UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5 ENTORNOS VIRTUALES

Incorporación de comportamientos a elementos que se mueven sobre grafos de camino.

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

Autores:

Dailyn García Dominguez.

Clariannis Gómez Barroso.

Tutores:

Msc. Yuniesky Coca Bergolla.

Ing. Yessy Cedeño González.

Ciudad de la Habana

junio de 2009

DATOS DE CONTACTO

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

- Graduado de Ciencias de la Computación en la Universidad Central de Las Villas en el año 2003.
- Profesor Asistente en la Universidad de las Ciencias Informáticas.
- Jefe de Dpto de Práctica Profesional del polo de Realidad Virtual de la Facultad 5.
- Obtuvo el Sello Forjadores del Futuro en el 2006 y 2008.
- Defendió la maestría en el año 2007.
- Árbitro de la serie científica de la Universidad.
- Coordinador de un diplomado de Realidad Virtual.
- Cursa un doctorado curricular colaborativo en Ciencias de la Educación en la Universidad de La Habana.

Ing. Yessy Cedeño González Yessy (ycgonzalez@uci.cu**)**

- Graduado con título de oro de Ingeniero en Ciencias Informáticas de la Universidad de las Ciencias Informáticas en el perfil de Realidad Virtual.
- Ha estado vinculado a proyectos productivos de RV desde 2do año de la carrera.
- Actualmente se desempeña como profesor adiestrado y pertenece al Polo de Realidad Virtual de la facultad 5.

AGRADECIMIENTOS

- ***A mis padres*** *por todo el amor y el sacrificio para que yo llegara hasta aquí.*
- ***A la Revolución cubana*** *por darme la oportunidad de convertirme en profesional.*
- ***A mi esposo*** *por su amor, comprensión y apoyo en los momentos difíciles.*
- ***A mis hermanos*** *por la confianza.*
- ***A mis maestros*** *todos, en especial a Yamir por contribuir a mi formación.*
- ***A Clarita*** *por ser compañera y amiga durante estos 5 años.*
- ***A mis amigos*** *de Camagüey y de la universidad por existir en mi vida.*
- *A todos los que de una forma u otra han ayudado a que este trabajo fuera posible.*

Dailyn.

- ***A mis padres*** *por todo el amor, comprensión y apoyo en cada momento.*
- ***A la Revolución*** *por brindarme la posibilidad de superarme en la vida.*
- ***A mi familia*** *por contribuir a mi educación.*
- ***A Leo*** *por el amor y la confianza.*
- ***A Dailyn*** *por ser amiga y hermana durante estos 5 años.*
- ***A mis tutores*** *por el apoyo incondicional.*
- ***A cada uno de mis maestros*** *por aportar un granito de arena a mi formación.*
- ***A mis amigos*** *del barrio y de la UCI por su incondicionalidad.*
- *A todos los que me han apoyado de una forma u otra y siempre han confiado en mí.*

Clariannis.

DEDICATORIA

- ***A mis padres**, fuente de inspiración.*
- ***A mi esposo**, parte indisoluble de mi vida.*
- *A toda mi familia.*

Dailyn

- ***A mis padres**, por predicar con el ejemplo.*
- ***A mi familia** y amigos por ser parte de mi vida.*

Clariannis

RESUMEN

El desarrollo de sistemas de simulación en 3 dimensiones (3D) es hoy muy valorado por los principales científicos del área de la inteligencia artificial. Gran cantidad de entornos requieren elementos que se muevan de manera natural, tarea fundamental de esta rama de las ciencias de la computación. Como en otros aspectos de la vida, el hombre ha observado la naturaleza y ha tratado de representarla en sistemas complejos. Varios de los modelos computacionales inspirados en ella se muestran en este trabajo y se propone uno de ellos como vía para la implementación de agentes creíbles dentro de un entorno virtual 3D, dejando abierta la posibilidad de incorporar otros. Se presenta una herramienta para facilitar la incorporación eficiente de comportamientos a los agentes que intervengan en una aplicación determinada y que se muevan sobre una red de caminos predefinidos, proponiéndose para ello la distribución de inteligencia por todo el entorno.

Palabras claves: agentes creíbles, comportamientos de locomoción, grafo de camino, inteligencia artificial.

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

En nuestro país se realizan grandes esfuerzos para desarrollar la informática como ciencia novedosa que le aporta cada vez más beneficios al hombre dentro de la sociedad. En este campo se ha ido introduciendo en los últimos años el tema de Realidad Virtual (RV) como rama que puede aplicarse para el desarrollo de diversas esferas como son la educación, la medicina, la industria armamentista, los juegos entre otros. Dentro del campo de la RV la Inteligencia Artificial (IA) ha ido tomando fuerza a medida que los recursos de hardware se han desarrollado, en particular los agentes inteligentes constituyen un tema muy novedoso que atrae a científicos de varias áreas del conocimiento. Dentro de la tipología de agentes, los llamados agentes creíbles juegan un papel fundamental en áreas como los videojuegos o los sistemas de simulación, ya sea con fines educativos, recreativos, terapéuticos, etc. Como característica fundamental, estos agentes, representados visualmente por modelos, generalmente 3D, deben moverse simulando la realidad (personas, animales, vehículos, etc.) por un entorno virtual.

El hombre históricamente se ha servido de la naturaleza para cumplir sus objetivos, también en el mundo de la programación esto ha ocurrido. Los llamados modelos bioinspirados son técnicas o algoritmos definidos a partir de una representación de la naturaleza, la vinculación de los agentes creíbles con estos modelos bioinspirados brindan una magnífica oportunidad para simular el movimiento de elementos sobre un entorno gráfico 3D.

Una de las técnicas para lograr el movimiento de agentes dentro de entornos virtuales son los comportamientos de locomoción o *Steering Behavior* (Buckland, 2005) según su nombre en inglés, que es la manera que se emplea para definir que una entidad asuma una forma o comportamiento para su locomoción (Vela, 2008). Los comportamientos de locomoción se derivan de un reconocido modelo bioinspirado como veremos más adelante; son aplicados en disímiles tareas vinculadas con juegos y entornos virtuales. Resultan muy sencillos de manipular e implementar, definen una fuerza que será aplicada

a los elementos para ser tratada dentro de un modelo matemático, este puede ser desde tan simple como la aplicación de la segunda ley de Newton, hasta bibliotecas físicas diseñadas para fines complejos como la *Open Dynamics Engine* (ODE) (Smith, 2007) que es de las más difundidas en el mundo de los videojuegos para ser integrada a motores gráficos.

La ODE es una biblioteca gratuita empleada en la simulación dinámica de sólidos rígidos como pueden ser vehículos terrestres, personas, así como otros elementos dentro de un entorno virtual. Su desarrollo dentro del prototipo del código libre ha posibilitado que diferentes programadores realicen mejoras y colaboren de esta forma con el código inicial.

La *Object Oriented Graphics Engine* (OGRE) fue diseñada para hacer más fácil el desarrollo de juegos 3D. Es una biblioteca empleada para el "renderizado" 3D flexible y orientado a escenas, realizado en el lenguaje de programación C++. Herramienta multiplataforma que funciona en entornos como Windows, Mac OS X o Linux. Su uso es gratuito debido a que es un proyecto de código abierto bajo licencia LGPL. La biblioteca de clase que propone evita la utilización de otras bibliotecas gráficas de nivel inferior como son *OpenGL* y *Direct3D*.

Graphics Three Dimensional Engine (G3D) es otra de las bibliotecas gráficas muy utilizadas por sus conocidas ventajas, fue escrita en C++ empleando código libre bajo la licencia BSD, se adapta fácilmente a las necesidades de los desarrolladores, utilizada para confeccionar juegos comerciales, simuladores militares entre otros. Suministra un conjunto de rutinas y estructuras tan frecuentes que son necesarias en la mayoría de los programas gráficos. Muy consistente con una plataforma fuertemente perfeccionada sobre la que pueden edificarse aplicaciones 3D. Su utilización posibilita la obtención de productos de alta calidad.

En estas circunstancias surge la Universidad de Ciencias Informáticas (UCI) la cual desarrolla desde sus inicios un arduo trabajo sobre el tema. En este sentido la facultad 5 realiza una labor abnegada para desarrollar aplicaciones de RV. A raíz de esto se crea en la facultad la biblioteca *SceneToolKit* (STK) que es una herramienta de apoyo para los programadores de aplicaciones finales de acuerdo con la tendencia internacional de

desarrollar los denominados "*Engines*" o motores gráficos. La STK tiene el objetivo de unir las funcionalidades que son afines a cualquier sistema de RV de modo que simplifique el trabajo de los programadores de juegos y simuladores a través de la reutilización de código.

Gran parte de las aplicaciones que se realizan dentro de un entorno virtual cuentan con herramientas para crear y modificar curvas tridimensionales que se emplean en animaciones de elementos móviles. Con el objetivo de ser incorporado a la STK se realiza una Herramienta de creación y modificación de grafos de camino la cual brinda la posibilidad de crear, modificar y deformar visualmente los caminos del entorno, mediante un juego de herramientas intuitivas y fáciles de utilizar; además permite la aplicación de cálculos de camino mínimo entre nodos del grafo y guardar toda la información en fichero. (Cedeño, 2008).

Dentro de las necesidades que aun persisten, se encuentra la de incorporarle a la STK facilidades para proveer de cierta inteligencia a los elementos dentro de los entornos virtuales. Una de las primeras acciones a realizar es incorporar comportamientos inteligentes de locomoción a elementos que se muevan sobre un grafo de camino, no solo para la STK, sino lo más genérico posible para ser utilizado sobre otras herramientas de visualización. Esto motiva a definir como problema científico de este trabajo *¿Cómo incorporar comportamientos inteligentes a elementos que se mueven sobre grafos de camino?*

Utilizando los trabajos realizados previamente en el polo de Realidad Virtual de la UCI, como son la creación y modificación de grafos de camino y la incorporación de comportamientos a autos (Saurín, 2008) (Sánchez, 2008), se propone como objetivo, *desarrollar una herramienta que facilite la incorporación de comportamientos inteligentes a elementos virtuales que utilicen grafos de camino*. Para darle cumplimiento a dicho objetivo se define como objeto de estudio el *proceso de incorporación de comportamientos a elementos virtuales.* Particularmente se trabajará en la *incorporación de comportamientos de locomoción a elementos que utilizan grafos de camino.*

Para ello se trazan como tareas investigativas.

1. Búsqueda bibliográfica sobre lo que se ha hecho en Cuba y el mundo referente a los comportamientos de elementos dentro de un entorno virtual.
2. Estudio de las herramientas necesarias para el desarrollo de la aplicación.
3. Desarrollo del análisis y diseño del sistema.
4. Cumplimiento de la fase de implementación del sistema.

Se piensa que al final del presente trabajo *con el desarrollo de un sistema que utilice la herramienta de edición de grafos se logrará facilitar la incorporación de comportamientos inteligentes a elementos virtuales.*

Para darle cumplimiento a los objetivos propuestos en el trabajo se emplean como Métodos teóricos el Método analítico-sintético que permite *analizar la bibliografía vinculada con el objeto de estudio y llegar a las características comunes existentes entre ellas así como darle solución al problema.* El Método Histórico-Lógico permitirá *el estudio de la evolución, desarrollo y estado actual de la inteligencia artificial, específicamente aplicada a los comportamientos inteligentes de agentes dentro de entornos virtuales.*

Con el desarrollo del presente trabajo se obtendrá una herramienta que permita la incorporación de comportamientos inteligentes a elementos virtuales que se muevan sobre un grafo de camino. Podrá ser utilizado por proyectos del polo de realidad virtual de la facultad 5 de la Universidad de las Ciencias Informáticas que trabajen la creación de juegos y simuladores y necesiten contar con estas características. Se obtendrá un informe donde se explicará detalladamente los resultados del proceso de desarrollo de la herramienta así como las investigaciones realizadas sobre el tema en cuestión, las que harán posible que posteriormente se le realicen mejoras o modificaciones para ser adaptadas a necesidades específicas. Una novedad científica en el polo de realidad virtual que aporta este trabajo es la distribución de la inteligencia entre los diversos elementos que forman parte de un entorno virtual, tanto los dinámicos como los estáticos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se hace una síntesis de las principales tecnologías a escala nacional e internacional vinculadas a nuestro campo de acción. Se aborda el tema de los modelos bioinspirados como técnica novedosa que sirvió de base a los comportamientos de locomoción, la teoría de agentes, además se hace una breve reseña sobre las más importantes bibliotecas de visualización existentes, biblioteca para la física ODE de reconocido prestigio en este campo así como también un algoritmo de Reubicación diseñado para utilizar de forma eficiente los recursos de hardware durante la animación de objetos dinámicos en entornos virtuales.

1.1 Modelos bioinspirados.

A lo largo de la historia ha sido la naturaleza fuente de inspiración en diversas áreas de la actividad humana por la capacidad que tienen los organismos naturales para enfrentar grandes problemas así como la habilidad de adaptarse y aprender del medio en la medida en que va evolucionando. Cuando se habla de IA resulta necesario hacer alusión a los diversos métodos de solución de problemas, los modelos bioinspirados se han ganado un espacio importante dentro de estos métodos. Técnicas conocidas dentro del mundo de la IA como las redes neuronales, algoritmos genéticos, colonias de hormigas, y bandadas de pájaros constituyen los ejemplos más representativos de los modelos inspirados en la naturaleza. La característica fundamental de ellos es su capacidad evolutiva. Se apoyan en una serie de atributos que caracterizan a los sistemas bioinspirados, como son su dinámica, flexibilidad, robustez, auto-organización, simplicidad en los elementos básicos y su descentralización. A continuación se muestran las características básicas de estos modelos.

1.1.1 Redes Neuronales Artificiales.

Una red neuronal artificial (RNA) es una técnica basada en el funcionamiento del cerebro

humano aunque no constituye una modelación en sí de este órgano. Se valen de la arquitectura del cerebro así como la manera que emplea para resolver los problemas. Una RNA está compuesta por un número de elementos denominados “neuronas” que se encuentran entrelazadas entre sí, para de esta manera fortalecer su funcionamiento. Este enlace contiene un peso, equivalente en el cerebro humano al enlace sináptico de nuestras neuronas que se fortalece, según expertos, cuando el individuo adquiere un conocimiento determinado. 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.

Las RNA contienen tres elementos fundamentales que la caracterizan, ellos son: la topología de la red, el modelo de neurona y el algoritmo de aprendizaje. La topología representa la manera en que se hallan organizadas las neuronas, las cuales se agrupan en unidades estructurales denominadas capas.

Existen actualmente varios tipos de topología. Las redes multicapa son de las más usadas en nuestros días para resolver diversos problemas. Cada neurona de la topología cuenta con una forma de respuesta que se expresa dentro del modelo de neurona, y cada modelo utiliza una función de activación para determinar la respuesta de la neurona (Prevot, 2008).

El estudio de la actividad cerebral ha arrojado resultados importantes dentro del mundo de las RNA, en este sentido la conexión entre las neuronas, el valor de activación de las mismas y la combinación entre ellos juegan un papel importante (Rabuñal, 2006). Existen varios tipos de modelos de neuronas, entre los que se tiene el modelo lineal, el lineal con umbral, modelo estocástico así como el modelo continuo que es 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.

Aprendizaje

El algoritmo de aprendizaje de la RNA tiene la responsabilidad como su nombre lo indica

de dotar de aprendizaje a la neurona, empleando en alguna medida la forma en que sucede en un sistema nervioso biológico, como es el establecimiento de nuevas conexiones, ruptura de otras, modelación de las actividades sinápticas entre otras. (Martín, 2001). El aprendizaje de las neuronas artificiales se traduce en encontrar el peso del enlace entre las mismas. Se conoce que las neuronas se auto-organizan, aprenden del entorno y se adaptan a él, comportamiento necesario para su supervivencia y desarrollo. Sin embargo el aprendizaje en las RNA a diferencia del de los humanos, no es continuo.

Se conoce la existencia de varios tipos de aprendizaje: el aprendizaje supervisado, el no supervisado, aprendizaje híbrido y el reforzado. (Trujillo, 2008). Una manera de entrenar a las RNA es haciendo uso de otro de los modelos bioinspirado “Los Algoritmos Genéticos”, para ello se construye un conjunto de patrones de formación, en este proceso el algoritmo genético se encarga de la obtención de los valores de los pesos de las conexiones, de forma tal que minimicen el error, garantizando al menos que sea menor al considerado aceptable.

Las RNA han sido y continúan siendo empleadas con buenos resultados en juegos y otras aplicaciones de simulación. (Rabuñal, 2006)

1.1.2 Algoritmos genéticos.

Otro de los modelos bioinspirados muy difundidos son los Algoritmos Genéticos (AG) inspirados en el proceso de evolución de las especies. Pueden considerarse métodos de búsqueda guiados de forma aleatoria. Están compuestos principalmente por individuos o cromosomas, una población que va a ir evolucionando a medida que pasa el tiempo, una función de aptitud o calidad del individuo, los operadores genéticos y los parámetros del algoritmo. Un cromosoma es una solución potencial a un problema y la población es un conjunto de soluciones posibles. La función de calidad es quien permite encontrar los mejores individuos dentro de la población para el próximo paso en el proceso evolutivo empleando para ello los operadores genéticos. Varias técnicas inspiradas en este modelo han incorporado variantes que garantizan mejores resultados, la premisa de los AG radica

en encontrar a medida que se va avanzando un individuo con mejores características que el que se tenía inicialmente. Para garantizar esto algunas implementaciones hacen uso de un teorema denominado *teorema de los esquemas* que establece que la calidad promedio de los individuos más avanzados en el proceso evolutivo sea mayor a los anteriores. Otras siguen además la premisa de conservar el mejor individuo de la población. Algunas de estas variantes han sido igualmente aplicadas en entornos virtuales para lograr comportamientos inteligentes de los elementos, sobre todo para encontrar elementos mejor adaptados al entorno, es decir que tengan comportamientos más realistas a partir de valores de las variables que representan sus características.

Entre los operadores genéticos se hallan los operadores de selección que cumplen la función de determinar cuales individuos usar para generar la nueva población que no es otra cosa que seleccionar los padres de la nueva población, el operador de cruzamiento se encarga de determinar cómo mezclar estos individuos y el operador de mutación, con el cual se crean nuevos individuos a través de la mutación, que consiste en cambiar determinados genes en la población para crear una nueva población y así avanzar en el proceso evolutivo. (Marrero, 2007). Para realizar estos operadores se emplean distintos parámetros necesarios dentro del AG, como son el tamaño de la población, el número de generaciones que se desea realizar, así como dos probabilidades, la probabilidad de cruzamiento y la probabilidad de mutación que son las que determinan la aleatoriedad del proceso.

Con todos estos elementos se inicia el proceso evolutivo, para ello el AG 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. A continuación se muestra el proceso descrito a través del pseudocódigo del AG.

Algoritmo Genético

```
1:  Generar una población inicial.
2:  Computar la función de evaluación de cada individuo.
3:  Mientras No Terminado Hacer
4:      Producir nueva generación
5:  Para Tamaño población (2) Hacer
6:   Ciclo reproductivo:
7:    Seleccionar  dos individuos de la anterior generación,
8:    Para cruzar (probabilidad de selección proporcional a la func  de evaluación del individuo).
9:        Cruzar con probabilidad obtenida los dos individuos descendientes.
10:       Mutar los dos descendientes con probabilidad obtenida.
11:       Computar la función de evaluación de los descendientes mutados.
12:       Insertar los dos descendientes mutados en la nueva generación.
13:  fin Ciclo reproductivo.
14:  Si la población ha convergido Entonces
15:      Terminado:=verdadero
16:  Sino
17:      volver al paso 6:
18:  fin Para
19:fin Algoritmo.
```

Figura 1: Pseudocódigo Algoritmo Genético.

