UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
CENTRO DE INFORMÁTICA INDUSTRIAL

HERRAMIENTA ABIERTA DE VISUALIZACIÓN INTERACTIVA PARA LA PRÁCTICA DE INTELIGENCIA ARTIFICIAL INTEGRADA.

Tesis presentada para optar por el Título de Ingeniería en Ciencias Informáticas

Autor: **Luz María Rojas Venereo**

Tutor: **MsC. Yuniesky Coca Bergolla**

Co-tutor: **Ing. Dailyn Garcia Dominguez**

La Habana
Junio de 2011

Agradecimientos

Quisiera agradecer especialmente a mis tutores por el apoyo que me han brindado durante todo el desarrollo de este trabajo.
A la profesora Ing. Yenifer del Valle por su ayuda.
Quisiera agradecer a mi familia por apoyarme en todo momento y servirme de guía a Nancy, tía Carmen, Eva, Erasmo, Eliza, Reyna, Aliuskita, Alexis, Alexey, Nena, Irina.

A mi Madre

Gracias madre por ser quien eres, por tu dedicación y entrega, por demostrarme que todo en la vida es posible que no importa lo que tengamos sino lo que seamos capaces de hacer con lo que tenemos. Mil gracias mami.

A mi Hermanifila

Gracias a mi hermana por servirme de ejemplo, Por ser la mejor hermana del mundo, por preocuparse por mi mucho más de lo que yo sería capaz de preocuparme por mi misma.

A mi novio

Por estar siempre a mi lado en las buenas y malas, por su ayuda, por su amor.

A mis amigas y amigos

A Ro por ayudarme a ver las cosas lindas de la vida y no preocuparme mucho por las cosas que no merece importancia. A Sureikis por ser mi amiga incondicional, por no olvidarse de mi nunca y por tener fé en mi. A Angel mi amigo del IPI por nunca olvidarse de mi. A Anay por su ayuda. A todos los que he conocido en esta escuela y que como yo les digo hojitas de mi árbol gracias por haber formado parte de mi vida a Sahyli, Indy, Adisleydis, Rodolfo, Grabriel, Drigss, Alvaro, Liodan, Yuniel, Juan Antonio, Juan Miguel, Jeydi, Rosana, Yugleinis, Yunet, Thais.

Dedicatoria.

A la memoria de mi padre y mi abuela que siempre están en mi corazón y se que ellos están orgullosos de mi donde quiera que estén.

A mi madre y mi hermana por estar siempre a mi lado.

A mi familia en general por ser las personas con el corazón más lindo que he conocido.

A mi novio por ser el compañero que siempre va a estar presente en mi vida.

DECLARACIÓN JURADA DE AUTORÍA

Declaro que soy la única autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____

Firma de la Autora

Luz María Rojas Venereo

_____

Firma del Tutor

Yuniesky Coca Bergolla

_____

Firma de la Co-tutora

Daylin García Dominguez

Datos de contacto.

Msc. Yuniesky Coca Bergolla (ycoca@uci.cu)

- Graduado de Ciencias de la Computación en la Universidad Central de Las Villas en el año 2003.
- Profesor Asistente en la Universidad de las Ciencias Informáticas.
- Jefe de Dpto de Práctica Profesional del polo de Realidad Virtual de la Facultad 5.
- Obtuvo el Sello Forjadores del Futuro en el 2006 y 2008.
- Defendió la maestría en el año 2007.
- Árbitro de la serie científica de la Universidad.
- Está insertado en un doctorado en SoftComputing en colaboración con la UCLV y universidades españolas.

Ing. Dailyn Garcia Domínguez (dgdominguez@uci.cu)

- Graduada de Ingeniera en Ciencias Informáticas en el año 2009.
- Obtuvo el Sello Forjadores del Futuro en el 2009.
- Desarrolla su tema de maestría en Informática Aplicada.
- Se desempeña en el Departamento de Integración y Despliegue del centro de Informática Industrial (CEDIN).

Resumen.

Las herramientas de visualización interactiva son en la actualidad una de las más utilizadas para la representación visual de datos cuando estos son muy grandes o necesitan un nivel de abstracción muy elevado. Tienen aplicación en varios campos, el más importante para esta investigación es la educación, facilitando el aprendizaje y logrando, mediante la visualización, un mejor entendimiento del tema. Tomando en cuenta las ventajas de este tipo de herramientas y la necesidad de una para la práctica de contenidos de Inteligencia Artificial de forma integrada se propone la utilización de *OpenSteerDemo* para la práctica de las técnicas estudiadas en dicha asignatura como son las redes neuronales, lógica difusa, algoritmos genéticos, máquina de estado finita y sistemas basados en reglas; logrando de esta manera que el estudiante se sienta motivado a la hora de practicar los contenidos recibidos en clases. Se espera con este trabajo motivar al profesorado en la utilización de esta herramienta para lograr la formación de un estudiante mejor preparado, sobre todo con habilidades prácticas.

Palabras clave: Inteligencia Artificial, profesorado, visualización interactiva

Índice general

Índice de figuras

Índice de tablas

Introducción

Las capacidades intelectuales y la inteligencia humana son un recurso inagotable de cada pueblo, constituye también una obligación de nuestra sociedad potenciar al máximo su desarrollo colectivo e individual, muy necesario en estos tiempos que nos han tocado vivir. Se trata de ignorar la injusticia, la desigualdad entre las naciones, la incultura, la inseguridad y la pobreza. Ante estas condiciones, se impone potenciar al máximo la inteligencia, descubrir los nuevos caminos para la captación del conocimiento, la aceleración del aprendizaje y el incremento de su calidad, constituye también un arma de lucha por el progreso científico, económico y social.

Cada día el hombre moderno necesita resolver problemas de carácter complejo y tomar decisiones ante un nivel de indeterminación significativo. Desde la época de Aristóteles (384-322 a. C.) se viene analizando el funcionamiento de la mente humana para obtener conclusiones racionales, el mismo Aristóteles fue quien propuso un conjunto de reglas sobre este tema. Desde entonces y hasta la actualidad han sido muchos los científicos que han hecho aportes significativos sobre el tema de la Inteligencia. Dentro de esos aportes se encuentra el desarrollo de la Inteligencia Artificial (IA) que ha tenido cabida gracias a la significativa revolución científica y técnica que ha conocido la humanidad desde inicios del siglo XX y hasta nuestros días, para contribuir con el desarrollo de esta ciencia se requiere la formación de un hombre nuevo pertrechado del más alto nivel de conocimientos posibles y dotado de una alta sensibilidad científica, técnica y humana. Debido a que la IA tuvo muchos padres no hay un consenso para definir ese concepto, pero se puede decir que se encarga de modelar la inteligencia humana en sistemas computacionales.[Tec, 2010]. Son diversas las aplicaciones que tiene la IA entre las que se pueden mencionar: Minería de datos, Industriales, Médicas, Mundos virtuales, Procesamiento de lenguaje natural (*Natural Language Processing*), Robótica, Sistemas de apoyo a la decisión, y muchos más, entre estos uno de los más interesantes resultan Los Mundos Virtuales que tienen un auge sorprendente y que cada día ganan más aficionados en el mundo. Este término fue utilizado por los creadores del juego *Ultima Online*. De hecho los Mundos Virtuales nacieron y se desarrollaron inicialmente como entornos de juego, existen varios ejemplos de mundos virtuales, entre ellos se encuentran: *SecondLife*, *Habbo*, *Sanalika*, *Smeet* entre muchos otros donde su principal función es el entretenimiento. Se han desarrollado varias Herramientas que permiten interactuar con un Mundo Virtual una de ella es *OpenSim*

[Ope, 2011]. Las Herramientas de Visualización Interactiva en estos tiempos juegan un papel importante en la educación, debido a que el estudiante se siente motivado a la hora de hacer las tareas que se le orientan porque además de aprender se entretienen.

En Cuba específicamente en la Universidad de las Ciencias Informáticas(UCI) que se encuentra inmersa en un nuevo modelo de integración de la formación, la producción y la investigación. En la facultad 5 se encuentra el Centro de Informática Industrial(CEDIN) y dentro de éste la línea de Visualización y Realidad Virtual donde es fundamental el trabajo con la IA, es de suma importancia la utilización de nuevas herramientas para la práctica [Bergolla, 2010], pero es pobre su utilización debido a que no existe una Herramienta de Visualización Interactiva para la práctica de IA donde se puedan integrar varias técnicas, para que los estudiantes puedan practicar los contenidos recibidos en clases.

Existen actualmente algunas herramientas utilizadas en el mundo aunque las más conocidas son destinadas al ocio como es el caso de los denominados mundos virtuales. También se han desarrollado otras herramientas para la simulación científica tanto en la medicina como en la robótica, algunas desarrolladas también en universidades con el fin de fomentar la práctica de los contenidos impartidos en las clases de IA como es el caso de la herramienta IAIC [Mandow,]. En nuestra universidad para la práctica de sistemas basados en reglas en la asignatura IA se usa el lenguaje Prolog, y en el curso optativo de Desarrollo de Elementos Inteligentes se ha propuesto la biblioteca OpenSteer con la interfaz gráfica *OpenSteerDemo*, como una de las variantes a utilizar por los estudiantes en los proyectos finales de la asignatura pero no ha sido muy utilizada, ya que no se ha formalizado la forma de utilización de la misma para integrar otras técnicas de IA.

Se plantea entonces el siguiente **problema científico** ¿Cómo lograr la práctica de contenidos de Inteligencia Artificial de forma integrada?

El **objeto** sobre el que se enfoca este estudio, tanto desde el punto de vista teórico como práctico, con vistas a solucionar el problema planteado consiste en las herramientas abiertas de visualización interactiva.

Con el propósito de brindarle una solución efectiva al problema, se plantea como **objetivo general** proponer una herramienta abierta de visualización interactiva para la práctica de contenidos de IA de forma integrada.

El **campo de acción** de la investigación son las herramientas abiertas de visualización interactiva para la práctica de Inteligencia Artificial.

Para satisfacer el cumplimiento del objetivo se plantean como tareas de investigación:

- Elaboración del diseño teórico metodológico.
- Elaboración de informe sobre las Herramientas Abiertas de Visualización Interactiva que más se utilizan en el mundo.
- Elaboración del Capítulo 1.

- Realización del Análisis y diseño del OpenSteer (Ingeniería inversa) (Artefactos fundamentales).
- Realización de informe acerca de la propuesta de incorporación de clases o módulos al OpenSteer para adaptarlo a la asignatura de IA (Informe y Artefactos).
- Elaboración del capítulo 2.
- Implementación de un Plugin con una de las técnicas de IA a incorporar.
- Elaboración del capítulo 3.

Para la realización de las tareas de investigación se utilizarán los siguientes métodos de la investigación:
Teóricos:

Histórico-lógico: Para conocer del objeto de estudio su devenir histórico, antecedentes, desarrollo y perfeccionamiento de la enseñanza de la IA en nuestro país y el mundo, así como tendencias actuales e investigaciones relacionadas.

Analítico-sintético: Para el análisis de las herramientas de visualización y las diversas tendencias educativas y su relación con la informática, concretamente con la IA.

El contenido de este trabajo se encuentra estructurado de la siguiente manera:

Capítulo 1 ”Fundamentación Teórica”, se hace un análisis donde se presentan las diferentes herramientas de visualización que se utilizan en el mundo para la práctica de IA. Se analizan sus características, funcionalidades y capacidad de adaptación para agregarle nuevos módulos que permitan la práctica integrada de IA, así como algunas técnicas de IA.

Capítulo 2 ”Solución Propuesta”, en este capítulo se realiza la selección de la herramienta y la ingeniería inversa a la herramienta seleccionada, se propone la incorporación de las técnicas estudiadas en la carrera de Ingeniería en Ciencias Informáticas estudiada en la UCI.

Capítulo 3 "Análisis de los Resultados”, Se describirá cómo se incorporó una de las técnicas de IA usando la herramienta seleccionada.

Capítulo 1

Fundamentación Teórica

Con el avance de la ciencia moderna, la creación de IA ha tomado dos caminos fundamentales: la investigación psicológica y fisiológica de la naturaleza del pensamiento humano, y el desarrollo tecnológico de sistemas informáticos cada vez más complejos.
En el segundo aspecto el término IA[Gonzalo Sanz, 1987] se ha aplicado a sistemas y programas informáticos capaces de realizar tareas complejas, simulando el funcionamiento del pensamiento humano, aunque todavía muy lejos de los extraordinarios logros de la madre naturaleza. Los campos de investigación más importantes en esta área son, el procesamiento de la información, el reconocimiento de modelos, los juegos y las áreas aplicadas, como el diagnóstico médico, la educación, entre otras. La IA es aún muy joven, a pesar de ello se hace imprescindible su uso en todas las ramas de la Computación, para proporcionar sistemas cada vez más capaces de resolver problemas complejos, dando la respuesta de una forma más rápida y acertada. El nivel alcanzado hasta el momento en cuanto a técnicas es muy alto y los conocimientos se adquieren de las más difíciles abstracciones que pueden causar dificultades en su entendimiento. Existen algunas oportunidades que se pueden aprovechar en este sentido como la visualización.
La visualización es una técnica muy usada para representar gráficamente un conjunto de datos. Cuando estos son muy complejos o grandes, la visualización puede ayudar a que los datos sean más fáciles de leer o entender. Hay herramientas de visualización para la búsqueda, la música, redes, comunidades online, y casi cualquier cosa que se pueda imaginar. Existe gran variedad de aplicaciones de escritorio y de herramientas basadas en web, hay muchas herramientas específicas que están disponibles en la web y que le permiten visualizar todo tipo de datos.
En el tema de la IA existen varias herramientas de visualización aunque son específicas para cada técnica. Entre las que se pueden encontrar herramientas para la representación del funcionamiento de los algoritmos de búsqueda como es el caso de IAIC, Simbad para el área de la robótica, *OpenSteerDemo* para los *Steering behaviours*, entre otros. Nuestro país no

se ha quedado atrás en la utilización de estas herramientas tal es el caso de la Universidad Central de Las Villas ”Marta Abreu”(UCLV)[Perez, 2002], que en nuestro país es abanderada en el estudio de las técnicas de IA, en la investigación se han mezclado varios departamentos como son: el departamento de Ciencia de la Computación, Física y de la facultad de Ingeniería Eléctrica que desarrolló una metodología para la enseñanza basándose en los sistemas expertos.
En el presente capítulo se describirán las herramientas de visualización interactiva más importantes, así como conceptos fundamentales de este tema. Se Analizará el uso de las mismas en la educación, la influencia que tienen en los estudiantes y cómo mejora esto el proceso de enseñanza/aprendizaje. Además, se estudiará la importancia de las herramientas y cómo mejora la visualización en el entendimiento de las diferentes técnicas de IA.

1.1. Conceptos Fundamentales

1.1.1. Inteligencia Artificial

La IA toma un sentido científico viable como disciplina informática moderna durante la segunda mitad del siglo XX, esto es el resultado directo de la confluencia de diversas corrientes intelectuales como son la teoría de la computación, cibernética, teoría de la información y el procesamiento simbólico, desarrolladas sobre los cimientos formales de la lógica y la matemática discreta e impulsadas por el desarrollo de los ordenadores digitales. El campo de la IA surge en la Conferencia de Dartmouth (1956) y en los años dorados (1956-1963) por los investigadores(Minsky, McCarthy, Newell, Simón, Samuel, Rochester, Shannon, Solomonoff, Selfridge, More) en esta conferencia se definen las directrices y líneas de actuación futuras. La IA [Gonzalo Sanz, 1987] como lo definen algunos es la inteligencia exhibida por artefactos creados por humanos y se encarga de modelar la inteligencia humana en sistemas computacionales. Es por ello que las técnicas de IA hasta nuestros días vienen muy ligada a la evolución del hombre como se explicará más adelante con algunas de ellas.

1.1.2. Mundo virtual

Es un tipo de comunidad virtual que simula un mundo o entorno artificial inspirado o no en la realidad, en el cual los usuarios pueden o no interactuar entre sí a través de personajes o avatares, y usar objetos o bienes virtuales. Es un entorno interactivo de simulación al que pueden acceder múltiples usuarios a través de una interfaz. A los mundos virtuales también se les llama ”Mundo digital”. En el mundo se ha fomentado mucho la utilización de entornos virtuales con varias aplicaciones, para investigaciones científicas en el estudio de varios campos como son la medicina, la robótica. Algunos con fines profesionales de aprendizaje como son

los simuladores de vuelo, de enseñanza (MMOLE)[sit, 2006], pero en la actualidad está siendo llevado por las empresas de ocio electrónico, que ven en esta tecnología una nueva era para los videojuegos. También son usados en la educación donde las tecnologías han generando nuevas percepciones y oportunidades en los múltiples ámbitos de las relaciones sociales en la dinámica de la vida diaria de hoy, planteando por ello, un reto constante de redefinición a las iniciativas educativas en todos los niveles a escala mundial [Guerrero,].

1.1.3. Boids

Los boids son resultado del estudio de la naturaleza, a fines de los años 1980 el científico Christopher Langton fue el primero en utilizar el término de vida artificial cuando se celebró la *"International Conference on the Synthesis and Simulation off Living Systems"*("Primera Conferencia Internacional de la Síntesis y Simulación de Sistemas Vivientes") también conocida como *Artificial Life I* en los Alamos *National Laboratory* en 1987. En inglés también se conoce como *alife*, por la contracción de *artificial life*. La vida artificial es el estudio de la vida y de los sistemas artificiales que exhiben propiedades similares a los seres vivos, a través de modelos de simulación. Modelo computacional para simular el movimiento de bandadas de pájaros, bancos de peces o manadas de mamíferos. Craig Reynolds desarrolló un proyecto en el año 1986 donde simula el comportamiento de las bandadas de pájaros compuesto de 3 comportamientos simples: cohesión para que los agentes se mantengan en grupo, separación para evitar el hacinamiento y alineación para que mantengan la misma trayectoria. Cada agente tiene información del entorno y es capaz de evadir los obstáculos que se encuentren en el camino.

