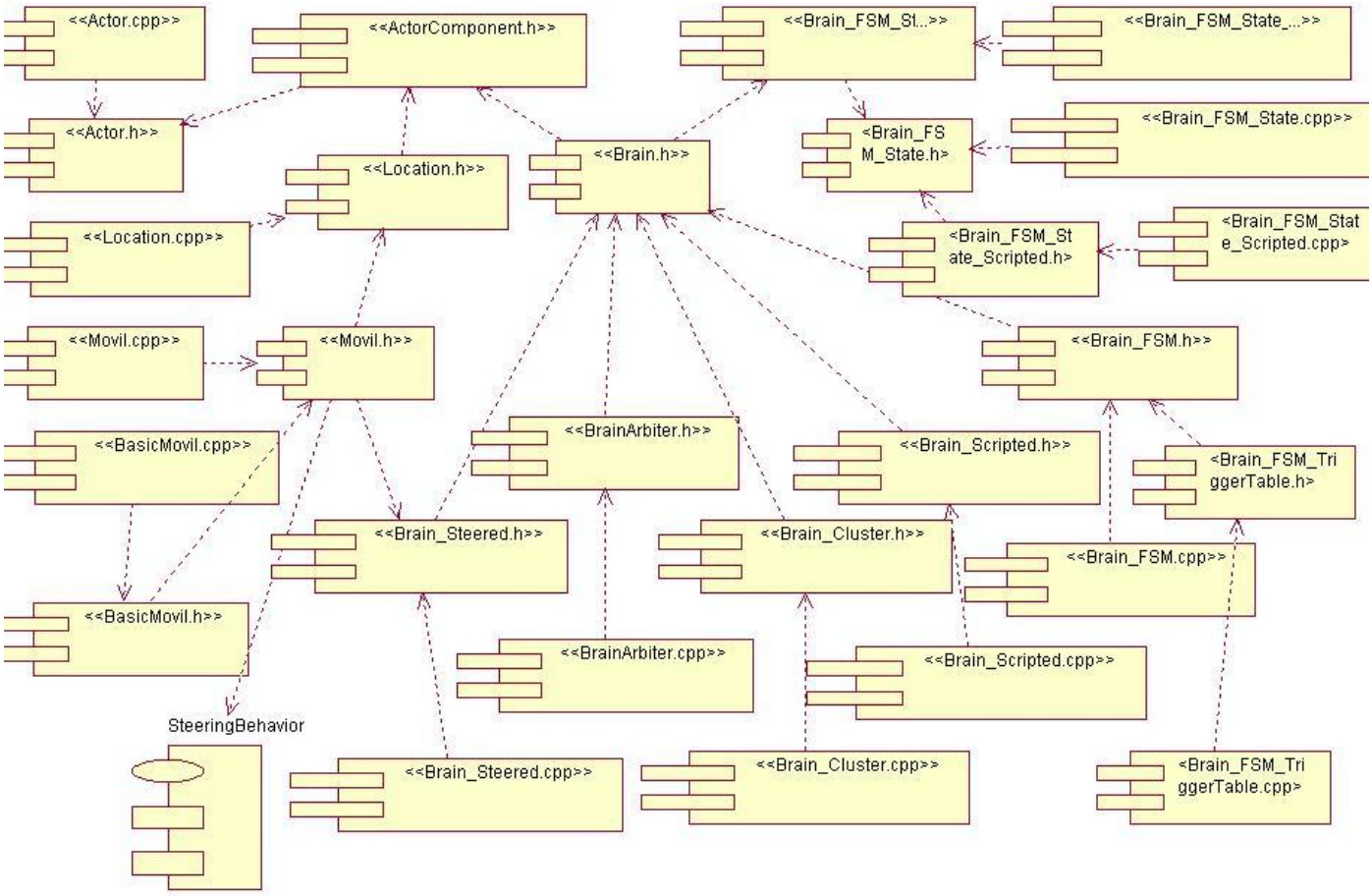
Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación.

El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación, especificando:

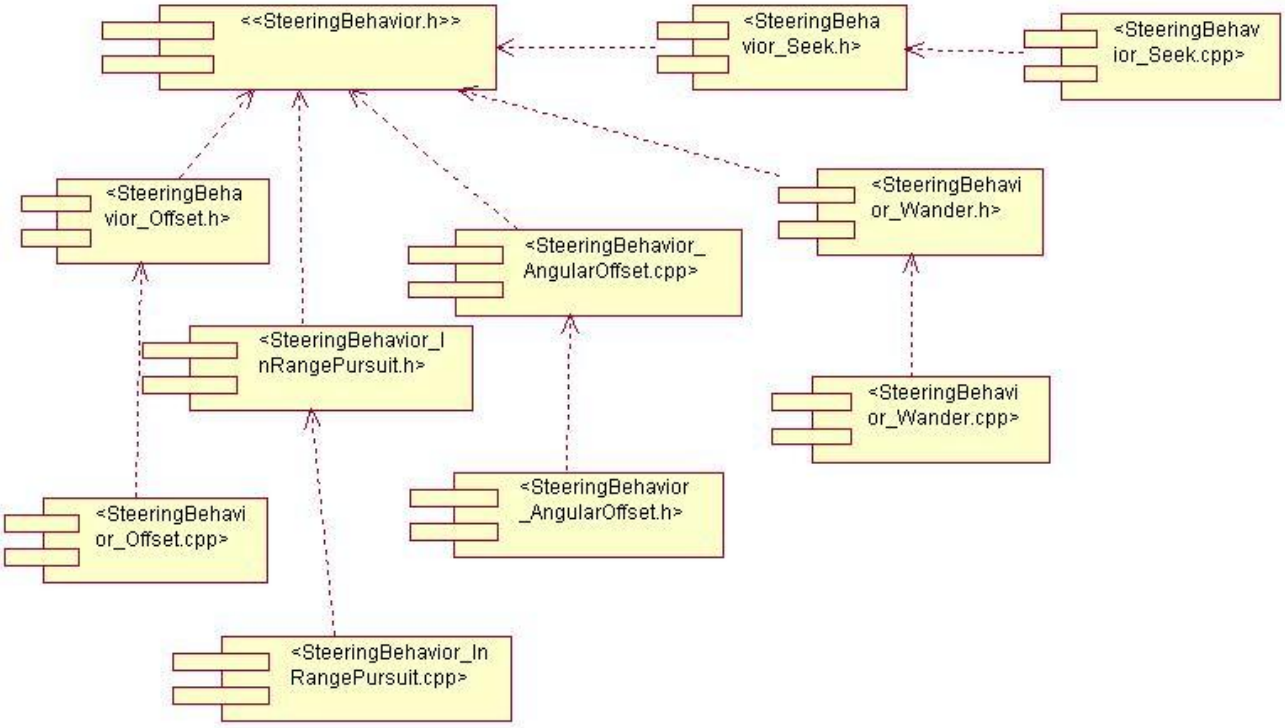
- Los subsistemas de implementación y sus dependencias a la hora de importar código.
- Organizar los subsistemas de implementación en capas.

También se utilizan para mostrar las dependencias de compilación de los ficheros de código, relaciones de derivación entre ficheros de código fuente y ficheros que son resultados de la compilación, dependencias entre elementos de implementación y los correspondientes elementos de diseño que son implementados **[24]**.

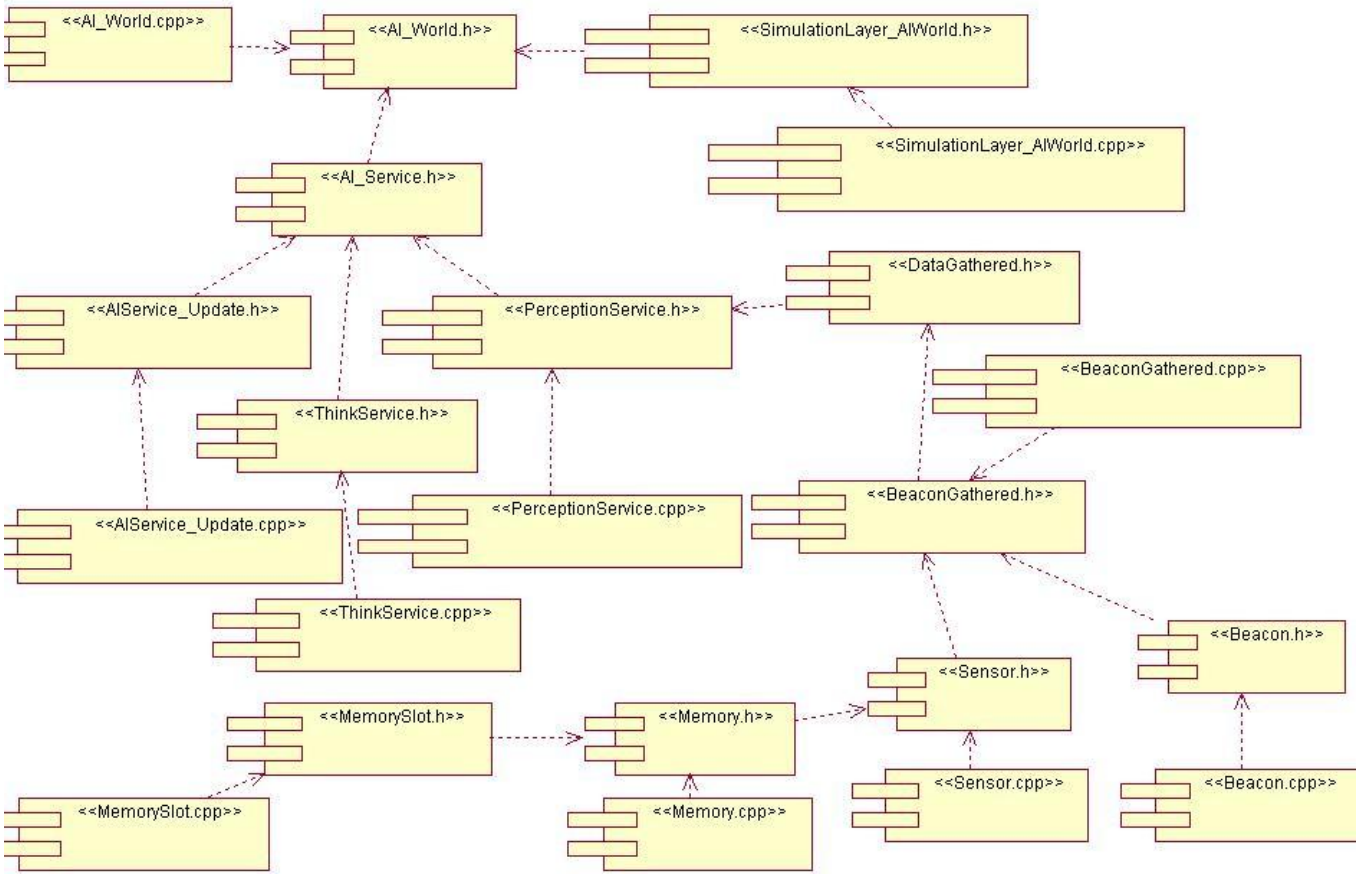
- ✓ **Diagrama de Componentes del paquete Gestión de Actor.**



- ✓ **Diagrama de Componentes del Paquete de Componentes Steering Behaviors.**



- ✓ **Diagrama de Componentes del paquete Gestión de Mundo.**



3.6 Descripción detallada de las Clases.

Antes de llegar a la descripción de cada una de las clases que conforman la Biblioteca se tuvo que pasar por dos pasos previos, comentar el código usando el estilo del proyecto de la Facultad 5 Scence Tolkit (STK) para mantener el estándar usado por esta Facultad, una vez concluido este paso se pasa al siguiente, usar la herramienta Doxygen para generar la documentación de la Biblioteca en formato HTML, esta herramienta genera un sitio Web el cual permite interactuar entre la descripción de las clases y su código (**Ver Anexo 4**). Una vez realizados los pasos anteriores se creó una tabla para cada una de las clases donde los elementos que la conforman son: nombre de la clase, descripción, nombre de los atributos y métodos seguido de sus descripciones, la herencia y la dependencia con otras clases. Tanto el código comentado, como el sitio Web y las tablas de las clases describen el funcionamiento de las

clases, así como su relación con otras. A continuación se muestra una de las tablas de la clase Actor, las demás se encuentran en el **(Anexo 5)**.

