Universidad de las Ciencias Informáticas

Módulo de algoritmos de locomoción con múltiples *Steering Behaviors*

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Ricardo Ernesto Falcón García

Tutor: Ing. Frank Puig Placeres

Ciudad de la Habana
Julio, 2008

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo, y autorizamos al Proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autor:

Ricardo Ernesto Falcón Garcia

_____

Tutor:

Frank Puig Placeres

_____

Datos de Contacto

Ing. Frank Puig Placeres

Graduado en Ingeniería en Ciencias Informáticas, 2 años de graduado. Profesor en la Facultad 5 de la Universidad de las Ciencias Informáticas. 10 años de experiencia.

Coautor de los libros SHADERX5,Game Programming Gems 5, Game Programming Gems 6 y AI Wisdoms 4.

E-mail: fpuig@uci.cu

Agradecimientos

A mi tutor Frank Puig por ayudarme a realizar este trabajo.

A mis compañeros del grupo que compartieron conmigo los mejores momentos en estos 5 años de carrera.

A mis amigos, los que me ayudaron y los que no pero de una forma u otra contribuyeron en este trabajo.

Y quiero agradecerle también a toda mi familia que siempre que los necesito se que puedo contar con ellos.

Dedicatoria

A mi abuelo Manengue, por su preocupación y su apoyo en todo momento, porque más que un abuelo ha sido como un padre para mi y porque lo quiero mucho.

A mis abuelos Juana y Quino porque en estos meses de trabajo en esta tesis no he podido ir a verlos pero mi pensamiento ha estado con ellos en todo momento.

A mi mamá, que es mi mayor inspiración, por estar siempre cuando la necesito y porque para mi es la mujer más linda del mundo.

A mi papá, por ser mi consejero, y porque a él le debo todo lo que soy hasta hoy.

En especial a mi hermano porque es el mejor amigo que tengo

Resumen

La inteligencia artificial (AI) tiene todo un potencial por descubrir que podría revolucionar los juegos y simuladores de cualquier género. Dentro de estos softwares de simulación existen los llamados agentes libres que tienen la capacidad de moverse e interactuar con su entorno virtual gracias a una serie de algoritmos, como los de locomoción llamados "*Steering Behaviors*". Teniendo como objetivo esta investigación realizar una biblioteca que contenga los diferentes algoritmos de locomoción de forma tal que estén disponibles para usarlos en un proyecto de simulación.

En la investigación realizada como parte de este trabajo se abordan los conceptos necesarios y el funcionamiento de los algoritmos "*Steering Behaviors*". En este documento se muestra las facilidades que brindan estos algoritmos así como la funcionalidad de la biblioteca con este módulo.

Se expone, a continuación de la fase de investigación, el diseño e implementación de un conjunto de clases que permiten conformar el módulo. Como resultado de este proceso el módulo que se obtuvo cuenta con las características necesarias para su acople a la biblioteca. Con el uso del módulo desarrollado se facilita la programación de la parte inteligente de los juegos, reduciendo además el tiempo de desarrollo de estas aplicaciones.

Palabras claves: *Steering Behaviors*, simulación, módulo, inteligencia artificial

Índice

Introducción

La Informática es una disciplina emergente-integradora que surge producto de la aplicación-interacción sinérgica de varias ciencias, como la computación, la electrónica, la cibernética, las telecomunicaciones, la matemática, la lógica, la lingüística, la ingeniería, la inteligencia artificial, la robótica, la biología, la psicología, las ciencias de la información, cognitivas, organizacionales, entre otras y contribuye al estudio y desarrollo de los productos, servicios, sistemas e infraestructuras de la nueva sociedad de la información.

En Cuba el desarrollo de la informática se ha incrementado en los últimos tiempos producto de la necesidad de lograr un crecimiento tecnológico, ya que esto es vital, sobre todo en una economía típica de un país del tercer mundo, el nuestro con pocos recursos naturales y sustentada básicamente en la agricultura. Una de las aristas de esta rama es precisamente la confección de videojuegos y simuladores que se encuentran enmarcados dentro del perfil de realidad virtual.

La elaboración de estos es de gran importancia ya que tienen la capacidad de simular entornos, procesos o situaciones de la vida real. Sus beneficios se dejan ver en la industria del entretenimiento (video-juegos, efectos especiales en el cine, etc.); en la educación (sistemas multimedia); en la visualización científica de datos y fenómenos no perceptibles al ojo humano; en la medicina (realización de prótesis, intervención médica a niveles celulares, ingeniería genética, tele operaciones, virtuoterapia, …); en la meteorología (estudio de tormentas eléctricas, impacto geológico de volcanes en erupción…); en el diseño de compuestos químicos, análisis molecular entre otras.

En la Universidad de la Ciencias Informáticas la facultad 5 se dedica, entre otras cosas, a realizar diferentes proyectos de simuladores y videojuegos. Un ejemplo es el grupo de trabajo que se dedica al desarrollo del Simulador de Conducción de Auto desde el año 2003 hasta la actualidad. En este tipo de software existen los llamados agentes libres (no son controlados por el usuario) que realizan los movimientos con una aparente inteligencia, es decir, son capaces de tomar decisiones a la hora de moverse.

Una vía efectiva para lograr esto, sería que los agentes se basen en un tipo de algoritmo denominado *Steering Behaviors* (comportamiento dirigido), que a su vez contienen una amplia gama de sub-algoritmos que son los que se encargan de llevar a cabo cada una de los tipos de decisiones o movimientos que requieren los agentes de acuerdo a la simulación que se esté desarrollando.

Resulta interesante saber que estos pueden representar diferentes tipos de movimientos (humanos, animales, criaturas alienígenas), vehículos (carros, aviones, naves espaciales) u otros tipos de agentes móviles. Pero su elaboración puede resultar un proceso muy complejo si se tienen que implementar en cada software por separado. Motivo por el cual es conveniente crear una biblioteca que contenga estos algoritmos de forma tal que cuando se vayan a emplear solo sea necesario ejecutar el que se requiera en ese caso.

Después de analizar la situación se identifica el siguiente **problema científico**:

¿Cómo aplicar los diferentes algoritmos de locomoción con múltiples *Steering Behaviors* en los simuladores y videojuegos de la facultad 5?

El **objeto de estudio** correspondiente al problema científico es:

Los softwares de simulación y videojuegos de la facultad 5.

Del objeto de estudio analizado, se define el **campo de acción** como: Los algoritmos de locomoción para agentes inteligentes.

Para lograr la solución del problema identificado se plantea el siguiente **objetivo general:** Realizar una biblioteca que contenga los diferentes algoritmos de locomoción de forma tal que estén disponibles para usarlos en un proyecto de simulación.

Producto de esto resaltan las siguientes **interrogantes científicas**:

- ¿Cuáles son los fundamentos teóricos sobre la implementación de algoritmos de locomoción (*Steering Behaviors*)?
- ¿Cuál es la forma más eficiente de implementar algoritmos de locomoción?
- ¿Cómo implementar los *Steering Behaviors* más usados y necesarios en los simuladores y video juegos?
- ¿Cómo diseñar una Biblioteca que permita agrupar los algoritmos de locomoción de forma que el proceso de cálculo sea eficiente y extensible a nuevos algoritmos?

- ¿Cuál es la efectividad de la implementación de la biblioteca diseñada para agrupar los algoritmos de locomoción de forma que el proceso de cálculo sea eficiente y extensible a nuevos algoritmos?

Para solucionar esto se definen los siguientes **objetivos específicos**:

- Analizar las soluciones encontradas por otros investigadores en cuanto a la implementación de algoritmos de locomoción (*Steering Behaviors*).
- Definir la forma más eficiente de implementar algoritmos de locomoción.
- Desarrollar los diferentes *Steering Behaviors*.
- Diseñar una biblioteca que permita agrupar los algoritmos de locomoción de forma que el proceso de cálculo sea eficiente y extensible a nuevos algoritmos.
- Implementar la biblioteca de *Steering Behaviors*.

Para cumplir estos objetivos se realizarán las siguientes **tareas de investigación**:

- Estudiar libro: AI programming by example y las Bibliotecas **MetaAgent** y **OpenSteer** para C++.
- Implementar los *Steering Behaviors* más usados y necesarios en los simuladores y video juegos.
- Utilizar el *Rational Rose* para realizar el diseño de la biblioteca.
- Implementar la biblioteca.

La investigación está estructurada en cuatro capítulos, primero se exponen los elementos que describen la fundamentación teórica de la misma a través de distintos sub-epígrafes, con el objetivo de tocar elementos como: ¿Qué es la Inteligencia Artificial?, Precedente Histórico sobre los "*Steering Behaviors*", algunos de los conceptos necesarios para un mejor entendimiento de este trabajo, las

bibliotecas que han sido creadas anteriormente, se mencionaran algunos de los "*Steering Behaviors*" que existen y su funcionamiento.

El capítulo 2 "Solución Técnica" expone las características técnicas que presentará el sistema como solución a los problemas planteados, se crea el modelo de Dominio, el Glosario de los términos del Dominio, se hace la captura de requisitos y se crean los modelos de casos de uso del sistema. Además se hará un análisis sobre las metodologías y lenguajes de desarrollo y las características que hicieron que sean escogidos para elaborar este trabajo.

Por su parte el capítulo 3 "diseño e Implementación" presentará ya el diseño del sistema propuesto, para ello se realizará el Diagrama de Clases de Diseño, la descripción de las clases, el Diagrama de Secuencia de los principales Casos de Uso, además contará con el Diagrama de los componentes del módulo a desarrollar. En el capitulo 4 se presentan los resultados de un conjunto de pruebas que se le realizan al software con el objetivo de comprobar el funcionamiento y medir el rendimiento y complejidad del mismo.

Para finalizar se brinda las conclusiones del trabajo.

Los métodos teóricos:

Para darle cumplimiento a las tareas la investigación se guía en los marcos de los métodos científicos teóricos y empíricos de investigación.

Dentro de los métodos teóricos se utilizan:

- **Analítico-sintético e inductivo-deductivo**, con el que se analizan cada uno de los elementos de manera independiente, posibilitando una mayor capacidad de comprensión y de síntesis sobre los aspectos más importantes.
- **Análisis histórico-lógico**, brindará la posibilidad de analizar toda la evolución del problema que se estará estudiando.
- **Enfoque de sistema** para la organización y aplicación de la propuesta.

Dentro de los métodos empíricos se usan:

- **Observación** que permite adquirir información necesaria en cualquiera de las fases de la investigación, el acercamiento a la realidad y la determinación de la posible solución del problema desde diferentes ángulos.
- **Análisis de documentos** para la verificación de existencia de otros productos informáticos similares.
- **Análisis de los productos de la actividad** para la evaluación periódica de la locomoción de las entidades en la propuesta.

1

Capítulo 1 Fundamentación Teórica

Introducción

En este capítulo se presenta una reseña histórica del surgimiento y evolución de los algoritmos de locomoción, junto con algunas de las aplicaciones generales de los mismos, fundamentalmente en la construcción de aplicaciones de realidad virtual. También se explica el funcionamiento de los mismos, arribando a su definición.

1.1 Inteligencia artificial

Se denomina Inteligencia Artificial (AI) a la rama de la informática que desarrolla procesos que imitan a la inteligencia de los seres vivos. La principal aplicación de esta ciencia es la creación de máquinas para la automatización de tareas que requieran un comportamiento inteligente.
Algunos ejemplos se encuentran en el área de control de sistemas, planificación automática, la habilidad de responder a diagnósticos y a consultas de los consumidores, reconocimiento de escritura, reconocimiento del habla y reconocimiento de patrones. Los sistemas de AI actualmente son parte de la rutina en campos como economía, medicina, ingeniería y la milicia, y se ha usado en gran variedad de aplicaciones de software, juegos de estrategia como ajedrez de computador y otros videojuegos.

La inteligencia artificial se divide en dos escuelas de pensamiento:

- La inteligencia artificial débil
- La inteligencia fuerte

1.1.1 Inteligencia artificial débil

Soluciona problemas específicos por lo que se aplica a procesos y aplicaciones tecnológicas, juegos, etc.

1.1.2 Inteligencia artificial fuerte

Se encarga de tratar de crear sistemas que imiten los procesos de pensamiento humano.

1.2 Conceptos fundamentales

Agente Autónomo:
Un agente autónomo es una entidad dentro de un entorno que cambia su estado con el tiempo. Para ello puede poner en práctica algoritmos de locomoción que dado una estructura a su alrededor (obstáculo, otra entidad, etc.) le permiten calcular los movimientos a seguir. Estos Agentes constituyen personajes en los juegos que tienen la capacidad de improvisación a la hora de seleccionar sus acciones, se les suele llamar caracteres no jugadores.

***Steering Behaviors* (Comportamientos):**
Son reglas de movimiento a seguir por los agentes autónomos a la hora de tomar decisiones ante los diferentes estímulos que ejercen sobre ellos los demás componentes de su entorno.

Obstáculo:
Un obstáculo es algo que impide el paso a través de él. Los agentes pueden reconocer los obstáculos y ser influidos por ellos. Un ejemplo de obstáculo dentro de un juego puede ser un muro.

Entorno:
Escenario en el que se encuentran los agentes y obstáculos.

1.3 Precedentes históricos

Los antecedentes de este trabajo se remontan a los años 40 como lo describe en 1948 Norbert Wieners en su libro "*Cybernetics, or Control and communication in the Animal and the Machine*" (Wiener, 1948). El término "*Cybernetics*" (cibernética) proviene de la palabra griega que significa timonel. A fines de los años 40 el Neurofisiólogo Grey Walter construyó un robot autónomo que contenía varios de los *Steering Behaviors* que se describen en este trabajo, además de que fue la primera maquina en vías de desarrollo que se elaboró para representar comportamientos de vida.
A principios de los 80 Valentino Braitenberg extrapoló el prototipo de Walter con la intención de experimentar con una serie de vehículos de fantasía que ya poseían comportamientos más complejos (Braitenberg, 1984). Por su parte David Zeltzer comenzó aplicando técnicas y modelos de inteligencia artificial para las aplicaciones de animación. (Zeltzer, 1983)

Ya en 1987 Craig W. Reynolds creó un modelo animado de comportamientos de manadas de aves usando técnicas estrechamente relacionadas con las que aquí se presentan. (Reynolds, 1987)

A continuación se presenta una lista de trabajos realizados que se encuentra dividida en tres categorías fundamentales: los relacionados con la robótica, la inteligencia artificial y la vida artificial, aunque en algunos casos la distinción es arbitraria. Estos tienen en común que en general todos están orientados a la animación (juegos, realidad virtual o multimedia).

