Universidad de las Ciencias Informáticas

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

Título: Módulo para el manejo de agentes inteligentes en OpenSim

Autor(es): Gabriela López León

Asiel Domínguez Martínez

Tutor(es): MSc. Mariela Nogueira Collazo

Ing. Alexey Broche Medina

Ciudad de La Habana, 2013

Año del 54 aniversario de la Revolución

Declaración de autoría

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

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

Gabriela López León

_____

Firma del Autor

Asiel Domínguez Martínez

_____

Firma del Autor

_____

Tutor: MSc. Mariela Nogueira Collazo

_____

Tutor: Ing. Alexey Broche Medina

DATOS DE CONTACTO

MsC. Mariela Nogueira Collazo.

Graduada de Ingeniera en Ciencias Informáticas en el 2007. Máster en Ingeniería de Software e IA por la Universidad de Málaga. Pertenece a la Línea Navegación y Comportamiento Inteligente del Departamento de Visualización y Realidad Virtual del CEDIN. Profesora de la Facultad 5. Con 5 años de experiencia de trabajo.

Ing. Alexey Broche Medina.

Graduado de Ingeniero en Ciencias Informáticas en el 2009. Pertenece a la Línea Navegación y Comportamiento Inteligente del Departamento de Visualización y Realidad Virtual del CEDIN. Especialista general de la Facultad 5. Con 3 años de experiencia de trabajo.

Agradecimientos

Gracias a nuestro Dios y Señor, Jesucristo, por su amor y misericordia, por la salvación, por darnos vida en abundancia, por ser nuestra ayuda en medio de las pruebas y dificultades, a Él sea toda la gloria y la honra por los siglos de los siglos. Amén.

A mis padres, mis hermanas por haber padecido conmigo cada instante de preocupación de, insomnio.

A mis hermanos en Cristo Jesús, por ser una familia especial, por estar cerca y haber compartido momentos únicos en mi vida, Dios los bendiga mucho más.

A mis compañeros de aula, de tesis, de apto a todos los que de una forma u otra ayudaron a ser más placentera mi estancia en la universidad.

Gracias a todos.

Gabriela.

A mi madre, por ser una madre y padre, por enseñarme a decidir mi propio camino, haciendo de mí un hombre de decisión. A mi abuela y abuelo por su cariño, a mis tías Belkis y Tania, por ser mis segundas madres, a mi primo Andy, por ser un hermano para mí. A Yoesdamy por ayudarme y desde pequeño incitarme a ser alguien en la vida. A toda mi familia la de lejos y la de cerca, no alcanzarían las páginas para agradecerle a cada uno, muchas gracias por formar parte de mi vida.

Dios los bendiga mucho.

Asiel.

RESUMEN

En el departamento de Visualización y Realidad Virtual del Centro de Desarrollo Informática Industrial, de la Universidad de las Ciencias Informáticas, se han desarrollado proyectos que simulan situaciones de la vida real. En este departamento se ha trabajado con diversas herramientas como el motor gráfico Ogre3D y la plataforma de entornos 3D OpenSim. Esta última no cuenta con un módulo que permita dotar de inteligencia a los bots (*Robots*), por lo que la interactividad de estas se limita a la respuestas a interrogantes.

Ante estas limitantes surge la idea de desarrollar un módulo que permita una mayor interactividad de las entidades con los usuarios del entorno, para proveer la sensación de inteligencia por parte de estas. Este módulo proveerá nuevas capacidades a los bots, por ejemplo la locomoción, la búsqueda de camino mínimo, algo no logrado antes por la complejidad de trabajar con una herramienta no diseñada para estos fines.

Para obtener los resultados esperados se hizo un análisis de las diferentes técnicas de Inteligencia Artificial, así como técnicas de percepción y locomoción, además del concepto de agente virtual inteligente. Como fruto de esta investigación se desarrolló un módulo que cumpliera con las necesidades existentes. En este trabajo los términos agente inteligente, agente virtual y agente virtual inteligente, se refieren al mismo concepto.

PALABRAS CLAVE

Agente virtual inteligente, comportamiento de locomoción, Inteligencia Artificial, 3D

ÍNDICE

Índice de tablas

Índice de ilustraciones

INTRODUCCIÓN

En los últimos tiempos el avance del software se ha producido de manera vertiginosa. Por ello son muchas las aplicaciones que se producen para varias esferas en las que se le incluyen materias como los gráficos por computadora y la inteligencia artificial (IA). Cuba, se dedica a la realización de proyectos informáticos que incluyen la realidad virtual y simuladores.

En el departamento de Visualización y Realidad Virtual del Centro de Desarrollo Informática Industrial (CEDIN), de la Universidad de las Ciencias Informáticas (UCI), se han desarrollado proyectos que simulan situaciones de la vida real, tal es el caso del proyecto Entrenador Aduanero, el cual provee un videojuego serio para el entrenamiento de aduaneros en las habilidades de este oficio. De manera simultánea la universidad ha trabajado con OpenSim, una plataforma de código abierto que permite crear ambientes virtuales, los cuales pueden ser accedidos a través de una gran variedad de visores (clientes) o protocolos (software y web). OpenSim es configurable y puede ser extendido con el uso de módulos. La licencia de OpenSim es BSD (*Berkeley Software Distribution*)[1], permitiéndole ser de código abierto y al mismo tiempo ser usado en proyectos comerciales (Sánchez., 2010).

Dadas las facilidades que ofrece OpenSim y las herramientas de fácil manejo que posee para diseñar estructuras dentro de la misma plataforma ,se opta por utilizarlo en la creación de videojuegos u otros sistemas de simulación con el fin de disminuir el tiempo de diseño en la creación de los mismos. Permitiendo reutilizar sus funcionalidades principalmente en el tema de la interacción entre los Robots (bots) en los entornos creados. Sin embargo OpenSim no cuenta con un módulo que permita el manejo inteligente de los agentes autónomos creados, lo cual es una premisa para cualquier sistema que simule situaciones de la vida real y pretenda simular el comportamiento humano en las interacciones entre los bots.

Debido a lo anteriormente abordado surge la necesidad de incorporar un módulo de clases a OpenSim que permita el manejo de las entidades en los entornos de manera inteligente mediante el uso de las técnicas tradicionales de IA. Por lo antes expuesto se formula el siguiente **problema científico**: ¿Cómo proveer de inteligencia artificial a los bots creados en OpenSim?

Por lo tanto se define como **objeto de estudio**: IA aplicada en los entornos 3D y **objetivo general**: Desarrollar un módulo que permita el manejo de agentes inteligentes en los entornos creados desde

[1] Berkeley Software Distribution: licencia de software libre permisiva, tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

OpenSim. Lo que determina que se proponga como **campo de acción**: Manejo de agentes virtuales a partir de técnicas de IA sobre OpenSim.

Para cumplir los objetivos planteados se realizarán las siguientes tareas de investigación:

- Búsqueda y análisis de la bibliografía que permita la asimilación de los conocimientos necesarios sobre el tema a investigar.
- Elaborar el diseño teórico-metodológico de la investigación para conformar la memoria escrita del trabajo.
- Diseñar la propuesta para la solución.
- Implementar la solución antes diseñada.
- Validar las funcionalidades de la solución desarrollada mediante la implementación de un demo.
- Elaborar la memoria de la investigación.
- Desarrollar un módulo que provea de IA a los bots en los entornos creados a través OpenSim, haciendo uso de las técnicas tradicionales.

Para darle cumplimiento a las tareas de investigación se ponen de manifiesto los siguientes métodos científicos teóricos y empíricos:

Dentro de los métodos teóricos se utilizan:

- Histórico-lógico: Con el fin de profundizar en el desarrollo de las tendencias actuales en la elaboración de software de IA.
- Análisis-síntesis: Con el propósito de determinar cuáles son las experiencias más generalizadas sobre la realización de software de IA.

Dentro de los métodos empíricos se usan:

- Consulta de fuentes de información: Se realizó el estudio de varias bibliografías para obtener más información de lo que se ha realizado hasta la actualidad, en lo que respecta a la problemática abordada.

El presente trabajo de diploma estructura los capítulos de la siguiente manera:

Capítulo 1 “Fundamentación teórica”: Capítulo en el que se relacionan y abordan los principales conceptos y categorías sobre el tema que se aborda en la investigación y el estudio de estos.

Capítulo 2: “Características del Sistema”: Describe los conceptos fundamentales, que debe incluir el módulo de IA. Proporciona una guía acerca de cómo se incluyen conceptos a la estructura principal, de acuerdo a la complejidad que se desea lograr en el videojuego.

Capítulo 3: “Diseño, Implementación y Resultados”: Se plantea el resultado de la investigación, que da como fruto un diseño de clases, que permite el trabajo con la plataforma seleccionada. Se expondrán los resultados de las pruebas realizadas al sistema con el fin de verificar las funcionalidades de este.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En los últimos años, los personajes virtuales, también conocidos como avatares, están en uso cada vez más en las interfaces de usuario para mejorar la interacción hombre-máquina. Gracias a estas nuevas tecnologías se ha conseguido una forma de comunicación más natural entre el usuario y dispositivos electrónicos basada en la comunicación real entre los humanos. Estos realizan tareas de sistema, como la de asistencia o guía a través de cualquier entorno, o sencillamente presentan la información.

1.1 Técnicas de IA para la toma de decisiones

Ciertamente casi resultaría imposible tratar el tema de los mundos virtuales sin hacer mención a la IA, rama de la investigación científica dentro de la informática que se encarga de modelar la inteligencia humana a través de sistemas computacionales. La IA utiliza técnicas complejas para tratar que un ordenador simule el proceso de razonamiento humano. Pretende también que el ordenador sea capaz de modificar su programación en función de su experiencia y que él mismo aprenda.

A partir de la idea anterior se puede decir que las técnicas de toma de decisiones deciden cuáles son las mejores opciones que pueden tomar los elementos del videojuego a partir de las condiciones del entorno que los rodea. Existe una gran multitud de técnicas de toma de decisiones que se han desarrollado a lo largo de los años, como son las máquinas de estados finitos, redes bayesianas, entre otras, algunas de las cuales serán abordadas en este epígrafe.

1.1.1 Máquinas de Estados Finitos

Las máquinas de estados finitos (en inglés, *finite state machine*, FSM) 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 y, a partir de la información que leamos en el sistema y los condicionales de las transiciones, se ha de decidir si se debe cambiar hacia otro estado.

Algunas de las ventajas asociadas a la utilización de FSM 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 FSM 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 nos permite saber fácilmente cómo ir de un estado a otro y qué condiciones son necesarias para la transición.
- Los recursos necesarios para ejecutar una FSM son muy pocos.

Desventajas de este tipo de sistemas son las siguientes:

- Normalmente son sistemas muy predecibles, sobre todo en el caso de una FSM determinista.
- Sin un buen diseño puede tener problemas a la hora de su implementación, sobre todo si se trata de una máquina con muchos estados diferentes.
- Sólo sirve para aquellos casos en los que se puedan separar el comportamiento en diferentes estados independientes y donde se definan transiciones claras entre estados.

Las FSM pueden utilizarse en diferentes niveles de la programación de la IA. Lo más normal es utilizarlas para la programación de decisiones estratégicas o tácticas.

1.1.2 Lógica difusa

La lógica difusa es una alternativa que permite cuantificar la incertidumbre, brinda la opción de asignar cierta probabilidad a cada uno de los posibles valores y, por lo tanto, añadir la relatividad del observador. Según su creador, la idea original es la de imitar el funcionamiento del razonamiento humano, el cual normalmente no trabaja con valores exactos sino con valores relativos. La lógica difusa no es una técnica que permite tomar una decisión a partir de la observación del sistema. Más bien es considerada como un sistema complementario, que provoca que las decisiones que se tomen tengan una vertiente más relativista y por tanto más "humana".

Si se combina la lógica difusa en un sistema de reglas *if/then*, las reglas parecerán más realistas. Al combinarse la lógica difusa con las máquinas de estado, tendría como resultado lo que se conoce en IA como máquinas de estado difusas. Este tipo de máquinas permiten que el sistema se encuentre en más de un estado a la vez, es decir, tenemos una serie de estados activos con una cierta probabilidad y en cada momento decidimos cuál es el que nos interesa utilizar. (Martín del Brío, 2001)

1.1.3 Redes bayesianas

En el caso de que se quieran introducir más tipos de incertidumbre en el juego, utilizará un sistema basado en probabilidades. La idea es muy simple: cuando es necesario tomar una decisión se asigna una serie de probabilidades a cada una de las opciones a partir de la observación y sobre estas probabilidades es tomada la decisión.

Una de las mejores técnicas para modelar la incertidumbre de las decisiones son las redes bayesianas. Las redes bayesianas son grafos[2] dirigidos y acíclicos que representan la relación entre diferentes variables y sus relaciones de dependencia. En una red bayesiana, cada nodo representa una condición que tiene asociada una cierta probabilidad; y los arcos representan causalidad, es decir, que es necesario que sea cierto el nodo origen para poder analizar la condición del nodo destino. (Gavaldà, 2011)

El uso de las redes bayesianas es técnicamente más complejo que el resto de técnicas vistas anteriormente**,** con lo que son recomendables tan sólo para problemas de decisión muy específicos. En otros casos pueden existir mejores soluciones que éstas, más simples e igual de efectivas.