	Nombre de la Clase: cAI_Actor		
	Descripción: Esta clase implementa las funciones que puede realizar un Actor.		
	Tipo	**Nombre**	**Descripción**
Atributos	list<cActorComponent*>	lpComponents	Estructura que guarda todos los Componentes de cada Actor.
	cVariableHolder	Datas	Información: (Max. velocidad, Vida, etc).
	void*	pOwnerEntity	Define que tipo de Actor es, ejemplo: Jugador, Ítem, etc.
Métodos	virtual void	Update	Método para actualizar el Actor en un instante de tiempo determinado.
	virtual void	RegisterIn	Método para Registrar al Actor en un Mundo.
	virtual void	UnRegisterFrom	Método para Eliminar al Actor de un Mundo.
	virtual void	Enable	Método que Activa al Actor y sus Componentes.
	virtual void	Disable	Método que Desactiva al Actor y sus Componentes.
	virtual void	InsertComponent	Método que le adiciona un nuevo componente al Actor si este no lo tiene todavía.
	virtual cActorComponent*	GetComponent	Método utilizado para obtener toda la información de un Componente especificado.
	virtual void	RemoveAllComponents	Método que borra todos los Componentes del Actor.
	virtual void	RemoveComponent	Método que borra un Componente especificado.
	virtual void	RemoveComponent	Método que borra el Componente que coincide con el identificador.
	Herencia		
	Dependencia		cActorComponent, cVariableHolder.

CONCLUSIONES

Una vez concluido este trabajo, se tiene como resultado la documentación de la Biblioteca de IA desarrolla en la Facultad 5, utilizando el Proceso Unificado de Rational (RUP) y el Lenguaje UML para la representación de los diagramas correspondientes. Por lo que se alcanzó, satisfactoriamente, el objetivo propuesto: conformar la documentación de la Biblioteca de Inteligencia Artificial desarrollada en la Facultad 5 haciendo uso de la metodología más adecuada.

La documentación obtenida describe cada una de las clases de la Biblioteca así como de sus atributos y métodos, esto permite que los futuros usuarios de la Biblioteca puedan hacer un uso óptimo de la misma.

Después de un análisis del código fuente se pudo detectar que la Biblioteca cuenta con tres técnicas de IA implementadas (Percepción, Maquina de Estado Finita y Steering Behaviors (Comportamientos)) lo que la hace aplicable a una gran variedad de géneros de videojuegos.

La relación de la misma con el Laberinto del Saber no le permite ser usada de forma independiente en otros videojuegos.

El análisis profundo de las técnicas presentes en la Biblioteca arrojó los siguientes resultados:

- ✓ Las técnicas de Percepción presentes en la Biblioteca son de las menos recomendadas para el desarrollo de videojuegos.
- ✓ Los Steering Behaviors implementados son simples por lo la Biblioteca no es asequible a los videojuegos donde están presentes equipos o grupos de un mismo bando.

RECOMENDACIONES

- Que se cree un nuevo módulo para la Biblioteca con las clases del juego Laberinto del Saber de las cuales esta hace uso, para de esta forma lograr independizarla de este juego.

- Que se perfeccionen las técnicas de Percepción implementadas para elevar el nivel de aceptación de la Biblioteca en el desarrollo de futuros productos.

- Que se le incrementen los Steering Behaviors (comportamientos) grupales para aumentar los géneros de videojuegos donde pudiera ser usada la Biblioteca.

- Una vez realizadas las recomendaciones anteriores se recomienda realizar un Manual de Usuario con algunos ejemplos para que la Biblioteca pueda ser usada con mayor facilidad por aquellas personas interesadas en su uso.

BIBLIOGRAFIA

[1] http://ampogames.wordpress.com/2006/09/04/sony-primara-la-inteligencia-artificial-sobre-los-graficos/

[2] INTELIGENCIA ARTIFICIAL Y ROBOTICA
http://www.inteligenciaartificial.cl/ciencia/software/ia/inteligencia_artificial.htm

[3] Charles River Media – "AI Game Engine Programming" pp 67-235

[4] Brownlee, Jason," Finite State Machines". [Disponible en]: http://ai-depot.com/FiniteStateMachines/

[5] Falcón García, Ricardo Ernesto," Módulo de algoritmos de locomoción con múltiples Steering Behaviors" [Disponible en: http://bibliodoc.uci.cu/TD/TD_1559_08.pdf]

[6] Percepción artificial
http://grvc.us.es/rar/mainFrame/percepcion/percepcion.html

[7] Puig, Frank Placeres **"**DEA Topics**"**

[8] Pontevia, Pierre. Open de Eyes of your Non Player Characters**.**

[9] http://wiki.gamedev.net/index.php/Libraries

[10] Revista Iberoamericana de Inteligencia Artificial. No.18 (2003), pp. 51-63. [Disponible en http**://**redalyc.uaemex.mx/redalyc/pdf/925/92501805.pdf]

[11] Schreiber, G.Knowledge Engineering and Management. The CommonKADS Methodology. The MIT Press, USA. (2002). [Citado en: Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.18 (2003), pp. 51-63.]

[12] Gómez A., Juristo N., Montes C. y Pazos J. 1997[Citado en: Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.18 (2003), pp. 51-63.]

[13] Analysis and Design of Multiagent Systems using MAS-CommonKADS. Carlos A. Iglesias, Mercedes Garijo, José C. González y Juan R. Velasco, 1998. [Citado en Revista Iberoamericana de Inteligencia Artificial. No.18 (2003), pp. 51-63.]

[14] Revista Iberoamericana de Inteligencia Artificial. No.18 (2003), pp. 51-63. [Disponible en http**://**redalyc.uaemex.mx/redalyc/pdf/925/92501805.pdf]

[**15]** Agile Project Management with Scrum, Ken Schwaber, Microsoft Press, January 2004, 163pp, ISBN 0-7356-1993-X [Citado en Revista Iberoamericana de Inteligencia Artificial. No.18 (2003), pp. 51-63.]

[16] Rational Unified Process(RUP). 2006. (2008). [Disponible en: https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducción%20a%20RUP.doc]

[17] http://www.acm.uiuc.edu/sigmil/RevEng/

[18] http://mayi.polanco.googlepages.com/TRABAJODEINGSOFTWAREII.doc

[19] Plataforma de Desarrollo [Disponible en]: http://www.babylon.com/definition/plataforma_de_desarrollo/Spanish

[20] http://www.aplicacionesempresariales.com/doxygen-documenta-el-codigo-por-ti.html

[21] "Business Modelling with UML" [Disponible en: http://teleformacion.uci.cu/mod/resource/view.php?id=11553]

[22] Scott W. Ambler, Introduction to UML 2 Package Diagrams. [Disponible en: http://teleformacion.uci.cu/mod/resource/view.php?id=14095]

[23] JACOBSON, Ivar; RUMBAUGH, James; BOOCH, Grady, "El proceso unificado de desarrollo".2004. Addison Wesley. Volumen II.

[24] JACOBSON, Ivar; RUMBAUGH, James; BOOCH, Grady, "El proceso unificado de desarrollo".2004. Addison Wesley. Volumen II.

BIBLIOGRAFIA CONSULTADA

McCarthy, John. **"WHAT IS ARTIFICIAL INTELLIGENCE",** [Online] 2007. [Disponible en: http://www-formal.stanford.edu/jmc/whatisai/]

An Introduction to the Science of Artificial Intelligence, [Online] 1997 [Disponible en: http://library.thinkquest.org/2705]**.**

Game AI, 2007 http://aigamedev.com/**.**

Gorniak, Peter. **"Beyond Behavior",** 2009 [Disponible en: http://www.ai-blog.net/]

Rabin, Steve. ***"The Challenge of Game AI in Next-Gen Games"*** 2006 [Disponible en: http://www.aiwisdom.com/]

Schreiner, Tim. "Artificial Intelligence in Game Design" [Disponible en: http://ai-depot.com/GameAI/Design.html]

Matthews, James "**How To Get Started with Gaming AI**" 2001 [Disponible en: http://www.generation5.org/content/2001/howto00.asp]

Alvarado, Gorge. **"Teoria de Juegos con Inteligencia Artificial**", 2007 [Disponible en: http://www.seccperu.org/?q=node/507]

Introducción a la Inteligencia Artificial [Disponible en: http://dmi.uib.es/~abasolo/intart/2-juegos.html]

Inteligencia Artificial en Videojuegos, 2008 [Disponible en: www.fing.edu.uy/inco/cursos/svti/assets/PPts/Inteligencia_Artificial_y_videojuegos]

Cortizo, Jose. "**Videojuegos: grandes retos para los Sistemas Inteligentes**" 2008 [Disponible en: http://weblogs.madrimasd.org/sistemas_inteligentes/archive/2008/04/24/89956.aspx]

GLOSARIO DE TERMINOS GENERALES

CASE: Ingeniería de Software Asistida por Ordenador.

FSA: Autómatas de Estados Finitos.

FSM: Maquinas de Estados Finitos.

FTPS: Juegos de Disparo

I+D: Investigación y Desarrollo.

IA: Inteligencia Artificial.

NPC: Personaje del juego.

OO: Orientado a Objeto.

RPG: Juegos de Rol.

RTS: Juegos de estrategia en tiempo real

RUP: Rational Unified Process.

RV: Realidad Virtual.

SB: Steering Behaviors.

SBC: Sistemas Basados en Conocimientos.

SMA: Sistemas Multi-Agentes.

STK: Scence Tolkit.

UML: Lenguaje Unificado de Modelado.

ANEXOS

Anexo 1: Foto de la Biblioteca en el juego Laberinto del Saber.

Anexo 2: Descripción Detallada de los Casos de Uso del Sistema.

- ✓ **Descripción detallada del Caso de Uso Gestionar Servicios.**

Caso de Uso:	Gestionar Servicios.
Actores:	Sistema
Resumen:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita registrar, eliminar o devolver información de un Servicio.
Precondiciones:	Se debe haber creado un mundo con anterioridad.
Referencias	**RF 2, RF 2.1, RF 2.2, RF 2.3**

Flujo Normal de Eventos	
Sección "Registrar Servicios**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema le especifica al Mundo el Servicio que quiere registrar.	1.1- Se guarda el Servicio en el Mundo según su prioridad.
Poscondiciones	Queda registrado un Servicio en el Mundo.
Sección "Eliminar Servicios**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a eliminar un Servicio.	1.1 -Busca el Servicio que se quiere eliminar. 1.2 - Elimina dicho Servicio.
Poscondiciones	Se elimina un Servicio del Mundo.
Sección "Devolver Información de un Servicio.**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a buscar información de un Servicio.	1.1 -Busca el Servicio del cual se quiere obtener la información. 1.2 - Devuelve la información del Servicio.
Poscondiciones	Se obtiene la información de un Servicio.

