UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5

PROPUESTA DE APLICACIÓN DE LOS MODELOS BIOINSPIRADOS MEMBRANA Y COLONIA DE HORMIGAS EN LOS ENTORNOS VIRTUALES

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

Autor: **Jeydi Alina Santiago Rubiera**

Tutor: **MsC. Yuniesky Coca Bergolla**

Co-tutor: **Ing. Dailyn García Domínguez**

La Habana

Junio 2011

Declaración Jurada de Autoría

Yo, Jeydi Alina Santiago Rubiera , 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 declaración de autoría en La Habana a los ___ días del mes de _____ del año 2011.

Jeydi Alina Santiago Rubiera

Firma del Autor

Yuniesky Coca Bergolla

Firma del Tutor

Si supiera lo que estoy haciendo no lo llamaría investigación,
¿no crees?

Albert Einstein

Datos de Contacto

MSc. Yuniesky Coca Bergolla (*ycoca@uci.cu*)

Miembro del Grupo de Animación del Centro de Desarrollo Electrónico (CDE) de la Universidad Central de Las Villas (UCLV) (2001-2003). Desarrollo del Software *Tensoft III* Sistema basado en casos con tratamiento de la incertidumbre para la detección de la Hipertensión Arterial (2003). Miembro del proyecto SIMPRO al frente del área de Inteligencia Artificial, desarrollando entre otras actividades la detección de infracciones del conductor y el control de los autos autónomos para un simulador de auto, presentado en la Feria Internacional *Informática 2007* (2003 - 2007). Jefe del departamento de la Especialidad en la facultad 5 de la UCI (2005-2007). Líder del proyecto de investigación *Interacción de Elementos Virtuales* (2005-2007). Miembro del comité científico de los talleres de Realidad Virtual e Inteligencia Artificial de las Conferencias Científicas UCiencia (2005 - actualidad). Jefe del departamento de Práctica Profesional y Realidad Virtual en la facultad 5 de la UCI (2007-2010). Miembro del grupo de Investigación de Visualización y Realidad Virtual (2007- actualidad). Árbitro de la publicación *Serie Científica* de la Universidad de las Ciencias Informáticas (2008 - actualidad). Subdirector de formación del Centro de Informática Industrial (2010 - actualidad).

Ing. Dailyn García Domínguez (*dgdominguez@uci.cu*)

Administradora de redes del Centro de Diagnóstico Integral *Angel Márquez* Anzuátegui, Venezuela (marzo - agosto 2008). Graduada de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas en el año 2009. Obtuvo el Sello Forjadores del Futuro en el año 2009. Miembro del Departamento Técnicas de Programación (2009-2010). Miembro del Centro de Informática Industrial (2010 - actualidad).

Agradecimientos

Agradecer a mis padres Jesús y Digna por todo el amor y el sacrificio para darme la oportunidad de estar aquí hoy.

A mis abuelos Digna, Marta, Giraldo y Félix, a mis hermanos Jesús y Dayán y a toda mi familia que siempre me han apoyado incondicionalmente de una forma u otra y han confiado en mí.

A mi novio Jandy por estar en cada momento y darme fuerzas para seguir porque sin su apoyo no hubiera llegado este momento.

Gracias a Lidia, Lidita, Clemente y a mis suegros Omar y Edenia por la gran acogida a la familia y el apoyo en estos años.

A mis tutores Coca y Dailyn por el apoyo brindado a lo largo de todo este año, por la confianza depositada en mí, los consejos constante y la dedicación.

A Consuegra, Rubén y Ray por su inmensa ayuda para llevar este trabajo adelante.

A mis amigas de siempre Iliana, Yeimy, Hanny y a todas mis amistades de Cárdenas, gracias por su incondicional amistad durante tantos años.

A mis amigos de la UCI Lisbeth, Capote, Tomás, Daylen, Yuniel y a todos mis compañeros, gracias por estos años maravillosos.

Muchas gracias a todos los que de una forma u otra contribuyeron con mi formación.

Gracias a la Universidad y a la Revolución por brindarme la oportunidad de convertirme en profesional.

Dedicatoria

Dedico este trabajo a las personas más importantes en mi vida, mis padres.

A mi abuelo Félix que donde quiera que esté estará orgulloso de mí.

Gracias por impulsarme a realizar mis sueños.

Resumen

Entre los temas principales de la Inteligencia Artificial están los modelos bioinspirados, estos son creados a partir del comportamiento de organismos naturales y son utilizados para resolver diversos problemas en entornos virtuales que involucran agentes inteligentes, que utilizan modelos cognitivos para moverse dentro del mundo. Entre los modelos bioinspirados están: Redes Neuronales, Algoritmos Genéticos, Colonias de Hormigas, Bandada de Pájaros y recientemente se incorporó el modelo Membranas. En el presente trabajo se realizó un estudio de los modelos bioinspirados antes mencionados, de los cuáles se analizaron las características y principales aplicaciones. Como resultado se presenta un nuevo modelo cognitivo denominado MCMemb, el cual se realizó utilizando el modelo Membranas y resulta una nueva opción para el movimiento de los agentes en el entorno. Además se creó una aplicación que evidencia el proceso que realiza el modelo Colonia de Hormigas para realizar la búsqueda eficiente del camino más corto, a partir del desarrollo de esta aplicación se obtuvo una herramienta para realizar pruebas al algoritmo que permite variar los parámetros del mismo. Para la creación de dicha aplicación se seleccionaron las herramientas adecuadas para el desarrollo de la misma, teniendo en cuenta que fuera software libre, así como la metodología óptima para el proceso de desarrollo.

Palabras clave: agentes creíbles, entorno virtual, inteligencia artificial, modelos bioinspirados, modelos cognitivos.

Índice general

Índice de figuras

Índice de tablas

Introducción

En la actualidad la Informática forma parte esencial en la vida de las personas, la encontramos presente en distintas esferas de la sociedad, desde la comunicación hasta la gestión de negocios. Una de las ramas de la informática que ha tenido un mayor auge en estos últimos tiempos es la Realidad Virtual (RV) debido a la gran cantidad de aplicaciones que tiene la misma: en la medicina, la meteorología, la carrera militar, la industria del entretenimiento e investigaciones científicas.

Otra área de estudio de la informática relacionada con la RV es la Inteligencia Artificial (IA), la cual según la Real Academia de la Lengua Española es el ”desarrollo y utilización de ordenadores con los que se intenta reproducir los procesos de la inteligencia humana”. La IA es una de las ramas de la Ciencia de la Computación que evoluciona con mayor rapidez, es una de las disciplinas más prometedoras y a la que varios científicos están dedicando muchísimos esfuerzos en todo el mundo [Bello, 1998]. La IA es muy utilizada en el área de los videojuegos y la simulación.

En Cuba se ha impulsado la informatización de la sociedad debido a la gran importancia que esta representa para el desarrollo del país, principalmente la economía. En la esfera de la Informática específicamente la RV se ha estado desarrollando en la Facultad 5 de la Universidad de las Ciencias Informáticas (UCI), en proyectos como simuladores militares, simuladores de autos, videojuegos y otros, en varios de estos aplicando técnicas de IA.

Uno de los contenidos principales de IA son los modelos bioinspirados que son aquellos creados a partir del comportamiento y la capacidad de los organismos naturales ante diversos problemas, teniendo en cuenta además las habilidades de adaptación y aprendizaje del me-

dio en la medida en que va evolucionando. Las ventajas de estos modelos radican en que son dinámicos, flexibles, robustos, auto-organizados, compuestos por elementos simples y descentralizados. En la esfera de Inteligencia Artificial estos modelos son muy utilizados para solución de problemas, algunos son: Redes Neuronales, Algoritmos Genéticos, Colonias de Hormigas, Bandada de Pájaros y recientemente se incorporó el modelo Membranas.

Una Red Neuronal es un modelo computacional que trata de simular el funcionamiento del cerebro humano, se basan principalmente en la arquitectura del mismo pero no llega a desarrollar una réplica del órgano.

El modelo de los Algoritmos Genéticos está inspirado en la teoría de la evolución de las especies de Charles Darwin. Este modelo presenta componentes que están inspirados en los cromosomas y en las operaciones que se pueden realizar entre estos.

Las Colonias de Hormigas están basadas en el comportamiento de las colonias de hormigas naturales, este modelo no está centrado en la inteligencia de un solo individuo, sino que está fundamentado en la inteligencia colectiva que emerge de la interacción entre los individuos. Este modelo se caracteriza por su simplicidad y robustez.

El modelo de las Bandadas de Pájaros se inspiró en el comportamiento social y biológico de los organismos, específicamente en las habilidades de algunos grupos de animales como las bandadas de aves. Las aplicaciones de esta metaheurística se basan principalmente en problemas de optimización continuos, optimización de funciones y multiobjetivo.

El modelo Membranas se basa en la estructura y funcionamiento de las células vivas de los organismos, así como también de la forma en que las células se organizan en tejidos. Se trata de sistemas de membranas que procesan objetos de manera localizada, en los que la comunicación entre los compartimentos y con el ambiente juega un papel esencial.

Un entorno virtual es la simulación de un mundo o entorno inspirado o no en la realidad, en el cual los usuarios pueden interactuar entre sí a través de personajes o avatares, y utilizar objetos virtuales. Estos entornos son representados por gráficos en dos dimensiones(2D) y tres dimensiones(3D). El desarrollo constante de la RV exige mejoras continuas en los entornos virtuales, razón por la cual es necesario aplicar diversas técnicas como los modelos

bioinspirados, con el fin de lograr entornos acorde al desarrollo. Por lo antes planteado es que se define como problema científico de este trabajo: *¿Cómo aplicar los modelos bioinspirados membranas y colonias de hormigas en los entornos virtuales?* Se plantea como objetivo, *proponer vías para la aplicación de los modelos bioinspirados membranas y colonias de hormigas en los entornos virtuales*. Para cumplir con el presente objetivo se tomará como objeto de estudio *la aplicación de modelos bioinspirados en sistemas de visualización gráfica*. Se trabajará principalmente en *la aplicación de los modelos bioinspirados membrana y colonia de hormigas en los entornos virtuales*.

Como tareas investigativas se trazaron:

- Elaboración de un informe sobre modelos bioinspirados y su relación con la Realidad Virtual.
- Confección de una propuesta de aplicación para la incorporación de los modelos bioinspirados Membrana y Colonia de Hormigas a entornos virtuales.
- Selección y justificación de las técnicas y algoritmos a utilizar.
- Selección y fundamentación de las herramientas necesarias para desarrollar la aplicación para realizar pruebas sobre el algoritmo colonia de hormiga.
- Desarrollo del análisis y diseño de la aplicación.
- Realización de la aplicación.

Los métodos de investigación que soportan el desarrollo del presente trabajo son la combinación dialéctica de los métodos Teóricos y Empíricos. Como Métodos Teóricos, el método Histórico-Lógico permite realizar el estudio de la evolución y desarrollo de la inteligencia artificial, específicamente aplicada a los modelos bioinspirados, con el fin de conocer su estado actual. Por otra parte el método Analítico-Sintético permite analizar la bibliografía y materiales relacionados con los modelos bioinspirados para utilizarlos en el desarrollo de la investigación. Como Método Empírico se emplea el método de Observación que permite obtener la información necesaria en todas las fases de la investigación, este método ofrece un gran acercamiento

a la realidad, a través del mismo se logrará encontrar la posible solución del problema, visto desde diferentes perspectivas. Es necesario observar los entornos virtuales para identificar en qué situación específica se puede aplicar un modelo bioinspirado.

El presente documento está estructurado de la siguiente manera:

Capítulo 1: Fundamentación Teórica. Se hace referencia a los elementos teóricos que constituyen la base de la investigación realizada. Se describen conceptos de IA, sus características y aplicaciones, enfatizando en los modelos bioinspirados.

Capítulo 2: Resultados de la Investigación. Se realiza un análisis sobre las mallas de navegación y se explican las principales características del modelo cognitivo propuesto. Además se explican las características necesarias para crear una aplicación basada en el comportamiento de las colonias de hormigas.

Capítulo 3: Análisis, Diseño e Implementación de la Aplicación. Se describe la aplicación propuesta, a partir de los diferentes modelos y diagramas presentes en los flujos de trabajo de Análisis, Diseño e Implementación.

Capítulo 1

Fundamentación Teórica

1.1. Entornos Virtuales

Los entornos virtuales tienen su origen en la necesidad militar de simular diversos ambientes y situaciones, en especial las simulaciones de vuelo. Con el tiempo esta tecnología se comenzó a comercializar para uso civil, principalmente como videojuegos. Estos entornos se desarrollaron gradualmente llegando a alcanzar una excelente calidad gráfica debido a la tecnología 3D, por una parte refuerza la metáfora del mundo virtual compartido y por otra, potencia las expectativas comportamentales de los objetos y los agentes que típicamente los habitan. En la ciencia de la computación los agentes creíbles o virtuales son aquellos que *simulan el comportamiento de humanos, animales u otros objetos en un entorno virtual* [Coca, 2009]. Estos agentes utilizan los modelos cognitivos para visualizar el entorno y poder obtener una representación de cómo está estructurado el mismo, para poder moverse dentro del entorno.

1.1.1. Modelos Cognitivos

Los modelos cognitivos se usan en la inteligencia artificial para representar el mundo virtual, cada parte del entorno presenta sus propias características y ofrece información, la cual es analizada para que los agentes del entorno puedan interactuar con el mismo. Para el caso de los agentes inteligentes se necesita de un modelo cognitivo para visualizar el mundo y po-

der tener una representación de cómo está estructurado el mismo para poder moverse dentro de él. En el entorno se incluyen detalles como, los obstáculos (puertas, ventanas, enemigos). Además almacena el estado de la información como las puertas cerradas, caminos bloqueados o el tipo de enemigo que se está moviendo. No siempre un modelo cognitivo es el óptimo a usarse en un entorno virtual, a veces es necesario realizar la representación usando varios de los modelos disponibles para lograr una representación adecuada para los agentes. Entre estos modelos están:

- **Mallas de Navegación**

  Uno de los modelos más usados para representar mundos virtuales son las mallas de navegación, las cuales son también son conocidas como NavMesh. Una malla de navegación es un grupo de polígonos convexos que describen la superficie del ambiente en tres dimensiones, además de que un polígono no se superpone a otro. Cada arista de los polígonos que conforman la malla tiene un punto de acceso hacia un polígono adyacente y cada una representa un punto de interés que se conecta a los polígonos adyacentes, este punto puede estar en el centro del polígono o de la arista. La convexidad garantiza el movimiento en línea recta dentro del polígono y de un polígono a otro. Haciendo uso de la programación dinámica se puede resolver el problema de la partición convexa en un tiempo óptimo de O (r2 n log n), donde n es el número de vértices, r puede ser visto como el número de aristas. Uno de los métodos más utilizados para la partición convexa es el método Herthel-Mehlhorn.