1.3.1 Trabajos relacionados con la robótica

Rodney Brooks populariza el, entonces radical, Noción de construcción de controladores reactivos para sistemas robóticos. (Brooks, 1985)
Aunque inicialmente inspirado por una investigación etológica (comportamiento animal), el trabajo de Ron Arkin se ha centrado en la aplicación de *Steering Behaviors* para robots móviles. (Arkin, 1987)

La labor de Zapata en los controladores de dirección para robots móviles rápidos se centró en estrategias para controlar la cantidad de movimiento y otros aspectos de la mecánica de locomoción rápida. (Zapata, 1992)

Maja Mataric ha trabajado arduamente en los robots colectivos (Mataric, 1993) y es válido señalar que un tema central en su labor son los *Steering Behaviors*.

1.3.2 Trabajos relacionados con Inteligencia Artificial

Ken Kahn creó un novedoso sistema para generar la animación de caracteres móviles. (Kahn, 1979)

David Zeltzer (Zeltzer, 1990) encaminó la animación basada en Inteligencia artificial, popularizando la idea de resumir “niveles de tarea” para la especificación de movimientos.

Gary Ridsdale creó caracteres capaces de improvisar complejos movimientos, moviéndose desde un punto A hasta otro B evitando obstáculos estáticos y otros actores. (Ridsdale, 1987)

Mónica Costa desarrolló la animación de agentes basados en *Steering Behaviors* (Costa, 1990) que permite desplazarse alrededor de una casa a la vez que evita obstáculos.

1.3.3 Trabajos relacionados con vida Inteligente (y otros campos)

En 1987 el Modelo de manadas de aves, rebaños, escuelas, y otros movimientos grupales (Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics), descompuso esos complejos comportamientos grupales en tres simples *Steering Behaviors* como niveles individuales.

En 1987 la tienda de trabajos de inteligencia artificial Mitchel Resnick presentó un trabajo implementado en LEGO LOGO. (Resnick, 1989)

Michael Travers demostró su AGAR *Animal Construction Kit* (Travers, 1989) (Véase además trabajos más reciente de estos autores (Resnick, 1993) y (Travers, 1994)).

Los *Steering Behaviors* fueron un elemento clave en la elaboración de El tanque virtual de peces, sistema multi-usuario de realidad virtual instalado en la computadora del Museo, y fue creado por los equipos de trabajo de MIT´s Media Lab y NearLife (Resnick, 1998).

En (Wilhelms, 1990)Jane Wilhelms y Robert Skinner investigaron sobre arquitecturas vehículos representados como caracteres.
Thalmann creó animación de comportamientos de caracteres que navegaban bajo corredores y entre obstáculos, simulados en 3D (Thalmann, 1990).

Michiel van de Panne creó controladores de tareas como parqueo paralelo de automóviles usando la búsqueda de espacios (Panne, 1990).

G. Keith Still modeló grandes conglomerados de personas usando un modelo de *Steering Behaviors* para cada uno individual (Still, 1994).

1.4 Bibliotecas de *Steering Behaviors*

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

OpenSteer es una biblioteca de C + + creada para ayudar a construir agentes autónomos de juegos de realidad virtual y simuladores. Además la biblioteca, OpenSteer ofrece una aplicación basada en OpenGL llamada OpenSteerDemo, que muestra como se manifiestan algunos *Steering Behaviors*.

OpenSteer proporciona un conjunto *Steering Behaviors*, definidos en términos de un agente móvil abstracto llamado "vehículo". Suministra además ejemplos de código, incluyendo la implementación

de un vehículo simple y ejemplos de la combinación de simples *Steering Behaviors* para producir comportamientos más complejos.

Las clases de OpenSteer han sido diseñadas con flexibilidad para ser integradas a la elaboración de nuevos juegos.

Además el demo permite al usuario ajustar de forma interactiva aspectos de la simulación, se puede detener, correr paso a paso, seleccionar el vehículo u otro carácter de interés, ajustar la cámara etc.
OpenSteer es distribuido como un software *open source*, es compatible con Linux, Mac OS X y Windows.
MetaAgent (Halleux, 2003) es una biblioteca de *Steering Behaviors* creada completamente en C++ por Jonathan de Halleux para construir agentes (caracteres) autónomos. Son explicados los funcionamientos de los agentes y las políticas de diseño de las clases, de esta forma está compuesta, como una librería donde usted puede crear sus propios Behaviors usando los *templates* elaborados previamente.

Según el autor, el objetivo de **MetaAgent** era “utilizar toda la potencia del lenguaje de desarrollo C++ y la programación genérica para crear *Steering Behaviors*”.

1.5 Tipos de *Steering Behaviors*

1.5.1 Simples

Seek (Seguir)

Este comportamiento es útil para dirigir un agente en la dirección correcta, esto lo logra ya que retorna una fuerza que lo impulsa hacia el objetivo. Para ello lo que hace es calcular la fuerza necesaria para llegar al objetivo. Figura 1.

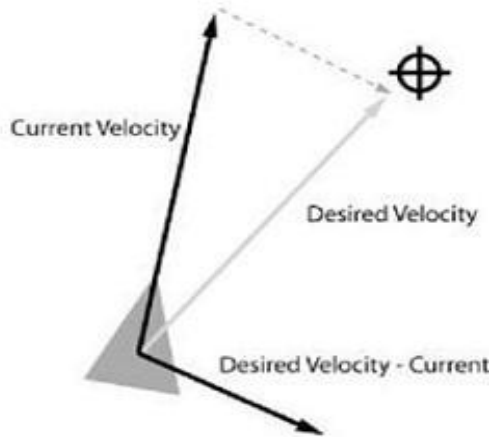

Figura 1: Representación de fuerzas del *Seek*

Flee (Huir)

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

Pursuit (Perseguir)

Se utiliza cuando un agente busca como interceptar un objetivo en movimiento. Por ejemplo: Un niño que juega en el patio del colegio a alcanzar a su compañero en movimiento, para ello debe mantenerse buscando la posición futura del curso de su objetivo y dirigirse en esa dirección. Este es el tipo de movimiento que describe el *Pursuit*.

El éxito de la función de búsqueda depende de lo bien que el demandante puede predecir la trayectoria del evasor. Esto puede ser muy complicado, por lo que debe hacerse una solución en la que se obtenga el rendimiento adecuado y sin usar demasiados ciclos de reloj. Figura 2.

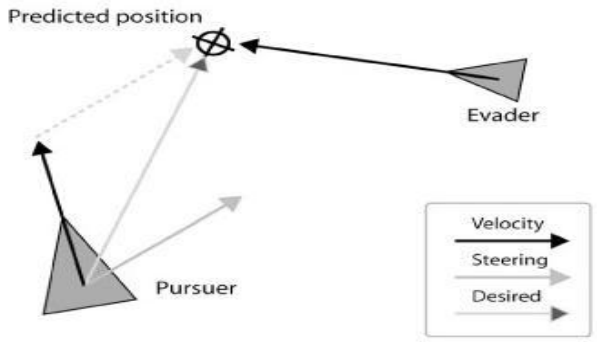

Figura 2: Representación del comportamiento *Pursuit*

Evade (Evadir)

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

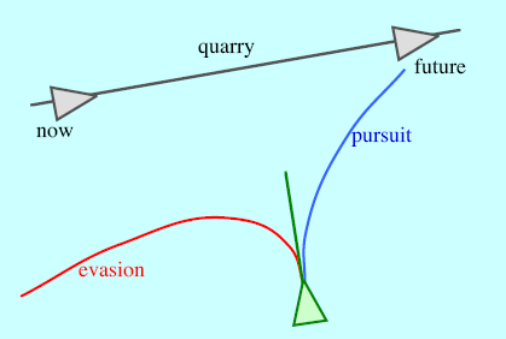

Figura 3: Representación del comportamiento *Pursuit* y *Evade*

Arrive (Arribar)

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

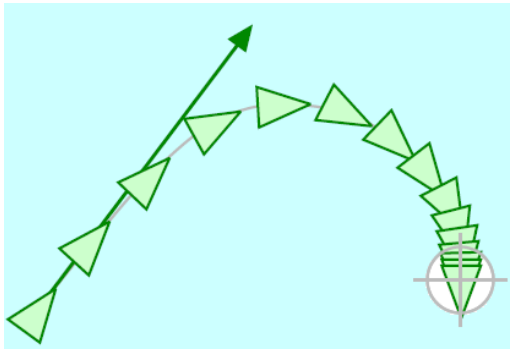

Figura 4: Representación del comportamiento *Arrive*

Wander (Aleatorio)

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

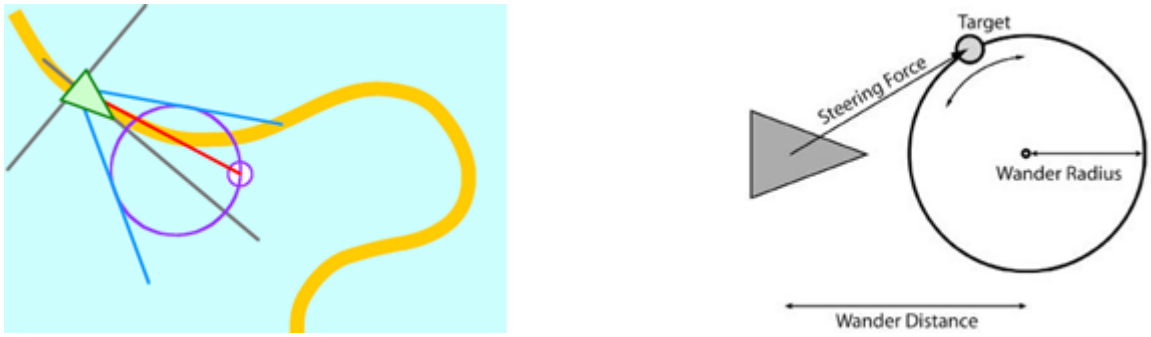

Figura 5: Representación del comportamiento *Wander*

Obstacle Avoidance (Evitar obstáculos)

Es un comportamiento que dirige un vehículo de modo que evite los obstáculos que yacen en su camino. Un obstáculo es cualquier objeto que pueda encontrarse en su camino (circulo si se trabaja en 2D). Esto se logra colocando un rectángulo (caja de detección) en la dirección del vehículo, que sea proporcional a la velocidad del mismo, en caso de que colisione con algún obstáculo cambia su dirección, manteniendo a este libre de colisiones. Figura 6.

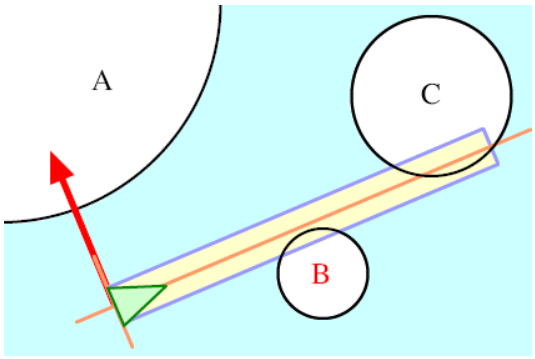

Figura 6: Representación del comportamiento *Obstacle Avoidance*

Wall Avoidance (Evitar muros)

Tiene como propósito evadir una pared (segmento de línea en 2D) que se encuentre en su camino, esto se logra proyectando tres rayos al frente del agente que son los que interceptan cualquier pared en su camino produciendo una fuerza normal en el punto donde se cruzan que repele al vehículo logrando que esta sea evitada. Su funcionamiento es similar al de los bigotes de un gato en la oscuridad. Figura 7.

Este proceso puede ser efectivo también colocando una caja de detección al frente del vehículo, de forma tal que al colisionar con el segmento de línea cambie su curso en la dirección contraria, este proceso es similar al del *Obstacle Avoidance*.

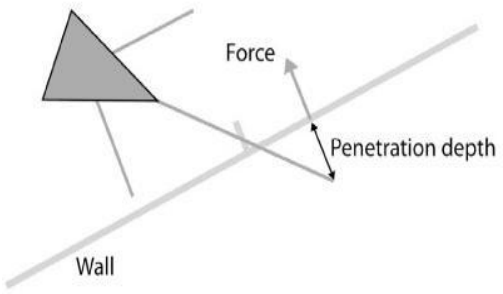

Figura 7: Representación del comportamiento *Wall Avoidance*

Interpose (Interponerse)

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

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

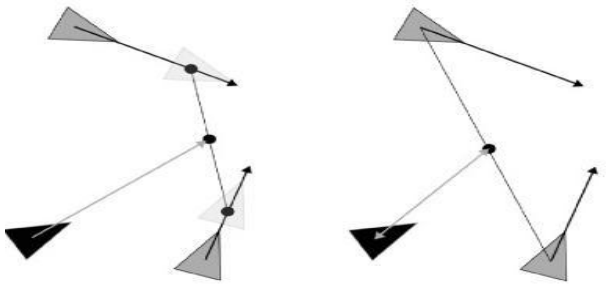

Figura 8: Representación del comportamiento *Interpose*

Hide (Ocultarse)

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

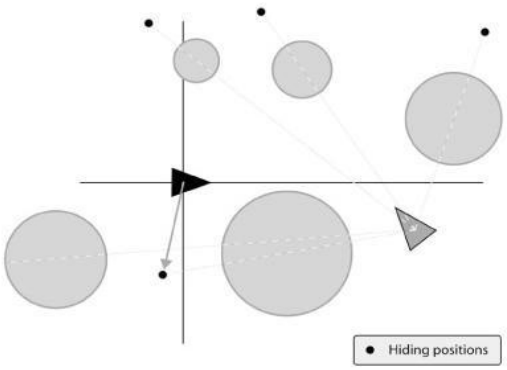

Figura 9: Representación del comportamiento *Hide*

PathFollowing (Seguimiento del camino)

Un agente con comportamiento *PathFollowing* contiene una lista de puntos que visitar, y crea una fuerza que lo dirige por el orden de la lista a cada uno de los puntos, los puntos visitados quedan descartados. Figura 10.