El modelo *boids* es un ejemplo de un modelo basado en individuales, una clase de simulación utilizado para capturar el comportamiento global de un gran número de agentes autónomos que interactúan. Estos modelos están siendo utilizados en la biología, ecología, economía y otros campos de estudio [Reynolds, b]. Estos modelos han sido ampliamente utilizados por los realizadores de Animados, donde los actores son animales tal es el caso del filme de animación El Rey León entre otros.

Se han desarrollado varios *software* aplicando los modelos void entre los que se encuentran: ***Boids*** 1986-1988, escrito en Common Lisp Simbólica, y se basa en sistemas 3D de Geometría Simbólica, sistema de modelado y animación dinámica del sistema Moderno [Reynolds, a]. ***C++ Boids*** de Christopher Kline con código fuente escrito en C++. ***Buzzz!*** es un protector de pantalla para las computadoras Macintosh de Simon Fraser implementa una versión con parámetros de *boids* incluyendo varias especies de animales (avispas, pájaros, peces, ovejas, entre otros).

1.2. Visualización Interactiva

Los avances de la visualización, conducen al avance de aquellas disciplinas que en número creciente encuentran espacios de aplicación. Estos avances ayudan a las personas a explorar o explicar los datos por medio de sistemas de *software* que proporcionan representaciones visuales estáticas o interactivas y extiende su utilidad al integrarse además con técnicas y herramientas analíticas de otras disciplinas, como la estadística, la minería de datos (data mining), el KDD (Knowledge Discovery in Databases) y el procesamiento de imágenes, para facilitar el análisis tanto desde una perspectiva cuantitativa como de una cualitativa[Johnson CR, 2006]. Consiste en mostrar de manera amplia y agradable conceptos, abstracciones y procesos en aplicaciones, mundos virtuales e incluso mediante el uso de herramientas simples que se encuentren a la disposición del que desea visualizar el contenido.

1.2.1. Control humano de la visualización

Las herramientas de visualización para ser interactiva debe cumplir con la entrada de datos, es decir que pueda ser controlado por el usuario. A continuación se listan los parámetros del control humano.

- Selección de una cierta parte de una representación visual existente.
- Localización de un punto de interés (puede no tener una representación existente).
- Movimiento en una trayectoria.
- Elección de una opción de una lista de opciones.
- Entrada de números.
- Escribir texto.

Todas estas acciones requieren un dispositivo físico. Existe una gran variedad de dispositivos de entrada, los más comunes son: teclados, ratones, *Trackball*, y *touchpads* entre otros.

1.2.2. Herramientas de Visualización Interactiva

Las herramientas de visualización interactiva proporcionan una mejor comprensión de un problema determinado. No existen herramientas para todos los propósitos sino que estas tienden a ser específicas para la clase de objetos que se quieren generar. Existen muchas herramientas de visualización interactiva, esto se debe a las nuevas tecnologías emergentes. Para dar una mejor perspectiva de lo que se quiere visualizar se emplean mucho las herramientas que simulan un Mundo Virtual en tiempo real, en las cuales el usuario puede controlar,

modificar, crear y eliminar un objeto determinado. Entre las herramientas más difundidas en el mundo se encuentran *Second Life*, *OpenSim* y *OpenSteer*.

"La función de la herramienta no es otra que la de servir de conductor de la influencia humana en el objeto de la actividad; se halla externamente orientada y debe acarrear cambios en los objetos. Es un medio a través del cual la actividad humana externa aspira a dominar y triunfar sobre la naturaleza. Por otro lado, el signo no cambia absolutamente en nada en el objeto de una operación psicológica. Así pues, se trata de un medio de actividad interna que aspira a dominarse a sí mismo; el signo, por consiguiente, está internamente orientado".[Vygotski, 2000]

Ejemplo de algunos programas:

***Bunkspeed*:** Es una aplicación diseñada para el procesamiento de imágenes. Permite importar datos 3D y aplicarle materiales e iluminación.
***DataTank*:** Está diseñado para la visualización científica, la minería de datos y desarrollo de algoritmos. Es lo suficientemente flexible como para utilizarlo con otros usos. Se puede utilizar como un depurador gráfico y herramienta para el desarrollo de algoritmos.
***Drishti*:** Se encarga de explorar conjuntos de datos volumétricos de tomografía computarizada. Se ha utilizado además para el análisis del flujo en enormes sistemas 3D.
***InfoScope*:** Es una herramienta de visualización interactiva para acceder, analizar y comunicar conjuntos de datos muy grandes y complejos.

1.2.3. Áreas de Aplicación de las Herramientas de Visualización Interactiva

A nivel mundial los sistemas convencionales han sido ampliamente utilizados y construidos para una gran diversidad de materias. Sin embargo, muchos sistemas inteligentes están limitados a ideas del saber relativamente bien estructuradas, como es el caso, entre otros, de: Matemáticas (WEST, BUGGY, LMS), en la programación de computadoras están las herrameintas (BIP, PROUST, SPADE), en el diagnóstico médico (GUIDON), en el campo de la electrónica (SOPHIE), entre otros .
En la IA es importante destacar que son muy escasas las herramientas que existen actualmente para la práctica en sentido general de varias técnicas o la mayoría de las técnicas de la IA.
A los sistemas inteligentes se le asocia el logro de métodos más generales y flexibles de representación y manipulación del conocimiento, por lo cual constituyen recursos poderosos para lograr un uso más adecuado de las computadoras como medios auxiliares del proceso de instrucción.

1.3. Las Herramientas de Visualización Interactiva en la Educación

Son diversas las aplicaciones de las herramientas de visualización interactiva en la educación. Se han desarrollado para facilitar de alguna manera el aprendizaje de futuros profesionales, tal es el caso de los sistemas tutores inteligentes para la enseñanza de la Física que se desarrolló en la Universidad Central ”Marta Abreu”de Las Villas [Perez, 2002] por el departamento de Física y orientado a apoyar el proceso de enseñanza-aprendizaje de las leyes de la dinámica de la partícula y del fenómeno de la interferencia luminosa. INTERFER, entrenador que le permite a un estudiante profundizar en la rama de la óptica, en específico la interferencia luminosa.

1.3.1. IAIC

En la Universidad de Málaga en el año 2008 se desarrollaron dos Herramientas de Visualización para una mejora en la enseñanza-aprendizaje de los algoritmos de búsqueda empleados comúnmente para la resolución de problemas en IA. Esta herramienta tomó el nombre de IAIC, la cual logró una buena aceptación por parte de los estudiantes, según estudios realizados concluido el desarrollo de la misma. Cuenta con varias funcionalidades que se describen a continuación.

Funcionalidades:

1. Editor gráfico de laberintos que permite crear, guardar, y cargar distintos mapas.
2. Algoritmos disponibles (Nilsson, 2001) (Korf, 2005):
   - Búsqueda primero en amplitud.
   - Búsqueda primero en profundidad.
   - Búsqueda frontera primero en amplitud.
   - Búsqueda A*.
   - Búsqueda Backtrack sin ciclos.
   - Búsqueda Backtrack con profundización progresiva (BID).
   - Búsqueda A* con profundización progresiva (IDA*).
   - Búsqueda frontera.

3. Heurísticos para los algoritmos A* e IDA*:
    - Búsqueda a ciegas.
    - Distancia Manhattan.
    - Distancia euclídea.
4. Límite de profundidad para el algoritmo Backtrack sin ciclos.
5. Control para pausar, continuar y reiniciar la simulación, así como para modificar la velocidad de la misma.
6. Informe sobre el consumo de memoria actual y máxima, y número de iteraciones del algoritmo en ejecución.

Este tipo de herramienta es muy necesaria hoy en día para el sector de la educación universitaria en nuestro país y es muy bueno el sentido y el objetivo para el que los programadores lo han desarrollado, sin embargo trabaja con una sola técnica: la Búsqueda de caminos.

1.3.2. Simbad

Es un simulador de multi-autómatas en Java y como característica principal cuenta con un ambiente en 3D, fue desarrollado para propósitos científicos y educativos. Es primordialmente dedicado para los investigadores y los estudiantes que quieren una base simple para estudiar IA, *Machine Learning*, y más generalmente los algoritmos de IA, en el contexto de Robótica Autónoma y los Agentes Autónomos. Simbad permite a los programadores escribirle a sus controladores automatizados, modificar el ambiente y usar los sensores disponibles (o la constitución de ellos). No está dirigido a proveer una simulación del mundo real y es mantenido voluntariamente. El hecho de que el código de la fuente no está disponible para usuarios estándar impide tener un entendimiento profundo y preciso del comportamiento del simulador.

El simulador provee las siguientes funcionalidades:

- Simulación sola o multi-automatizada.
- Color Cámaras Mono-Scopio.
- Sensores de contacto y de rango.
- Simulación en línea.
- *Python scripting* con *Jython*.
- Interfaz extensible de usuario.
- Motor físico simplificado

Cuenta con tres paquetes

1. El simulador Simbad: Un simulador de robot móvil extensible que puede manejar escenas complejas en 3D y la interacción basada en la física.
2. *PicoNode*, biblioteca de red neuronal.
3. *PicoEvo*, biblioteca de algoritmos Evolutivos.

Por lo que respecta a la implementación, el usuario sólo provee un ambiente derivado de la clase *EnvironmentDescription* y un control del robot derivada de la clase Robot. Este último tiene un método de inicialización (*initBehavior*) y un método que se pidió a cada paso de simulación (*performBehavior*). El simulador ejecuta las órdenes de control del motor de una manera similar a un controlador del robot real, es decir, las llamadas repetitivas al micro-controlador.
La biblioteca *PicoNode* provee un *Framework* basado en gráficas general de representación junto con dos implementaciones: Las redes *feed-forward* y neuronales recurrentes. Las redes neuronales proporcionan un marco de representación muy fuerte en el ámbito de control del robot debido a su capacidad para manejar los datos continuos. Por otra parte, los estados internos pueden estar representados en las arquitecturas recurrentes.
Una de las principales limitantes para nuestra facultad es su desarrollo en lenguaje java, además su entorno es único lo que limita la creatividad de los estudiantes y de los profesores al utilizar dicha herramienta.

1.3.3. Robocode

Es un entorno gratuito de simulación de guerras de robots, desarrollado por *Alphaworks* de IBM, en el que hay que programar un tanque en Java para pelear en el campo de batalla contra tanques programados por otros jugadores.
Didácticamente se emplea en un nivel básico para enseñar a programar en Java (cómo escribir código Java, el manejo de una API, manejo de eventos, clases internas, herencia, etc.) y en un nivel más avanzado, para la simulación de sistemas de *software* autónomos dotados de inteligencia con una estrategia y unas tácticas diseñadas para ganar la batalla. El juego transcurre en un campo de batalla en 2D y su duración se mide en turnos.
El usuario puede configurar el tamaño del campo de batalla o el número de rondas, así como el tiempo real empleado para cada turno, y por tanto para cada partida. Para llevar a cabo una batalla es necesario especificar, como mínimo, el número y tipo de robots que van a tomar parte en ella.
Robocode proporciona, además, la posibilidad de hacer un seguimiento de los robots e incluso de llevar a cabo la depuración de los mismos. Los robots avanzados son aquellos que extienden la clase *Robocode AdvancedRobot*. Entre sus características destaca el hecho de que

permiten la ejecución de eventos no bloqueantes. Por ejemplo, un robot avanzado que ejecute las dos órdenes indicadas anteriormente, es decir, girar *90* ° y avanzar, lo haría describiendo una curva del mismo modo que lo haría un automóvil. Dicha curva se describiría hasta el momento en que la orientación del robot hubiera cambiado *90* ° respecto con la orientación inicial. Además los robots avanzados presentan otras ventajas, como es el hecho de que permiten la edición de eventos y la escritura de datos en archivos.
El simulador es gratuito y está programado en Java, con lo que funciona bajo cualquier plataforma que tenga instalado JRE (en particular, Windows y Linux). Simplemente hay que instalar la plataforma *Robocode*. Presenta los mismos inconvenientes que Simbad.

1.3.4. SecondLife(SL)

Fue lanzado el 23 de junio de 2003 y desarrollado por Linden Lab, cuenta con un motor físico: *Havok*, el audio es controlado por *FMOD* es accesible gratuitamente desde Internet. Se puede acceder también mediante programas llamados Viewers (visores). La última versión disponible hasta el momento fue lanzada el 12 de Febrero de 2011 y es la 2.4.0.216989. Corre en las plataformas *Microsoft Windows*, *Windows* XP SP2 y SP3, Windows Vista, *Windows* 7, Mac OS X (10.4.11 o higher), Linux i686. En marzo de 2008 SL contaba con aproximadamente unos 13 millones de cuentas, en enero de 2010 ya tenía registrados más de 18 millones de cuentas, superando los 20 millones en agosto del mismo año. Pese a ello, no existen estadísticas fiables en relación al uso consistente de estas cuentas a largo plazo.
Esta herramienta permite al usuario o residente que es como se les llama, interactuar con otros usuarios en un entorno virtual o mundo virtual mediante un avatar. Proporcionando varias funcionalidades como pueden ser: establecer relaciones sociales, participar en diversas actividades tanto individuales como en grupo, crear y comercializar propiedad virtual y servicios entre ellos. Con la posibilidad de contar con una moneda local el *Linden Dólar* (L), así el usuario puede vivir una segunda vida pero en un mundo virtual. Muchas veces estos sistemas donde pueden acceder cualquier cantidad de usuarios algunos con conocimiento de programación y otros solo con el básico de navegación, fomentan la forma de ganar dinero fácil para los más aventajados, en vez de usarlo como un sistema colaborativo lo usan en beneficio propio que no es el objetivo que se quiere para la sociedad cubana.
Cuenta con una variedad de aplicaciones en diversos campos entre las que se pueden encontrar la educación por muchas instituciones, como universidades, bibliotecas y entidades de gobierno. La investigación científica, colaboración, y visualización de datos. Brinda a las compañías la opción para crear espacios de trabajos virtuales para potenciar el trabajo en equipo, además de facilitar cualquier clase de comunicaciones corporativas, las sesiones de entrenamiento de conducta en espacios educativos 3D inmersivos, simular procesos comerciales y prototipo de productos nuevos. Por lo que se ha convertido en una herramienta muy usada por los internautas.

La forma de crear objetos con comportamiento como son carros, armas, aparatos de música entre otros es mediante modelado en 3D y la programación de los artículos creados mediante Linden Scripting Language o (LSL). Esto es otra forma de ganar dinero en este mundo virtual pues todo lo que un residente crea es de su propiedad y puede hacer lo que quiera con ello. En el trabajo de crear estos objetos es donde entra la parte de la práctica de la IA ya que al programar el comportamiento de estos objetos es necesario tener conocimiento de la materia. Este *software* a pesar de ser muy popular para los usuarios tiene como desventaja que necesita conexión a internet. Debido a estos en los países subdesarrollados no todas las personas tienen acceso a la tecnología necesaria para poder acceder a SL.
Para acceder a Second Life los desarrolladores sugieren que la máquina cumpla con ciertas características: se necesita tener acceso a internet, con 256 MB RAM tanto en Windows como Linux, 512 MB RAM (Mac), espacio en disco duro para caché de disco de 1000 MB, procesador x86 a 800 MHz o superior tanto en *Windows* como *Linux*. En Mac se recomienda 1 GHz G4 o superior/Intel Core Processor, *NVidia GeForce* 2, GeForce4 MX o superior, ATI Radeon 8500, *Radeon* 9250 o superior, **De gráfico se requiere** NVidia GeForce 6800 u otros. ATI Radeon X1700, *Radeon* X1800 u otros. Y los controles pueden ser mediante el teclado, mouse, gamepad, entre otros. Por las limitantes de conexión este software no está disponible en nuestro país o el acceso es demasiado lento.

1.3.5. OpenSim

OpenSim es una plataforma capaz de manipular de forma interactiva un Mundo Virtual, soporta varias regiones conectadas a un solo *Grid* centralizado, sus creadores se basaron en SL. Además, utiliza el mismo estándar de comunicación. Es de código abierto (licencia BSD). Al ser una aplicación de código abierto facilita la flexibilidad para adaptarse a nuevos requerimientos. Se encuentra disponible para *Windows*, *Linux* y *Mac*.
OpenSim apoya la creación de mundos múltiples en una sola instancia de aplicación. Además, soporta múltiples clientes y protocolos de acceso al mismo mundo, al mismo tiempo a través de múltiples protocolos. Posee una amplia capacidad de personalizar su avatar, tanto con la ropa personalizada, pieles y objetos conectados. Facilita la creación de contenido en tiempo real en el medio en el que se desenvuelve el avatar, utilizando las herramientas de creación de un mundo virtual[Guerrero,]. Para la programación de script cuenta con una serie de lenguajes diferentes, incluyendo LSL / ossl, C, *JScript* y VB.NET. Cuenta con varios motores para la física actualmente, son *basicphysics, OpenDynamicsEngine*, *RealPhysX* y *BulletX* (versión modificada). También puede conectarse fácilmente a cualquiera de las muchas redes públicas en Internet.
Base de datos OpenSim es compatible con los siguientes motores de bases de datos:

1. *SQLite*: Por defecto - una base de datos ligera que viene incluida con OpenSim y se puede utilizar sin necesidad de configuración adicional.

2. *MySQL* 5.1: Esta es la base de datos recomendada para un uso más allá de la experimentación o pequeñas aplicaciones independientes.

Arquitectura General

La arquitectura es basada en modo Grid, en este modo varios aspectos de la simulación son separados entre múltiples procesos que puedan existir en diferentes máquinas, tiene el potencial para escalar el crecimiento del número de usuarios.