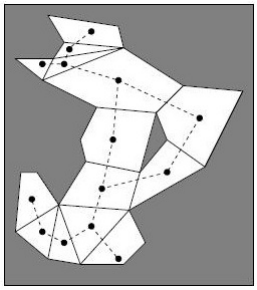

Figura 1.1: Puntos de acceso en el centro del polígono

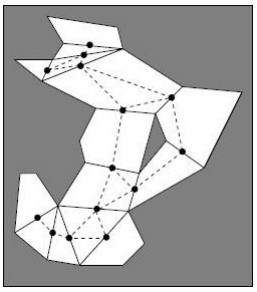

Figura 1.2: Puntos de acceso en el centro de la arista

Este es un simple, y altamente intuitivo modelo cognitivo donde los caracteres de la inteligencia artificial realizan un plan que pueden usar para la navegación y la búsqueda de caminos en el mundo. Además permite una óptima manipulación de terrenos interiores y exteriores. Este modelo cognitivo brinda un completo conocimiento sobre las zonas navegables, esta información se aprovecha para determinar el camino más corto y natural de todas las rutas posibles, así los agentes del juego son capaces de evitar obstáculos que bloquean su camino o incluso seleccionar una ruta alternativa. [Smed and Hakonen, 2006]

El tamaño de la malla no depende del tamaño del mundo, pero sí de las figuras geométricas, de las barreras o paredes y de la geometría del entorno. Adicionalmente como en el interior de un polígono convexo el agente puede moverse libremente, entonces es muy fácil de calcular el movimiento en batallas donde el agente inteligente tiene necesidad de moverse repentinamente en cualquier dirección para evitar una colisión o el fuego de un enemigo. Sin embargo la desventaja es que es muy difícil y tedioso encontrar todos los polígonos a través de los cuales se pueda mover o andar y los caminos para alcanzarlos desde polígonos vecinos. Algunas veces este proceso no es acertado en las propiedades físicas de los modelos y esto puede marcar algunos polígonos que pueden ser alcanzados cuando en realidad las leyes físicas impuestas por la simulación no lo permitan. Otro inconveniente de las mallas de navegación es que son difíciles de generar, las mismas pueden crecer en complejidad y provocar un uso de memoria más allá de lo recomendable.

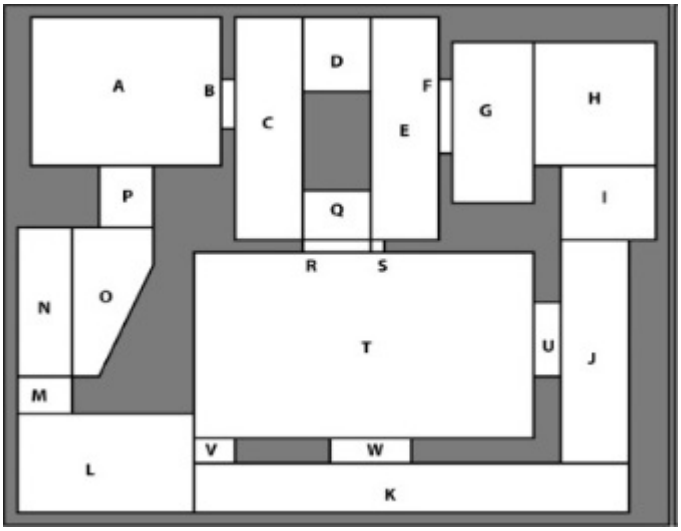

Figura 1.3: Representación de una Malla de navegación

Las mallas de navegación son utilizadas para la discretización del mundo virtual, esto consiste en la forma en que es convertida la geometría del mapa de juego y la capacidad de movimiento de los agentes dentro del mismo a nodos de grafo de navegación, así como la reducción de las rutas de acceso a las conexiones entre ellos. De esta forma, se transforma el problema de la búsqueda de camino en el de encontrar un camino dentro del grafo. Algunos ejemplos donde se utilizan son: juegos de estrategia y simuladores de multitudes.

Existen dos tipos de mallas de navegación, las basada en triángulos, que como su nombre lo indica requiere que todos los polígonos de la malla sean triángulos; y las basada en polígonos de N lados, las cuales permiten usar polígonos de cualquier cantidad de lados siempre que se mantenga la convexidad, estas a su vez pueden representar un espacio de búsqueda más simple que el que puede ser representado con triángulos, permitiendo una búsqueda de camino más rápida. [Autores, 2004]

- **Puntos de Visibilidad**

  Dentro de la informática actual la teoría de grafos juega un papel de singular importancia. En nuestro campo se le conoce como un conjunto de vértices o nodos interconectados

mediante aristas o arcos. De ahí que sea ampliamente usado para representar redes de comunicación. Algunos de los ejemplos son los entornos de ciudades y puntos de visibilidad. [García and Gómez, 2009]

Los grafos de navegación de puntos de visibilidad son creados por la inserción de puntos en lugares importantes del mundo virtual, tal que cada nodo del grafo tiene una línea de observación que lo conecta con los nodos que están visibles desde ese nodo. Cada punto describe un movimiento válido o un punto de acceso a otros lugares dentro del entorno [Millington, 2006]. Estos puntos se suelen colocar de forma manual y describen el terreno accesible a los agentes que interactúan en el entorno.

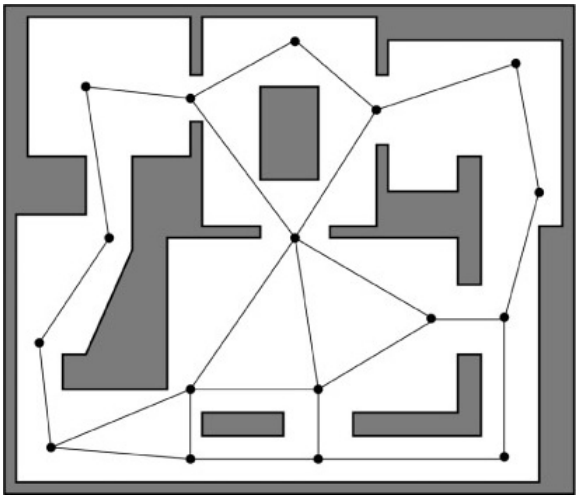

Figura 1.4: Puntos de Visibilidad

En la Inteligencia Artificial los puntos de visibilidad son muy utilizados debido a que son fáciles de crear y usar. Además evita los problemas de colisiones con las paredes y en grafos poco complejos se encuentra rápidamente el camino óptimo. En grafos con gran cantidad de nodos los algoritmos para recorrerlo demoran en realizar los cálculos, para buscar el camino más corto entre dos nodo. Pero la principal desventaja de este modelo es que no entregan información acerca de los obstáculos que se encuentran a su alrededor o de la posibilidad de usar caminos alternos donde existan obstáculos que se puedan mover o desplazar.

Cuando se colocan los puntos de visibilidad se deben evitar crear puntos ciegos. Estos no son más que áreas del entorno que quedan fuera del rango de los nodos del grafo de búsqueda. Aunque en ocasiones se insertan los puntos en áreas ciegas dentro del grafo de búsqueda con el fin de lograr áreas específicas donde se puedan esconder los jugadores de los agentes, en el caso de los juegos de estrategia.

- **Rejilla Regular**

  Una rejilla regular no es más que un mosaico de polígonos. El centro de cada polígono representa un punto de interés, y sus polígonos adyacentes forman las posibles conexiones con otros puntos de interés. Las rejillas generalmente soportan consultas de acceso aleatorio, ya que cada polígono debe ser accesible en cualquier instante de tiempo. [Smed and Hakonen, 2006]

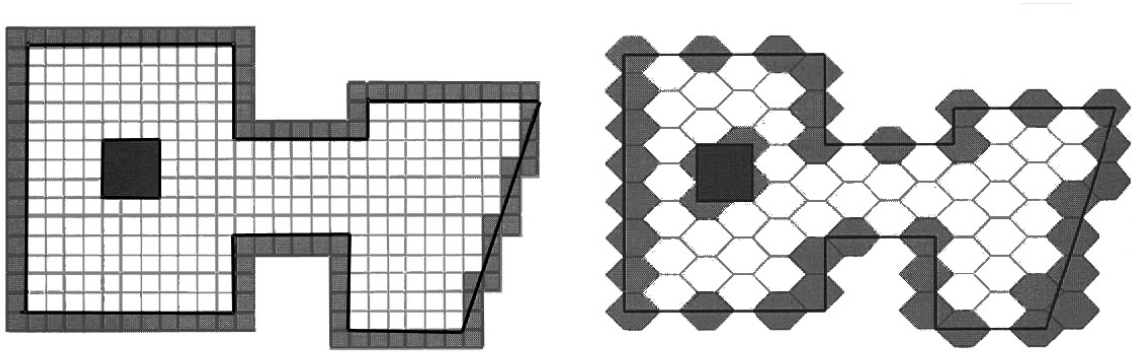

Figura 1.5: Rejillas Regulares

Este modelo cognitivo presenta un significativo inconveniente: la rejilla regular 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. Existen tres tipos de representaciones: Rejilla cuadriculada, rejilla triangulada y rejilla hexagonal.

1.1.2. Algoritmos de Búsqueda de Caminos

Generalmente para realizar la búsqueda de caminos se obtiene un grafo del entorno donde se desea calcular el camino de un nodo a otro. A cada arista se le asigna un peso que representará el costo de trasladarse de un nodo a otro, por tanto el camino con menor costo será el camino mínimo entre esos dos puntos. Para la búsqueda de camino los algoritmos más utilizados son[Duch and Tejedor, 2009]:

- **Recorrido en profundidad**

  El recorrido en profundidad es el algoritmo más sencillo y consiste en visitar todos los nodos posibles a partir del inicial. En este caso se tiene una lista de nodos, cuando se comprueba un nodo se introducen en la lista sus vecinos. Para evitar ciclos se crea una lista de nodos visitados a la cual se agregarán los nodos comprobados. Una búsqueda en profundidad significa que se visitarán primero los nodos a los que se puede llegar desde un determinado nodo, antes que los nodos que están en la lista por visitar.

- **Recorrido en amplitud**

  En el recorrido en amplitud la prioridad de búsqueda a partir de un nodo del árbol, son primero sus vecinos y después sus hijos. Este algoritmo se implementa de forma similar al anterior con la particularidad de que al agregar los nodos adyacentes a la lista de nodos por visitar, estos se agregan al inicio y no al final. En el caso de que los pesos de las aristas no varíen, este algoritmo asegura encontrar el camino mínimo.

- **Dijkstra**

  Este algoritmo fue diseñado en 1959 por el científico holandés Edsger Dijkstra, este consiste en generar un árbol en el que la raíz es el vértice de inicio y se ramifica por todos los nodos del grafo sin crear ciclos. Este algoritmo asegura que, utilizando estas ramificaciones, el camino entre un vértice y cualquier otro sea mínimo. Dijkstra es muy costoso debido a que calcula todos los caminos posibles a partir de un nodo. Este algoritmo se utiliza para calcular los posibles caminos que al inicio del juego se deben seguir en el

entorno. A partir de aquí, un algoritmo que planifique determinados movimientos puede utilizar esta información sin tener que volver a calcularla. Por el contrario, si el entorno se modifica, se debe calcular la nueva información de los caminos; lo cual generaría bastante coste de proceso.

- **A***

  El algoritmo A* está basado en los algoritmos de búsqueda en profundidad y amplitud, la diferencia radica en que este algoritmo deduce en cada momento qué nodo se debe visitar mediante una función heurística. Este algoritmo cuenta con una lista de nodos visitados y los nodos a los cuales se conoce cómo llegar partiendo de los nodos visitados. Mediante la función heurística se elige el nodo que posee menor costo y el más probable en teoría. Si a medida que se avanza por un camino, éste se vuelve más costoso que sus alternativas, el algoritmo de selección excluye el camino y continuará por otro nuevo y más prometedor.

1.2. Modelos Bioinspirados

Los Modelos Bioinspirados son aquellos creados a partir del comportamiento y la capacidad de los organismos naturales ante diversos problemas, teniendo en cuenta además las habilidades de adaptación y aprendizaje del medio en la medida en que va evolucionando. Las ventajas de estos modelos radican en que son dinámicos, flexibles, robustos, autoorganizados, compuestos por elementos simples y descentralizados. En la esfera de Inteligencia Artificial estos modelos son muy utilizados para algunos métodos de solución de problemas como son:

- Redes Neuronales
- Algoritmo Genético
- Bandada de Pájaros
- Colonia de Hormigas

- Membranas

1.2.1. Redes Neuronales Artificiales

Una Red Neuronal Artificial (RNA) es un modelo computacional que trata de simular el funcionamiento del cerebro humano, se basan principalmente en la arquitectura del mismo pero no llega a desarrollar una réplica del órgano. Básicamente, entendemos una red neuronal como un sistema que recibe un conjunto de entradas (percepciones que recibimos del sistema), las procesa por medio de un conjunto de elementos ocultos que se encuentran conectados entre sí (en un sistema parecido a las conexiones de las neuronas) y que produce un resultado de salida (la acción que vamos a tomar).

Las RNA están compuestas por neuronas (elementos computacionales simples) y por arcos dirigidos y ponderados que conectan las neuronas, el peso de cada arco indica la significación de la información que llega a través del mismo. A partir de los pesos las neuronas alcanzan un nivel de activación, es decir, la respuesta a una entrada determinada. Cada neurona individual es capaz de realizar procesamientos muy pequeños, al estar conectada con otras miles, puede trabajar en paralelo y alcanzar una actividad global de procesamiento enorme. [Martín and Sanz, 2001]

Las redes neuronales poseen tres elementos esenciales:

- Topología de la red: Es la forma en que están organizadas las neuronas. En la Tesis de Grado *Aplicaciones de las Redes Neuronales en entornos virtuales*. Se plantea que la red neuronal más usada para resolver varios problemas es la red perceptrón multicapa. [Prevot and Herrera, 2008]

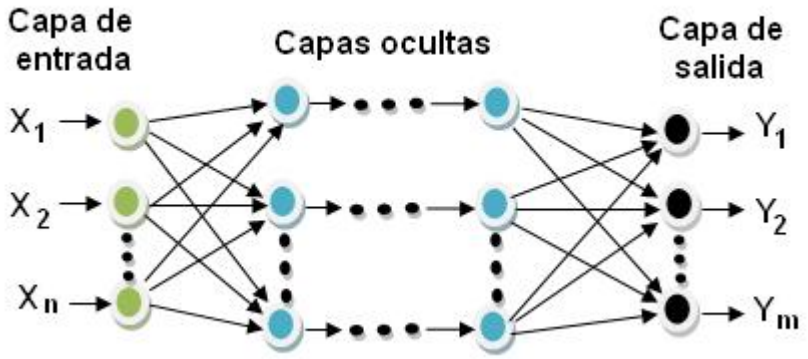

Figura 1.6: Representación de una Red Neuronal Multicapa

- Modelo de Neurona [Bello, 2002]:

  Modelo Lineal. La activación de la neurona es igual a la entrada total.

  Modelo lineal con umbral. La activación de la neurona toma un valor binario en dependencia del signo de la entrada total.

  Modelo estocástico. La respuesta de la neurona es probabilística basada en la entrada total.