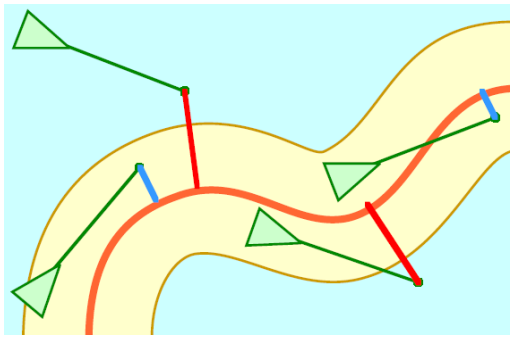

Figura 10: Representación del comportamiento *PathFollowing*

Offset Pursuit (Perseguir a distancia)

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

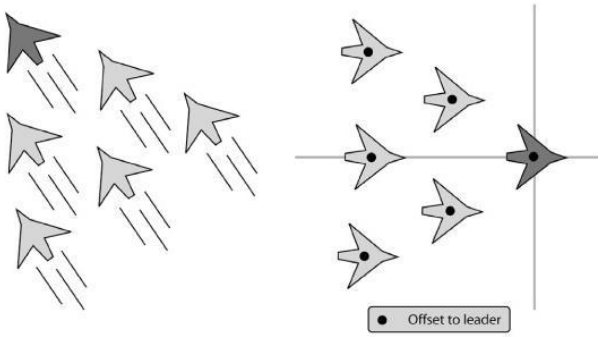

Figura 11: Representación del comportamiento Offset Pursuit

1.5.2 Combinados y Grupales

Flocking (Rebaño)

Es un comportamiento de grupos que combina tres comportamientos de grupo más simples: Cohesión, Separación, Alineación. Lo que hace es determinar la dirección de la fuerza para un grupo de comportamientos, un vehículo tomará en consideración todos los otros vehículos dentro de un área circular de tamaño predefinido - conocido como vecindario - centrada en el vehículo. En la figura: el blanco es el vehículo líder y el círculo gris muestra el tamaño de su vecindario. En consecuencia, todos los vehículos en negro se consideran sus vecinos y los vehículos mostrados en gris no lo son. Figura 12.

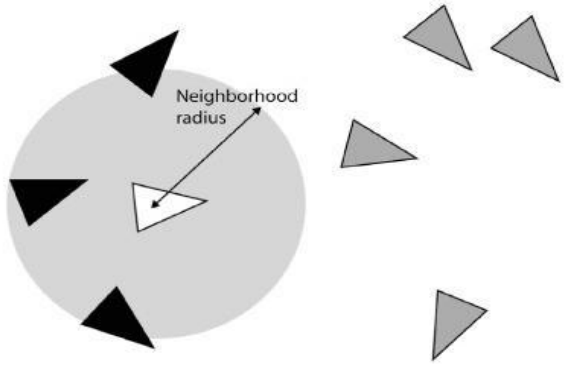

Figura 12: Representación del comportamiento Flocking

Cohesion (Cohesión)

Cohesión produce una fuerza de dirección que mueve un vehículo hacia el centro de masa de sus vecinos. Esta fuerza se utiliza para mantener un grupo de vehículos lo más unido posible. Figura 13.

Separation (Separación)

Produce una fuerza que dirige un vehículo lejos de sus vecinos. Cuando se aplica a un grupo de vehículos se esta tratando de maximizar la distancia entre ellos. Figura 13.

Alignment (Alineación)

Trata de mantener un vehículo en alineación (con la cabeza en la misma dirección) con sus vecinos. Figura 13.

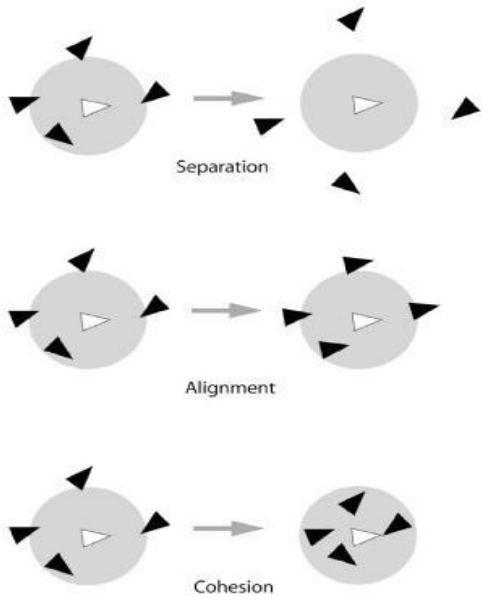

Figura 13: Representación de los comportamientos *Cohesion, Separation y Alignment*

Leader Following (Seguimiento de un líder)

Este algoritmo logra que un agente siga otro con un mayor valor de liderazgo. Que se encuentre en su vecindario. Si existen dos con mayor valor de liderazgo, el agente toma como líder al de mayor valor entre ellos. Una vez que ha elegido un líder, no cambia hasta que esté fuera de su vecindario. Figura 14.

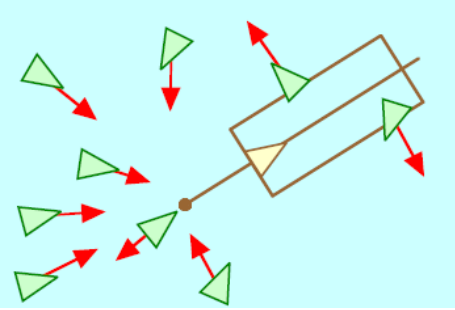

Figura 14: Representación del comportamiento *Leader Following*

Unaligned Collision Avoidance (Evadir colisiones no alineadas)

Este algoritmo se encarga de predecir posibles colisiones y alterar su dirección y velocidad para prevenirlas. Para ello tiene en cuenta cada uno de los otros agentes y determina (basado en la velocidad actual) cuando se realizará un acercamiento, y para evadirlo los vehículos se desplazaran lateralmente o variarán su velocidad. Este comportamiento se puede identificar en el desplazamiento de una persona por una plaza llena de gente y cada uno evita colisionar con los demás, lo que implica variaciones en la velocidad y dirección constantemente. Figura 15.

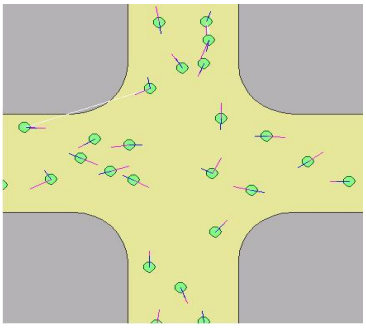

Figura 15: Representación del comportamiento *Unaligned Collision Avoidance*

Crow path following (Seguimiento de camino por manadas)

Este algoritmo simula un conjunto de vehículos siguiendo un camino determinado, para lograr esto utiliza la técnica de *path following* para seguir el camino descrito, pero además, incluye la de *separation* para evitar la superposición entre agentes. Figura 16.

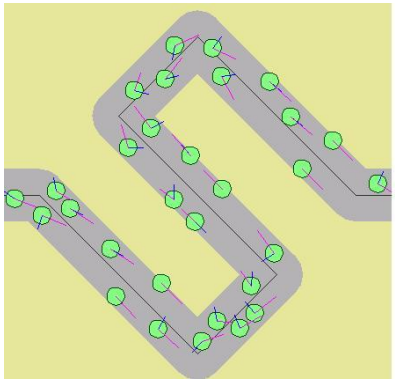

Figura 16: Representación del comportamiento *Crow path following*

Queuing (Salir por una puerta)

Con este algoritmo se trata de simular un conjunto de agentes en una sala que tratan de salir en masa por una pequeña puerta. Esto se logra frenando el vehículo si detecta otro que se encuentra: delante y con velocidad menor. Además son atraídos a la puerta utilizando el comportamiento *Seek*, además, para un mejor funcionamiento, se utiliza el *Separation* y el *Wall Avoidance*. Figura 17.

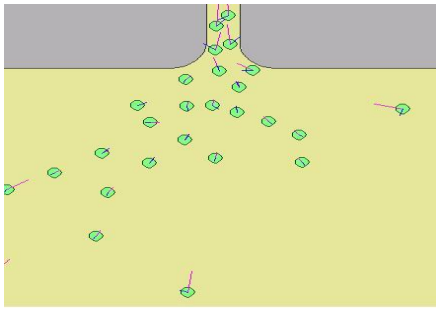

Figura 17: Representación del comportamiento *Queuing*

2

Capítulo 2 Propuesta de Solución

Introducción

En el presente capítulo se proponen soluciones específicas para lograr la integración de los comportamientos a la biblioteca, así como para lograr una manipulación y ejecución eficiente de los mismos. Además se comienza a tener una visión del sistema a realizar y se dan los primeros pasos en su concepción práctica, basándose en las necesidades y dificultades.

2.1 Descripción de la solución

Dada la necesidad de obtener una biblioteca que maneje los diferentes "*Steering Behaviors*" que se utilizan en softwares de simulación y que esta pueda ser de uso en diferentes plataformas. Muchas veces será muy útil o necesario trabajar los comportamientos de forma grupal, esto permitiría que una entidad pueda tomar decisiones teniendo en cuenta un mayor número de elementos en su entorno.
Para ello se creó un comportamiento encargado de controlar y manipular los demás comportamientos, cuenta con funcionalidades como: Evaluar los comportamientos de las entidades, además, permitirá elegir la forma adecuada de evaluación; adicionar comportamientos a una entidad y eliminar comportamientos. Estas funcionalidades permitirán un funcionamiento correcto de la biblioteca.
El lenguaje de programación que se empleará es el C++ debido a su alto nivel, el cual incorpora una variedad de características que facilitan una programación elegante y modular, además, el código escrito en C++ estándar permite mantener su portabilidad hacia otras plataformas.

2.2 Descripción de los Algoritmos de Locomoción

Cada algoritmo retornará una fuerza que será la encargada de dirigir la entidad hacia un objetivo determinado, este proceso se realiza gracias a la utilización de la clase Vector2D, que se encarga de analizar las coordenadas de un punto y devolver una fuerza hacia este. Además esta clase contiene los métodos necesarios para el trabajo con vectores.

Algoritmo Seek

```
Vector VelocidadDeseada = Normaliza(Objetivo – Posición_Entidad)*
MaxSpeed_Entidad.

Vector fuerza= VelocidadDeseada – Actual velocidad_Entidad

Retorna fuerza
```

Donde:

Normaliza: función que normaliza un vector especificado.

Objetivo: representa el vector de posición del objetivo.

Posición_Entidad: representa el vector de posición de la entidad.

MaxSpeed_Entidad: valor decimal de la máxima velocidad de la entidad.

Actual velocidad_Entidad: velocidad de la entidad.

VelocidadDeseada: vector que se forma desde la entidad hasta el objetivo.

Fuerza: es la fuerza que se ejerce sobre la entidad para que se dirija hacia el objetivo

Algoritmo Flee

```
Decimal DistanciaDePánico = 100.0 * 100.0
Decimal Distance
Distance= distancia entre vectores (Posición_Entidad , Objetivo)
```

```
SI ( Distance > DistanciaDePánico)
    Retorna (0,0)
SINO
 Vector VelocidadDeseada = Normaliza(Posición_Entidad – Objetivo)
  * MaxSpeed_Entidad.

Vector fuerza= VelocidadDeseada – Actual velocidad_Entidad

Retorna fuerza
```

Donde:

DistanciaDePánico: valor de la distancia de pánico.

MaxSpeed_Entidad: valor decimal de la máxima velocidad de la entidad.

Actual velocidad_Entidad: velocidad de la entidad.

VelocidadDeseada: vector que se forma desde la entidad hasta el objetivo.

Fuerza: es la fuerza que se ejerce sobre la entidad para que se dirija hacia el objetivo

Algoritmo Pursuit

```
Vector VelocidadDeseada = Normaliza(Posición_Objetivo –
Posición_Entidad)

VelocidadDeseada *= MaxSpeed_Entidad

     Vector fuerza= VelocidadDeseada – Actual velocidad_Entidad

Retorna fuerza
```

Donde:

Posición_Objetivo: representa la posición de la entidad objetivo

Normaliza: función que normaliza un vector especificado.

Posición_Entidad: representa el vector de posición de la entidad.

MaxSpeed_Entidad: valor decimal de la maxima velocidad de la entidad.

Actual velocidad_Entidad: velocidad de la entidad.

Algoritmo Hide

```
Decimal
SI  Distancia entre vectores(Posición_Entidad, Posición_Cazador)>50000
          Retorna Dirección_Entidad * MaxSpeed_Entidad
SINO
     decimal DistanciaMasCerca = MaxDouble
     Vector Mejor_Escondite

  Recorre la lista de obstáculos y para cada uno:
    Se calcula la posición de Escondite

decimal dist = distancia entre vectores (Escondite,Posición_Entidad)

     SI (dist < DistanciaMasCerca)
    ENTONCES
   DistanciaMasCerca = dist
   Mejor_Escondite = Escondite

   Se cumple la condición de parada

   Retorna el algoritmo Seek_Behavior para el Mejor_Escondite
```

Donde:

Posición_Cazador: posición de la entidad cazadora.

MaxDouble: valor máximo de un número decimal.

Escondite: posición detrás de un obstáculo opuesta a la posición de la entidad cazadora.

Algoritmo Arrive

```
 Vector ToTarget = Posición_Objetivo – Position_Entidad
 decimal distancia = longitud de ToTarget

SI (distancia > 0)
ENTONCES
 constante decimal regulador = 8000
 Decimal velocidad = distancia / entero Deceleracion * regulador
 velocidad= Valor Menor(velocidad, MaxSpeed_Entity)

 Vector VelocidadDeseada= ToTarget * velocidad/ distancia

 retorna (VelocidadDeseada - Velocidad_Entidad)
SINO
Retorna (0,0)
```

Donde:

ToTarget: vector que se forma entre la posición de la entidad y el objetivo.

Regulador: valor constante para aumentar la deceleración.

Deceleración: valor entero que se pasa por parámetros.