1.2 Técnicas de Percepción

La percepción obedece a los estímulos cerebrales logrados a través de los cinco sentidos, vista, olfato, tacto, auditivo, gusto, los cuales dan una realidad física del medio ambiente. La percepción se considera como uno de los temas más importantes de la robótica y los videojuegos que son creados bajo la base de la IA. Cada videojuego dirige la percepción de manera diferente, se conoce que la percepción más sofisticada puede imitar las limitaciones del mundo real. La percepción brinda toda la información del entorno virtual a sus agentes, para que su comportamiento sea lo más real posible. Según sea el grado de percepción que tenga un agente virtual será el grado de comportamiento creíble que el desarrolle.

Trazado de rayos

El algoritmo de trazado de rayos intenta combinar la idea de trazar rayos para determinar las superficies visibles con la de un proceso de sombreado, donde se tiene en cuenta los efectos globales producidos por las reflexiones, refracciones y sombras arrojadas por otros objetos de la escena. Para

[2] En matemáticas y ciencias de la computación, un grafo (del griego grafos: dibujo, imagen) o gráfica es el principal objeto de estudio de la teoría de grafos. Informalmente, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.

conseguir los efectos de reflexión y refracción se trazan rayos recursivamente desde el punto intersección que se está sombreando teniendo en cuenta las propiedades del material del objeto interceptado.

Ventajas:

- Gran realismo en comparación con otros métodos como el *Raycasting* o el *Scanline Rendering*.
- Efectos como sombreado, sombras, reflexión y refracción son un resultado natural del proceso.
- Relativamente fácil de implementar.

Desventajas:

- Intenta simular de forma muy simplificada la interacción de la luz con los objetos.
- Traza rayos de luz desde el observador hacia los objetos de la escena donde se reflejará.
- Brindan una pobre percepción del entorno.
- Afecta considerablemente la eficiencia del sistema.
- Debes encuestar varios rayos para obtener una respuesta a una pregunta cómo — ¿Dónde me puedo esconder de un enemigo?

Técnicas de ordenamiento espacial

Existen diferentes técnicas para lograr ordenar el espacio 3D. El ordenamiento más sencillo que puede pensarse es una simple subdivisión del espacio en cajas idénticas. Esta sencilla técnica se conoce como “divide y vencerás” y consiste en dividir cada coordenada en un número adecuado de partes. El principal problema es la gran cantidad de partes vacías.
Una variante del método, que se utiliza en los juegos en primera persona con recorridos por salas separadas, consiste en la obvia agrupación de objetos los cuales se denominan métodos de celdas y portales. Hay muchas variantes para lograr el ordenamiento espacial en un entorno 3D, las más conocidas son:

- Diagrama Voronoi.
- Jerarquías de Volúmenes Fronteras.
- Árboles Octales.
- Árboles BSP.

Datos manualmente generados

Los datos son generados manualmente con el objetivo de compensar la poca percepción que presentan los agentes inteligentes, en este caso son los desarrolladores los encargados de añadir los datos topológicos de manera manual en un mundo virtual. Un ejemplo simple constituye el caso de

realizar una emboscada, se entrarán manualmente datos topológicos como: la posición de escondites, la activación de zonas, etc. Sin embargo estos datos no constituyen una solución aceptable para la percepción limitada de los agentes inteligentes, porque:

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

Detección de Colisiones

El objetivo de la detección de colisiones es el de reportar de manera automática una intersección entre dos objetos cualesquiera. En el caso particular en que los objetos están en reposo entonces se habla del problema de la detección de intersecciones. El problema de la detección de colisiones aparece en múltiples aplicaciones, tales como el diseño asistido por computadora, la robótica, la automática, la computación gráfica, la animación y la realidad virtual. La importancia de la detección de colisiones es que permite el diseño basado en la simulación, el análisis en ingeniería, la planificación del movimiento en robótica, la animación basada en modelos físicos, la navegación en modelos virtuales, entre otros.

1.3 Locomoción en agentes inteligentes

Los agentes inteligentes tienen movilidad y autonomía, porque deciden a dónde moverse y qué algoritmo ejecutar. Para ello intervienen técnicas de representación del entorno conjuntamente con algoritmos de búsqueda de caminos.

1.3.1 Representación del entorno

Para que una entidad inteligente interactúe con los obstáculos dinámicos o estáticos del mundo en el que se encuentra es necesaria la discretización del mismo. Esto consiste en la división del entorno virtual en nodos de grafo de navegación. Para lograrlo existen varias técnicas:

Grafo de camino

Cada vértice en el grafo representa un estado libre y cada arista representa un camino libre de colisiones entre dos estados. Estas estructuras tienen el inconveniente de que el movimiento de los agentes se ve restringido al camino generado. (Campos, 2010)

Grafo finamente granulado

Es una asociación de nodos y aristas que permite representar relaciones binarias. Tienen la ventaja de que mejoran los puntos ciegos que generan los grafos de visibilidad y mejoran el suavizado de los caminos. Al hacer uso de este tipo de grafo se presenta el inconveniente de que el número de nodos aumenta considerablemente y por consiguiente la búsqueda de caminos se hace más lenta. (Campos, 2010)

Rejilla regular

Se utiliza para dividir el mundo de polígonos iguales, donde cada una de ellas comparte un borde a lo sumo con la otra. El centro de la rejilla representa un punto en el camino y las rejas vecinas forman las conexiones con el punto. Existen 3 tipos de rejillas: triangulares, cuadradas o de hexágonos regulares. (Smed, 2006)

Las rejillas generalmente soportan consultas de acceso aleatorio, ya que cada celda debe ser accesible en cualquier instante de tiempo. No tiene en cuenta o no presta atención la geometría actual del mundo, esto provoca que puedan quedar desconectadas algunas partes del mundo si la granularidad de la rejilla regular no es lo suficientemente fina. Además, para el almacenamiento de la información de la rejilla regular se necesita memoria, aunque esto puede eliminarse haciendo uso de tablas de consultas jerárquicas. (Campos, 2010)

Malla de Navegación

Es una partición convexa en la geometría del mundo del juego. Es decir, es una serie de polígonos donde ninguno solapa al otro y están unidos en solo dos puntos y un borde. Cada polígono adyacente representa que un punto tiene conexiones con ellos, el centro de cada polígono es un punto en el camino y el centro de cada borde que se comunica también representa uno. (Smed, 2006)

Entre las ventajas que nos brinda la malla de navegación está el conocimiento completo sobre las zonas navegables lo que permite que los agentes sean capaces de evitar obstáculos o seleccionar una ruta alternativa mostrando así un comportamiento autónomo. (Campos, 2010)

1.3.2 Búsqueda de caminos

En los mundos virtuales los agentes inteligentes sin duda necesitan moverse por el entorno recreado, por lo que es imprescindible dotar al sistema de una manera de hacerlo. Algunos algoritmos básicos para implementar la búsqueda de caminos dentro de un grafo podrían ser los siguientes aunque existen otros de mayor complejidad.

Dijkstra: Este algoritmo fue nombrado por el matemático Edsger Dijkstra. Fue creado originalmente para resolver un problema en teoría de grafos matemáticos. Encuentra el camino mínimo desde un vértice hasta los demás, pero es muy costoso porque calcula todos los caminos posibles a partir de un nodo. (Gavaldà, 2011)

Algoritmo A*: Presentado en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael, es uno de los más empleados en IA, permite el trabajo con grafos dinámicos. Este es muy eficaz ya que expande la menor cantidad de nodos, es fácil de implementar y tiene muchas posibilidades de optimización, además puede ser empleado para planes de movimiento complejos. (Gavaldà, 2011)

Algoritmo de Floyd: El creador de este algoritmo es Robert W. Floyd fue un prominente científico estadounidense en Computación. Floyd recibió el Premio Turing de la ACM en 1978 "por tener una clara influencia en las metodologías para la creación de software eficiente y confiable, y por haber contribuido a la fundación de las subáreas, teoría del reconocimiento de frases, semántica de los lenguajes de programación, verificación automatizada de programas, síntesis automatizada de programas y análisis de algoritmos."[3]

Una primera solución para encontrar el camino mínimo para todo par de vértices de un grafo, consiste en usar repetidamente el Algoritmo de Dijkstra variando el vértice inicial. Ahora bien, también se dispone del llamado Algoritmo de Floyd, que proporciona una solución más compacta y elegante, pensada especialmente para esta situación. Un camino óptimo entre dos vértices de un grafo, cumple con el Principio de Optimalidad de Bellman: “dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima”. Este principio de optimalidad es la base del Algoritmo de Floyd. (Algoritmos sobre Grafos I.)

1.3.3 Técnicas para definir el movimiento de los agentes inteligentes

Para lograr el movimiento de agentes en un entorno, todo algoritmo por sencillo que sea, tiene como entrada la posición actual del agente y hacia donde desea trasladarse, teniendo como salida una manera de representar el movimiento que se desea hacer. Los algoritmos, a pesar de que tiene el mismo objetivo, se separan en dos grupos fundamentales debido a su complejidad:

Los que realizan movimiento cinemático, este tipo de movimiento no tiene en cuenta la manera en que los caracteres aceleran y reducen la velocidad, solamente se preocupan por la dirección y llegar al blanco. Mientras que en el segundo grupo están los que logran un comportamiento dinámico, necesita

[3] Tomado de http://es.wikipedia.org/wiki/Robert_W._Floyd

saber además de la posición la velocidad, y tiene como salida una fuerza o aceleración que le permiten cambiar la velocidad de la entidad (Millington, 2006).

Algoritmos para comportamientos grupales de locomoción

Los agentes inteligentes se pueden trasladar de dos maneras: de forma individual o en manada, sea cual sea la forma en que lo hagan es necesario que tengan una dirección determinada. Estos movimientos de dirección son a los que llamamos comportamientos de locomoción.

Las diferencias entre el movimiento de los agentes de forma individual y en grupos, está dada principalmente por la información que manejan. Craig Reynolds (1987) propone en "su algoritmo más básico para el comportamiento de manadas la visión de un agente como una unidad individual, que posee información sobre los vecinos y de acuerdo a ella se mueve".

Tanto una unidad individual como un colectivo pueden realizar los mismos movimientos de dirección incorporándole en la implementación de los miembros del grupo tres parámetros:

1. **Separación:** Para evitar colisiones entre los elementos o que estos se acerquen demasiado.
2. **Cohesión:** Se utiliza una fuerza que mantiene unido al grupo, la misma está dirigida al centro de masa de sus vecinos.
3. **Alineamiento:** Su función es que los individuos vayan todos en la misma dirección.

Todos estos parámetros se implementan de forma sencilla apoyándose en el trabajo con vectores.

1.4 Agentes Virtuales Inteligentes

Un Agente Virtual Inteligente (AVI) es un programa de software con las siguientes características:

- Su hábitat natural (entorno de ejecución) es un mundo gráfico simulado 3D.
- "Posee una representación gráfica (cuerpo modelado geométricamente) dentro del mundo que habita, y es capaz de percibir, adaptarse, y reaccionar a su entorno; exhibiendo gráficamente un comportamiento para los cuales fueron diseñados". (Luis Daniel Hernández López, 2008)
- "Un agente es un sistema situado en alguna parte de un entorno que percibe dicho entorno y actúa en él en beneficio de su propia agenda, el efecto de su actuación se nota en el entorno". (Luis Daniel Hernández López, 2008)
- "Los agentes son entidades que colaboran con sus usuarios para mejorar el cumplimiento de sus tareas de usuario" (Luis Daniel Hernández López, 2008).

- Es común confundir el concepto de AVI con el de Avatar. Un Avatar es un Agente Virtual cuyo comportamiento es explícitamente controlado por un animador externo, mediante comandos de control; por lo tanto, no es inteligente.

1.5 Biblioteca OpenMetaverse

Es creada por la empresa Linden Lab. Esta Librería es realmente útil para conectarnos a cualquier metaverso basado en Second Life y OpenSim, sin embargo, es bastante difícil de utilizar y existe muy poca documentación en internet sobre cómo utilizarla.

La librería OpenMetaverse es compatible con Microsoft.NET Framework 2.0 y versiones posteriores, así como Mono 1.9 y posterior. Este archivo es una Librería de enlace Dinámico. Esta librería puede ser cargada y ejecutada en cualquier proceso en ejecución.

La librería consta de varios **.NET**:

- **OpenMetaverse.dll**: El núcleo de la librería OpenMetaverse, que puede ser usado para crear aplicaciones clientes y servidor.
- **OpenMetaverseTypes.dll**: Funciones específicas 3D y librerías matemáticas.
- **OpenMetaverse.StructuredData.dll**: Un conjunto de librerías para dar un completo soporte a *Linden Lab Structured Data* (LLSD) y (*JavaScript Object Notation*) JSON. Un envoltorio .NET para codificar y descodificar los datos JPEG2000.

1.6 Mundos Virtuales

El mundo virtual no es más que una simulación por ordenador de un espacio, que habitualmente es una recreación en 3D de accidentes geográficos, ciudades y simulaciones computacionales de la realidad. Dentro de un mundo virtual, nos desplazamos e interactuamos con el resto de usuarios representados por una unidad denominada avatar. Los mundos virtuales en línea, o la llamada Internet 3D, integran una serie de recursos que permiten servicios al usuario como la comunicación con otros en la red. El éxito en el uso de la Internet 3D consiste en la virtualización del propio usuario, algo no visto anteriormente en el cómputo y las telecomunicaciones hasta los presentes días.