  Modelo continuo. Usado junto a una red multicapa para resolver problemas de clasificación no lineales como son la mayoría de los problemas a los cuales nos enfrentamos a diario.

- Algoritmo de Aprendizaje: El aprendizaje de las neuronas artificiales se traduce en encontrar el peso del enlace entre las mismas. Se conoce que las neuronas se autoorganizan, aprenden del entorno y se adaptan a él, comportamiento necesario para su supervivencia y desarrollo. Una manera de entrenar a las RNA es haciendo uso de otro de los modelos bioinspirado como los Algoritmos Genéticos. [Rabuñal and Dorado, 2005]

Las redes neuronales son aplicadas a un número creciente de problemas reales del mundo. Su ventaja primaria es que ellos pueden resolver a menudo problemas que son demasiado complejo para las tecnologías convencionales. Estos problemas incluyen reconocimiento

de patrones y predicción, lo que requiere el reconocimiento de las tendencias en los datos. [van Waveren, 2001]

Las RNA son muy usadas para implementarlas en los agentes inteligentes, principalmente para los agentes de los videojuegos. Uno de los principales usos de una red neuronal es para implementar el sistema de decisión del comportamiento de los agentes inteligentes de una manera más rápida y natural.

En el libro *Inteligencia Artificial* de Duch y Navarro proponen que en el caso particular del desarrollo de videojuegos, las redes neuronales ofrecen algunas ventajas respecto a otras técnicas de Inteligencia Artificial tradicional:

- Permite simplificar la programación de máquinas de estado o sistemas de reglas complejos para decidir el comportamiento del juego.
- Ofrecen la posibilidad de que la IA de los agentes se adapte a medida que se desarrolla el juego, aprendiendo de las acciones que va tomando el agente, y por tanto adaptando la dificultad del juego a las acciones del usuario. [Duch and Tejedor, 2009]

En el libro de varios autores *Inteligencia Artificial Aplicada al Análisis de Sistemas Energéticos con Matlab* se expone que para muchas aplicaciones la respuesta de una RNA, es frecuentemente más acertada que las generadas por modelos ajustados o por modelos teóricos simplificados ya que se fundamentan en la experiencia de la interacción, sin preconcebir ideas, ni la formulación y el conocimiento previo de complicadas ecuaciones. En teoría y en muchos casos prácticos, una red bien entrenada produce resultados que pueden considerarse correctos. [Armas et al., 2008] La aplicación de los agentes autónomos y las redes neuronales logrará crear agentes inteligentes que interactúen dinámicamente con el terreno en el que se encuentren. Los agentes inteligentes son creados y colocados en los diferentes escenarios para que el jugador se encuentre con ellos [Graham, 2006]. Aplicando redes neuronales se podrán crear agentes inteligentes que interactúen entre sí para lograr un mismo fin ya que este es uno de los aspectos básicos de los agentes.

Ejemplo de la utilización de una RNA se encuentra en el libro *AI for Game Developers* [Bourg and Seeman, 2004], donde se explica cómo implementar un Perceptrón Multicapa en

cualquier escenario virtual de estrategias. Dicha implementación se realizó con el objetivo de monitorear y controlar el comportamiento de una criatura en un juego, la red neuronal se empleó con la idea de optimizar el proceso de toma de decisiones de la criatura.

En el libro *AI Game Programming Wisdom 2* [Champandard, 2004]] aparece explicado otro ejemplo de la utilización de las redes neuronales. En este caso se utilizó una RNA para generar una decisión de un mapa de influencia. Los mapas de influencia permiten realizar una valoración del juego y tomar decisiones basadas en el estado actual del mismo. Al utilizar las RNA se obtuvo la capacidad para dar la apariencia de un entorno más humano y realista.

Debido a las ventajas antes presentadas de las RNA se aplican en distintos tipos de videojuegos por ejemplo para implementar el sistema de decisión del comportamiento de los agentes inteligentes de una manera más rápida y natural muy usado en juegos de autos, también pueden ayudarnos a decidir el nivel de peligro de una situación en juegos de estrategia y uno de los mejores usos de las redes neuronales es el de implementar la capacidad de anticipar las acciones de los jugadores muy útil en los juegos de combate.

1.2.2. Algoritmo Genético

Este modelo está inspirado en la teoría de la evolución de las especies de Darwin. Los principales componentes que posee este modelo bioinspirado planteados en la Tesis de Grado *Incorporación de comportamientos a elementos que se mueven sobre grafos de camino* son: los individuos o cromosomas que no son más que la solución potencial de un problema, la población es el conjunto de cromosomas que representa el conjunto de soluciones posibles del problema, una función de aptitud o calidad del individuo que busca cuál de los individuos es mejor, los operadores genéticos que permiten crear nuevas poblaciones con mejores cualidades a partir de una existente y los parámetros del algoritmo. [García and Gómez, 2009]

En el libro *Biologically Inspired Artificial Intelligence for Computer Games* se encuentran varios tipos de operadores genéticos [Charles et al., 2008]:

- Operador de Selección o Darwiniano: determina qué individuos de una población se van a utilizar como padres para la siguiente. Existe una variedad de estrategias para selec-

cionar qué miembros de la población deben reproducirse. A cada individuo se da una oportunidad para reproducirse, esta es proporcional a su aptitud. Un individuo con buena salud debe obtener más oportunidades para reproducirse que un individuo que es incapaz. Por consiguiente un individuo más saludable sobrevive mucho más tiempo por lo que tendrá más oportunidad para reproducirse. Esto es similar al proceso de reproducción de la vida real. Para realizar la selección es calculada la aptitud de la población en general, luego la probabilidad de que un individuo sea seleccionado para la reproducción va a ser igual a la división de su aptitud por de la aptitud de la población.

- Operador de Cruzamiento o Mendeliano: se van a tomar dos individuos de una población y se mezcla su código genético para obtener un nuevo individuo. Por ejemplo, si existen dos padres, PADRE A 01010101 y PADRE B 11110000, en el punto entre el bit 2 y el bit 3 está el punto de cruzamiento entonces se obtienen dos nuevos cromosomas, HIJO A 11010101 y el HIJO B 01110000, los dos primeros bits del HIJO A pertenecen al PADRE B y en el HIJO B se notan en los últimos 6 bits. Se puede observar que el bit 4 no cambió en ninguno de los hijos, este es el mismo que en ambos padres.

- Operador de Mutación: se toma un individuo de una población y se le cambian algunos valores de sus genes y se genera el nuevo individuo. Si solo se utiliza el proceso de cruzamiento en una población, quedarán muchas regiones del espacio de soluciones sin explorar, por lo que se podría estar perdiendo un potencial alto en la población original. Para que esto no ocurra se utiliza la mutación, que no es más que cambiar algunos valores en los bits o solamente cambiar la posición de dos de ellos.

Entre los parámetros del Algoritmo Genético se pueden encontrar: el tamaño de la población, número de generaciones, probabilidad de cruce y mutación.

Un Algoritmo Genético posee el siguiente funcionamiento; primero se genera de forma aleatoria o construida de acuerdo al objetivo, la población inicial, luego se realiza una evaluación de los individuos de acuerdo a la función objetivo, a partir de esto se genera un proceso iterativo usando un criterio de parada para la optimización del objetivo ya sea por el número de generaciones que se desea determinar, o porque se ha encontrado la calidad deseada. Se

realiza la selección, el cruzamiento, la mutación, luego se actualiza la nueva población y se comienza nuevamente el proceso hasta encontrar los resultados deseados. [Reyes, 2008]

En el libro *AI for Game Developers* [Bourg and Seeman, 2004], Bourg y Seeman plantean que se pude realizar una implementación de un algoritmo genético en cuatro pasos esenciales.

- Creación de la primera generación con sus rasgos.
- Proceso de evaluación de la aptitud de cada uno de los individuos de la población.
- elección de los miembros más aptos para crear la nueva generación.
- Combinar los miembros más aptos para obtener un individuo con mejores características, es decir esta etapa es la de evolución.

Después de la cuarta etapa se vuelve al segundo paso, estas operaciones se realizaran hasta encontrar los individuos que cumplan con las características deseadas o se llegue a la creación de una determinada generación.

Según Natyhelem Gil Londoño en su libro *Algoritmos Genéticos* [Londoño, 2001] lo que aventaja a los Algoritmos Genéticos frente a otros algoritmos tradicionales de búsqueda es que se diferencian de estos en los siguientes aspectos:

- Trabajan con una codificación de un conjunto de parámetros, no con los parámetros mismos.
- Trabajan con un conjunto de puntos, no con un único punto y su entorno (su técnica de búsqueda es global.)
- No necesitan conocimientos específicos sobre el problema a resolver; es decir, no están sujetos a restricciones.
- Utilizan operadores probabilísticos, en vez de los típicos operadores determinísticos de las técnicas tradicionales.
- Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivas en paralelo.

Actualmente, la computación evolutiva es un campo que ha tenido grandes avances demostrando claramente su capacidad y su potencial, los algoritmos genéticos están ayudando en la resolución de problemas de interés cotidiano, en áreas de estudio tan diversas como la predicción en la bolsa y la planificación de la cartera de valores, ingeniería aeroespacial, diseño de microchips, bioquímica y biología molecular.

Los Algoritmos Genéticos se han utilizado para la resolución de diversos problemas como los de optimización, entre los que se encuentran el problema del viajante y la calidad de sonido. También para la programación automática en los autómatas celulares y las redes de clasificación. Además para muchas aplicaciones de aprendizaje automático, así como para diseñar redes neuronales. Son usados también para el estudio de los aspectos evolutivos de los sistemas sociales, como el comportamiento de las hormigas. [Parra and Marrero, 2007]

La evolución proporciona un mecanismo para que los agentes inteligentes se adapten mejor al entorno que los rodea, seleccionando aquellos agentes que se desenvuelven mejor para su posterior reproducción. En los videojuegos, los Algoritmos Genéticos se suelen utilizar cuando existe bastante incertidumbre sobre las características del jugador. En este caso, es muy difícil diseñar los adversarios para que sean capaces de encontrarse al mismo nivel que cualquier jugador. Los enemigos deben ser capaces de proporcionar un reto que sea difícil independientemente de las combinaciones posibles de atributos. Una posibilidad es modelar los comportamientos de los enemigos mediante el uso de un Algoritmo Genético, de modo que aquellos comportamientos que se adapten mejor a los atributos del jugador serán los que sobrevivirán, mientras que los comportamientos que resulten inútiles desaparecerán. [Duch and Tejedor, 2009]

Un Algoritmo Genético puede usarse en una colección de agentes donde cada agente es ligeramente diferente. También puede usarse para perfeccionar subsistemas usados para la inteligencia artificial de un agente. Para el juego Quake III Arena se usa un algoritmo genético para perfeccionar la lógica difusa para los propósitos específicos. [van Waveren, 2001]

En la Tesis de Grado *Propuesta de técnicas de aprendizaje de elementos virtuales*, los autores plantean, que aunque los Algoritmos Genéticos se han utilizado mucho en los juegos de

rol, también han sido utilizados en otros juegos como son: Damas, Ajedrez, Lemmings: un juego de estrategia en el que el jugador debe salvar a los lemmings que intentan suicidarse, para la implementación de cada nivel fueron utilizados Algoritmos Genéticos para hacer cada nivel más difícil que el anterior, otro juego donde se utilizó este modelo fue el Lunar Lander Games el cual tiene como objetivo que una nave espacial aterrice en una plataforma de aterrizaje. [Martínez and Antúnez, 2008]

1.2.3. Bandada de pájaros

La técnica de los algoritmos de cúmulos de partículas o Particle Swarm Optimization (PSO) fue propuesta por primera vez por Kennedy y Eberhart en su libro *Swarm Intelligence* [Kennedy and Eberhart, 2001]. PSO es un algoritmo evolutivo poblacional inspirado en el comportamiento social de las bandadas de pájaros y los bancos de peces. La optimización por nubes de partículas es una técnica que une conceptos de inteligencia artificial y computación evolutiva. Se basa en los patrones de comportamiento de los animales sociales (enjambres de abejas, bandadas de pájaros, bancos de peces etc.) y en particular en la hipótesis según la cual compartir información entre miembros de una misma especie proporciona una ventaja evolutiva.

El funcionamiento básico del PSO simula el comportamiento del vuelo de las bandadas de aves en busca de comida. La estrategia lógica a utilizar es seguir al ave que está más cerca de la comida. La bandada de pájaros se va a mover por el espacio de soluciones hasta encontrar la solución al problema, para ello cada pájaro va a ajustar su trayectoria en base a dos aspectos fundamentales: su mejor posición conocida y la mejor posición de la bandada en su conjunto. Para esto es necesaria una función de calidad que permite determinar la calidad de la nueva posición obtenida por el pájaro, siendo la interacción del individuo con el resto de la bandada lo que decide cual va a ser la nueva posición de cada elemento. [Pfeil, 2006]

Principios básicos de este algoritmo:

- Cada una de las partículas tiene una posición y velocidad en el espacio de búsqueda.
- Cada partícula posee una medida de calidad.

- Cada partícula o individuo puede interactuar con un grupo de vecinos.
- Cada uno de los individuos aprende ajustando su posición y velocidad. Parcialmente atraído a su mejor posición hasta el momento y a la mejor posición de la bandada.

Parámetros:

- Número de partículas (tamaño de la bandada).
- Número de generaciones.
- Peso de la inercia (valor que afecta la velocidad, los cambios de velocidades no pueden ser bruscos).
- Razón de aprendizaje cognitivo (importancia de la mejor posición alcanzada por cada pájaro en su dirección actual).
- Razón de aprendizaje social (importancia del mejor estado encontrado por cualquier individuo de la bandada en la posición de un pájaro en particular).

El modelo de bandadas de pájaros y otros movimientos grupales son complejos comportamientos (conocidos como *Steering Behaviors*) que se han descompuestos en tres más simples como niveles individuales. Los *Steering Behaviors* no son más que reglas de movimiento a seguir por los agentes autónomos a la hora de tomar decisiones ante los diferentes estímulos que ejercen sobre ellos los demás componentes de su entorno. Basados en las bandadas de pájaros se tomaron la separación, la alineación y la cohesión. [Green, 2000]

Entre las ventajas que presenta este modelo están:

- El algoritmo de PSO puede ser muy eficaz para resolver los problemas de optimización globales.
- Aplica conceptos de interacción social a la resolución de problemas de búsqueda/optimización.
- Resuelve problemas similares a los resueltos por algoritmos de Computación Evolutiva.