- ✓ **Descripción detallada del Caso de Uso Gestionar Información del Mundo.**

Caso de Uso:	Gestionar Información del Mundo.
Actores:	Sistema
Resumen:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita registrar, eliminar o buscar información obtenida del mundo en el Servicio de Percepción.
Precondiciones:	Se debe haber creado un mundo con anterioridad.
Referencias	**RF 3, RF 3.1, RF 3.2, RF 3.3**
Flujo Normal de Eventos	
Sección "Registrar información obtenida del ambiente**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a registrar una nueva información.	1.1 -Comprueba de que esta información no esté registrada. 1.2 -Registra esta información en el Servicio de Percepción.
Poscondiciones	Se registra información en el Servicio de Percepción.
Sección "Eliminar información obtenida del ambiente.**"**	
Acción del Actor	**Respuesta del Sistema**
1- El Sistema manda a eliminar una información.	1.1-Comprueba la existencia de la información a eliminar. 1.2-Elimina la información del Servicio de Percepción.
Poscondiciones	Se elimina la información del Servicio de Percepción.
Sección "Buscar información obtenida del ambiente.**"**	
Acción del Actor	**Respuesta del Sistema**

1-El Sistema manda a buscar la información que quiere.	1.1-Busca la información la cual se quiere obtener. 1.2- Devuelve la información.
Poscondiciones	Se obtiene la información deseada.

- ✓ **Descripción detallada del Caso de Uso Gestionar Servicios de Actualizar.**

Caso de Uso:	Gestionar Servicios de Actualizar.
Actores:	Sistema
Resumen:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita registrar, eliminar una actualización determinada o eliminar todas las actualizaciones del Servicio de Actualizar.
Precondiciones:	Se debe haber creado un mundo con anterioridad.
Referencias	**RF 4, RF4.1, RF 4.2, RF 4.3**
Flujo Normal de Eventos	
Sección "Registrar actualización.**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a registrar una nueva actualización.	1.1-Comprueba de que esta actualización no esté registrada. 1.2-Registra esta actualización en el Servicio Actualizar.
Poscondiciones	Se registra una actualización en el Servicio Actualizar.
Sección "Eliminar actualización.**"**	
Acción del Actor	**Respuesta del Sistema**

1- El Sistema manda a eliminar una actualización.	1.1-Comprueba la existencia de la actualización a eliminar. 1.2-Elimina la actualización especificada.
Poscondiciones	**Se elimina la actualización.**
Sección "Eliminar todas las actualizaciones.**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a eliminar todas las actualizaciones.	1.1-Elimina todas las actualizaciones existentes.
Poscondiciones	Quedan eliminadas todas las actualizaciones.

- ✓ **Descripción detallada del Caso de Uso Gestionar Componentes.**

Caso de Uso:	Gestionar Componentes.
Actores:	Sistema
Resumen:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita insertar, eliminar u obtener los componentes de un actor.
Precondiciones:	Se debe haber creado un mundo y un actor con anterioridad.
Referencias	**RF 5, RF5.1, RF 5.2, RF 5.3, RF 5.4**
Flujo Normal de Eventos	
Sección "Insertar componentes.**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a insertar un componente a un Agente.	1.1-Verifica de que el Agente no cuenta con este componente.

	1.2-Le adiciona el componente al Agente.
Poscondiciones	Se le adiciona un nuevo componente a un Agente.
Sección "Eliminar todos los componentes de un Agente.**"**	
Acción del Actor	**Respuesta del Sistema**
1- El Sistema manda a eliminar todos los componentes de un Agente.	1.1-Elimina todos los componentes de un Agente.
Poscondiciones	Se eliminan los componentes de un Agente.
Sección "Eliminar componente.**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a eliminar un componente determinado de un Agente.	1.1-Busca el componente que quiere eliminar. 1.2-Elimina este componente del Agente.
Poscondiciones	Se elimina el componente del Agente.
Sección "Obtener información de un componente.**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema especifica el componente del cual quiere obtener su información.	1.1-Busca el componente. 1.2-Devuelve la información de este componente.
Poscondiciones	Se obtiene información de un componente dado.

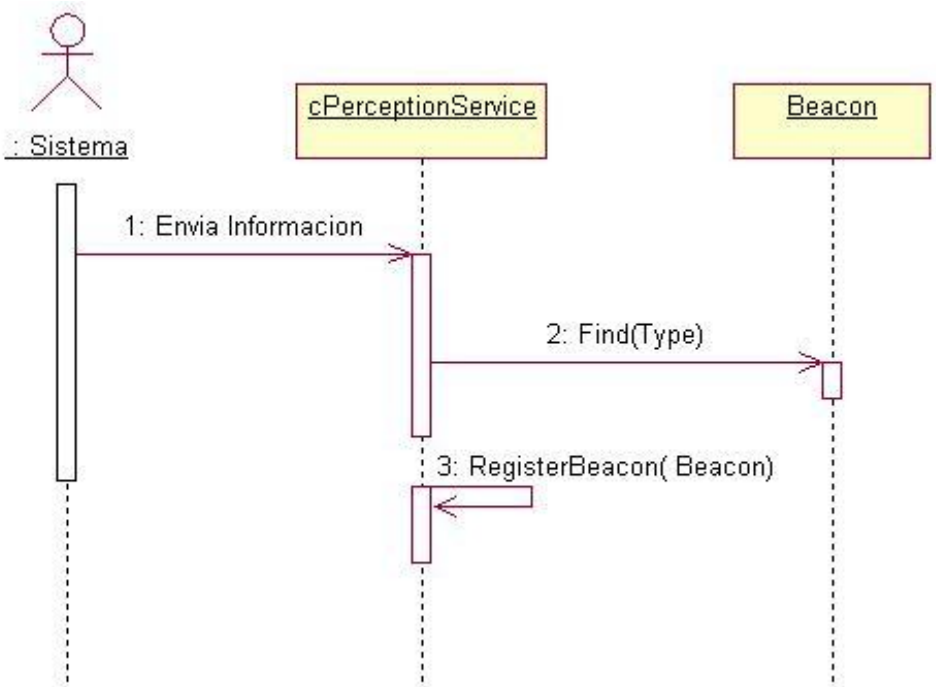
✓ **Descripción detallada del Caso de Uso Actualizar Mundo.**

Caso de Uso:	Actualizar Mundo.
Actores:	Sistema
Resumen:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita actualizar el Mundo.
Precondiciones:	Se debe haber creado un mundo con anterioridad.
Referencias	**RF 6, RF6.1, RF 6.2**
Flujo Normal de Eventos	
Sección "Actualizar información obtenida.**"**	
Acción del Actor	**Respuesta del Sistema**
1-El Sistema manda a que se actualicen los datos recogidos por los sensores.	1.1 -Mundo manda a actualizar datos al Servicio de Percepción. 1.2 -El Servicio de Percepción manda a los sensores a capturar información. 1.3 -Los sensores capturan la información. 1.4 -La información se guarda en el Servicio de Percepción, actualizando la información.
Poscondiciones	Quedan actualizados los datos que recogen los sensores del entorno.

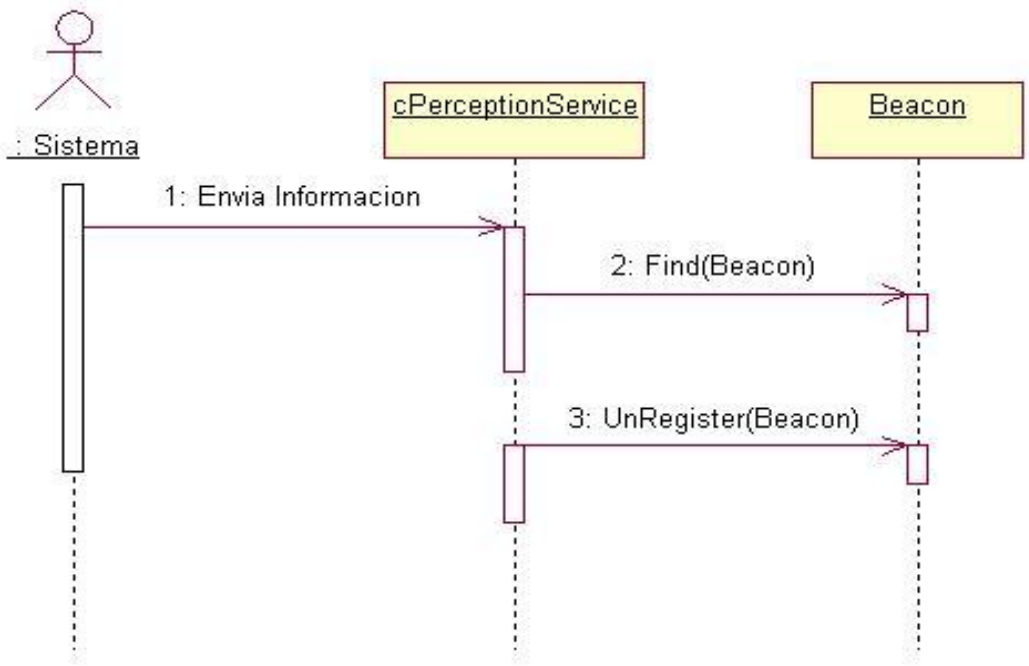
Sección "Actualizar Servicios de Pensar.**"**	
Acción del Actor	**Respuesta del Sistema**
1- El Sistema manda a actualizar el Servicio de Pensar.	1.1-Mundo manda a actualizar el Servicio Pensar. 1.2- El Servicio Pensar manda a los Agentes a decidir que comportamiento tomar. 1.3– Los Agentes toman un comportamiento.
Poscondiciones	Se verifica que el Agente esté pensando bien.

Anexo 3: Diagramas de Secuencias.

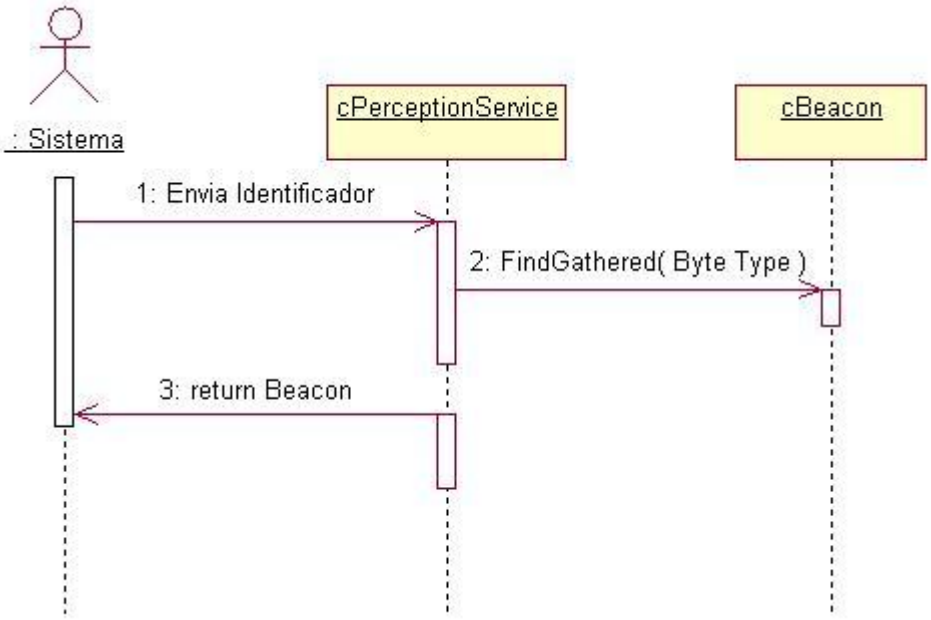
- **Diagrama de Secuencia: "Registrar información obtenida del ambiente"**