1.1.3 Colonias de hormigas.

Uno de los más recientes modelos bioinspirados son las colonias de hormigas a partir de las cuales se ha definido la técnica computacional “Optimización basada en colonias de hormigas”, en inglés *Ant Colony Optimization* (ACO) (Charles, 2008). Los elementos de este modelo se encuentran muy relacionados, siendo su base la inteligencia colectiva. Las colonias de hormigas son sistemas que independientemente de la simplicidad de sus elementos se caracterizan por poseer una buena organización social. Como criterio importante de los comportamientos de hormigas se tiene que estos animalitos por separados no son capaces de resolver sus problemas, sin embargo en unión a otros elementos comunes pueden resolver la dificultad que las ocupa, por lo que el basamento de dicho modelo radica en comportamientos grupales y no individuales de los elementos (Pfeil, 2006). Este tipo de reglas que emplean se caracterizan por dos aspectos fundamentales: la simplicidad y la robustez. 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.

En la figura 2, tomada de (Pfeil, 2006) se muestra como inicialmente las hormigas (a) al encontrarse con un obstáculo, en este caso un puente toman distintas direcciones, pero a medida que pasa el tiempo (b) se van organizando de modo tal que alcanzan el camino más corto para trasladarse del origen al destino.

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 ello cada hormiga artificial va a contar con una memoria y se va a desarrollar en un entorno discreto, o sea se va a ir actualizando su posición en cada ciclo del proceso. La ACO se modela como un grafo en que como es común en este tipo de representación se cuenta con un grupo de nodos, que en este caso representan los distintos estados por donde pasarán las hormigas.

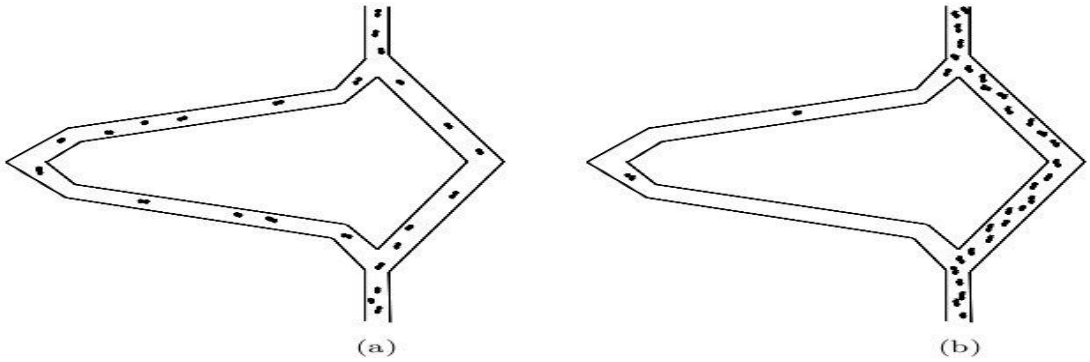

Figura 2: Comportamiento de la colonia de hormigas en su búsqueda del camino mínimo para llegar al destino

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 cual será el próximo nodo a visitar a partir de lo cual aparece la regla de selección en la cual se basa este modelo. Las aristas

van a tener asociado un peso que representan el nivel de feromona de las hormigas. Mientras mayor sea el peso de la arista 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 se incrementen. De esta manera el modelo de búsqueda utilizado por la ACO se representa en el siguiente algoritmo:

Proceso general de búsqueda de ACO

```
1:  Inicializar los valores de feromona.
2:  Nciclo <- 1
3:  Repetir
4:      Algoritmo de Optimización de Colonias de Hormigas.
5:      Nciclo <-  Nciclo +1
6:  Hasta fin.
```

Figura 3: Proceso general de búsqueda ACO.

El proceso general se detalla a continuación. (Pfeild, 2006).

En el proceso general de búsqueda del Algoritmo ACO, el primer paso es la inicialización del nivel de feromona asociada a todas las aristas, es decir el peso de cada arista, después habrá un ciclo, en cada iteración del ciclo todas las hormigas van a encontrar una solución. Al concluir, las soluciones encontradas por cada hormiga se evalúan y se procede a la actualización del nivel de feromona asociada a cada arista. De tal manera solo se va a modificar las feromonas asociadas a las aristas de la solución, resultando las aristas 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.

Algoritmo Optimización de Colonias de Hormigas

```
1:  Repetir
2:  Si la cantidad de hormigas < cantidad máxima de hormigas hacer
3:      Crear una nueva hormiga
4:      Cambiar estado inicial
5:  Fin Si
6:  Para todas las hormigas  hacer
7:  Determinar los estados vecinos factibles (considerar las hormigas de los estado visitados)
8:  Si no se encuentra una solución factible en un estado vecino  hacer
9:      Eliminar hormiga
10:     Si usamos la actualización de la feromona por retraso  entonces
11:         Evaluar la solución
12:         Depositar feromona en los estados usados como camino de la solución
13:     Fin Si
14:     Sino
15:     Seleccionar estocásticamente el estado vecino factible (dirigido por la memoria de las
        hormigas, la concentración de feromona en los estados vecinos y la heurística local)
16:     Si usamos la actualización de la feromona paso a paso  entonces
17:       Depositar feromona en la arista usada
18:     Fin Si
19:   Fin Si
20: Fin Para
21: Evaporar la feromona
22: Hasta que se satisfaga el criterio (se encontró solución satisfactoria)
```

Figura 4: Algoritmo de Optimización de Colonias de Hormigas.

1.1.4 Bandada de pájaros.

El modelo original de Optimización basada en Partículas como una de las técnicas correspondiente al modelo "Banda de pájaros", popularizada *Particle Swarm Optimization* (PSO) según sus siglas en inglés, se inspiró en el comportamiento social y biológico de los organismos, específicamente en las habilidades de algunos grupos de especies de animales como las bandadas de aves (Bratton, 2007).

El modelo inicial radica en el estudio del comportamiento de estas bandadas en el entorno natural. En 1987 se creó un modelo computacional denominado "Boids" que se basó en el comportamiento de una bandada de pájaros en el que se pudo identificar tres fuerzas locales: Una con el objetivo de evitar una colisión entre los pájaros, algo muy natural que ocurre en estos grupos de animales a pesar de volar cientos de ellos al mismo tiempo, otra fuerza para establecer la velocidad de encuentro, con el objetivo de mantener la misma velocidad de sus demás compañeros de banda y la última fuerza está relacionada con la estructura de la banda para lograr un vuelo centrado con el objetivo de que los

pájaros se muevan hacia el centro de la banda.

El modelo artificial de PSO comparte algunos elementos comunes con algoritmos genéticos (Settles, 2005); ambos parten de una población inicial donde cada individuo representa una posible solución al problema y la banda en conjunto sería el equivalente a la población de cromosomas, de forma similar también cuentan con una función de evaluación que determina la mejor solución potencial y además ambos repiten el mismo conjunto de procesos para una determinada cantidad de tiempo.

La bandada de pájaros va a moverse por el espacio 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 banda en su conjunto. Por lo que el movimiento de cada individuo va a estar definido por el extremo que históricamente ha obtenido de manera individual adicionado a la mejor posición histórica de la banda, que determinan la nueva posición que va a obtener el individuo.

La función de calidad permite determinar la calidad de la nueva posición obtenida por el pájaro, siendo la interacción del individuo con el resto de la banda lo que decide cual va a ser la nueva posición de cada elemento. Por lo que se define como principios básicos de la técnica PSO:

1. Cada una de las partículas tiene una posición y velocidad en el espacio de búsqueda.
    a. La posición determina la solución candidata.
2. Cada elemento posee una medida de calidad.
    a. Búsqueda heurística.
3. Cada partícula o individuo puede relacionarse con un grupo de vecinos.
    a. Cada partícula conoce el valor de calidad y posición de cada uno de sus vecinos.
    b. Emplea los principios de “comparar” e “imitar”.
4. Cada uno de los individuos aprende ajustando su posición y velocidad.
    a. Impulsado por su mejor posición.
    b. Atraído por la mejor posición de la banda.

PSO contiene un grupo de parámetros necesarios para resolver sus problemas; cuenta con un número de partículas que no es otra cosa que el tamaño de la banda, un número de generaciones que significan las soluciones encontradas hasta el momento, un valor de peso relacionado con la inercia, que es el valor que afecta la velocidad del individuo y dos razones, razón de aprendizaje cognitivo y de aprendizaje social.

Teniendo en cuenta las características y elementos de esta técnica, se define el algoritmo general de PSO.

Algoritmo Optimización basada en Parctículas

```
1:  Inicializar aleatoriamente la población:
2:    Valores de cada partícula.
3:    Sus velocidades.
4:    Mejores puntos localizados para cada partícula.
5:    Mejor posición de la banda.
6:  Repetir
7:    Para cada partícula:
8:      Calcular el nuevo valor de velocidad  Vi (t+1) y limitarla a [-Vmax, +Vmax].
9:      Actualizar cada partícula Xi:
10:       Para cada Xi:
11:           Evaluar Xi.
12:           Actualizar mejor posición de cada partícula Xpbest(i)
13:       fin Para.
14:   fin Para.
15:   Actualizar la mejor posición de la banda Xgbest
16: Hasta la condición de terminación.
17: fin Algoritmo Optimización basada en Parctículas.
```

Figura 5: Algoritmo Optimización basada en Partículas.

La optimización basada en partículas ha tenido aplicaciones exitosas en diversos campos (Pfeild, 2006), entre los que se encuentran los problemas de funciones multiobjetivo (Santana, 2006), entrenamiento de redes neuronales artificiales (Heng, 2006), para encontrar parámetros de sistemas borrosos, así como la solución de problemas de satisfacción de restricciones. Resulta muy eficiente en problemas de optimización continuos a diferencia de la optimización de colonias de hormigas, que es eficaz para resolver problemas discretos.

1.2 Comportamientos de locomoción. Avances en el mundo.

El término “comportamiento” está asociado a varios significados. Se puede vincular a la acción compleja de humanos y animales basados en su voluntad e instinto. También en gran medida a la acción previsible de simples sistemas mecánicos o la acción compleja de sistemas caóticos (Reynolds, 1999).

Los comportamientos de locomoción popularizado *“Steering Behavior”* en inglés, constituye otra de las técnicas derivadas del modelo bandada de pájaros igual que la Optimización basada en partículas. Los *Steering Behavior* tienen su basamento en el comportamiento de los elementos dentro de un entorno simulado, a partir del empleo de una fuerza sobre ellos con el uso de sencillas fórmulas matemáticas (Coca, 2007).

La idea de esta técnica surge por el científico Craig Reynolds en 1987, aunque se concreta realmente doce años más tarde por el mismo investigador. En aplicaciones de realidad virtual y multimedia el término “behavior” se asocia a la animación de los elementos y el comportamiento que poseen los elementos autónomos dentro del entorno. El comportamiento de estos elementos se dividen en tres capas (Reynolds, 1999); la primera, **selección de acción** vinculada con la estrategia, objetivos y planificación de los elementos. La segunda es el **comportamiento** que se emplea en la determinación de la ruta a seguir y la última se le conoce como **locomoción** que no es otra cosa que la animación y articulación de los elementos.

La locomoción es la capa que controla las señales para la capa de comportamiento dentro del movimiento de los elementos, movimiento dirigido constantemente por modelos basados en la física como son la interacción del impulso y las limitaciones de la fuerza que es aplicada a los cuerpos. El movimiento de los elementos emplea el modelo físico para lograr un alto nivel de realismo en la animación de los personajes.

1.2.1 Evolución de los comportamientos de locomoción.

Los comportamientos de locomoción para elementos autónomos se encuentran relacionados con diversos campos dentro de la informática, entre ellos; las máquinas autónomas y las teorías de control. Los comportamientos de locomoción han jugado un

significativo papel en el mundo de la robótica. Entre las grandes figuras del campo se encuentra Rodney Brooks quien construyó un sistema robótico inspirado originalmente en comportamiento de animales (Brooks, 1985), posteriormente el investigador Ron Arkin aplicó los *steering behavior* para sus trabajos con robots móviles (Arkin, 1987), (Arkin, 1989), (Arkin, 1992). David Zeltzer quien fue pionero en trabajos de animación basados en inteligencia artificial popularizó la idea "nivel de tarea" de la especificación del movimiento (Zeltzer, 1983), (Zeltzer, 1990). Otra figura que destacó fue Mónica Costa, quien trabajó en animación de comportamientos basado en agentes (Costa, 1990). En 1988 se empleó los comportamientos de locomoción para evitar obstáculos (Reynolds, 1988). Un trabajo interesante donde fue usado los comportamientos de locomoción fue el modelo creado por Xiaoyuan Tu con aportes en el área de la biomecánica, locomoción, percepción y comportamiento de peses (Tu, 1994). Más tarde Dave Pottinger estableció la discusión del tema vinculado a los comportamientos y coordinación de grupos de personajes en juegos (Pottinger 1999).

1.2.2 Tipos de comportamientos de locomoción

Existen diferentes tipos de comportamientos para la locomoción de los elementos que se mueven dentro de un entorno simulado, que van desde algo tan sencillo como la búsqueda de un objetivo hasta cruzar una calle en una ciudad u otros muchos más complejos**.**

Los comportamientos de locomoción se encuentran divididos principalmente en comportamientos individuales y grupales de los elementos, algunos de los cuales les exponemos a continuación (Buckland, 2005).

"Búsqueda" comportamiento que basa su funcionamiento en la búsqueda de un objetivo. Plantea la locomoción de los elementos de manera que sus velocidades sean radialmente alineadas.

"Huir" se expresa contrario al comportamiento de "Búsqueda". El vector relacionado con la velocidad deseada de los elementos se representa en dirección opuesta al objetivo, es

decir su función es huir de un elemento dado.

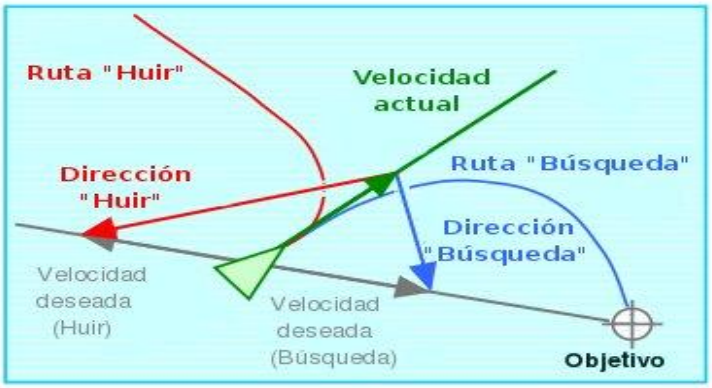

Figura 6: Comportamiento "Búsqueda" y "Huir"

La principal diferencia entre estos dos comportamientos radica en la dirección de la fuerza que se le aplica a los elementos para su locomoción. Esta dirección se calcula a partir de la velocidad actual del individuo y su posición con respecto al objetivo ya sea para acercarse como para alejarse de él. La posición del objetivo se representa como un vector localizado entre el objetivo y el elemento que no es otra cosa que la velocidad deseada. A partir de la diferencia entre la velocidad deseada y la actual se obtiene el valor correspondiente a la dirección de la fuerza necesaria para el movimiento del elemento en cuestión (ver figura 6). Si el comportamiento a seguir es la búsqueda de un objetivo se resta a la velocidad deseada del elemento su actual velocidad, si por el contrario lo que se quiere es huir entonces la fuerza para la locomoción se calcula a partir de la resta de la velocidad deseada del individuo a la velocidad actual de mismo.

"Perseguir" se comporta de manera similar al comportamiento de "Búsqueda" con la excepción que en este comportamiento el objetivo se representa como otro personaje en movimiento. Necesita para su efectividad poder predecir la siguiente posición del objetivo.

"Evadir un obstáculo" es el comportamiento que como su nombre lo indica radica en la acción de evadir un obstáculo determinado. En este sentido es conveniente aclarar que existe una marcada diferencia entre "huir" y evadir un obstáculo; en el primer caso implica que siempre el individuo va a alejarse de una ubicación dada, en cambio en el segundo

caso solo se realiza esta acción cuando se encuentra un obstáculo frente al elemento durante el recorrido del mismo.

"Esconderse" comportamiento algo similar al "huir", pero en este caso el individuo trata de esconderse de otro elemento usando para ello los obstáculos que le puedan servir de refugio. Es muy práctico en entornos simulados donde se necesite hacer uso de este tipo de comportamiento, como puede ser el caso de videojuegos donde se desarrolle un enfrentamiento, digamos por ejemplo una lucha entre varios contrincantes.

Comportamientos grupales

En el mundo real los seres vivos se encuentran constantemente relacionados entre si, hasta los animales más insignificantes muestran comportamientos grupales. Es imposible la vida sin la relación entre individuos sean o no de la misma especie. Es por ello que entre los comportamientos de locomoción se tiene además este tipo de comportamiento. Estos comportamientos sirvieron de base para definir el modelo de bandadas de pájaros. Se basan en tres comportamientos básicos que se integran para lograr este movimiento. La **Cohesión**, permite agrupar a varios elementos, logrando la atracción entre un grupo de individuos, permite un comportamiento coherente de los elementos. La **Separación** plantea lo contrario, mantener cierta distancia entre los elementos, evita el apiñamiento de los individuos y la **Alineación** mantiene los elementos dirigidos hacia un mismo objetivo, es decir la velocidad de todos en la misma dirección y sentido.

1.2.3 Combinación de comportamientos.

Raramente suelen usarse comportamientos aislados. En reiteradas ocasiones es necesario emplear varios de ellos al mismo tiempo para obtener un determinado objetivo, como en el caso de los comportamientos grupales. La combinación de comportamientos conforma una estructura compleja, por lo general no tiene mucho sentido verlos de forma individual, similar al funcionamiento del organismo humano, no nos basta con el trabajo aislado de los órganos que conforman nuestro cuerpo, los necesitamos a todos para vivir. Así también funciona con los entornos simulados. Un ejemplo de ello lo evidencia (Wang,

2004) cuando emplea la intercepción de comportamientos para regular las interacciones de los vehículos dentro de un entorno virtual de modo tal que durante su trayectoria los mismos se muevan por una ciudad respetando algunas señales.

En lo primero que se puede pensar para combinarlos es en una sumatoria de las fuerzas, pero esto puede traer resultados no deseados. Cada elemento tiene una lista de comportamientos que debe respetar en un momento determinado. Cada comportamiento es una clase y cada elemento tiene una lista de objetos de esa clase. Existe un método donde se calcula la fuerza resultante de llamar al método "calcular" de cada *Steering Behaviors* teniendo en cuenta que en los elementos se necesita una lista de comportamientos. El comportamiento final se calcula sumando los valores de ir multiplicando el valor de cada comportamiento por un peso. Un importante inconveniente de este tipo de cálculo de la suma es que si tiene muchos comportamientos el agente, puede ser muy costosa la ejecución de todos ellos, pero además es complejo determinar un peso realmente factible a cada comportamiento. Otro problema serio es que en algunos casos puede existir conflicto entre las fuerzas lo que podría conllevar a una violación importante de los elementos en el entorno.

Varios de estos inconvenientes se pueden atenuar definiendo una prioridad a los comportamientos, no para calcularlos todos sino ir calculando y adicionando a la variable "fuerza total" hasta llegar a que esa fuerza total acumulada sea igual a la fuerza total admisible por el elemento. Con esto se logra que los principales comportamientos se analicen, todo esto se realiza en un instante de tiempo, porque al siguiente, como ya será menor la fuerza que aplican los más importantes, se irán analizando los menos importantes.