El término Grid se refiere a una infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que son propiedad y están administrados por diferentes instituciones. Puesto que la colaboración entre instituciones envuelve un intercambio de datos o de tiempo de computación, el propósito del Grid es facilitar la integración de recursos computacionales.

Las responsabilidades son divididas entre 5 servidores:

1. *User server* -servidor de control de usuarios.

2. *Grid server* -servidor de control de Grid.

3. *Asset server*.

4. *Invetory server* -servidor de control de inventario.

5. *OpenSim server* -simulador de regiones.

El objetivo del User server es: controlar los usuarios, el mismo consta de 3 responsabilidades primarias.

1. Identificar al usuario en el simulador.

2. Recuperar información del usuario.

3. Eliminar la sección de los usuarios cuando estos no entran durante un tiempo.

Ya en la facultad 5 se tiene instalado y se comienza a crear una comunidad que requiere de tiempo para familiarizarse con dicho entorno, tanto profesores como estudiantes deben comenzar a incorporar IA al entorno, algo que no es trivial, más bien se hace complicado en una herramienta no definida para esos fines.

1.3.6. OpenSteer

Es una biblioteca de código abierto escrita en C++, para ayudar a construir comportamientos de locomoción de personajes autónomos en los juegos y otros tipos de simulaciones multi-agente. Para ello se hace necesario explicar qué es un agente y un sistema multi-agente.
Agente: No es más que un objeto o entidad, por decirlo de alguna manera, que se comporta de manera autónoma. Un agente debe cumplir con tres características fundamentales:
Debe comportarse de forma Reactiva, esto quiere decir que debe ser capaz de responder a los cambios que ocurren en el entorno en que se encuentra.
Proactivo, debe ser capaz de cumplir con sus propios objetivos.
Social, debe poder comunicarse con los demás agentes que interviene en el entorno mediante algún tipo de lenguaje de comunicación entre agentes.
Sistema Multi-agente: Es un sistema en el cual interactúan varios agentes, es importante aclarar que en un sistema multi-agente, los agentes podrían ayudarse unos a otros siempre y cuando esto no impida cumplir sus objetivos.
Estos agentes pueden representar personajes como seres humanos, animales, entre otros. Y vehículos como son los automóviles, aviones, naves espaciales u otros tipos de agentes móviles. *OpenSteer* proporciona un conjunto de comportamientos de locomoción, que se define en términos de un agente móvil abstracto.
OpenSteer proporciona una aplicación interactiva llamada *OpenSteerDemo* que cuenta con un conjunto de *plugins* donde están definidos varios comportamientos de locomoción. *OpenSteerDemo* brinda la posibilidad de agregarles nuevos comportamientos a los agentes por la arquitectura que posee que facilita esta función y utiliza gráficos OpenGL. Todas las llamadas OpenGL se encuentran en un módulo aparte que facilita sustituirlo por otra API de gráficos.
OpenSteer posee varios comportamientos definidos: comportamiento de vagar, búsqueda, comportamiento de huída, seguir un camino, evasión de obstáculo, evasión de colisión no alineada, separación, alineación, cohesión, mantenimiento de velocidad, etc.

1.4. Técnicas de Inteligencia Artificial

1.4.1. Sistemas basados en reglas

Los sistemas basados en reglas (SBR) son una de las técnicas de IA más utilizadas en la historia de las herramientas de visualización interactiva y en los videojuegos. En su representación más simple consisten en un conjunto de instrucciones `"If ... then ..."` que se utilizan para evaluar estados y tomar decisiones [Elaine Rich, 1994]. Los sistemas de reglas presentan dos ventajas principales respecto al resto de técnicas usadas en sistemas de IA. En primer lugar, se trata de un sistema que intenta representar la manera de pensar y razonar de los humanos ante una situación a partir de la información que obtienen del entorno en el que

se desenvuelven. En segundo lugar, es un sistema muy fácil de programar.
Como todo, los sistemas basados en reglas también tienen desventajas en su utilización y la principal es que resulta poco escalable. Cuando se necesitan implementar muchas reglas, el sistema se vuelve difícil de gestionar y el tiempo que se necesita para el estudio de las reglas puede provocar un efecto negativo en el rendimiento del proceso de IA. Además, los SBR tampoco son muy seguros en situaciones poco frecuentes que no están definidas dentro de las reglas.
Un sistema basado en reglas está formado por tres componentes principales:

1. La memoria de trabajo: Aquí se incluyen todos los hechos y todas las afirmaciones que tengan relación con la situación actual, tanto aquello que se puede leer directamente desde el entorno (como, por ejemplo, que se hallan cinco enemigos cerca) como aquello que inferimos nosotros mediante las reglas (como, por ejemplo, que existe un elevado riesgo de ataque).

2. El conjunto de reglas: Se trata de secuencias del tipo ”If... then ...”, con una serie de condiciones, y que pueden dar como resultado o bien inferir más información para la memoria de trabajo, o bien encontrar la decisión final sobre qué hemos de realizar en ese momento.

3. Una condición que determine si se ha hallado la decisión adecuada para la situación o si no existe ninguna dentro del sistema. Esto es necesario para terminar el bucle de decisión del sistema de reglas.

Un caso particular donde se usan los sistemas basados en reglas son los sistemas expertos, los mismos son diseñados para dar solución a los problemas planteados en una esfera específica entre las que se encuentran sistemas expertos para el diagnóstico médico, sistema de gestión de finanzas entre otros. En los videojuegos no es recomendable su uso por el alto coste computacional. Se pueden desarrollar sistemas expertos conexionistas a partir de la combinación de los sistemas basados en reglas (SBR) o de otras técnicas con las redes neuronales (RN).
La combinación de SBR con RNA ofrece potencialmente beneficios debido a que ambos modelos computacionales se complementan en cuanto a su capacidad de explicación, requerimientos y tiempo de desarrollo, pudiendo aprovechar así las ventajas y desventajas de cada uno para formar un sistema más eficiente:

Criterio	Sistemas Basados en Reglas	Redes Neuronales Artificiales
Capacidad de Explicación	Buenas	———
Requerimientos	Expertos que articulen su conocimiento	Muchos ejemplos
Tiempo de desarrollo	12 a 18 meses	Semanas o meses

Tabla 1.1: Comparación entre SBR y RN.

1.4.2. Máquina de estado finito

Una máquina de estado es una técnica que permite hacer un seguimiento de la vida de un objeto en el transcurso de un tiempo finito.
Las máquinas de estado finitas(MEF)[Vilca, 2008] fueron ideadas en el campo de las matemáticas y es muy usada en el campo de la IA. En sus inicios se utilizó para la representación del lenguaje y actualmente en la IA a alcanzado un nivel de utilidad elevado para la toma de decisiones. Básicamente se basa en un conjunto de reglas que cuando se cumplen se activa un estado. Es posible que se activen varias reglas, cuando esto ocurre se dice que es un grupo de conflicto. Puesto que solo debe haber una transición del estado actual a otro estado, no pueden activarse dos estados a la vez. Para la solución de este conflicto es necesario tomar estrategias consistentes para seleccionar una de las reglas activadas para dispararse y así realizar la transición de estado.
Debido a esto se toman dos tipos de MEF las deterministas que, dada una entrada y el estado actual, puede predecirse la transición de estado y las no deterministas lo opuesto a esto, dada una entrada no puede predecirse el estado que tomará. El uso de las máquinas de estado finitas deterministas en los videojuegos no es recomendable debido a que tiende a eliminar el factor de diversión del juego. Para lograr la implementación de una MEF no determinista sería mediante la aplicación de otra técnica experimentada en la IA, la Lógica Difusa, tomando así el nombre de Máquina de Estados Difusa.
Las MEF están compuestas por 4 elementos principales:

1. Estados que definen el comportamiento y pueden producir acciones.
2. Transiciones de estado que son movimientos de un estado a otro.
3. Reglas o condiciones que deben cumplirse para permitir un cambio de estado.

4. Eventos de entrada que son externos o generados internamente, que permiten el lanzamiento de las reglas y permiten las transiciones.

Las máquinas de estado finitas nos permiten modelar el comportamiento de un sistema, especificando una serie de estados y de condiciones que deben cumplirse para realizar las transiciones entre ellos. El sistema se encuentra siempre en un estado activo, a partir de la información que leamos en el sistema y los condicionales de las transiciones, hemos de decidir si debemos cambiar hacia otro estado.

Algunas de las ventajas asociadas a la utilización de la MEF para el cálculo de las decisiones de la IA son las siguientes:

- Su simplicidad las hace perfectas para desarrolladores con poca experiencia, ya que se diseñan e implementan de manera rápida.
- En el caso de las MEF deterministas se pueden predecir fácilmente las situaciones que provocan la transición, lo cual permite un mayor control de la depuración de la IA.
- Se trata de un sistema flexible, fácil de extender con nuevos estados y transiciones sin complicar el funcionamiento global.
- Proporcionan una representación gráfica del comportamiento que permite saber fácilmente cómo ir de un estado a otro y qué condiciones son necesarias para la transición.
- Los recursos necesarios para ejecutar una MEF son muy pocos.

Por otro lado, las desventajas de este tipo de sistemas son las siguientes:

- Normalmente, son sistemas muy predecibles, sobre todo en el caso de una MEF determinista puede no resultar conveniente en algunos dominios como los videojuegos.
- Sin un buen diseño podemos tener problemas a la hora de su implementación, sobre todo si se trata de una máquina con muchos estados diferentes.
- Si se implementa un sistema grande usando MEF puede ser difícil de administrar y mantener sin un buen diseño. Las transiciones entre estados pueden causar cierto grado de factor *spaghetti* al intentar seguir una línea de ejecución.
- Sólo sirve para aquellos casos en los que podemos separar el comportamiento en diferentes estados independientes y donde podemos definir unas transiciones claras entre estados.
- Las MEF pueden utilizarse en diferentes niveles de la programación de la IA. Lo más normal es utilizarlas para la programación de decisiones estratégicas o tácticas.

1.4.3. Redes Neuronales

Las redes neuronales artificiales son un intento de reproducir este tipo de estructura biológica desde un punto de vista muy simplificado. Básicamente, entendemos una red neuronal como un sistema que recibe un conjunto de entradas (percepciones que recibe del sistema), las procesa por medio de un conjunto de elementos ocultos que se encuentran conectados entre sí (en un sistema parecido a las conexiones de las neuronas) y que produce un resultado de salida (la acción que se toma).
En el mundo de los video-juegos [Urgellés, 2008], los más populares son aquellos que son capaces de ofrecerles a los usuarios un reto significativo a su inteligencia sin importar el nivel de destreza de los mismos. Este reto va variando a medida que el jugador adquiere las habilidades necesarias para desempeñarse dentro del juego.
El uso de las Redes Neuronales (RN) en los entornos virtuales se hace un poco complejo debido a que necesita pasar por un proceso de aprendizaje, dicho proceso puede ser de varios tipos.

Aprendizaje supervisado

El aprendizaje supervisado entrena a la red neuronal dando una salida deseada para cada vector de entrada. Dicho de otra manera consiste en que la red dispone de los patrones de entrada y los patrones de salida que deseamos para esa entrada y en función de ellos se modifican los pesos de las sinapsis para ajustar la entrada a esa salida. Este es el caso del método *Backpropagation* [Andy Trujillo Rivero, 2008]. La belleza de esta aproximación es la de ser capaz de entrenar la red tanto de manera *online* como *offline*. No obstante, adquirir los datos a usar en el entrenamiento es un proceso importante, para juegos existen varias opciones.

Aprendizaje no supervisado

Con el aprendizaje no supervisado, no se necesita un comportamiento feedback para que la red aprenda. En algunos casos, esto se logra usando redes neuronales como parte de un esquema de aprendizaje nivelado mucho más alto.

Aprendizaje reforzado

En el aprendizaje reforzado la información que se le proporciona a la red es mínima solo de le indica si la respuesta de la red es correcta o incorrecta.

La implementación de una red neuronal tiene dos fases:

1. Fase de entrenamiento. En una primera fase, hemos de enseñar a nuestra red neuronal para que sepa relacionar un conjunto de entradas conocidas con una salida esperada que también es conocida. El objetivo de este entrenamiento es definir los pesos de las ecuaciones de estado de todas las neuronas. De esta manera, vamos repitiendo el entrenamiento hasta que demos con los valores que hacen que para las entradas que

conocemos el sistema nos brinde siempre la salida que queremos.

2. Fase de explotación. Cuando ponemos la red neuronal en un caso real, la red recibe todo tipo de entradas, posiblemente muchas más de las que hemos utilizado durante la fase de entrenamiento. A partir de las fórmulas que tenemos en la red neuronal obtenidas en el entrenamiento, nuestro sistema nos dará la acción que él cree que es la más acertada.

El procesamiento de datos
El objetivo del proceso de aprendizaje es permitir a la red neuronal ejecutarse bien para todos los casos en los que es invocada. Parte de esto se logra mediante la generalización intrínseca, y otra parte es responsabilidad de los diseñadores. Esto se hace para procesar los datos durante el entrenamiento y la simulación. Finalmente, uno de los mejores usos de las redes neuronales es el de implementar la capacidad de anticipar las acciones de los jugadores.

1.4.4. Lógica Difusa

La lógica difusa se utiliza en la IA cuando la complejidad del proceso en cuestión es muy alta y no coexisten modelos matemáticos precisos, para procesos altamente no lineales. Cuando se manejan definiciones y conocimiento no estrictamente específico (impreciso o subjetivo). La lógica difusa generalmente se ve muy vinculada a otras técnicas de IA.
Aplicaciones generales
La lógica difusa se utiliza cuando la complejidad del proceso en cuestión es muy alta y no existen modelos matemáticos precisos, para procesos altamente no lineales y cuando se envuelven definiciones y conocimiento no estrictamente definido (impreciso o subjetivo).
Esta técnica se ha empleado con bastante éxito en la industria, y cada vez se está usando en gran multitud de campos. La primera vez que se usó de forma importante fue en el metro japonés, con excelentes resultados. A continuación se citan algunas de sus aplicaciones.

- Sistemas de control de acondicionadores de aire.
- Sistemas de foco automático en cámaras fotográficas.
- Electrodomésticos familiares (frigoríficos, lavadoras).
- Optimización de sistemas de control industriales.
- Sistemas de escritura.
- Sistemas expertos del conocimiento (simular el comportamiento de un experto humano).
- Tecnología informática.
- Bases de datos difusas: Almacenar y consultar información imprecisa. Para este punto, por ejemplo, existe el lenguaje FSQL.

Ventajas e inconvenientes

Como principal ventaja, cabe destacar los excelentes resultados que brinda un sistema de control basado en lógica difusa: ofrece salidas de una forma veloz y precisa, disminuyendo así las transiciones de estados fundamentales en el entorno físico que controla.
También está la indecisión de decantarse bien por los expertos o bien por la tecnología (principalmente mediante redes neuronales) para reforzar las reglas heurísticas iniciales de cualquier sistema de control basado en este tipo de lógica. No se debe usar cuando algún modelo matemático ya soluciona eficientemente el problema, cuando los problemas son lineales o cuando no tienen solución.

1.4.5. Algoritmos Genéticos

En los años 70, de la mano de John Henry Holland, surgió una de las líneas más prometedoras de la IA [Argente, 2002], la de los algoritmos genéticos. Son llamados así porque se inspiran en la evolución biológica y su base genético-molecular. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados que sobreviven y cuáles los menos aptos, que son descartados. Es incluido dentro de los algoritmos evolutivos, que incluyen también las estrategias evolutivas, la programación evolutiva y la programación genética.
Los algoritmos genéticos nos permiten modelar el proceso de selección natural para poderlo aplicar a la resolución de todo tipo de problemas.

Aplicación de los algoritmos genéticos a los videojuegos

En los videojuegos, los algoritmos genéticos se suelen utilizar cuando existe bastante incertidumbre sobre las características del jugador. En este caso, es muy difícil diseñar los adversarios para que sean capaces de encontrarse al mismo nivel que cualquier jugador. Estos algoritmos han sido implementados en varios juegos entre los que se pueden mencionar juegos de damas, Juegos de adversarios, juegos de Ajedrez, entre otros.

Capítulo 2

Solución Propuesta

2.1. Introducción al capítulo.

En este capítulo se seleccionará la herramienta necesaria para cumplir los objetivos de la presente investigación, tomando en cuenta las analizadas en el capítulo anterior, se puntualizan algunas ventajas y desventajas tomadas en cuenta para la selección de dicha herramienta. Se siguen ciertos criterios para elegir la herramienta adecuada y se le realizará la ingeniería inversa para analizar el funcionamiento lógico de la misma.

2.2. Selección de la herramienta

La selección de la herramienta constituye un punto crítico en el logro de los objetivos propuestos para esta investigación, es de vital importancia tener en cuenta la diversidad de *software* existentes y anteriormente analizados que pueden servir como base para la solución requerida. De los *software* mencionados en el primer capítulo se analizarán *Robocode*, *Simbad*, *SecondLife*, *OpenSim* y *OpenSteerDemo* teniendo en cuenta los requisitos que a continuación se plantean y que es necesario tener en cuenta para estar en concordancia con las políticas de la universidad y en consecuencia con las potencialidades económicas del país, por lo tanto, los que siguen a continuación son los aspectos fundamentales a tener presente:

- Debe ser una herramienta de código abierto.
- Facilidad de agregar nuevos módulos.
- Lenguaje que requiere para la programación de los comportamientos.
- Documentación disponible para la programación.
- Experiencia de uso en la Universidad.

- Utilización del mínimo de recursos de cómputo.

Herramientas	Código abierto	Facilidad de agregar nuevos módulos	Lenguaje de programación.	Abundante documentación	Experiencia de uso en la universidad	Recursos de cómputo
Robocode	si	no	Java	si	si	medio
OpenSim	si	si	Liden Scripting Languaje (LSL)	si	si	medio
SecondLife	si	no	LSL	no	no	elevado
Simbad	si	si	Java	si	no	medio
OpenSteerDemo	si	si	C++	si	si	bajo

Tabla 2.1: Comparación entre los simuladores.