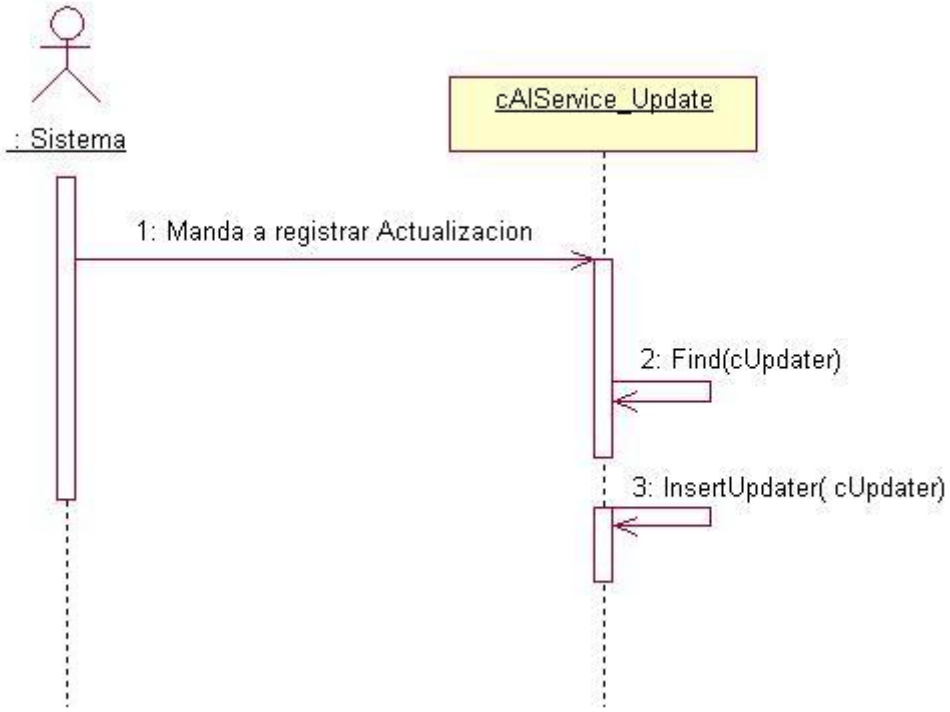
- **Diagrama de Secuencia: "Eliminar información obtenida del ambiente."**



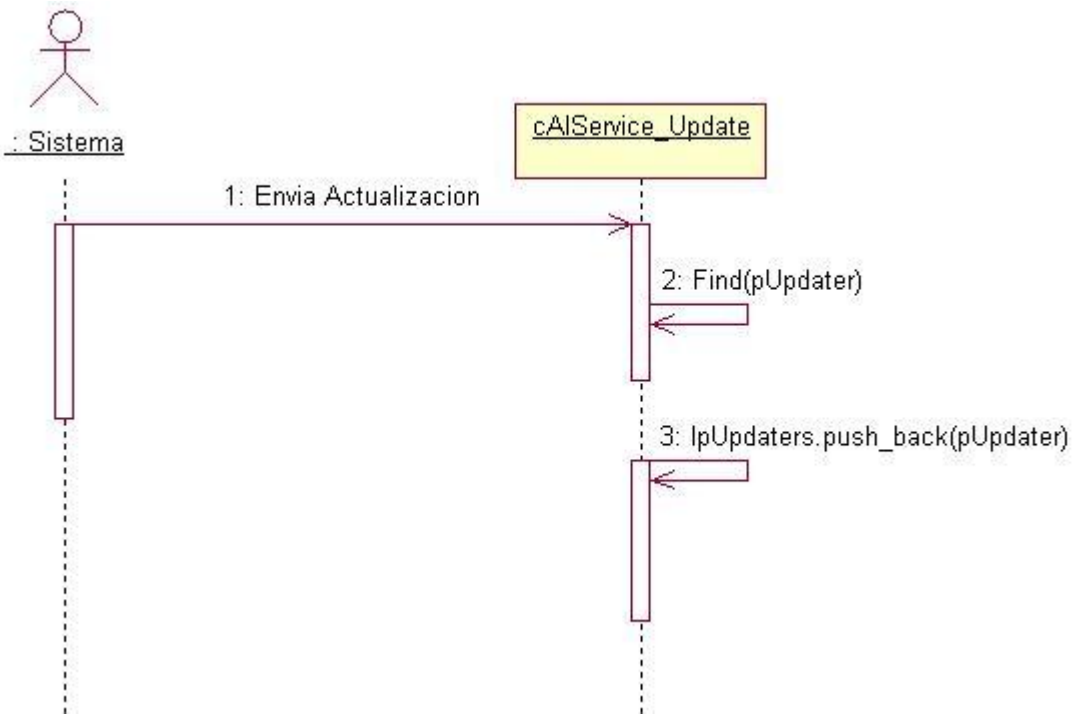
- **Diagrama de Secuencia: "Buscar información obtenida del ambiente."**



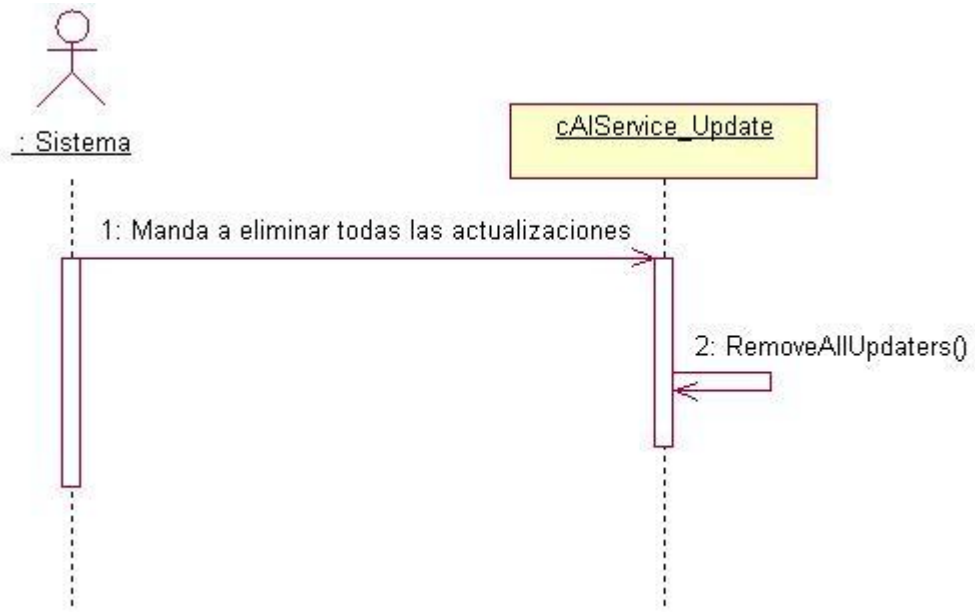
- **Diagrama de Secuencia: "Registrar actualización."**



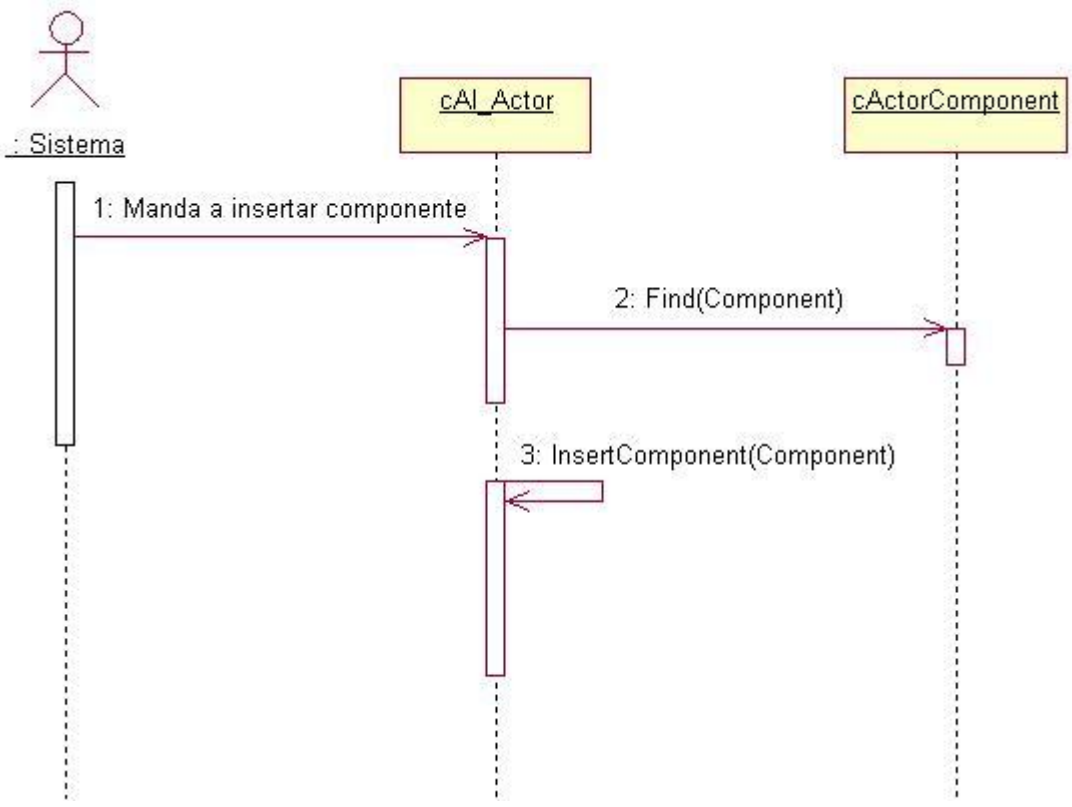
- **Diagrama de Secuencia: "Eliminar actualización."**



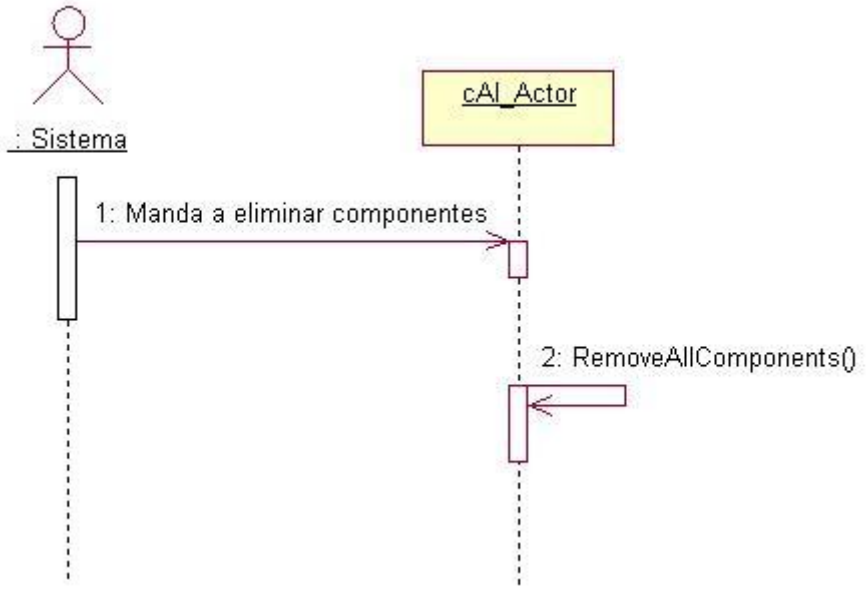
- **Diagrama de Secuencia: "Eliminar todas las actualizaciones."**