Otra variante para lograr un análisis mucho más realista, es hacer una diferencia entre comportamientos prioritarios y otros no prioritarios, si la combinación de los prioritarios determina una fuerza es la que se analiza y no se analizan los otros posibles comportamientos, en el próximo instante de tiempo les tocará el turno a ellos, esto permite mantener mucho más estable el movimiento.

Estas variantes de combinar los comportamientos permiten además y de manera especial

la distribución de la Inteligencia Artificial por el entorno. (Ver epígrafe 2.1)

1.2.4 Limitaciones de los comportamientos de locomoción.

La combinación de comportamientos de locomoción es un proceso difícil y en ocasiones trae aparejado algunas dificultades (Ben, 2003). Los diversos comportamientos simples que conforman en su conjunto a un comportamiento complejo basan su conducta independientemente de los demás. Esto puede causar contradicción entre ellos y hasta contrarrestar los efectos de los demás comportamientos, provocando la ineficiencia del comportamiento complejo. La prioridad en la que se basa la combinación de comportamientos según (Reynolds, 1999) trata de resolver estos problemas, sin embargo puede causar que las limitaciones de algunos comportamientos sean violadas. Es común que este tipo de limitantes sean también tratadas en otros comportamientos de locomoción. Sin embargo en ocasiones estas medidas no resultan suficientes, por ejemplo para evitar que durante el movimiento de un individuo este pueda estrellarse contra un obstáculo puede provocar que otro elemento dentro del entorno sea quien impacte contra el obstáculo ya que en el contexto espacial el comportamiento que tuvo su origen no se tuvo en cuenta con anterioridad. Una posible solución a este problema pudiera estar en el uso de los efectos que cada acción puede causar en la capa activa. En un entorno muy dinámico, la planificación de cada paso desde el principio hasta alcanzar un objetivo es un despilfarro de recursos, porque lo más probable es que algunas o todas las partes de una ruta calculada previamente se invaliden en un futuro. La utilización combinada de otros modelos (bioinspirados o no) para lograr comportamientos, ayudaría a mejorar el resultado final del movimiento de los agentes creíbles. Ya se han desarrollado aplicaciones con redes neuronales y algoritmos genéticos, resta adaptarlas a las características de la aplicación que en este trabajo se presenta.

1.3 Comportamientos de locomoción. Polo de Realidad virtual.

En la Universidad de las Ciencias Informáticas, se presta especial interés al desarrollo de

diferentes técnicas de Inteligencia Artificial, específicamente en la Facultad 5 en el Polo de Realidad Virtual se han realizado aplicaciones que utilizan los *Steering Behavior* como técnica novedosa para el desarrollo de videojuegos y simuladores, lo cual ha despertado gran interés por la utilización de esta técnica, ya que brinda la posibilidad de que las personas que interactúan con este medio perciban cierto grado de realismo e inteligencia en los elementos que se mueven dentro de un entorno simulado. En el curso 2007-2008 se desarrolla en el polo de Realidad virtual, una tesis de grado que hace uso de los comportamientos de locomoción para agentes, titulada *"Comportamientos de autos en pistas de carreras basados en Steering Behaviors para el videojuego "Rápido y Curioso" .* En este trabajo se planteó la creación de un módulo de *Steering Behaviors,* el cual está conformado por tres comportamientos de locomoción que juntos lograron el comportamiento inteligente de los carros autónomos en un entorno de pistas de carreras, estos fueron: *Behavior Angle*, *Behavior Brake* y *Behavior Accelerator*. El primero fue usado en el cálculo del ángulo de giro que debía tomar el vehículo para moverse a un punto específico, *Behavior Accelerator* fue empleado para calcular la posición del pedal de aceleración del carro y el último de ellos se utilizó para calcular el valor de la fuerza de frenado aplicada al carro en un instante determinado (Sánchez, 2008).

Con la incorporación de estos comportamientos se logró proveer de un comportamiento inteligente a los carros autónomos del videojuego "Rápido y Curioso", haciendo este juego más interesante e interactivo. Este trabajo demostró que el control de autos dentro de un videojuego con esta técnica de Inteligencia Artificial resulta verdaderamente eficiente.

En este mismo año se diseña e implementa un sistema titulado *"DLL Intelligent Soccer Juego de fútbol para jugadores virtuales",* que se basó en la obtención de una aplicación que fuese capaz de desarrollar un partido de fútbol sin intervención del usuario, o sea, que los equipos de forma inteligente se apoyarán en una estrategia previamente programada por el usuario, para elegir la mejor jugada en cada momento. En esta aplicación se implementó un subsistema de comportamiento de locomoción que contó con comportamientos conocidos como el *Seek*, *Stop*, *Arrive*. Este trabajo permite recopilar estrategias de juego que posteriormente se pueden utilizar para ser acopladas a cualquier entidad en un entorno virtual que se desee tenga el mismo comportamiento inteligente.

(Vela, 2008).

También se construyó un módulo de comportamiento inteligente para autos, con nombre "*Módulo para el comportamiento autónomo de autos en un Entorno Virtual urbano"* (Saurín, 2008). Que permitió que estos se movieran de forma autónoma, respetando las leyes del tránsito. De esta forma se lograba simular el comportamiento de choferes en el mundo real con sus aciertos y desaciertos en el arte de manejar. Este módulo contiene todos los elementos para manipular la manera en la que se deben comportar los vehículos y por donde deben realizar su desplazamiento. Lográndose que estos tuvieran cierta autonomía. Para ello se crearon comportamientos que se acoplaran a las situaciones propias del entorno como el comportamiento "Movimiento Libre"**,** que provee a los vehículos de la escena el movimiento básico con el que se desplazarán, el comportamiento "Seguimiento" que permite que los autos no colisionen entre sí, el "Intersección" encargado de guiar a los vehículos cuando se encuentren frente a una intersección y el "Paso Peatonal" que tuvo la responsabilidad de guiar a los vehículos cuando estos se encontraran frente a un paso peatonal. Estos comportamientos trabajan directamente con la velocidad, aceleración y distancia para desarrollar sus funciones.

Otro trabajo interesante sobre el tema desarrollado en la universidad fue el "*Módulo de algoritmos de locomoción con múltiples Steering Behaviors"* que contiene implementados diversos algoritmos de locomoción. En la investigación realizada se abordaron diferentes conceptos necesarios para el desarrollo y funcionamiento de estos algoritmos. Brindó la posibilidad de manejar los diferentes comportamientos de locomoción que se utilizan en software de simulación y de trabajar los comportamientos grupales, lo que permite que una entidad pueda tomar decisiones teniendo en cuenta un mayor número de elementos en su entorno. Se creó un comportamiento encargado de controlar y manipular los demás comportamientos. Entre los comportamientos usados se encuentran algunos tan conocidos como el *Flee, Seek, Hide, Arrive y Alignment*, teniendo como responsabilidad cada uno de ellos el retorno de una fuerza, la cual se encarga de dirigir la entidad hacia un objetivo determinado. Este módulo facilita la programación de la parte inteligente de los juegos, reduciendo además el tiempo de desarrollo de estas aplicaciones. (Falcón, 2008).

1.4 Agentes.

La IA juega un papel importante en lograr entornos virtuales realistas, por lo que ha dedicado esfuerzos a la investigación de novedosas técnicas y teorías para llevar a cabo sus propósitos, una de ellas es la relacionada con los "agentes". Numerosos científicos e investigadores de la rama se han expresado al respecto, sin encontrar un punto de coincidencia en sus criterios. Diccionarios de la lengua española en su primera acepción lo definen como: "*Persona que trabaja en una agencia prestando determinados servicios*", por lo que si pensamos en este término en el mundo computacional de forma genérica, podríamos sustituir "persona" por "entidad", la frase "trabaja en una agencia" por "actúa en un entorno" y "prestando determinados servicios" por "transformando dicho entorno" quedando la definición luego de algunos ajustes de la siguiente manera: "e*ntidad que actúa de manera autónoma en un entorno, transformándolo mediante la interrelación con otras entidades* ".

La teoría de agentes permite tratar de una forma eficiente la construcción de sistemas inteligentes ambiciosos para ser aplicados a diversos campos. En este sentido varios son los criterios positivos e importancia que se le ha atribuido. Por ejemplo en ocasión del Congreso Internacional de Inteligencia Artificial celebrado en Estocolmo en 1999 el Dr. Nicholas Jennings durante su discurso pronunciado luego de recoger el premio al mejor investigador novel del evento expresó: "*Los agentes constituyen el próximo avance más significativo en el desarrollo de sistemas y pueden ser considerados como la nueva revolución en el software"* Tal es su nivel de relevancia que es considerado como un nuevo paradigma de programación, lo que significa una forma nueva de llevar al lenguaje de programación, el mundo que nos rodea.

Características definitorias.

Los agentes poseen varias características, pero sin lugar a dudas existen tres que los distinguen (Coca, 2008).

Activos: Los agentes poseen objetivos propios que cumplir, deben ser capaces de llevarlos a cabo independientemente de cualquier anormalidad o situación que se

presente en el entorno.

Reactivos: Son capaces de responder ante los distintos cambios y eventos del entorno sin que ello afecte sus propios objetivos.

Sociales: Pueden comunicarse con otros agentes del entorno mediante algún lenguaje de comunicación.

Otras características.

Existen otras características que en la literatura se suelen atribuir a los agentes en mayor o menor grado para resolver problemas particulares y que han sido descritos por varios científicos (Ferver, 1999). Algunas de ellas las presentamos a continuación.

Continuidad temporal: Es considerado un agente un proceso infinito, de forma tal que constantemente se encuentre desarrollando su función.

Autonomía: Posee la capacidad de actuar basado en su experiencia y habilidad para adaptarse al entorno.

Racionalidad: Realiza lo correcto, a partir de la información que recibe del entorno.

Movilidad: Tiene la facultad de moverse por un entorno dado.

Veracidad: Un agente no comunica información falsa a propósito.

Los agentes creíbles como uno de los tipos de agentes deben respetar estas características como se verá en el (epígrafe 2.3).

1.5 Bibliotecas de visualización.

Un Motor Gráfico conocido también por su término en inglés (Graphic Engine), constituye el núcleo de un videojuego o de una aplicación que utiliza gráficos en tiempo real. Es la parte del programa que controla, gestiona y actualiza los gráficos 2D y 3D. Permite la abstracción de la plataforma y se apoya en bibliotecas gráficas como las conocidas “OpenGL” o “DirectX”. Se encarga de la visibilidad, el mapeado, la gestión de mallas 3D, entre otros aspectos. (Tirado, 2007). Su principal responsabilidad radica en la realización

de tareas de bajo nivel tales como comunicarse con el adaptador gráfico, administrar la escena y transformar la geometría del mundo 3D. Dentro de los motores de código abierto más importantes y utilizados en la actualidad se encuentran **"*Object Oriented Graphics*"** y **"*Graphics Three Dimensional*"**.

1.5.1 Graphics Three Dimensional.

Graphics Three Dimensional conocido por sus siglas (G3D) es un robusto motor 3D usado en juegos comerciales, investigaciones y simuladores. Brinda soporte para el trabajo con texturas, imágenes, modelos 3D, ventanas, efectos especiales y redes. Está escrito en el lenguaje de programación C++ y licenciado bajo Distribución de Software *Berkeley* (BSD) según sus siglas en inglés, que es una licencia de software libre muy popular en nuestros días. Soporta *"rendering"* en tiempo real, *"rendering"* desconectado y cálculos de propósito general basado en las Unidades de Procesamiento Gráfico o como se conoce en idioma inglés *"Graphics Processing Unit"* (GPU). Proporciona un conjunto de rutinas y estructuras comunes que son necesitadas en todas las aplicaciones gráficas. (Skilton, 2008). Posibilita que bibliotecas gráficas de bajo nivel como OpenGL sean más fáciles de usar sin limitar su funcionalidad.

Algunas características de G3D son útiles para cualquier programa, sin importar si este ejecuta cálculos 3D o corre sobre un procesador gráfico. Presenta una arquitectura robusta y optimizada, permitiendo integrar otras bibliotecas de forma sencilla y rápida; convirtiéndose en una excelente herramienta para el desarrollo de videojuegos y simuladores (McGuire, 2007). En la figura 7 podemos observar la estructura de este motor de visualización.

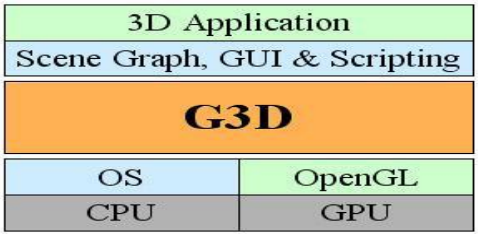

Figura 7: Estructura del motor G3D

Cuando se usa G3D como biblioteca utilitaria no es necesario el *"framework"* para la "*Graphical User Interface" (GUI)*. Para soportar usos utilitarios la biblioteca se divide en dos partes: G3D.lib y GLG3D.lib. Toda la manipulación de eventos, ventanas y código OpenGL está en la porción GLG3D.lib. Esto significa que se puede usar G3D.lib sin estar atado a OpenGL o al establecimiento de alguna rutina de manipulación de eventos. Su versión 7.0 constituye una completa solución gráfica para la construcción de juegos 3D y simuladores (Lombera, 2008). Constituye una excelente biblioteca que se ajusta fácilmente a las necesidades de los programadores. Presenta un poderoso soporte de red con el que encapsula todos los *"sockets"* de bajo nivel. Soporta envío de paquetes usando los dos tipos de protocolos, TCP y UDP. A continuación se exponen las características fundamentales de esta biblioteca gráfica en la Tabla 1 (Lombera, 2008).

Características de G3D	
Autor	Morgan McGuire
API Grafica	OpenGL,DirectX
Sistema Operativo	Windows, Linux
Ultima Versión	Versión 7,0
Animación	KeyFrame
Herramientas	Visualizador de Modelos Matriz n*m Optimizada Configuración de Ficheros

Tabla 1: Características de G3D Engine.

1.5.2 Object Oriented Graphics Rendering Engine.

La biblioteca Object Oriented Graphics Rendering Engine (OGRE) es una biblioteca gráfica de *"rendering"* 3D multiplataforma, escrita en el lenguaje de programación C++ haciendo uso del paradigma orientado a objeto. Fue diseñada para hacer más sencillo el trabajo de los desarrolladores de juegos y aplicaciones que utilizan dispositivos gráficos. Está construida de forma que no se compromete con una API en específico, soporta tanto el uso de DirectX como de OpenGL. Es un proyecto de código libre bajo la licencia Pública General Reducida de GNU conocida por sus siglas en inglés (LGPL), su uso es gratuito y con pocas exigencias.

Con OGRE se pueden crear juegos y aplicaciones que requieran gráficos tridimensionales con calidad, debido a que su interfaz es simple, intuitiva y fácil de usar, diseñada para que requiera poco esfuerzo el *"rendering"* de escenas, este no trae soporte nativo para sonido ni física pero existen módulos especialmente diseñados que permiten añadir estas funcionalidades. Brinda soporte para texturas PNG, JPEG, BMP, DDS y DXT/S3TC, así como texturas variables en tiempo real. OGRE no está diseñada exclusivamente para la creación de videojuegos, se puede utilizar también para hacer aplicaciones de simulación, corporativas o de investigación. Utiliza una jerarquía de clases flexibles, puesto que no supone qué tipo de juegos o demos se van a realizar. Esto permite diseñar complementos para especializar la organización de la escena.

Su arquitectura es flexible, se basa en *"plugins",* que brindan la posibilidad de extender sus funcionalidades. Suministra una serie de técnicas en las que se puede observar la calidad de los gráficos que es capaz de manejar. Existen muchas extensiones que añaden funcionalidad a la biblioteca, como las *"Entity's"* que representan todos los objetos que pueden representarse por una malla 3D, los *"SceneNodes"* que realizan un seguimiento de la ubicación y orientación de todos los objetos que se le atribuyen y los *"SceneManagers"* son los encargados de seguirle la pista a todos los objetos que se dibujan por pantalla. Con OGRE se han desarrollado muchas aplicaciones, entre las que además de videojuegos, se incluye software educativo e infantil, juegos serios y demostraciones técnicas. Ejemplo de esto son los simuladores *"Pacific Storm"* y *"FirstAid Sim"*. (Tirado, 2007).

Aunque la mayoría de las veces es utilizada para la creación de videojuegos, OGRE no es propiamente una biblioteca de juego. Fue diseñada principalmente para proveer a los programadores con una solución de primera para gráficos, para aplicar otras características, como sonido, inteligencia artificial, colisión, leyes físicas entre otras. Su principal ventaja radica en su generalidad, ya que puede usarse para todo lo que se desee crear: juegos, simulaciones, aplicaciones de negocios, entre otros. Su diseño es coherente, la documentación detallada y consistente. Posee algunas debilidades como es el caso del soporte de red lo cual es muy importante hoy en día. (McGuire, 2007). Sus principales características las podemos constatar en la tabla 2. (Lombera, 2008).

Características de OGRE.	
Autor	Steve Streeting
API Grafica	OpenGL,DirectX
Sistema Operativo	Windows,Linux,Max OS
Lenguaje de Programación	C/C++
Arquitectura	Arquitectura de Plugins
Física	Incluye acoples para (ODE)
Superficies	Splines

Tabla 2: Características de la biblioteca "OGRE"

1.5.3 SceneToolKit.

Debido a la necesidad de la UCI de contar con una biblioteca propia para el desarrollo de diversas aplicaciones de Realidad Virtual surge la SceneToolKit (STK) que agrupa las funcionalidades comunes a cualquier sistema de realidad virtual, facilitando así el trabajo a los programadores de juegos y simuladores a través de la reutilización de código, desarrollado por miembros de los proyectos del polo de Realidad Virtual de la facultad 5.

Permite además de la visualización de los entornos sintéticos la aplicación de leyes físicas y matemáticas, animaciones, usando las bibliotecas gráficas OpenGL y DirectX, sobre plataforma Windows y Linux. Se retroalimenta de las necesidades de los programadores que la utilizan, así como de sus investigaciones. La STK deja de ser un "motor gráfico", para convertirse en un "motor de juegos", debido a que se comienzan a incorporar elementos físicos. Se encuentra en desarrollo un formato propio de fichero para almacenar la información de los entornos virtuales, el "stk". Posee módulos de comunicación por redes, audio, física, seguimiento de terreno, *"shaders"* e interfaz gráfica de usuario. Su arquitectura se divide en tres capas: "*Engine"* donde se encuentra el procesamiento importante de carga y manejo de objetos; "*Renderer"* donde se define la biblioteca gráfica con la cual se va a dibujar los entornos y la capa "*Application"* que se

encarga del manejo de eventos del sistema operativo específico. (Jiménez, 2004). En la figura 8 se muestra la arquitectura de esta biblioteca.

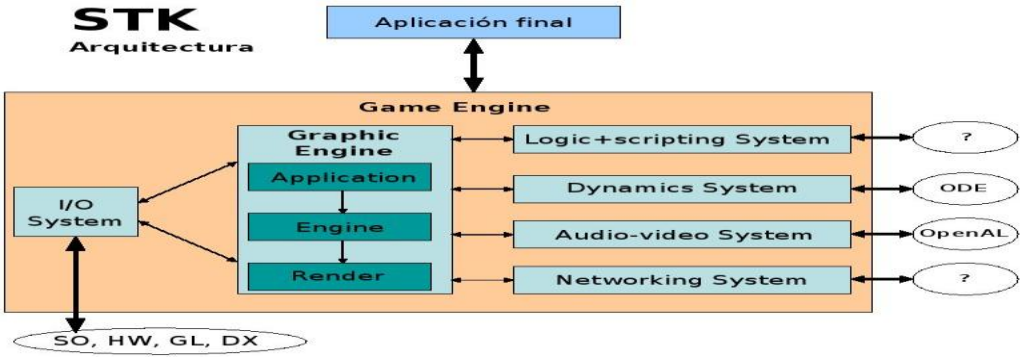

Figura 8: Arquitectura de la STK

1.6 Biblioteca para la física.

Existen diversas formas de lograr que un cuerpo se comporte con alto grado de realismo en un medio totalmente simulado y cuyas propiedades sean configuradas o manejadas por un ordenador. La idea de delegar un modelo físico utilizado por los juegos o simuladores en una biblioteca es algo que en los últimos tiempos se ha vuelto muy común. Son numerosos los sistemas informáticos modernos y videojuegos que utilizan motores físicos para producir animaciones realistas del movimiento de los objetos y personajes, que no son más que conjuntos rígidos dentro del marco del diseño para la simulación. Entre los motores físicos más conocidos se pueden mencionar: NovodeX, Newton, Tokamak, ODE, PhysX, Endorphin y Ragdoll (Hawkins, 2007). A continuación se hace referencia a uno de ellos.