Un ejemplo de la utilización de este modelo bioinspirado es la *Swarm Robotic* que no es más que el estudio de sistemas robóticos constituidos por un grupo de robots que interactúan y cooperan para cumplir un grupo de tareas. Una de las investigaciones más interesante proyectada en este campo han sido el proyecto de *Swarmbots*. Estos sistemas son desarrollados por la NASA para el apoyo a las misiones de exploración de varios astros. [Hinchey and Otros, 2006]

En los gráficos por computadoras este modelo también se ha utilizado ampliamente en las simulaciones. Jonas Pfeil plantea en *Swarm Intelligence* que para la trilogía del Señor de Los Anillos fue desarrollado un sistema con miles de agentes animados para las gigantescas escenas de batallas. Cada agente tenía un comportamiento inteligente individual, pudiendo seleccionar de una variada colección de movimientos el más adecuado para responder dependiendo del estímulo exterior. Masivamente se han usado desde entonces para muchos proyectos incluso películas y anuncios. Aparte de estas simulaciones realistas, los proyectos de arte como SwarmArt usan las simulaciones de grupos sólo debido a su belleza. [Pfeil, 2006]

Actualmente son usados los grupos de agente como una técnica de modelado por computadora y también se han aplicado como una herramienta para estudiar los sistemas complejos. Los ejemplos de simulaciones que se han emprendido incluyen bandadas de pájaros, así como el negocio y economía, y los sistemas ecológicos. También está investigándose para el uso en aplicaciones como transmisión de señales de televisión, telefonía, enrutamiento de red, categorización de datos y optimización en la búsqueda del camino más corto.

1.2.4. Colonias de Hormigas

El modelo de las colonias de hormigas o *Ant Colony Optimization* (ACO), está basado en el comportamiento de las colonias de hormigas naturales, este modelo no está basado en la inteligencia de un solo individuo sino que se basa en la inteligencia colectiva, esta inteligencia colectiva emerge de la interacción entre los individuos. Este modelo se caracteriza por su simplicidad y robustez.

Existen varios algoritmos de ACO los cuales se clasifican como algoritmos constructivos

ya que en cada iteración del algoritmo cada hormiga construye una solución al problema recorriendo un grafo en construcción. Cada arco o arista del grafo representa los posibles caminos que la hormiga puede elegir, estas aristas tienen asociadas dos tipos de información que guían el movimiento de la hormiga, estos son:

- Información heurística: es la probabilidad de moverse desde el nodo *i* al nodo *j* es decir recorrer la arista a_{ij}. Esta información se denota como n_{ij} y la misma no es cambiada por las hormigas mediante el proceso de ejecución del algoritmo.
- Información de los rastros de feromonas artificiales: es la que mide deseabilidad aprendida del movimiento de i_{aj}, la que imita a la feromona real que dejan las hormigas reales en su camino. Esta información se denota como t_{ij} y es cambiada por las hormigas mediante el proceso de ejecución del algoritmo dependiendo de las soluciones encontradas.

A lo largo del tiempo, el rastro de feromonas se va evaporando reduciendo así la atracción de las hormigas. Mientras un camino sea más largo, más tiempo tardarán las hormigas que tomen ese camino y más rastro de feromona se evaporará. Por tanto el camino más corto tendrá mayor concentración de feromonas. La información recolectada a lo largo del proceso de búsqueda se actualiza en lo que se llama la matriz de feromonas. Esta información consiste en rastros de feromonas asociados a los arcos del grafo. Ante cada toma de decisión, una hormiga utiliza la matriz de feromonas tomando la información propia, la del nodo en el que se encuentra y la de los arcos adyacentes (feromonas). Para cada arco transitado por una hormiga, esta deja es rastro (arco) registrado en la matriz. [Nieto, 2007]

En resumen la base de las Colonias de Hormigas radica en que dichos animales se trasladan de la casa a la comida usando mayoritariamente el camino mínimo para trasladarse de un punto a otro. Cada una de las hormigas va a tratar de resolver el problema y transmitirle información a las demás del grupo para juntas dar la respuesta óptima. Para modelar esto se utiliza un grafo y cada nodo va a ser un estado donde se encuentre la hormiga, los nodos van a estar conectados por arcos cuyos pesos van a ser el nivel de feromona de las hormigas, el principal objetivo del algoritmo radica en guiar la hormiga en el proceso de recorrer los nodos y la mayor dificultad se localiza en decidir cuál será el próximo nodo a visitar para lo cual se va

a tener en cuenta la regla de que mientras mayor sea el peso del arco mayor será el número de hormigas que han pasado por ella, lo que significa que la probabilidad de que se tome este camino para llegar al objetivo sea mayor.

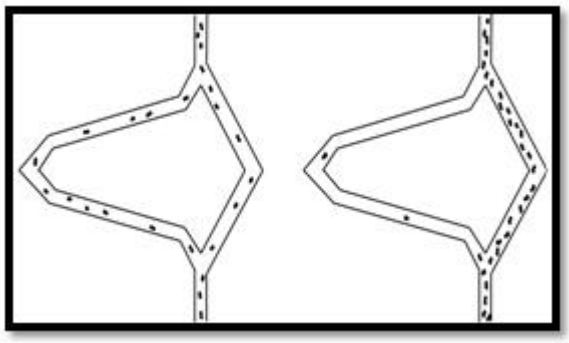

Figura 1.7: Colonia de Hormigas

El algoritmo que se implementa sigue los siguientes pasos:

- Se inicializa el peso de cada arco o el nivel de feromona del mismo.
- Realizar un ciclo, en cada iteración del ciclo todas las hormigas van a encontrar una solución.
- Las soluciones encontradas por cada hormiga se evalúan y se actualiza el nivel de feromona a cada arco de la solución, resultando los arcos con mayor cantidad de feromonas como mejor solución al problema.
- Este algoritmo se repite hasta tanto no se encuentre la condición de parada, es decir, cuando se termine un ciclo o hasta que se encuentre un elemento con la calidad óptima en dependencia del problema.

Las aplicaciones de este modelo se basan en problemas de optimización discretos. Entre ellas se encuentran el enrutamiento en redes, problemas clásicos de enrutamiento de vehículos, de ordenación secuencial, de secuenciación, y de coloreo de grafos.

José Manuel García Nieto plantea que estos algoritmos se aplican con éxito a problemas de optimización combinatoria, donde la dimensión del espacio completo de soluciones es ex-

ponencial con respecto a la dimensión de la representación del problema (NP-duros). Debido a su propia naturaleza poblacional son fáciles de paralelizar. Se ha demostrado que esta metaheurística es altamente competitiva en problemas académicos complejos como: problema del viajero o TSP, problema de asignación cuadrática, ordenación secuencial, problemas de planificación de tareas, entre otros. [Nieto, 2007]

Varios algoritmos de Colonias de Hormigas han sido desarrollado y aplicados satisfactoriamente a problemas tales como el problema del viajero, rutas de redes, y planificación u organización de recursos. Para estos casos es necesaria la búsqueda del camino más corto, se han realizado varios experimentos con distancias iguales y diferentes entre el punto de partida de las hormigas y el objetivo. Un ejemplo de lo antes mencionado: [Charles et al., 2008]

Considerando que hay una Colonia de Hormigas, una fuente de alimentos y dos caminos de igual distancia hacia la comida. Al inicio las hormigas se irán por los dos caminos, luego la mayoría toma uno de los caminos, como el nivel de feromonas aumenta en dicho camino poco a poco las hormigas se irán pasando para el camino con mayor nivel de feromonas. Por otra parte considerando que las distancias de estos caminos fueran diferentes inicialmente las hormigas irán por ambos caminos, pero debido a que la distancia de uno de ellos es mayor que la del otro, en el camino más largo las feromonas se irán evaporando y por esta razón las hormigas irán eventualmente tomando el camino más corto hacia el objetivo. Simulando colonias de hormigas con el algoritmo *Simple Ant Colony Optimisation* o S- ACO: Inicialmente se considera que hormigas artificiales se mueven por una estructura de grafos, en el ejemplo de 3 nodos el 1 representa el nido y el 3 la comida. Los 3 arcos que conectan los nodos tienen la misma longitud.

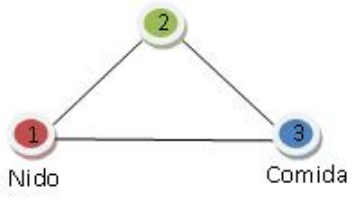

Figura 1.8: Grafo para ACO

Cada arco conecta dos nodos *i* y *j*, el nivel de feromona del arco t_{ij}, el cual es inicializado con un valor positivo pequeño, *a*, que es el mismo para todos los arcos. Cuando una hormiga *k*, está en el nodo *i*, la probabilidad de que seleccione el arco que conecta con el nodo *j* es expresada como el nivel de feromona en el arco a_{ij}, como una proporción de la suma de las cantidades de feromonas en todos los arcos en el vecindario de la hormiga. Este vecindario es el juego de nodos que una hormiga, *k*, puede escoger como el próximo nodo para viajar, y se representa por *N*. El cálculo de probabilidad se resume en la ecuación:

$$P_{ij}^{k} = \frac{t_{ij}}{\sum_{l \in N_i^k} t_{il}}, si\ j \in N_i^k$$

En el libro *Biologically Inspired Artificial Intelligence for Computer Games* se explica cómo pueden encontrarse los caminos más cortos en las estructuras del grafo por el algoritmo de S-ACO, y esto puede aplicarse a búsqueda de caminos en los juegos de computadora. Este método se aplicó a un juego de combate obteniendo resultados exitosos a la hora de controlar el comportamiento de los agentes inteligentes del juego. Los resultados obtenidos muestran que el tanque puede moverse hacia el jugador contrario y adaptarse a los cambios en el ambiente, incluso cuando los movimientos del jugador humano son evasivos, y también puede retirarse cuando juzgue apropiado hacerlo. Claramente, el algoritmo de S-ACO ha tenido el éxito encontrando los caminos más cortos al antagonista en un ambiente dinámico. [Charles et al., 2008]

Los algoritmos de ACO se han aplicado a un gran número de problemas de optimización combinatoria. Las aplicaciones actuales de la ACO se distribuyen dentro de dos clases fundamentales. La primera clase de problemas está compuesta por los problemas de optimización combinatoria NP-duros, para los que las técnicas clásicas ofrecen a menudo un comportamiento pobre. Una característica común a casi todas las aplicaciones exitosas de la ACO es la combinación de las hormigas con algoritmos de búsqueda local que refinan las soluciones ofrecidas por las hormigas. La segunda clase de aplicaciones se compone de problemas dinámicos de caminos mínimos, donde la instancia del problema que hay que solucionar cambia durante la ejecución del algoritmo. [Reyes, 2008]

En su tesis *Uso del Sistema de la Colonia de Hormigas para Optimizar Circuitos Lógicos Combinatorios*, Mendoza propone aplicar los ACO para el diseño de circuitos lógicos, con el fin de optimizarlos de acuerdo a determinados parámetros como el número de compuertas. [Mendoza, 2001]

Entre otras aplicaciones que de la ACO se encuentran resoluciones para el problema de la asignación cuadrática y el problema de la secuenciación de tareas, en enrutamiento de redes y de vehículos, en problemas de ordenación secuencial, de secuenciación, y coloreo de grafos. Además, la ACO ha sido usada recientemente para aprendizaje automático, concretamente para el diseño de algoritmos de aprendizaje para estructuras de representación del conocimiento como las clásicas reglas lógicas, reglas difusas, y redes bayesianas, demostrando resultados bastante prometedores.

En la economía los ACO se aplican para buscar las rutas óptimas para reducir los gastos en cuanto a transporte, además para modelar la distribución de los diferentes departamentos de una planta productiva para determinar cuál es la mejor distribución para garantizar un flujo de trabajo constante y a un costo mínimo.

1.2.5. Membranas

La Membrana o Sistema P es un modelo basado en la arquitectura de las células biológicas, las reacciones químicas que ocurren en ellas y la capacidad que presentan las mismas para dividirse. Este concepto fue introducido en 1998 por el informático Gheorghe Paun, cuyo apellido es el origen de la letra P en Sistemas P. Las variaciones en el modelo del Sistema P condujeron a la formación de una rama de investigación conocida como Computación con Membranas. Aunque es inspirado por procesos biológicos, el interés de la investigación con respecto a Sistemas P se refiere principalmente a su uso como modelo de cómputo, más bien que para el modelado biológico, aunque esto también se está investigando.

Desde el punto de vista de la biología de sistemas, los sistemas de membranas proporcionan un enfoque basado en el modelado discreto para describir sistemas de reacciones biológicas compuestos de membranas interconectadas. Cada membrana delimita una región

espacial en la cual pueden ocurrir reacciones químicas. Dentro de una membrana, las partículas moleculares son representadas por un multiconjunto de objetos, mientras que mecanismos dedicados a la reescritura de términos ejecutan simultáneamente las reglas de reacciones asociadas a cada membrana. Además, reglas suplementarias pueden controlar el intercambio de objetos entre las membranas o bien modificar la estructura de las mismas. De ahí, que una de las principales ventajas de los sistemas de membranas es la capacidad de capturar aspectos que describen las dinámicas estructurales.

El modelo básico de una estructura de la membrana consiste en algunas membranas que son jerárquicamente incluidas en una membrana principal, llamada la membrana superficial. Las membranas delimitan regiones donde están los objetos. Los objetos evolucionan según la evolución dada por las reglas asociadas con las regiones. Una regla se aplica a los objetos en la región asociada y puede modificar los objetos, enviándolos fuera de la membrana actual o a una membrana interna, y también puede disolver la membrana. Cuando tal una acción tiene lugar, todos los objetos de la membrana disuelta permanecen libres en la membrana que contenía a la misma, pero las reglas de evolución de la membrana disuelta desaparecen. La membrana superficial nunca se disuelve. Las membranas son separadores y canales de comunicación, pero son los participantes pasivos al proceso, el funcionamiento completo del sistema se gobierna por las reglas de la evolución. [Sempere, 2007]

En resumen la Computación con Membranas es un área de las ciencias de la computación cuyo objetivo es abstraer ideas y modelos computacionales de la estructura y funcionamiento de las células vivas, así como también de la forma en que las células se organizan en tejidos o estructuras de orden superior. Se trata de sistemas de membranas, también llamados Sistemas P, los cuales constituyen modelos algebraicos paralelos y distribuidos que procesan multiconjuntos de objetos de manera localizada (las reglas de evolución y los objetos que evolucionan son encapsulados en compartimentos delimitados por membranas), en los que la comunicación entre los compartimentos y con el ambiente juega un papel esencial.

En el libro *P Systems with Active Membranes: Attacking NP Complete Problems* se plantea que en los Sistemas P se consideran: las reglas de evolución que son asociadas con los

objetos y membranas, mientras la comunicación a través de las membranas se realiza con la participación directa de las membranas; es más, las membranas no sólo pueden disolverse, también se pueden multiplicar por división. Una membrana elemental puede ser dividida por medio de una interacción con un objeto de esa membrana. Se supone que cada membrana tiene una polarización eléctrica, se definen tres posibles cargas: positivo, negativo, o neutral. Si en una membrana tenemos dos membranas internas inmediatas de polarizaciones opuestas, una positiva y una negativa, entonces esa membrana se puede dividir de tal manera que las dos membranas de carga opuesta estén separadas; se reproducen todas las membranas de carga neutra y todos los objetos y una copia de cada uno de ellos se introduce en cada una de las dos nuevas membranas. La membrana superficial nunca es dividida. [Paun, 1999]

Los algoritmos de fuerza bruta se han utilizado ampliamente en el diseño de soluciones para problemas NP (NP es el acrónimo en inglés de *nondeterministic polynomial time*, tiempo no determinista polinomial. Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista) en la Computación con Membrana. La idea principal de los algoritmos de fuerza bruta es codificar cada solución factible en una membrana. El número de candidatos a la solución es exponencial en el tamaño de la entrada, pero el proceso de codificación puede hacerse en un tiempo polinómico. Una vez generados todos estos candidatos, cada uno de ellos se prueba para verificar si representa una solución al problema o no. Esta fase de la comprobación es realizada simultáneamente en todas las membranas usando el paralelismo masivo inherente a la Computación con Membranas. Una vez terminada la fase de comprobación, el Sistema P detiene y envía una señal al usuario con la salida del proceso.