El avatar, es literalmente una representación del usuario en el mundo virtual, pero no solo tiene las características corporales humanas convencionales sino incluso acciones como volar sin la asistencia de algún aparato, también puedes personalizar el avatar tanto como desees, o incluso falsificarlo para

que no se parezca en lo absoluto al original mediante un grupo de herramientas de personificación que permite agregar o quitar cabello, cambiar el tipo de cara, ojos y nariz, amoldar el cuerpo del avatar como uno desee y vestirlo con las telas y estilos que el usuario prefiera. Los avatares viven en la Internet 3D tienen características sociales, pueden tener propiedades que se construyen en diversas islas. (Bell, 2008)

Los sistemas centrales de la sociedad paralela permiten la adquisición de espacios en el mundo virtual donde es posible construir edificios a los cuales se les puede agregar, por ejemplo, una sala de proyección para distribuir video en demanda, una sala de *podcasting*[4] para audio, diversos niveles o plantas especializadas en el tipo de información que el usuario quiere proporcionar e incluso, vender a los demás. (Bartle, 2003)

1.7 Plataformas para el desarrollo de Mundos Virtuales

En la actualidad existen varias plataformas que permiten el desarrollo de mundos virtuales, las cuales varían de acuerdo a la edad de los usuarios, hasta la posibilidad de poder realizar cambios en el funcionamiento de las mismas. En el siguiente epígrafe abordaremos dos de las principales plataformas para estos fines.

1.7.1 Second Life

Second Life (SL) desarrollado por Linden Lab[5], cuenta con un motor físico: Havok[6], el audio es controlado por FMOD, es accesible gratuitamente desde Internet, fue lanzado el 23 de junio de 2003. Se puede acceder a SL mediante programas llamados *Viewers* (visores). SL puede ser utilizado en plataformas Microsoft Windows, Windows XP SP2 y SP3, Windows Vista, Windows 7, Mac OS X (10.4.11 o higher), Linux i686. A los residentes como se les llama a los usuarios, se les permite mediante esta herramienta interactuar con otros usuarios en un entorno virtual o mundo virtual, mediante un avatar. Proporciona funcionalidades como son: participar en actividades tanto

[4] El podcasting consiste en la distribución de archivos multimedia (normalmente audio o vídeo, que puede incluir texto como subtítulos y notas) mediante un sistema de redifusión (RSS) que permita suscribirse y usar un programa que lo descarga para que el usuario lo escuche en el momento que quiera.

[5] Empresa concebida por Philip Rosedale en 1999. Desarrolló un programa que permitía a sus usuarios la inmersión en un mundo virtual. En sus inicios la firma intentó desarrollar y comercializar hardware destinado a este fin.

[6] Havok Game Dynamics SDK, es un motor físico (simulación dinámica) utilizada en videojuegos y recrea las interacciones entre objetos y personajes del juego. Por lo que detecta colisiones, gravedad, masa y velocidad en tiempo real llegando a recrear ambientes mucho más realistas y naturales.

individuales como en grupo, establecer relaciones sociales, comercializar propiedad virtual y servicios entre ellos. Cuenta con una moneda local el Linden Dólar (L), así el usuario puede vivir una segunda vida pero en un mundo virtual.

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. (Venereo, 2011)

La forma de crear objetos en SL, como son armas, carros aparatos de música entre otros, es a través de diseño 3D y la programación de estos en 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 y en los países subdesarrollados no todas las personas tienen acceso a la tecnología necesaria para poder acceder a SL.

Para acceder a SL 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. Los controles pueden ser mediante el teclado, *mouse*, *gamepad*, entre otros. (Venereo, 2011) Por las limitantes de conexión este software no está disponible en nuestro país o el acceso es demasiado lento.

1.7.2 OpenSim

OpenSim es una plataforma capaz de manipular de forma interactiva un Mundo Virtual, soporta varias regiones conectadas a un solo Grid[7] centralizado, sus creadores se basaron en SL. Además, utiliza el

[7]Grid (cuadricula en español), es el nivel que organiza las regiones y sus posiciones en el mundo, y maneja las cosas que necesita para existir la región, como el inventario de usuarios. Puede pensarlo similar al mapa del mundial.

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 varios 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.

Para la programación de script[8] 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.

La base de datos OpenSim es compatible con los siguientes motores de bases de datos:

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

[8] En informática un guión, archivo de órdenes o archivo de procesamiento por lotes, vulgarmente referidos con el barbarismo script, es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.

- *Robust server* -servidor que agrupa un conjunto de servicios que en otras versiones ejercían como servidores, como son: *Grid server, Asset server, Invetory server, User server*.
- *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.

En la facultad 5 se tiene instalado dicho entorno, la incorporación de IA al entorno, no es algo trivial, más bien se hace complicado en una herramienta no definida para esos fines.

1.8 Arquitectura cognitiva SOAR

SOAR (*State Operator And Result*) es una arquitectura software desarrollada por varias universidades estadounidenses con el objetivo de implementar en máquinas para distintas aplicaciones un algoritmo de comportamiento y pensamiento que se acerque lo más posible al humano.

1.8.1 Visión general de la estructura SOAR

Las características principales en el desarrollo de la arquitectura SOAR son:

Conocimiento: SOAR representa uniformemente el conocimiento a corto plazo como una red de símbolos activos. El conocimiento a largo plazo es un conjunto de reglas condición-acción. Las condiciones de cada regla forman un patrón que se comparará con la red de símbolos activos. Cuando la condición de la regla es equivalente a dicha red, la regla se ejecuta mediante la realización de sus acciones. Estas acciones pueden implicar la inclusión (o supresión) de símbolos en la estructura de conocimiento a corto plazo. Cuando la condición de una regla se cumple, dicha regla se ejecuta llevando a cabo las tareas que tiene programadas. Estas acciones se darán como resultado la creación (o eliminación) de símbolos en la estructura del conocimiento a corto plazo.

Acción orientada al objetivo: para el control complejo, SOAR incluye una jerarquía orientada al objetivo, permitiendo la descomposición sucesiva de problemas en subproblemas. Así mismo, incluye mecanismos para la creación automática de nuevos objetivos como respuesta al conocimiento a largo plazo y a la situación actual.

Reacción: A diferencia de otros lenguajes de programación, SOAR no fuerza un flujo de control en serie. Las acciones tienen lugar en cualquier momento en función del cumplimiento de las condiciones

en las re-glas. Múltiples reglas pueden dispararse (*fire*) en paralelo, pero SOAR provee mecanismos de preferencia y la creación de subobjetivos para hacer frente a los posibles conflictos, si se dan.

Aprendizaje: SOAR incluye un mecanismo automático de aprendizaje inspirado en el concepto psicológico de troceado (*chunking*).Compila secuencias de acciones en nuevas unidades de conocimiento (trozos) que pueden acortar los pasos de razonamiento cuando el sistema se encuentre ante situaciones similares en un futuro. Los nuevos trozos se unen de forma uniforme en un conjunto de reglas del sistema a largo plazo. Gracias a este mecanismo, puede aprender de forma incremental aprendiendo nuevos detalles sobre el mundo, así como disponer de una representación más eficiente de su conocimiento a largo plazo inicial.

Capacidades: Desde su desarrollo, la arquitectura SOAR ha sido utilizada para investigar una gran variedad de sistemas de desarrollo y aplicaciones comerciales. (Herranz, 2005)

1.8.2 Principios básicos

SOAR basa su diseño en una serie de principios que le guían en su intento de aproximarse al comportamiento racional:

- El número de mecanismos arquitecturales debe ser mínimo. En SOAR, hay un marco simple para todas las tareas y subtareas (espacio del problema), una representación simple del conocimiento permanente (producciones), una representación simple del conocimiento temporal (objetos con atributos y valores), un mecanismo simple de generar objetivos (mecanismo automático de generar subobjetivos) y un simple mecanismo de aprendizaje (*chunking*).
- Todas las decisiones son hechas a través de la combinación de conocimiento relevante en tiempo de ejecución. En SOAR, cada decisión está basa-da en la interpretación real de los datos sensoriales, el contenido de la memoria de trabajo creada por el espacio del problema y algún otro conocimiento obtenido de la memoria permanente. (Herranz, 2005)

1.8.3 Concepto de arquitectura

En general, el concepto de arquitectura es útil porque permite factorizar algunos aspectos comunes del ancho rango de comportamientos que caracterizan el contenido. Una arquitectura particular, esto es, un particular conjunto mezclado de mecanismos y estructuras, permanece como una teoría de qué es común entre muchos de los comportamientos al nivel superior. Usando esta idea, se puede definir una arquitectura cognitiva como una teoría de mezcla de mecanismos y estructuras que subrayan el fenómeno cognitivo humano.

Analizando lo que tienen en común los comportamientos cognitivos y el fenómeno explicado en las microteorías, parece que producir una teoría cognitiva unificada es un paso significante. Como un ejemplo de arquitectura cognitiva, SOAR es una teoría de lo que tienen en común la gran variedad de comportamientos de la inteligencia.

Para construir un modelo de comportamiento en SOAR se debe entender primero que aspectos del comportamiento soportará la arquitectura directamente y poner el trabajo preliminar para enlazar la arquitectura con el contenido. (Herranz, 2005)

Semejanza entre los comportamientos cognitivos

Para entender cómo trabajan algunas estructuras computacionales, se necesita usar un modelo con algún comportamiento, la arquitectura por sí sola no hace nada. Una arquitectura cognitiva debe ayudar a producir comportamiento cognitivo. Leer, requiere cierta habilidad cognitiva, así mismo: resolver ecuaciones, hacer la cena, conducir un coche, contar una broma o jugar al baloncesto. De hecho, la mayoría de los comportamientos diarios parecen requerir algún grado de pensamiento que media entre las percepciones y las acciones. Porque cada arquitectura es una teoría sobre qué es común al contenido que procesa; SOAR es una teoría acerca de lo que tienen los comportamientos en común. En particular, la teoría sugiere como principio básico tener al menos las siguientes características:

- Ser orientada al objetivo: a pesar de cómo se siente a veces, no se tropieza en la vida, actuando en modos que no están relacionados con nuestros deseos e intenciones. Si se quiere hacer la cena, se va al lugar apropiado (cocina, por ejemplo), se mezclan los ingredientes, se trocean, se cocinan y sazonan hasta que se produce el resultado deseado. Para ello, sería necesario aprender nuevas acciones: (poner el fuego lento en lugar de una fogata), o aprender el orden correcto de los pasos a realizar (añadir líquidos a sólidos, no viceversa), pero se aprende simplemente por el hecho de actuar varias veces de forma aleatoria.
- Reflejar un rico, complejo y detallado ambiente/medio: Aunque los modos es los que percibimos y actuamos en el mundo son limitados, el mundo que percibimos y en el que actuamos no es simple. Hay un elevadísimo número de objetos, cualidades de objetos, acciones y demás, cada una de las cuales debe ser la llave para entender cómo conseguir nuestros objetivos.
- Requiere una gran cantidad de conocimiento: Tiene que intentar describir todas las cosas que conoce para saber cómo resolver ecuaciones. Algunas de ellas son obvias: poner la variable a resolver en uno de los lados del igual, mover términos constantes mediante suma o resta y coeficientes mediante multiplicación o división. Pero también necesita saber cómo hacer multiplicaciones y restas, nociones básicas de números, cómo leer y escribir números y letras,

cómo coger el lápiz y usar una goma, qué hacer si el lápiz se rompe o si la habitación se queda sin luz.

- Requiere el uso de símbolos y abstracciones: De algún modo parte del conocimiento que posee puede ser obtenido por algo más que lo que es en sí la percepción en todo su detalle. Esto se llamará símbolo (o conjunto de símbolos). Porque el hombre representa el mundo internamente usando símbolos, el hombre crea abstracciones.
- Requiere aprendizaje del ambiente y la experiencia: La gente no nace sabiendo cómo contar bromas, resolver ecuaciones, jugar baloncesto o hacer la cena. Aun así, la gente llega a ser muy habilidosa (e incluso experta) en una de estas o muchas más actividades y cientos de otras. Efectivamente, quizá la cosa más admirable sobre la gente es cómo aprenden a hacer muchas cosas con poco, pareciendo que nacen sabiendo cómo hacerlo.
- Hay muchas otras propiedades que marcan las capacidades cognitivas (por ejemplo, la cualidad de ser consciente de uno mismo), y hay otros modos de interpretar los mismos comportamientos previamente mencionados.

Habiendo identificado las propiedades comunes del comportamiento cognitivo que SOAR debe soportar, se deberían motivar las estructuras y mecanismos específicos uniéndolos con estas propiedades. Requiere contenido para producir comportamiento. Por ello, hay que tener claro:

COMPORTAMIENTO = ARQUITECTURA CONTENIDO

Es claro que si se quiere ver cómo la arquitectura contribuye al comportamiento, entonces necesita explorar la arquitectura en términos de un contenido en particular. Por ejemplo, describiendo cómo SOAR soporta la importante característica de ser orientado al objetivo no producirá por sí mismo comportamiento. Se necesita ser orientado al objetivo sobre algo. (Herranz, 2005)

Comportamiento como movimiento a través del espacio de problema

Cada vez, se dan unas circunstancias diferentes para llevar a cabo un mismo objetivo. Si se intentan dar todas las circunstancias las cuales darían lugar a conseguir el objetivo, nos superarían, sería prácticamente inviable. A pesar de que haya muchas posibilidades, sólo se podrá elegir una en el momento en el que se está en el juego real. Por ello, el sujeto deberá ser capaz de actuar basándose en las elecciones que tienen sentido en ese momento, pero a su vez, teniendo en cuenta las consecuencias que puede haber en el futuro.