Open Dynamics Engine (ODE) es una biblioteca de código abierto, excelente para la simulación de la dinámica de cuerpos rígidos articulados pues permite dotar a los objetos de masa y solidez física, aplicarles fuerzas y hacer que todos se comporten de una forma más realista. ODE fue desarrollado por Russel Smith y lanzada bajo la licencia BSD de código abierto. Es una biblioteca gratuita de calidad industrial para la simulación dinámica de sólidos rígidos y el movimiento de objetos en entornos virtuales. Es independiente de la plataforma, posee articulaciones avanzadas y detección de colisiones propia. Es

considerada estable. Se encuentra escrita en C++, posee una interfaz en C que facilita la integración. Se distribuyen las fuentes de la biblioteca que están escritas en ANSI C++. Agrupa familia de funciones por prefijo, lo cual facilita la búsqueda de funcionalidades comunes a este tipo de bibliotecas.

ODE se encarga de asociar, a cada objeto que se le indique una entidad física, que normalmente coincidirá con su forma visible, y que será la responsable del comportamiento físico del objeto. A partir de ese momento la guía será la entidad física, independientemente de la forma visible del objeto. También actualiza el estado de los cuerpos rígidos a través del tiempo, atiende el proceso de acción y respuesta a las colisiones que pueden ocurrir entre los cuerpos, facilita hacer visible la edición de los objetos de forma óptima, relaciona varios parámetros (posición inicial, velocidad (angular, lineal), masa, fuerzas, coeficientes de fricción, un ambiente o mundo de la escena que también cuenta con sus particularidades (gravedad, fricción, resistencia al medio, entre otras), así como permite asociar volúmenes de geometrías (cúbicas, cilíndricas, esféricas) a los objetos, teniendo con anterioridad las dimensiones del mismo.

Al ser ODE de código abierto da mayor control sobre el producto que se realice. Si se quiere elaborar un motor físico se pueden usar partes de esta biblioteca como referencia. La propia biblioteca ODE restringe la simulación a sólidos rígidos, que simplifican el cálculo de la física, lo que repercute en una mayor velocidad de proceso.

ODE ha sido utilizada en investigaciones biomecánicas y robóticas. Su ecuación para el movimiento de cuerpos rígidos es derivada del modelo de velocidad basado en los multiplicadores de Lagrange. Es rápida, robusta y estable, además el usuario tiene la completa libertad de cambiar la estructura del sistema aun cuando la simulación esté corriendo. Enfatiza velocidad y estabilidad por encima de la precisión física. (Smith, 2007).

Desventajas

A pesar de las grandes facilidades que brinda ODE presenta algunas desventajas debido a que trabaja únicamente sobre la dinámica de sólidos rígidos, en la simulación del movimiento de vehículos no se encarga del funcionamiento del motor y los diferentes

componentes de este. (Gilimas, 2008).

1.7 Áreas de aplicación de los grafos.

Dentro de la informática actual la teoría de grafos juega un papel de singular importancia. El término tiene su origen en el diccionario griego que lo define como dibujo e imagen. Sin embargo 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.

1.7.1 Entornos de ciudades.

La simulación de entornos de ciudades constituye una de las áreas donde se ha empleado con aciertos la teoría de grafos. En este sentido los nodos representan las intersecciones de las calles así como elementos fijos del entorno como pueden ser edificios, señales del tránsito, árboles entre otros tantos, mientras que las aristas son las calles que comunican a los distintos elementos del entorno. Teniendo en cuenta que las ciudades carecen de sentido sin la presencia de objetos dinámicos tales como humanos, animales, autos, aviones así como todo elemento móvil del entorno se ha hecho uso de las estructuras de grafo no sólo para representar los elementos estáticos de una ciudad con sus vías de comunicación sino además como medio para guiar la locomoción de los elementos dinámicos que se encuentran en ella.

En nuestros días nos encontramos con diversos problemas donde se dan situaciones en el entorno que provocan que objetos dinámicos se comporten de una forma en particular. Varios son los ejemplos que pueden citarse al respecto dentro de un entorno de ciudad.

1. Velocidad máxima limitada.

Supongamos que un auto transita por una calle por donde se hace necesario establecer un límite de velocidad debido a que en ella se encuentran enclavadas instituciones donde

intervienen personas que corren el riesgo de ser atropelladas por un automóvil, como pueden ser niños que van o regresan de la escuela, ancianos que transitan por la calle para acceder al hogar de ancianos o club de abuelos, entre otros. También por curvas peligrosas u otros accidentes geográficos que hagan peligroso el exceso de velocidad.

2. Señalizaciones de PARE.

Las distintas señales del tránsito que provocan que tanto peatones como máquinas automotoras se detengan entran dentro de este acápite, entre las que se encuentran, paso peatonal, paso a nivel, las de PARE propiamente dicho.

3. Otras señalizaciones.

Otras muchas señalizaciones forman parte de una ciudad como pueden ser: seda el paso, seguir una dirección, prohibido estacionar, información de zona de derrumbe entre otras muchas.

Cada una de estas señalizaciones pudiera ser representada mediante características incorporadas a los nodos y las aristas de un grafo como se verá más adelante.

1.7.2 Puntos de Visibilidad.

Otro de los campos donde se ha empleado la teoría de grafos con resultados favorables son los puntos de visibilidad (POV) según sus siglas en inglés. Los puntos de visibilidad constituyen uno de los modelos cognitivos más usado en nuestros días. Los modelos cognitivos son empleados para representar el mundo virtual, brindando información y datos topográficos al agente inteligente, ayudando así en la toma de decisiones, como puede ser por ejemplo el camino a tomar hacia un objetivo o la mejor posición de defensa o ataque. Son utilizados para visualizar el mundo y poder tener una representación de como está estructurado este, para que los agentes conozcan la información referente al entorno virtual con el cual interactúan y puedan desplazarse dentro del mismo. El uso de esta técnica reduce los espacios contiguos del terreno a espacios más discretos que los algoritmos de búsqueda puedan interpretar (Oconor, 2008).

Los agentes de inteligencia artificial de un entorno virtual necesitan de un modelo que les

brinde la información del mundo con el cual interactúan y esto se logra mediante los modelos cognitivos.

Entre los modelos cognitivos más populares en la actualidad se encuentran la rejilla regular, el quadtree, los cilindros generalizados, las mallas de navegación y los puntos de visibilidad. (Stout, 2000).

El modelo cognitivo puntos de visibilidad, resulta fácil de implementar y brinda gran cantidad de información a partir de los nodos del grafo que componen. Este modelo es creado por la inserción de nodos dentro de un mundo tridimensional. A cada uno de estos nodos se conoce como "*waypoint",* término en inglés que significa puntos del camino.

Los *waypoints* son puntos dentro de un mapa que indican posibles lugares a recorrer. Estos puntos se encuentran interconectados formando un grafo de navegación, siendo los puntos o nodos las distintas localizaciones del entorno y las aristas los caminos que los comunican. Estas interconexiones tienen propiedades específicas, una de las más importantes es que facilitan la navegación de un nodo a otro sin la necesidad de estar interconectados.

Las conexiones entre estos *waypoints* determinan los caminos viables, los cuales son representados en el grafo a partir de las interconexiones que se forman entre ellos (Puig, 2008). Mientras mayor número de *waypoints* posea un grafo la representación del mundo será más exacta, además de mostrar con mayor claridad todas las propiedades asociadas al entorno.

1.8 Reubicación.

La inclusión de elementos dinámicos en entornos virtuales imprime realismo y creatividad a los entornos simulados pero al mismo tiempo exigen mayor cantidad de recursos de hardware. Para resolver este problema en la Universidad de las Ciencias Informáticas en el año 2007 se desarrolló una tesis de grado con título *"Algoritmos para la manipulación eficiente de objetos dinámicos en escenas virtuales urbanas"* (Nogueira, 2007) donde se

empleó un algoritmo de reubicación con el objetivo de usar de forma eficiente los recursos de hardware durante el uso de objetos dinámicos en entornos virtuales urbanos. Para ello se plantea la disminución de los elementos dinámicos del entorno de forma tal que la simulación de la escena posea mayor interactividad.

La idea básica del algoritmo radica en la ubicación de elementos dinámicos no visibles del entorno en lugares con mayor probabilidad de visibilidad dentro del mismo. Con este objetivo se localizan las trayectorias dentro de la escena con mayor probabilidad de visibilidad en los *frames* más cercanos y se procede a la ubicación en esta zona de los elementos que se mueven dentro del entorno.

A partir de esto se definieron como pasos del algoritmo de reubicación (Nogueira, 2007) los siguientes:

LNV: lista de objetos dinámicos no visibles
LPR: lista de *Puntos de Reubicación*.
COR: cantidad de objetos a reubicar por cada nodo.
LOR: lista de objetos a reubicar.
CNR: cantidad de nodos que están relacionados con el nodo actual de la cámara.

1. Determinar el tamaño que va a tener *LNV* a partir de la cantidad de objetos que se van a reubicar por cada *punto de reubicación* (*COR*) definida por el usuario.
2. Confeccionar *LPR*. Se obtienen los nodos relacionados con el *nodo actual* de la cámara (que constituyen los posibles *puntos de reubicación*) y se verifica que no sean visible, en caso de que alguno lo sea, se busca un nodo alternativo que será el *punto de reubicación*.

A continuación se describen estos pasos en el siguiente pseudocódigo.

Algoritmo de Reubicación (pasos 1 y 2)

```
1:  Nodo_Actual  <-  Cámara . Obtener_Nodo_Actual ();
2:  Lista   <-  Nodo_Actual . Obtener_Lista_Nodos_Relacionados ();
3:  Desde i  <- 1  hasta CNR  con paso 1
4:     Si ( ! Lista[i]. Es_Visible ()), entonces:
5:        LPR. Adicionar (Lista[i]);
6:     Sino,   entonces:
7:        Nodo   Lista[i]. Buscar_Nodo_Alternativo ();
8:        LPR. Adicionar (Nodo);
9:  fin Desde.
10: fin Algoritmo de Reubicación (pasos 1 y 2)
```

Figura 9: Algoritmo de Reubicación. Descripción de los pasos 1 y 2.

3. Reubicar por cada nodo de LPR la *cantidad* seleccionada de objetos dinámicos.
4. En cada nodo de LPR se reubica la cantidad de objetos definida en el paso 1. El objeto que se va a reubicar es siempre el que está en la primera posición de LNV esto se debe a que la lista es rotada cada vez que se reubica un objeto, es decir, el objeto que se reubica es el primero de LNV y una vez reubicado se elimina de LNV y se coloca al final de la lista. Todo esto se explica detalladamente en el pseudocódigo que le mostramos a continuación.

Algoritmo de Reubicación (pasos 3 y 4)

```
1:  Para cada nodo n  ∈  LPR, hacer:
2:  Mientras (COR != 0), hacer:
3:    Objeto_A_Reubicar = LNV [0];
4:    Si ( !Objeto_A_Reubicar. Esta_en_Trayectoria_Anterior ()), entonces:
5:       Objeto_A_Reubicar. Reubicar(n); /*se reubica el objeto en el punto de reubicación n  */
6:       Decrementar COR;
7:    fin Si.
8:    Rotar LNV;
9:  fin Mientras.
10: fin Algoritmo de Reubicación (pasos 3 y 4)
```

Figura 10: Algoritmo de Reubicación. Descripción de los pasos 3 y 4.

CAPÍTULO 2: SOLUCIÓN TÉCNICA

Teniendo en cuenta los elementos abordados en el Capitulo 1 donde se tratan los principales conceptos y características relacionadas con la IA, con el objetivo de darle solución al problema se emplea la técnica de comportamientos de locomoción para lograr que los elementos que se muevan dentro del entorno posean cierto grado de realismo e inteligencia.

Se desarrolla una herramienta que permita la incorporación de comportamientos inteligentes a elementos que utilizan grafos de camino, esta herramienta consta de dos módulos: *Aplicación QT* donde se carga toda la información referente al grafo de camino y se asignan los comportamientos de locomoción a los nodos, aristas y elementos y el *Módulo de Comportamientos* donde se crean los nuevos comportamientos que serán asignados en dependencia de las necesidades de cada desarrollador y se visualiza la asignación de estos en tiempo de ejecución a cada uno de los elementos del entorno.

Antes de detallar cada uno de esos módulos se presentan algunos aspectos importantes para comprender y justificar la propuesta realizada.

2.1 Grafo de Camino.

La posibilidad de disponer de herramientas para la creación y edición de caminos tridimensionales es vital para los motores gráficos de hoy día. La *Herramienta de creación de grafos de caminos para la biblioteca "SceneToolkit"* es un sistema innovador que se adapta perfectamente a las necesidades requeridas por el motor gráfico y brinda la posibilidad de crear, modificar y deformar visualmente los caminos del entorno, mediante un juego de herramientas intuitivas y fáciles de utilizar; además permite la aplicación de cálculos de camino mínimo entre nodos del grafo de caminos y guardar toda la información en ficheros.

El grafo del entorno está compuesto por un conjunto de aristas y nodos. Las aristas representan los caminos del entorno y los nodos las intersecciones entre los caminos o lugares destacados, que constituyen puntos de toma de decisión para los agentes inteligentes. Cuando un agente llegue a un nodo debe decidir qué nuevo camino va a seguir, pues de un nodo pueden partir varias aristas. El grafo es dirigido y entre dos vértices solo pueden existir dos aristas ponderadas, las cuales no podrán tener el mismo sentido. La información almacenada en el grafo es un conjunto de vértices 3D que marcan el eje central del camino. (Cedeño, 2008).

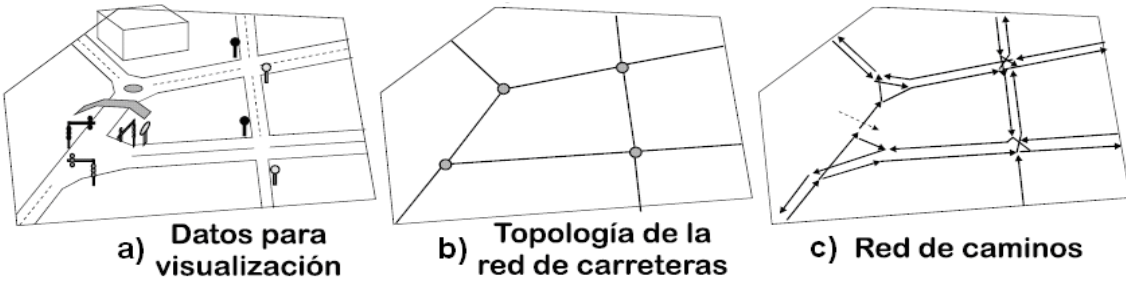

Figura 11: Distintas vistas de un grafo de caminos del entorno.

En la parte (a) de la figura 9 se observa el grafo de camino en el entorno virtual una vez creado por parte del usuario. En la sección (b) se muestra la información topográfica del grafo que junto a la información de navegación mostrada en (c) constituyen los únicos datos que necesita el algoritmo de planificación para ejecutarse.

Para brindar un mayor realismo a la representación de los grafos de caminos, las aristas son modeladas mediante *"NURBS"*. Estas curvas también ofrecen una formulación matemática común para representar formas analíticas y diseñar con precisión curvas de forma libre. Debido a que todas las entidades, normalmente utilizadas en diseño geométrico, pueden expresarse bajo una única forma matemática, facilitan la integración funcional y la física.

Su evaluación es razonablemente rápida y computacionalmente estable; los resultados son precisos y consistentes debido al alto nivel de integración. Son invariantes a las

transformaciones tales como el escalado, la rotación, la traslación. Se comportan adecuadamente en las proyecciones paralela y perspectiva. Tienen una clara interpretación física, lo que las hace especialmente útiles para el diseño gráfico asistido por computador. Poseen un poderoso juego de herramientas geométricas: puntos de control (número y posición), elección del grado independiente del número de puntos de control, pesos (ponderación de los puntos de control), vector de nudos (definición, inserción, refinamiento, eliminación, etc.), lo que proporciona gran flexibilidad para diseñar interactiva y cómodamente una amplia gama de formas. (Cedeño, 2008).

El formato de fichero GraphML proporciona etiquetas para representar toda la información referente a las estructuras de datos, como son los grafos, nodos, aristas y sus atributos. Dentro del fichero el grafo se denota con el elemento graph y dentro de él se encuentran declarados los nodos, aristas, vértices de control y caminos.

Definición de un grafo.

```
<graph id="G">
  <node id="n0"/>
  <node id="n1"/>
   ...
  <node id="n10"/>
  <edge source="n0" target="n2"/>
  <edge source="n1" target="n2"/>
  ...
  <edge source="n8" target="n10"/>
</graph>
```

Para representar los caminos del entorno se hará uso de este grafo de camino debido a que ha sido desarrollado por ingenieros del polo de realidad virtual de la facultad 5 y continúa en desarrollo. Además se dejará preparada la herramienta resultado de esta investigación para poder utilizar otras herramientas que generen grafos de camino, con el objetivo de lograr un sistema más portable.

2.2 Descentralización de la IA.

Por lo general cuando pensamos agregar inteligencia a un entorno virtual se asocia exclusivamente a los elementos dinámicos del entorno, de manera que se comporten como autómatas que conozcan en cada momento que acción seguir. Expertos en el tema han planteado las notables ventajas que tiene hacer uso adecuado de la inteligencia artificial implementada en nuestros entornos virtuales, especialmente cuando se trata de sistemas que se ejecutan en tiempo real, como simuladores y algunos tipos de juegos, con el objetivo de lograr cada vez más situaciones realistas. Mclean plantea en (Mclean, 2002) la necesidad de distribuir el trabajo de forma eficiente en estos tipos de sistemas. En este sentido destaca tres aspectos fundamentales:

Separación: Donde se plantea dividir los componentes de inteligencia en dos, los que se ejecutan cada cierto tiempo y los que lo hacen en cada instante de tiempo.

Distribución: Propone la distribución del trabajo de forma tal que se use la misma cantidad de tiempo en cada tarea.

Exclusión: Expone la necesidad de excluir en cada instante de tiempo la ejecución de aquellas tareas que no sean priorizadas o no jueguen en ese momento un papel importante dentro de la ejecución del sistema.

Sin embargo aun quedan problemas por resolver en la lucha por lograr sistemas inteligentes lo más cercanos a la realidad. La descentralización de la IA que plantean especialistas constituye un nuevo enfoque que merece la pena tener en cuenta a la hora de implementar sistemas de realidad virtual (Coca, 2008). Cuando creamos un entorno virtual y le agregamos inteligencia a sus elementos dinámicos estamos creando los comportamientos a seguir por cada uno de estos elementos al interactuar con otros elementos dinámicos del entorno así como con los objetos estáticos que se encuentren dentro del mismo. Pero en caso de necesitar incorporar nuevos elementos al entorno ya sean dinámicos o estáticos, tendremos que reimplementar todos los elementos que teníamos a punto, si además cada elemento móvil tiene que saber cómo comportarse con

cada objeto estático del entorno, en la medida que estos crezcan en número habrá que ampliar la IA de cada agente lo que posiblemente provocaría la sobrecarga de IA de estos elementos, haciéndose más compleja e ilegible su implementación y en muchos casos más costosa en tiempo.