Después de haber mostrado los parámetros anteriormente mencionados en cada una de las herramientas se hace necesario un análisis para la selección de la herramienta. En el caso de *Robocode* y Simbad usan el lenguaje Java, el cual no se utiliza en el proceso docente de nuestra facultad y muy poco en la producción, lo que llevaría a los estudiantes a aprender el lenguaje para poder utilizar dichas herramientas, lo que se toma como una desventaja fundamental para seleccionar una de estas herramientas. En el caso de *SecondLife* se necesita poseer conocimiento del lenguaje LSL, además no está disponible en la universidad. Teniendo en cuenta los requisitos anteriormente analizados se llega a la conclusión de que solo 2 de ellas cumplen con la mayoría de los requisitos planteados las mismas son OpenSim y *OpenSteerDemo*. Para ello es necesario analizar de cada una de ellas las características particulares que poseen y que de alguna manera se podrían aprovechar.

OpenSim

Se instala en un servidor o servidores propios funcionando en paralelo (Linux o Windows) lo que nos permite escalar la capacidad del mismo a nuestras necesidades, pero esto requiere de acceso a la red y solo se podría usar cuando la máquina cuente con este servicio. Brinda una gran cantidad de comportamientos predeterminados para los caracteres que podrían ser usados en la simulación de las técnicas. En la facultad 5 se encuentra instalado pero aún es muy pobre la creación de edificios, parques entre otros, debido a que cuando se monta el servidor hay que crear todo el entorno. Brinda varios servicios que pueden o no ser necesarios

para la aplicación. Aún no cuenta con el suficiente desarrollo de su entorno como para poder definir vías para aplicar las técnicas de IA. No se cuenta aún con suficiente personal para agilizar este proceso. Se necesita poseer conocimiento de *Liden Scripting Languaje* para la programación de los comportamientos que se le pueden aplicar a los objetos creados en el entorno.

OpenSteerDemo

OpenSteerDemo se copia el proyecto y se compila, está disponible para (Linux y Windows). Posee además varios comportamientos definidos para los agentes que podrían ser usados en la simulación. Usa la biblioteca *OpenSteer* que está diseñada para los comportamientos de locomoción de los agentes, lo cual sirve como apoyo al curso optativo Desarrollo de Elementos Inteligentes que se imparte en la facultad 5. Solo es necesario tener conocimiento del lenguaje C++ que es el que se estudia en la facultad 5.
Después de haber analizado estas dos herramientas en detalle se decide que OpenSim reúne facilidades que pudieran ser aprovechadas en un futuro cercano, sin embargo *OpenSteer* es la herramienta que posee mejor capacidad de adaptación ya que puede ser utilizada en cualquier máquina sin necesidad de muchos requerimientos de Hardware. Por todo lo anteriormente planteado *OpenSteerDemo* reúne más condiciones para cumplir los objetivos que se ha trazado esta investigación. Para un análisis más detallado de esta herramienta es necesario realizar la ingeniería inversa.

2.3. Ingeniería inversa del OpenSteerDemo

La ingeniería inversa es un método de resolución, aplicarla al *software* supone profundizar en el estudio de su funcionamiento, hasta el punto de que podemos llegar a entender, modificar, y mejorar dicho *software*. En general si el producto u otro material que es sometido a la ingeniería inversa fue obtenido en forma apropiada, entonces el proceso es legítimo y legal. De la misma forma, pueden fabricarse y distribuirse, legalmente, los productos genéricos creados a partir de la información obtenida de la ingeniería inversa, como es el caso de algunos proyectos de *software* libre ampliamente conocidos. Ejemplo de esto es el simulador Simbad, el cual se detalló en el capítulo anterior.
La reingeniería e ingeniería inversa prolongan la vida del *software*, dado que es una labor estratégica, es conveniente conocer cuando realizar la tarea de reingeniería para una aplicación y cuándo es más rentable sustituirla e implementar una nueva. Las aplicaciones para el primer paso, son aquellas en la que se produce las siguientes situaciones:

- Fallos frecuentes, que son difíciles de localizar.
- Son poco eficientes, pero realizan la función esperada.

- Dificultades en la integración con otros sistemas.
- Calidad pobre del *software* final.
- Resistencia a introducir cambios.
- Pocas personas capacitadas para realizar modificaciones.
- Dificultades para realizar pruebas.
- El mantenimiento consume muchos recursos.
- Es necesario incluir nuevos requisitos, pero los básicos se mantienen.

Desarrollo de software para su reutilización

Consiste en desarrollar una aplicación usando *software* ya existente. Cualquier profesional lo utiliza para la construcción de un sistema con la intención de reutilizar partes de él en futuros desarrollos. Estudios realizados determinan que la práctica de reutilización del *software* en un proyecto aumenta la productividad durante el desarrollo de dicho proyecto. Sin embargo, la reutilización del *software* no cubre solo el reuso de códigos, abarca todo un amplio campo de posibilidades en los diferentes niveles, metodología, ciclos de vida, planes del proyecto, especificaciones de requisitos, diseños, arquitectura de *software*, planes de validación, juegos de prueba y documentación.

2.3.1. Metodología de desarrollo de software a utilizar

Es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. Una metodología de desarrollo de *software* se refiere a un *framework* que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. A lo largo del tiempo, una gran cantidad de métodos han sido desarrollados diferenciándose por su fortaleza y debilidad. Estos frameworks son a menudo vinculados a algún tipo de organización, que además desarrolla, apoya el uso y promueve la metodología. No existe una metodología específica para un proyecto determinado, sino que esta puede ser adaptada según las necesidades y requisitos del proyecto.

Comparación entre tipos de metodologías

Tradicionales

Las metodologías tradicionales se usan por lo general para proyectos muy grandes donde es necesario una cantidad de personal considerable, y por ello es preciso llevar un control de todo lo que se genera en el proyecto de manera minuciosa.

Ágiles

Las metodologías ágiles están dirigidas a proyectos no muy grandes donde no se necesita generar muchos artefactos, sino que se centra en los más importantes.

Selección de la metodología a utilizar

Por las características antes expuestas de cada tipo de metodología es que se propone usar las metodologías ágiles. Esta investigación no requiere llevar un control tan detallado de los artefactos que se generan, sino que se centra en lo más importante: el funcionamiento lógico.

Metodologías ágiles

Scrum: Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de *software* se realiza mediante iteraciones, denominadas sprints.

OpenUP: El Open UP es un Proceso Unificado que aplica propuestas de gestión ágil como son: desarrollo iterativo e incremental dentro del ciclo de vida, tratando de ser manejable en relación con el RUP(Rational Unified Process), ya que mantiene sus características esenciales. Debemos estar claros que el uso de Open UP se debe realizar cuando se tiene un equipo pequeño, cuando se quiere evitar ser cargado excesivamente con metodologías formales improductivas.

La metodología que se utilizará por ser la más conveniente es **OpenUp** ya que scrum es para la gestión de proyecto y lo que se quiere es una metodología para realizar el análisis del *software* y **OpenUp** es ideal para esto, además basa su funcionamiento en RUP.

2.3.2. Herramientas de desarrollo de software

Para el desarrollo del *plugin* es necesario utilizar herramientas de desarrollo de *software*. A continuación se describirán algunas para seleccionar la más adecuada.

IDE

Traducido al español significa Entorno de desarrollo integrado (en inglés *integrated development environment*). Es un programa informático compuesto por un conjunto de herramientas de programación y consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). El IDE que se utiliza para la implementación del *plugin* es Visual Studio 2008.

Lenguaje de programación

El lenguaje de programación a utilizar es C++, porque es el lenguaje en que se encuentra programada la biblioteca *OpenSteer*. Dicho lenguaje se estudia en la UCI en varias facultades entre ellas la facultad 5. C++ es un lenguaje de programación derivado del lenguaje C, incorporando mecanismos que permiten la manipulación de objetos (POO).

Lenguaje de modelado

Como lenguaje de modelado se usará el Lenguaje Unificado de Modelado (UML por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

Herramienta Case

La herramienta CASE a utilizar para el análisis del *OpenSteerDemo* es Visual Paradigm. Ya que facilita la ingeniería inversa de código a modelo y código a diagrama. Soporta lenguajes como Java, C++, Esquemas XML, entre otros. Proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar.

2.3.3. Requisitos funcionales

Los requisitos funcionales se refieren a qué hace el *software*, es la base para el desarrollo posterior del proyecto [Schmuller, 2000]. Todos los requisitos que se encuentran implementados en la herramienta sirven para la implementación de la nueva extensión con las técnicas de inteligencia artificial a incorporar.
La aplicación, al iniciarse, contiene un registro con todos los *plugins* que se encuentran en la implementación de la solución y escoge uno, y lo ejecuta sin la necesidad de ninguna intervención humana, luego se pueden realizar las siguientes acciones que resultan en requisitos funcionales.

RF1: Cambiar de plugin

La herramienta le permite al usuario cambiar de *plugin* activo e iniciar la simulación del siguiente *plugin*.

RF2: Reiniciar el plugin seleccionado

La herramienta permite al usuario reiniciar el *plugin* activo en el momento que se encuentra ejecutándose. Esta funcionalidad es de vital importancia porque permite que se pueda comenzar la simulación del *plugin* en cualquier momento que el usuario estime conveniente.

RF3: Cambiar modo de cámara

La herramienta permite al usuario cambiar el modo de la cámara permitiendo visualizar la escena desde varios puntos y con distintas visualizaciones todas en perspectiva.

RF4: Cambiar de vehículo

La herramienta permite al usuario cambiar de vehículo seleccionado, cuando el *plugin* consta de varios vehículos.

RF5: Cambiar la visualización de vectores

La herramienta permite al usuario poder ver los vectores de los que se auxilian los vehículos para el cálculo de las distancias, fuerza necesaria para los distintos comportamientos, velocidad y dirección en la que se desplazan estos vehículos.

2.3.4. Actor del sistema

El actor del sistema es el usuario que necesita visualizar los resultados o comportamientos de los agentes en la simulación usando las técnicas de inteligencia artificial que posee el *plugin* seleccionado. Puede tratarse de un estudiante o profesor, en el caso de este trabajo lo seguiremos tratando como usuario porque es un término general que nos permite abarcar todos los posibles usuarios del *software*.

2.3.5. Comportamiento del sistema

Para realizar el diseño lógico del funcionamiento del sistema, es necesario investigar y definir su comportamiento como una caja negra. El comportamiento del sistema es una descripción de lo que hace, sin explicar la manera en que lo hace. Una parte de la descripción es un diagrama de secuencia del sistema el cual será explicado más adelante.

Casos de uso

- Cambiar de *plugin*.
- Reiniciar *plugin*.
- Cambiar modo de cámara.
- Seleccionar próximo vehículo.
- Cambiar visualización de vectores.

Estos casos de uso (CU) se encuentran recogidos en el diagrama de casos de uso del sistema representado en la figura 2.1.

Diagrama de casos de uso del sistema

Es el diagrama que permite representar las funcionalidades del sistema en relación con el usuario y otros sistemas externos. Esta representación se realiza desde el punto de vista del usuario y tiene al usuario como centro de atención del diagrama.

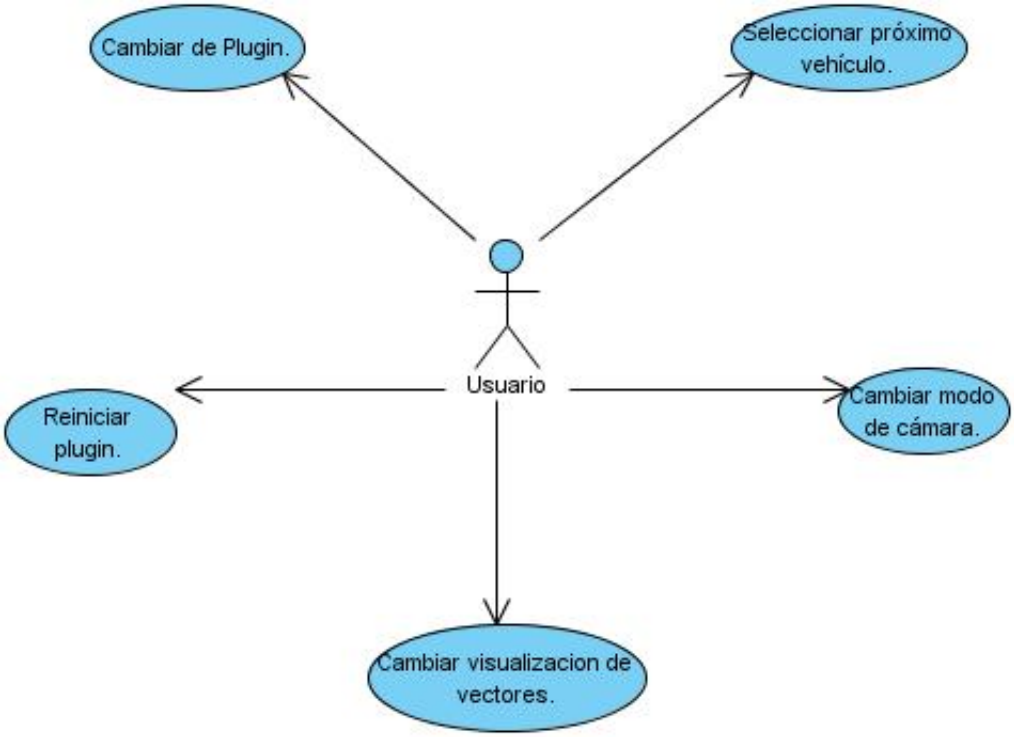

Figura 2.1: Diagrama de casos de uso del sistema.

2.3.6. Descripción de casos de uso

La descripción de los casos de uso, es donde se explica la forma de interactuar entre el sistema y el usuario. [Schmuller, 2000]Se representa mediante una tabla que contiene los actores, que son los que interactúan con el sistema proporcionando una entrada. El resumen es donde se describe de forma breve lo que hace el caso de uso, brindando una idea general de su funcionamiento. Las precondiciones son los hechos que se han de cumplir para que el flujo de evento se pueda llevar a cabo. Luego aparece el flujo de eventos, que corresponde a la ejecución normal y exitosa del caso de uso. Los flujos alternativos son los que nos permiten indicar qué es lo que hace el sistema en los casos menos frecuentes e inesperados y las poscondiciones son los hechos que se ha de cumplir si el flujo de eventos normal se ha ejecutado correctamente.

CU1 Cambiar de Plugin

Objetivo	Permitir al usuario poder cambiar de plugin activo.	
Actor	Usuario.	
Resumen	El caso de uso se inicia cuando el usuario genérico presiona la tecla ***tab*** y finaliza cuando el sistema muestra el próximo *plugin* del registro.	
Complejidad	Medio	
Requisitos de Referencia	RF 1	
Prioridad	Crítico	
Precondiciones	El usuario debe haber leído las indicaciones que aparecen en la documentación de la biblioteca.	
Postcondiciones	El sistema ejecuta el siguiente *plugin*.	
Flujo de eventos		
Flujo básico “Cambiar de *plugin*”		
	Actor	**Sistema**
	El usuario presiona la tecla ***tab***.	1.1. La aplicación verifica que la tecla presionada sea ***tab***. Busca el *plugin* siguiente y lo ejecuta.

Tabla 2.2: Descripción de caso de uso Cambiar *Plugin*.

CU2 Reiniciar Plugin

Objetivo	Permitir al usuario reiniciar la ejecución del *plugin* activo.	
Actor	Usuario.	
Resumen	El caso de uso se inicia cuando el usuario presiona la tecla ***r*** y termina cuando el *plugin* es reiniciado.	
Complejidad	Medio.	
Requisitos de Referencia	RF 2.	
Prioridad	Auxiliar.	
Precondiciones	El usuario debe haber leído las indicaciones que aparecen en la documentación de la biblioteca.	
Postcondiciones	El *plugin* reinicia su ejecución.	
Flujo de eventos		
Flujo básico "Reiniciar *plugin*"		
	Actor	**Sistema**
1.	El usuario presiona la tecla ***r***.	La aplicación verifica que la tecla presionada es ***r***. Reinicia el *plugin* activo. Imprime en la consola el nombre del *plugin* que se reinició.

Tabla 2.3: Descripción de caso de uso Reiniciar *Plugin*.

CU3 Cambiar modo de cámara

Objetivo	Permitir al usuario cambiar el modo de visualización de la escena.
Actor	Usuario.
Resumen	El caso de uso se inicia cuando el usuario presiona la tecla ***c*** y termina cuando la cámara cambia de modo mostrando otra perspectiva de la escena.
Complejidad	Medio.
Requisitos de Referencia	RF 3.
Prioridad	Auxiliar.
Precondiciones	El usuario debe haber leído las indicaciones que aparecen en la documentación de la biblioteca.
Postcondiciones	Se visualiza la escena de diferentes puntos de vista.

Flujo de eventos

Flujo básico "Cambiar modo de cámara"

	Actor	**Sistema**
1.	El usuario presiona la tecla ***c***.	La aplicación verifica que la tecla presionada es ***c***. Selecciona el siguiente modo de cámara. Activa el modo seleccionado. Imprime un mensaje en la consola con el modo de cámara activo.

Tabla 2.4: Descripción de caso de uso Cambiar modo de cámara.

CU4 Seleccionar próximo vehículo

Objetivo	Permitir al usuario cambiar de vehículo.
Actor	Usuario.
Resumen	El caso de uso se inicia cuando el usuario presiona la tecla ***s*** y termina cuando es activado otro vehículo.
Complejidad	Medio.
Requisitos de Referencia	RF 4.
Prioridad	Auxiliar.
Precondiciones	El usuario debe haber leído las indicaciones que aparecen en la documentación de la biblioteca.
Postcondiciones	La visualización de la escena va siguiendo al vehículo seleccionado.