La idea de los espacios del problema data a los primeros días de las investigaciones en IA y modelos cognitivos usando computadores, y es una de las construcciones computacionales principales en la teoría cognitiva de SOAR. Podría ser duro ver el problema de los espacios en el cerebro cuando se focaliza en una masa de neuronas. (Herranz, 2005)

1.8.4 Estructura SOAR

El diseño de SOAR está basado en la hipótesis de que todo el comportamiento enfocado a conseguir un objetivo puede ser moldeado como una selección y aplicación de operadores (*operators*) hacia un estado.

Cuando SOAR está corriendo, continuamente trata de aplicar el operador actual y seleccionar el próximo, ya que un estado, sólo puede tener un operador activo a la vez, hasta que el *goal* (objetivo) es alcanzado.

SOAR tiene memorias separadas, con diferentes representaciones, para describir la situación real y su conocimiento a largo plazo. En SOAR, la situación actual se compone de los datos recibidos por los sensores, resultados de deducciones intermedias, objetivos activos y los operadores activos que se encuentran en la memoria de trabajo.

La memoria de trabajo está organizada en objetos y los objetos son descritos en términos de atributos; el valor de los atributos puede corresponder a sub-objetivos, por tanto, la descripción del estado tiene una organización jerárquica.

El conocimiento a largo plazo, que especifica cómo responder ante distintas situaciones en la memoria de trabajo, se puede decir que es el programa que hay que programar para SOAR. Así, una arquitectura SOAR no puede solucionar problemas si no dispone del conocimiento a largo plazo. No hay que confundir, por tanto, una arquitectura SOAR con un programa SOAR.

- Arquitectura SOAR hace referencia al sistema estructural de SOAR, tipos de memoria, operadores, atributos, impasses, etc., común para cualquier usuario.
- Programa SOAR hace referencia al conocimiento que se le ha de añadir a dicha estructura. Un programa contiene el conocimiento que ha de ser usado para solucionar una tarea específica (o un conjunto de tareas), incluyendo información acerca de cómo seleccionar y aplicar operadores para transformar los estados del problema y cómo reconocer cuando el objetivo ha sido alcanzado. (Herranz, 2005)

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

En el presente capítulo se muestra la propuesta de solución para el problema planteado y se describe todo el funcionamiento para los dos casos de estudio que se proponen, se realiza un resumen de la fase Análisis y Diseño para la implementación de los casos de estudio. Se definirán los requisitos, tanto funcionales como no funcionales que deberán cumplir los casos de estudio. Se presentarán los modelos conceptuales, dígase diagramas, descripción de las clases y realización de casos de uso.

2.1 Propuesta de solución

Para darle solución al problema planteado se propone la creación de un módulo inteligente para OpenSim, que permita la creación de varios bots en el entorno virtual. Este módulo contará con comportamientos de locomoción los cuales se gestionaran a través de una FSM, y un grafo de camino, el cual utilizará como algoritmo de búsqueda de camino mínimo el algoritmo Floyd. Además permitirá interactuar mediante la voz para proporcionar respuesta a un conjunto de consultas. Este módulo permitirá una mayor interactividad entre los bots y los usuarios del entorno ya que proveerá nuevas funcionalidades con las cuales no se contaba hasta el momento y propicia una mayor sensación de inteligencia por parte de los bots. Para el desarrollo de este bot se han utilizado programación en lenguaje C# y tecnologías y aplicaciones como bases de datos mySQL, librerías específicas como OpenMetaverse para la conexión a mundos virtuales y síntesis de voz mediante Loquendo TTS.

2.2 Metodología de desarrollo del software

El uso de las metodologías permite definir una guía para desarrollar un proceso de software con un nivel de calidad elevado *Rational Unified Process* (RUP) es una metodología tradicional que responde a dicho propósito.

RUP se centra en controlar de forma rigurosa el proceso de desarrollo de software. Define dicho proceso como iterativo e incremental, las iteraciones se basan en los pasos de los flujos de trabajo y los incrementos, en versiones incrementales que se acercan al producto terminado. Está dirigido por casos de uso ya que éstos sirven para especificar los requisitos del sistema, guían el diseño, la implementación y las pruebas. Está centrado en la arquitectura para tener una clara perspectiva del sistema y de sus partes más relevantes.

La metodología escogida, RUP agrupa las actividades en grupos lógicos, para ello delimita nueve flujos de trabajo: Modelamiento del negocio, Requerimientos, Análisis y Diseño, Implementación, Pruebas, Instalación, Administración de proyecto, Gestión de Configuración y Cambios y Ambiente; a los seis primeros se les denomina flujos de ingeniería y los tres últimos son conocidos como de apoyo. Cada uno de estos flujos pasa, en mayor o menor medida, por cuatro fases: Inicio, Elaboración, Construcción y Transición. (S.Pressman, 2002) Es robusta y dirigida por casos de uso, lo cual implica que desde la captura de requisitos a través del usuario, los mismos sirvan de hilos conductores en todo el proceso de desarrollo. Centrada en la arquitectura, la cual describe al utilizar diferentes vistas; además de ser iterativa e incremental, por lo que divide el desarrollo en pequeñas partes consideradas cada una como iteraciones que al concluir aportan un incremento al producto final. Posee una gran organización en todas sus etapas además de ser muy potente.

2.3 Herramienta de modelado

Visual Paradigm es una herramienta CASE que usa UML como lenguaje de modelado, permite la generación automática de reportes en formato pdf y html, el reconocimiento de artefactos de ingeniería a partir de reconocimiento de textos formales o informales, implementa una actualización automática del modelo de diseño y código permitiendo mantener la documentación de ambos modelos actualizadas con los cambios que ocurran en ambos sentidos, optimiza la descripción textual de elementos de código a partir de la descripción visual. Es capaz de importar y exportar elementos de otras herramientas CASE como Rational Rose y presenta una interfaz de uso intuitiva y con increíbles facilidades a la hora de modelar los diagramas que soportan la ingeniería de requerimientos. Es una herramienta de software libre, actualmente es la herramienta más empleada tanto en el proceso docente, como productivo. Por tales facilidades se escoge esta herramienta, como herramienta de modelado. (EcuRed, 2013)

2.4 Lenguaje de modelado

Lenguaje Unificado de Modelado (LUM o 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; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados. Permitiendo realizar clases con descripciones específicas de gran

ayuda para el entendimiento del proyecto, y base para el momento de la implementación de las clases. (Larman, 2da Edición)

2.5 Herramienta de voz

El SAPI de Windows, que es el encargado de controlar las propiedades de voz, que convierte el texto que escribimos en una voz, y de esta manera el bot habla.

2.6 Lenguaje de programación

Se escoge este lenguaje por ser el utilizado en la librería OpenMetaverse. C# conocido como C sharp es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común. Fue creado por el danés Anders Hejlsberg que diseño también los lenguajes Turbo Pascal y Delphi. Su sintaxis básica deriva de C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes. (Wikipedia,2012)

2.7 Herramientas de desarrollo

- Microsoft Visual Studio Team System 2008 es una plataforma para herramientas del ciclo de vida del desarrollo de software extensible, integrado y productivo que ayuda a los equipos de desarrollo de software mediante la mejora de las comunicaciones y la colaboración durante todo el proceso de desarrollo. Soporta lenguaje C#, es la herramienta que se utiliza actualmente en el proyecto Entorno de desarrollo integrado (IDE). (Wikipedia, 2013)

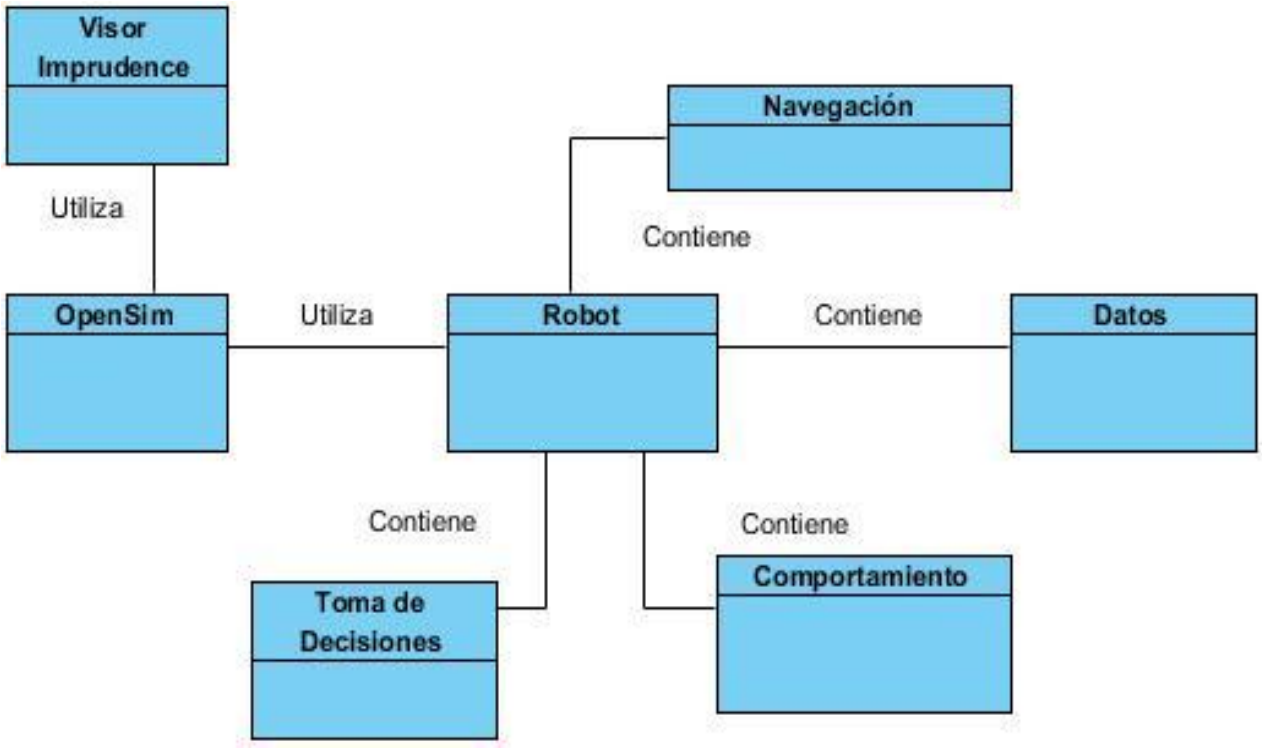
- XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo de X (para cualquiera de los diferentes sistemas operativos), Apache, MySQL, PHP, Perl. El programa está liberado bajo la licencia GNU[9] y actúa como un servidor web libre, fácil de usar y capaz de interpretar páginas dinámicas. Actualmente XAMPP está disponible para Microsoft Windows, GNU/Linux, Solaris y

[9] Orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

MacOS X. Por tales ventajas se escoge a esta herramienta para el manejo de la base de datos. (Wikipedia, 2013)

2.8 Modelo de dominio

El modelo de Dominio servirá como punto de partida para los posibles usuarios, ofreciendo mayor comprensión de todos los términos utilizados en la elaboración del módulo. Se escogió esta representación porque la misma es un subconjunto del modelo del negocio y porque en el caso que se enfrenta las fronteras del mismo no son claras. Es una representación visual de las clases abstractas del mundo real. Su función es ayudar a comprender el problema que se plantea.



2.8.1 Glosario de términos del modelo de dominio

- Visor Imprudence: Se denomina al visor para el mundo OpenSim.
- OpenSim: Se denomina a la plataforma utilizada para generar el mundo 3D.
- Robot: Se denomina a la clase que representa a la entidad donde se va a incluir la IA.
- Navegación: Se denomina al componente de navegación, encargado de realizar los distintos tipos de búsqueda de caminos, que pueda necesitar el agente virtual. Es quien va a calcular

los vectores necesarios para que el bot se mueva por el mundo de acuerdo a la decisión tomada por el sistema de toma de decisiones. Para que el agente controlado por la computadora tenga una buena navegación debe conocer su ubicación así como la posición de las otras entidades.

- Datos: Se denomina al módulo de bases de datos.
- Toma de Decisiones: Se denomina al módulo toma de decisiones, dependiendo de la complejidad del sistema, puede estar relacionada con una a más clases. Este componente es esencial para que el sistema que se está desarrollando parezca inteligente, por lo que debe prestársele especial atención.
- Comportamiento: Se denomina al módulo que va a incluir los distintos comportamientos que puede poseer el sistema.

2.9 Especificación de los requisitos del software

Los requisitos rigen el desarrollo de los servicios necesarios a construir para dar respuesta al objetivo de esta investigación. Aquí se definen cuáles son las funcionalidades del sistema. Estos pueden dividirse en requisitos funcionales y requisitos no funcionales.

2.9.1 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. Estos no alteran la funcionalidad del producto, lo que quiere decir que se mantienen invariables sin importar con qué propiedades o cualidades se relacionen. (Ivar Jacobson, 2000)

RF 1. Inicializar un bot inteligente para integrarlo en los entornos de OpenSim.

RF 1.1 Conectar con la base de datos.

RF 2 Realizar comportamiento simple.

RF 2.1 Identificar los objetos que lo rodean.

RF2.2 Interactuar con otro bot o avatar.

RF 2.3 Volar dada una orden.

RF 2.4 Realizar la acción sentarse dada una orden.

RF 2.5 Permitir la comunicación mediante chat.

RF 3 Realizar comportamientos de locomoción especificados por el programador.

RF3.1 Seguir un camino dado.

RF 3.2 Buscar camino de un punto a otro.

RF 3.3 Seguir a un avatar dada una orden a una distancia predefinida.

RF 4 Mostrar información de la base de datos.

RF 5 Responder mediante voz.