Una solución a este problema sería **agregarle IA al mundo virtual.** En este sentido los agentes del entorno tendrían implementados comportamientos básicos y los objetos estáticos tendrían asociados una serie de comportamientos que serían cumplidos por los agentes una vez que interactúen con estos objetos, es decir serían los elementos no móviles del entorno quienes dirían a los elementos móviles que hacer cuando se relacionen con ellos. Vale aclarar que los agentes tienen la responsabilidad de decidir si cumplen o no con estos comportamientos en dependencia de sus propios objetivos. De esta manera estaremos creando entornos más extensibles y realistas.

2.3 Agentes creíbles.

La teoría de agentes creíbles cobra cada vez mayor importancia dentro del mundo de la RV por la dinámica y credibilidad que aportan a los entornos virtuales ya sean en juegos o simuladores. En este trabajo la implementación de los elementos del entorno se llevó a cabo de manera que estos se comporten como verdaderos agentes creíbles. Para ello se tuvo en cuenta las tres características que identifican a los agentes que fueron mencionadas en el epígrafe 1.4.

Activos

Los agentes deben poseer objetivos propios que cumplir dentro del entorno. El objetivo principal de los elementos radica en su locomoción por todo el mundo virtual, para lograr esto se le asignan un grupo de comportamientos básicos, en dependencia de las características propias de cada agente. Cada uno de estos comportamientos será adicionado a los elementos en la “Aplicación QT”, información que será cargada por el módulo de comportamiento una vez inicializado el entorno.

Proactivos

La teoría plantea que los agentes deben ser capaces de dar respuestas ante las distintas condiciones que se dan en el entorno. En el presente trabajo se ha defendido la idea de "agregarle inteligencia al mundo virtual" de manera que sea el entorno quien le informe a los elementos como deben comportarse ante situaciones determinadas, evitando así sobrecargar la inteligencia de los elementos. Para ello las aristas y los nodos que representan al mundo le asignan un grupo de comportamientos a los elementos que estén en su radio de acción. A continuación los agentes serán capaces de decidir cuales acciones deben seguir, teniendo en cuenta sus propios objetivos. La combinación de comportamientos a la que se hizo referencia en el epígrafe 1.2.3 se encarga de establecer la conducta adecuada que debe cumplir cada uno de los elementos.

Sociales

Según esta característica los agentes deben poseer la habilidad de relacionarse con otros elementos del entorno. Vivir en sociedad, como sucede con los seres humanos en el mundo real. En el módulo de comportamientos los agentes conocen el entorno en el cual se mueven, se relacionan con los otros elementos del mundo a través de comportamientos específicos para ello, algunos forman parte de sus comportamientos básicos y otros lo reciben a través del entorno. En cualquier caso se comunican directamente a través del entorno pero el diseño del presente trabajo debe facilitar una futura implementación de cualquier lenguaje o vía de comunicación que se proponga.

Otras características

Otra de las características de agentes muy relacionadas con las antes mencionadas y que también se evidencian en este trabajo es la racionalidad de los elementos, que les permite hacer lo correcto a partir de las condiciones del entorno. Así como su movilidad evidenciada en cada uno de los distintos tipos de comportamientos de locomoción que son capaces de asumir los elementos, pero además la implementación realizada para una aplicación específica de los agentes puede ser utilizada sin modificaciones en otros entornos y aplicaciones diferentes, siempre que utilicen la herramienta presentada en este

trabajo.

2.4 Descripción de la herramienta.

2.4.1 Aplicación QT.

El módulo Aplicación QT brinda una interfaz gráfica de usuario para la asignación de comportamientos a los distintos objetos del entorno, ya sean estáticos o dinámicos. La aplicación está diseñada con varias ventanas para la interacción con el usuario, las cuales están estructuradas de la siguiente forma:

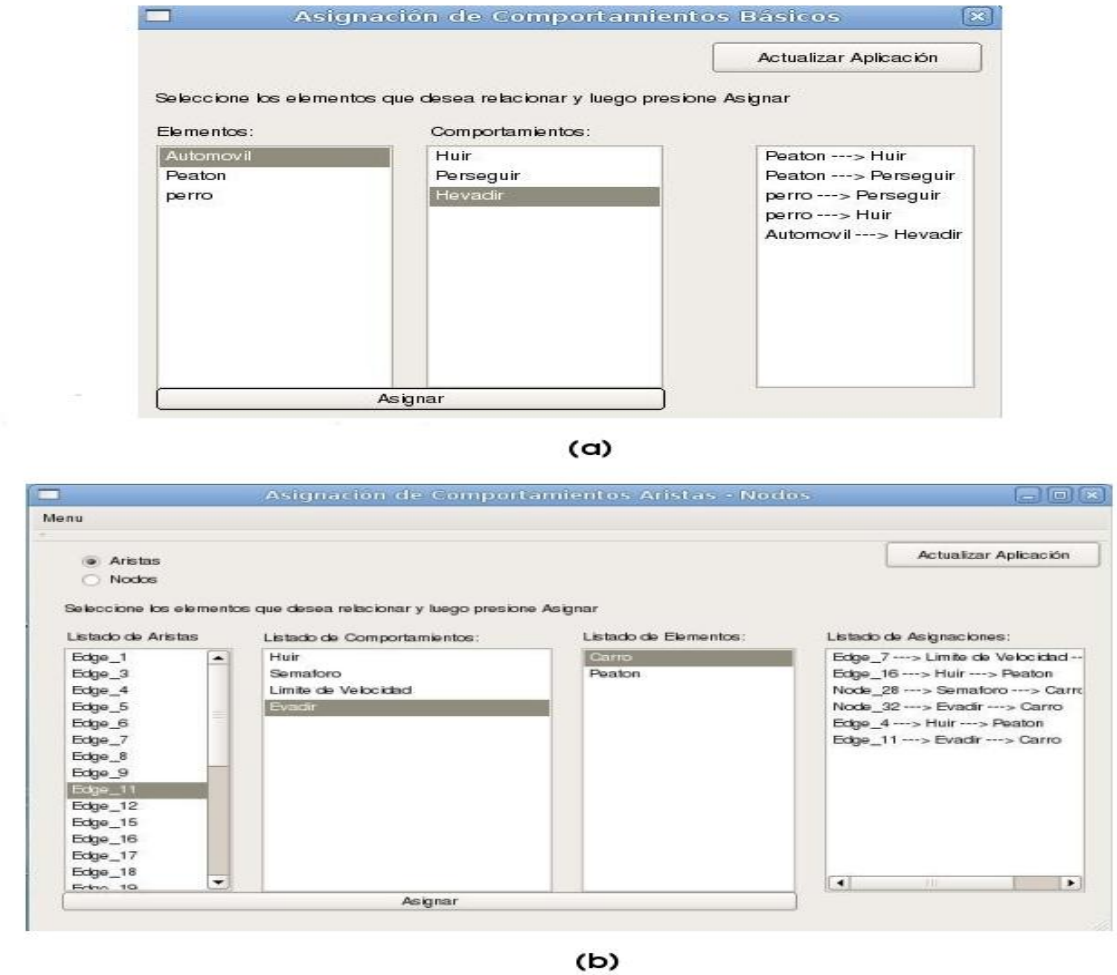

Figura 12: Ventanas de la Aplicación QT.

Ventana “Cargar Grafo” tiene la responsabilidad de cargar el fichero XML con la información referente al grafo de camino del entorno virtual.

Ventana “Adicionar Elemento” en ella se adiciona el nombre y el identificador del tipo de cada elemento que va a interactuar en el entorno.

Ventana “Adicionar Comportamientos” cuya función radica en la adición del nombre e identificador de un nuevo comportamiento.

Ventana “Asignación de comportamientos Básicos” ver figura 12 (a), donde se asigna a los elementos sus comportamientos básicos.

Ventana “Asignación de comportamientos Aristas-Nodos” ver figura 12 (b), donde se realizan las tareas de asignación de comportamientos a los objetos del mundo virtual.

Ventana “Exportar asignación de comportamientos” es la encargada de exportar la información definida anteriormente en la aplicación, la cual será almacenada en un fichero denominado “DCFile.xml“.

Para lograr las funcionalidades antes descritas se usaron un grupo de clases que responden a las necesidades de la aplicación. La clase **CBehavior** contiene la información necesaria para identificar un comportamiento. Por su parte en **CElement** se definen las principales características que van a tener los distintos elementos que se mueven dentro del entorno. La clase **CPathGrap** representa el grafo de camino, posee una lista de aristas, nodos y operaciones con las cuales se puede modificar y obtener información del grafo. Las clases **CGrapNode** y **CGrapEdge** representan un nodo y una arista respectivamente, ambas poseen una lista de comportamientos asociados. **MainWindows** es la clase principal, controla el trabajo de las demás clases del módulo, se realizan las operaciones de adición de comportamientos y elementos, se asignan los comportamientos a los distintos objetos y agentes del entorno y se realiza el trabajo de lectura y escritura de fichero.

Estructura del fichero DCFile

El fichero DCFile.xml está conformado por un conjunto de etiquetas que representan la

información relacionada con las diferentes estructuras de datos que se manejan en el trabajo como son los comportamientos de locomoción, elementos, grafos, nodos, aristas y algunos de sus atributos. A continuación se explica como están estructuradas cada una de las etiquetas del fichero.

Estructura general del fichero.

```
−<DCFile>
   −<BAsing> Asignación Comportamientos Básicos a los elementos.
      <data key> atributos </data>
   −</BAsing>

   −<BAsing> Asignación Comportamientos a las Aristas o Nodos.
      <data key> atributos </data>
   −</BAsing>
−<DCFile>
```

Ejemplo de fichero DCFile.

```
 −<DCFile>
   −<BAsing id="0">
      <data key="asig_name">Carro</data>
      <data key="asig_id">1</data>
      <data key="typeObject">2</data>
      <data key="Name_Behavior">Movimiento Aleatorio</data>
      <data key="ID_Behavior">3</data>
    </Basing>

   −<BAsing id="1">
      <data key="asig_name">Edge_53</data>
      <data key="asig_id">13200</data>
      <data key="typeObject">0</data>
      <data key="Name_Behavior">Limite de velocidad</data>
      <data key="ID_Behavior">2</data>
      <data key="Name_Element">Carro</data>
      <data key="ID_Element">1</data>
    </BAsing>

 −<BAsing id="5">
      <data key="asig_name">Node_63</data>
      <data key="asig_id">62</data>
      <data key="typeObject">1</data>
      <data key="Name_Behavior">Pare</data>
```

```
    <data key="ID_Behavior">1</data>
    <data key="Name_Element">Carro</data>
    <data key="ID_Element">1</data>
  </Basing>
</DCFile>
```

Cada asignación tiene un identificador único que facilita la búsqueda por la estructura, en el fichero se representa con el atributo **BAsing id**, para definir el tipo de objeto se utiliza el atributo **typeObject** que toma diferentes valores, 0 cuando es una arista, 1 cuando es un nodo y 2 si el objeto es un elemento.

Asignación de comportamientos básicos a elementos.

Se utiliza para definir cuales son los comportamientos de locomoción propios de cada uno de los elemento dentro del entorno. En la etiqueta se definen el nombre del elemento al cual se le va a asignar el comportamiento, el identificador, que es único, el tipo de objeto, en este caso un elemento y el nombre e identificador del comportamiento que es asignado.

Ejemplo de etiqueta de asignación de comportamientos básicos a los elementos.

```
−<BAsing id="0">
    <data key="asig_name">Carro</data>
    <data key="asig_id">1</data>
    <data key="typeObject">2</data>
    <data key="Name_Behavior">Movimiento Aleatorio</data>
    <data key="ID_Behavior">3</data>
  </Basing>
```

Atributos.

- **asig_name** -> Nombre del elemento.
- **asig_id ->** Identificador del elemento .
- **TypeObject** -> Tipo de objeto.
- **Name_Behavior ->**Nombre de Comportamiento.
- **ID_Behavior** ->Identificador de Comportamiento.

Asignación de comportamientos a nodos y aristas.

Se asignan los comportamientos asociados a las aristas y a los nodos, definiéndose para ello los elementos que deberán ejecutar dichos comportamientos. Para lograr esto se define el nombre de la arista o del nodo, el identificador, el tipo de objeto en este caso sería una arista o un nodo, el nombre e identificador del comportamiento y el nombre e identificador del elemento que va a ejecutar el comportamiento.

Ejemplo de etiqueta de comportamientos a nodos y aristas.

```
−<BAsing id="3">
    <data key="asig_name">Pare</data>
    <data key="asig_id">60</data>
    <data key="typeObject">1</data>
    <data key="Name_Behavior">Pare</data>
    <data key="ID_Behavior">1</data>
    <data key="Name_Element">Carro</data>
    <data key="ID_Element">1</data>
  </Basing>
```

Atributos.

- **asig_name** -> Nombre de arista o nodo.
- **asig_id ->** Identificador de arista o nodo.
- **TypeObject** -> Tipo de objeto.
- **Name_Behavior ->**Nombre del comportamiento.
- **ID_Behavior** ->Identificador del comportamiento.
- **Name_Element** -> Nombre del elemento.
- **ID_Element ->** Identificador del elemento.

2.4.2 Módulo de Comportamientos.

Para la implementación de la propuesta se separó este módulo en dos paquetes donde se encuentran la mayoría de las clases usadas en la solución.

Paquete IA

En este paquete se encuentran las clases relacionadas con la “Inteligencia Artificial” que es agregada tanto a los agentes como al entorno en cuestión. Para ello se definen clases

básicas de la herramienta así como un grupo de ellas que pueden variar en correspondencia con las necesidades de los desarrolladores. Las clases básicas son:

"CElement": Que representa a los distintos agentes del entorno.

"CBehavior": Constituye una clase abstracta de la cual heredarán cada una de las clases comportamientos que se deseen implementar de acuerdo a las necesidades y características del entorno y los elementos. Estas clases hijas son consideradas *"no básicas"* debido a que varían en dependencia de las necesidades del desarrollador.

Las clases **"CIntEdge"** y **"CIntNode"** contienen la información referente a la inteligencia del entorno. Se crearon con el objetivo de separar la IA del grafo que se esté utilizando, de este modo se logra un sistema más extensible, dejando la posibilidad de usar en un futuro otro grafo sin que ello afecte la propuesta planteada. Se hace uso de la clase **"CInterfaceGraph"** cuya responsabilidad radica en obtener toda la información que se necesita del grafo.

Se utilizó además la clase **"CElemBehavior"** para establecer la relación entre cada comportamiento en particular y los distintos tipos de agentes a los que son aplicables.

Por último la clase **"CControl"** se encarga de controlar el trabajo de las demás clases de este paquete.

Por su parte el **paquete Graph** contiene las clases e implementaciones del grafo de camino que ha sido empleado para desarrollar esta herramienta (Cedeño, 2008). Como se dijo anteriormente cualquier nueva implementación se incluiría en este paquete y solo se modificaría la clase **"CInterfaceGraph".**

2.5 Tecnología utilizada en la solución técnica.

Para llevar a cabo el modelado y diseño del sistema fue utilizada la herramienta **Visual Paradigm** y la metodología de desarrollo de software **RUP.** Posteriormente en la fase de implementación se hace uso del **QTCreator** para la confección de la Aplicación QT donde

se realiza el proceso de asignación de comportamientos a los distintos elementos estáticos y dinámicos del entorno. Se utilizó **Code::Blocks** para la creación del módulo de comportamiento del sistema, empleándose la biblioteca **SceneToolKit** como motor gráfico y se utiliza el formato de **Fichero XML** para cargar y almacenar toda la información relacionada con el grafo de camino y la asignación de comportamientos a los distintos elementos, empleándose para el trabajo con ficheros la biblioteca **TinyXML.**

2.5.1 Visual Paradigm versión 6.4.

Visual Paradigm es una herramienta que utiliza UML como lenguaje de modelado, resulta intuitiva y fácil de utilizar, soporta el ciclo de vida completo del desarrollo de software. Brinda la posibilidad de construir la aplicación de forma rápida, con mayor calidad y menor coste. Permite dibujar todos los tipos de diagramas de clases, utilizar código inverso, además de generar código desde diagramas y generar documentación. Esta herramienta proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML.

Cuenta con un editor de detalles de casos de usos y con un entorno integrado para la especificación de los detalles de cada caso de uso. Permite la importación y exportación de ficheros XML y corre en plataformas como Windows, Linux y Mac OS X.

2.5.2 Metodología de Desarrollo.

Una metodología es el conjunto ordenado de pasos a seguir para desarrollar un software de alta calidad que cumpla con las necesidades del usuario. Permite asignar tareas y responsabilidades, determinar un camino metódico y sistemático para desarrollar, diseñar y validar una arquitectura y reducir en gran medida los riesgos que representa la construcción de software. En el presente trabajo de diploma se emplea la metodología de desarrollo “RUP” para el diseño del sistema.

Proceso Unificado de Desarrollo (RUP).

La metodología RUP es un Proceso Unificado de Desarrollo. Brinda un marco de trabajo

genérico que puede especializarse para gran variedad de sistemas (Jacobson, 1999). Utiliza el lenguaje unificado de modelado UML para la modelación de los sistemas a desarrollar, constituye la metodología estándar más utilizada para el análisis, diseño y documentación de sistemas orientados a objetos. Puede considerarse un conjunto de metodologías adaptables al contexto y necesidades de cada proyecto ya que describe como aplicar enfoques para el desarrollo del software y permite comprender los requerimientos que hacen crecer al sistema. Se basa en diseñar una arquitectura flexible, fácil de modificar, comprensible fundamentada en la reutilización de sus componentes. Las principales características que lo identifican se enumeran a continuación (Jiménez, 2008).

1. Es iterativo e incremental.
2. Centrado en la arquitectura.
3. Guiado por los casos de uso.

RUP guía a los equipos de proyecto en cómo administrar el desarrollo iterativo de un modo controlado mientras se balancean los requerimientos, el tiempo y los riesgos. El proceso describe los diversos pasos involucrados en la captura de los requerimientos y en el establecimiento de una guía arquitectónica, para diseñar y probar el sistema hecho de acuerdo a los requerimientos y a la arquitectura.

RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al culminar cada uno de ellos, estos a la vez se dividen en fases.

Características fundamentales de RUP: (Jiménez, 2008).

1. Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
2. Pretende implementar las mejores prácticas en Ingeniería de Software.
3. Desarrollo iterativo.
4. Administración de requisitos.
5. Uso de arquitectura basada en componentes.
6. Control de cambios.
7. Modelado visual del software.
8. Verificación de la calidad del software.

2.5.3 QTCreator versión 1.0.

QTCreator 1.0 está conformado por un conjunto de librerías multiplataforma utilizadas en el desarrollo de aplicaciones con interfaces gráficas. Es un mecanismo de comunicación seguro, flexible y totalmente orientado a objetos e implementado en C++.

Esta herramienta es una aplicación mediante la cual se pueden realizar el diseño de aplicaciones de Interfaz gráfica de usuarios (*GUI) según sus siglas en inglés.* Permite diseñar de una forma muy sencilla y rápida ventanas de diálogo.

La principal característica de QTCreator son los *slots* y *signals*, los cuales son un mecanismo de comunicación entre objetos, rasgo que lo hace distinto del resto de herramientas para la elaboración de interfaces gráficas.