Flujo de eventos

Flujo básico "Seleccionar el próximo vehículo"

	Actor	**Sistema**
1.	El usuario presiona la tecla ***s***.	La aplicación verifica que la tecla presionada es ***s***. Imprime en la consola que seleccionó el próximo vehículo. Selecciona el próximo vehículo Buscando desde el principio no teniendo en cuenta el vehículo anterior y hasta el final.

Flujos alternos
Nº Evento Si el *plugin* posee un solo vehículo.

	Actor	**Sistema**
1	1.El usuario genérico presiona la tecla s	1.1 La aplicación emitirá un error.

Tabla 2.5: Descripción de caso de uso Seleccionar próximo vehículo.

CU5 Cambiar visualización de vectores

Objetivo	Permitir al usuario cambiar entre visualización o no visualización de los vectores.
Actor	Usuario.
Resumen	El caso de uso se inicia cuando el usuario presiona la tecla ***a*** y termina cuando el sistema activa o desactiva la visualización de vectores.
Complejidad	Medio.
Requisitos de Referencia	RF 5.
Prioridad	Auxiliar.
Precondiciones	El usuario debe haber leído las indicaciones que aparecen en la documentación de la biblioteca.
Postcondiciones	Se activa o desactiva la visualización de vectores.
Flujo de eventos	
Flujo básico "Cambiar visualización de vectores"	

	Actor	**Sistema**
1.	El usuario genérico presiona la tecla ***a***.	La aplicación verifica que la tecla presionada sea ***a***. Cambia el estado de visualización de vectores si está activo lo desactiva y si está desactivado lo activa. Imprime el mensaje de la ejecución de la acción en la consola.

Tabla 2.6: Descripción de caso de uso Cambiar visualización de vectores.

2.3.7. Diagrama de clases

El Diagrama de Clases, es el diagrama principal de diseño y análisis para un sistema. La estructura de las clases del sistema se especifica con relaciones entre clases y estructuras de herencia. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal.

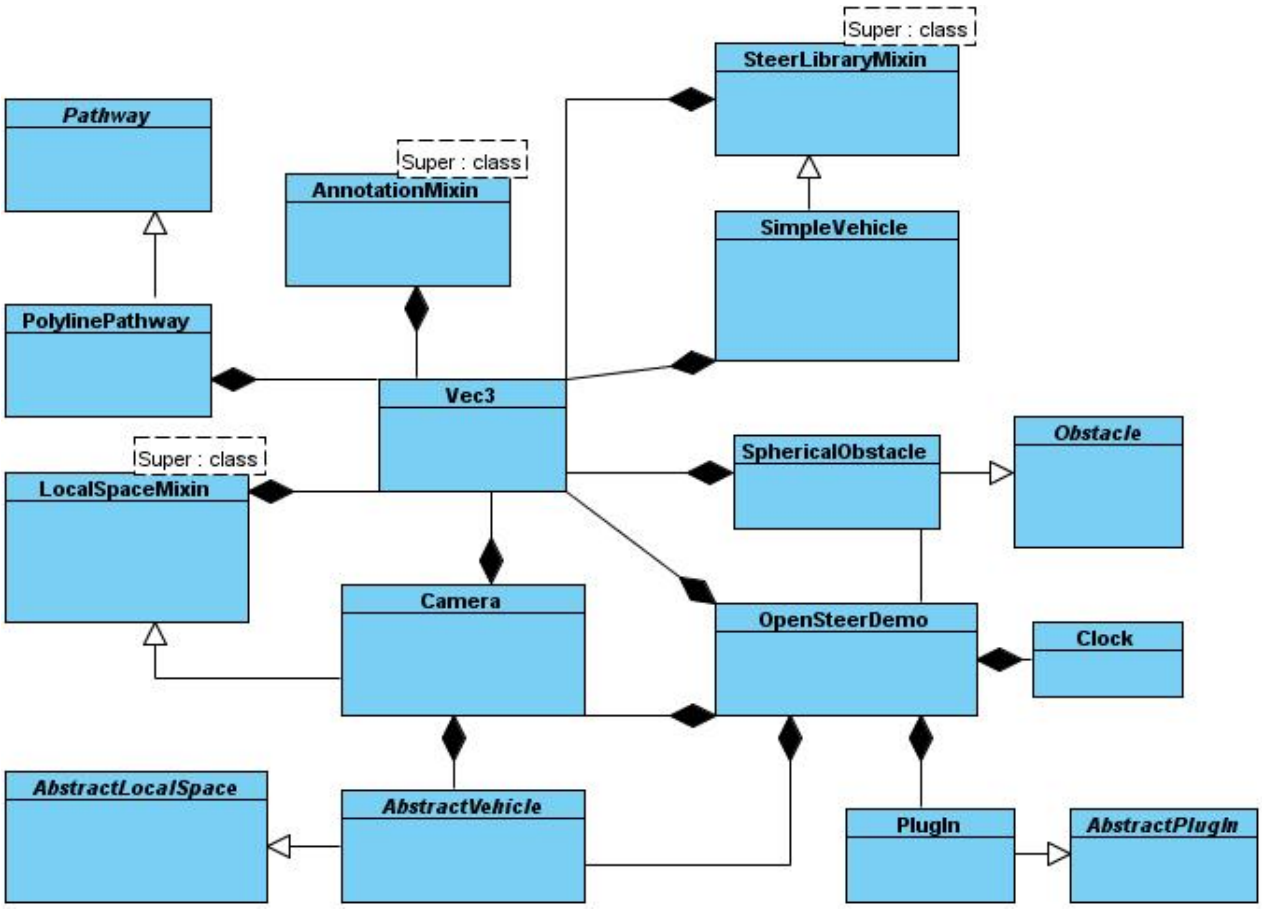

Figura 2.2: Diagrama de clase de OpenSteerDemo.

Descripción de las clases

OpenSteerDemo

OpenSteerDemo permite al usuario controlar de forma interactiva el tiempo de simulación (detener, iniciar, solo paso), seleccionar el vehículo, personaje, agente de interés, y ajustar la vista y el comportamiento de seguimiento de la cámara. Es la clase controladora principal que contiene el estado en que se encuentra *OpenSteerDemo*, provee los recursos necesarios al *plugin*

que se esté corriendo en ese momento. Los *plugin* se encuentran almacenados en un arreglo de *plugin*, también contiene una cámara, que visualiza desde distintos modos la simulación, también contiene el reloj, que lleva el tiempo de la simulación. Además contiene todos los métodos que visualizan la simulación del *plugin*.

AbstractPlugIn

Clase abstracta que contiene los métodos para dar soporte a las operaciones básicas con los *plugin*: *Métodos básicos*

- Open (abrir).
- Update (actualizar).
- Redraw (redibujar necesita que le pasen por parámetro el tiempo de simulación y el del reloj).
- Close (cerrar el *plugin*).
- Name (imprime el nombre del *plugin*).
- allVehicles (devolver un conjunto de vehículos) Reset (Recomenzar el *plugin*).

Plugin

Hereda de la clase *AbstractPlugin*, implementa las funcionalidades virtuales de la clase padre para cumplir con el protocolo *AbstractPlugIn* e incorpora métodos para funcionalidades avanzadas de los *plugin*. También contiene un registro de los *plugin*.

AbstractVehicle

Clase abstracta para la manipulación de los agentes a los que se le define el comportamiento (vehículos).

SimpleVehicle

Implementa los métodos de la clase *AbstractVehicle* por una herencia de tercero SimpleVehicle3 de la clase *SteerLibrary*. Las instancias de esta clase poseen el comportamiento de locomoción característico de *OpenSteer* heredados de la clase *SteerLibraryMixin* a través de SimpleVehicle3.

SteerLibraryMixin

Provee a *AbstractVehicle* todos los comportamientos para las instancias de vehículos en los *plugin* y provee 3 tipos de vehículos con comportamientos de distinta complejidad.

Vec3

Es la clase usada en toda la biblioteca *OpenSteer* para tratar los vectores 3D y su representación y operaciones matemáticas, además sobrecarga los operadores.

AbstractTokenForProximityDatabase

Es una clase abstracta que contiene una base de datos de los tokens que son todos los objetos que se manipulan en el espacio, contiene la base de los métodos necesarios para el cálculo

de los vecinos y la proximidad entre tokens.

AbstractProximityDatabase

Clase abstracta que se usa para el control de la proximidad de todos los tokens de un tipo en particular.

Pathway

Es una clase abstracta utilizada como base para la elaboración de caminos en el espacio y también utilizada en los comportamientos de seguir caminos que se encuentran implementados en la clase *SteerLibraryMixin*.

PolylinePathway

Es la clase utilizada para describir un camino compuesto por una polilínea que puede ser cerrada o abierta, hereda de Pathway e incorpora nuevos atributos y métodos útiles en los comportamientos de seguir caminos.

Obstacle

Clase genérica usada para representar las formas que se utilizan como obstáculo en el cálculo para la evación de los mismos.

SphericalObstacle

Se usa para la implementación de obstáculos de forma esférica, hereda de la clase Obstacle.

AbstractLocalSpace

Clase abstracta usada como interfaz para la representación de todos los objetos en el espacio.

LocalSpaceMixin

Se usa como plantilla para la implementación de *LocalSpace* para la representación de todos los objetos en la interfaz.

Clock

Esta clase es usada para controlar el tiempo en las simulaciones en *OpenSteer* para detenerlo en caso necesario y entender con mayor claridad la simulación.

Draw

Este es el módulo de visualización donde todos los gráficos del sistema se encuentran definidos, usando en este caso la librería de *OpenGL*. La función de este módulo es usar todos los datos que proporciona el *OpenSteer* para visualizarlo usando *OpenGl* como motor gráfico. En caso que se desee incorporar otra API es muy simple ya que los demás motores gráficos funcionan de la misma manera.

Concluido el análisis de los diferentes módulos por los que está compuesta la biblioteca *OpenSteer* se llega a la conclusión de que se encuentra organizada de manera tal que cada funcionalidad se implementa aparte para en caso de que la implementación de un comportamiento no requiera del uso de alguna de ellas no se incluye el módulo correspondiente. Y así lograr un trabajo más limpio y eficiente.

2.3.8. Diagramas de secuencia

Para tener una mejor perspectiva del funcionamiento del sistema se realiza el diagrama de secuencia que consta de objetos que se representan del modo usual: rectángulos con nombre, mensajes representados por líneas continuas con una punta de flecha y el tiempo representado como una progresión vertical.

El diagrama de secuencia de un sistema Es una representación que muestra, en determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema. Los diagramas se centran en los eventos que trascienden las fronteras del sistema y que fluyen de los actores a los sistemas.

CU Cambiar de plugin

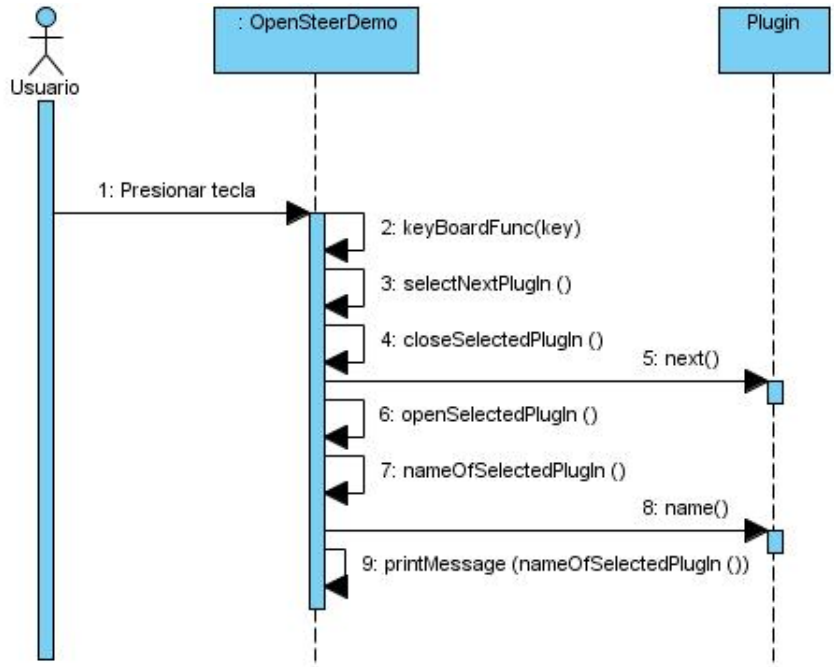

Figura 2.3: Diagrama de Secuencia de CU Cambiar de Plugin.

CU Reiniciar Plugin

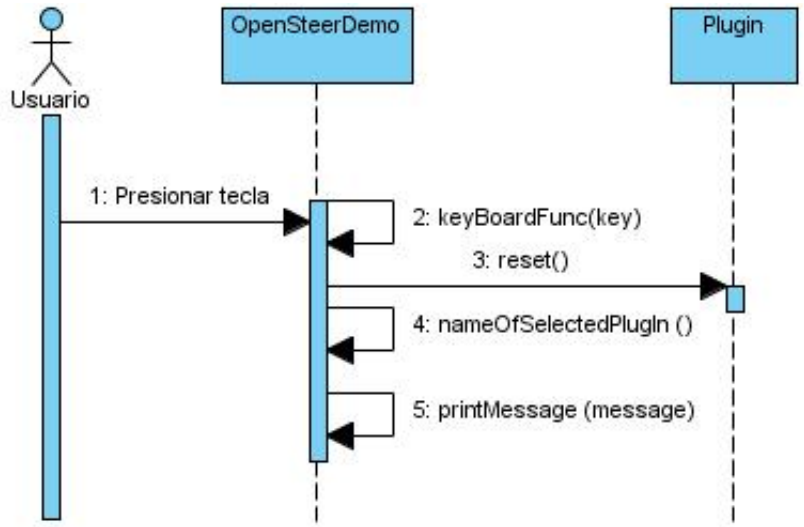

Figura 2.4: Diagrama de Secuencia de CU Reiniciar Plugin.

CU Cambiar modo de cámara

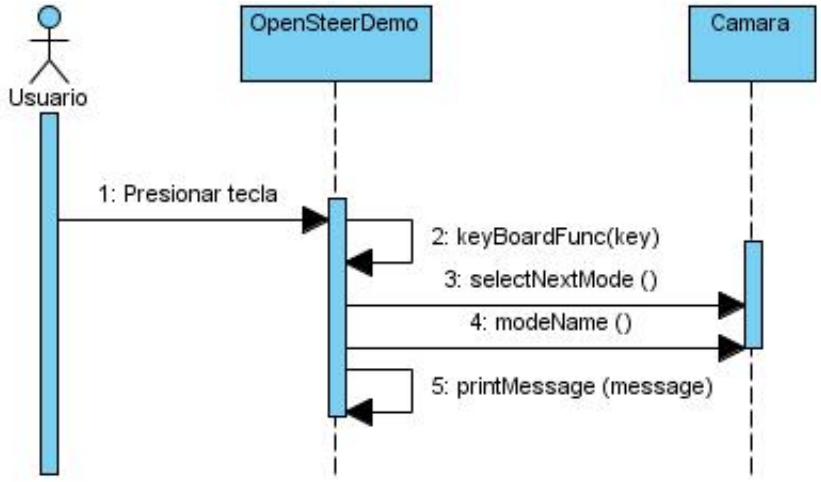

Figura 2.5: Diagrama de Secuencia de CU Cambiar modo de cámara.

CU Seleccionar próximo vehículo

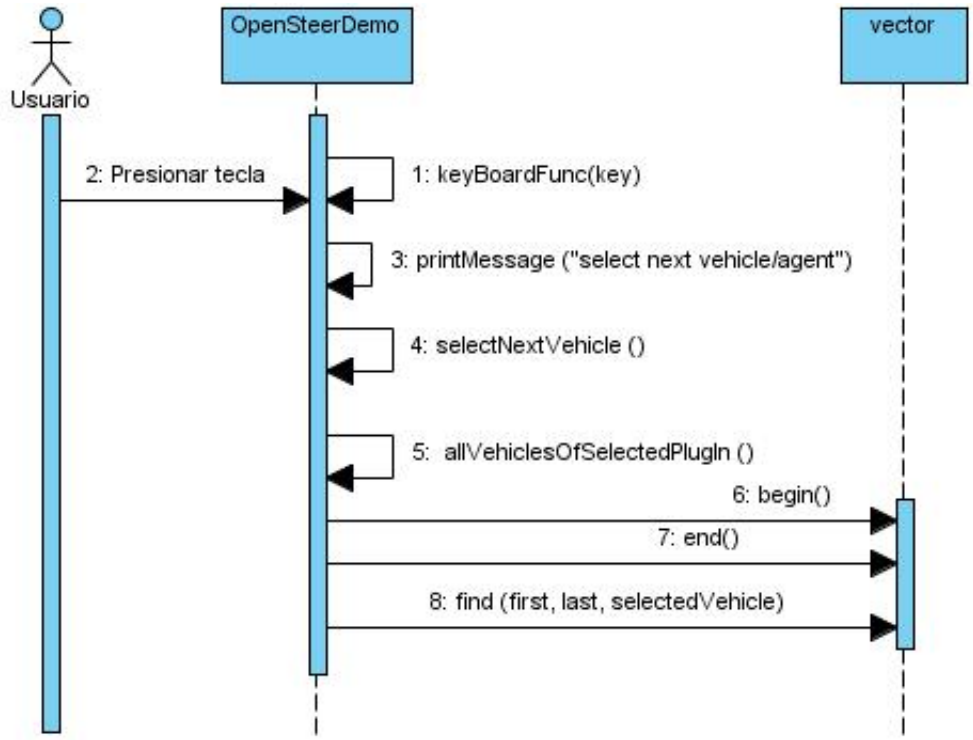

Figura 2.6: Diagrama de Secuencia de CU Seleccionar próximo vehículo.