También se ha explorado la posibilidad de soluciones a los problemas NP con técnicas de Computación con Membranas, pero tomando las ideas de Inteligencia Artificial en lugar de usar los algoritmos de fuerza bruta. El peor caso de cualquier solución de un problema NP necesita una cantidad exponencial de recursos, pero no siempre estamos en el peor caso. La contribución de usar las estrategias de búsqueda de Inteligencia Artificial es esa, por término medio, el número de recursos para resolver varios casos de un problema NP disminuye con

respecto al número de recursos usado por la fuerza bruta.

Problema de las n Reinas

El problema de las n reinas se trata de un acertijo en el que se colocan n reinas en un tablero de ajedrez de nxn sin que se amenacen entre ellas. En el juego de ajedrez la reina amenaza a aquellas fichas que se encuentren en su misma fila, columna o diagonal. El problema de las n Reinas es una generalización del problema propuesto por el ajedrecista alemán Max Bezzel en 1848.

En *Membrane Computing Meets Artificial Intelligence: A Case Study* [Gutiérrez and Pérez, 2010], se plantea una solución para este problema utilizando un Sistema P, el primer paso es determinar los espacios de estados. Existen dos formulaciones básicas. Una formulación completa de estado, que empieza con N reinas en el tablero y se mueve a su alrededor y una formulación incremental, donde los operadores mejoran la descripción del estado, a partir del estado vacío y cada acción añade una reina al estado. Esta segunda formulación reduce el espacio de estados drásticamente, desde que una nueva reina es agregada a la descripción de un estado sólo puede ponerse en una casilla no prohibida.

- Estados: Arreglos de *k* reinas *(1 - k - n)*, una por columna en lo más a la izquierda de las *k* columnas.
- Transiciones *(x, y)*: El estado y es el estado *x* donde una nueva reina se agrega la columna mas a la izquierda vacía. Esta nueva reina no es atacada por ninguna otra.

La idea básica del Sistema P seria codificar la posición de una reina como un conjunto de cuatro objetos x_i, y_j, u_{i-j} y v_{i+j}, donde x_i representa una columna y y_j representa una fila *(1 - i, j - n)*. Los objetos u_{i-j} y v_{i+j} representan las diagonales ascendentes y descendentes respectivamente y sus subíndices son determinados por sus correspondientes columna y fila *i* y *j*.

Poner una reina en el tablero de ajedrez significa escoger una casilla, es decir, un juego $\{x_i, y_j, u_{i-j}, v_{i+j}\}$ entre los objetos elegibles y eliminarlos de la membrana correspondiente. La opción escogida se guarda y se compara con el estado final. Si el estado final se alcanza terminamos el proceso; si no hacemos *backtracing* y escogemos otra opción elegible.

La Figura 1.9 muestra una solución encontrada por un Sistema P para un problema 20 reinas. Casillas: (1-20, 2-1, 3-3, 4-5, 5-2, 6-4, 7-13, 8-10, 9-17, 10-15, 11-6, 12-19, 13-16, 14-18, 15-8, 16-12, 17-7, 18-9, 19-11, 20-14)

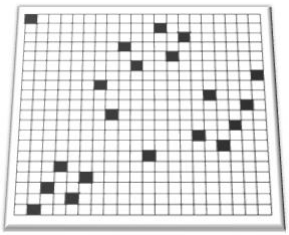

Figura 1.9: Solución para un problema 20 reinas

Capítulo 2

Resultados de la Investigación

2.1. Modelos Membrana y Colonia de Hormigas en sistemas de Realidad Virtual.

Como se ha podido observar los modelos bioinspirados han sido muy utilizados en diversos problemas presentes en los sistemas de realidad virtual. Las Redes Neuronales tienen infinidad de utilización desde el reconocimiento de patrones hasta el sistema de toma de decisiones de los agentes. Los Algoritmos Genéticos son muy utilizados para incrementar el nivel de complejidad de los videojuegos, incorporando la capacidad de adaptarse y evolucionar a los agentes creíbles. Por otra parte las bandadas de pájaros se aplican en simulaciones donde existen varios agentes que cooperan entre sí. De ahí que el presente trabajo se centre en dos modelos poco estudiados en nuestra universidad y que tienen gran potencialidad para continuar el desarrollo de los entornos virtuales.

Las membranas no tienen gran aplicación práctica en la actualidad, sus principales aplicaciones han estado en el marco de la resolución de los problemas NP y una solución concreta para el problema de las n reinas, aunque se ha estudiado la posibilidad de aplicarlo en áreas de la bioinformática. Este modelo representa un nicho abierto para futuras investigaciones y por tanto las características propias del mismo deben ser estudiadas a fondo ya que podrían representar nuevas ventajas cuando se aplique en sistemas de realidad virtual.

En los entornos virtuales las Colonias de Hormigas se utilizan principalmente en la búsqueda dinámica de caminos cortos, es decir donde el problema a resolver va cambiando a medida que se ejecuta el algoritmo. Esto se aplicaría a los agentes que se mueven por el modelo cognitivo del entorno permitiéndoles no solo encontrar la mejor ruta para ir de una zona a otra sino también para moverse en la región donde están. Una de sus grandes ventajas es que puede funcionar continuamente y adaptarse a los cambios en tiempo real. Pudiendo ser aplicado como algoritmo de búsqueda de camino en un entorno virtual, sin embargo se plantea que encontrar los parámetros adecuados a la hora de aplicar el algoritmo puede ser engorroso y al igual que la definición de una red neuronal requiere de un proceso de prueba y error que puede tardar, a raíz de lo cual se propone en el presente trabajo una aplicación de pruebas que ayude al programador a definir a partir de un grafo creado por el usuario, los parámetros que permitan ejecutar el algoritmo de la forma más eficiente posible.

2.2. Propuesta de modelo cognitivo utilizando Membranas: MC-Memb

El modelo Membrana presenta una estructura representada por un diagrama de Venn, por ejemplo en la siguiente estructura que contiene 8 membranas; las membranas 3, 5, 6, y 8 son elementales y la 1 es la membrana superficial.

$P = [1[2[5]5[6]6]2[3]3[4[7[8]8]7]4]1$

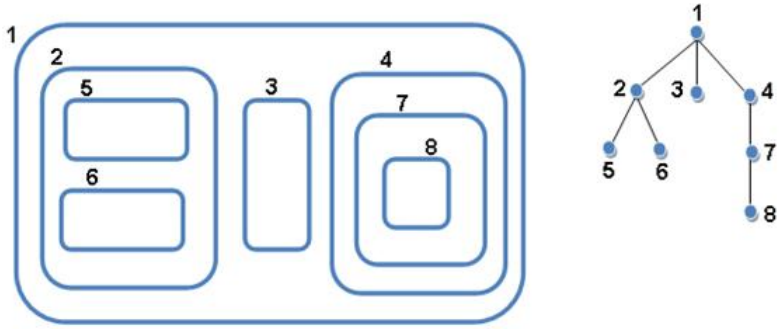

Figura 2.1: Estructura de una membrana y su árbol asociado

La idea de representar una descripción instantánea del mundo como un estado y un paso de un estado al siguiente como un borde en el gráfico es tan general que muchos problemas de la vida real pueden ser modelados como un problema de espacio de estados. La computación con membrana provee todos los ingredientes necesarios para encontrar una solución para cualquier problema representado como un espacio de estados y ser una herramienta útil para resolver muchos problemas de la vida real. Los principales ingredientes usados, los cuales son a su vez los más poderosos, son: los inhibidores, la cooperación, prioridades y disolución.

De una manera abstracta, la representación de un problema P = (α, S, E, F) como un espacio de estados consiste en, [Paun, 1999]:

- Un grupo de estados *S* y un estado inicial $\alpha \in S$.
- Un conjunto *E* de pares ordenados*(x,y)* llamados transiciones donde *x* y *y* son estados y de *x* a *y* se puede llegar en un paso.
- Un grupo de estados finales *F*.

Además se tienen los costos de cada transición *(x,y)*.

Para un problema P = (α, S, E, F) se considera un Sistema P

$\prod = (\Gamma, H, \mu, w_e, w_s, R_1, R_2, R_3, R_1 > R_2 > R_3)$ donde:

- Alfabeto $\Gamma = S \cup \{p_e, r_e | e \in E\}$
- Conjunto de etiquetas $H = \{U, S\}$
- Estructura de la membrana $\mu = [[]_u]_s$
- Conjunto inicial $w_u = \{a\}\ yw_s = \emptyset$
- Reglas
  - $R_1 = \{[x]_u \rightarrow \lambda : x \in F\}$ Para cada estado final tenemos una regla de disolución que disuelve la membrana *u*.
  - $R_2 = \{[x \neg p_y \rightarrow yr_{xy}]_u : (x, y) \in E\}$ Para cada transición *(x, y)*, *x* puede cambiarse por yr_{xy} si p_y no ocurre en la membrana *u*, es decir, p_y actúa como un inhibidor.

- $R_3 = \{[yr_{xy} \rightarrow xp_y]_u : (x, y) \in E\}$ Para cada transición *(x, y)* tenemos una regla cooperativa donde el conjunto r_{xy} se reescribe como el p_y de *x* en la membrana *u*.

A partir de las características fundamentales de las membranas sobre todo en la discretización del espacio se propone un nuevo modelo cognitivo el cual posee similitudes con el modelo cognitivo mallas de navegación. Este modelo al igual que las mallas de navegación utiliza las características de los polígonos convexos para representar áreas específicas del mundo virtual, donde los agentes pueden desplazarse libremente dentro de estos polígonos.

El modelo está compuesto por una membrana principal, la cual contiene el resto de las membranas o polígonos que conforman el mundo. Basado en la estructura multicelular cada membrana interior posee una o varias conexiones con las membranas adyacentes, semejante a los canales de proteínas de las células biológicas; a través de estas conexiones o juntas es por donde se podrá desplazar un agente de una región a otra. Esto facilitará la navegación de los agentes entre polígonos y aporta mayor cantidad de rutas cortas entre dos puntos.

Cada región o membrana podrá contener a otras en su interior, esta característica diferencia este modelo de las mallas de navegación, donde los polígonos no pueden contener a otros. Para cada una de estas regiones al igual que en los sistemas P existirán reglas que serán las que definan en dependencia de la situación, si los agentes pueden salir o no de esa región, estas definen además si una membrana puede ser disuelta o dividida. Un ejemplo de la utilidad de disolverse las membranas sería en un videojuego, en el mismo existiría una membrana que delimitaría un edificio dentro del cual existen otras membranas, que definen cada una de las habitaciones del mismo, en una situación en que una de estas habitaciones se vea afectada y por tanto los agentes no puedan acceder a ella, simplemente se disolvería esa pequeña membrana y no afectaría la navegación por el edificio ni por el resto de las habitaciones.

Al igual que las membranas este modelo puede ser representado por un árbol jerárquico, en el cual el padre sería la membrana principal, de esta se ramificarían los hijos, es decir cada una de las membranas interiores. Cada nodo representa una membrana, los agentes que se encuentran en el mismo nodo poseerán una comunicación directa, el intercambio de información entre agentes en el mundo virtual va a disminuir según la diferencia entre los

niveles de los nodos del árbol en que se encuentren.

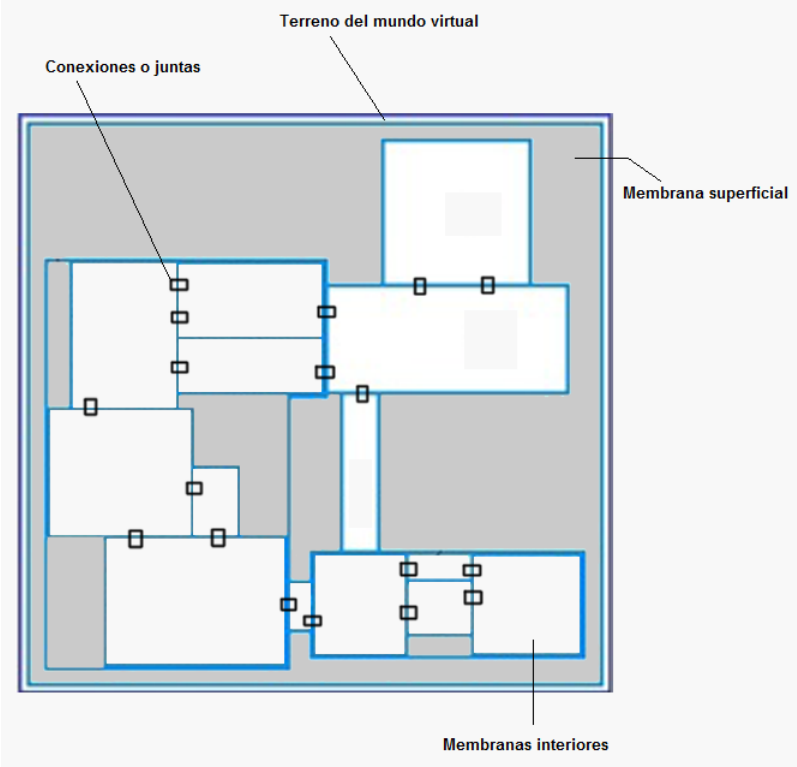

Figura 2.2: Ejemplo de un MCMemb

La búsqueda de camino se realizará de forma global al igual que en las mallas de navegación, incluyendo también en zonas particulares. En la figura mostrada a continuación se efectuarán búsquedas globales para los caminos entre las membranas: D, E, L, M, N y O. Mientras que en las membranas interiores de O y N se realizarán búsquedas particulares, reduciendo así la cantidad de nodos a comprobar en la búsqueda para estos casos. Si se necesita encontrar el camino más corto entre membranas que se encuentran entre membranas que no están al mismo nivel entonces se ejecutarán las dos búsquedas, por ejemplo para hallar el camino entre I y M se realiza búsqueda global para el camino desde O a M, una vez hallado se realiza la búsqueda interna para el camino desde I hasta una de las juntas que une a O con L que se encuentran en J y K.