Incluye otras herramientas para la implementación de las aplicaciones como la aplicación *uic* que permite implementar la clase del modelo realizado con *QtDesigner* a partir del archivo *.ui,* es decir obtener los archivos *.cpp* y *.h*. (Corte, 2007)

2.5.4 Code::Blocks.

Es un entorno de desarrollo integrado, de código libre, entre sus características más atractivas se encuentra su capacidad de correr en múltiples sistemas operativos, muy utilizado para desarrollar programas en lenguaje de programación C++. Posee compatibilidad con varias bibliotecas entre las que se encuentran aplicación de consola, aplicaciones GUI entre otras. Puede incorporar distintos plug-ins, es capaz de generar ficheros XML para proyectos, entre otras muchas características.

2.5.5 SceneToolKit 2.3.

La biblioteca gráfica SceneToolKit desarrollada por miembros del proyecto productivo Herramientas de desarrollo para sistemas de realidad virtual de la facultad 5, permite la visualización del entorno tridimensional, además de la aplicación de leyes físicas, matemáticas y animaciones mediante la integración de bibliotecas gráficas como OpenGL

y DirectX además bibliotecas para la física como ODE.

Agrupa funcionalidades comunes a cualquier sistema de realidad virtual permitiendo así la reutilización de código.

2.5.6 Biblioteca TinyXML.

Atendiendo a las grandes facilidades que proporciona el formato de archivos e**X**tensible **M**arkup **L**anguage (XML) sobre todo para el trabajo con grafos (Cedeño, 2008) y teniendo en cuenta que fue este formato de archivo el utilizado por el grafo de camino sobre el cual se ha desarrollado el presente trabajo, se decidió usar este tipo de formato para los archivos con la información del grafo y la asignación de los comportamientos a los distintos objetos del entorno y a los agentes del mismo.

Un fichero XML es un fichero en formato de texto, que contiene etiquetas para identificar los nodos que componen el documento, en este formato se almacenarán de forma persistente las propiedades de los elementos, el documento XML tiene una estructura anidada de forma jerárquica. Una de las características principales de los ficheros XML es que son auto-descriptivos, es decir el mismo fichero XML describe la estructura de los datos que contiene y además los datos en sí mismos. Esto simplifica enormemente el trabajo con estos ficheros tanto para el desarrollo de programas que los utilicen, como su uso para configuraciones o envío de datos entre diferentes entornos o plataformas. (Román, 2009)

La biblioteca STK que ha sido utilizada como motor gráfico en este trabajo se encuentra aún en desarrollo. Debido a que no contiene entre sus facilidades el trabajo con archivos XML, se ha hecho necesario buscar una biblioteca especializada en ello, para de esta manera enriquecer con este servicio a la biblioteca de visualización del Polo de RV de la facultad 5.

Para lograr esto se hizo un estudio de las distintas bibliotecas que son empleadas para el trabajo con archivos XML. Entre las más utilizadas se encuentran la Libxml, Domxml,

Libxslt, TinyXML entre algunas otras. Se decidió usar esta última ya que entre sus características fundamentales resaltan su simplicidad y facilidad de uso. Además está diseñada para ser rápida de aprender. Se adapta a distintos programas y está publicada bajo licencia de código abierto, lo que permite que sea usada tanto en software libre como software propietario.

2.6 Estándar de codificación.

El empleo de estándares de codificación ayuda a los ingenieros de software a producir código de calidad, así como aumentar su capacidad de mantenimiento y reutilización. Para llevar a cabo la implementación del sistema se determinó usar el estándar de codificación que propone el motor gráfico SceneToolKit. Está escrito en inglés por ser el idioma universal además de ser ampliamente generalizado en el mundo de la Informática. Se tiene en cuenta además los estándares de codificación para C++.

Nombre de los ficheros:

Se nombrarán los ficheros .h y .cpp de la siguiente manera:
STKNameOfUnits.cpp

Constantes:

Las constantes se nombran con mayúsculas, utilizándose "_" para separar las palabras:
MY_CONST_ZERO = 0;

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

Enumerados:

enum **E**MyEnum {**ME**_VALUE, **ME**_OTHER_VALUE};

Indicando con "E" que es de tipo enumerado. Nótese que las primeras letras de las constantes de enumerados son las iniciales del nombre del enumerado.

Estructuras:

struct SMyStruct {…};

Indicando con “S” que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

Clases:

class CClassName;

Indicando con “C” que es una clase

Interfaces:

IMyInterface

Indicando con “I” que es una interfaz.

Listas e iteradores STD:

vector<> **T**Name**List**;
TNameList::iterator **T**Name**ListIter;**
map<> **T**Name**Map**;
TNameMap::iterator **T**Name**MapIter**;
multimap<> **T**Name**MultiMap**;
TNameMultiMap::iterator **T**name**MultiMapIter**;

Declaración de variables:

Los nombres de las variables comenzarán con un identificador del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador “m_” (en minúscula), si son globales se les antepondrá la letra “g”, y en caso de ser argumentos de algún método, se les antepondrá el prefijo “*arg_*”.

Tipos simples:

bool bVarName;
int iName;

unsigned int uiName;

float fName;

char cName;

char* acName; // arreglo de caracteres

char* pcName; // puntero a un char

char** aacName; // bidimensional

char** apcName; // arreglo de punteros

bool m_bMemberVarName; //variable miembro

char gcGlobalVarName; //variable global, no se le antepone “”

short sName;

Instancias de tipos creados:

EMyEnumerated eName;

SMyStructure kName;

CClassName kObjectName;

CClassName* pkName; //puntero a objeto

CClassName* akName; //arreglo de objetos

CClassName* akName; // variable miembro de clase

IMyInterface* pIName; //puntero interfaces

Métodos:

En el caso de los métodos, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada. Solo los constructores y destructores omitirán el tipo de dato de retorno.

En el caso de los argumentos se les antepone el prefijo “*arg_*”.

Constructor y destructor:

CClassName (bool arg_bVarName, float& arg_fVarName);

~CClassName ();

Funciones:

bool bFunction1 (…);

int* piFunction2 (…);

```
CClassName* pkFunction3 (…);
```

Procedimientos:

```
void Procedure4 (…);
```

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán “Gets” y “Sets”, sino como los demás métodos, pero con el nombre de la variable a la que se accede y sin “m_”:

```
int m_iMyVar; //variable
```

Obtención del valor:

```
int iMyVar()
{
    return iMyVar;
}
```

Establecimiento del valor:

```
void MyVar(char* arg_iMyVar)
{
   m_iMyVar = arg_iMyVar;
}
```

Obtención y establecimiento del valor:

```
int& iMyVar()
{
   return m_iMyVar;
}
```

CAPÍTULO 3: CARACTERÍSTICAS Y DISEÑO DEL SISTEMA.

En este capítulo se exponen las características que va a tener el sistema. Se definen las reglas del negocio y el modelo de dominio, los requisitos funcionales y no funcionales. Se realiza el análisis del sistema que incluye los distintos diagramas de análisis previstos con el propósito de brindar una visión inicial de las distintas clases de diseño que exponen las características y responsabilidades de cada clase dentro del sistema. Posteriormente la fase de diseño que constituye un refinamiento de las fases anteriores.

3.1 Reglas del negocio.

1. Un elemento debe contener al menos un comportamiento.
2. El usuario que utilice la Aplicación QT para asignar los distintos comportamientos del entorno requiere un conocimiento profundo de la aplicación que se desea realizar.
3. El desarrollador que haga uso del módulo de comportamientos debe poseer sólidos conocimientos de programación e inteligencia artificial además de tener un estrecho vínculo con el usuario de la herramienta de asignación de comportamientos.
4. Los nuevos comportamientos a crear tienen que heredar de la clase genérica que brinda el sistema.
5. Cada tipo de elemento debe heredar de la clase genérica que brinda el sistema.

3.2 Glosario de términos del dominio.

AplicaciónQt: interfaz gráfica de usuario basada en *QtCreator* para crear los elementos y sus comportamientos dentro del entorno virtual. Así como la asignación de

comportamientos a los objetos dinámicos y estáticos del entorno.

Arista: son arcos o uniones creados entre un par de vértices o nodos, las cuales tendrán una dirección indicando la navegabilidad entre los nodos. Poseen un conjunto de vértices de control.

Vértice de control: estructuras que son utilizadas para deformar una arista y suavizar los caminos.

Nodo: estructuras que sirven de guías a la hora de crear caminos en el entorno 3D, son vértices o posiciones específicas del entorno que describen la secuencia de los caminos.

Grafo de Camino: hace referencia a un camino o trayectoria, que está integrado por un conjunto de aristas a lo largo de nodos predefinidos por el diseñador y que servirá para trasladar los agentes de una parte a otra en el mundo.

Fichero GraphML: fichero con la información referente al grafo de caminos.

Fichero DC: fichero con información referente a la asignación de comportamientos a elementos, aristas y nodos.

Mundo: entorno virtual que contiene la información gráfica de varios objetos 3D.

Elemento: objetos móviles que se comportan como agentes creíbles.

STK: motor gráfico para la visualización de objetos 3D.

Comportamiento: Conducta a seguir por los elementos para su movimiento dentro del mundo virtual.

3.3 Modelo de dominio.

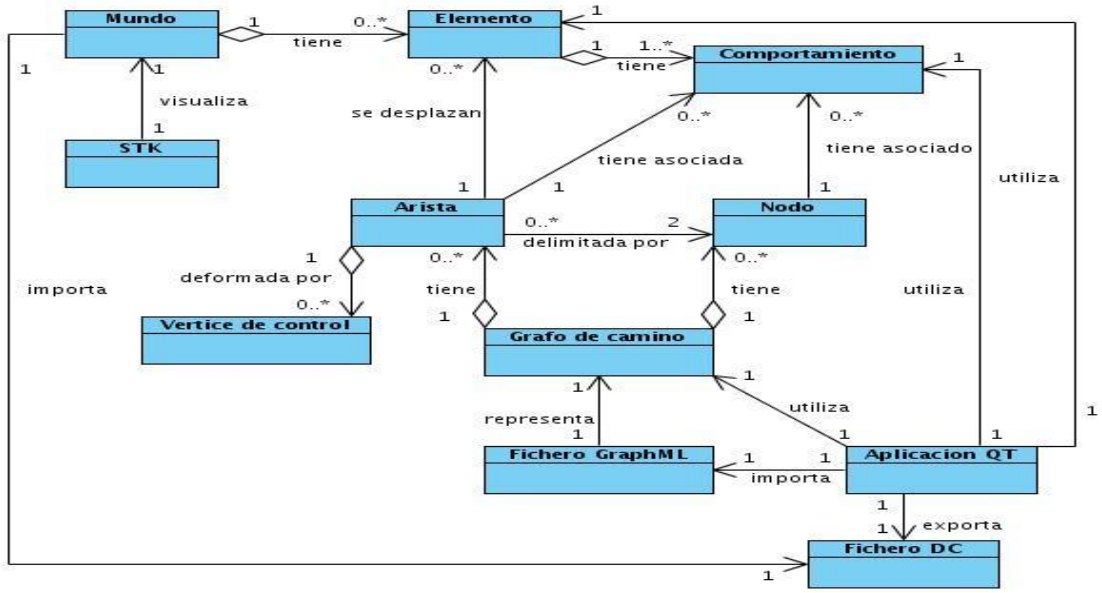

Figura 13: Modelo de dominio del sistema

En la figura 13 se muestra el modelo de dominio que representa la distribución conceptual de la herramienta a desarrollar. La Aplicación QT representa al módulo que se usará para asignar los distintos comportamientos a los objetos estáticos (aristas y nodos) y dinámicos del entorno. Esta información será guardada en el Fichero DC, el cual será exportado por la Aplicación QT y posteriormente cargado en el módulo de comportamientos para ser mostrado por una biblioteca de visualización, en este caso la STK, conjuntamente será leída la información referente al grafo del entorno representado en el Fichero GraphML. El Grafo de camino está compuesto por nodos y aristas y estas últimas serán deformadas por vértices de control.

3.4 Captura de requisitos.

A continuación se detallan los requisitos funcionales y no funcionales del sistema. Los

requisitos funcionales constituyen las capacidades o condiciones que el sistema debe cumplir y los no funcionales las propiedades o cualidades que el producto debe tener.

3.4.1 Requisitos funcionales

1. Gestionar fichero.
   1.1 Importar fichero
      1.1.2 Cargar fichero del mundo virtual.
      1.1.3 Cargar fichero del grafo.
   1.2 Exportar fichero del mundo virtual.
2. Gestionar comportamientos de locomoción
   2.1. Crear comportamiento.
      2.1.1. Asignar comportamiento a elementos, aristas y nodos.
      2.1.2. Aplicar comportamiento.
3. Crear escena con todos los objetos que se definan en el mundo virtual.
4. Crear elemento del mundo virtual.

3.4.2 Requisitos no funcionales

3.4.2.1 Requisitos de software.

1. Sistema operativo Linux, distribución Debian, Kernel 2.6 con entornos de escritorio KDE o GNOME.
2. Freeglut 3.0 o superior.
3. Lenguaje de programación C++.
4. Uso del paradigma de programación Orientado a Objetos.
5. Empleo del IDE Code::Blocks para programar.
6. Para diseñar la interfaz gráfica se hace uso de la aplicación QTCreator 1.0.
7. Empleo de la biblioteca gráfica Scene Toolkit para la visualización del entorno virtual.
8. Uso de la STL de C++ para representar las estructuras de datos.

3.4.2.2 Requisitos de Hardware

1. PC Pentium 4 de 2.4 GHz y 512 MB de memoria RAM.
2. Tarjeta aceleradora de gráficos Nvidia de 128 MB o superior.

3.4.2.3 Requisitos de soporte.

1. En su primera versión debe ser compatible con el sistema operativo Linux en su distribución Debian, pero deberá estar preparada para que con mínimas modificaciones pueda migrar a otras como Ubuntu y Nova, así como la plataforma Windows.

3.4.2.4 Requisitos de rendimiento.

1. Alto grado de velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación y disponibilidad.

3.4.2.5 Requisitos de apariencia.

1. El módulo de interfaz de usuario debe poseer una interfaz amigable y de fácil interacción.
2. Se debe proporcionar un menú con las funcionalidades.

3.5 Modelo de casos de uso del sistema.

3.5.1 Casos de uso del sistema.

Se enumeran a continuación los casos de uso del sistema, los cuales se corresponden con los requisitos funcionales descritos con anterioridad.

1. Cargar grafo.
2. Crear comportamiento.
3. Asignar comportamiento
4. Exportar definición de comportamiento.
5. Cargar definición de comportamiento.
6. Aplicar comportamiento.

3.5.2 Actores del sistema.

En la siguiente tabla se detallan los actores que intervienen en el sistema.

Actores	Justificación
Desarrollador	Es quien hará uso y se beneficiará de las funcionalidades que brinda el sistema: Tendrá la responsabilidad de crear comportamientos, así como cargar desde fichero la información referente al grafo y la asignación de comportamientos.
Asignador de comportamiento	Define las relaciones entre los comportamientos y cada elemento del entorno las cuales son exportadas en un fichero.
Sistema	En tiempo de ejecución aplica los comportamientos a los distintos elementos del entorno.

Tabla 3: Actores del sistema.

3.5.3 Diagrama de casos de uso del sistema.

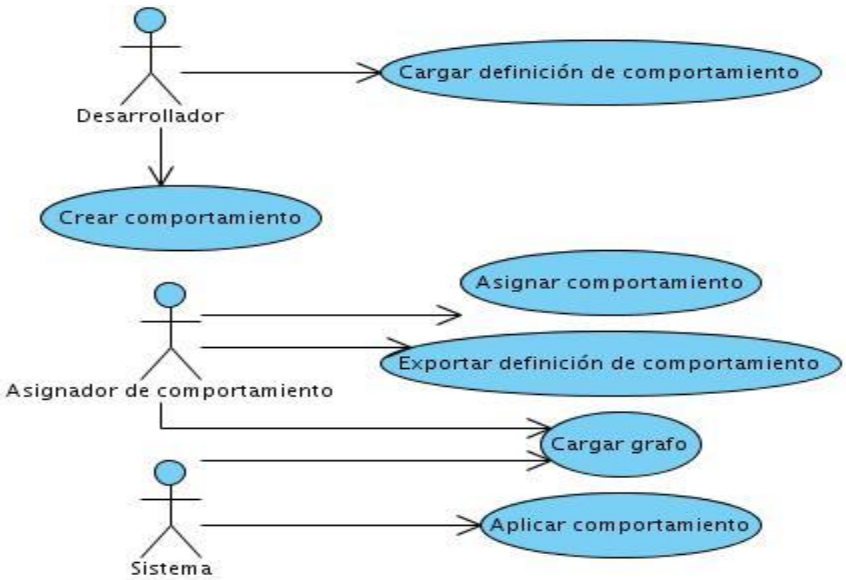

Figura 14: Diagrama de casos de uso del sistema

En la figura 14 se detalla el diagrama de casos de uso del sistema, donde se establece la relación existente entre los actores y los casos de uso del sistema. La relación entre el actor Desarrollador y los casos de usos: Cargar definición de comportamiento y Crear comportamiento tiene lugar en el módulo de comportamientos. Por su parte la relación entre el actor: Asignador de comportamiento y sus respectivos casos de usos asociados se corresponde a las acciones a desarrollar en la Aplicación QT. El caso de uso Aplicar comportamiento solo se ejecuta en tiempo real por el sistema.

3.5.4 Descripción de los casos de uso del sistema.

Caso de uso:	Cargar definición de comportamiento
Actor(es):	Desarrollador (inicia).
Propósito:	Obtener información referente a la definición de los comportamientos y sus elementos del entorno.
Resumen:	El caso de uso inicia cuando el usuario accede a la opción “Cargar definición”, donde deberá especificar el camino donde se encuentra el fichero con dicha información. Si no existe ningún problema en el proceso de lectura del fichero, se almacena la información de forma persistente.
Referencias:	RF1.1.2
Curso Normal de los Eventos	
Acción del Actor	Respuesta del Sistema
1 – Selecciona la opción “Cargar definición”.	1.1 – Solicita el nombre del fichero y el camino donde se encuentra.

2 – Especifica los datos solicitados.	2.1 – Verifica la existencia del fichero y si tiene extensión válida. 2.3 – Se verifica la validez del fichero en el encabezado XML y se carga la información.
Cursos Alternos	
2.1 – Si el fichero no existe o su extensión no es una de las esperadas se informa al usuario. 2.3 – Si el fichero no es válido se detiene la lectura del fichero y se informa al usuario.	
Post-condiciones:	Cargada en memoria la definición de los comportamientos almacenado en fichero.
Prioridad:	Crítico.

Tabla 4: Descripción del caso de uso "Cargar definición de comportamiento"

Caso de uso:	Cargar grafo
Actor(es):	Asignador de comportamiento (inicia).
Propósito:	Obtener información referente a las aristas y los nodos del entorno para poder asignarle comportamientos.
Resumen:	El caso de uso inicia cuando el usuario accede a la opción “Cargar grafo”, donde deberá especificar el camino donde se encuentra el fichero con dicha información. Si no existe ningún problema en el proceso de lectura del fichero, se almacena la información de forma persistente. En caso contrario se notifica al Asignador el problema acontecido.
Referencias:	RF1.1.2

Curso Normal de los Eventos	
Acción del Actor	Respuesta del Sistema
1 – Selecciona la opción “Cargar Grafo”.	1.1 – Solicita el nombre del fichero y el camino donde se encuentra.
2 – Especifica los datos solicitados.	2.1 – Verifica si tiene extensión válida. 2.3 – Se carga la información.
Cursos Alternos	
2.1 – Si su extensión no es la esperada se informa al usuario y se detiene la lectura del fichero.	
Post-condiciones:	Cargada en memoria la estructura del grafo almacenado en fichero.
Prioridad:	Crítico.