2.9.2 Requisitos no funcionales de interfaz e implementación

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto. (Ivar Jacobson, 2000)

Usabilidad

RNF 1.1 El sistema debe permitir el acceso a los servicios de manera rápida y segura.

Fiabilidad

RNF 2.1 Las solicitudes de eventos e invocación de los distintos tipos de operaciones van a ser siempre dependientes de los mecanismos creados y de la lógica creada del sistema.

RNF 2.2 Debe existir una comunicación eficiente y fiable, para garantizar que no existan pérdidas de información.

RNF 2.3 Deben persistir las conexiones entre los módulos que componen el sistema.

RNF 2.4 Debe proveerse, a los servicios, de mecanismos ante fallos que permitan recuperarse ante posibles errores.

Soporte

RNF 4.1 El desarrollo del sistema debe brindar alta interoperabilidad.

RNF 4.2 Se debe implementar un sistema flexible y que permita agregar o modificar funcionalidades con bastante facilidad de forma eficiente sin impactar en los clientes.

Hardware

RNF 5.1 Instalación de la herramienta. Para la instalación del servidor de OpenSim la computadora deberá cumplir con los siguientes requisitos:

- Memoria RAM de 1 Gb o Superior.
- Velocidad de Microprocesador a 2.0 GHz o superior.
- Recursos de Video de 256 Mb o superior.

2.10 Modelo de Caso de Uso del Sistema

En esta sección se definen los actores del sistema para la aplicación a desarrollar. A partir de la anterior captura de requisitos, se definieron los casos de usos del sistema, los cuales permiten dividir una aplicación en varias partes para su mejor elaboración y comprensión.

2.10.1 Actores del sistema

Actor	Descripción
Sistema	Actor encargado de ejecutar el caso de uso inicializar bot en el entorno
Usuario	Actor que se encarga de interactuar con el sistema

Tabla 1. Actor del Sistema

2.10.2 Diagrama de Caso de Uso del Sistema

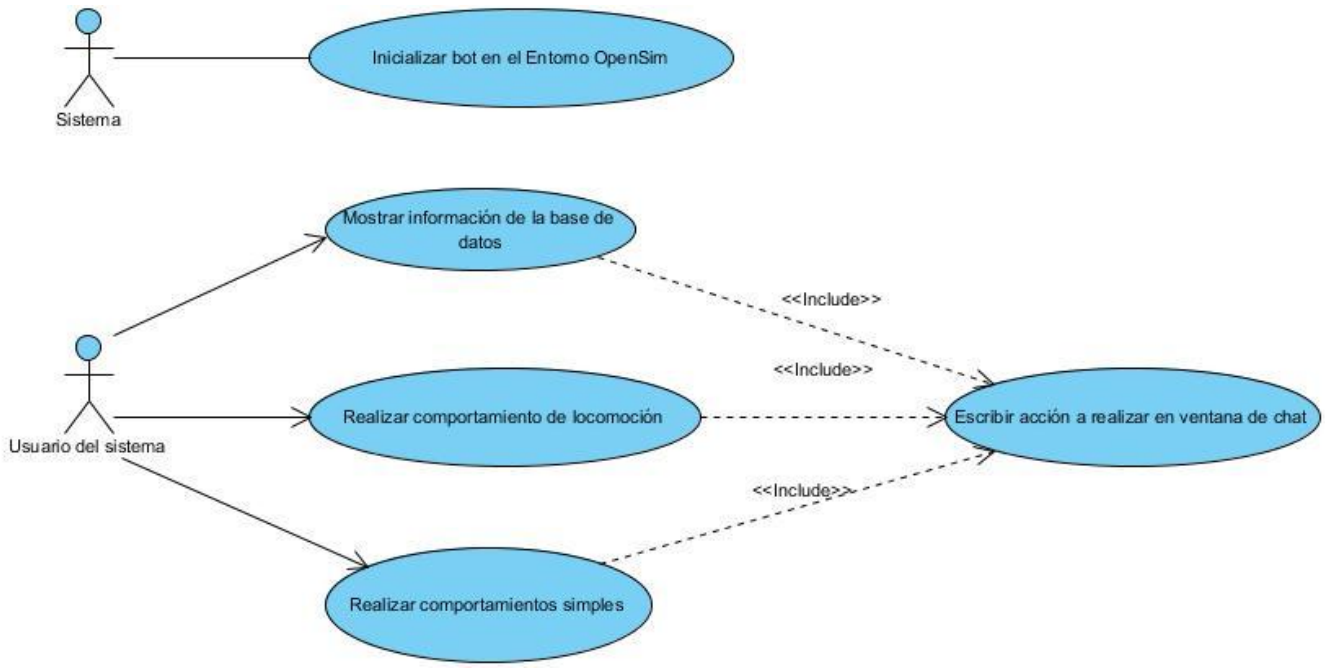


Ilustración 1. Diagrama de caso de uso del sistema

2.10.3 Descripción de los Casos de Uso del Sistema

Caso de Uso	Realizar comportamiento de locomoción
Actores	Usuario
Resumen	El caso de uso se inicia cuando un usuario del entorno virtual le solicita al bot mediante el chat una petición que requiera la locomoción de este.
Precondiciones	El sistema debe tener implementado el comportamiento correspondiente.
Referencias	RF3, RF3.1, RF3.2, RF3.3
Post condiciones	Realizar casos de uso que se desprendan de requisitos funcionales.
Prioridad	Crítico
Complejidad	Alta

Flujo Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1.El caso de uso inicia cuando el usuario desea realizar algunas de las siguientes acciones: • El usuario solicita la al bot mediante chat la opción de dirigirse a un lugar. • El usuario solicita al bot mediante chat la opción sígueme. • El usuario solicita al bot mediante chat la opción no me sigas.	
Sección "Dirigirse a un lugar"	
Acción del Actor	**Acciones del Sistema**
1. El usuario solicita al bot mediante chat la opción ir a lugar.	2. El sistema le muestra un conjunto de lugares a los cuales puede guiarle.
3. El usuario le informa mediante chat al bot el lugar al cual quiere que lo guie.	4. El sistema valida que el bot se encuentre en el estado libre o global. 5. El sistema termina el comportamiento actual. El sistema cambia al estado guía. 6. El sistema valida si la posición en la que se encuentra el bot es parte del grafo de camino del entorno. 7. De no ser parte del grafo de camino la posición del bot, este se dirige a la posición más cercana. 8. El sistema busca el camino mínimo entre la posición en la que se encuentra y la posición destino, a través del algoritmo para este fin.

	9. El sistema devuelve una lista que contiene el camino a recorrer. 10. El sistema ejecuta la locomoción del bot a través de la lista que contiene el camino a recorrer. 11. Termina el caso de uso.
Sección "Sígueme"	
Acción del Actor	**Acciones del Sistema**
1. El usuario solicita al bot mediante chat la opción sígueme.	2. El sistema valida que el bot se encuentre en el estado libre o global. 3. El sistema termina el comportamiento actual. 4. El sistema inicializa el comportamiento "FollowBehavior". 5. Termina el caso de uso.
Sección "No me sigas"	
Acción del Actor	**Acciones del Sistema**
1. El usuario solicita al bot mediante chat la opción no me sigas.	2. El sistema valida que el bot se encuentre en el estado libre o global. 3. Elsistema valida que el comportamiento actual sea sígueme. 4. El sistema finaliza el comportamiento "FollowBehavior". 5. Termina el caso de uso.

Tabla 2. Descripción de caso de uso Realizar comportamiento de locomoción

Caso de Uso	Inicializar bot en el entorno de OpenSim
Actores	Sistema
Resumen	El sistema proporciona una estructura fija con los componentes básicos para lograr un agente inteligente.
Precondiciones	El sistema permita la integración de la estructura a la

	arquitectura de OpenSim.
Referencias	RF1, RF1.1
Post condiciones	Realizar los casos de uso inmersos en el sistema
Prioridad	Crítica
Complejidad	Alta
Flujo Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el Sistema se ejecuta.	2. El sistema valida que el bot no haya sido autenticado con antelación. 3. El sistema ejecuta una estructura fija con los componentes básicos para lograr un agente inteligente. El sistema inicializa el estado global. 4. Termina el caso de uso.

Tabla 3. Descripción de caso de uso Inicializar bot en el entorno de OpenSim

Caso de Uso	Mostrar información de la base de datos
Actores	Usuario
Resumen	El caso de uso se inicia cuando un usuario del entorno virtual le solicita al bot mediante el chat la petición de conocer el horario.
Precondiciones	Que se haya logrado la conexión con la base de datos correctamente.
Referencias	RF4
Post condiciones	Mostrar al usuario los datos solicitados al bot.
Prioridad	Crítico
Complejidad	Media
Flujo Normal de los Eventos	

Acción del Actor	Respuesta del Sistema
1. El caso de uso inicio cunado el Usuario solicita la al bot mediante chat la opción de mostrar el horario.	2. El sistema valida que el bot se encuentre en el estado libre o global. 3. El sistema termina el comportamiento actual. 4. El sistema cambia al estado ayuda. 5. El sistema realiza una consulta a la base de datos. 6. El sistema devuelve los datos solicitados por el usuario. 7. Termina el caso de uso.

Tabla 4. Descripción del caso de uso Mostrar información de la base de datos

Caso de Uso	Realizar comportamiento simple.
Actores	Usuario
Resumen	El caso de uso se inicia cuando un usuario del entorno virtual le solicita al bot mediante el chat que vuele, deje de volar, se pare o se siente.
Precondiciones	El sistema debe tener implementado el comportamiento correspondiente.
Referencias	RF2, RF2.1, RF2.2, RF2.3, RF2.4, RF2.5
Post condiciones	Realizar casos de uso que se desprendan de requisitos funcionales.
Prioridad	Crítica
Complejidad	Alta
Flujo Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
El caso de uso inicia cuando el usuario desea realizar algunas de las siguientes acciones:	

• El usuario solicita la al bot mediante chat la opción volar. • El usuario solicita la al bot mediante chat la opción dejar de volar. • El usuario solicita la al bot mediante chat la opción sentarse. • El usuario solicita la al bot mediante chat la opción pararse.	
Sección “Volar”	
Acción del Actor	**Respuesta del Sistema**
1. El usuario solicita la al bot mediante chat la opción volar.	2. El sistema valida que el bot se encuentre en el estado libre o global. 3. El sistema termina el comportamiento actual. 4. El sistema inicializa el comportamiento “FlyBehavior”. 5. Termina el caso de uso.
Sección “Dejar de volar”	
Acción del Actor	**Respuesta del Sistema**
1. El usuario solicita la al bot mediante chat la opción dejar de volar.	2. El sistema valida que el bot se encuentre en el estado libre o global. 3. El sistema valida que el estado actual sea volar. 4. El sistema finaliza el comportamiento “FlyBehavior”. 5. Termina el caso de uso.
Sección “Sentarse”	
Acción del Actor	**Respuesta del Sistema**
1. El usuario solicita la al bot mediante	2. El sistema valida que el bot se

chat la opción sentarse.	encuentre en el estado libre o global. 3. El sistema termina el comportamiento actual. 4. El sistema activa la función “Sit”. 5. Termina el caso de uso.
Sección “Pararse”	
Acción del Actor	**Respuesta del Sistema**
1. El usuario solicita la al bot mediante chat la opción pararse.	2. El sistema valida que el bot se encuentre en el estado libre o global. 3. El sistema valida que el comportamiento actual sea sit. 4. El sistema activa la función “Stand”. 5. Termina el caso de uso.

Tabla 5. Descripción de caso de uso Realizar comportamiento simple

CAPÍTULO 3: DISEÑO, IMPLEMENTACIÓN Y RESULTADOS

En este capítulo se aborda la implementación del sistema, basado en todo el trabajo acumulado a lo largo del desarrollo de los capítulos anteriores y muestra la realización de los distintos artefactos correspondientes a cada uno de los flujos de trabajo analizados.

3.1 Modelo de diseño

En el diseño se modela el sistema y se confecciona su estructura, que sirve de soporte para la realización de todos los requisitos.
La realización de los CU del diseño contiene una descripción de los flujos de eventos textuales, diagramas de clases y diagramas de interacción.

3.1.1 Diagrama de clases del diseño.

Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como las relaciones existentes entre las mismas. A una clase del diseño, en la implementación del sistema, se le atribuyen visibilidad a sus atributos y operaciones.