CU Cambiar visualización de vectores

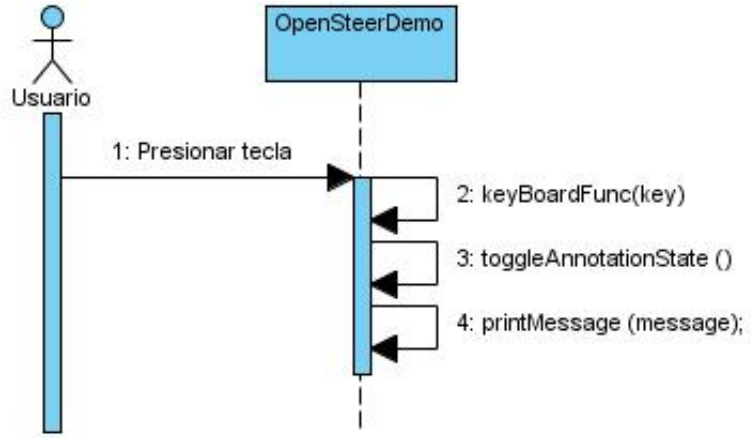

Figura 2.7: Diagrama de Secuencia de CU Cambiar visualización de vectores.

2.3.9. Diagrama de Paquete

Un diagrama de paquete muestra cómo 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 del sistema.

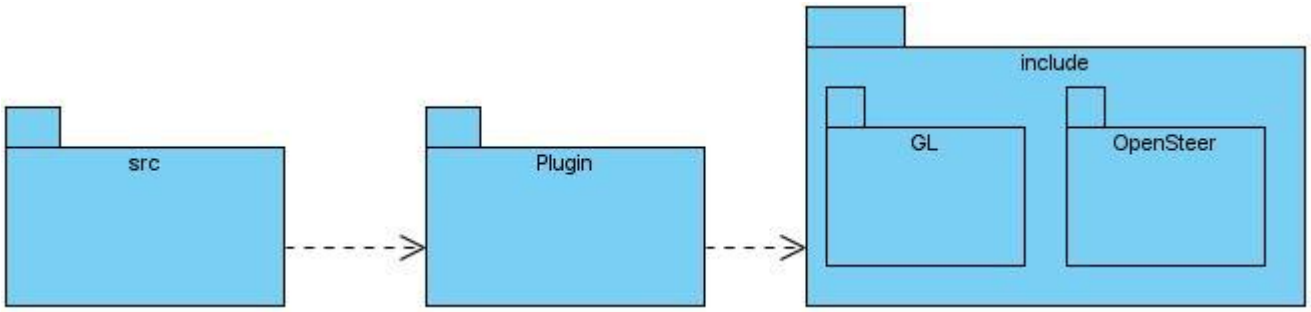

Figura 2.8: Diagrama de Paquete.

En el paquete src se encuentran los .cpp

- *Camera*.cpp
- *OpenSteerDemo*.cpp
- *Clock*.cpp
- *Draw*.cpp
- lq.c
- *main*.cpp
- *Pathway*.cpp
- *PlugIn*.cpp
- *SimpleVehicle*.cpp
- Vec3.cpp

En el paquete *Plugin* se encuentran todos los *plugin* o extensiones que serán ejecutadas por *OpenSteerDemo*, los que se mencionan a continuación son los *plugins* que trae *OpenSteer-Doemo* 0.8.2 a manera de ejemplo de cómo lograr los comportamientos que se deseen simular.

- *Boids*.cpp
- *CaptureTheFlag*.cpp
- *LowSpeedTurn*.cpp
- *MapDrive*.cpp
- *MultiplePursuit*.cpp
- *OneTurning*.cpp
- *Pedestrian*.cpp
- *Soccer*.cpp

En el paquete Include se encuentran los dos paquetes, el que contiene la biblioteca de gráfico OpenGl y la biblioteca de comportamientos de locomoción *Opensteer*. El sub-paquete GL contiene los siguientes archivos.

- *freeglut*.h
- GL.h
- *glut*.h
- *freeglutext*.h
- *GLAux*.h
- *freeglutstd*.h
- GLU.h

El sub-paquete Opensteer contiene los archivos que a continuación se mencionan:

- *AbstractVehicle*.h
- *Annotation*.h
- *Camera*.h
- *Clock*.h
- *Draw*.h
- *LocalSpace*.h
- lq.h

- *Obstacle*.h
- *OpenSteerDemo*.h
- *Pathway*.h
- *PlugIn*.h
- *Proximity*.h
- *SimpleVehicle*.h
- *SteerLibrary*.h
- *Utilities*.h
- Vec3.h

2.3.10. Representación clásica del OpenSteer

Camino

La forma de representación en *OpenSteer* de un camino es una línea o polilínea que tiene un radio que no es más que una distancia a cada lado de la polilínea cuya área pertenece al camino, es decir, que cualquier elemento que esté a una distancia de la polilínea menor que el radio del mismo se encuentra dentro del camino como se muestra en la figura 2.9:

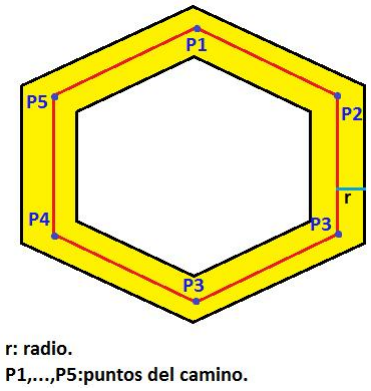

Figura 2.9: Camino

En este caso se trata de un camino cíclico o camino cerrado de radio r y puntos contenidos en el P= $< P1, P2, P3, P4, P5 >$. Aunque en la representación gráfica solo se representa la polilínea, como se muestra en la figura 2.10. También puede definirse que se dibuje toda el área del camino:

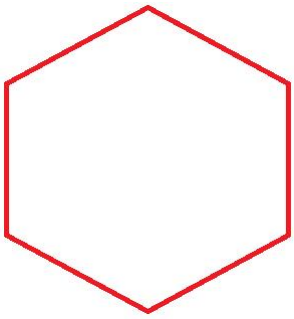

Figura 2.10: Camino

Obstáculo

Es un concepto que es muy importante en *OpenSteer* porque permite lograr un nivel alto de realismo y aunque muchas veces complica los cálculos de trayectorias logra que los agentes tengan un comportamiento mucho más complejo y realista al tener en cuenta estos objetos que se presentan en la dirección en la que se mueven. A continuación se mostrará un ejemplo de la representación del modelo de *OpenSteer* para un obstáculo esférico.

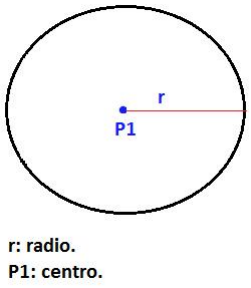

Figura 2.11: Obstáculo.

La representación gráfica es la siguiente:

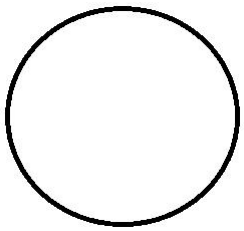

Figura 2.12: Representación de Obstáculo en la Escena.

Aunque en nuestra simulación usamos obstáculos esféricos en *OpenSteer* se pueden utilizar de distintas formas, rectangulares, etc.

Agente

Un agente está representado en *OpenSteer* de forma esférica al igual que un obstáculo aunque posee otras características como son la masa, la velocidad, la fuerza, etc. Gráficamente la representación de un agente o vehículo que es como se denomina en *OpenSteer*; se muestra en la siguiente figura 2.13:

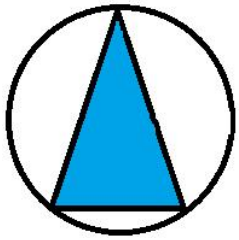

Figura 2.13: Vehículo.

2.4. Incorporación de las Técnicas de Inteligencia Artificial

Las técnicas de IA que se incorporarán a la herramienta como complemento para la práctica en la asignatura son las mencionadas en el capítulo uno, aquí se propone una vía de cómo pueden ser implementadas en el *OpenSteerDemo*, ellas son Redes Neuronales, Algoritmos Genéticos, Lógica Difusa, Máquinas de Estados Finitos y Sistemas Basados en Reglas. Se propone incorporar estas técnicas porque son las más importantes y porque se imparten de forma teórica en la asignatura de IA de la carrera de Ingeniería en Ciencias Informática . Estas técnicas pueden ser diseñadas a partir de un juego e implementarlas usando la biblioteca *OpenSteer* y como interfaz gráfica *OpenSteerDemo*. Para la implementación de cualquier técnica en *OpenSteerDemo* se debe crear un nuevo *plugin*, lo ideal sería crear uno para cada técnica con el objetivo de simplificar su desarrollo. La clase controladora de la implementación de la técnica se pone a heredar de la clase *Plugin* que es la que posee los métodos básicos como se explicó anteriormente. En el caso de los agentes que se encuentren presentes en la modelación del problema se tratan como vehículos y es a este al que se le incorporan ciertos comportamientos en dependencia de lo que se quiera visualizar. Al crear este vehículo se pone a heredar de la clase *SimpleVehicle*, que es el que posee los atributos necesarios de los agentes. El agente debe ser capaz de percibir los elementos del entorno en que se desenvuelve como son los obstáculos y el camino en caso de que sea necesario. Para la representación de los obstáculos en la escena se crea un objeto de tipo *SphericalObstacle*. En

el caso de que se requiera agregar un camino se crea un objeto de tipo *PolylinePathway* al que se le pasan los puntos. Teniendo en cuenta estos datos el agente procesa esta información y actúa en consecuencia a lo que fue programado (proporcionando una salida) comportándose de manera racional al generar una respuesta correcta o resultado esperado.

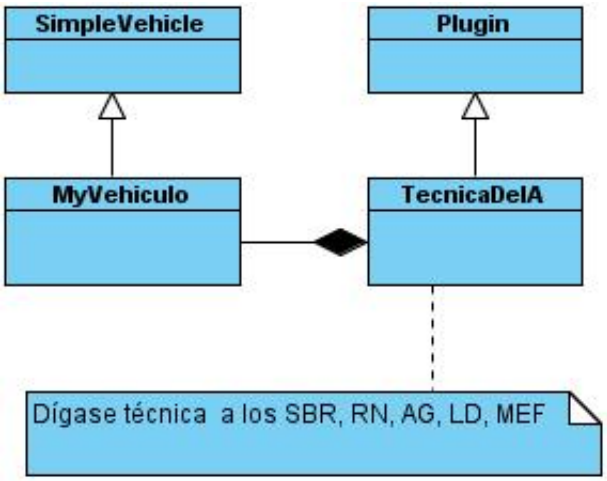

Figura 2.14: Representación de las clases.

2.4.1. Sistemas basados en reglas (SBR)

Para un mejor entendimiento se presenta un caso de estudio que ayudará a explicar con claridad la forma de utilizar los SBR.
Se tiene un entorno con varios obstáculos, enemigos y un refugio, donde el agente debe recoger tres banderas dispersas por el entorno y refugiarse en su cueva, evitando los obstáculos y evadiendo los enemigos. Esta técnica es la más sencilla de todas y se puede incorporar en cualquier situación que se presente. La cuestión se encuentra en definir las reglas según las características del entorno.
Por ejemplo teniendo en cuenta el caso de estudio descrito anteriormente las reglas pueden tener como datos la distancia a los enemigos, banderas o refugio y los objetivos o consecuentes de la regla pueden ser los comportamientos a aplicar: huir, evadir, moverse hacia un lugar, entre otros. Ejemplo:
Si *enemigo cerca* y *bandera lejos* y *obstáculo cerca* entonces *ocultar detrás de obstáculo*.

Se puede incorporar de forma muy fácil incluso con tratamiento de la incertidumbre al poder definirse valores de certeza para la cercanía, cantidad, importancia, etc, de los obstáculos, enemigos, las banderas o el refugio. Esto se puede incorporar de acuerdo a la situación que

se modela, en cualquier clase de vehículo e incluso se podría implementar en la misma clase de la técnica que en el diseño del problema es la clase controladora, en la mayoría de las situaciones se implementa en los métodos actualizar.

2.4.2. Máquina de estado finita

Se definen estados a cada vehículo y los comportamientos de los vehículos en cada estado que se pueda encontrar. Esto depende del problema que se vaya a simular, un ejemplo del uso de esta técnica en *OpenSteerDemo* se explicará en detalles en el Capítulo 3.

2.4.3. Algoritmos Genéticos (AG)

El problema fundamental del diseño de cualquier algoritmo genético consiste en la codificación de las soluciones y la función objetivo, sin dudas de las técnicas propuestas es la más compleja para utilizar e el *OpenSeerDemo* ya que no es una técnica de toma de decisiones como las demás, a raíz de lo cual se expondrá más detalladamente. Antes de que un algoritmo genético pueda ponerse a trabajar en un problema, se necesita un método para codificar las soluciones potenciales del problema, de forma que una computadora pueda procesarlas. Un enfoque común es codificar las soluciones como cadenas binarias: secuencias de 1s y 0s, donde el dígito de cada posición representa el valor de algún aspecto de la solución. Otro método similar consiste en codificar las soluciones como cadenas de enteros o números decimales, donde cada posición, de nuevo, representa algún aspecto particular de la solución. Este método permite una mayor precisión y complejidad que el método comparativamente restringido de utilizar sólo números binarios, a menudo está intuitivamente más cerca del espacio de problemas.
Los AG contienen varios aspectos de importancia que deben ser analizados antes de tratar de modelar cualquier problema mediante esta técnica.

1. Módulo evolutivo: mecanismo de decodificación (interpreta la información de un cromosoma) y función de evaluación (mide la calidad del cromosoma). Sólo aquí existe información del dominio.

2. Módulo poblacional: tiene una representación poblacional y técnicas para manipularla (técnica de representación, técnica de arranque, criterio de selección y de reemplazo). Aquí también se define el tamaño de la población y la condición de parada.

3. Módulo reproductivo: contiene los operadores genéticos.

Módulo evolutivo:
El problema de modelar esta técnica consiste en obtener estos tres módulos para poder realizar las operaciones características de los AG y obtener la solución buscada. Para lograr esto

se debe analizar, qué proporciona OpenSteer que sirva para la construcción de los módulos, en el caso del módulo evolutivo es necesario obtener una codificación de los cromosomas que no sería más que obtener la codificación de las soluciones que componen la población. Para esto es necesario mencionar que OpenSteer se especializa en comportamientos de locomoción de determinados vehículos que además tienen una serie de características como son: masa, velocidad, velocidad máxima y fuerza máxima. Para mostrar cómo trabajan los algoritmos genéticos sería interesante diseñar esta técnica para obtener una distribución de los valores a estos atributos de manera que un vehículo sea capaz de realizar un recorrido a un circuito complejo. Cada vehículo se codifica en un cromosoma en el cual cada gen contiene el valor del atributo correspondiente a esa variable.
Ejemplo:
El vehículo que se codifica a continuación posee los atributos antes mencionados masa=4, velocidad$= 3$, velocidad máxima$= 7$ y fuerza máxima $= 6$, entonces la codificación sería como se muestra en la tabla:

Masa	**Velocidad**	**Velocidad Máxima**	**Fuerza Máxima**
4	3	7	6

Este vehículo seria codificado en el cromosoma: 4376.
Luego para terminar con este módulo solo resta diseñar la función de aptitud. Como se trata de recorrer un circuito complejo o simple recorrido desde un punto de partida hasta una meta con dificultades intermedias como se muestra en la figura 2.15.

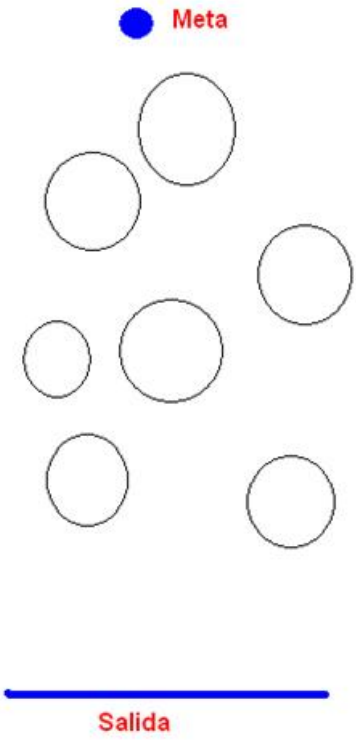

Figura 2.15: Diseño del entorno.

En este caso la función de aptitud obtendría su valor evaluando el recorrido realizado por el vehículo en cuestión al realizar la simulación del recorrido. Consiste en la evaluación individual de cómo se comportó el vehículo al evadir los obstáculos. Si no pasa por encima del obstáculo se le suman 5 a *RecObst* y en caso contrario se le resta 10. Esto permite que en caso que el vehículo le pase por encima a un obstáculo se penalice fuertemente y esto reciba prioridad sobre todo lo demás, en relación con el tiempo en que hace el recorrido.

$$F = \frac{RecObst}{TiempoF}$$

Donde:
RecObst: efectividad en sobrepasar todos los obstáculos.
TiempoF: tiempo que demora el vehículo en llegar a la meta.

Módulo Poblacional:
Para este módulo solo se necesita definir el tamaño de la población y generar esa cantidad de vehículos con los valores de los atributos aleatoriamente y definir los criterios de selección y reemplazo que pueden utilizarse con los métodos diseñados para esto.