Tabla 5: Descripción del caso de uso "Cargar grafo"

Caso de uso:	Asignar comportamiento
Actor(es):	Asignador de comportamiento (inicia).
Propósito:	Asignar los comportamientos de locomoción a los distintos elementos del entorno.
Resumen:	El caso de uso inicia cuando el Asignador de comportamiento selecciona al elemento, arista y/o nodo al cual desea asignarle comportamiento y a continuación escoge el comportamiento deseado de la lista de comportamientos disponible.

Referencias:	RF2.
Curso Normal de los Eventos	
Acción del Actor	Respuesta del Sistema
1 – Selecciona la opción “Asignar comportamiento”.	1.1 –Solicita al actor escoger tipo de objeto al cual se le va a asignar comportamiento “Elemento”, “Arista” o “Nodo”.
2 – Escoge el tipo de objeto	2.1 – Muestra el listado con los objetos disponibles del tipo seleccionado.
3 -- Selecciona el objeto al cual desea asignar comportamiento.	3.1 – Muestra el listado de comportamientos disponibles
4-- Escoge el comportamiento deseado y selecciona la opción “Asignar”	4.1– Guarda en memoria la información referente a la asignación 4.2– Termina el caso de uso.
Cursos Alternos	
4.1 – Si existe algún problema con el almacenamiento de la información se le informa al usuario.	
Pre-condiciones	Existencia del comportamiento solicitado así como de los objetos a los cuales se les asignará dicho comportamiento.
Post-condiciones:	El comportamiento fue asignado a uno o varios elementos, aristas o nodos.
Prioridad:	Crítico.

Tabla 6: Descripción del caso de uso "Asignar comportamiento"

Caso de uso:	Exportar definición de comportamiento
Actor(es):	Asignador de comportamiento (inicia).
Propósito:	Salvar en fichero la información referente a la definición de los comportamientos y sus elementos.
Resumen:	El caso de uso inicia cuando el usuario accede a la opción “Exportar definición”, donde deberá especificar el camino donde se guardará el fichero con dicha información. Si no existe ningún problema en el proceso de escritura del fichero, se almacena la información de forma persistente. En caso contrario se notifica al Asignador el problema acontecido.
Referencias:	RF1.2

Curso Normal de los Eventos	
Acción del Actor	Respuesta del Sistema
1 – Selecciona la opción “Exportar definición”.	1.1 – Solicita el nombre del fichero y el camino donde se encuentra.
2 – Especifica los datos solicitados.	2.1 – Crea el fichero de definición en el lugar definido por el usuario. 2.2 – Termina el caso de uso.
Cursos Alternos	
2.1 – Si el fichero no existe o su extensión no es válida se informa al usuario.	

Post-condiciones:	Creada la escena del entorno.
Prioridad:	Crítico.

Tabla 7: Descripción del caso de uso "Exportar definición de comportamiento"

Caso de uso:	Crear comportamiento
Actor(es):	Desarrollador (inicia).
Propósito:	Programar los comportamientos que sean necesarios cumplir por los elementos del entorno.
Resumen:	El caso de uso inicia cuando el desarrollador del sistema implementa un nuevo comportamiento para elementos determinados, como una nueva clase que herede de la clase genérica **Comportamiento** brindada por el sistema.
Referencias:	RF2.1.

Curso Normal de los Eventos	
Acción del Actor	Respuesta del Sistema
1 – Crea nueva clase con implementación del nuevo comportamiento.	1.1-- El sistema guarda información de la nueva clase.

Post-condiciones:	Se ejecuta correctamente el comportamiento al correr la aplicación.
Prioridad:	Crítico.

Tabla 8: Descripción del caso de uso "Crear comportamiento"

Caso de uso:	Aplicar comportamiento
Actor(es):	Sistema (inicia).
Propósito:	Permitir que los elementos se muevan por el entorno de acuerdo al comportamiento deseado.
Resumen:	El caso de uso inicia cuando el sistema decide aplicar el (los) comportamientos a uno o varios elementos del entorno.

Referencias:	RF2.1.
Curso Normal de los Eventos	
Acción del Actor	Respuesta del Sistema
1 – El sistema decide dada las condiciones del ambiente aplicar el comportamiento.	1.1-- El sistema genera a partir de las clases definidas, el comportamiento solicitado y lo aplica.
Pre-condiciones:	El sistema se está ejecutando.
Post-condiciones:	Ejecución correcta del comportamiento.
Prioridad:	Crítico.

Tabla 9: Descripción del caso de uso "Aplicar comportamiento"

3.6 Modelo de análisis.

3.6.1 Diagrama de clases de análisis.

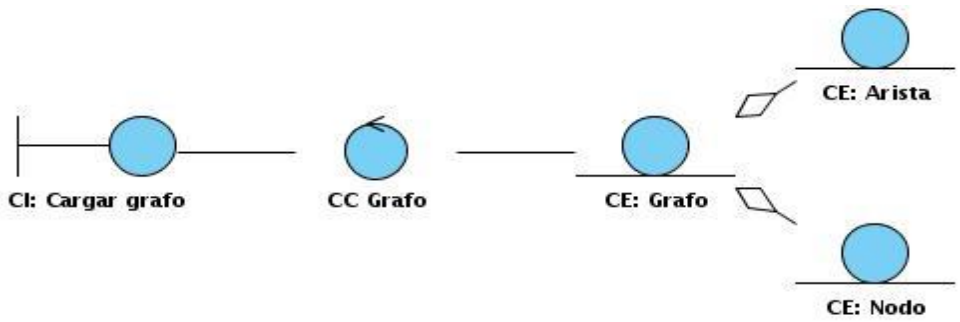

Figura 15: Diagrama de clases de análisis CU: Cargar grafo

La figura 15 se corresponde al diagrama de clases de análisis para el CU: Cargar grafo, la clase CI: Cargar grafo es la interfaz visual que interactúa con el actor para llevar a cabo el

caso de uso.

Figura 16: Diagrama de clases de análisis CU: Crear comportamiento

En el diagrama de la figura 16 se relacionan las clases pertenecientes al diagrama de clases de análisis del CU: Crear comportamiento. En esta ocasión no existe ninguna interfaz visual debido a que el desarrollador del sistema interactúa directamente con la clase creadora del comportamiento.

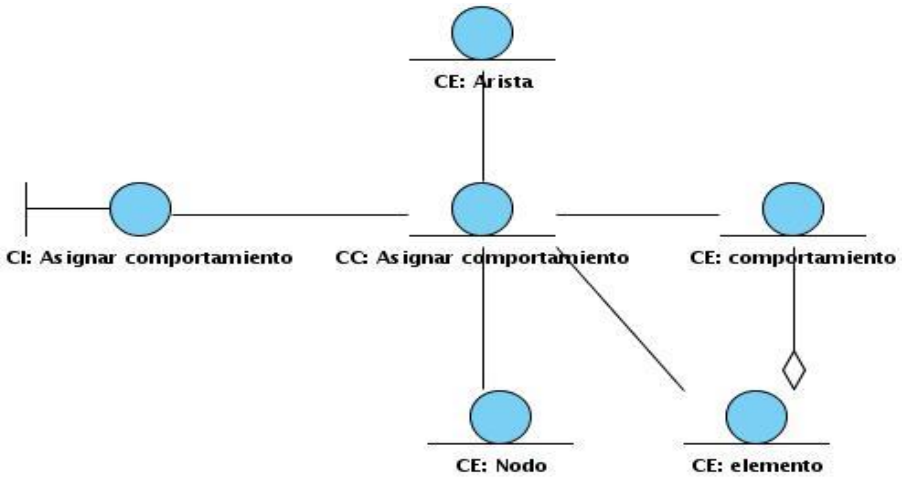

Figura 17: Diagrama de clases de análisis CU: Asignar comportamiento

La figura 17 representa el diagrama de clases de análisis correspondiente al CU Asignar comportamiento. El Asignador de comportamiento accede a la interfaz "CI: Asignar comportamiento" y esta a su vez le informa a la clase controladora "CC: Asignar comportamiento" la petición del actor de este caso de uso. Luego la clase controladora accede a la clase "CE: Comportamiento" y ejecuta las acciones pertinentes, accediendo a las clases "CE: Arista", "CE: Nodo" o "CE: Elemento" según corresponda.

Figura 18: Diagrama de clases de análisis CU: Exportar definición de comportamiento

En la figura 18 se relacionan las clases miembros del diagrama de clases de análisis para el CU: Exportar definición de comportamiento. Para ello el actor Asignador de comportamiento accede a la clase "CI: Exportar definición de comportamiento" la cual se relaciona con la clase controladora "CC: Exportar definición de comportamiento" encargada de llevar a cabo la acción indicada, para lo cual accede a la clase "CE: Fichero DC".

3.7. Modelo de diseño.

3.7.1. Diagrama de paquetes del diseño.

Para un mejor entendimiento se separó el diseño del sistema en dos paquetes, el paquete Aplicación QT relacionado con la parte visual del sistema, donde se asignan los comportamientos a los elementos, aristas y nodos. El paquete Módulo de comportamiento que representa la aplicación principal donde se crearán los nuevos comportamientos y se apliquen dichos comportamientos en tiempo real. Con el objetivo de lograr mayor extensibilidad en el sistema como se abordó en el capítulo 2, el módulo de comportamiento cuenta con 2 paquetes, el paquete de IA y el grafo de camino.

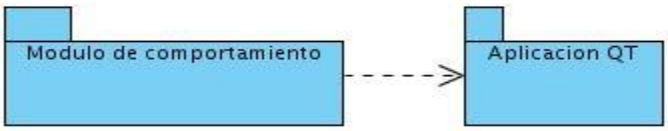

Figura 19: Diagrama de paquetes del diseño

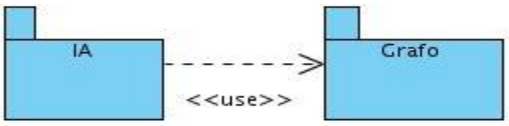

Figura 20: Diagrama de paquetes del módulo de comportamiento.

3.7.2 Diagramas de clases del diseño.

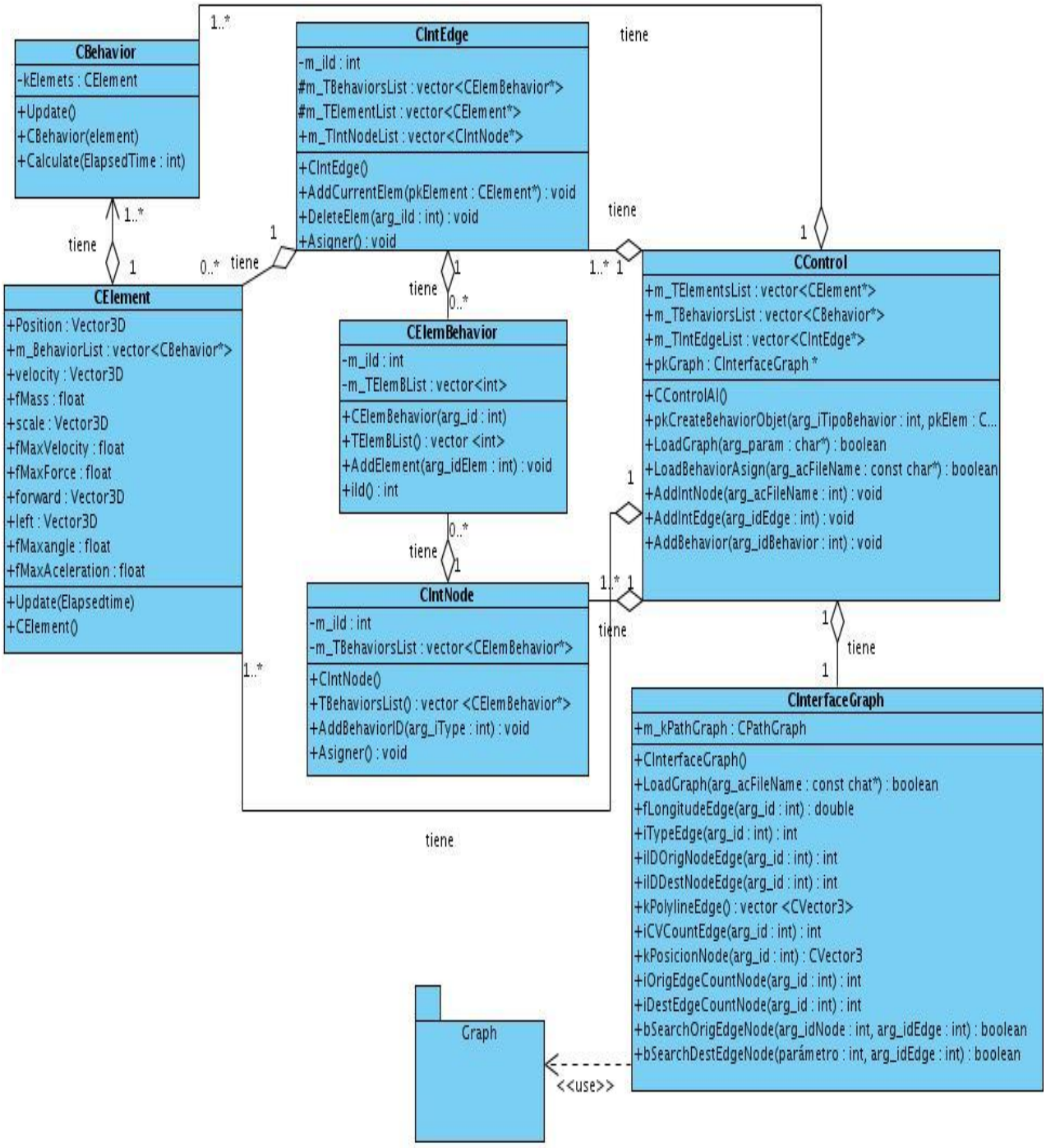

Figura 21: Diagrama de clases del diseño Módulo de comportamiento.

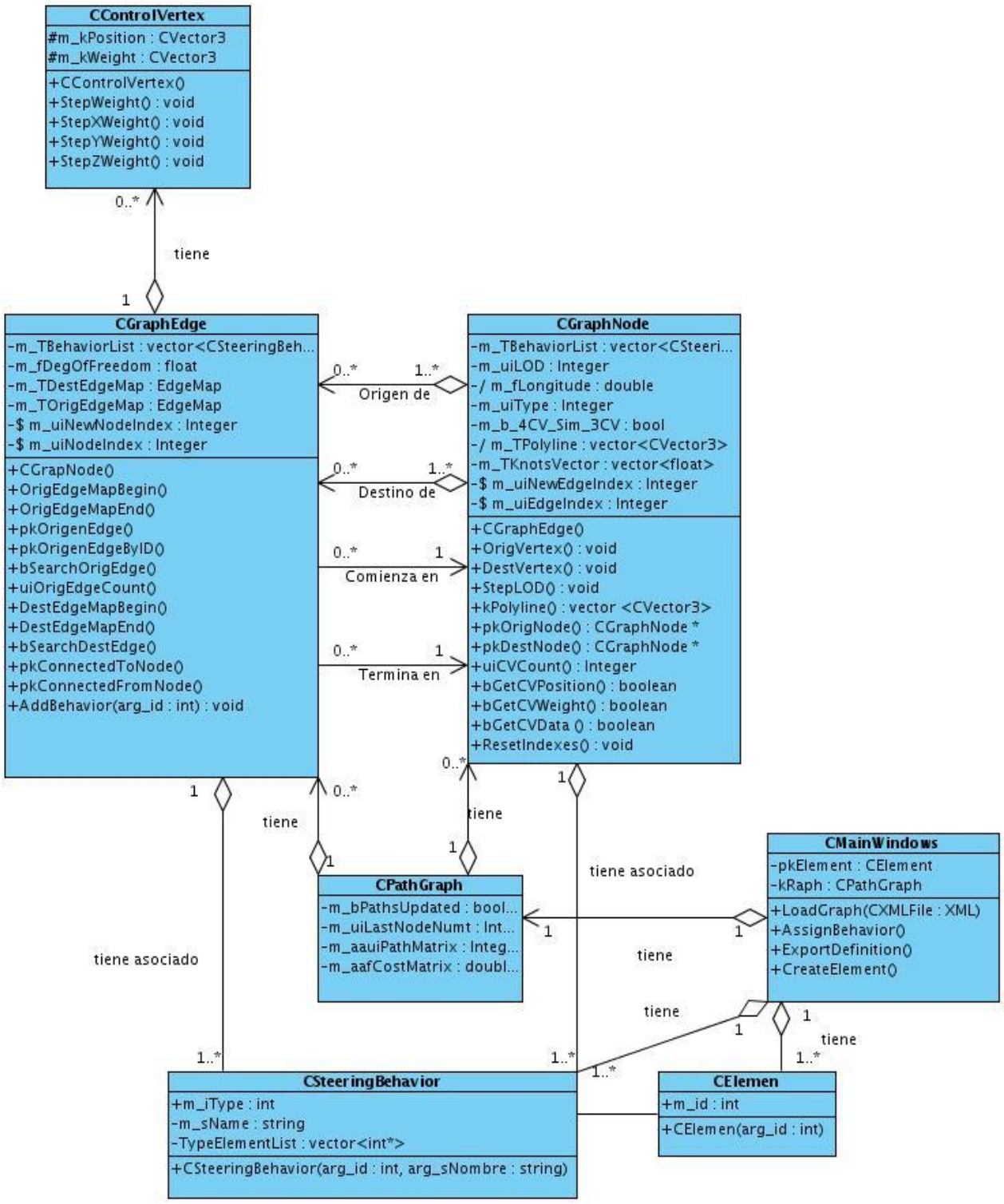

Figura 22: Diagrama de clases del diseño "Paquete: Aplicación QT"

3.7.3 Diagramas de secuencia del diseño.

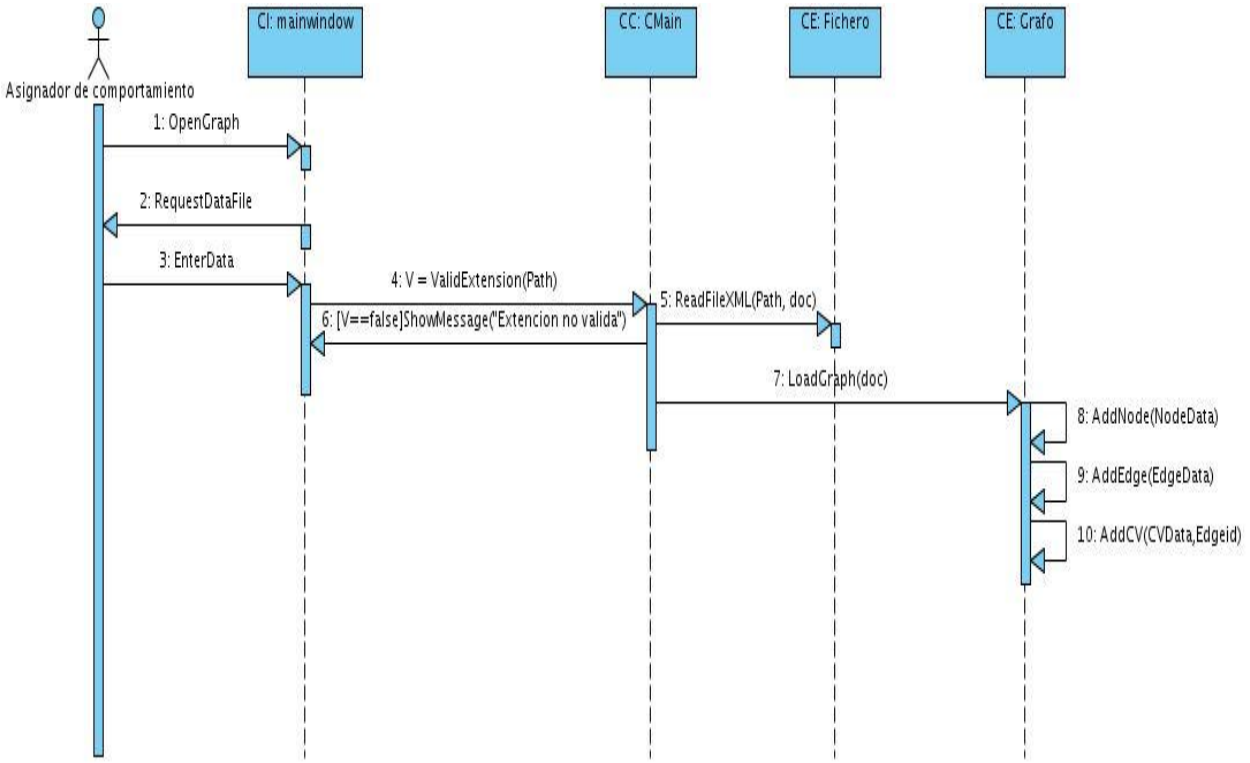

Figura 23: Diagrama de secuencia "CU: Cargar grafo"

En el diagrama de secuencia para el CU: Cargar grafo representado en la figura 23, se muestran los pasos a seguir para desarrollar el caso de uso. El Asignador de comportamiento interactúa con la clase interfaz MainWindow seleccionando del menú la opción “Cargar grafo”. A continuación la clase controladora CMain ejecuta las funcionalidades necesarias relacionándose para ello con las clases Fichero y Grafo a quienes serán adicionadas los nodos, aristas y vértices de control que lo componen.