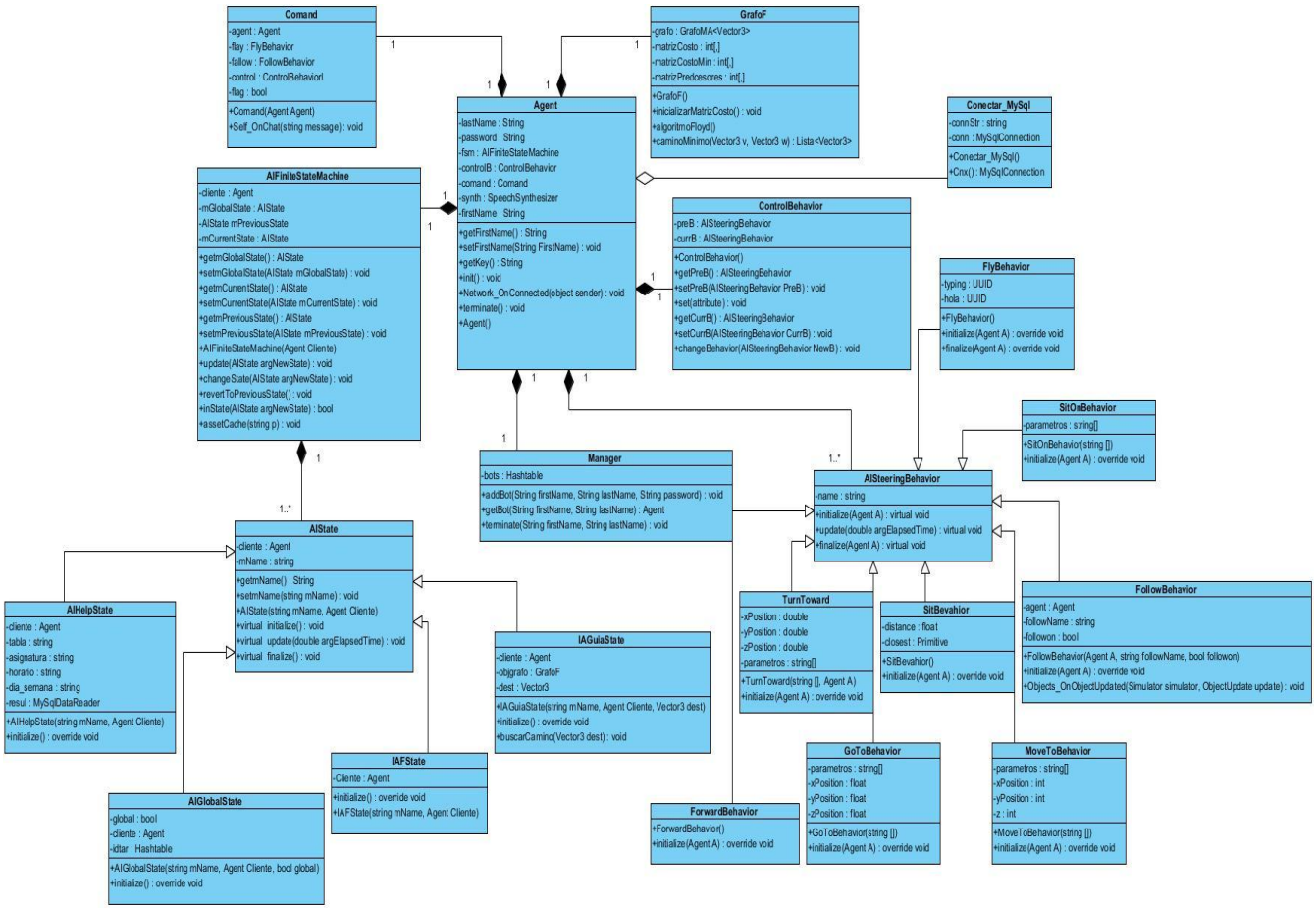


Ilustración 2. Diagrama de clases del diseño

3.1.2 Descripción de las clases del diseño

Nombre	Comand
Tipo de clase	Controladora
Atributo	**Tipo**
agent	Agent
flay	FlyBehavior
fallow	FollowBehavior
control	ControlBehavior
flag	bool
Responsabilidades	

Nombre	Descripción
Comand(Agent Agent)	Constructor de la clase.
void Self_OnChat(string message, OpenMetaverse.ChatAudibleLevel audible, OpenMetaverse.ChatType type, OpenMetaverse.ChatSourceType sourceType, string fromName, OpenMetaverse.UUID id, OpenMetaverse.UUID ownerid, OpenMetaverse.Vector3 position)	Controla todo el tráfico a través del chat.

Tabla 6. Descripción de la clase Comand

Nombre	Agent
Tipo de clase	Entidad
Atributo	**Tipo**
firstName	String
lastName	String
password	String
fsm	AIFiniteStateMachine
controlB	ControlBehavior
comand	Comand
synth	SpeechSynthesizer
Responsabilidades	
Nombre	**Descripción**
Agent(String firstName, String lastName, String password)	Constructor de la clase.
String getKey()	Devuelve el el nombre y el apellido.
String getFirstName()	Devuelve el valor de FirstName.

void setFirstName(String FirstName)	Cambia el valor de FirstName.
void init()	Inicia la sesión en el servidor.
void Network_OnConnected(object sender)	Emite un mensaje dando a conocer la correcta conexión con servidor.
void terminate()	Emite un mensaje dando a conocer la finalización de la sesión en el servidor.

Tabla 7. Descripción de la clase Agent

Nombre	AIFiniteStateMachine
Tipo de clase	Controladora
Atributo	**Tipo**
cliente	Agent
mGlobalState	AIState
mCurrentState	AIState
mPreviousState	AIState
Responsabilidades	
Nombre	**Descripción**
AIFiniteStateMachine(Agent Cliente)	Constructor de la clase.
void initialize()	Inicializa la máquina de estados.
void update(bool N)	Actualiza la máquina de estados.
void finalize()	Finaliza la máquina de estados.
void changeState(AIState argNewState)	Cambia el estado actual de la máquina de estados por uno nuevo.
void revertToPreviousState()	Revierte el cambio de estado al previo.
bool inState(AIState argNewState)	Devuelve verdadero si el estado

	actual es el pasado por parámetro.
AIState getMPreviousState()	Devuelve el estado previo.
setMPreviousState(AIState mPreviousState)	Cambia el estado previo por uno entrado por parámetro.
AIState getMCurrentState()	Devuelve el estado en que se encuentra la máquina de estados.
void setMCurrentState(AIState mCurrentState)	Cambia el estado actual de la máquina de estados.
AIState getMGlobalState()	Devuelve el estado global en que se encuentra la máquina de estados.
void setMGlobalState(AIState mGlobalState)	Cambia el estado global de la máquina de estados.

Tabla 8. Descripción de la clase AIFiniteStateMachine

Nombre	GrafoF
Tipo de clase	Entidad
Atributo	**Tipo**
grafo	GrafoMA<Vector3>
matrizCosto	int[,]
matrizCostoMin	int[,]
matrizPredcesores	int[,]
Responsabilidades	
Nombre	**Descripción**
GrafoF()	Constructor de la clase.
void inicializarMatrizCosto()	Inicializa la matriz de costos.
void algoritmoFloyd()	Realiza el algoritmo floyd.
ListaSE<Vector3> caminoMinimo(Vector3	Devuelve una lista con el camino

v, Vector3 w)	mínimo dado dos puntos.

Tabla 9. Descripción de la clase GrafoF

Nombre	Conectar_MySql
Tipo de clase	Entidad
Atributo	**Tipo**
connStr	string
conn	MySqlConnection
Responsabilidades	
Nombre	**Descripción**
Conectar_MySql()	Constructor de la clase.
MySqlConnection getCnx()	Devuelve el estado de la conexión.

Tabla 10. Descripción de la clase Conectar_MySql

Nombre	ControlBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
prevB	AISteeringBehavior
currB	AISteeringBehavior
Responsabilidades	
Nombre	**Descripción**
ControlBehavior()	Constructor de la clase.
AISteeringBehavior getCurrB()	Devuelve el behavior actual.
void setCurrB(AISteeringBehavior mCurrB)	Cambia el behavior actual.
AISteeringBehavior getPreB()	Devuelve el behavior previo.

setPreB(AISteeringBehavior mPreB)	Cambia el behavior previo.
void changeBehavior(AISteeringBehavior NewB)	Método que se encarga de realizar la transición de estados.

Tabla 11. Descripción de la clase ControlBehavior

Nombre	FlyBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
typing	UUID
hola	UUID
Responsabilidades	
Nombre	**Descripción**
FlyBehavior()	Constructor de la clase.
void initialize(Agent A)	Inicializa el behavior pasándole un agente.
finalize(Agent A)	Finaliza el behavior pasándole un agente.

Tabla 12. Descripción de la clase FlyBehavior

Nombre	SitBevahior
Tipo de clase	Entidad
Atributo	**Tipo**
distance	float
closest	Primitive
Responsabilidades	
Nombre	**Descripción**
SitBevahior()	Constructor de la clase.

initialize(Agent A)	Inicializa el behavior pasándole un agente.

Tabla 13. Descripción de la clase SitBevahior

Nombre	SitOnBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
parametros	string[]
Responsabilidades	
Nombre	**Descripción**
SitOnBehavior(string[] parametros)	Constructor de la clase.
void initialize(Agent A)	Inicializa el behavior pasándole un agente.

Tabla 14. Descripción de la clase SitOnBehavior

Nombre	Manager
Tipo de clase	
Atributo	**Tipo**
bots	Hashtable
Responsabilidades	
Nombre	**Descripción**
void addBot(String firstName, String lastName, String password)	Adiciona un bot pasándole por parámetros nombre, apellidos y contraseña.
Agent getBot(String firstName, String lastName)	Devuelve un bot dado el nombre y apellido.
void terminate(String firstName, String lastName)	Termina la sesión de un bot identificado por el nombre y el

	apellido.

Tabla 15. Descripción de la clase Manager

Nombre	AIState
Tipo de clase	Entidad
Atributo	**Tipo**
mName	string
cliente	Agent
Responsabilidades	
Nombre	**Descripción**
AIState(string mName,Agent Cliente)	Constructor de la clase.
string getmName()	Devuelve el nombre del estado.
setmName(string mName)	Cambia el nombre del estado.
virtual void initialize()	Método virtual redefinido en las clases que heredan de esta.
virtual void update(double argElapsedTime)	Método virtual redefinido en las clases que heredan de esta.
virtual void finalize()	Método virtual redefinido en las clases que heredan de esta.

Tabla 16. Descripción de la clase AIState

Nombre	AIHelpState
Tipo de clase	Entidad
Atributo	**Tipo**
cliente	Agent
table	string
asignatura	string

horario	string
dia_semana	string
resul	MySqlDataReader
Responsabilidades	
Nombre	**Descripción**
AIHelpState(string mName, Agent Cliente)	Constructor de la clase.
override void initialize()	Método que imprime la respuesta de la consulta a la base de datos.

Tabla 17. Descripción de la clase AIHelpState

Nombre	AIGlobalState
Tipo de clase	Entidad
Atributo	**Tipo**
global	bool
cliente	Agent
idtar	Hashtable
Responsabilidades	
Nombre	**Descripción**
AIGlobalState(string mName, Agent Cliente, bool global)	Constructor de la clase.
override void initialize()	Inicializa el estado.
void Objects_OnObjectUpdated(Simulator simulator, ObjectUpdate update, ulong regionHandle, ushort timeDilation)	Actualiza los componentes del entorno virtual.

Tabla 18. Descripción de la clase AIGlobalState

Nombre	IAGuiaState

Tipo de clase	Entidad
Atributo	**Tipo**
cliente	Agent
objGrafo	GrafoF
Responsabilidades	
Nombre	**Descripción**
IAGuiaState(string mName, Agent Cliente)	Constructor de la clase.
void buscarCamino(Vector3 dest)	Método que realiza la búsqueda de camino mínimo de una posición a otra, y realiza el recorrido de este por el bot.

Tabla 19. Descripción de la clase IAGuiaState

Nombre	AISteeringBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
name	string
Responsabilidades	
Nombre	**Descripción**
virtual void initialize(Agent A)	Método virtual redefinido en las clases que heredan de esta.
virtual void update(double argElapsedTime)	Método virtual redefinido en las clases que heredan de esta.
virtual void finalize(Agent A)	Método virtual redefinido en las clases que heredan de esta.

Tabla 20. Descripción de la clase AISteeringBehavior

Nombre	TurnToward
Tipo de clase	Entidad
Atributo	**Tipo**
xPosition	double
yPosition	double
zPosition	double
parametros	string[]
Responsabilidades	
Nombre	**Descripción**
override void initialize(AILibrary.Agent A)	Método que inicializa el behavior.

Tabla 21. Descripción de la clase TurnToward

Nombre	ForwardBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
Responsabilidades	
Nombre	**Descripción**
ForwardBehavior()	Constructor de la clase.
override void initialize(Agent A)	Método que inicializa el behavior.

Tabla 22. Descripción de la clase ForwardBehavior

Nombre	GoToBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
parametros	string[]

xPosition	float
yPosition	float
zPosition	float
Responsabilidades	
Nombre	**Descripción**
override void initialize(Agent A)	Método que inicializa el behavior.

Tabla 23. Descripción de la clase GoToBehavior

Nombre	MoveToBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
parametros	string[]
xPosition	int
yPosition	int
zPosition	int
Responsabilidades	
Nombre	**Descripción**
MoveToBehavior(string[] parametros)	Constructor de la clase.
override void initialize(Agent A)	Método que inicializa el behavior.

Tabla 24. Descripción de la clase MoveToBehavior

Nombre	FollowBehavior
Tipo de clase	Entidad
Atributo	**Tipo**
agent	Agent
followName	string

followon	bool
Responsabilidades	
Nombre	**Descripción**
FollowBehavior(Agent A,string followName,bool followon)	Constructor de la clase.
override void initialize(Agent A)	Método que inicializa el behavior.
void Objects_OnObjectUpdated(Simulator simulator, ObjectUpdate update, ulong regionHandle, ushort timeDilation)	Actualiza los componentes del entorno virtual.

Tabla 25. Descripción de la clase FollowBehavior

3.1.3 Diagramas de secuencia del diseño

A continuación se encuentran los diagramas de secuencia del diseño para cada uno de los distintos CU y clases existentes en dicho sistema.