Valor menor: función que compara dos valores y toma el menor.

Algoritmo Alignment

```
    Vector Average_De_Dirección
Entero  Contador_Vecinos

   Se marcan las entidades del radio del vecindario como vecinas

   Recorre la lista de vecinos y  para cada uno:

    SI(está marcado)
    ENTONCES
     Average_De_Dirección += Dirección_Entidad
     Se incrementa Contador_Vecinos

    Se cumple la condición de parada

SI (Contador_Vecinos > 0)
     ENTONCES
     Average_De_Dirección /= Contador_Vecinos

  Retorna  Average_De_Dirección
```

Donde:

Contador_Vecinos: contador de la entidades marcadas como vecinas.

Dirección_Entidad: vector de dirección de la entidad vecina marcada.

Algoritmo Cohesion

```
     Se marcan las entidades del radio del vecindario como vecinas

     Vector CentroDeMasa
     Entero Contador_Vecinos
   Recorre la lista de vecinos y  para cada uno:
```

```
          SI(está marcado)
         ENTONCES
     CentroDeMasa += Posición_Entidad_Vecina
     Se incrementa Contador_Vecinos

    Se cumple la condición de parada

    SI (Contador_Vecinos > 0)
      ENTONCES
    CentroDeMasa /= Contador_Vecinos

  Retorna el algoritmo Seek_Behavior hacia el CentroDeMasa
```

Donde:

Posición_Entidad_Vecina: posición de la entidad marcada como vecina.

Algoritmo Separation

```
Se marcan las entidades del radio del vecindario como vecinas
Vector fuerza
Recorre la lista de vecinos y  para cada uno:

SI(está marcado)
ENTONCES
 Vector ToAgent = Posición_Entidad – Posición_Entidad_Vecina
      fuerza += Normaliza(ToAgent)/longitud de ToAgent
    Se cumple la condición de parada
  Retorna fuerza
```

Donde:

ToAgent: vector de una entidad a la entidad vecina.

Algoritmo Interpose

```
Vector PuntoMedio = (Posición_EntidadA + Posición_EntidadB) / 2.0
Decimal TiempoPara_PuntoMedio= Distancia entre vectores
(Posición_Entidad, PuntoMedio)/ MaxSpeed_Entidad

Vector APos = Position_EntidadA + Velocidad_EntidadA *
TiempoPara_PuntoMedio

Vector BPos = Position_EntidadB + Velocidad_EntidadB *
TiempoPara_PuntoMedio
     Vector Objetivo= (APos + BPos) / 2.0
Retorna el algoritmo Arrive_Behavior hacia el Objetivo
```

2.3 Metodologías y herramientas de desarrollo

Para la realización de la tesis se hizo un estudio de cada una de las posibles metodologías y herramientas a utilizar. A continuación se presenta la metodología que se utilizó y cada una de las herramientas con sus características distintivas que se tuvieron en cuenta para su selección.

2.3.1 Metodología

La metodología de desarrollo utilizada en la realización de esta Tesis es: RUP (Proceso Unificado de Desarrollo). RUP sirve de guía para realizar el análisis y diseño de la aplicación, debido a que es una metodología que ha probado su efectividad durante muchos años, ya que numerosos proyectos la han utilizado para desarrollar su software. Además, tiene un gran número de documentos publicados que

se pueden consultar para esclarecer dudas. También porque siguiendo sus pasos propuestos se obtiene una buena documentación de la tesis.

A continuación se muestran las características que más influyeron en la selección de esta metodología:

- Guiado por casos de uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean**,** constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo del sistema.

- Centrado en arquitectura: La arquitectura muestra la visión común del sistema completo. Permite además, implementar el Framework (plataforma sobre la que se implementa el soporte para todas las funcionalidades del sistema) y luego ir desarrollando cada uno de los módulos según se van necesitando.
- Iterativo e Incremental: RUP divide el proyecto en fases de desarrollo, propone además que cada una de ellas se desarrolle en iteraciones, las cuales aportan un incremento en el proceso de desarrollo y terminan con el cumplimiento del punto de control trazado en la fase.
- Utilización de un único lenguaje de modelado: UML.

2.3.2 Herramientas

Microsoft Visual Studio 2005

Posee numerosas herramientas asociadas que ayudan a escribir, analizar y distribuir el código, es un compilador rápido y con muy buena detección y corrección de errores. Posee facilidad de trabajo con los elementos visuales y buena integración de estos con el código. Contiene muchas librerías con códigos pre-escritos que ayudan en la escritura del código de la aplicación.

2.4 Lenguajes

En el desarrollo de la aplicación se utiliza C++ como lenguaje de programación, para la implementación de cada uno de los paquetes que conforman el diseño de la herramienta.
Para crear la documentación se utiliza UML como lenguaje de modelado, por las potencialidades descriptivas que posee.

2.4.1 Lenguaje de programación

Existen principalmente tres lenguajes que se utilizan para desarrollar aplicaciones gráficas en 3D: Lenguaje Ensamblador, C y C++, por ser los que con más velocidad ejecutan el código (menor costo de ejecución del programa). A estos se ha unido recientemente el Java como una opción para el desarrollo de este tipo de aplicaciones.
Es decisión de la entidad cliente, implementar este proyecto mediante el lenguaje C++, que ha estado en su línea de trabajo con magníficos resultados.
Es el lenguaje en el que se tiene mayor experiencia por parte de los desarrolladores del sistema que ocupa este proyecto.
Si se estudian las características de este lenguaje, se podrá apreciar lo acertado de la elección, dado que C++ es un lenguaje de programación de propósito general, especialmente indicado para la programación de sistemas por su flexibilidad y potencia. Es uno de los más utilizados por la comunidad de desarrollo de *software,* incluyendo la programación gráfica. (BIRN, 2000)
C++ es la evolución de C adaptada a la programación orientada a objetos. Tiene algunas cuestiones más pulidas como el control más estricto en el manejo de datos, y otras características que ayudan a la programación libre de errores. (BIRN, 2000)
En general puede llegar a ser un lenguaje tan rápido como C (el más rápido después del lenguaje ensamblador), sin embargo, si se maneja herencia múltiple, funciones virtuales y polimorfismo en forma inadecuada, o se accede mucho en niveles de profundidad en la llamadas a objetos (Objeto1, Objeto2, Objeto3…), puede llegar a hacerse más lento, lo cual no es conveniente para una aplicación en tiempo real. (BIRN, 2000)

2.4.2 Lenguaje de Modelado

Para modelar el análisis y el diseño del software se escogió el lenguaje UML (Unified Modeling Language, Lenguaje Unificado de Modelación). Esta decisión se debe a que se ha convertido en un estándar que tiene las siguientes características:

- Permite modelar sistemas utilizando técnicas orientadas a objetos (OO).
- Permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.

UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

2.5 Modelo del Dominio

El diagrama a continuación muestra el modelo de dominio de la biblioteca de "*Steering Behaviors*". Este modelo ayudará a los programadores en la comprensión y el análisis de la relación conceptual a diseñar.

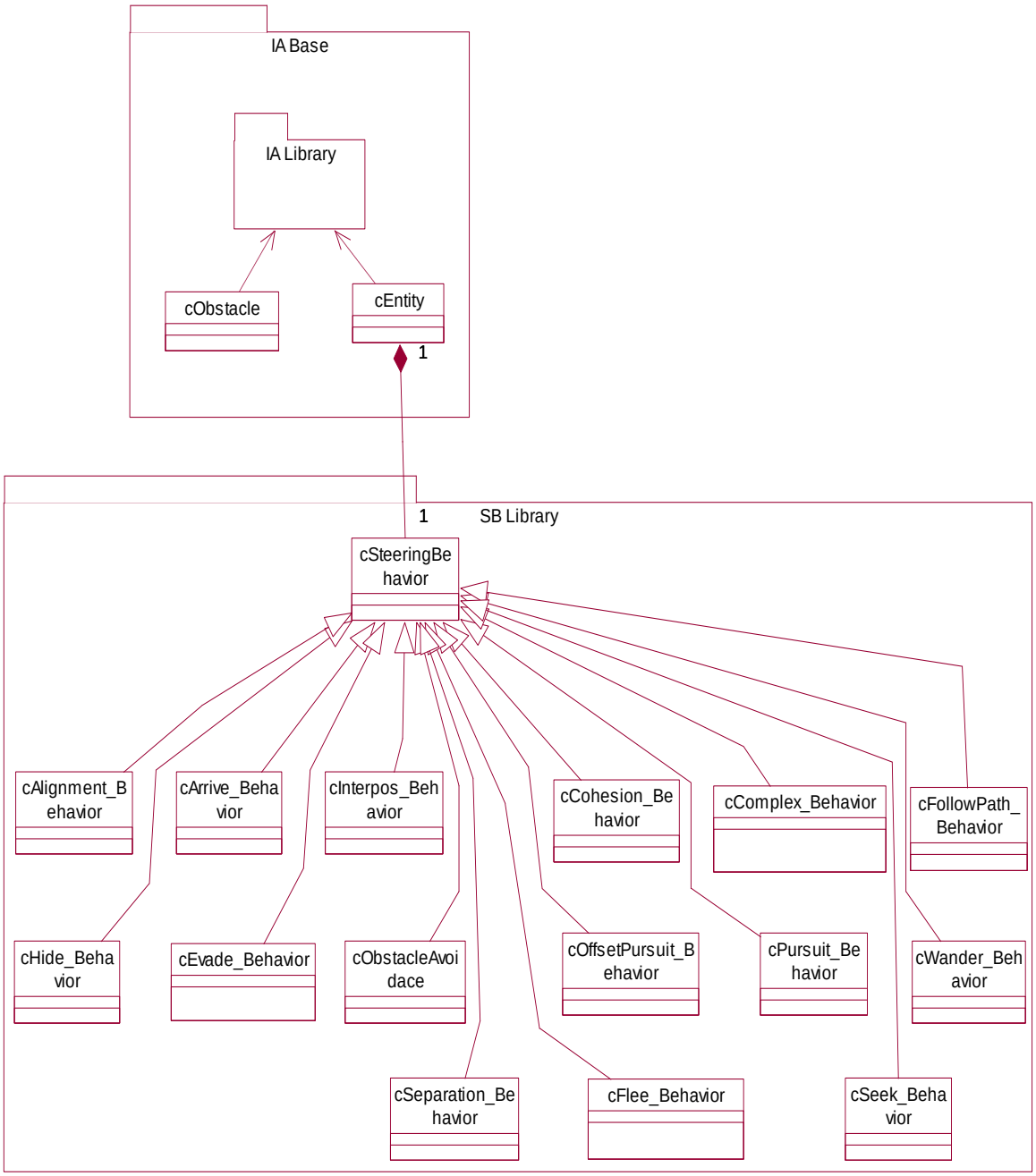

Figura 18: Modelo de dominio de la biblioteca de "*Steering Behaviors*"

2.5.1 Descripción de los términos del dominio

A continuación se presentan un conjunto de conceptos que conforma descripción de los términos que componen el modelo del dominio, con el propósito de facilitar un mayor entendimiento de los términos manejados en el diagrama anterior

IA Base: Paquete de clases de las clases al cual es agregada **SB_Library.**

SB Library: Paquete de clases que componen la Biblioteca de "*Steering Behaviors*"

cEntity: Objeto al cual se le van a aplicar los comportamientos de la biblioteca.

cObstacle: Clase que define un objeto de forma circular y sin movimiento en el entorno.

cAlignment_Behavior: Comportamiento que permite que las entidades se mantengan alineadas.

cArrive_Behavior: Comportamiento que le permite a las entidades disminuir su velocidad según llegue a su objetivo.

cInterpose_Behavior: Comportamiento que le permite a una entidad interponerse entre otras dos.

cCohesion_Behavior: Comportamiento que le permite a las entidades de una manada mantenerse cohesionadas.

cWander_Behavior: Comportamiento que le permite a las entidades describir un movimiento aleatorio

cFollowPath_Behavior: Comportamiento que le permite a una entidad seguir un camino de puntos predeterminados.

cHide_Behavior: Comportamiento que le permite a una entidad ocultarse de otra detrás de un obstáculo.

cEvade_Behavior: Comportamiento que le permite a una entidad huir de otra entidad

cObstacleAviodance: Comportamiento que le permite a las entidades evitar la colisión con obstáculos.

cOffsetPursuit_Behavior: Comportamiento que le permite a una entidad seguir a otra desde una posición a distancia, simulando una formación.

cSeek_Behavior: Comportamiento que le permite a una entidad seguir a otra entidad.

cFlee_Behavior: Comportamiento que le permite a una entidad huir de otra entidad calculando la posición futura de esta

cSeparation_Behavior: Comportamiento que le permite a entidades entidad seguir a otra entidad.

cComplex_Behavior: Permite combinar los comportamientos de la entidades, y además escoger la forma de combinarlos.

2.6 Captura de Requisitos

2.6.1 Requisitos Funcionales

Los siguientes requisitos establecen las funcionalidades e instrucciones que la biblioteca debe cumplir en su implementación.

R1. - Crear "*Steering Behaviors*".

- R1.1- Definir Comportamiento.
- R1.2- Asignarle el Peso.
- R1.3- Ejecutar Comportamiento

R2. - Asignarle comportamientos a entidades.

R3.- Eliminar de una entidad referencias directivas a los comportamientos.

R4.- Seguir una Entidad a otra Entidad.

R5.- Huir de otra Entidad

R6.- Una Entidad Persiga a otra.

R7.- Una Entidad evada a otra.

R8.- Arribar:

- R8.1- Una Entidad arribe en la posición de otra.
- R8.2- Una Entidad arribe en la posición de un punto

R9.- Una Entidad describa un movimiento aleatorio por el entorno virtual.

R10.- Las Entidades deben tener la capacidad de evadir los obstáculos

R11.- Interponerse (posicionarse en el punto medio):

- R11.1- Las Entidades deben tener la capacidad de interponerse entre otras dos entidades.
- R11.2- Las Entidades deben tener la capacidad de interponerse entre una entidad y un obstáculo.
- R11.3- Las Entidades deben tener la capacidad de interponerse entre dos obstáculos.