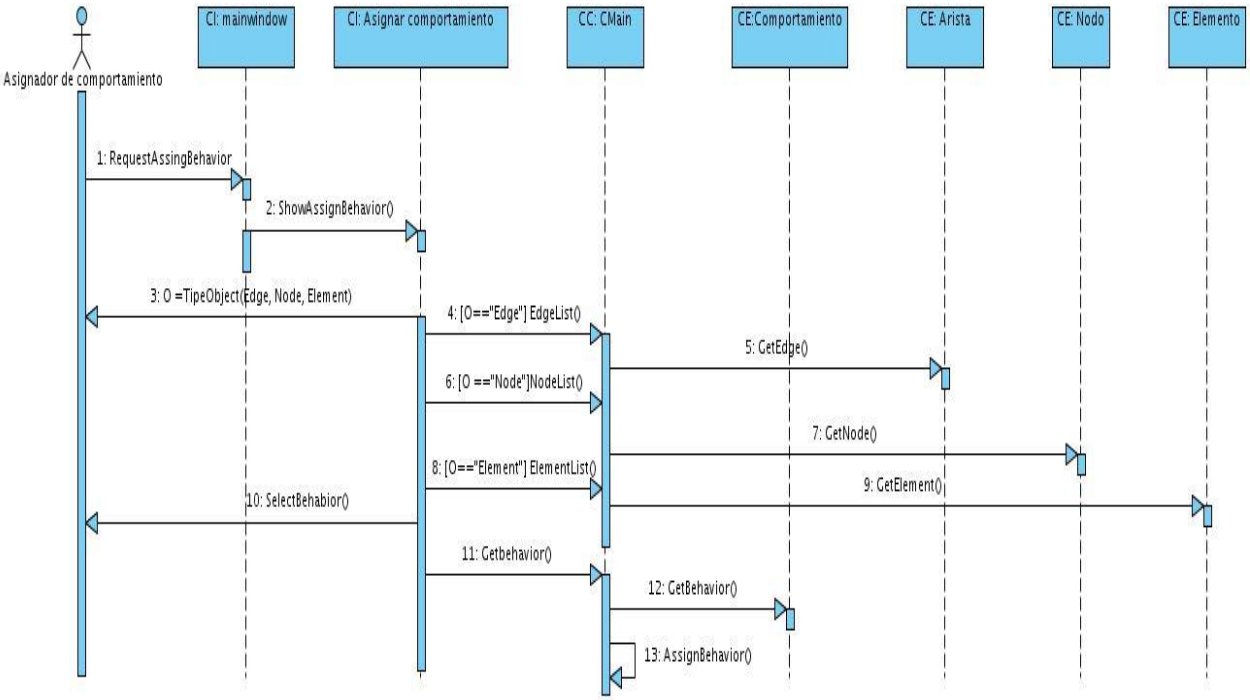

Figura 24: Diagrama de secuencia "CU: Asignar comportamiento"

La figura 24 representa al diagrama de secuencia del diseño para el CU: Asignar comportamiento. El usuario interactúa con la clase interfaz MainWindow, el actor selecciona a quien desea asignarle comportamientos, si es un nodo, arista o elemento y en dependencia de ello se procede a la asignación del comportamiento, relacionándose para ello la clase controladora con las clases Nodo, Arista o Elemento según corresponda y con la clase Comportamiento.

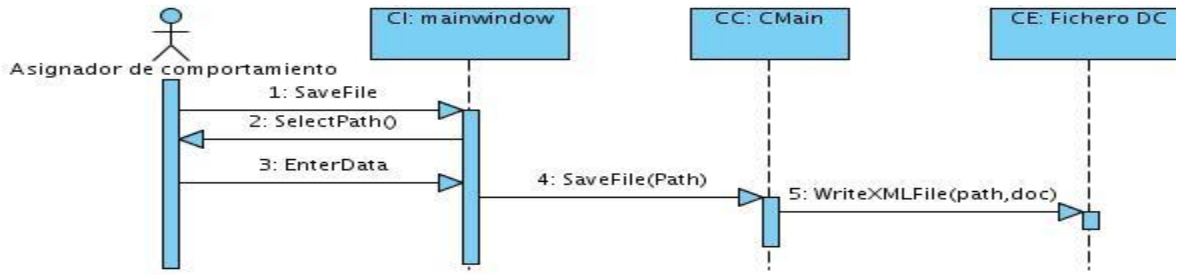

Figura 25: Diagrama de secuencia "CU: Exportar definición de comportamiento"

En la figura 25 se muestra a través del diagrama de secuencia correspondiente al CU: Exportar definición de comportamiento, las secuencia de pasos que se realizan para llevar a cabo este caso de uso. El actor Asignador de comportamiento selecciona la opción “Salvar fichero” del menú en la ventana principal del módulo Aplicación QT, La clase principal “MainWindow” ejecuta las funcionalidades necesarias interactuando en el Main con la clase Fichero DC.

Al terminar este capítulo se obtiene el diseño completo del sistema y la secuencia de pasos traducida a mensajes entre clases correspondientes a los casos de uso a desarrollar.

CONCLUSIONES

Se logra implementar una herramienta que permite incorporar comportamientos a elementos virtuales que se mueven sobre un grafo de camino en entornos 3D. Estos elementos logran cumplir con las características fundamentales definidas para los agentes, cumpliendo objetivos propios, reaccionando a cambios o a situaciones presentes en el entorno y comunicándose con otros agentes. La distribución de la inteligencia por el entorno, incorporándole a cada arista y cada nodo los comportamientos que deben cumplir los elementos que entren en su área de acción es uno de los mayores aportes de del presente trabajo, permitiendo agilizar el proceso y brindar posibilidades adicionales para definir comportamientos nuevos sin modificar los agentes.

Se utilizó para el trabajo con fichero el estándar XML y se incorpora la biblioteca TinyXML garantizando la portabilidad de la herramienta. Con este mismo objetivo se implementó una interfaz para el acceso al grafo de camino y con ligeras modificaciones la presente herramienta puede ser utilizada por otras bibliotecas gráficas además de la STK.

Su arquitectura facilita incorporar otras técnicas derivadas de modelos bioinspirados para ponerse en práctica, como las redes neuronales, las colonias de hormigas y los algoritmos genéticos, así como el algoritmo de reubicación necesario para lograr un uso eficiente de los recursos de hardware.

RECOMENDACIONES

- Enriquecer el módulo de comportamientos con la incorporación de nuevos modelos bioinspirados.
- Incorporar el algoritmo de reubicación referenciado en este trabajo.
- Perfeccionar la interfaz visual de la Aplicación QT.

GLOSARIO DE TÉRMINOS

E:

Engine (biblioteca o motor gráfico): Componente principal de un videojuego u otra aplicación interactiva con gráficos en tiempo real.

Entornos virtuales o mundos virtuales: se trata de la simulación de mundos o entornos, denominados virtuales, en los que el hombre interacciona con la máquina en entornos artificiales semejantes a la vida real.

G:

Grafo de camino: Grafo usado en entornos virtuales para brindar información acerca de la estructura del entorno y para la aplicación de algoritmos de inteligencia artificial como la búsqueda de caminos.

I:

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

M:

Modelos bioinspirados: Representación computacional de algunos elementos de la naturaleza.

O:

OpenGL: Es una especificación estándar que define una interfaz de programación de aplicaciones multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

R:

Realidad Virtual: es un sistema o interfaz informático que genera entornos sintéticos en tiempo real, representación de las cosas a través de medios electrónicos o representaciones de la realidad, una realidad ilusoria, pues se trata de una realidad perceptiva sin soporte objetivo, existe sólo dentro del ordenador.

Renderizado: proceso de generar una imagen desde un modelo.

REFERENCIAS BIBLIOGRÁFICAS

[Arkin, 1987]. Arkin, Ronald (1987) “Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming by Behavior”, Proceedings of IEEE Conference on Robotics and Automation, 1987.

[Arkin, 1989]. Arkin, Ronald (1989) “Motor Schema-Based Mobile Robot Navigation”, International Journal of Robotics Research, 1989.

[Arkin, 1992]. Arkin, Ronald (1992) “Behavior-based Robot Navigation in Extended Domains”, Journal of Adaptive Behavior, 1992.

[Ben, 2003]. Ben, Heni. Obst, Oliver. Murray, Jan. Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents, 2003.

[Bratton, 2007]. Bratton, Daniel. Kennedy, James. Defining a Standard for Particle Swarm Optimization, 2007.

[Brooks, 1985]. Brooks, Rodney A. "A Robust Layered Control System for a Mobile Robot," IEEE Journal of Robotics and Automation 2(1), March 1986, pp. 14--23; also MIT AI Memo 864, September 1985. Disponible en: http://www.ai.mit.edu/people/brooks/papers/AIM-864.pdf

[Buckland, 2005]. Buckland, Mat. Programming Game AI by Example, 2005.

[Cedeño, 2008]. Cedeño González, Yessy. Hernandez Gil, Ismael. *Herramienta de creación de grafos de caminos para la biblioteca ““SceneToolkit”*, Tesis de grado, Universidad de las Ciencias Informáticas ,2008.

[Charles, 2008]. L. . M. Charles, Fyfe, Ant Colony Optimization, IGI Global. (2008).

[Coca, 2007]. Coca Bergolla, Yuniesky. “Control del tránsito en un simulador de auto.” Dpto. Práctica Profesional e Ingeniería de Software facultad 5, Universidad de las Ciencias Informáticas, 2007.

[Coca, 2008]. Coca Bergolla, Yuniesky. *Agentes inteligentes. Aplicación a la Realidad virtual*, 2008.

[Costa, 1990]. Costa, Mônica; Feijó, Bruno; and Schwabe, Daniel (1990) Reactive Agents in Behavioral Animation, Anais do SIBGRAPI 95, Lotufo and Mascarenhas editors, São Carlos. Disponible en: http://www.icad.puc-rio.br/~monica/pubs.html

[Falcón, 2008]. Falcón García, Ricardo Ernesto. *Módulo de algoritmos de locomoción con múltiples Steering Behaviors*, Tesis de grado, Universidad de las Ciencias Informáticas ,2008.

[Ferver, 1999]. J. Ferber "Multi-Agent Systems" Addison-Wesley. 1999.

[Gilimas, 2008]. Gilimas Alvarez, Reinier. Alemañy Socarras, Leoder. *Módulo de clases para la modelación del comportamiento físico-matemático de autos,* Tesis de grado, Universidad de las Ciencias Informáticas ,2008.

[Hawkins, 2007]. Hawkins, Kevin. Astle, Dave. Chapter 20 - Building a Game Engine. OpenGL Game Programming, 2007.

[Heng, 2006]. Heng, Xing Chen. Qin, Zheng. Wang, Xian Hui. Shao, Li Ping. Research on learning Bayesian Networks by Particle Swarm Optimization. Research Institute of Computer Software of Xi an Jiaotong University, Xian Shanxi, People´s Republic of China, 2006.

[Lombera, 2008]. Lombera Rodríguez, Hassán. Pacheco Allende, Alberto Eliseo. *Edición de sistemas articulados de cuerpos rígidos para las animaciones en los videojuegos*, Tesis de grado, Universidad de las Ciencias Informáticas ,2008.

[Jacobson, 1999]. Jacobson, Booch, Rumbaugh. El Proceso Unificado de Desarrollo de Software ,1999.

[Jiménez, 2004]. Jiménez López, Fernando. Camacho Román, Yanoski Rogelio. *Biblioteca gráfica para sistemas de realidad virtual,* 2004.

[Jiménez, 2008]. Jiménez Garzón, Darwin. Ingeniería de Software II, 2008.

[Marrero, 2007]. Marrero Borges, Irina.Parra Nápoles, Yoan. *Incorporación de algoritmos genéticos a la biblioteca de inteligencia artificial.* Tesis de grado, Universidad de las Ciencias Informáticas, 2007.

[Martín, 2001]. Martín, Bonifacio. Sanz, Alfredo. Redes Neuronales y Sistemas Difusos. 2da Edición ampliada y revisada, 2001.

[McGuire, 2007]. McGuire, Morgan. G3D Engine ScreenShots. *G3D Engine* [En línea] 2007. http://g3d-cpp.sourceforge.net/screenshots.html.

[Mclean, 2002]. Alex W. Mclean. An efficient AI Architecture Using Prioritized Task Categories, 2002.

[Nogueira, 2007]. Nogueira, Mariela. Correa, Omar. Algoritmos para la Manipulación

Eficiente de Objetos Dinámicos en Escenas Virtuales Urbanas. Tesis de grado, Universidad de las Ciencias Informáticas, 2007.

[Oconor, 2008]. Oconor, Dalila. Búsqueda de caminos en entornos virtuales. Tesis de grado, Universidad de las Ciencias Informáticas, 2008.

[Pfeil, 2006]. Pfeil, Jonas. Swarm Intelligence. Communication and Operating Systems Group . Berlin University of Technology, 2006.

[Pottinger, 1999]. Pottinger, Dave (1999) "Coordinated Unit Movement" and "Implementing Coordinated Movement", Game Developer Magazine, January, 1999. See:
http://www.gamasutra.com/features/game_design/19990122/movement_01.htm
http://www.gamasutra.com/features/game_design/19990129/implementing_01.htm

[Prevot, 2008]. Prevot Urgellés, Yunerkkis. Herrera Bacallao, Arianna. *Aplicaciones de las redes neuronales en entornos virtuales.* Tesis de grado, Universidad de las Ciencias Informáticas, 2008.

[Puig, 2008]. Puig, Frank. Coca, Yuniesky. Miguelevich, Andrés. Castro, Miguel. Herramientas de Edición y Análisis Topográfico de Estructuras de Navegación, 2008.

[Rabuñal, 2006]. Rabuñal, juan R. Dorado, Julián. Artificial Neural Networks in Real-Life Applications. IDEA GROUP PUBLISHING, 2006.

[Reynolds, 1988]. Reynolds, C. W. (1988) Not Bumping Into Things, in the notes for the SIGGRAPH 88 course Developments in Physically-Based Modeling, pages G1-G13, published by ACM SIGGRAPH. Disponible en:
http://www.red.com/cwr/nobump/nobump.html

[Reynolds, 1999]. Reynolds, Craig. "Steering Behaviors For Autonomous Characters" [En linea] Game developers conferences. 1999. Disponible en:
http://www-eco.enst-bretagne.fr/~phan/complexe/steer/index.htm

[Santana, 2006]. Santana Quintero, Luis V. Ramírez Santiago, Noel. Coello Coello, Carlos. A New Proposal for Multiobjective Optimization using Particle Swarm Optimization and Rough Sets Theory, 2006.

[Sánchez, 2008]. Sánchez Cruz, Dalian. Herrera Pérez, Alexei .*Comportamientos de autos en pistas de carreras basados en Steering Behaviors para el videojuego "Rápido y Curioso"*, Tesis de grado, Universidad de las Ciencias Informáticas ,2008.

[Saurín, 2008]. Saurín, Gelson. Benítez, Alejandro. Módulo para el comportamiento

autónomo de autos en un entorno virtual urbano. Tesis de grado, Universidad de las Ciencias Informáticas ,2008.

[Settles, 2005]. Settles, Matthew. An Introduction to Particle Swarm Optimization. Department of Computer Science, University of Idaho, Moscow, Idaho U.S.A, 2005.

[Skilton, 2008]. Skilton, Frank**.** 3D Engines Database, 2008. [En línea]. Disponible en: http://www.devmaster.net/engines

[Smith, 2007]. Smith, Russel**.** ODE Documentation. *Open Dynamics Engine* [En línea], 2007. Disponible en: http://www.ode.org.

[Stout, 2000]. Stout, W.B. The Basics of A* Path Planning, 2000, Game Programming Gems 1: 254 – 263.

[Tirado, 2007]. Tirado Granados, Gonzalo .Introducción a OGRE 3D, 2007. [En línea]. Disponible en: http://tamudo84.googlepages.com/IntroduccinaOgre-GonzaloTirado-Defin.pdf

[Trujillo, 2008]. Trujillo Rivero, Andy. Márquez Rodríguez, Marvyn Amado. Definición del comportamiento de carros autónomos en un videojuego de carreras empleando redes neuronales artificiales. Tesis de grado, Universidad de las Ciencias Informáticas, 2008.

[Tu, 1994]. Tu, X. D. Terzopoulos. 1994. Artificial fishes: Physics, locomotion, perception, behavior. In Proceedings of SIGGRAPH '94 , Computer Graphics, July.

[Vela, 2008]. Vela Díaz, Yordan Edilberto. Ruiz Romero, Alberto Enrique. *DLL Intelligent Soccer Juego de fútbol para jugadores virtuales*, Tesis de grado, Universidad de las Ciencias Informáticas ,2008.

[Wang, 2004]. Wang, Hongling. Kearney, Joseph K. Cremer, James. Willemsen, Peter. Steering Autonomous Driving Agents Through Intersections in Virtual Urban Environments, 2004.

[Xiaoyuan, 1994]. Tu, Xiaoyuan and Terzopoulos, Demetri (1994) "Artificial Fishes: Physics, Locomotion, Perception, Behavior", Proceedings of SIGGRAPH 94, Computer Graphics Proceedings, Annual Conference Series, Andrew Glassner editor, ACM SIGGRAPH, ISBN 0-89791-667-0, pages 43-50. Disponible en: http://www.dgp.toronto.edu/people/tu/papers/sig94.ps

[Xiaoyuan, 1996]. Tu, Xiaoyuan (1996) Artificial Animals for Computer Animation:

Biomechanics, Locomotion, Perception, and Behavior, PhD dissertation, Department of Computer Science, University of Toronto. Disponible en: http://www.dgp.toronto.edu/people/tu/thesis/thesis.html

[Zeltzen, 1983]. Zeltzer, David (1983) Knowledge-Based Animation, Proceedings SIGGRAPH/SIGART Workshop on Motion, 1983.

[Zeltzer, 1990]. Zeltzer, David (1990) "Task Level Graphical Simulation: Abstraction, Representation and Control," in Making Them Move: Mechanics, Control and Animation of Articulated Figures, N. Badler, B. Barsky and D. Zeltzer, editors. Morgan Kaufmann Publishers, 1990.