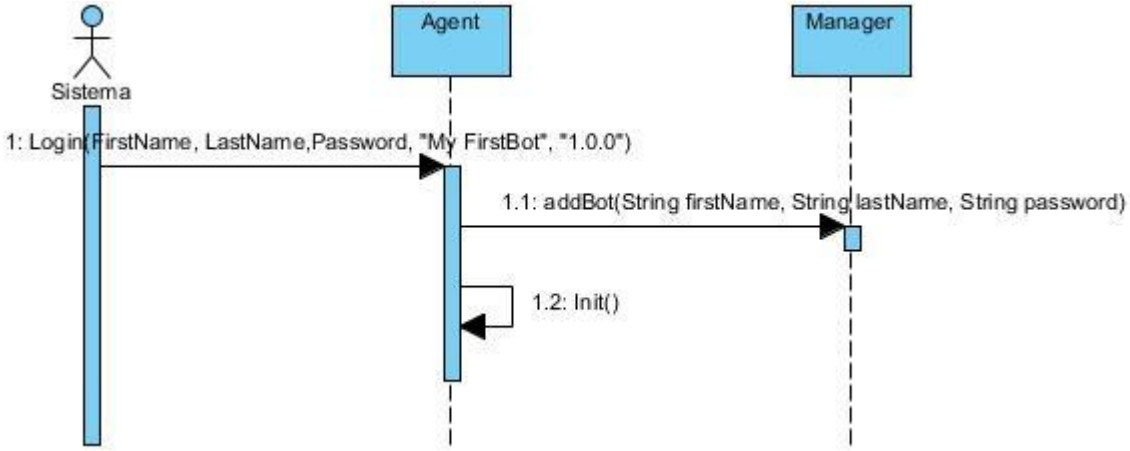


Ilustración 3. Caso de uso bot en el entorno de OpenSim

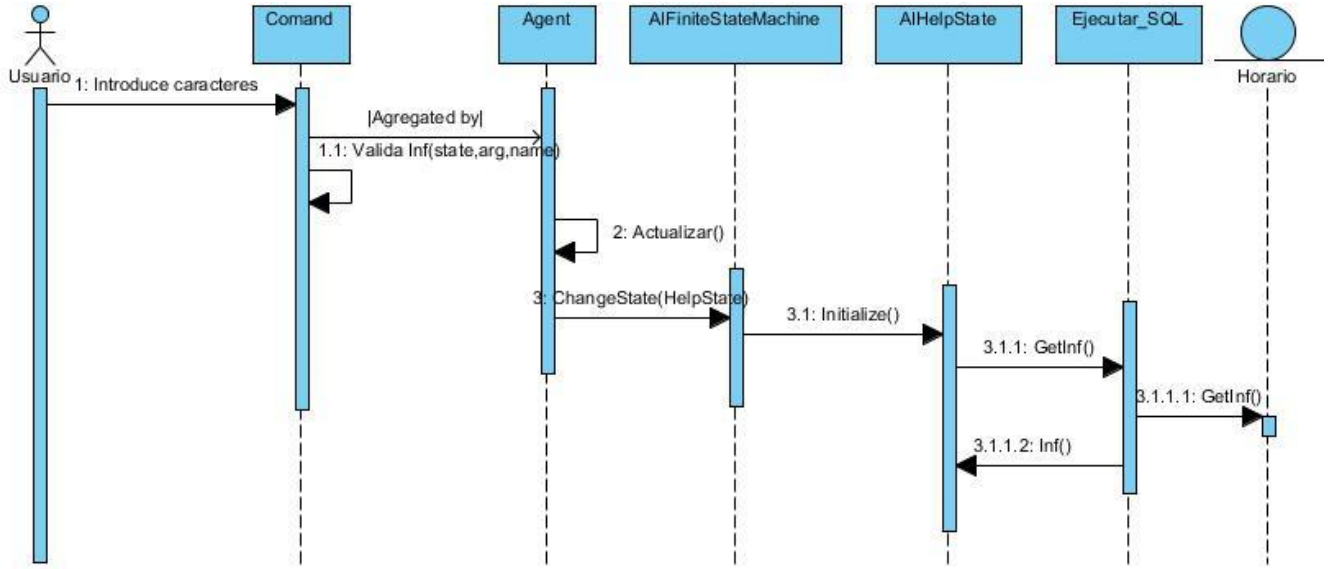


Ilustración 4. Caso de uso Mostrar información de la base de datos

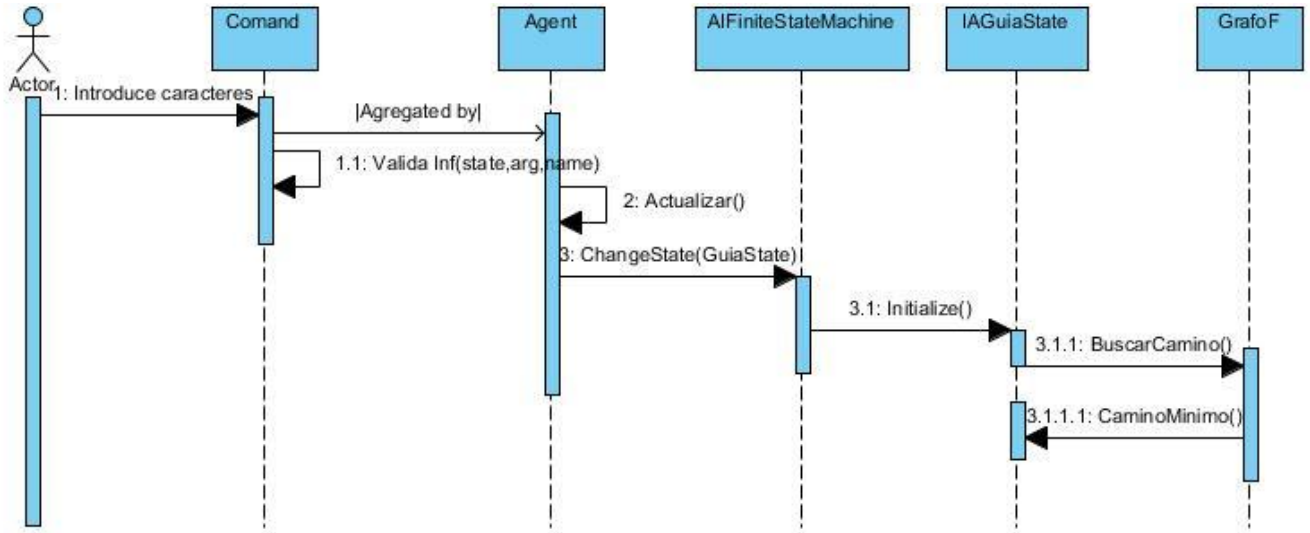


Ilustración 5. Caso de uso realizar comportamiento de locomoción. Sección Ir a lugar

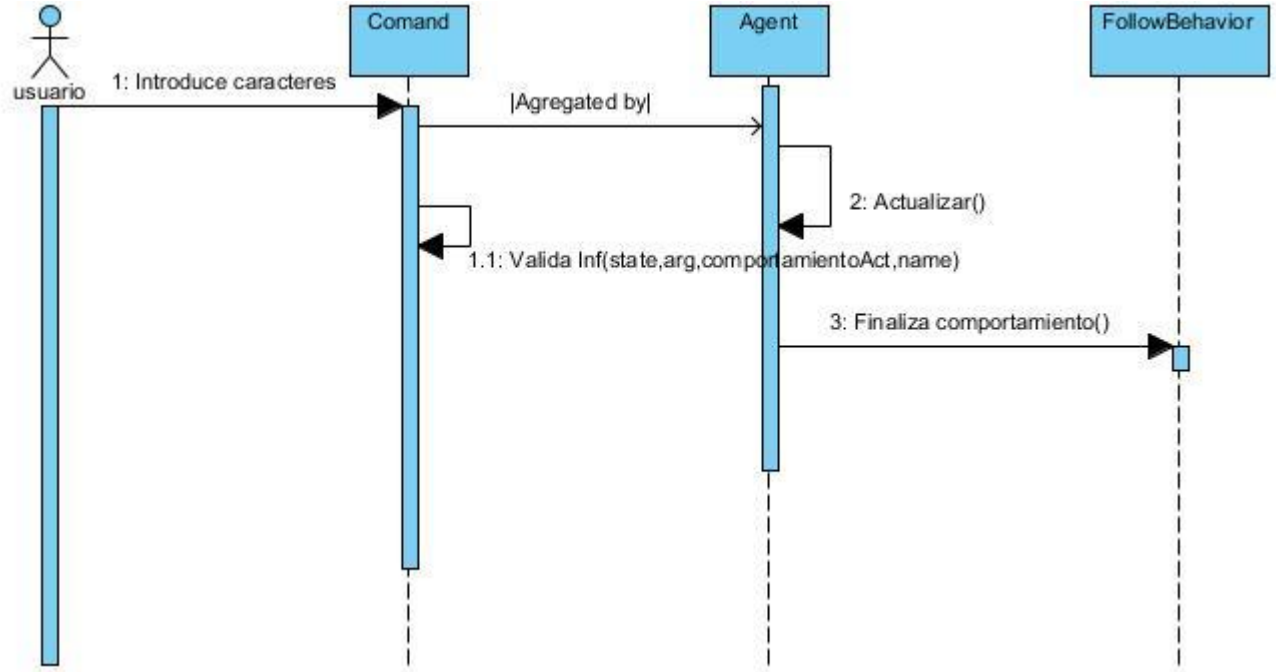


Ilustración 6. Caso de uso realizar comportamiento de locomoción. Sección no me sigas

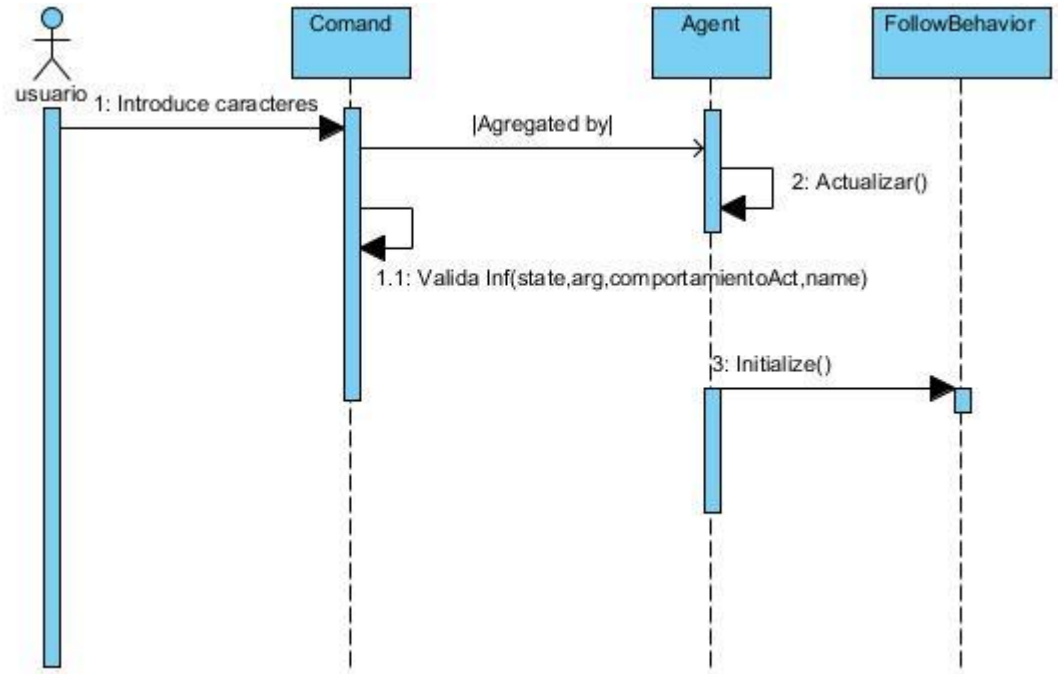


Ilustración 7. Caso de uso realizar comportamiento de locomoción. Sección Sígueme

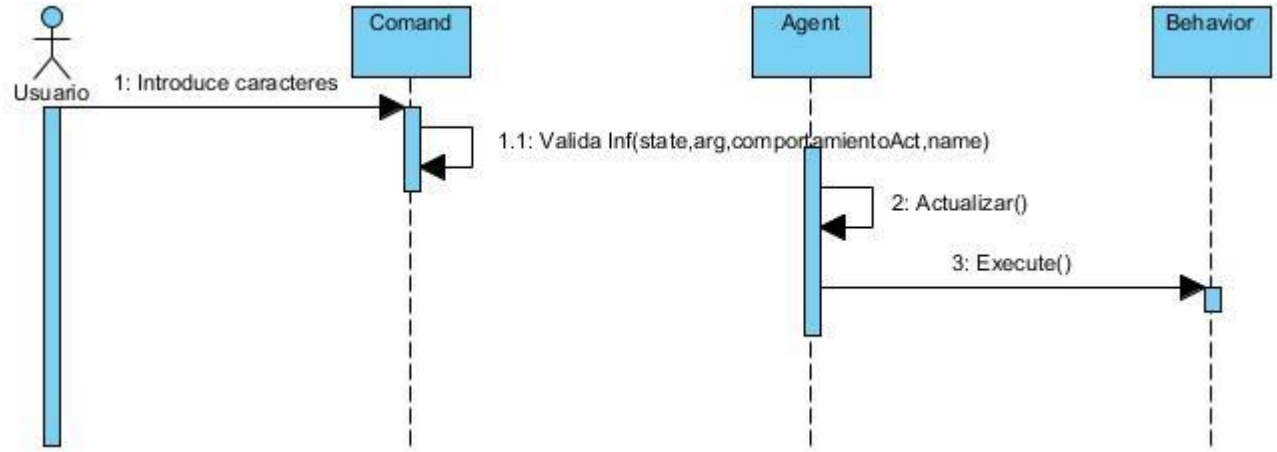


Ilustración 8. Caso de uso Realizar comportamiento simple

3.2 Patrones de diseño

Los patrones de diseño son una solución reutilizable de problemas recurrentes que ocurren durante el desarrollo del software. Ayudan a entender las soluciones del problema con un vocabulario igual lo que permite un mejor entendimiento. (Ivar Jacobson)

- **Singleton**

Este patrón se empleó en la relación de clases existente entre Agent y IAFiniteStateMachine. El patrón Singleton consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella**,** por lo que si más de un objeto necesita utilizar una instancia de dicha clase, esos objetos comparten la misma instancia de la clase Singleton. (Erich Gamma, 1995)

- **Adaptador**

El patrón adaptador se empleó para el diseño de los modelos de comportamiento, porque permite convertir la interfaz de una clase para que se adapte a lo que el cliente que la usa necesita. Los modelos de comportamiento no tiene todos las mismas entradas en los datos por lo que requieren interfaces diferentes, este problema se solucionó con la inclusión de este patrón. La clase IASteeringBehavior sería la clase a adaptar, pues no resulta práctico adaptar cada uno de los modelos por separado. La utilización del adaptador brinda flexibilidad al diseño, para que un adaptador trabaje con muchas clases a adaptar. (Erich Gamma, 1995)

3.3 Modelo de implementación

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

Los diagramas de componentes y de despliegue conforman lo que se conoce como un modelo de implementación.