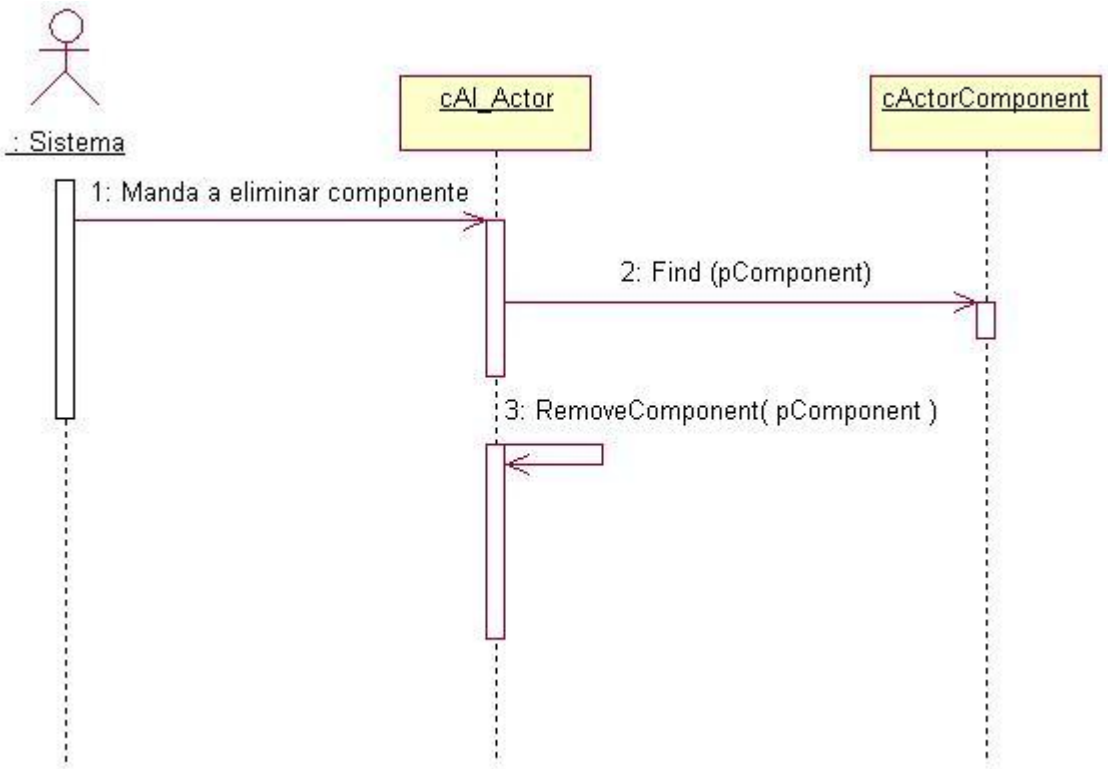
- **Diagrama de Secuencia: "Insertar componentes."**



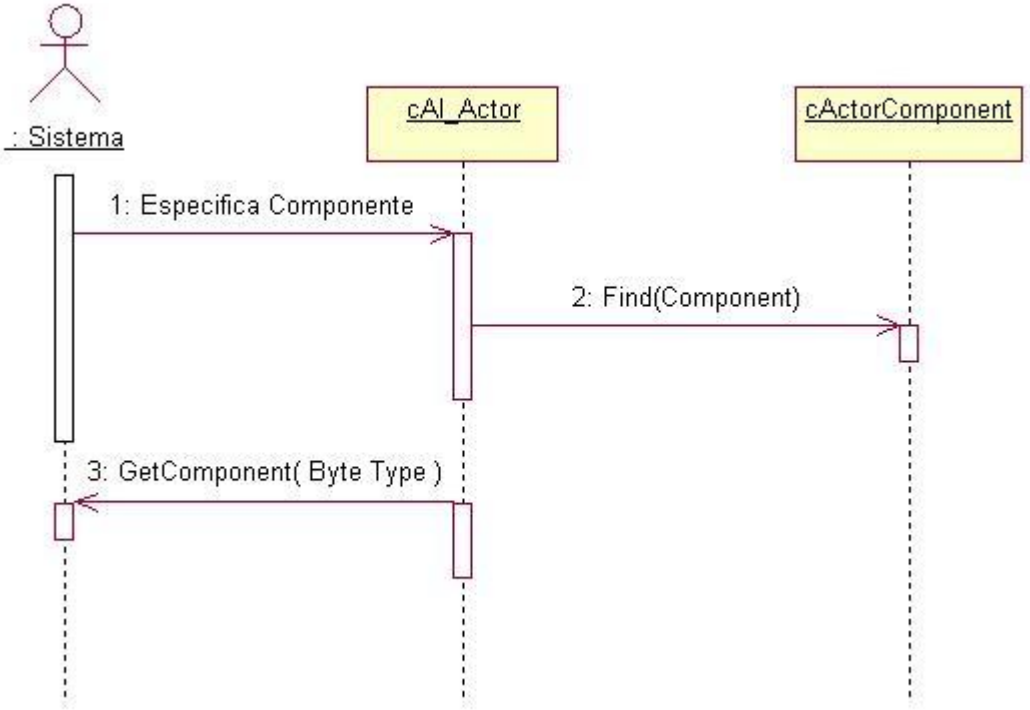
- **Diagrama de Secuencia: "Eliminar todos los componentes de un actor."**



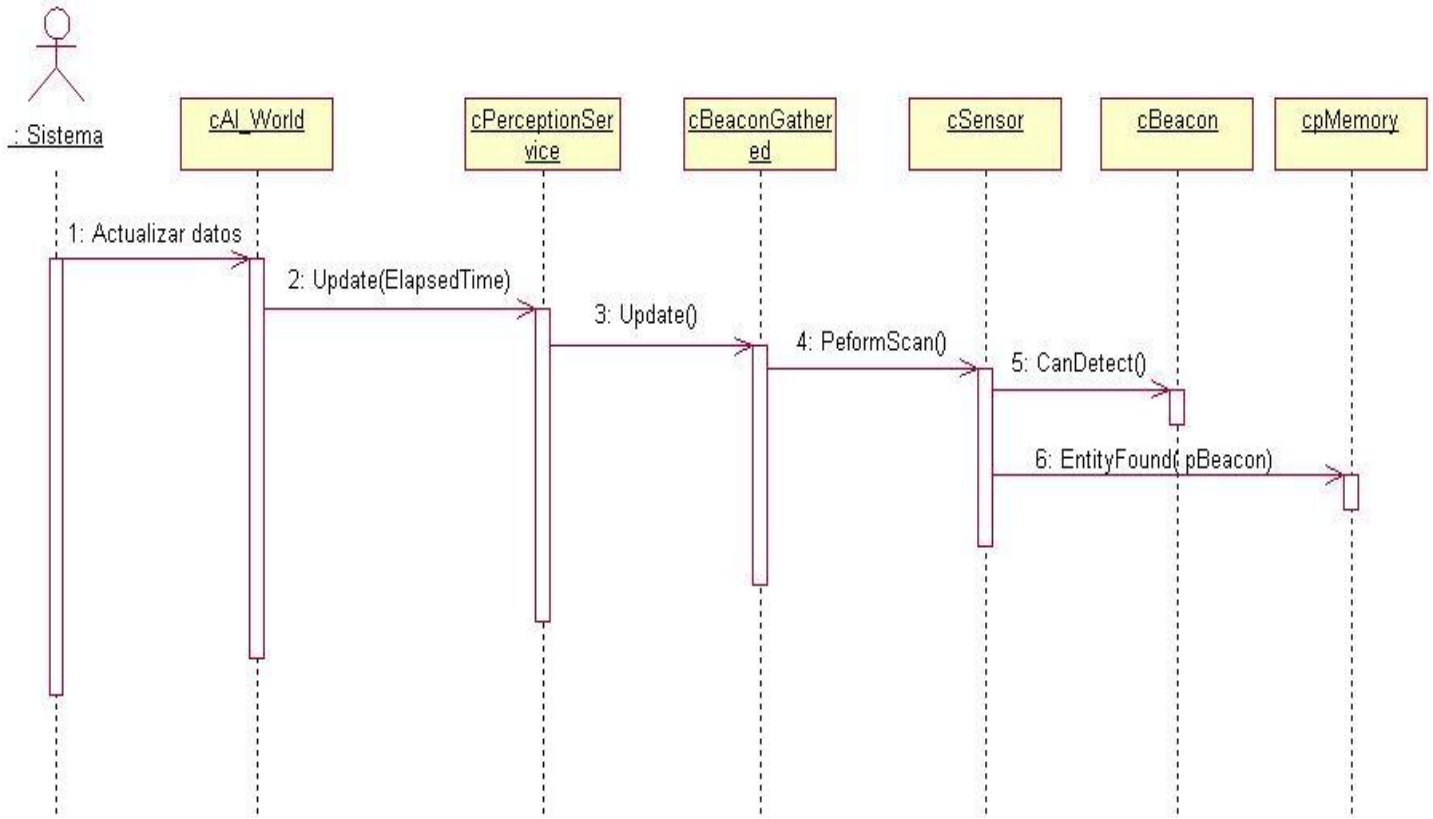
- **Diagrama de Secuencia: "Eliminar componente."**



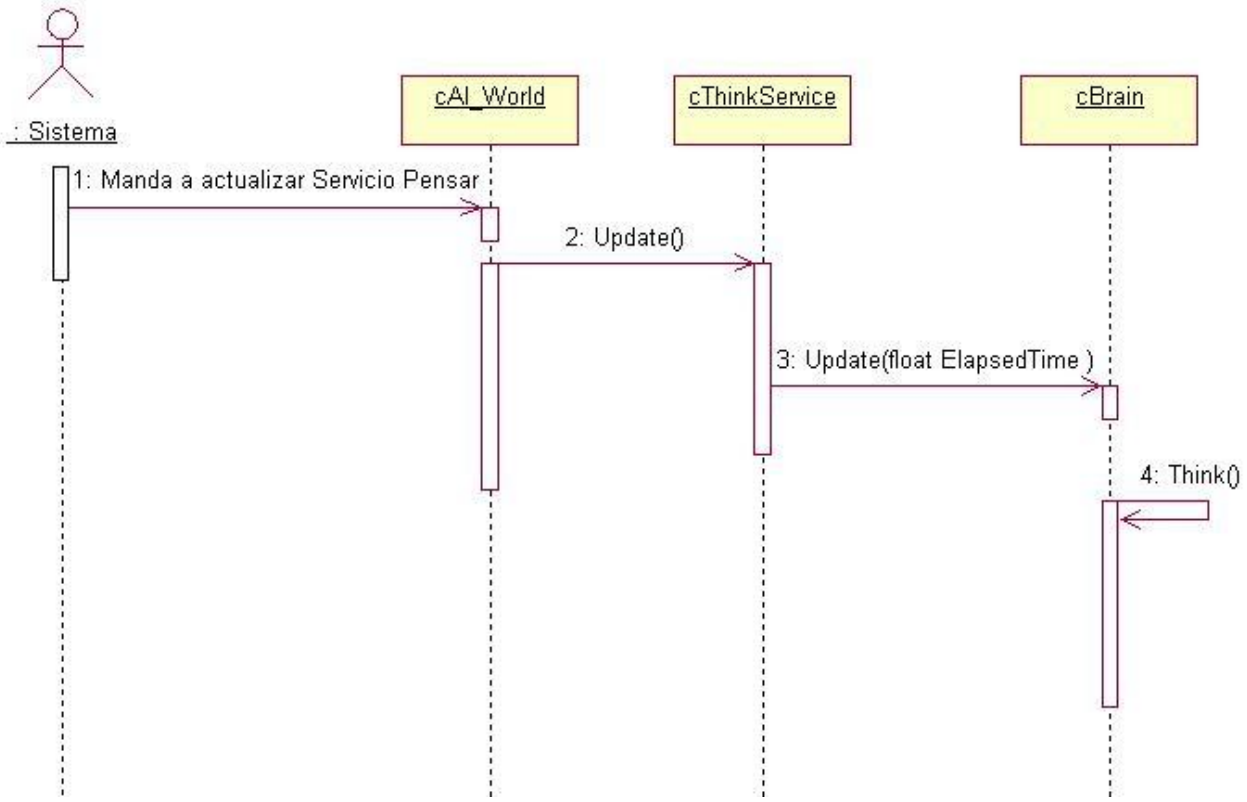
- **Diagrama de Secuencia: "Obtener información de un componente."**



- **Diagrama de Secuencia: "Actualizar información obtenida."**



- **Diagrama de Secuencia: "Actualizar Servicios de Pensar."**



Anexo 4: Sitio generado por Doxygen.