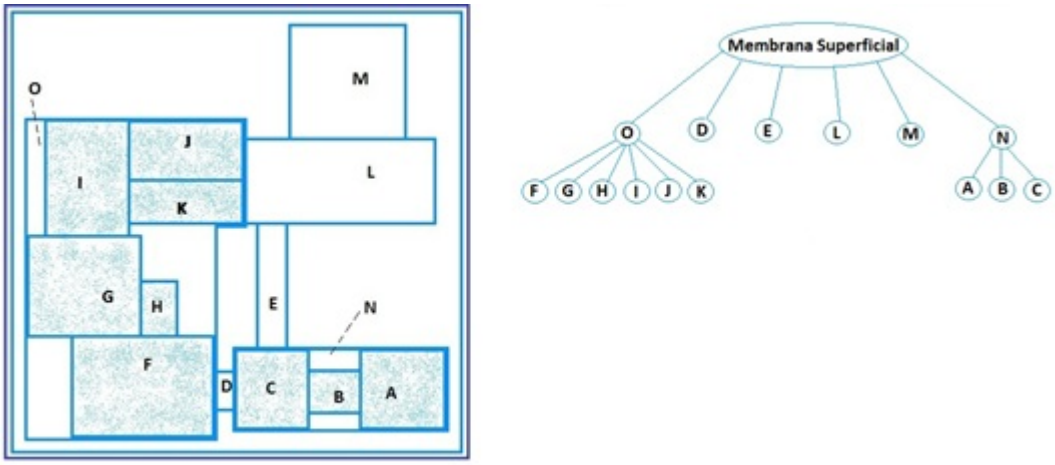

Figura 2.3: MCMemb y su árbol asociado

Como se observa existen marcadas diferencias entre este nuevo modelo cognitivo MCMemb y las mallas de navegación, las cuáles se resumen en la siguiente tabla:

Modelo Cognitivo	Polígonos Convexos	Un polígono contiene otros	Varios puntos de acceso entre los polígonos	Diferenciación de la comunicación entre agentes	Búsqueda de Caminos a nivel global y particular
Malla de Navegación	Sí	No	No	No	No
MCMemb	Sí	Sí	Sí	Sí	Sí

Tabla 2.1: Características de MCMemb y Malla de Navegación

Se debe tomar en cuenta que las membranas aportarían una discretización lógica del entorno y no física, esta característica puede atentar contra la eficiencia del entorno, en tanto siempre un modelo MCMemb requerirá más membranas que polígonos una malla, pero aportaría información adicional que puede ser de mucho interés en entornos específicos donde los límites de regiones no sean físicos sino lógicos, entiéndase en un juego, un campamento, una ciudad, un bosque, entre otros, entornos muy difíciles de representar mediante una malla de navegación.

Por otro lado este modelo pudiera mantener la restricción de crear los polígonos (membranas) a partir de espacios libres de obstáculos estáticos, asumiendo incluso el tránsito libre de los elementos de membranas superiores a través de las membranas interiores, lo que facilita el movimiento de los agentes, sin embargo no estaría limitado a este tipo de entornos, manteniendo otras características ya descritas anteriormente.

Es necesario continuar la presente investigación ya que esta es una versión inicial de lo que podrá ser el modelo cognitivo, una vez lograda la implementación y aplicación de MCMemb en un entorno virtual surgirán nuevas interrogantes respecto al tema.

2.3. Aplicación de pruebas al algoritmo Colonia de Hormigas

Como consta en el capítulo anterior el modelo bioinspirado Colonias de Hormigas es muy utilizado para resolver diversos problemas, se ha aplicado eficientemente en problemas de aproximación, segmentación de imágenes, para el análisis de grandes árboles filogenéticos que son aquellos que revelan la relación evolutiva entre varias especies. Además se utiliza para realizar los planes de producción y mantenimiento de empresas. Estas aplicaciones han demostrado que la principal ventaja de usar las Colonias de Hormigas es que establece la solución con menos esfuerzo computacional y tiempo, sin deteriorar la calidad de la solución.

En los entornos virtuales, se utilizan principalmente en la búsqueda dinámica de caminos cortos, es decir donde el problema a resolver va cambiando a medida que se ejecuta el algoritmo. Esto se aplicaría a los agentes que se mueven por el modelo cognitivo del entorno permitiéndoles no solo encontrar la mejor ruta para ir de una zona a otra sino también para moverse en la región donde están. Pudiendo ser aplicado como algoritmo de búsqueda de camino en un entorno virtual.

Se creará una aplicación para realizar pruebas al algoritmo Colonia de Hormigas con el fin de mostrar no solo el proceso que realizan las colonias de hormigas para encontrar el camino más corto, sino que dicha aplicación permitirá además seleccionar los puntos a través de los cuáles se ejecutará el algoritmo, así como variar diversas componentes que influirán en la resolución del problema.

2.3.1. Descripción de la Aplicación

La aplicación de pruebas del algoritmo colonia de hormigas permitirá encontrar el camino más corto para recorrer todos los puntos del área, para obtener este resultado se proponen funcionalidades como:

- Agregar puntos.
- Mostrar todos los caminos entre los puntos.
- Bloquear la escena, no se permitirá agregar nuevos puntos.
- Ejecutar el algoritmo.
- Modificar variables del algoritmo.
- Reiniciar la escena.

Para realizar el algoritmo es necesario tener en cuenta diferentes aspectos recogidos en el artículo *Ant colonies for the travelling salesman problem* de Dorigo [Dorigo and Gambardella, 1997]. Entre estos se encuentran:

- El algoritmo utiliza una matriz de feromonas ($T = T_{i,j}$) para la construcción de soluciones potenciales. Esta matriz representa la cantidad de feromona que se va almacenando entre cada par de nodos (*i, j*). Los valores iniciales son fijados a un valor constante: T (concentración inicial de feromonas) = T_0 para todo (*i, j*) siendo T mayor que cero.
- Se utiliza una matriz de distancias ($D = D_{ij}$) para guardar la distancia a recorrer del nodo *i* al *j*. Situados en el nodo *i*, n_i representa el conjunto de nodos aún no visitados. La probabilidad de escoger el nodo *j* estando en el nodo *i* está definida por la siguiente expresión:

$$p_k(i,j) = \begin{cases} \dfrac{[t_{i,j}]^\alpha * [n_{i,j}]^\beta}{\sum_{h \in J_k(i)} [t_{ih}]^\alpha * [n_{ih}]^\beta} & \text{si} \quad j \in J_k(i) \\ \\ 0 & \text{en otro caso} \end{cases}$$

donde:

$p_k(i,j)$ es la probabilidad de que la hormiga *k* se mueva del nodo *i* al *j*.

$t_{i,j}$ es la concentración de feromonas que hay en la arista a_{ij}.

$J_k(i)$ es el conjunto de nodos alcanzables desde *i* no visitados aún por la hormiga *k*.

$n_{i,j}$ es la información heurística de la arista a_{ij}.

α es la importancia concedida a las feromonas en las aristas.

β es la importancia concedida a la heurística de la distancia entre las aristas.

Por lo tanto la probabilidad de que una ciudad sea elegida está en función de qué tan cerca de la ciudad se está y que cantidad de feromona existe en ese camino. Además, se puede determinar cuál de ellos tiene un peso mayor al ajustar los parámetros α y β.

- Para actualizar las feromonas en las aristas, se utiliza:

$$t(i,j) = pt(i,j) + \sum_{k-1}^{m} \Delta t_k(i,j)$$

$$\triangle t_k(i,j) = \begin{cases} \dfrac{1}{L_k} & \text{si la hormiga } \quad k \quad \text{ha visitado la arista } \quad a_{ij} \\ \\ 0 & \text{en otro caso} \end{cases}$$

donde:

$pt(i,j)$ es la multiplicación entre la concentración de feromonas que hay en la arista a_{ij} y *p* que es el coeficiente de evaporación de feromonas, *p* será un número entre 0 y 1. A valores más pequeños de p, más rápido será la evaporación de feromonas.

$\sum_{k-1}^{m} \Delta t_k(i,j)$ es la cantidad de feromona depositada por la hormiga *k* en la arista, es definido por L_k que es la longitud del camino creado por esta hormiga. Por tanto, viajes cortos se traducirá en mayores niveles de feromona depositada en las aristas.

Capítulo 3

Análisis, Diseño e Implementación de la Aplicación

3.1. Metodología y Herramientas utilizadas para el desarrollo de la aplicación

Se selecciona la metodología ágil XP para guiar el desarrollo de la aplicación, debido a que el objetivo fundamental es desarrollar una aplicación con calidad, sin necesidad de generar un exceso de documentación [Cortizo et al., 2003]. Además porque XP es una metodología centrada en potenciar las relaciones interpersonales como clave para el éxito del desarrollo de software, ya que promueve el trabajo en equipo, se preocupa por el aprendizaje de los desarrolladores y propicia un buen clima de trabajo [Canos et al., 2005]. Como IDE es seleccionado QTcreator debido a que utiliza el lenguaje de programación C++, por lo que se puede utilizar la programación orientada a objetos. Además permite la construcción de interfaces gráficas visualmente atractivas en poco tiempo a partir del framework QT, posibilitando que los esfuerzos se concentren en la funcionalidad de la aplicación. Una de sus principales ventajas es que funciona en las principales plataformas como Windows, Mac y Linux; [Corporation, 2008] también se encuentra entre los software libres y de código abierto que más apoyo de la comunidad de programadores recibe. Utiliza como principal lenguaje de programación C++. Este lenguaje es

versátil, potente y orientado a objetos de probada eficacia para generar aplicaciones de alto rendimiento. Además es uno de los lenguajes de sistemas más conocido en el mundo, altamente compatible y soporta gran variedad de bibliotecas gráficas que son muy utilizadas en la industria de los videojuegos.

3.2. Personal relacionado con el sistema

En la siguiente tabla se definen las personas que de una forma u otra van a interactuar con la aplicación.

Persona relacionada con el sistema	Descripción
Usuario	Es la persona que podrá realizar cualquier actividad dentro del sistema, sin ningún tipo de restricción.

Tabla 3.1: Personal relacionado con el sistema

3.3. Exploración

La metodología de desarrollo XP comienza con su fase de exploración, los clientes plantean las historias de usuario que son de interés para la primera entrega del producto. Al inicio de esta etapa el equipo de desarrollo deberá estudiar y familiarizarse con las tecnologías, herramientas y prácticas que se utilizarán para el desarrollo del proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. Esta fase no suele extenderse más de unas semanas o unos pocos meses, en dependencia del tamaño del proyecto y la familiaridad de los programadores con la tecnología.

Las historias de usuarios son la técnica utilizada para especificar los requisitos del software, representar una breve descripción del comportamiento del sistema, donde se emplea la terminología del cliente sin lenguaje técnico, se realiza una por cada funcionalidad del sistema. El tratamiento de las historias de usuario es muy dinámico y flexible, cada historia de usuario

debe ser lo suficientemente comprensible y delimitada para que un programador pueda implementarla. Son tareas que responden a objetivos específicos que un programador debe realizar para darles respuesta. Las mismas reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación.

Las funcionalidades reflejadas en las historias de usuarios deben ser programadas en un tiempo entre una y tres semanas, por lo que las mismas no deben tener una complejidad muy elevada que impida esto. Si la estimación de una historia de usuario es superior a tres semanas, debe ser dividida en dos o más historias.

La metodología XP no propone ningún formato específico para las historias de usuario. Plantea que pueden ser tan simples como una tarjeta de cartón, con una descripción de la funcionalidad deseada, y el equipo de desarrollo será el encargado de decidir los detalles adicionales que contendrán.

A partir de los datos necesarios para la planificación y estimación de las historias de usuario y las plantillas utilizadas para el desarrollo de diversas aplicaciones, se propone la utilización de la siguiente plantilla:

Historia de Usuario	
Número: *Número de la historia de usuario*	**Nombre:** *Nombre de la historia de usuario*
Usuario: *Usuario del Sistema que utiliza o realiza la acción*	**Puntos de Estimación:** *Tiempo estimado para realizar la actividad*
Prioridad: *Cuán importante es para el cliente o el equipo de desarrollo*	**Nivel de Complejidad:** *Nivel de dificultad para desarrollar la historia de usuario por el desarrollador*
Iteración: *Iteración a la que pertenece*	
Descripción: *Se describe en qué consiste la historia de usuario teniendo en cuenta las acciones de los usuarios y la respuesta del sistema.*	
Observaciones: *Información que sea necesaria agregar para un mejor entendimiento de la historia de usuario.*	

Tabla 3.2: Plantilla de historia de usuarios

3.3.1. Historias de Usuarios

Se especificaron 6 historias de usuario para todas las funcionalidades de la aplicación demostrativa.

Historia de Usuario	
Número: 1	**Nombre:** Agregar puntos a la escena.
Usuario: Todos.	**Puntos de Estimación:** 1
Prioridad: Alta.	**Nivel de Complejidad:** Media.
Iteración: 1	
Descripción: El usuario selecciona la opción agregar puntos y coloca con el *mouse* al dar *clic* los puntos que desee. El sistema muestra en la escena los puntos en la medida que el usuario los agrega.	
Observaciones:	

Tabla 3.3: Historia de usuario: Agregar puntos a la escena

Historia de Usuario	
Número: 2	**Nombre:** Mostrar todos los caminos.
Usuario: Todos.	**Puntos de Estimación:** 1
Prioridad: Baja.	**Nivel de Complejidad:** Baja.
Iteración: 1	
Descripción: El usuario selecciona la opción Mostrar Caminos. El sistema no permitirá la creación de nuevos puntos y mostrará todas las conexiones posibles entre los puntos existentes.	
Observaciones: Si se desea añadir nuevos puntos deberá seleccionarse la opción agregar puntos.	

Tabla 3.4: Historia de usuario: Bloquear escena

Historia de Usuario	
Número: 3	**Nombre:** Ejecutar Algoritmo.
Usuario: Todos.	**Puntos de Estimación:** 2
Prioridad: Alta.	**Nivel de Complejidad:** Alta.
Iteración: 2	
Descripción: El usuario selecciona la opción ejecutar algoritmo. El sistema mostrará el grafo resultante del proceso.	
Observaciones:	

Tabla 3.5: Historia de usuario: Ejecutar Algoritmo

Historia de Usuario	
Número: 4	**Nombre:** Mostrar Formulario Opciones.
Usuario: Todos.	**Puntos de Estimación:** 1
Prioridad: Media.	**Nivel de Complejidad:** Baja.
Iteración: 1	
Descripción: El usuario selecciona la opción Opciones. El sistema mostrará el formulario con los parámetros correspondientes.	
Observaciones: Los parámetros serán visualizados con sus valores originales.	

Tabla 3.6: Historia de usuario: Mostrar Formulario Opciones

Historia de Usuario	
Número: 5	**Nombre:** Modificar parámetros de las opciones.
Usuario: Todos.	**Puntos de Estimación:** 1
Prioridad: Media.	**Nivel de Complejidad:** Media.
Iteración: 2	
Descripción: El usuario introduce los valores deseados para cada parámetro y selecciona la opción aceptar. El sistema ejecuta los cambios y regresa a la interfaz anterior.	
Observaciones:	

Tabla 3.7: Historia de usuario: Modificar parámetros de las opciones

Historia de Usuario	
Número: 6	**Nombre:** Reiniciar Escena.
Usuario: Todos	**Puntos de Estimación:** 1
Prioridad: Baja.	**Nivel de Complejidad:** Media.
Iteración: 2	
Descripción: El usuario selecciona la opción Reiniciar Escena. El sistema limpia la escena borrando todos los puntos y conexiones creados hasta el momento.	
Observaciones:	

Tabla 3.8: Historia de usuario: Reiniciar Escena

3.4. Estimación y Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario y los desarrolladores realizan una estimación del esfuerzo necesario de cada una de ellas. Estas estimaciones de esfuerzo asociadas a la implementación son medidas en puntos, generalmente a las historias de usuarios son asignados entre 1 y 3 puntos, cada punto equivale a una semana de programación.