Entre los métodos de selección se encuentran:
Selección elitista: se garantiza la selección de los miembros más aptos de cada generación. La mayoría de los algoritmos genéticos no utilizan elitismo puro, sino que son una forma modificada por la que el individuo mejor o algunos de los mejores, son copiados hacia la siguiente generación en caso de que no surja nada mejor.
Selección proporcional a la aptitud: los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.
Selección por rueda de ruleta: Es una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores.
Selección escalada: al incrementarse la aptitud media de la población, la fuerza de la presión selectiva también aumenta y la función de aptitud se hace más discriminadora. Este método puede ser útil para seleccionar más tarde, cuando todos los individuos tengan una aptitud relativamente alta y sólo les distingan pequeñas diferencias en la aptitud.
Selección por torneo: se eligen subgrupos de individuos de la población y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.
Selección por estado estacionario: la descendencia de los individuos seleccionados en cada generación vuelven al acervo genético pre-existente, reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.
Selección por rango: a cada individuo de la población se le asigna un rango numérico basado en su aptitud, y la selección se basa en este rango, en lugar de las diferencias absolutas en aptitud. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.
Selección generacional: la descendencia de los individuos seleccionados en cada generación se convierte en toda la siguiente generación. No se conservan individuos entre las generaciones.
Selección jerárquica: los individuos atraviesan múltiples rondas de selección en cada generación. Las evaluaciones de los primeros niveles son más rápidas y menos discriminatorias, mientras que los que sobreviven hasta niveles más altos son evaluados más rigurosamente. La ventaja de este método es que reduce el tiempo total de cálculo al utilizar una evaluación más rápida y menos selectiva para eliminar a la mayoría de los individuos que se muestran poco o nada prometedores y sometiendo a una evaluación de aptitud más rigurosa y computacionalmente más costosa sólo a los que sobreviven a esta prueba inicial.
Módulo reproductivo:
Para este módulo se definen los operadores genéticos que se podrían utilizar, los tradicionales cruzamiento y mutación.

Es importante tener en cuenta que al realizar estos operadores la velocidad no sobrepasará la velocidad máxima y en caso de que en el cruzamiento esto sucediera, la velocidad se quedaría en la velocidad máxima.
La condición de parada del algoritmo sería una cantidad de operaciones definidas y la solución sería la mejor solución obtenida en la corrida completa del algoritmo.
Para la implantación de un algoritmo genético en *OpenSteer* al igual que en las demás técnicas se crea una clase AG que hereda de la clase *plugin*. En esta clase es necesario tener lo referente a los tres módulos mencionados anteriormente y además algunos atributos adicionales para la condición de parada. Para los individuos es necesario definir una clase que herede de *SimpleVehicle* y se le define un atributo para la codificación de las soluciones. La clase AG se define de forma que se comience a correr el algoritmo genético en el método *open* y se debe definir los métodos de selección y demás elementos necesarios de manera que se utilice el método *update* como corrida de la función objetivo y se obtenga el resultado solo cuando el vehículo seleccionado (solución que se está evaluando en ese momento) llegue a su meta o termine el recorrido según sea el caso. Como se puede apreciar la dificultad de la implementación recae en este último detalle, la implementación de la clase AG debe realizarse cuidadosamente y en cuanto a la estructura de las clases es la misma que la de la implementación general ya explicada.

2.4.4. Redes Neuronales

Para el diseño de esta técnica se debe analizar uno de los problemas más elementales de OpenSteer y es cuando se tiene un comportamiento complejo de un vehículo que incluye a otros comportamientos más simples como es el caso que se analizará en el capítulo 3. En ese caso se tiene un comportamiento en el que el vehículo debe llegar a un lugar determinado (*xxxsteerForSeek* en *OpenSteer*), evadir obstáculos (*steerToAvoidObstacles* en *OpenSteer*) y evadir a los vecinos (*steerToAvoidNeighbors* en *OpenSteer*). Para esto es necesario definir una prioridad entre los comportamientos (esto no es más que multiplicar el vector resultante de la llamada al método por una constante que sería la prioridad). Lograr que estas prioridades estén asignadas de manera que el comportamiento general sea el adecuado es una tarea difícil. En este caso lograr que estos pesos se asignen de manera eficiente y automática sería un gran paso en la solución de cualquier problema de este tipo. Esto se puede lograr mediante una red neuronal, la propuesta que se hace es una red neuronal de tipo Mapa Auto-Organizado con una pequeña variación:

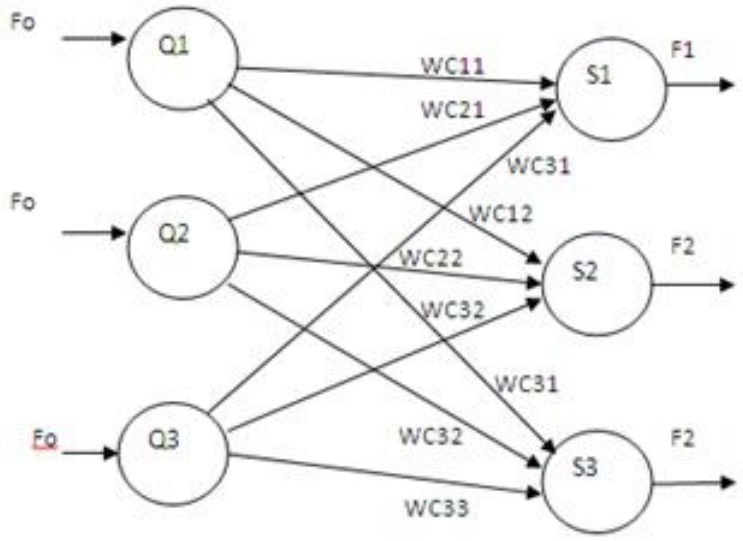

Figura 2.16: Versión del mapa de Kohonen o mapa auto-organizado.

Donde:

Fo: es la distribución de una configuración de los comportamientos.

Qn: ofrece como salida el resultado del comportamiento n.

Wcij: es el peso o prioridad destinada para el comportamiento i para la salida j.

Fn: el vector fuerza que resulta de la llamada a los comportamientos multiplicado por el peso de la arista y simulado el recorrido con esa fuerza y evaluando el comportamiento con la misma función utilizada en algoritmos genéticos.

Luego se toma la mayor salida Fn y se pasa como entrada (Fo) en la siguiente iteración, la condición de parada es el número de iteraciones.

Téngase en cuenta que la situación del recorrido es la misma que la utilizada en los algoritmos genéticos solo que se incrementan otros agentes para complejizarla aun más y en la función se le agrega al denominador un nuevo factor, sería multiplicar el tiempo por la distancia mínima entre los vehículos evadidos.

El diagrama de clase correspondiente a esta situación estaría dado de la siguiente manera, teniendo en cuenta que para cada tipo de vehículo debe implementarse una forma distinta de la red neuronal que varía de acuerdo a los comportamientos presentes en el vehículo:

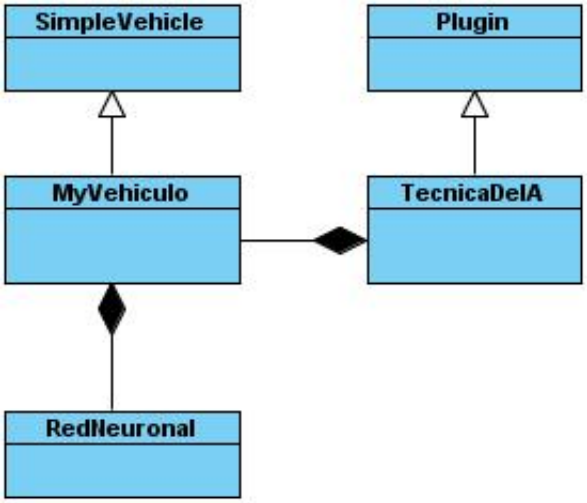

Figura 2.17: Diagrama de clase general para la incorporación de una red neuronal.

2.4.5. Lógica difusa

Esta técnica se puede integrar con cualquiera de las otras técnicas solamente fuzificando las características de cualquier elemento del entorno. Como puede ser en los cálculos de distancia entre otros, como se explicó en el caso de los sistemas basados en regla. Aunque es importante tener en cuenta que el incremento notable de los cálculos podría ser perjudicial para la simulación en tiempo real.

Capítulo 3

Análisis de resultados

3.1. Introducción al capítulo

En este capítulo se describirá cómo se implementó el plugin con una de las técnicas de inteligencia artificial incorporada, en este caso máquina de estado finita. Esto servirá de base para la implementación de las otras técnicas mencionadas anteriormente.

3.2. Descripción de la situación a modelar

En muchas ocasiones se presenta la necesidad de hacer guardia para proteger determinado medio u objeto de valor. Esta situación en la que a menudo se ven involucradas varias personas, conlleva varios comportamientos complejos como es: recorrer un determinado perímetro para vigilar las posibles amenazas. En caso de una amenaza inminente defender el objetivo y en algunos casos también la persecución del enemigo. Esta situación se podría describir como una máquina de estado finito y es precisamente lo que se modelará en este capítulo. Para esta tarea es necesario tener en cuenta que las máquinas de estado finito modelan el comportamiento de un elemento o agente y en este caso en particular es necesario tener la presencia de dos agentes, para lo cual se necesita la utilización de dos máquinas de estado finitas. En la modelación del problema se deben tener en cuenta varios aspectos como:
Entorno: El entorno es una de las cuestiones más importantes que hay que tener en cuenta en la simulación del problema porque se trata del escenario en el que los agentes se moverán, por tanto, este determinará el comportamiento de los agentes. El entorno es la fuente de información más importante en la cual se apoyarán los agentes para lograr sus objetivos. En este caso el escenario o entorno estará salpicado de obstáculos aleatoriamente de forma que los agentes tengan la necesidad de evadirlos para lograr un mayor realismo en la simulación y también complejizar los comportamientos de los agentes. En el centro de la escena se encontrará el objeto a custodiar por el Centinela y estará rodeado por un camino que seguirá este

mientras recorre el perímetro para la búsqueda de posibles amenazas.
Centinela: El Centinela es el encargado de proteger el objeto que se encuentra en el centro del entorno, el cual posee un valor para este y su objetivo es asegurar su protección, este agente sigue un camino cíclico que le permite percibir los alrededores del objeto.
Enemigo: Su objetivo es obtener el objeto que el Centinela está custodiando. Una característica importante del Enemigo es que su origen es un punto en el espacio 3D que es escogido de manera aleatoria fuera del área en la que el Centinela se encuentra patrullando para lograr así mayor realismo y que el comportamiento no sea repetitivo.

3.3. Comportamiento de los agentes

Centinela

El comportamiento de este agente es muy apropiado a su nombre, su objetivo es proteger al objeto que custodia dando recorridos alrededor de este, siguiendo un camino trazado para ese fin. Este comportamiento está determinado por la Máquina de Estado Finita que se presenta en la figura 3.1. El primer estado en el que se encuentra el Centinela al iniciar la simulación es el estado Patrullando en el que tiene que recorrer el camino cíclico que se encuentra alrededor del objeto a custodiar, además de evadir los obstáculos el Centinela también está continuamente verificando que no se encuentre el Enemigo cerca del objeto o del propio Centinela, en uno de estos casos el Centinela pasa al estado de Perseguir en el cual tiene que seguir al Enemigo. El Centinela posee un contador de calidad el cual al inicio de la simulación comienza en 50 y va disminuyendo de cantidades de 10 (cuando el Enemigo llega al objeto que el Centinela custodia), cuando el contador llega a cero indica que el Centinela ha perdido el objeto en 5 ocasiones y en tal caso se reinicia el plugin. De lo cual se concluye que el comportamiento del Centinela es menos efectivo que el del Enemigo. En el estado de perseguir el Centinela sigue al Enemigo hasta que lo captura o hasta que se aleja demasiado del objetivo, en ambos casos el Centinela pasaría nuevamente al estado de Patrullar con el comportamiento anteriormente mencionado. En el caso especial en que el Centinela atrape al Enemigo, el Enemigo pierde calidad que es necesaria para determinar la efectividad de los comportamientos de ambos agentes.

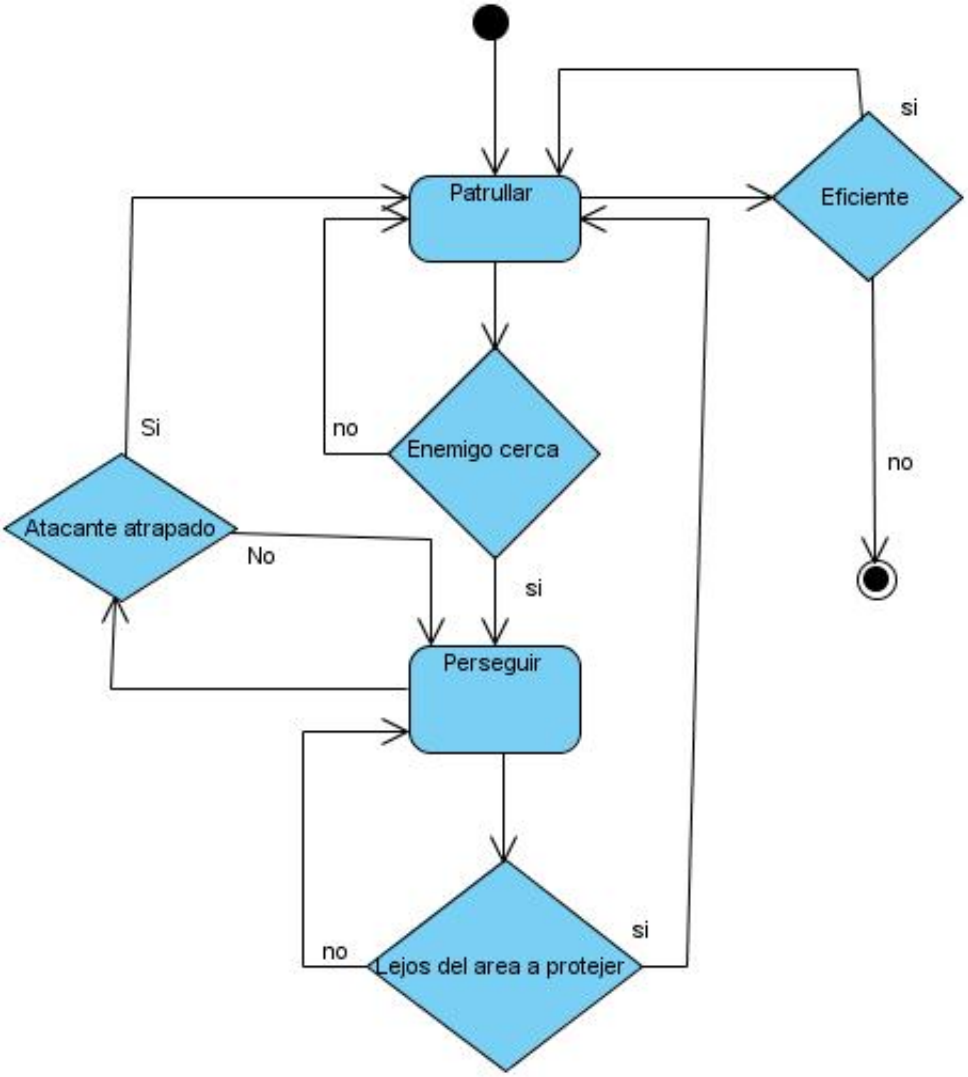

Figura 3.1: Máquina de Estado Finita del Centinela

Enemigo

El comportamiento del Enemigo es un poco más complejo que el del Centinela aun cuando la máquina de estado parece más sencilla en su representación gráfica. El comportamiento comienza al igual que el Centinela en un estado base, este es el estado Atacar, en este sentido, es necesario aclarar que el Enemigo al reiniciarse aparece en una posición aleatoria fuera del área que patrulla el Centinela, esto nos permite evaluar el comportamiento de los agentes ante situaciones que manejan datos del entorno de gran diversidad. En el estado Atacar el comportamiento del agente es determinado por varios aspectos. El principal objetivo es llegar al objeto que custodia el Centinela. En caso de que el Centinela se encuentre cerca, entonces

pasa al estado de Huir y en el caso que la vida del Enemigo se encuentre en cero, la simulación se reinicia y resulta entonces ser menos efectivo en su comportamiento que el Centinela. En el estado de Huir el Enemigo trata de alejarse del Centinela que lo persigue teniendo en cuenta los obstáculos, pero también tomando la dirección más cercana posible al objeto. Dado el caso en que llegue a obtener el objeto se descuenta la calidad del Centinela y el Enemigo es reiniciado volviendo al estado Atacar. El diseño de la Máquina de Estado Finita se muestra en la figura 3.2.

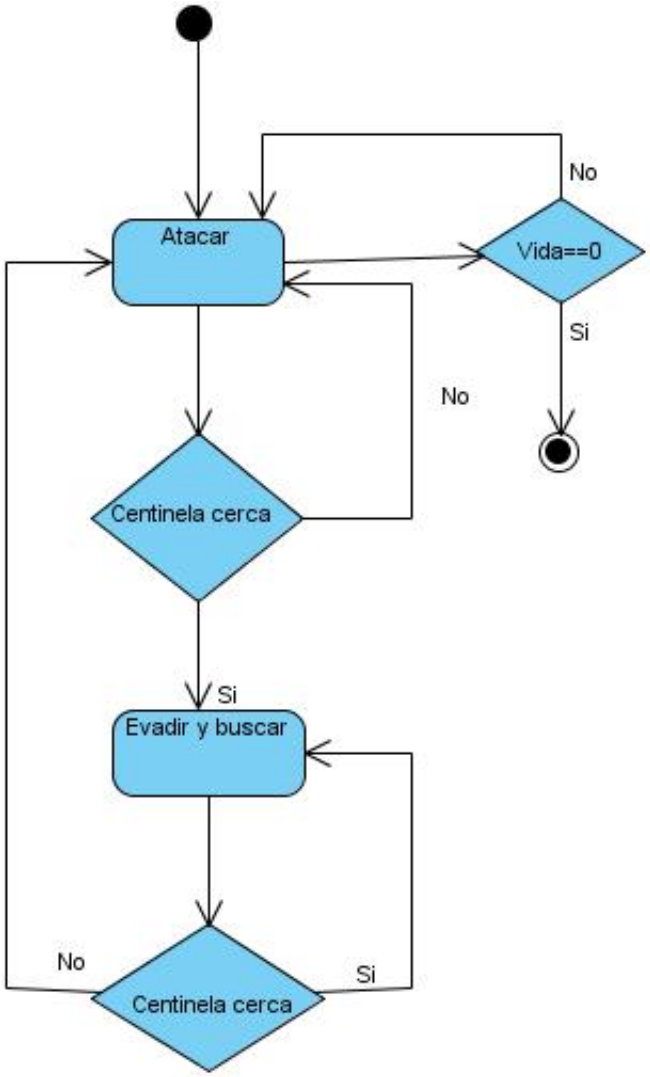

Figura 3.2: Máquina de Estado Finita del Enemigo

Hasta este punto se ha descrito el problema a simular y las cuestiones más importantes en cuanto a la modelación de esta situación, a partir de este momento se explicará cómo se realiza la representación mediante OpenSteer del problema en discusión, para esto se debe

establecer el modelo que utiliza OpenSteer para los elementos necesarios.