R12.- Las Entidades deben tener la capacidad ocultarse de otra entidad detrás de un obstáculo.
R13.- Seguir un camino de puntos.
R14.- Una entidad debe seguir a otra desde una distancia manteniendo una formación.
R15.- Una entidad debe tener la capacidad de alinearse con otra(s).
R16.- Una entidad debe tener la capacidad de conglomerarse con otra(s).
R17.- Una entidad debe tener la capacidad de no superponerse con otra(s).
R18.- El Software debe permitir acceder a los comportamientos de un agente.
R19.- El Software debe permitir seleccionar la forma de mezclar los comportamientos de agente.
R20.- Cada comportamiento debe ser lo más modular posible.
R21.- La biblioteca debe acoplarse al sistema de Plugins.

2.6.2 Requisitos no Funcionales

- **Usabilidad.**

Para los niveles apropiados de usabilidad, el código se encuentra fácil de entender para el programador, aunque se requiere de un personal calificado en el lenguaje de programación C++. Todo será implementado con terminologías del idioma inglés.

- **Rendimiento.**

La biblioteca debe tener una velocidad de Frame superior a 15FPS para 10 entidades, Haga uso óptimo de la memoria, mínimo de fugas de memoria, alinear el tamaño de la estructura de los Behaviors a la cache del sistema para un uso óptimo del bus del CPU.

- **Portabilidad.**

La aplicación se implementará en C++ estándar, esto hace más viable una futura migración hacia otras plataformas.

- **Requerimientos de Software.**

El sistema debe funcionar sobre sistemas operativos Windows 2000, XP o versiones superiores de Windows.

- **Requerimientos de Hardware.**

Memoria RAM 64 MB o superior, 128 Mb de disco como mínimo, procesador Intel P3 a 1.8 GHz

- **Restricciones en la implementación.**

El sistema está desarrollado en C++ con la ayuda del Visual Studio.NET, el cual se regirá por la filosofía de Programación Orientada a Objetos.

2.7 Definición de Casos de Uso

Para definir las funcionalidades que debe de cumplir la aplicación se elaboraron cuatro casos de usos: Insertar Behaviors, Eliminar Behaviors, Evaluar Behaviors, Configurar Behaviors. Cada uno de ellos ayuda a obtener una mejor comprensión del sistema a cualquier interesado en desarrollar la aplicación, independientemente del lenguaje que decida emplear. En la Descripción textual de cada uno de los Casos de Uso se representa además, las interacciones entre el sistema y el actor, lo que elimina cualquier duda posible en el funcionamiento de la aplicación.

2.8 Diagrama de Casos de Uso

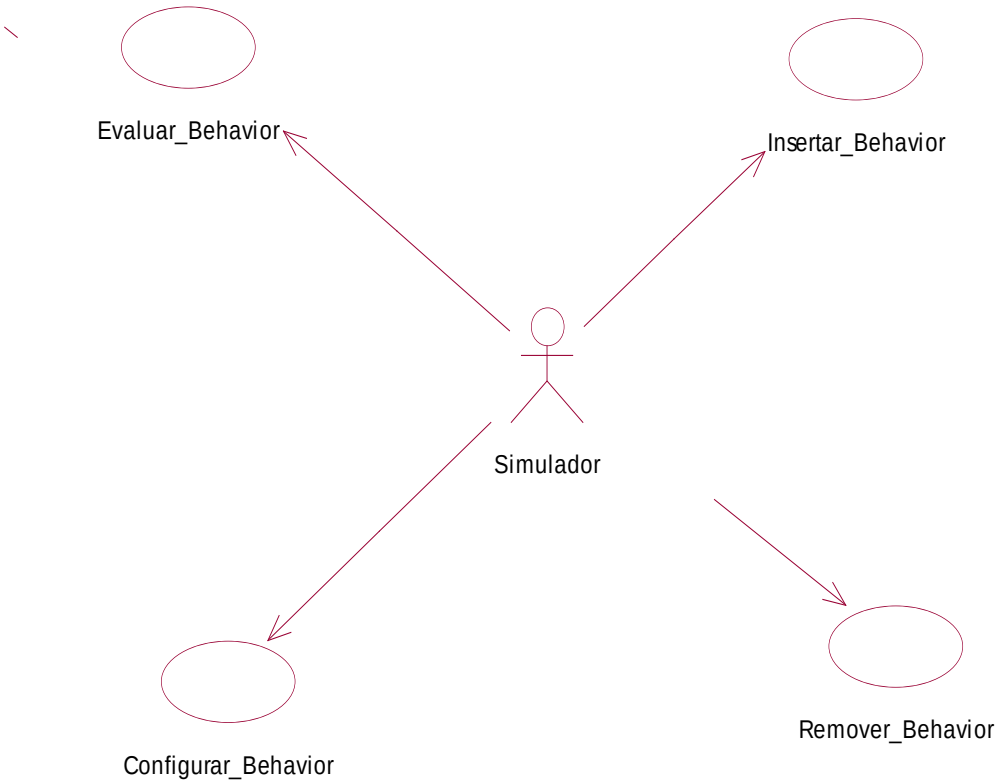

Figura 19: Diagrama de Casos de Uso

2.9 Definición actores del Sistema

Tabla 1: Actor del Sistema

Actores	**Justificación**
Simulador	Es la aplicación que hará uso de la biblioteca, aplicando a sus entidades los comportamientos deseados de la biblioteca implementada.

2.10 Descripciones de los casos de uso

Tabla 2: Caso de Uso Adicionar Behaviors a un Agente

CU -1	**Adicionar Behavior**
Actor	Simulador.
Descripción	Dada una entidad este Caso de Uso se encarga de adicionar un comportamiento de los existentes y con la prioridad deseada a la lista de esta entidad.
Referencias	R1

Tabla 3: Caso de Uso Eliminar Behaviors de un Agente

CU -2	**Eliminar Behavior**
Actor	Simulador.
Descripción	Busca un comportamiento que se pasa con antelación en la lista del comportamiento

	Complejo y si lo encuentra lo elimina automáticamente.
Referencias	R3

Tabla 4: Caso de Uso Evaluar Behaviors

CU -3	**Evaluar Behavior**
Actor	Simulador.
Descripción	Se encarga de calcular la fuerza resultante que se ejerce sobre la entidad, producto de cada uno de los comportamientos que hayan sido adicionados. Este cálculo se realiza teniendo en cuenta la forma de evaluación especificada en el caso de uso Configurar Behaviors.
Referencia	R1.3

Tabla 5: Caso de Uso Configurar Behaviors

CU -4	**Configurar Behavior**
Actor	*Simulador*.
Descripción	Consiste en asignarle los comportamientos que se deseen que realice una entidad y los respectivos valores de prioridad con que se desean ejecutar, así como la forma en que

	se combinarán las fuerzas resultantes de cada Behaviors.
Referencias	R1.1,R1.2,R2

2.11 Especificación de los casos de uso en formato expandido

Caso de Uso		
CU-1	Insertar Behaviors	
Propósito	Permitirle a una Entidad realizar un determinado "*Steering Behaviors*".	
Actores: Simulador.		
Resumen: El CU comienza cuando el programador solicita Adicionarle un "*Steering Behaviors*" a la lista de comportamientos que esta en la clase cComplex_Behavior. Termina cuando la Entidad tiene la capacidad de realizar dicho comportamiento.		
Referencias	R1	
Curso Normal de Eventos.		
Acción del Actor		**Respuesta del sistema.**
1._Se solicita adicionar el "*Steering Behaviors*", con la respectiva fuerza (prioridad) con que se desea que se ejecute, a la lista de la clase cComplex_Behavior.		2._ Adiciona el "*Steering Behavior*" a la lista de comportamientos
Precondiciones		• Que se encuentre implementado y creado el "*Steering Behaviors*".
Pos condiciones		• Queda guardado el comportamiento en la lista, con su respectiva prioridad de ejecución.

Prioridad	Crítico.

Tabla 6: CU Expandidos: Insertar Behaviors

Caso de Uso	
CU-2	Eliminar Behaviors
Propósito	Permitir eliminar un "*Steering Behaviors*" de la lista contenedora.
Actores: Simulador.	
Resumen: El CU comienza cuando el Simulador solicita eliminar un "*Steering Behaviors*". Termina cuando el comportamiento está eliminado, y la entidad no posee referencias directivas a este comportamiento.	
Referencias	R3
Curso Normal de Eventos.	
Acción del Actor	**Respuesta del sistema.**
1._Se solicita eliminar el "*Steering Behaviors*" pasándole el comportamiento especificado.	2._Verifica la existencia del comportamiento en la lista de la clase cComplex_Behavior. 3._Si lo encuentra, lo elimina de la lista de comportamientos.
Precondiciones	• Que el "*Steering Behaviors*" especificado se encuentre en la lista. • Que este implementado el "*Steering Behaviors*
Pos condiciones	• Queda eliminado el comportamiento de la lista de la entidad.

Curso alternativo	
	3._Si no lo encuentra Retorna un Warning
Prioridad	Crítico.

Tabla 6: CU Expandidos: Eliminar Behaviors

Caso de Uso	
CU-3	Evaluar Behaviors
Propósito	Permitir Evaluar los comportamientos que se encuentran en la lista contenedora.
Actores: Simulador.	
Resumen: El CU comienza cuando el Simulador solicita evaluar los comportamientos que han sido adicionados a la lista, brindando la posibilidad de seleccionar la forma de evaluación, las cuales son Average, Suma Truncada, Priorización.	
Referencias	R1.3

Curso Normal de Eventos.	
Acción del Actor	**Respuesta del sistema.**
1. Se solicita Evaluar los "*Steering Behaviors*" adicionados pasándole el comportamiento especificado y la prioridad con que quiere que se ejecute y la vía mediante la cual se requiere que sean evaluados los comportamientos.	2. Identifica la vía mediante la cual se requiere evaluar los comportamientos.
	3. Le imprime a la entidad la fuerza que ejercen sobre ella cada uno de los comportamientos que se encuentran en su lista.

Precondiciones	• Que el "*Steering Behaviors*" especificado se encuentre en la lista de la Entidad.
Pos condiciones	• La entidad se encontrara bajo la influencia de cada comportamiento evaluado por el método especificado.
Prioridad	Crítico.

Tabla 7: CU Expandidos Evaluar Behaviors

Caso de Uso	
CU-4	Configurar Behaviors
Propósito	Permitir Configurar un "*Steering Behaviors*" de una Entidad.
Actores: Simulador.	
Resumen: El caso de uso comienza cuando el Simulador detecta irregularidades en comportamiento de una entidad y desea configurarlo asignándole otro tipo de comportamiento y la prioridad con que se desea que se ejecute.	
Referencias	R1.1,R1.2,R2
Curso Normal de Eventos.	
Acción del Actor	**Respuesta del sistema.**
1. El Simulador cambia los valores de prioridad de un comportamiento y elige la vía de evaluarlo.	2. El sistema selecciona la vía de evaluación de los comportamientos.
	3. Reconoce los datos introducidos por el Simulador.

	4. Imprime a la entidad la fuerza que ejercen sobre ella los comportamientos con los nuevos datos.
Precondiciones	• Que la entidad no tome las decisiones correctas.
Pos condiciones	• Comportamiento de las entidades de la forma correcta.
Prioridad	Crítico.

Tabla 8: CU Expandidos Configurar Behaviors

3

Capítulo 3 Diseño e Implementación

Introducción

En el presente capítulo se presenta el diseño del sistema propuesto, con un diagrama de clases, donde se definen las responsabilidades de estas y sus relaciones. Además se presentan otros artefactos involucrados en el diseño como los diagramas de secuencia por casos de uso, facilitando con esto el entendimiento de la biblioteca.
En esta etapa del proyecto representa el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .cpp correspondiente a la implementación en C++.

3.1 Diagrama de clases de diseño del módulo de algoritmos de locomoción

Un **diagrama de clases** es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