Por otra parte se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Al analizar los requisitos de forma ágil, y producir estimaciones acerca del tiempo de desarrollo, donde todos los integrantes del equipo de trabajo participaron en su confección, las planificaciones serán respetadas por todos los implicados en el desarrollo y permitirán un desarrollo ágil y eficiente de la aplicación. La planificación se realiza en base al tiempo o el alcance. Si se hace por tiempo se multiplica el número de iteraciones por el tiempo que tomará implementar todas las historias de usuarios, determinándose cuántos puntos se pueden completar. Si la planificación se realiza en base al alcance se divide la suma de puntos de las historias de usuario seleccionadas entre el tiempo que tomará implementar todas las historias de usuarios, obteniendo así el número de iteraciones necesarias para su implementación.

A pesar de que la planificación realizada cumpla con todos los requerimientos, los riesgos son altos ya que el tiempo es corto, algunas de las historias de usuarios poseen una alta complejidad y equipo de trabajo está poco familiarizado con el IDE y el *framework* que se utilizarán para el desarrollo de la aplicación. Debido a estas circunstancias, se tomaron medidas que permitirán el desarrollo de la solución de forma segura y eficiente.

- Mantener un control de versiones del desarrollo de la aplicación, que permita regresar a una versión anterior y funcional si ocurriese algún problema.
- Comprobación periódica de la aplicación, que permita probar el funcionamiento correcto de la aplicación y evitar que algunas funcionalidades presenten errores debido a la incorporación de las nuevas funcionalidades o por el constante cambio en el código.

3.4.1. Estimación de esfuerzos por historia de usuario

Para desarrollar correctamente la aplicación, se realizó una estimación para cada una de las historias de usuario identificadas, llegando a los siguientes resultados:

Historia de Usuario	Puntos de Estimación
Agregar puntos a la escena	1
Mostrar todos los caminos.	1
Ejecutar Algoritmo.	2
Mostrar Formulario Opciones.	1
Modificar parámetros de las opciones.	1
Reiniciar Escena	1

Tabla 3.9: Estimación de esfuerzo por historias de usuarios

3.4.2. Planificación de iteraciones

El sistema antes de ser entregado deberá pasar por tantas iteraciones como se consideren necesarias. En la primera iteración se intenta establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto, así como las funcionalidades básicas del mismo.

Una vez definidas las historias de usuarios y estimado el esfuerzo propuesto para la realización de cada una de ellas, se decide realizar el sistema en 2 iteraciones.

Planificación de historias de usuario de la primera iteración.

En esta iteración se seleccionaron las funcionalidades más básicas del sistema y las que permiten moldear la arquitectura de la aplicación. Las historias de usuarios asociadas a esta iteración son:

- Agregar puntos a la escena.
- Bloquear escena.
- Mostrar Formulario Opciones.

Planificación de historias de usuario de la segunda iteración.

En esta última iteración serán implementadas las historias de usuarios con mayor peso de la aplicación y las restantes que no eran básicas, por lo que no fue necesario implementarlas en la primera iteración. Al integrar el resultado de la iteración anterior con los de esta se

obtendrá la aplicación demostrativa en su versión 1.0. Al culminar esta iteración la aplicación podrá ser sometida a diversos procesos de prueba para comprobar su eficiencia y desempeño.

Las historias de usuarios asociadas a esta iteración son:

- Ejecutar Algoritmo.
- Modificar parámetros de las opciones.
- Reiniciar Escena.

3.4.3. Plan de duración de iteraciones

El plan de duración de cada una de las iteraciones que se llevarán a cabo, tiene como finalidad mostrar la duración de cada iteración, así como el orden en que serán implementadas las historias de usuarios correspondientes a las mismas.

Iteración	Historias de Usuarios	Duración total de iteraciones
1	Agregar puntos a la escena	3 semanas
	Mostrar todos los caminos.	
	Mostrar Formulario Opciones.	
2	Ejecutar Algoritmo.	4 semanas
	Modificar parámetros de las opciones.	
	Reiniciar Escena.	

Tabla 3.10: Plan de duración de iteraciones

3.5. Diseño de la Aplicación

La metodología XP no requiere la descripción del sistema por medio de diagramas de clase utilizando notación UML, sino que se guía por técnicas como las tarjetas CRC (Contenido, Responsabilidad y Colaboración). Lo cual no implica que no se utilicen diagramas para obtener una mejor visión y comunicación entre el equipo de desarrollo, siempre y cuando su complejidad no sea alta y defina información importante.

3.5.1. Tarjetas CRC

Las tarjetas CRC tienen dos características principales, su simpleza y adaptabilidad. Una tarjeta CRC no es más que una ficha de papel o cartón que representa a una entidad del sistema. Estas permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Además permiten que el equipo completo contribuya en la tarea del diseño. Estas tarjetas se utilizan para estructurar las clases y a su vez definir las responsabilidades sobre las mismas, así como la simulación de escenarios en el sistema.

A partir los datos que se consideran principales y las plantillas anteriormente utilizadas para el desarrollo de otras aplicaciones, se propone la utilización de la siguiente plantilla:

Tarjeta CRC	
Clase: *Nombre de la clase que se está modelando.*	
Súper Clase: *Nombre de la clase padre en la herencia.*	
Sub Clase(s): *Nombre de la(s) clase(s) hija en la herencia.*	
Responsabilidades: *Es una descripción del propósito de la clase.*	**Colaboraciones:** *Indica con cuáles otras clases se requiere relación para cumplir la responsabilidad.*

Tabla 3.11: Plantilla de Tarjeta CRC

Tarjeta CRC	
Clase: CHormiga	
Súper Clase: -	
Sub Clase(s): -	
Responsabilidades: Permite la creación de las hormigas, con sus nodos visitados, el camino recorrido y la longitud total de su camino.	**Colaboraciones:** -

Tabla 3.12: Tarjeta CRC: CHormiga

Tarjeta CRC	
Clase: CColoniaHormigas	
Súper Clase: -	
Sub Clase(s): -	
Responsabilidades: Permite la creación de la colonia de hormigas. Simula las hormigas, actualiza las aristas y reinicia la colonia.	**Colaboraciones:** CHormiga Point Qlist

Tabla 3.13: Tarjeta CRC: CColoniaHormigas

Tarjeta CRC	
Clase: Point	
Súper Clase: QGraphicsEllipseItem	
Sub Clase(s): -	
Responsabilidades: Permite la creación de los puntos, guardando sus coordenadas y los valores para mostrarlos.	**Colaboraciones:** QObject

Tabla 3.14: Tarjeta CRC: Point

Tarjeta CRC	
Clase: Scene	
Súper Clase: QGraphicsScene	
Sub Clase(s): -	
Responsabilidades: Permite la creación de la escena y es donde se pintan todos los elementos tales como los nodos y caminos.	**Colaboraciones:** CColoniaHormigas Qlist QPainter Point View

Tabla 3.15: Tarjeta CRC: Scene

Tarjeta CRC	
Clase: View	
Súper Clase: QGraphicsView	
Sub Clase(s): -	
Responsabilidades: Permite la creación de un widget para mostrar el contenido de la escena.	**Colaboraciones:** -

Tabla 3.16: Tarjeta CRC: View

3.5.2. Interfaces de Usuario

La aplicación desarrollada cuenta con diferentes funcionalidades, a continuación se mostrarán las interfaces de usuario para cada una de ellas.

Al ejecutar la aplicación se puede observar la interfaz inicial y las cinco funcionalidades principales.

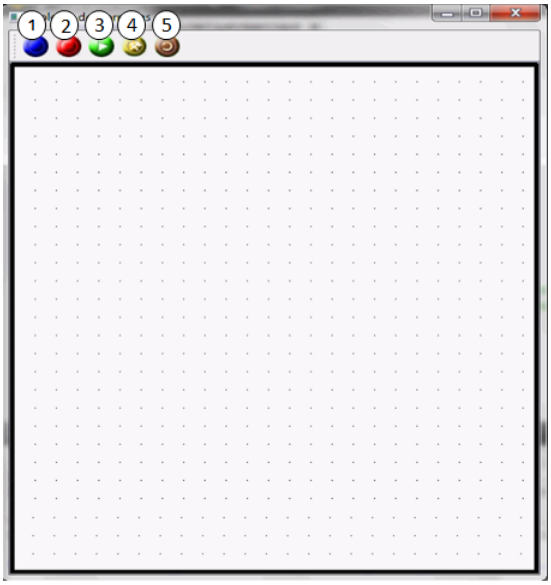

Figura 3.1: Interfaz inicial de la aplicación

1. Agregar Puntos: Permite agregar los puntos deseados a la escena. Ver figura 3.2.

2. Mostrar Caminos: Muestra todos los caminos posibles entre los nodos y bloquea la escena impidiendo agregar nuevos puntos. Ver figura 3.3.

3. Ejecutar Algoritmo: Ejecuta la funcionalidad principal del algoritmo dando paso a la solución y mostrando el camino final. Ver figura 3.4.

4. Opciones: Muestra un nuevo formulario donde posibilita la modificación de los diferentes parámetros del algoritmo. Ver figura 3.5.

5. Reiniciar: Reinicia la escena y todos los valores para comenzar a ejecutar el algoritmo nuevamente. Ver figura 3.6.

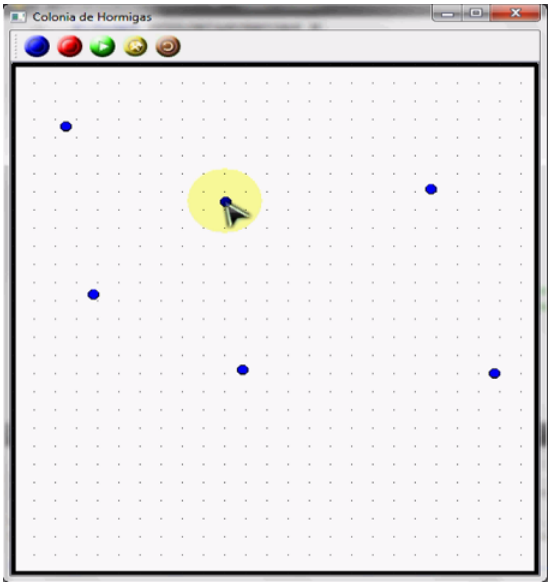

Figura 3.2: Interfaz Agregar Puntos

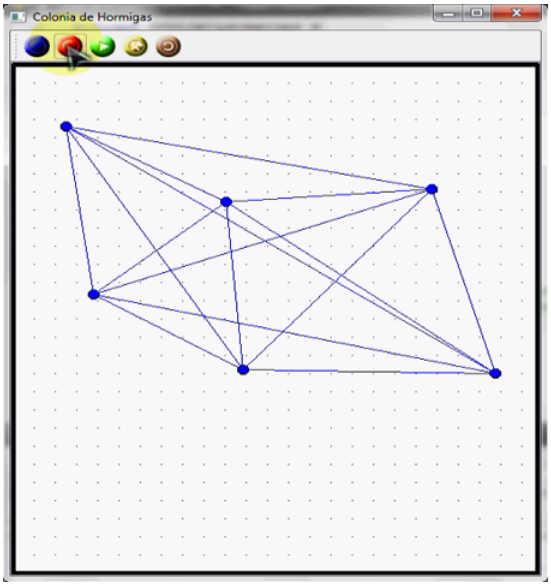

Figura 3.3: Interfaz Mostrar Caminos

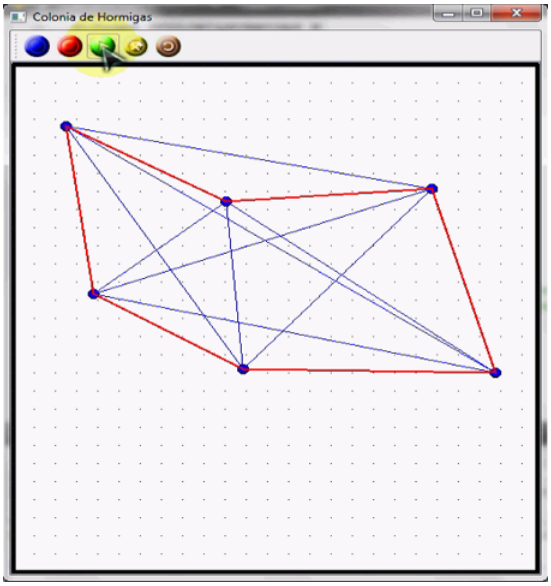

Figura 3.4: Interfaz Ejecutar Algoritmo

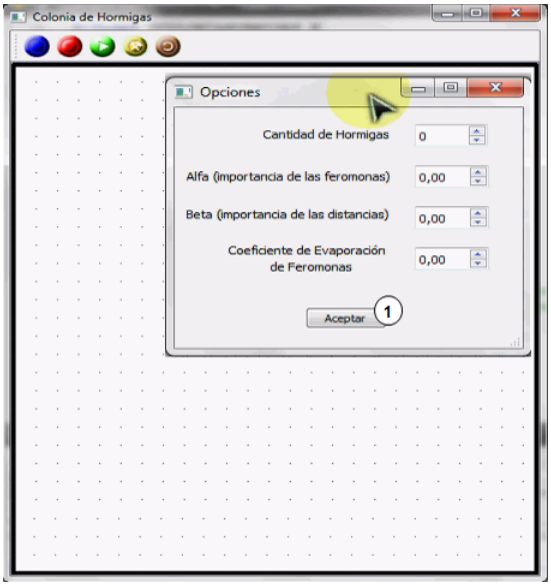

Figura 3.5: Interfaz Opciones

1. Botón Aceptar: Luego de modificar los diferentes valores permite actualizar los datos

para ejecutar el algoritmo.

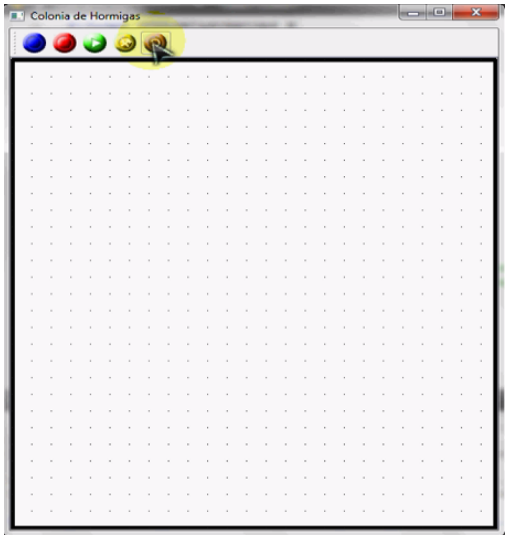

Figura 3.6: Interfaz Reiniciar Colonia

3.6. Desarrollo de Iteraciones

En la fase de Planificación se detallaron las historias de usuario agrupadas en cada iteración a desarrollar. Al iniciar cada iteración se lleva a cabo una revisión del plan de iteraciones y se modifica en caso de ser necesario. Como parte de este plan se descomponen las historias de usuarios en tareas de programación o ingeniería, a cada tarea se le asigna un responsable que puede ser un grupo de desarrollo o una persona. Estas tareas no tienen que necesariamente ser entendidas por el cliente, pueden ser escritas en lenguaje técnico y son solo para el uso de los programadores.