Documentación de las funciones miembro

void cAI_World::Update (float *ElapsedTime*) [virtual]

Metodo que actualiza todos los elementos que se encuentran en el Mundo. \

Autor:
Facultad #5, Universidad de las Ciencias Informaticas \

Fecha:
Curso 2008-2009

void cAI_World::RegisterActor (cAI_Actor * *pActor*) [virtual]

Metodo para Registrar un Actor en el Mundo. \

Autor:
Facultad #5, Universidad de las Ciencias Informaticas \

Fecha:
Curso 2008-2009

Anexo 5: Descripción detallada de las clases.

	Nombre de la Clase: cAI_Actor		
	Descripción: Esta clase implementa las funciones que puede realizar un Actor.		
	Tipo	**Nombre**	**Descripción**
Atributos	list<cActorComponent*>	lpComponents	Estructura que guarda todos los Componentes de cada Actor.
	cVariableHolder	Datas	Información: (Max. velocidad, Vida, etc).
	void*	pOwnerEntity	Define que tipo de Actor es, ejemplo: Jugador, Item, etc.
Métodos	virtual void	Update	Método para actualizar el Actor en un instante de tiempo determinado.
	virtual void	RegisterIn	Método para Registrar al Actor en un Mundo.
	virtual void	UnRegisterFrom	Método para Eliminar al Actor de un Mundo.
	virtual void	Enable	Método que Activa al Actor y sus Componentes.
	virtual void	Disable	Método que Desactiva al Actor y sus Componentes.
	virtual void	InsertComponent	Método que le adiciona un nuevo componente al Actor si este no lo tiene todavía.
	virtual cActorComponent*	GetComponent	Método utilizado para obtener toda la información de un Componente especificado.
	virtual void	RemoveAllComponents	Método que borra todos los Componentes del Actor.
	virtual void	RemoveComponent	Método que borra un Componente especificado.
	virtual void	RemoveComponent	Método que borra el Componente que coincide con el identificador.
	Herencia		
	Dependencia		cActorComponent, cVariableHolder.

	Nombre de la Clase: cActorComponent		
	Descripción: Clase genérica que define las funciones generales de todos los Componentes de los Actores.		
	Tipo	**Nombre**	**Descripción**
Atributos			
Métodos	virtual void	Register	Método para Registrar un nuevo Componente.
	virtual void	UnRegister	Método utilizado para Eliminar un Componente.
	virtual Byte	GetType	Método utilizado para devolver el tipo de Componente.
	virtual bool	HasType	Método utilizado para comprobar el tipo del Componente.
	virtual void	Enable	Método para activar el Componente.
	virtual void	Disable	Método para desactivar el Componente.
Herencia			
Dependencia	cAI_Actor, cAI_World.		

	Nombre de la Clase: cBrain		
	Descripción: Representa el componente del Actor. Dirige todo el funcionamiento que lleva el Actor durante el transcurso del juego.		
	Tipo	**Nombre**	**Descripción**
Atributos	Byte	BrainFlags	Bandera que nos dice el estado en que se encuentra el Cerebro.
Métodos	virtual Byte	Think	Método principal del componente Cerebro. Analiza los datos con los que cuenta el Actor y determina que hacer en cada momento además de verificar de que el Actor este haciendo lo correcto en cada situación.

virtual void	Enable	Método que activa el Cerebro.
virtual void	Disable	Método que desactiva el Cerebro.
inline bool	IsEnable	Método que nos dice si el Cerebro esta activado o no.
virtual void	Register	Método que utilizado para registrar el Cerebro junto con el Actor al cual pertenece en la clase cThinkService.
virtual void	UnRegister	Método que utilizado para dar de baja al Cerebro junto con el Actor al cual pertenece en la clase cThinkService.
virtual Byte	GetType	Método que devuelve el Byte que identifica al Cerebro dentro de los componentes del Actor
Herencia		cActorComponent
Dependencia		cActorComponent, cThinkService, cAI_World, AI_constants.

	Nombre de la Clase: cpBasicMovil		
	Descripción: Representa los el movimiento del Actor por el mundo, ella controla la velocidad, impulso, etc.		
	Tipo	**Nombre**	**Descripción**
Atributos	float	MaxSpeed	Máxima velocidad que alcanza el Actor.
	cVector3f	Velocity	Velocidad de movimiento.
Métodos	virtual void	ApplyImpulse	Método que aplica un impulso al movimiento que lleva el Actor.
	virtual void	Update	Método que pone en movimiento al Actor durante un tiempo determinado
	virtual void	SetVelocity	Método que cambia la velocidad del Actor.
	virtual cVector3f	GetVelocity	Método que devuelve la velocidad que lleva el Actor dentro del Mundo.
	virtual float	GetMaxSpeed	Método que devuelve la máxima velocidad que puede tomar el Actor dentro del Mundo.

virtual void	SetMaxSpeed	Método que cambia la máxima rapidez que puede tomar el Actor dentro del Mundo.
virtual void	Register	Método que le asigna este movimiento a un Actor.
virtual void	UnRegister	Método que le retira este movimiento a un Actor.
Herencia		cpMovil.
Dependencia		cpMovil, cVector3f.

	Nombre de la Clase: cpMovil		
	Descripción: Representa los movimientos del Actor por el mundo, ella controla la velocidad, impulso, etc. No maneja la información de dirección u orientación dentro del Mundo.		
	Tipo	**Nombre**	**Descripción**
Atributos	float	MaxSpeed	Rapidez que puede alcanzar el Actor dentro del Mundo.
Métodos	virtual void	ApplyImpulse	Método que aplica un impulso al movimiento que lleva el Actor.
	virtual void	Update	Método que actualiza el movimiento que lleva el Actor.
	virtual void	SetVelocity	Método que cambia la velocidad del Actor.
	virtual cVector3f	GetVelocity	Método que devuelve la velocidad que lleva el Actor dentro del Mundo.
	virtual float	GetMaxSpeed	Método que devuelve la máxima rapidez que puede tomar el Actor dentro del Mundo.
	virtual void	SetMaxSpeed	Método que cambia la máxima rapidez que puede tomar el Actor dentro del Mundo.
	virtual bool	HasType	Metodo que verifica de que tipo es el movimiento.
Herencia		cpLocation	
Dependencia			

	Nombre de la Clase: cpLocation		
	Descripción: Clase que representa uno de los componentes del Actor. Maneja toda la información del Actor correspondiente a su posición en el Mundo así como el cambio de lugar, la rotación, la orientación y otras tareas de movimiento.		
	Tipo	**Nombre**	**Descripción**
Atributos	cMatrix4f	Matrix	Atributo principal que define un objeto en el espacio.
	Byte	Flags	Atributo que define el tipo de movimiento por el que pasa el Actor.
Métodos	virtual cVector3f	GetPosition	Método que devuelve la posición donde se encuentra el Actor.
	virtual void	SetPosition	Método que cambia la posición del Actor
	virtual void	MoveTo	Método que mueve al Actor hacia una posición especificada, es posible que en el movimiento pueda ocurrir una colisión.
	virtual void	MoveWs	Método que a partir del vector que define la posición del Actor se le suma el vector Deltha pasado por parámetro, por lo que el resultado de ejecutar este método es una nueva posición del Actor.
	virtual void	RotateBd	Método utilizado para la rotación del Actor a partir de su posición, sin cambiar esta el Actor rota.
	virtual void	RotateWd	Método que rota al Mundo junto con todos sus elementos excepto el Actor al que le corresponde este componente.
	virtual void	Orient	Método que orienta al Actor hacia una posición de acuerdo con los vectores de Arriba y Adelante.
	virtual void	SetLocation	Método que cambia la matriz del Actor y lo cambia de lugar como si fuera teletransportación.
	virtual void	MoveToLocation	Método que cambia la matriz del Actor y lo cambia de lugar siguiendo una trayectoria.
	virtual Byte	GetType	Método que devuelve el Byte que identifica la Localización dentro de los Componentes del Actor.

virtual bool	HasType	Método que verifica el Byte que identifica a la Localización con el especificado por parámetros.
Herencia		cActorComponent
Dependencia		cMatrix4f, cQuaternionf.

	Nombre de la Clase: cBrain_Cluster		
	Descripción: Clase que contiene todos los Cerebros creados.		
	Tipo	**Nombre**	**Descripción**
Atributos	list<cBrain*>	lpBrains	Lista de Cerebros.
Métodos	virtual Byte	Think	Método principal del componente Cerebro. Analiza los datos con los que cuenta el Actor y determina que hacer en cada momento además de verificar de que el Actor este haciendo lo correcto en cada en cada situación.
	virtual void	Enable	Método que activa el Cerebro.
	virtual void	Disable	Método que desactiva el Cerebro.
Herencia		cBrain	
Dependencia			

	Nombre de la Clase: cBrain_Steered		
	Descripción: Esta clase dentro del Cerebro es la encargada de dirigir el Comportamiento (Steering Behaviors) según los datos recolectados por los sensores, define que Comportamiento a seguir durante el transcurso del juego.		
	Tipo	**Nombre**	**Descripción**
Atributos	cSteeringBehavior*	pSteeringBehavior	Puntero a la clase SteeringBehavior.
	cpMovil*	pMovil	Puntero a la clase Móvil.
	Byte	SteeringFlags	Byte que define el estado en que esta (activado (1), desactivado (0)).

Métodos	virtual Byte	Think	Método principal del componente Cerebro. Analiza los datos con los que cuenta el Actor y determina que hacer en cada momento además de verificar de que el Actor este haciendo lo correcto en cada situación.
Herencia		cBrain	
Dependencia		cSteeringBehavior, cpMovil.	

	Nombre de la Clase: cBrain_Scripted		
	Descripción: Clase que se encarga de verificar y cargar los cambios que puedan ocurrir en los Script que se dejan para poder modificar cierta información de los Actores.		
	Tipo	**Nombre**	**Descripción**
Atributos	cScriptRunner*	pRunner	Instancia de la clase cScriptRunner.
	cScriptVariables	var	Instancia de la clase cScriptVariables.
	cScriptPlayer*	pScriptPlayer	Instancia de la clase cScriptPlayer.
	bool	bStarted	Variable que controla la verificación de los cambios.
Métodos	virtual Byte	Think	Método principal de esta clase encargado de realizar las operaciones de verificar y cargar los cambios.
	virtual void	Enable	Método que Activa este componente.
	virtual void	Disable	Método que desactiva este componente.
Herencia		cBrain	
Dependencia		cScriptPack, cScriptRunnerBehavior, cAI_Actor.	

	Nombre de la Clase: cBrainArbiter		
	Descripción: Clase encargada de manejar el orden en que cada Cerebro del Actor va a trabajar y realizar sus funciones.		
	Tipo	**Nombre**	**Descripción**
Atributos	list<cEvaluatedGoal>	lpGoals	Lista de pareja Cerebro-Evaluador.
	list<cGoalEvaluator*>	lpEvaluators	Lista de Evaluador para cada Cerebro.
Métodos	virtual Byte	Think	Método principal del componente Cerebro. Analiza los datos con los que cuenta el Actor y determina que hacer en cada momento además de verificar de que el Actor este haciendo lo correcto en cada situación. En este caso en particular además define el orden en que van a funcionar los Cerebros.
	virtual void	AddSubGoal	Método que adiciona un nuevo elemento en la Lista Cerebro-Evaluador.
	virtual void	RemoveAllSubGoals	Método que elimina todos los elementos de la Lista Cerebro-Evaluador.
	void	Arbitrate	Método que recorre la Lista de Evaluador y según el resultado mantiene actualizada y ordenada la Lista de Cerebro-Evaluador logrando así el orden adecuado en que trabajan los Cerebros.
	virtual void	AddEvaluator	Método que adiciona un nuevo elemento en la Lista Evaluador.
Herencia		cBrain	
Dependencia		cGoalEvaluator.	

	Nombre de la Clase: cGoalEvaluator		
	Descripción: Clase encargada de calcular el peso o la importancia de un Cerebro.		
	Tipo	**Nombre**	**Descripción**
Atributos	float	CharacterBias	Predisposición del personaje.