3.3.1 Diagrama de componentes

A continuación se muestran los componentes que forman parte del diseño del sistema, imprescindibles para mostrar la funcionalidad del componente Módulo IA.

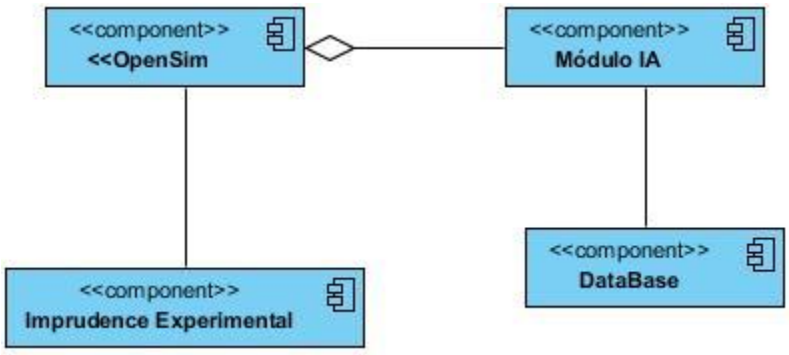
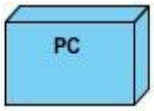


Ilustración 9. Diagrama de componentes

3.3.2 Diagrama de despliegue

En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir, se sitúa el software en el hardware que lo contendrá una vez terminada la realización del mismo. Cada hardware se representa como un nodo. Un nodo se representa como un cubo y es un elemento donde se ejecutan los componentes desarrollados. (Ivar Jacobson, 2000)



3.4 Modelo de prueba

Los modelos de pruebas responden a la implementación de los casos de uso más importantes de la aplicación. Para documentarlos se utilizó el formato de los documentos oficiales del CEDIN. Se realizan pruebas de caja negra, especialmente dirigidas a interacción del usuario con el sistema y las respuestas de este.

3.4.1 Diseño de caso de prueba. Caso de uso Visualizar bot inteligente para el entorno de OpenSim

Descripción general:

El sistema inicia el caso de uso al ejecutarse y proporciona una estructura fija con los componentes básicos para lograr un agente inteligente en el entorno de OpenSim.

Condiciones de ejecución:

- Estar funcionando el Servidor de OpenSim.
- Estar correctos los datos de conexión.

Secciones a probar en el caso de uso:

Nombre de la sección	Escenarios de la sección	Descripción de la Funcionalidad	Flujo central

SC 1. Inicializar bot inteligente para el entorno de OpenSim	EC 1.1 Inicializar bot inteligente para el entorno de OpenSim de forma exitosa.	Al ejecutarse el sistema, este se conecta con el servidor OpenSim, el usuario debe introducir por consola los datos necesarios para realizar la conexión del bot, como son nombre, apellido y contraseña.	Se ejecuta el sistema. Se conecta con la base de datos. Se introducen los datos mediante consola. Se muestra un mensaje de conexión satisfactoria.
	EC 1.2 Error de integración del bot inteligente en el entorno OpenSim.	Al ejecutarse el sistema, este se conecta con el servidor OpenSim, y se evidencia que este usuario ya se encuentra conectado.	Se ejecuta el sistema. Se conecta con la base de datos. Se introducen los datos mediante consola. Se muestra un mensaje de conexión ya realizada.

Tabla 26. Diseño de caso de prueba. Caso de uso Visualizar bot inteligente para el entorno de OpenSim

3.4.2 Diseño de caso de prueba. Caso de uso Mostrar información de la base de datos

Descripción general:

El usuario inicia el caso de uso cuando le solicita al bot mediante el chat la petición de conocer el horario.

Condiciones de ejecución:

- Estar funcionando el Servidor de OpenSim.
- Iniciar el bot correctamente.
- Estar funcionando la conexión con la base de datos.

Secciones a probar en el caso de uso:

Nombre de la	Escenarios de la	Descripción de la	Flujo central

sección	sección	Funcionalidad	
SC 1. Mostrar información de la base de datos	EC 1.1 Mostrar información de la base de datos.	El usuario solicita el horario y el sistema realiza una consulta a la base de datos que contiene la información deseada.	Se realiza la solicitud de mostrar el horario mediante chat. El sistema realiza la consulta a la base de datos y muestra por consola y mediante voz el horario contenido en la misma.
	EC 1.2 No se logra mostrar horario.	El usuario solicita el horario y el sistema se encuentra inmerso en otra funcionalidad por lo que niega el caso mostrar horario.	Se realiza la solicitud de mostrar el horario mediante chat. El sistema identifica que se no se encuentra en un estado en el que pueda atender la solicitud y devuelve un mensaje mediante la consola informando al usuario.

Tabla 27. Diseño de caso de prueba. Caso de uso Mostrar información de la base de datos

3.4.3 Diseño de caso de prueba. Caso de uso realizar comportamiento de locomoción

Descripción general:

El usuario inicia el caso de uso cundo le solicita mediante el chat al bot la opción ir a un lugar determinado, señalado por un posición. El sistema realiza los cálculos necesarios para hallar el camino mínimo entre la posición del bot y la posición deseada.

Condiciones de ejecución:

- ❖ Estar funcionando el Servidor de OpenSim.
- ❖ Iniciar el bot correctamente.
- ❖ El bot no esté a una distancia excesiva del grafo de camino.

Secciones a probar en el caso de uso:

Nombre de la sección	**Escenarios de la sección**	**Descripción de la Funcionalidad**	**Flujo central**
SC 1. Realizar comportamiento de locomoción.	EC 1.1 Realizar comportamiento de locomoción.	El usuario solicita al bot mediante chat que este le lleve a una posición, y este le guía hasta llegar a la misma.	El usuario solicita al bot mediante chat, la opción de llevarle a un lugar. El sistema le devuelve un conjunto de posiciones a las que le puede guiar. El usuario selecciona la posición deseada y el bot recorre el camino hasta llegar a la posición. Se muestra al bot como realiza el comportamiento.
	EC 1.2 Falla en realizar comportamiento de locomoción.	El usuario solicita al bot mediante chat que este le lleve a una posición, y este le muestra un mensaje negándose a cambiar a dicho estado.	El usuario solicita al bot mediante chat, la opción de llevarle a un lugar. El sistema le informa mediante un mensaje de consola que se encuentra en un estado en el cual no puede atender su solicitud.

Tabla 28. Diseño de caso de prueba. Caso de uso realizar comportamiento de locomoción

3.4.4 Diseño de caso de prueba. Caso de uso realizar comportamiento simple

Descripción general:

El usuario inicia el caso de uso cuando le solicita al bot mediante el chat la petición de realizar alguno de los comportamientos simples, como son: volar, dejar de volar, sentarse y pararse.

Condiciones de ejecución:

- Estar funcionando el Servidor de OpenSim.
- Iniciar el bot correctamente.
- Estar funcionando la conexión con la base de datos.

Secciones a probar en el caso de uso:

Nombre de la sección	Escenarios de la sección	Descripción de la Funcionalidad	Flujo central
SC 1. Realizar comportamiento simple.	EC 1.1 Realizar comportamiento simple.	El usuario solicita mediante chat al bot la opción de realizar un comportamiento simple, entre los que se encuentra, volar, dejar de volar, sentarse y pararse.	El usuario realiza mediante chat al bot la opción de realizar un comportamiento simple. El sistema activa el comportamiento. Se muestra como el bot realiza el comportamiento.
	EC 1.2 Negarse a realizar comportamiento simple.	El usuario solicita mediante chat al bot la opción de realizar un comportamiento simple, entre los que se encuentra, volar, dejar de volar, sentarse y pararse y el sistema se niega.	El usuario realiza mediante chat al bot la opción de realizar un comportamiento simple. El sistema comprueba el comportamiento actual y se da cuenta que el comportamiento que se

			solicita no cumple las condiciones para ejecutarse. El bot muestra un mensaje informativo.

Tabla 29. Diseño de caso de prueba. Caso de uso realizar comportamiento simple

CONCLUSIONES

Se obtuvo un módulo que cumple con los requerimientos propuestos, donde el usuario de OpenSim, puede tener una mayor sensación de inteligencia brindada por el sistema. La aplicación permite dada su estructura una futura expansión de las funcionalidades.

Como resultado de la investigación realizada se pudo evidenciar que la biblioteca OpenMetaverse es realmente útil para conectarnos a cualquier metaverso basado en Second Life y OpenSim, sin embargo, existe muy poca documentación sobre cómo utilizarla.

Se pudo conocer que la plataforma OpenSim es de un alto grado de extensibilidad. Sin embargo dada las características del sistema se requiere de un tratamiento exhaustivo de las excepciones para cualquier tipo de arquitectura que se pretenda integrar a la plataforma.

RECOMENDACIONES

Introducir nuevas técnicas para la toma de decisiones con el fin ampliar las capacidades, del módulo.

Proveer de una discretización del entorno, como malla de navegación, para disminuir el tiempo de respuesta en la búsqueda de caminos dentro del mismo.

Referencia bibliográfica

Wikipedia. (20 de diciembre de 2012). Recuperado el 20 de diciembre de 2012, de Wikipedia: http://es.wikipedia.org/wiki/C_Sharp

EcuRed. (1 de junio de 2013). Recuperado el 1 de junio de 2013, de EcuRed: http://www.ecured.cu/index.php/Visual_Paradigm

Wikipedia. (1 de junio de 2013). Recuperado el 1 de junio de 2013, de Wikipedia: http://es.wikipedia.org/wiki/XAMPP

Wikipedia. (30 de mayo de 2013). Recuperado el 30 de mayo de 2013, de Wikipedia: http://es.wikipedia.org/wiki/Visual_Studio_2008#Visual_Studio_2008

Algoritmos sobre Grafos I. (s.f.). Departamento de Técnicas de Programación, Universidad de las Ciencias Informáticas.

Bartle, R. (2003). *Designing Virtual Worlds. New Riders Games.*

Bell, M. W. (2008). *Toward a definition of virtual worlds. Journal of Virtual Worlds Research.*

Brownlee, J. (s.f.). *Finite State Machines.*

Erich Gamma, R. H. (1995). *Design patterns: Elements of reusable object-oriented sotfware.* Addison Wesley.

Gavaldà, D. J. (2011). *Inteligencia Artificial.* Universidad Oberta de Catalunya.

Guerra, A. (s.f.). *A Framework for Building Intelligent.* Pace University.

Herranz, S. S. (2005). *Integracion de SOAR en ICa.*

HUDSBERGER, S. (2009). *Foreign language learn-ing in Second Life and the implications for resource.*

Ivar Jacobson, G. B. (2000). *El Proceso Unificado de Desarrollo de Software.* Malaga: Addison Wesley.

Jean-Marc Jézéquel, M. T. (1999). *Design Patterns and contracts.* Addison Wesley.

Larman, C. (2da Edición). *UNL y Patrones.* Prentice Hall.

Legal, P. (2005-2009). *Principio Legal* . Recuperado el 21 de enero de 2013, de Licencias de Software Libre: www.Licencias de software libre - Principio Legal.htm

Luis Daniel Hernández López, J. L. (2008). *Sistema de Percepción para Agentes Virtuales.* Ciudad de La Habana: UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS.

Martín del Brío, B. M. (2001). *Redes Neuronales y Sistemas Difusos.* Zaragoza, España.

PerezSchofield, J. B. (s.f.). Desarrollo rápido de aplicaciones: Patrones de Diseño. Grupo IMO SI1.

Rosenbloom, P. J. (1991). *A preliminary analysis of the Soar architecture as a basis for general intelligence.* EEUU.

S.Pressman, R. (2002). *Igeniería del software, un enfoque práctico.*

Sánchez., E. R. (Octubre de 2010). Aplicación de la Herramienta Open Source Sloodle y las Tecnologías del Procesamiento del lenguaje Natural para el Desarrollo de una Plataforma de Virtual Learning en la Universidad Carlos III de Madrid. *Memoria_PFC_EduardoRojo* . Madrid, España: Universidad Carlos III de Madrid.

Venereo, L. M. (2011). HERRAMIENTA ABIERTA DE VISUALIZACION INTERACTIVA. La Habana, La Habana, Cuba: UNIVERSIDAD DE LAS CIENCIAS INFORMATICAS.