A partir de la planificación realizada se llevaron a cabo dos iteraciones de desarrollo sobre la aplicación demostrativa, permitiendo al final de la última iteración obtener una aplicación operativa cumpliendo con todas las funcionalidades definidas. A continuación se especifican cada una de las iteraciones desarrolladas.

3.6.1. Iteración 1

En esta iteración se desarrollaron las funcionalidades básicas de la aplicación que darán soporte a las funciones más complejas de la segunda iteración.

Historias de Usuarios	Tiempo estimado (semanas)	Tiempo real (semanas)
Agregar puntos a la escena	1	1
Mostrar todos los caminos.	1	1
Mostrar Formulario Opciones.	1	1

Tabla 3.17: Historias de Usuarios desarrolladas en la primera iteración

Tareas de ingeniería de las historias de usuario (HU) correspondientes a esta iteración:

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 1
Nombre de la tarea: Crear interfaz visual de la aplicación	
Tipo de tarea: Diseño	**Puntos estimados:** 0.3
Fecha inicio: 27/03/2011	**Fecha fin:** 29/03/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Esta tarea define el diseño de la pantalla que mostrará la escena donde el usuario agregará los puntos.	

Tabla 3.18: Tarea de Ingeniería: Crear interfaz visual de la aplicación

Tarea de Ingeniería	
No. De la Tarea: 2	**No. De la HU:** 1
Nombre de la tarea: Agregar puntos a la escena	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio: 30/03/2011	**Fecha fin:** 02/04/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Implementación de las clases necesarias para agregar y mostrar los puntos en la escena.	

Tabla 3.19: Tarea de Ingeniería: Agregar puntos a la escena

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 2
Nombre de la tarea: Mostrar conexiones.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio: 03/04/2011	**Fecha fin:** 07/04/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Implementación de las clases necesarias para mostrar las conexiones entre los puntos agregados por el usuario.	

Tabla 3.20: Tarea de Ingeniería: Mostrar conexiones

Tarea de Ingeniería	
No. De la Tarea: 2	**No. De la HU:** 2
Nombre de la tarea: Bloquear escena.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.3
Fecha inicio: 08/04/2011	**Fecha fin:** 09/04/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Implementar la funcionalidad para que el usuario no pueda agregar nuevos puntos a la escena.	

Tabla 3.21: Tarea de Ingeniería: Bloquear Escena

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 4
Nombre de la tarea: Crear interfaz visual para el formulario opciones.	
Tipo de tarea: Diseño	**Puntos estimados:** 0.3
Fecha inicio: 10/04/2011	**Fecha fin:** 11/04/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Esta tarea define el diseño de la pantalla que mostrará el formulario donde se podrán modificar diversos valores del algoritmo.	

Tabla 3.22: Tarea de Ingeniería: Crear interfaz visual para el formulario opciones

Tarea de Ingeniería	
No. De la Tarea: 2	**No. De la HU:** 4
Nombre de la tarea: Mostrar formulario opciones.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.7
Fecha inicio: 12/04/2011	**Fecha fin:** 16/04/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Implementar la funcionalidad para mostrar el formulario opciones a partir de un botón en la interfaz principal.	

Tabla 3.23: Tarea de Ingeniería: Mostrar formulario opciones

3.6.2. Iteración 2

En esta iteración se desarrollaron las funcionalidades necesarias para culminar el desarrollo de la aplicación.

Historias de Usuarios	Tiempo estimado (semanas)	Tiempo real (semanas)
Ejecutar Algoritmo.	2	2
Modificar parámetros de las opciones.	1	1
Reiniciar Escena.	1	1

Tabla 3.24: Historias de Usuarios desarrolladas en la segunda iteración

Tareas de ingeniería de las historias de usuario correspondientes a esta iteración:

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 3
Nombre de la tarea: Ejecutar algoritmo principal.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 2
Fecha inicio: 17/04/2011	**Fecha fin:** 30/04/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Implementación de las clases y funcionalidades necesarias para ejecutar el algoritmo y mostrar el camino resultante.	

Tabla 3.25: Tarea de Ingeniería: Ejecutar algoritmo principal

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 5
Nombre de la tarea: Modificar parámetros.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 01/05/2011	**Fecha fin:** 07/05/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Implementación de las funcionalidades necesarias para modificar las variables correspondientes**.**	

Tabla 3.26: Tarea de Ingeniería: Modificar parámetros

Tarea de Ingeniería	
No. De la Tarea: 1	**No. De la HU:** 5
Nombre de la tarea: Reiniciar	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 08/05/2011	**Fecha fin:** 14/05/2011
Programador responsable: Jeydi A. Santiago Rubiera	
Descripción: Implementación de las funcionalidades para reiniciar la escena, así como todos los valores necesarios para ejecutar el algoritmo desde el comienzo.	

Tabla 3.27: Tarea de Ingeniería: Reiniciar

3.7. Resultados obtenidos luego de desarrolladas las iteraciones

Como resultado de las iteraciones realizadas se obtiene una aplicación completamente funcional, la cual utiliza el algoritmo colonia de hormigas para encontrar el camino más corto entre diferentes puntos insertados por el usuario en la escena. Esta aplicación consta de una serie de particularidades a la hora de la ejecución, debido a que está basado en valores aleatorios para que las hormigas se puedan desplazar. Estas particularidades surgen en el caso que se aumenta en gran medida la cantidad de puntos y no varían el resto de los parámetros. A propósito de estas características en ocasiones la aplicación puede demorarse en encontrar la solución y a veces mostrar la solución más cercana a la óptima, pues los valores aleatorios obtenidos en la gran cantidad de iteraciones no serían los necesarios para obtener la respuesta correcta.

Esta aplicación podrá ser utilizada por los programadores para estudiar el comportamiento del algoritmo al cambiar los diferentes parámetros del mismo, lo que les permitirá ajustarla a las necesidades de la aplicación donde lo deseen insertar. Además podrá ser utilizada como material de ayuda para las asignaturas de Inteligencia Artificial y Desarrollo de Elementos Inteligentes impartidos en la universidad.

Conclusiones

Se propuso la aplicación del modelo bioinspirado membranas para la creación de un nuevo modelo cognitivo, denominado MCMemb, el cual se inspiró en las mallas de navegación al ser uno de los modelos más utilizados en los entornos virtuales. Este nuevo modelo cognitivo aprovecha las características de las mallas de navegación y presenta otras características que pueden ser aprovechadas para la creación de nuevos juegos o simuladores virtuales.

La selección de la Metodología XP utilizada para el desarrollo de la aplicación posibilitó la creación de los artefactos fundamentales, que permitió el desarrollo de la misma con calidad y cumpliendo con las funcionalidades necesarias.

Se obtuvo una aplicación, que permite observar el proceso que se realiza para hallar el camino más corto entre diversos puntos de una determinada área, utilizando para ello el algoritmo de colonia de hormigas. Se logró crear una aplicación para realizar pruebas al algoritmo la cual podrán utilizar los programadores que lo decidan para comprobar los diferentes valores y las variaciones del algoritmo para adaptarlo a sus necesidades.

Recomendaciones

- Implementar y aplicar el modelo cognitivo MCMemb que se propone, en un entorno virtual y realizar una comparación con otros modelos cognitivos con el fin de comprobar la efectividad de este nuevo modelo.
- Agregar la colonia de hormigas implementada a la biblioteca de inteligencia artificial que se desarrolla en la facultad.

Glosario de Términos

Agente: Entidad autónoma que observa y actúa sobre un entorno para cumplir un objetivo específico. Es la representación en un entorno virtual de personas, animales u objetos.

Entorno Virtual: Espacio conceptual en el que un usuario establece una comunicación, en condiciones de tiempo y espacio posiblemente distintas, con otros usuarios, o con elementos propios del entorno.

Heurística: Regla que permite orientar un algoritmo hacia la solución de un problema. Técnica de programación muy empleada en la Inteligencia artificial, porque permite a un sistema ir construyendo una solución óptima a partir de resultados obtenidos en procedimientos anteriores.

Inteligencia Artificial: Ciencia que intenta reproducir los procesos de la inteligencia humana, a partir de la creación de programas de ordenadores.

Metaheurística: Estrategia general para el diseño de procedimientos heurísticos inteligentes para la resolución de problemas con un alto rendimiento.

Modelos bioinspirados: Representación computacional de algunos elementos y comportamientos presentes en la naturaleza.

Modelos cognitivos: Representación del mundo virtual, ofrece información a los agentes del entorno para que puedan interactuar con el mismo de manera eficiente.

Realidad Virtual: Técnica que trata de modelar ambientes reales en los ordenadores.

Referencias bibliográficas

[Armas et al., 2008] Armas, M. A., Gómez, J., Pérez, C., Meriño, L., and Sepúlveda, J. (2008). *Inteligencia Artificial Aplicada al Análisis de Sistemas Energéticos con Matlab*. Centro de Investigaciones y Desarrollo Tecnológico, Sede Barrio España.

[Autores, 2004] Autores, C. (2004). *AI Game Programing Wisdom 2*. CHARLES RIVER MEDIA.

[Bello, 1998] Bello, R. (1998). *Curso de Métodos de Solución de Problemas para la Inteligencia Artificial*. Universidad Central de Las Villas.

[Bello, 2002] Bello, R. (2002). *Aplicaciones de la Inteligencia Artificial*. Universidad de Guadalajara, Mexico.

[Bourg and Seeman, 2004] Bourg, D. and Seeman, G. (2004). *AI for Game Developers*. O´Reilly.

[Canos et al., 2005] Canos, J., Letelier, P., and Penadé, C. (2005). *Todo Ágil*. Universidad Politécnica de Valencia. [En línea] [Citado el: 17 de marzo de 2011]. http://www.willydev.net/descargas/prev/TodoAgil.pdf.

[Champandard, 2004] Champandard, A. J. (2004). *The Dark Art of Neural Networks. AI Game Programming Wisdom 2*. Charles River Media, INC.

[Charles et al., 2008] Charles, D., Fyfe, C., Livingstone, D., and McGlinchey, S. (2008). *Biologically Inspired Artificial Intelligence for Computer Games*. IGI Publishing.

[Coca, 2009] Coca, Y. (2009). Agentes inteligentes. aplicación a la realidad virtual. *Revista Cubana de Ciencias Informáticas*, 3(1-2):49–54.

[Corporation, 2008] Corporation, N. (2008). *QT Creator IDE and tools*. [En línea] [Citado el: 21 de marzo de 2011]. http://qt.nokia.com/products/developer-tools/.

[Cortizo et al., 2003] Cortizo, J., Gil, D. E., and Ruiz, M. (2003). extreme programming. Technical Report 1, AINetSolutions.

[Dorigo and Gambardella, 1997] Dorigo, M. and Gambardella, L. (1997). Ant colonies for the travelling salesman problem. *Biosystems*, 43(2).

[Duch and Tejedor, 2009] Duch, J. and Tejedor, H. (2009). *Inteligencia Artificial*.

[García and Gómez, 2009] García, D. and Gómez, C. (2009). *Incorporación de comportamientos a elementos que se mueven sobre grafos de camino*. Universidad de la Ciencias Informáticas, Cuba. Tesis de Grado.

[Graham, 2006] Graham, R. (2006). *Real-time Agent Navigation with Neural Networks for Computer Games*. Institute of Technology Blanchardstown.

[Green, 2000] Green, R. (2000). *Steering Behaviors*. Disponible en: http://www.steeringbehaviors.de/.

[Gutiérrez and Pérez, 2010] Gutiérrez, M. and Pérez, M. (2010). Membrane computing meets artificial intelligence: A case study.

[Hinchey and Otros, 2006] Hinchey, M. and Otros (2006). *Requirements of an Integrated Formal Method for Intelligent Swarms*. Information Systems Division NASA, USA.

[Kennedy and Eberhart, 2001] Kennedy, J. and Eberhart, R. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers.

[Londoño, 2001] Londoño, N. (2001). Algoritmos genéticos. Master's thesis, Universidad Nacional de Colombia.

[Martín and Sanz, 2001] Martín, B. and Sanz, A. (2001). *Redes Neuronales y Sistemas Difusos*. RA-MA, 2da edition.

[Martínez and Antúnez, 2008] Martínez, T. and Antúnez, Y. (2008). *Propuesta de Técnicas de Aprendizaje de Elementos Virtuales*. Universidad de la Ciencias Informáticas, Cuba. Tesis de Grado.

[Mendoza, 2001] Mendoza, B. (2001). Uso del sistema de la colonia de hormigas para optimizar circuitos lógicos combinatorios. Master's thesis, Universidad Veracruzana.

[Millington, 2006] Millington, I. (2006). *Artificial intelligent for games*. San Francisco.

[Nieto, 2007] Nieto, J. G. (2007). Algoritmos basados en inteligencia colectiva para la resolución de problemas de bioinformática y telecomunicaciones. Master's thesis, Universidad de Málaga.

[Parra and Marrero, 2007] Parra, Y. and Marrero, I. (2007). *Incorporación de algoritmos genéticos a la biblioteca de inteligencia artificial*. Universidad de la Ciencias Informáticas, Cuba. Tesis de Grado.

[Paun, 1999] Paun, G. (1999). *P Systems with Active Membranes: Attacking NP Complete Problems*.

[Pfeil, 2006] Pfeil, J. (2006). Swarm intelligence.

[Prevot and Herrera, 2008] Prevot, Y. and Herrera, A. (2008). *Aplicaciones de las redes neuronales en entornos virtuales*. Universidad de la Ciencias Informáticas, Cuba. Tesis de Grado.

[Rabuñal and Dorado, 2005] Rabuñal, J. R. and Dorado, J. (2005). *Artificial Neural Networks in Real-Life Applications*. Idea Group Publishing.

[Reyes, 2008] Reyes, D. (2008). *Algoritmos Bioinspirados*. Universidad de la Ciencias Informáticas, Cuba.

[Sempere, 2007] Sempere, J. (2007). Sistemas basados en membranas. sistemas p. Master's thesis, Universidad Politécnica de Valencia, España.

[Smed and Hakonen, 2006] Smed, J. and Hakonen, H. (2006). *Algorithms and Networking for Computer Games*. Wiley.

[van Waveren, 2001] van Waveren, J. (2001). The quake iii arena bot. Master's thesis, University of Technology Delft, Faculty ITS, Netherlands.