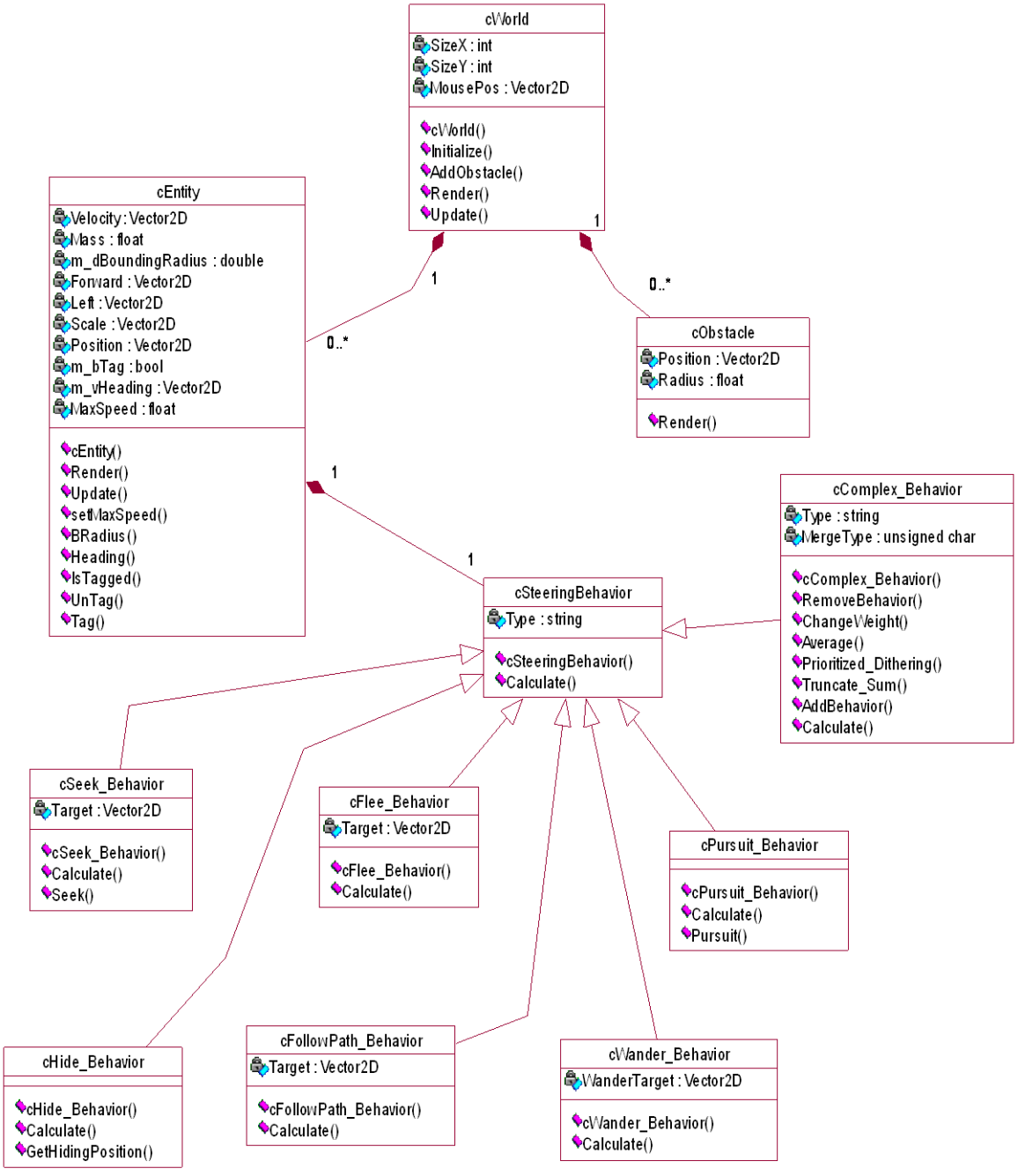

Figura 20: Diagrama de clases del paquete módulo de algoritmo genético

3.2 Descripción de las clases de diseño del módulo de algoritmos de locomoción

Tabla 9: Descripción de la clase cEntity

Nombre: cEntity	
Tipo de clase: Entidad	
Atributo	**Tipo**
Velocity	Vector2D
Mass	float
m_dBoundingRadius	double
Forward	Vector2D
Left	Vector2D
Position	Vector2D
m_bTag	bool
m_vHeading	Vector2D
MaxSpeed	float
Para cada responsabilidad	
Nombre	**cEntity()**
Descripción	Constructor de la clase.
Nombre	**Render()**
Descripción	Inicializa el estado del contexto de visualización con sus funciones.
Nombre	**Update(** int ElapsedTime **)**
Descripción	Actualiza la escena en el momento en curso.
Nombre	**setMaxSpeed(**float aSpeed**)**
Descripción	Cambia la velocidad máxima de la entidad.
Nombre	**BRadius()**
Descripción	Establece el radio del vecindario de una entidad.

Nombre	**Heading()**
Descripción	Establece la dirección de la entidad.
Nombre	**IsTagged()**
Descripción	Indica si está marcada o no la entidad.
Nombre	**UnTag()**
Descripción	Indica que la entidad no está marcada
Nombre	**Tag()**
Descripción	Indica que la entidad está marcada.

Tabla 10: Descripción de la clase cObstacle

Nombre: cObstacle	
Tipo de clase: Entidad	
Atributo	**Tipo**
Position	Vector2D
Radius	float
Para cada responsabilidad	
Nombre	**cObstacle()**
Descripción	Constructor de la clase.
Nombre	**Render()**
Descripción	Ejecuta la forma en que se va a pintar el obstáculo

Tabla 11: Descripción de la clase cSteeringBehavior

Nombre: cSteeringBehavior	
Tipo de clase: Entidad	
Atributo	**Tipo**

Type	String
Para cada responsabilidad	
Nombre	**cSteeringBehavior(cEntity* entity)**
Descripción	Constructor de la clase.
Nombre	**Calculate(int ElapsedTime)**
Descripción	Método virtual que se redefine en los demás comportamientos

Tabla 12: Descripción de la clase cComplex_Behavior

Nombre: cComplex_Behavior	
Tipo de clase: Controladora	
Atributo	**Tipo**
Type	String
MergeType	unsigned char
Para cada responsabilidad	
Nombre	**cComplex_Behavior(cEntity* entity)**
Descripción	Constructor de la clase.
Nombre	**Calculate(int ElapsedTime)**
Descripción	Método que escoge una vía para evaluar los comportamientos y devuelve el resultado de la fuerza.
Nombre	**AddBehavior(cSteeringBehavior* aBehavior, float aWeight)**
Descripción	Adiciona un comportamiento prioridad que le corresponde
Nombre	**RemoveBehavior(cSteeringBehavior* pBehavior)**
Descripción	Elimina un comportamiento específico de la lista
Nombre	**ChangeWeight(cSteeringBehavior* pBehavior, float newWeight)**
Descripción	Dado un comportamiento le cambia la prioridad con que se va a ejecutar

Nombre	**Average(int ElapsedTime)**
Descripción	Evalúa la acción que ejercen los comportamientos sobre una entidad mediante el método del promedio de sus fuerzas.
Nombre	**Prioritized_Dithering(int ElapsedTime)**
Descripción	Evalúa la acción que ejercen los comportamientos sobre una entidad escogiéndolos por la prioridad que les corresponde.
Nombre	**Truncate_Sum(int ElapsedTime)**
Descripción	Evalúa la acción que ejercen los comportamientos sobre una entidad, pero sólo toma en cuenta el siguiente comportamiento si la suma te la fuerza es menor que 1.

Tabla 13: Descripción de la clase cSeek_Behavior

Nombre: cSeek_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Target	Vector2D
Para cada responsabilidad	
Nombre	**cSeek_Behavior(cEntity* entity, Vector2D Target)**
Descripción	Constructor de la clase
Nombre	**Calculate(int ElapsedTime)**
Descripción	Método que retorna el valor del método ”Seek”
Nombre	**Seek(cEntity* pEntity, Vector2D& Target)**
Descripción	Método que retorna la fuerza de atracción hacia un objetivo.

Tabla 14: Descripción de la clase cFlee_Behavior

Nombre: cFlee_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Target	Vector2D
Para cada responsabilidad	
Nombre	**cFlee_Behavior(cEntity* entity, Vector2D Target)**
Descripción	Constructor de la clase.
Nombre	**Calculate(int ElapsedTime)**
Descripción	Retorna una fuerza de repulsión hacia un objetivo.

Tabla 15: Descripción de la clase cHide_Behavior

Nombre: cHide_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Para cada responsabilidad	
Nombre	**cHide_Behavior(cEntity* entity, cEntity* target)**
Descripción	Constructor de la clase
Nombre	**Calculate(int ElapsedTime)**
Descripción	Método que retorna el valor del método ”Hide”
Nombre	**Hide(cEntity* pEntity, cEntity* pHunter)**
Descripción	Método que retorna la fuerza que le permite a una entidad ocultarse de otra entidad cazadora detrás de un obstáculo.
Nombre	**GetHidingPosition(const Vector2D& posOb, const double**

	radiusOb, const Vector2D& posHunter)
Descripción	Dado la posición de una entidad cazadora, retorna la posición más factible para que una entidad se oculte detrás de un obstáculo.

Tabla 16: Descripción de la clase cPursuit_Behavior

Nombre: cPursuit_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Para cada responsabilidad	
Nombre	**cPursuit_Behavior(cEntity* entity, cEntity* target)**
Descripción	Constructor de la clase
Nombre	**Calculate(int ElapsedTime)**
Descripción	Método que retorna el valor del método ”Pursuit”
Nombre	**Pursuit(cEntity* pEntity, cEntity* pTarget)**
Descripción	Retorna la fuerza que le permite a una entidad perseguir a otra entidad.

Tabla 17 Descripción de la clase cPursuit_Behavior:

Tabla 17: Descripción de la clase cWander_Behavior

Nombre: cWander_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
WanderTarget	Vector2D
Para cada responsabilidad	
Nombre	**cWander_Behavior(cEntity* entity)**

Descripción	Constructor de la clase.
Nombre	**Calculate(int ElapsedTime)**
Descripción	Retorna una fuerza de atracción hacia un objetivo que le provoca a la entidad un movimiento aleatorio por la escena.

Tabla 18: Descripción de la clase cEvade_Behavior

Nombre: cEvade_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Para cada responsabilidad	
Nombre	**cEvade_Behavior(cEntity* entity, cEntity* pTarget)**
Descripción	Constructor de la clase.
Nombre	**Calculate(int ElapsedTime)**
Descripción	Retorna una fuerza de repulsión hacia la posición futura de una entidad cazadora.

Tabla 19: Descripción de la clase cAlignment_Behavior

Nombre: cAlignment_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Para cada responsabilidad	
Nombre	**cAlignment_Behavior(cEntity* entity, vector<cEntity*> neighbors)**
Descripción	Constructor de la clase.

Nombre	**Calculate(int ElapsedTime)**
Descripción	Retorna una fuerza que coloca la entidad en la dirección en que se encuentran las demás en su vecindario.

Tabla 20: Descripción de la clase cCohesion_Behavior

Nombre: cCohesion_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Para cada responsabilidad	
Nombre	**cCohesion_Behavior(cEntity* entity, vector<cEntity*> neighbors)**
Descripción	Constructor de la clase.
Nombre	**Calculate(int ElapsedTime)**
Descripción	Retorna una fuerza que dirige una entidad hacia el centro de masa de su vecindario.

Tabla 21: Descripción de la clase cSeparation_Behavior

Nombre: cSeparation_Behavior	
Tipo de clase: Entidad	
Atributo	**Tipo**
Type	String
Para cada responsabilidad	
Nombre	**cSeparation_Behavior(cEntity* entity, vector<cEntity*> neighbors)**
Descripción	Constructor de la clase.

Nombre	**Calculate(int ElapsedTime)**
Descripción	Retorna una fuerza que impide que las entidades se superpongan.