3.4. Implementación de la solución al problema planteado.

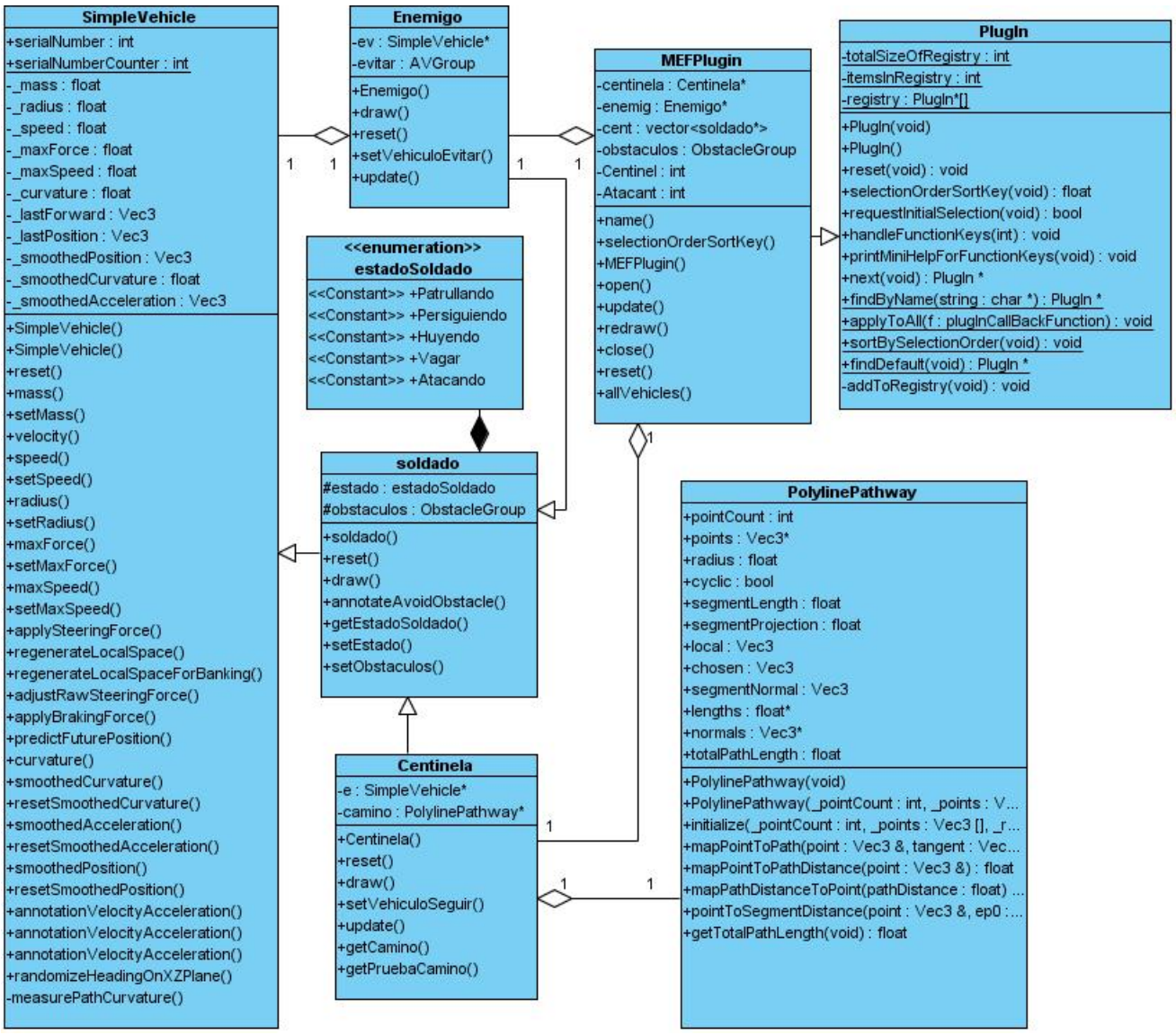

Figura 3.3: Diagrama de clases del Plugin

Anteriormente explicamos todo lo referente al problema de forma teórica, ahora se analizará la resolución práctica del problema para ello es necesario tener en cuenta que el soporte gráfico de OpenSteer es *OpenSteerDemo* que se encarga del control de las simulaciones de los plugin existentes, para la implementación del problema planteado anteriormente se debe

crear un nuevo plugin. Para tal tarea se diseñan las clases que están presentes en el siguiente diagrama de la figura 3.3.
Los comportamientos de los agentes se implementaron en los métodos update de cada una de las clases, teniendo en cuenta el estado en el que se encuentra el agente. Es en este método donde la MEF toma su sentido como técnica de comportamiento. Para la implementación de este método se utilizaron comportamientos ya definidos en la biblioteca OpenSteer y que son de gran ayuda en este sentido.

3.4.1. Soldado

Esta clase es abstracta y su objetivo es brindar una configuración básica para la implementación de los agentes Centinela y Enemigo. También es necesario destacar que hereda de la clase *SimpleVehicle* de OpenSteer para la representación del agente y obtener los comportamientos con los que cuenta un vehículo. Esta posee un atributo que permite establecer los estados en los que se encuentran tanto el Centinela como el Enemigo y este atributo es de vital importancia en la simulación de la MEF. Esta clase posee el atributo estadoSoldado, es el que utiliza la máquina de estado para definir el comportamiento de los agentes y además implementa métodos comunes de los dos agentes (Centinela y Enemigos) como son:
Reset
Este método se encarga de inicializar al vehículo con los parámetros adecuados para la simulación, estos parámetros son:
Speed: Velocidad del vehículo.
MaxForce: Fuerza máxima que se le puede aplicar al vehículo.
estado: Es el estado en el que se encuentra el vehículo, pero como es general no necesita estar en ninguno de los pertenecientes a Centinela o Enemigo.
Draw
Método encargado de dibujar al vehículo en la escena, dibuja la representación gráfica vista anteriormente. Al método drawBasic2dCircularVehicle se le pasa por parámetro un vehículo y el vector referente al color que se va a pintar el triángulo del centro.

3.4.2. Centinela.

Clase que hereda de Soldado y es la que define el comportamiento definitivo del Centinela en cada uno de sus estados, y es necesario destacar que posee un atributo de tipo *Polyline-Pathway* que no es más que el camino a seguir por el Centinela mientras se encuentra en el estado Patrullar. A continuación se describen algunos de los métodos más importantes.
Reset
El método reset del Centinela es el encargado de inicializar los parámetros del Centinela que son de importancia para la simulación, estos son:

camino: es el camino que debe seguir el Centinela en el recorrido de vigilancia mientras patrulla, se inicializa con una llamada al método *getPruebaCamino* que es el encargado de hacer el camino.
estado: es el atributo que se discutió anteriormente y se inicializa en Patrullando.
e: es el Enemigo que va a tener en cuenta el Centinela para la defensa del objetivo, inicialmente NULL por la imposibilidad de que al crearse el Centinela sepa de la existencia del Enemigo.
Update
Este método es el más importante de la clase *SimpleVehicle* porque es el que define finalmente cómo se comporta el vehículo en la escena. En este caso es también el que le da forma como tal a la máquina de estado del Centinela. En este método se utilizan conceptos importantes de la clase que es necesario explicar para su mejor entendimiento. Los conceptos son los siguientes:

- Un vehículo necesita que se le aplique determinada fuerza para lograr un movimiento, esta fuerza es un vector que le indica al vehículo hacia donde moverse.
- La fuerza se le aplica a un vehículo mediante el método *applySteeringForce*.
- Existen algunos comportamientos predefinidos para los vehículos, que no son más que métodos que devuelven el vector fuerza necesario para dirigir al vehículo de manera que cumpla con su objetivo, tal es el caso de:
  - *steerToAvoidObstacles*: Para evadir un grupo de obstáculos en un tiempo determinado.
  - *steerToFollowPath*: Para seguir un camino en determinada dirección (a favor o en contra de las manecillas del reloj).
- *forward*: Devuelve el vector del frente del vehículo.
- *annotationVelocityAcceleration* : Pinta los vectores velocidad y aceleración.
- *recordTrailVertex*: Deja un rastro del lugar por donde pasó el vehículo.

Primero se verifica que exista un posible Enemigo y si existe entonces se calcula la distancia a la que se encuentra este del objetivo, si es una distancia pequeña el Centinela pasa al estado Persiguiendo, luego se verifica que la distancia a la que se encuentra el Centinela del objeto a custodiar no es demasiado grande, en caso que lo sea se procede a pasar al Centinela al estado Patrullando sin importar la distancia a la que se encuentre el Enemigo. Luego se verifica el estado en el que se encuentra el Centinela, y se define el vector fuerza necesario para que el vehículo siga el comportamiento definido para ese estado. Esto se consigue llamando a los métodos anteriormente mencionados y en caso que se necesite varios comportamientos de los descritos se consigue sumando ambos vectores.

3.4.3. Enemigo

Esta clase al igual que la clase Centinela hereda de Soldado y es la que modela el comportamiento del Enemigo en sus diferentes estados. Al igual que la clase Centinela contiene un atributo muy importante que es imprescindible mencionar, se trata del atributo distancia, que no es más que la distancia a la que se encuentra el Enemigo del Centinela. Este atributo es el que coordina el cambio de un estado a otro. En esta clase podemos encontrar los métodos siguientes:

Reset

Esta función es la más compleja de las funciones reset que se utiliza en toda la implementación del plugin, es por la causa de generar la posición inicial del vehículo de forma aleatoria, para esta tarea se utiliza la función frandom2 que recibe por parámetro dos números reales y devuelve un número random entre esos dos argumentos.

setVehiculoEvitar

Este método es el que le dice al Enemigo el vehículo al que tiene que evitar para alcanzar su objetivo y luego este vehículo lo coloca en un vector para poder lograr el comportamiento de evitarlo pasándoselo por parámetro a la función *steerToAvoidNeighbors*.

Update

Este es, al igual que el update del Centinela, el método más importante de la clase Enemigo porque define su comportamiento. Para esto se verifican las condiciones en que se encuentra la simulación para establecer en qué estado se encuentra el Enemigo, esto se realiza mediante la verificación de la distancia a la que se encuentra el Enemigo del Centinela si es muy cerca (3 unidades) entonces se pasa al estado Huyendo, luego se verifica si el Enemigo se alejó lo suficiente del Centinela, si esto último se cumple entonces se pasa al estado Atacando. Después de establecido el estado en que se encuentra el Enemigo se procede a definir el comportamiento de acuerdo con ese estado. En el estado atacando se busca llegar al punto 5,0,5 que es donde se encuentra el objetivo (*xxxsteerForSeek*) y se trata de evitar los obstáculos (*steerToAvoidObstacles*), por el contrario, si el estado es Huyendo se trata de evadir al Centinela (*steerToAvoidNeighbors*), a la vez se evitan los obstáculos (*steerToAvoidObstacles*) y se trata de llegar al objetivo (xxxsteerForSeek). Para un mejor entendimiento de este método es necesario tener en cuenta la explicación del update del Centinela y además conocer que se integran nuevos comportamientos como son:

- *xxxsteerForSeek*: devuelve el vector fuerza que es necesario aplicar al vehículo para que busque cómo llegar a la posición que se le pasa por parámetro, en este caso 5,0,5.
- *steerToAvoidNeighbors*: devuelve el vector fuerza necesario para evadir a los vehículos que se encuentran en el vector que se le pasa como primer parámetro en el tiempo que se le pasa como segundo parámetro.

3.4.4. MEFPlugin

Esta clase es la controladora que contiene todo lo que se necesita para la simulación de nuestra situación. Esta clase hereda de la clase Plugin y es la que permite la integración con la interfaz del *OpenSteerDemo*. También es necesario mencionar que esta clase contiene un vector de *SphericalObstacle* que no son más que los obstáculos presentes en la escena y que los agentes evitarán en sus respectivos comportamientos. En esta clase podemos encontrar los métodos necesarios para el plugin como:

Name

Este es el encargado de imprimir el nombre del plugin, en este caso Patrullar-Perseguir v/s Atacar-Huir.

Open

La función de este método es de suma importancia porque es el que se encarga de inicializar todos los elementos de la simulación, tanto de los agentes como de los obstáculos y la cámara que se le asigna al Centinela, para que sea este último quien documento la simulación.

Update

Se encarga de actualizar a los dos vehículos y de verificar si los contadores de eficiencia de ambos están hábiles todavía.

Open

Se encarga de toda la parte de la visualización tanto del escenario como de los vehículos, los textos y los obstáculos, es el que proporciona la visualización de lo que ocurre en la simulación en un momento determinado.

Close

Método encargado de cerrar el plugin, esto consiste en eliminar los objetos existentes.

Reset

La función de este método es reiniciar la simulación, esto se refiere a los vehículos no al escenario.

Conclusiones

Al finalizar esta investigación se dio cumplimiento al objetivo propuesto.

- Se determinaron las técnicas de Inteligencia Artificial fundamentales que se deseaban incorporar a la herramienta.
- Se seleccionó la herramienta de Visualización Interactiva más conveniente.
- Se realizó la propuesta de incorporación de las técnicas de Inteligencia Artificial.
- Se obtuvo como resultado la implementación de una de las técnicas de IA en la herramienta seleccionada.

Recomendaciones

Se recomienda aplicar la herramienta propuesta en la asignatura de Inteligencia Artificial y el curso optativo Desarrollo de Elementos Inteligentes.

Referencias bibliográficas

[sit, 2006] (2006). Virtual worlds review.

[Tec, 2010] (2010). Tecnología e informática.

[Ope, 2011] (2011). Tecnología e informática.

[Andy Trujillo Rivero, 2008] Andy Trujillo Rivero, M. A. M. R. (2008). Definición del comportamiento de carros autónomos en un videojuego de carreras empleando redes neuronales artificiales. Master's thesis, Universidad de las Ciencias Informáticas,Cuba.

[Argente, 2002] Argente, A. (2002). *Homo cibersapiens. La inteligencia artificial y la humana*. 1era edition.

[Bergolla, 2010] Bergolla, Y. C. (2010). UtilizaciÓn de herramientas de visualizaciÓn para el aprendizaje de la inteligencia artificial. *J. Graph. Tools*, 3(3):1–9.

[Elaine Rich, 1994] Elaine Rich, K. K. (1994). *Inteligencia Artificial*. Microelectronics and Computer Technology Corporation, Carnegie Mellon University, 2da edition.

[Gonzalo Sanz, 1987] Gonzalo Sanz, L. M. (1987). *Inteligencia humana e inteligencia artificial*. Universidad de Guadalajara.

[Guerrero,] Guerrero, C. S. Los entornos virtuales de aprendizaje como instrumento de mediación. Universidad de Salamanca.

[Johnson CR, 2006] Johnson CR, Moorhead R, M. T. P. H. R. P. (2006). *Visualization research challenges report*.

[Mandow,] Mandow, L. Herrameinta software para la enseñanza de algoritmos de busqueda.

[Perez, 2002] Perez, R. E. B. (2002). *Aplicasiones de la Inteligencia Artificial*. Universidad de Guadalajara, 1era edition.

[Reynolds, a] Reynolds, C. Boid brains. Technical report.

[Reynolds, b] Reynolds, C. Boids background and update antecedentes y actualización.

[Schmuller, 2000] Schmuller, J. (2000). *Aprendiendo UML en 24 horas*. PEARSON EDUCACION, MCxico, 1era edition.

[Urgellés, 2008] Urgellés, Y. P. (2008). Aplicaciones de las redes neuronales en entornos virtuales. Master's thesis, Universidad de las Ciencias Informáticas,Cuba.

[Vilca, 2008] Vilca, H. D. C. (2008). MatemÁticas discretas para la ciencia computaciÓn.

[Vygotski, 2000] Vygotski, L. (2000). El desarrollo de los procesos psicológicos superiores.

Glosario de Términos

Glosario de términos

API: (*Application Program Interface*). Conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación. Cuando se intenta estandarizar una plataforma, se estipulan unos APIs comunes a los que deben ajustarse todos los desarrolladores de aplicaciones.

POO: Programación Orientada a Objetos. Es un paradigma de programación que permite la manipulación de objetos abstractos.

Arcade: Máquinas recreativas de videojuegos disponibles en lugares públicos de diversión.

Motor físico: Tienen como propósito hacer que los efectos físicos de los objetos creados o modelados tengan las mismas características que en la vida real.

Motor gráfico: Manejan operaciones a bajo nivel brindando al desarrollador de aplicaciones una capa de abstracción.

Realidad Virtual: Representación de escenas u objetos producidos por un sistema informático, dando la sensación de su existencia real.

Simuladores: Un simulador es un aparato que permite la simulación de un sistema, reproduciendo su comportamiento. Los simuladores reproducen sensaciones que en realidad no están sucediendo.

Sistemas de Realidad Virtual: Aplicaciones informáticas basadas en técnicas de realidad virtual (ver Realidad Virtual).

Videojuego: Programa informático normalmente asociado a un hardware específico, que recrea un ejercicio sometido a reglas, se debe lograr uno o varios objetivos, donde los jugadores pueden interactuar y tomar decisiones.