Métodos	virtual float	CalculateDesirability	Método que calcula la prioridad de los Cerebros con los que cuenta cada Actor, para definir el orden en que cada uno de ellos debe Pensar.
	virtual cBrain*	GetBrain	Método que devuelve el Cerebro con mayor prioridad.
Herencia			
Dependencia		cBrain.	

	Nombre de la Clase: cBrain_FSM		
	Descripción: Clase que representa la parte del cerebro encargada de manejar los Estados por los que pasa el Actor durante el transcurso del juego así como las Transiciones de un Estado a otro.		
	Tipo	**Nombre**	**Descripción**
Atributos	vector<cBrain_FSM_State*>	apStates	Conjunto de Estados.
	list<Word>	lpPreviousState_Heap	Lista ordenada de Estados por los que ha paso el Actor.
	Word	iActualState	Estado actual en el que se encuentra el Actor.
	cBrain_FSM_TriggerTable	TriggerTable	Objeto de tipo cBrain_FSM_TriggerTable.
Métodos	virtual Byte	Think	Método que principal del componente Cerebro. Determina el funcionamiento del Cerebro y comprueba el buen funcionamiento del mismo.
	bool	MoveToState	Método que lleva al Actor hacia un Estado determinado. Actualiza la lista de Estos previos.
	bool	MoveToPreviousState	Método que lleva al Actor hacia el Estado.
Herencia		cBrain	
Dependencia		cBrain_FSM_State, cBrain_FSM_TriggerTable.	

	Nombre de la Clase: cBrain_FSM_State		
	Descripción: Clase que representa los estados por los que puede transitar un Actor		
	Tipo	**Nombre**	**Descripción**
Atributos	list<cBrain_FSM_Transition*>	lpTransitions	Lista de transiciones asociadas a este estado.
Métodos	virtual void	OnArrive	Método que ejecuta las acciones pertinentes una vez el Actor llegue a este estado.
	virtual void	OnLeave	Método que ejecuta las acciones pertinentes una vez el Actor salga de este estado.
	virtual Byte	OnUpdate	Método que busca todas las transiciones que están asociadas a este estado y le aplica la acción correspondiente a la misma, esto da lugar que el Actor pueda cambiar de estado a estado.
	virtual void	EnableAllTransitions	Método que pone como activadas todas las transiciones que llegan a este estado.
	virtual void	DisableAllTransitions	Método que pone como desactivadas todas las transiciones que llegan a este estado.
Herencia			
Dependencia		cBrain_FSM, cBrain_FSM_Transition.	

	Nombre de la Clase: cBrain_FSM_TriggerTable		
	Descripción: Clase que representa un conjunto de las zonas calientes del juego, lugar donde se ejecuta una acción cuando un Actor ejecuta cierta acción,(Llegar a un lugar determinado, que se agache,etc).		
	Tipo	**Nombre**	**Descripción**
Atributos	vector<cPair_Trigger_Bool>	aTriggers	Lista de zonas calientes.

Métodos	bool	Update	Método que verifica si ha ocurrido algún cambio, algún evento en el mundo donde se desarrolla el juego que pueda activar una zona caliente.
	void	EnableTrigger	Método que activa una zona caliente.
	void	DisableTrigger	Método que desactiva una zona caliente.
	void	ClearAll	Método que desactiva o ignora todas las acciones o eventos que pueden activar una o todas las zonas calientes.
	bool	IsTriggerLaunched	Método que busca si una zona caliente tiene algún evento que la pueda activar.
	void	ClearTrigger	Método que desactiva o ignora todas las acciones o eventos que pueden activar una zona caliente.
	bool	SetTriggerLaunched	Método que activa todas las acciones o eventos que pueden activar una zona caliente.
	void	UseTrigger	Método que representa el suceso cuando es activada una zona caliente.
	void	SetMaxNumberTrigger	Método que cambia el número máximo de zonas calientes que puede haber en un mundo.
Herencia		cEventCatcher	
Dependencia		cTrigger	

	Nombre de la Clase: cBrain_FSM_Transition		
	Descripción: Clase que representa las transiciones o eventos que deben ocurrir para que un Actor llegue a cierto estado.		
	Tipo	**Nombre**	**Descripción**
Atributos			
Métodos	virtual void	Enable	Método que pone a la transición como activada o como que ya ocurrió el evento al cual representa la transición.
	virtual void	Disable	Método que pone a la transición como desactivada.
	virtual bool	Apply	Método que le aplica una acción a la transición que este asociado a esta transición.
	virtual bool	HasLaunched	Método que verifica si cierta transición está asociada a esta o tiene cierta dependencia.
Herencia			
Dependencia		cBrain_FSM	

	Nombre de la Clase: cBrain_FSM_State_Brain		
	Descripción: Clase que representa los estados por los que puede transitar un Cerebro.		
	Tipo	**Nombre**	**Descripción**
Atributos	cBrain*	pBrain	Cerebro al que pertenece el estado.
Métodos	virtual Byte	OnUpdate	Método que busca todas las transiciones que están asociadas a este estado y le aplica la acción correspondiente a la misma esto da lugar que el Cerebro pueda cambiar de estado a estado.
	virtual void	OnArrive	Método que ejecuta las acciones pertinentes una vez el Cerebro llegue a este estado.

virtual void	OnLeave	Método que ejecuta las acciones pertinentes una vez el Cerebro salga de este estado.
Herencia	cBrain_FSM_State	
Dependencia	cBrain.	

	Nombre de la Clase: cBrain_FSM_State_Scripted		
	Descripción: Clase encargada de la gestión de los cambios hechos por los Script para la modificación de los estados.		
	Tipo	**Nombre**	**Descripción**
Atributos	cScriptPack*	pPack	Instancia de la clase cScriptPack.
	Word	iScript	Identificador del Script.
	cScriptRunnerBehavior*	pScriptBehavior	Instancia de la clase cScriptRunnerBehavior.
	cScriptVariables	var	Instancia de la clase ScriptVariables.
	cScriptPlayer*	pScriptPlayer	Instancia de la clase cScriptPlayer.
Métodos	virtual void	OnArrive	Método que ejecuta las acciones pertinentes una vez el Scrpit llegue a este estado.
	virtual void	OnLeave	Método que ejecuta las acciones pertinentes una vez el Script salga de este estado.
Herencia		cBrain_FSM_State	
Dependencia		cScriptPack, cScriptRunnerBehavior	

Nombre de la Clase: cBrain_FSM_Transition_Triggered		
Descripción: Clase que representa las transiciones o eventos que deben ocurrir para que se active una zona caliente o esta cambie de estado.		
Tipo	**Nombre**	**Descripción**

Atributos	Word	iTrigger	Identificador de la zona caliente.
	Word	iNewState	Identificador del estado al que da lugar esta transición.
Métodos	virtual void	Enable	Método que pone a la transición como activada o como que ya ocurrió el evento al cual representa la transición.
	virtual void	Disable	Método que pone a la transición como desactivada.
	virtual bool	Apply	Método que le aplica cierta acción a la transición que este asociado a esta transición.
	virtual bool	HasLaunched	Método que verifica si cierta transición está asociada a esta o tiene cierta dependencia.
Herencia		cBrain_FSM_Transition	
Dependencia			

	Nombre de la Clase: cAI_Service		
	Descripción: Representa los Servicios que presta el Mundo a los Actores.		
	Tipo	**Nombre**	**Descripción**
Atributos	Byte	Type	Identificador de cada Servicio.
	Word	Priority	Prioridad de cada Servicio.
Métodos	virtual void	Update	Método que actualiza todos los Servicios y así el todo lo que el Mundo contiene.
Herencia			
Dependencia		cAI_World	

	Nombre de la Clase: cDataGathered		
	Descripción: Esta clase representa toda la información que los Sensores toman del Mundo donde se encuentra en Actor.		
	Tipo	**Nombre**	**Descripción**
Atributos	Byte	Type	Identificador de la Información.
Métodos	virtual void	Update	Método que no se implementa aquí, sino en sus hijos, utilizado para que los Sensores vuelvan a recolectar información del ambiente.
Herencia			
Dependencia		cMemorySlot	

	Nombre de la Clase: cpMemory		
	Descripción: Esta clase representa el lugar físico donde se guarda la información recogida por los Sensores.		
	Tipo	**Nombre**	**Descripción**
Atributos	Byte	ReportEntitiesFlag	Identificador de las Entidades.
	Word	ReportMinimumPriority	Prioridad de la Información guardada.
	void*	pOwner	Para no guardar información de sí mismo.
	list<cMemorySlot*>	lpMemorySlots	Lista de espacios de memorias.
Métodos	virtual Byte	GetType	Método que devuelve el número que representa a la Memoria.
	virtual bool	HasType	Método que verifica el número que representa a la Memoria.
	virtual void	GetAllPerceivedEntities	Método que guarda en una lista todas las Entidades percibidas y guardadas en la memoria corta.

	virtual void	ReportEntities	Método que cambia una entidad guardada en la Memoria por otra.
	virtual void	StopReportingEntities	Método que borra la información guardada en la Memoria.
	virtual void	SetAwarenessLevel	Método que cambia la prioridad de la información guardada en la Memoria.
	void	EntityFound	Método que busca una Entidad dentro de las memorias corta, si la encuentra la borra y si no este la crea con todos los datos pasados por parámetro.
	void	EntityDisappear	Método que busca y elimina una Entidad.
	virtual cMemorySlot*	FindSlot	Método que busca un espacio de memoria donde se encuentra la Entidad especificada.
	void	RemoveDisappearedEntities	Método que borra todas las entidades de las memorias cortas.
Herencia		cActorComponent	
Dependencia		cMemorySlot.	

	Nombre de la Clase: cSensor		
	Descripción: Esta clase representa los Sensores que posee cada Actor para obtener información del mundo.		
	Tipo	**Nombre**	**Descripción**
Atributos	Word	MinPriority	Prioridad de cada Sensor.
	Word	ScanEntitiesFlags	Elementos que se pueden detectar (Fuego de un arma, ruidos de pasos, cadáveres).
	cpMemory*	pMemory	Memoria donde se guarda lo detectado.

Métodos	virtual bool	CanDetect	Método que analiza lo detectado y procesa la información
Herencia			
Dependencia		cpMemory,cBeacon.	

	Nombre de la Clase: cSensorSurround		
	Descripción: Esta clase representa los Sensores de Sonido que posee cada Actor para obtener información del mundo.		
	Tipo	**Nombre**	**Descripción**
Atributos	float	Radius	Radio de escucha.
	cMatrix4f*	pLocation	Localización dentro del juego.
Métodos	virtual bool	CanDetect	Método que analiza lo detectado y procesa la información.
Herencia		cSensor	
Dependencia		cMatrix4f	

	Nombre de la Clase: cSensorFrustum		
	Descripción: Esta clase representa los Sensores de Visibilidad que posee cada Actor para obtener información del mundo.		
	Tipo	**Nombre**	**Descripción**
Atributos	cFrustum*	pFrustum	Objeto detectado.
Métodos	virtual bool	CanDetect	Método que analiza lo detectado y procesa la información.
Herencia		cSensor	
Dependencia		cFrustum	

	Nombre de la Clase: cPerceptionService		
	Descripción: Clase que representa el servicio de Percepción, técnica utilizada por los Actores.		
	Tipo	**Nombre**	**Descripción**
Atributos	vector<cDataGathered*>	apGathereds	Lista de información obtenida a través de los Sensores.
Métodos	virtual void	Update	Método manda a todos los sensores que vuelvan a tomar información del Entorno, y así tener nueva información registrada.
	Virtual cDataGathered*	FindGathered	Método que devuelve una información recogida por los sensores según su identificador.
	virtual void	RegisterGathered	Método que registra una nueva información.
	virtual void	UnRegisterGathered	Método que elimina una Información especificada.
	virtual void	UnRegisterGathered	Método que elimina una Información especificada.
Herencia		cAI_Service	
Dependencia		cDataGathered.	