3.3 Diagramas de secuencia de los Casos de Uso

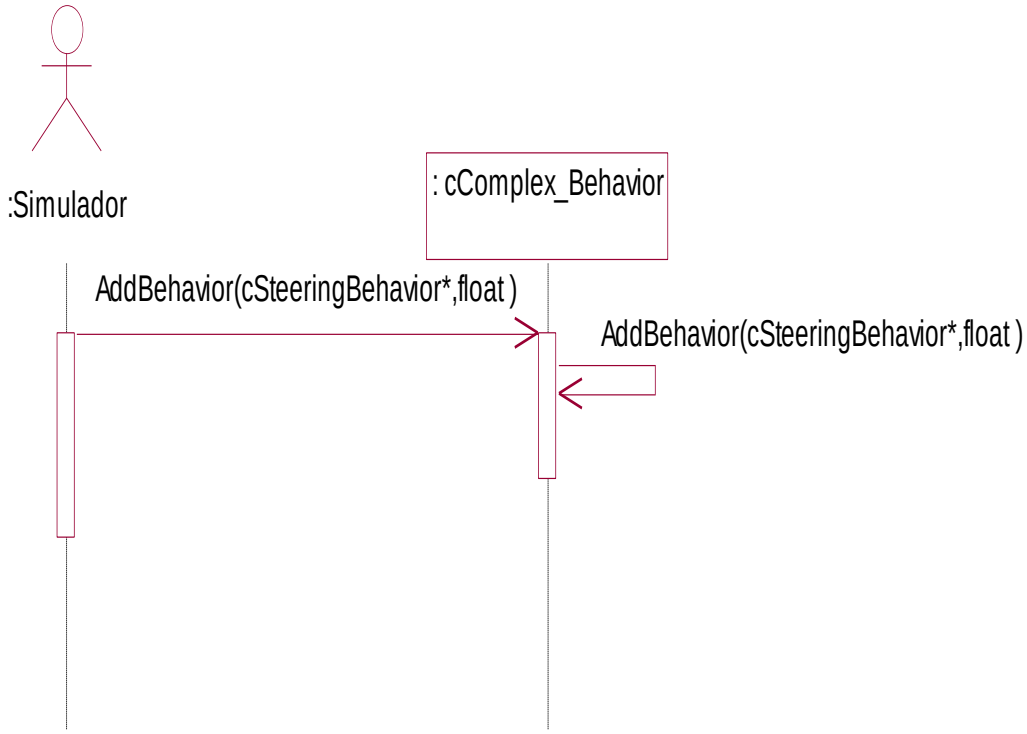

Figura 21: Diagrama de secuencia del Caso de Uso Adicionar Behaviors

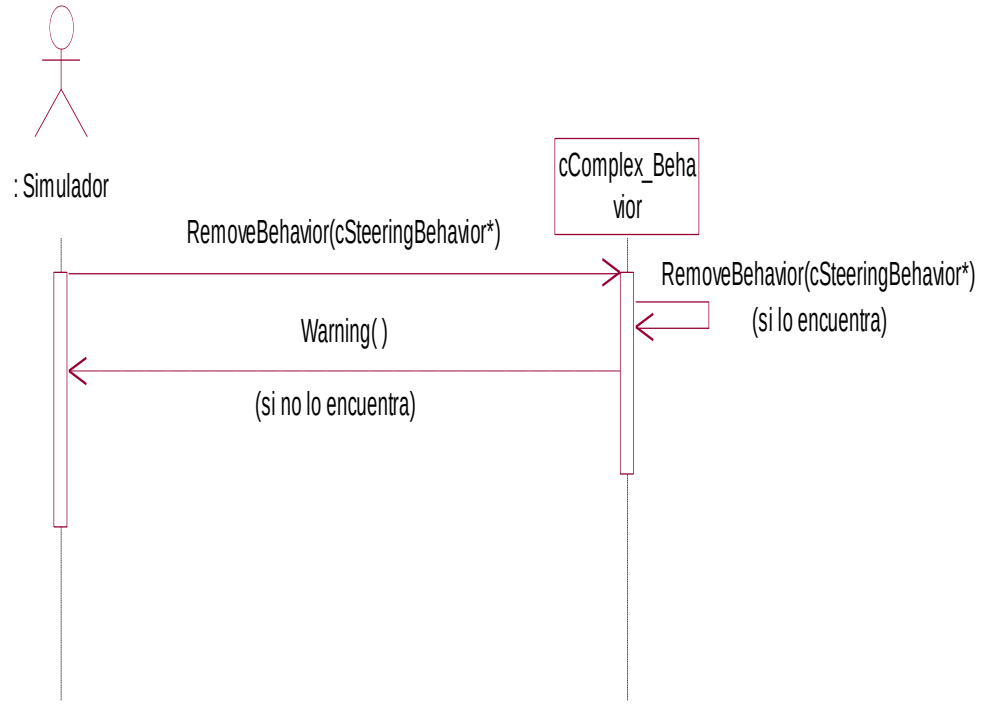

Figura 22: Diagrama de secuencia para el Caso de Uso Eliminar Behavior

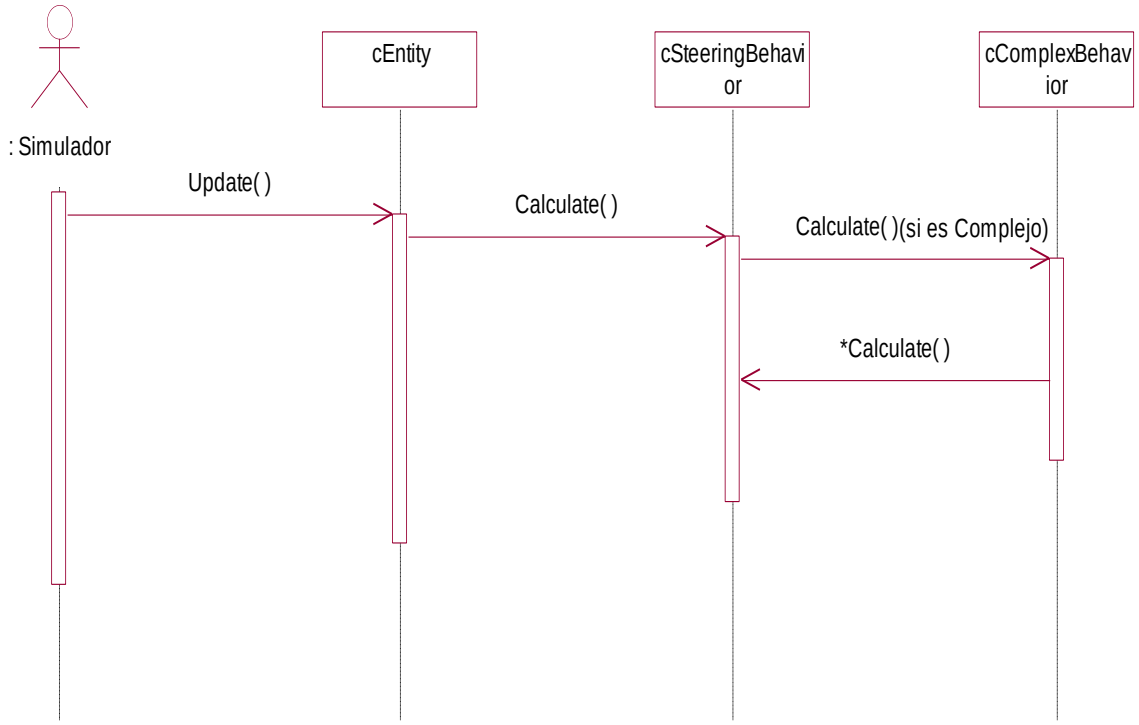

Figura 23: Diagrama de Secuencia del Caso de Uso Evaluar Behavior

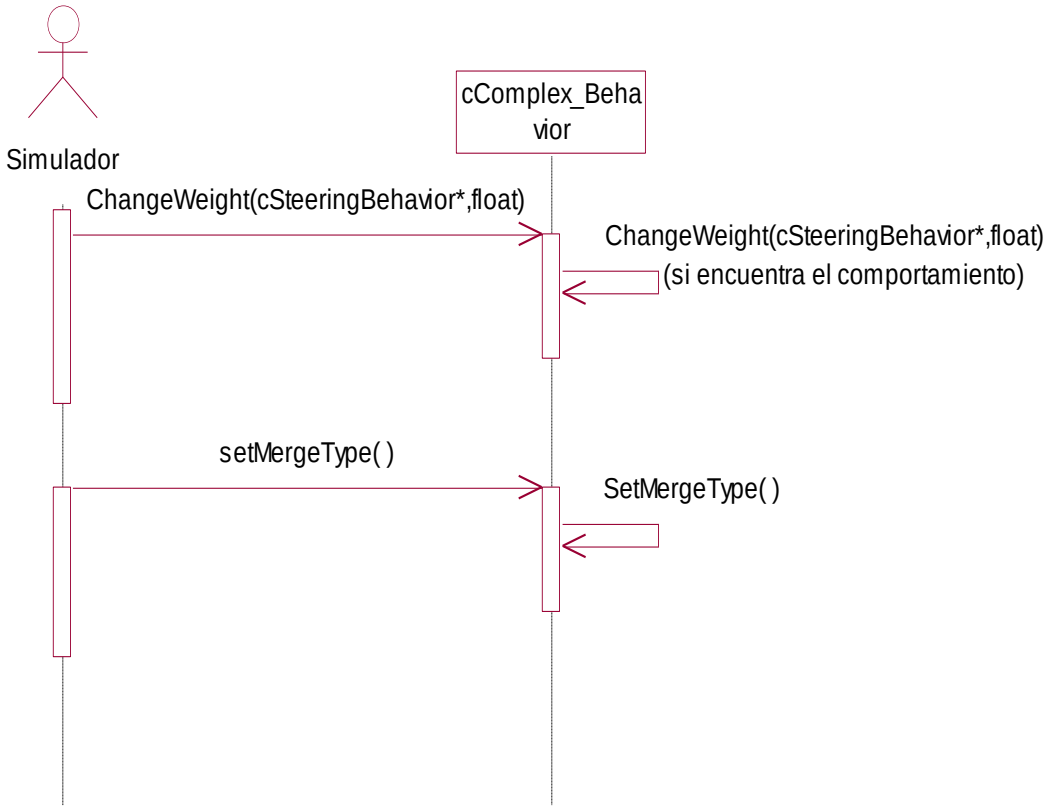

Figura 24: Diagrama de Secuencia del Caso de Uso Configurar Behavior.

3.4 Diagramas de componentes

Figura 25: Estructura del modelo de implementación

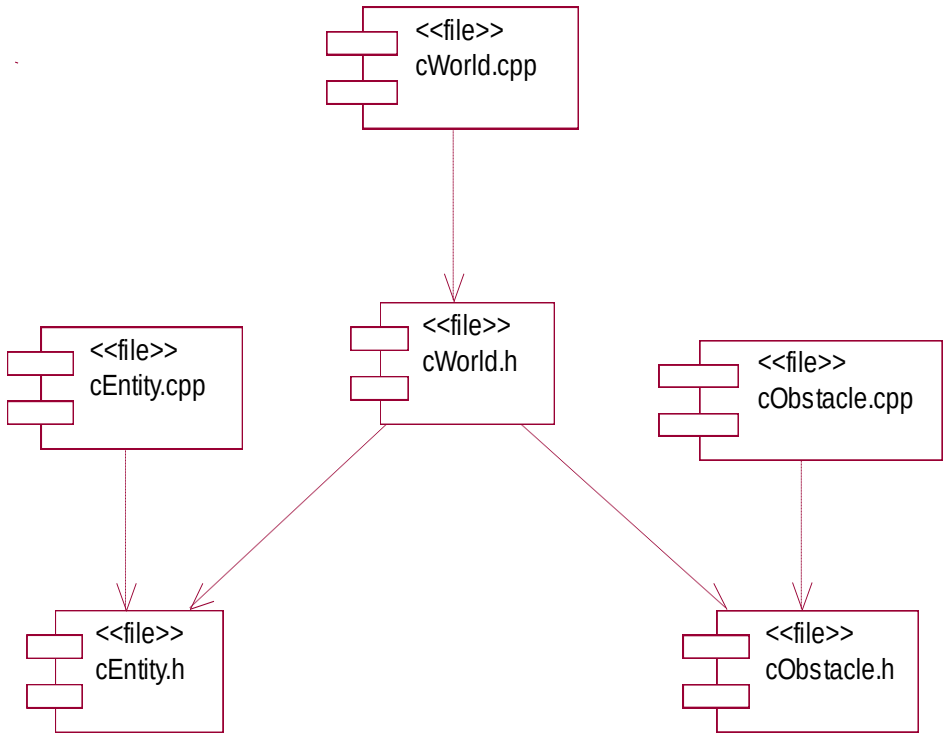

Figura 26: Componentes de IA Base

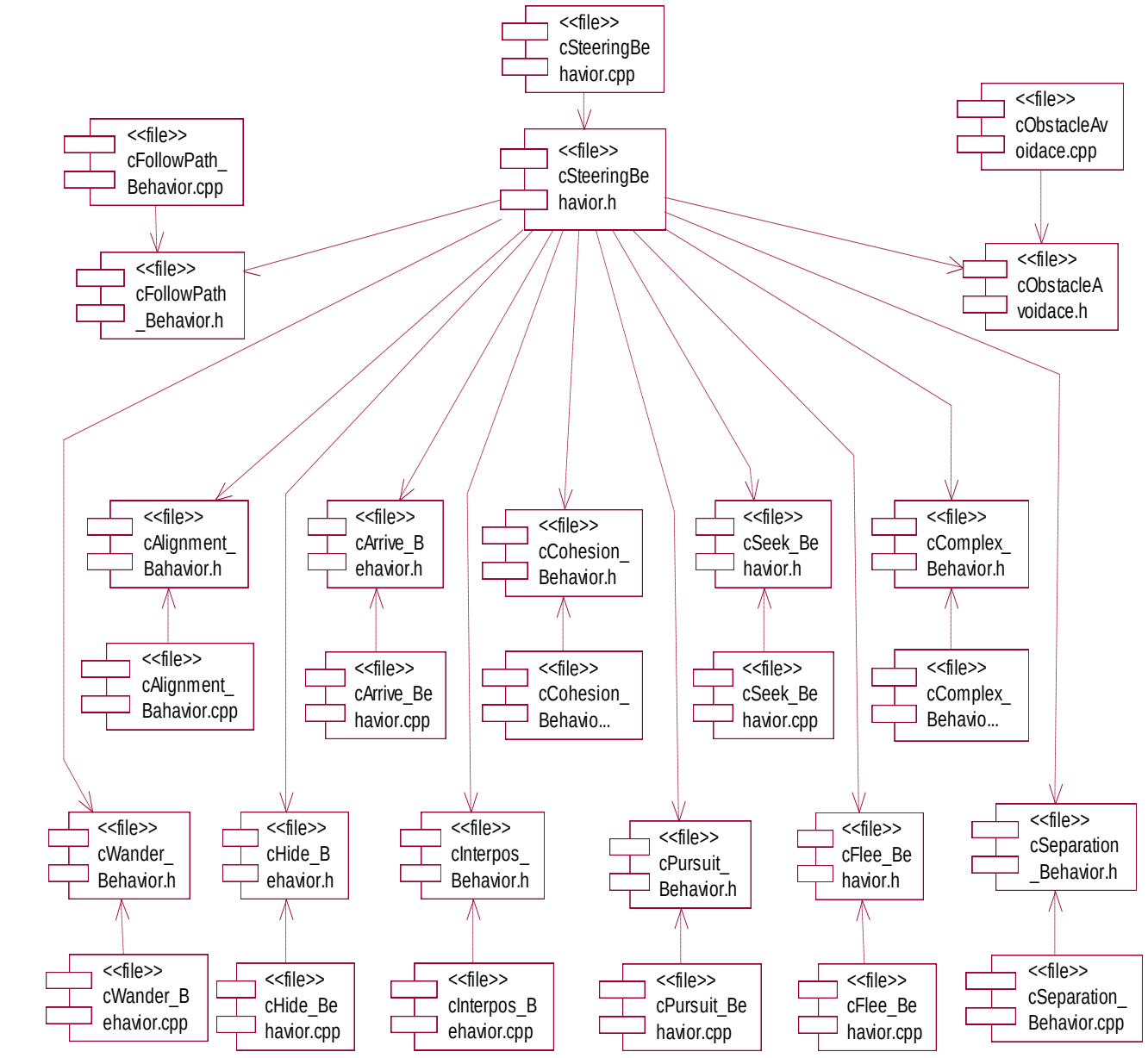

Figura 27: Componentes de SB Library

Conclusiones

En este momento se encuentra todo preparado para pasar a la etapa de programación de los casos de uso desarrollados en el segundo capítulo. Como posibilidad que brinda el *Rational Rose*, ya es posible generar el código fuente de los componentes relacionados con los casos de uso a desarrollar.

4

Capítulo 4 Pruebas al Módulo de Algoritmos de Locomoción

Introducción

En este capítulo se le hacen pruebas al módulo de algoritmo de locomoción. Haciendo uso de un ejemplo o demo para garantizar la funcionalidad del mismo. Se prueba el rendimiento así como también la complejidad del algoritmo.

4.1 Funcionamiento del Módulo

Demo: Para probar el funcionamiento del módulo de algoritmos de locomoción, se realizan ejemplos en los cuales las entidades (triángulos) son quienes se encargan de ejecutar cada uno de los comportamientos.

4.2 Rendimiento

Con esta prueba se busca medir el impacto que tiene el aumento del número de entidades y de comportamientos en el módulo implementado. Para que este mantenga un correcto funcionamiento debe mantenerse por encima de 15 frames por segundos (FPS).
Esto es fundamental, pues en los softwares que se aplicará seguramente se requiere simular un gran número de entidades, para simular una multitud de personas o una colmena de insectos. Por lo que el módulo debe mantener un funcionamiento correcto con un número grande de entidades.

4.2.1 Primera prueba de rendimiento

La primera prueba que se realiza (Ver tabla 22) muestra cómo se comportan los tiempos al aumentar el número de entidades. Lo que ha demostrado que si se aumenta el número de entidades el tiempo aumenta, aunque se necesita incrementarlo considerablemente para que este provoque efectos negativos en el software, por lo que se infiere que el Módulo tiene un buen rendimiento en este sentido.