	Nombre de la Clase: cMemorySlot		
	Descripción: Representa el lugar fisico donde se guarda la informacion recogida por los Sensores, representa una porcion de la Memoria.		
	Tipo	**Nombre**	**Descripción**
Atributos	void*	pEntity	Entidad guardad en la Memoria corta.
	Word	BasePriority	Prioridad de la Entidad guardada.
	Word	EntityTypeFlags	Tipo de Entidad guardada.
	float	HowLongWasVisible	Numero que representa la visibilidad de la Entidad.
	float	TimeOfLastPerception	Hora de la última Percepción.
	cVector3f	vLastPosition	Ultima posición de la Entidad.
	cVector3f	vLastVelocity	Ultima velocidad de la Entidad.
Métodos	Word	GetPriority	Método que devuelve la prioridad de la Entidad guardada.
Herencia			
Dependencia		cVector3f	

	Nombre de la Clase: cBeaconGathered		
	Descripción: Clase que representa un Evento donde se encuentra el Actor y este toma toda la información de este Evento, (Evento de interés para el Actor).		
	Tipo	**Nombre**	**Descripción**
Atributos	list<cSensor*>	lpSensors	Lista de Sensores.
	list<cBeacon*>	lpStaticBeacons	Lista de eventos estáticos.
	list<cBeacon*>	lpDynamicBeacons	Lista de eventos dinámicos.
Métodos	virtual void	Update	Método que manda a todos los Sensores a buscar información del Evento.

virtual void	RegisterSensor	Método que Registra un nuevo Sensor.
virtual void	UnRegisterSensor	Método que elimina un Sensor especificado.
void	RelocateSensors	Método que reorganiza los Sensores.
virtual void	RegisterBeacon	Método que Registra un nuevo Evento.
virtual void	UnRegisterBeacon	Método que elimina un Evento especificado.
void	RelocateBeacons	Método que reorganiza los Eventos.
void	PeformScan	Método que manda a un Sensor en especifico a recolectar Información, este método es utilizado en el UpDate.
Herencia	cDataGathered	
Dependencia	cBeacon, cSensor.	

	Nombre de la Clase: cThinkService		
	Descripción: Clase que representa el Servicio Pensar, esta es una de las cualidades que debe tener un Actor.		
	Tipo	**Nombre**	**Descripción**
Atributos	map<cAI_Actor*, cBrain*>	mpBrains	Lista de de Actores relacionados con sus Cerebros.
Métodos	virtual void	Update	Método que manda a pensar a todos los Actores del Mundo para que puedan tener cada cierto tiempo diferentes comportamientos según las circunstancias y lograr así un comportamiento razonable.
	virtual void	InsertBrain	Método que inserta un nuevo Actor con su Cerebro.

virtual void	RemoveBrain	Método que elimina un Actor con su Cerebro.
Herencia	cAI_Service	
Dependencia	cBrain, cAI_Actor.	

	Nombre de la Clase: cBeacon		
	Descripción: Esta clase representa un evento en el Mundo, el Actor puede tomar la información de este evento.		
	Tipo	**Nombre**	**Descripción**
Atributos	Word	TypeFlags	Tipo de Evento (Disparo, Sonido de pasos, Cadáver, etc).
	float	Itensity	Intensidad del suceso.
	float	Radius	Radio de visibilidad del evento.
	Word	BasePriority	Prioridad del Evento a la hora de que el Actor le preste atención.
	cMatrix4f*	pLocation	Localización dentro del Mundo.
	cVector3f	Position	Posición del Evento.
	cVector3f*	pVelocity	Velocidad del Evento.
	float	ExpirationTime	Tiempo que dura el Evento.
Métodos	cVector3f	GetPosition	Método que devuelve la posición del Evento.
	cVector3f	GetVelocity	Método que devuelve la velocidad del Evento.
Herencia			
Dependencia	cMatrix4f, cVector3f.		

	Nombre de la Clase: cAIService_Update		
	Descripción: Clase donde se registran todas las Actualizaciones realizadas.		
	Tipo	**Nombre**	**Descripción**
Atributos	list<cUpdater*>	lpUpdaters	Lista de Actualizaciones.
Métodos	virtual void	Update	Método que actualiza todos lo elementos que componen al Mundo.
	void	InsertUpdater	Método que registra una nueva Actualización.
	void	RemoveUpdater	Método que elimina una Actualización.
	void	RemoveAllUpdaters	Método que elimina todas las Actualizaciones.
Herencia		cAI_Service	
Dependencia		cUpdater.	

	Nombre de la Clase: cThinkService		
	Descripción: Clase que representa el Servicio Pensar, esta es una de las cualidades que debe tener un Actor.		
	Tipo	**Nombre**	**Descripción**
Atributos	map<cAI_Actor*, cBrain*>	mpBrains	Lista de de Actores relacionados con sus Cerebros.
Métodos	virtual void	Update	Método que manda a pensar a todos los Actores del Mundo para que puedan tener cada cierto tiempo diferentes comportamientos según las circunstancias y lograr así un comportamiento razonable.

virtual void	InsertBrain	Método que inserta un nuevo Actor con su Cerebro.
virtual void	RemoveBrain	Método que elimina un Actor con su Cerebro.
Herencia	cAI_Service	
Dependencia	cBrain, cAI_Actor.	

	Nombre de la Clase: cSteeringBehavior		
	Descripción: Clase genérica cSteeringBehavior. Clase que define el comportamiento de los Actores y la trayectoria dentro del Mundo.		
	Tipo	**Nombre**	**Descripción**
Atributos			
Métodos	virtual cVector3f	Calculate	Método que calcula el movimiento del Actor.
Herencia			
Dependencia	cpMovil.		

	Nombre de la Clase: cSteeringBehavior_Evade		
	Descripción: Clase que define el comportamiento de los Actores a la hora de evitar obstáculos.		
	Tipo	**Nombre**	**Descripción**
Atributos	cVector3f	Target	Movimiento que define el comportamiento.
Métodos	virtual cVector3f	Calculate	Método que calcula el movimiento del Actor.
	static cVector3f	Evade	Método que define todo sobre el comportamiento (velocidad, dirección, etc).
Herencia	cSteeringBehavior		
Dependencia			

	Nombre de la Clase: cSteeringBehavior_EvadeObject		
	Descripción: Clase que define el comportamiento de los Actores a la hora de evitar obstáculos.		
	Tipo	**Nombre**	**Descripción**
Atributos	cpMovil*	pTarget	Movimiento que define el comportamiento.
Métodos	virtual cVector3f	Calculate	Método que calcula el movimiento del Actor.
Herencia		cSteeringBehavior_Evade	
Dependencia			

	Nombre de la Clase: cSteeringBehavior_Offset		
	Descripción: Clase que define el comportamiento de los Actores a la hora de mantenerse a cierta distancia de un objetivo.		
	Tipo	**Nombre**	**Descripción**
Atributos	cpMovil*	pTarget	Movimiento que define el comportamiento.
	float	MoveSpeed	Velocidad de desplazamiento.
Métodos	cVector3f	Calculate	Método que define todo sobre el comportamiento (velocidad, dirección, etc.).
Herencia		cSteeringBehavior	
Dependencia			

	Nombre de la Clase: cSteeringBehavior_Wander		
	Descripción: Clase que define el comportamiento de los Actores a la hora tener un movimiento sin objetivo alguno.		
	Tipo	**Nombre**	**Descripción**
Atributos	float	WanderValue	Valor que define el desinterés del Actor.
	float	RandomSize	Duración del periodo por el cual el Actor va tomar este comportamiento.
Métodos	virtual cVector3f	Calculate	Método que define todo sobre el comportamiento (velocidad, dirección, etc).
	static cVector3f	Wander	Método que define todo sobre el comportamiento (velocidad, dirección, etc).
Herencia		cSteeringBehavior	
Dependencia			

	Nombre de la Clase: cSteeringBehavior_AngularOffset		
	Descripción: Clase que define el comportamiento de los Actores a la hora de mantenerse a cierta distancia de un objetivo.		
	Tipo	**Nombre**	**Descripción**
Atributos	cpMovil*	pTarget	Movimiento que define el comportamiento.
	cVector3f	Offset	Velocidad a la cual el Actor va a perseguir su objetivo.
	float	MoveSpeed	Velocidad de desplazamiento.
Métodos	cVector3f	Calculate	Método que define todo sobre el comportamiento (velocidad, dirección, etc).
Herencia		cSteeringBehavior	
Dependencia			

	Nombre de la Clase: cSteeringBehavior_InRangePursuit		
	Descripción: Clase que define el comportamiento de los Actores a la hora de perseguir un objetivo.		
	Tipo	**Nombre**	**Descripción**
Atributos	cpMovil*	pTarget	Movimiento que define el comportamiento.
	float	MinDist	Atributos que define la distancia mínima para perseguir.
	float	MaxDist	Atributos que define la distancia máxima para perseguir.
Métodos	virtual cVector3f	Calculate	Método que define todo sobre el comportamiento (velocidad, dirección, etc.).
	static cVector3f	PursuitInRange	Método que define todo sobre el comportamiento (velocidad, dirección, etc.).
Herencia		cSteeringBehavior	
Dependencia			

	Nombre de la Clase: cSteeringBehavior_Seek		
	Descripción: Clase que define el comportamiento de los Actores llegar hasta un Punto.		
	Tipo	**Nombre**	**Descripción**
Atributos	cVector3f	Target	Movimiento que define el comportamiento.
Métodos	virtual cVector3f	Calculate	Método que define todo sobre el comportamiento (velocidad, dirección, etc).
	virtual float	PredictTimeToReachTarget	Método para calcular el tiempo en alcanzar el objetivo.

static cVector3f	Seek	Método que define todo sobre el comportamiento (velocidad, dirección, etc.).
Herencia	cSteeringBehavior	
Dependencia		

	Nombre de la Clase: cSteeringBehavior_SeekObject		
	Descripción: Clase que define el comportamiento de los Actores llegar hasta un Punto.		
	Tipo	**Nombre**	**Descripción**
Atributos	cpMovil*	pTarget	Movimiento que define el comportamiento.
Métodos	Virtual cVector3f	Calculate	Método que define todo sobre el comportamiento (velocidad, dirección, etc.).
Herencia		cSteeringBehavior_Seek	
Dependencia			

	Nombre de la Clase: cAI_World		
	Descripción: Clase rectora dentro del modulo. Implementa las funciones que permiten más adelante poder darle funcionalidad a este modulo.		
	Tipo	**Nombre**	**Descripción**
Atributos	multimap<Word, cAI_Service*>	mpServices	Estructura que guarda los Servicios del Mundo.
Métodos	virtual void	Update	Método que actualiza todos los elementos que se encuentran en el Mundo.
	virtual void	RegisterActor	Método para Registrar un Actor en el Mundo.
	virtual void	UnregisterActor	Método para Eliminar un Actor del Mundo.

virtual void	RegisterService	Método para Registrar un Servicio en el Mundo.
virtual void	UnregisterService	Método para Eliminar un Servicio del Mundo.
virtual cAI_Service*	GetService	Método para buscar un Servicio dentro del Mundo.
Herencia	cUpdater	
Dependencia	cAI_Actor, cAI_Service.	

	Nombre de la Clase: cSimulationLayer_AIWorld		
	Descripción: Clase que simula el funcionamiento del Mundo.		
	Tipo	**Nombre**	**Descripción**
Atributos	cAI_World	World	Objeto de Tipo Mundo.
Métodos	virtual void	OnUpdate	Método que en un determinado tiempo actualiza la simulación.
	virtual void	OnEnable	Método que comienza la simulación.
	virtual void	OnDisable	Método que detiene la simulación.
Herencia		cSimulationLayer	
Dependencia		cAI_World.	