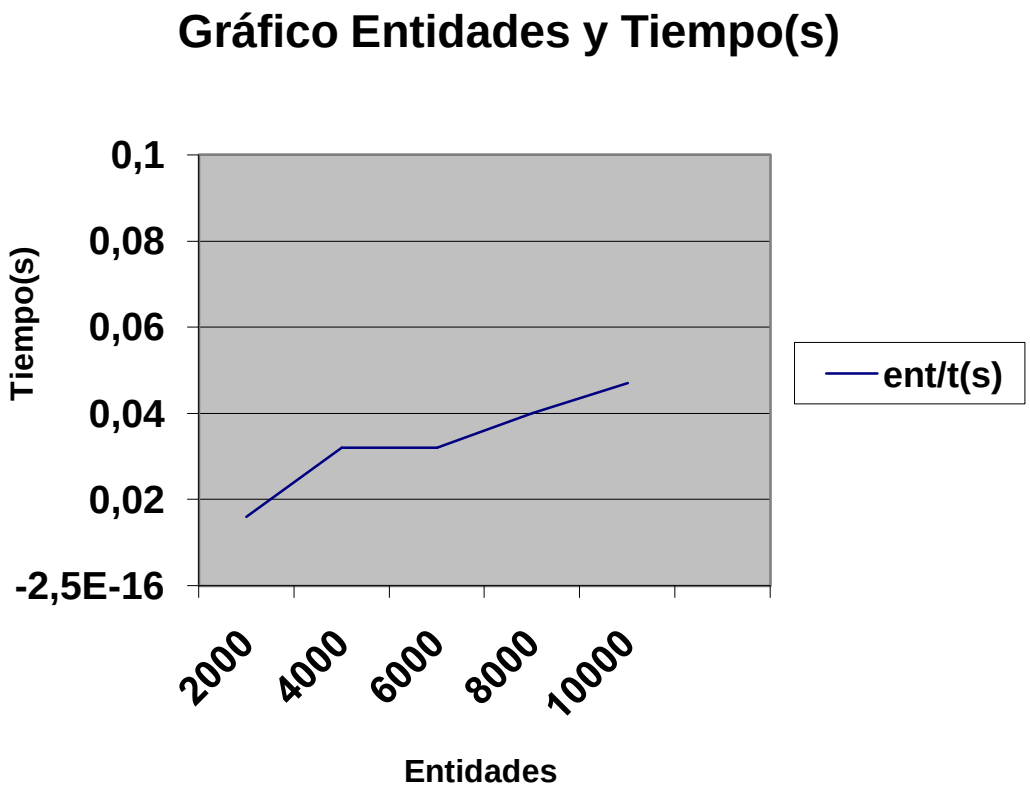

Tabla 22: Gráfico de Entidades y Tiempo.

4.2.2 Segunda prueba de rendimiento

Se realizó la segunda prueba (Ver Tabla 23) donde se mantuvo constante el número de entidades (10 entidades) pero se incrementó el número de comportamientos para ver el efecto que estos ejercen sobre el módulo en cuanto a los FPS.

Los resultados se demuestran que el tiempo es muy pequeño para un número grande de comportamientos, y sólo comienza a aumentar a partir de que se procesan 2000 comportamientos, pero no es un número que afecte el funcionamiento del módulo ya que entra dentro de los parámetros establecidos.

Constantes: número de entidades=10.

Comportamientos	**Tiempo(s)**
100	**0.016**
500	**0.016**
1000	**0.016**
2000	**0.032**
5000	**0.047**
10000	**0.065**

Tabla 23: Comportamientos y tiempo de ejecución

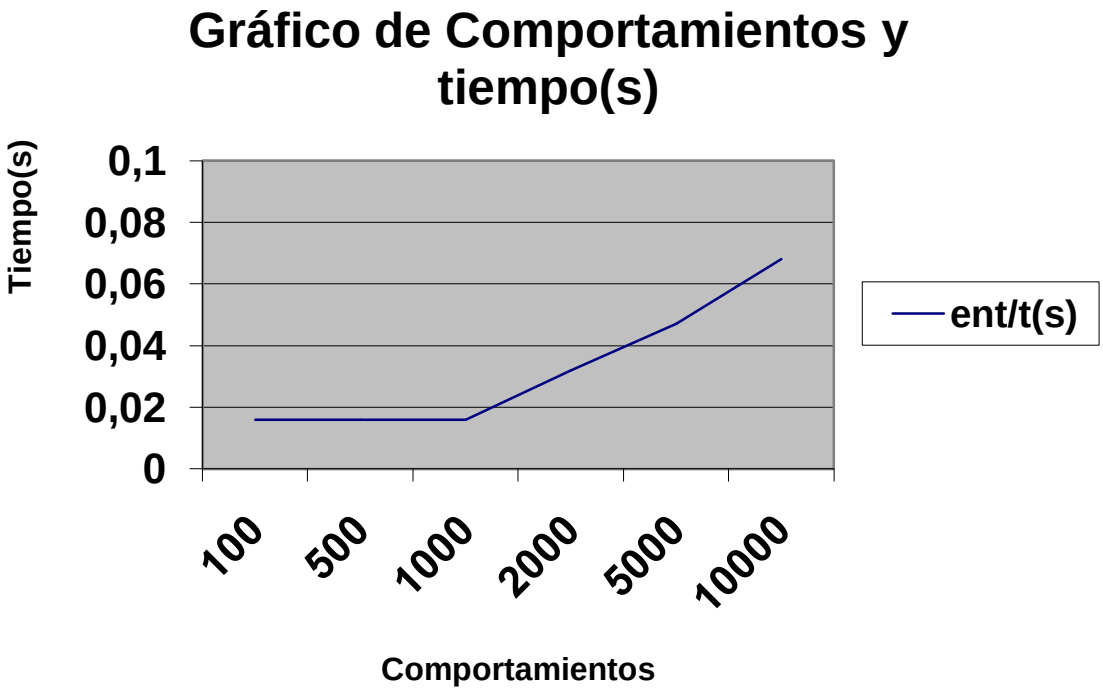

Tabla 24: Gráfico de comportamientos y Tiempo

4.3 Complejidad del Software

La complejidad del software en general va a ser igual a la complejidad del algoritmo que mayor complejidad tenga.

La complejidad de los algoritmos más sencillos como el **Seek** es de O (1).

En otros como el **Alignment, Separation** o el **Cohesion** la complejidad es O(n) donde n es el número de entidades que contenga su vecindario.

El **cComplex_Behavior** tiene complejidad O(n^2) donde n es el número de algoritmos que contenga en la lista, ya que estos algoritmos pueden ser de complejidad O(n).

Por tanto la aplicación tiene complejidad O(n^2) ya que se toma la del algoritmo que mayor complejidad tenga.

Es válido aclarar que la complejidad depende de las necesidades del usuario, podría variar desde O(n) si se tratan comportamientos simples, hasta O (n^x) si se crean comportamientos en aplicaciones futuras, que requieran que el algoritmo **cComplex_Behavior** contenga en su lista otros **cComplex_Behavior**.

Conclusiones Generales

Para el cumplimiento de los objetivos de este proyecto, se realizó primeramente un estudio de las técnicas, tecnologías y tendencias actuales en relación con el tema de los *Steering Behaviors*. En el cual se analizaron sus principales características para elaborarlos. Además a partir de esta investigación se proponen las características técnicas de la aplicación propuesta como solución.

Posteriormente se pasó a implementar los algoritmos de locomoción y se realizaron pruebas que garantizaron su correcto funcionamiento aun en condiciones extremas.
Al terminar esto, se unieron y pasaron formar parte de la aplicación propuesta. Demostrando con buenos resultados en la creación y combinación de los *Steering Behaviors* el cumplimiento de los requisitos y finalmente de los objetivos trazados para el módulo de algoritmos de locomoción.

Recomendaciones

- Se recomienda implementar nuevos *Steering Behaviors* que se adicionen a los de la aplicación.
- Profundizar en el estudio de las fuerzas que actúan en cada comportamiento simple para crear otros tipos de comportamientos grupales.
- Acoplar la biblioteca de comportamientos al proyecto de biblioteca de AI de la Facultad 5
- Implementar algoritmos de partición espacial para optimizar la búsqueda de agentes y obstáculos en la vecindad y de esta forma reducir la complejidad del algoritmo.
- Hacer uso de técnicas de procesamiento paralelo (para las máquinas con más de un procesador o multithreading) de forma que se disminuya el tiempo de procesamiento.

Referencias Bibliográficas

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, pages 264-271.* 1987.

BIRN, Jeremy. 2000. *Digital lightning and rendering. New Rider Publisher. USA. .* 2000.

Braitenberg, Valentino. 1984. *Vehicles: Experiments in Synthetic Psychology, The MIT Press, Cambridge,MA.* 1984.

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

Costa, Monica, Feij., Bruno and and Schwabe, Daniel. 1990. *Reactive Agents in Behavioral Animation. .* 1990.

Kahn, Kenneth. 1979. *Creation of Computer Animation from Story Descriptions.* 1979.

Mataric, Maja. 1993. *Designing and Understanding Adaptive Group Behavior . Adaptive Behavior 4(1), pages 51-80.* 1993.

Panne, van de, M., Fiume and E., Vranesic. 1990. *Reusable Motion Synthesis Using State-Space Controllers.: Proceedings of SIGGRAPH `90, In Computer Graphics Proceedings, ACM SIGGRAPH, pages 225-234, 1990.* 1990.

Reinold, Craig. 1986. *dhgdgf.* 1986.

Resnick, Mitchel. 1993. Behavior Construction Kits. [En línea] 1993. http://el.www.media.mit.edu:80/groups/el/Papers/mres/BCK/BCK.html. CACM, 36(7)..

—. 1989. *Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (ALife Õ87), 1989. pages 397-406.* 1989.

—. 1998. The Virtual Fishtank. [En línea] 1998. http://el.www.media.mit.edu/groups/el/projects/fishtank/.

Reynolds, C. W. 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics,21(4) (SIGGRAPH Õ87 Conference Proceedings) pages 25-34. . [En línea] 1987. http://www.red.com/cwr/boids.html.

Reynolds, Craig. 2004. OpenSteer. *Steering Behaviors for Autonomous Characters.* [En línea] 2004. http://opensteer.sourceforge.net/.

Ridsdale, Gary. 1987. *The Directors Apprentice: Animating Figures in a Constrained Environment.* 1987.

Still, G Keith. 1994. *Simulating Egress using Virtual Reality - a Perspective View of Simulation and Design. .* 1994.

Thalmann, Daniel, Renault, Olivier and Nadia, Magnenat-Thalmann. 1990. *A Vision-Based Approach to Behavioral Animation. : Journal of Visualization and Computer Animation, 1990. John Wiley & Sons, 1(1) pages 18-21.* 1990.

Travers, Michael. 1994. LiveWorld: A Construction Kit for Animate Systems: Proceedings of ACM CHI 94 Conference on Human Factors in Computing Systems, pages 37-38. [En línea] 1994.

Travers, Michael. 1989. *1989. Animal Construction Kits : Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (ALife 87) SFI Studies in the Sciences of Complexity, Volume 6, pages 421-442.* 1989.

Wiener, Norbert. 1948. *Cybernetics, or control and communication in the animal and the machine.* 1948.

Wilhelms, Jane and Skinner, Robert. 1990. *A "Notion" for Interactive Behavioral Animation Control, IEEE Computer Graphics and Applications, 10(3).* 1990.

Zapata, R. 1992. *Reactive Behaviors of Fast Mobile Robots in Unstructured Environments: Sensor-Based Control and Neural Networks.* 1992.

Zeltzer, B. Barsky and and N. Badler and D. 1990. *Control and Animation of Articulated Figures. 1990.* 1990.

Zeltzer, David. 1983. *Knowledge-Based Animation. Proceedings SIGGRAPH/SIGART Workshop on Motion. pages 187-192 : s.n.* 1983.

Bibliografía Consultada

Buckland, Mat. 2005. *Programming Game AI by Example.* 2005.

Cooper, Adrien Treuille and Seth. 2006. Continuum Crowds. [En línea] 2006. http://grail.cs.washington.edu/projects/crowd-flows/continuum-crowds.pdf.

Halleux, Jonathan de. 2003. MetaAgent, a Steering Behavior Template Library. [En línea] 2003. http://www.codeproject.com/KB/library/metaagent.aspx.

Jaganathan, Sivakumar and Koshti, Jitendra. 2007. *Intelligent Agents: Incorporating Personality into Crowd Simulation.* 2007.

Kuffner, Manfred Lau and James. 2005. *Behavior Planning for Character Animation, ACM SIGGRAPH / Eurographics Symposium on Computer Animation.* Los Angeles : s.n., 2005.

Macey, Jonathan. 2003. *Emergent Behaviour Programming Language E.B.P.L.* 2003.

Wang, Hongling and Kearney, J.K. and Cremer, J. and Willemsen, P. 2005. *Steering behaviors for autonomous vehicles in virtual environments: Proceedings. VR 2005. IEEE Volume , Issue , Page(s):155 - 162.* 2005.

Índice de Figuras y Tablas

ÍNDICE DE FIGURAS

Índice de Tablas

Glosario de Términos

Animación: Simulación de un movimiento creado por la muestra de una serie de imágenes o fotogramas.

Frame: Cada una de las imágenes que componen una animación.

Vector: Cantidad que expresa magnitud y dirección.

Agente Autónomo:

Un agente autónomo es una entidad dentro de un entorno que cambia su estado con el tiempo. Para ello puede poner en práctica algoritmos de locomoción que dado una estructura a su alrededor (obstáculo, otra entidad, etc.) le permiten calcular los movimientos a seguir. Estos agentes constituyen personajes en los juegos que tienen la capacidad de improvisación a la hora de seleccionar sus acciones, se les suele llamar caracteres no jugadores.

***Steering Behaviors* (Comportamientos):**

Son reglas de movimiento a seguir por los agentes autónomos a la hora de tomar decisiones ante los diferentes estímulos que ejercen sobre ellos los demás componentes de su entorno.

Obstáculo:

Un obstáculo es algo que impide el paso a través de él. Los agentes pueden reconocer los obstáculos y ser influidos por ellos. Un ejemplo de obstáculo dentro de un juego puede ser un muro.

Entorno:

Escenario en el que se encuentran los agentes y obstáculos.

Glosario de Abreviaturas

AI: Inteligencia Artificial.

UCI: Universidad de las Ciencias Informáticas.

RV: Realidad Virtual.

UML: Lenguaje Unificado de Modelado.

FPS: Frame por Segundos.

OO: Orientada a Objetos.