**Universidad de las Ciencias Informáticas**

**“Facultad 5 Entornos Virtuales”**


# Titulo del trabajo: Módulo para el comportamiento autónomo de autos en un Entorno Virtual urbano.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

**Autores: Alejandro Benítez Herrera**

**Gelson Rafael Saurín Ojeda.**

**Tutor: Msc. Yuniesky Coca Bergolla**

**Asesor: Msc. Oscar Julian Villar Barroso**

Ciudad de La Habana, Junio del 2008

**DECLARACIÓN DE AUTORÍA**

Declaramos que somos los únicos autores de este trabajo y autorizamos a 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 _____.

Alejandro Benítez Herrera

Gelson Rafael Saurín Ojeda

_____

_____

Tutor: Msc. Yuniesky Coca Bergolla

_____

**DATOS DE CONTACTO**

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

Graduado de Licenciatura en Ciencias de la Computación en el curso 2002-2003 en la Universidad Central de Las Villas.
Se incorpora a trabajar en la Universidad de las Ciencias Informáticas donde se desempeña como Jefe de Dpto. de Práctica Profesional e Ingeniería de Software.
Defendió satisfactoriamente en el curso 2006-2007 su maestría en Informática Aplicada.
Ha sido tutor de dos tesis de grado vinculadas al tema de la IA para Aplicaciones de Realidad Virtual.
Es miembro fundador del Grupo de Desarrollo de Elementos Virtuales Inteligentes (UVi-Bot).

Msc. Oscar Julian Villar Barroso (villar@uci.cu).

Licenciado en Historia.
Máster en Relaciones Internacionales por la Universidad de La Habana.
Jefe del Colectivo de Disciplina de Ciencias Sociales de la Facultad 5.
Colaborador del Centro de Estudios sobre Asia y Oceanía del CC del PCC.
Colaborador del Centro de Estudios de Investigación y desarrollo del MINREX de la República Argentina.
Ha participado en eventos nacionales e internacionales.
Tiene varias publicaciones en Cuba y en el exterior.
Es profesor asistente.

**AGRADECIMIENTOS**

*A la dirección de la Revolución Cubana, a nuestro Comandante Fidel Castro y a la UCI, por permitirnos formar parte de este proyecto.*

***Alejandro y Gelson***

*A mi familia: por su amor, sus enseñanzas, sus años de sacrificio, su confianza, sus consejos, y apoyo.*

*A mi madrina: que siempre estuvo apoyándome en todas las etapas de mi vida.*

*A todos mis amigos (William, Duniel, Macia, Julio, Amaury, Maydelis, Yeilin, Neivy, Orlando) por siempre ayudarme en los momentos difíciles.*

*A mi compañero de tesis Gelson por apoyarme en estos 5 años de la carrera y por ser mas que un amigo.*

*A mis compañeros de aula (Parra, Jorge, Armando, Dayani) que me ayudaron mucho.*

*A mi novia que a pesar de estar distante siempre estuvo pendiente y apoyándome.*

***Alejandro***

*A mis padres por su educación, por su preocupación, por su apoyo incondicional, por siempre estar a mi lado y por hacer de mí el hombre que soy hoy.*

*A mi abuela Esther por sus consejos y por sus palabras de aliento.*

*A mi abuelo Alberto que no me pudo ver en este día.*

*A mis tíos Jorge y Francisco por su preocupación.*

*A mis tías Mercy y Chichi.*

*A mis primos José Enrique, Norgel, Jani y Ernestico.*

*A mi novia Daimeris, por todo su apoyo, por darme tanta fuerza y por siempre estar a mi lado.*

*A mis amigos William, Duniel, Macias, Julio, Yeilin y Amaury por su apoyo y por su amistad.*

*A mis otros amigos que no mencioné, pero que son igual de importantes para mi.*

***Gelson***

**DEDICATORIA**

*A mis padres por el apoyo y por el cariño brindado.*

*A mi familia por siempre estar presente y apoyándome.*

*A todos mis amigos*

*A mi novia*

***Alejandro***

*A mis padres por el apoyo y por el cariño brindado, a mi familia por seguir tan de cerca mi formación.*

*A mi novia Daimeris que siempre estuvo cuando la necesitaba.*

*A mis amigos, parte de este logro es gracias a ustedes.*

***Gelson***

**RESUMEN**

Desde el comienzo del estudio de la Inteligencia Artificial en nuestra universidad, principalmente en la Facultad 5 se ha fomentado el aprendizaje de las diversas técnicas que existen; esto tiene como único fin dominar este campo tan novedoso y que tantos provechos le puede aportar a la informática en nuestro país.

Siguiendo la línea de trabajo de la Facultad se ha planteado crear un módulo de comportamiento inteligente para autos, de forma que estos se muevan de forma autónoma, respetando las leyes del tránsito, o sea, cumpliendo con las normativas que siguen los choferes en el mundo real.

Para poder obtener los resultados esperados de esta investigación se realizó un análisis de las diferentes técnicas de Inteligencia Artificial, así como también se profundizó en la temática de los Agentes Autónomos; con lo que pudimos definir exactamente que desarrollaríamos para darle solución a lo planteado.

Se desarrolló un módulo que contiene todos los elementos para manipular la manera en la que se deben comportar los vehículos y por donde deben realizar su desplazamiento, viéndose de esta forma cumplido el principal objetivo del trabajo.

La realización de este trabajo ha dado como fruto un producto para contribuir al desarrollo de aplicaciones que requieran de entornos virtuales en el que deban simularse el ajetreo diario que realizan los vehículos en las calles de cualquier ciudad.

**Palabras Claves:** Inteligencia Artificial, módulo de comportamiento inteligente, Agentes Autónomos, entorno virtual de ciudad.

**ÍNDICE DE CONTENIDOS**

**ÍNDICE DE FIGURAS**

## ÍNDICE DE TABLAS

## INTRODUCCIÓN

La tecnología ha progresado realmente rápido, gracias, fundamentalmente, a nuestra capacidad de imaginar qué queremos lograr. Hoy, podemos ver e incluso realizar procesos, tareas, que hasta hace solo unos años atrás parecían imposibles, pero lo cierto es que la tecnología ha llegado a un punto en el que el término “imposible” no existe, transformando así el mundo, no solo de la informática sino también de diversas áreas como la medicina, la arquitectura, la educación y la ingeniería, entre otras.

Como parte del proceso de informatización y desarrollo en Cuba, poco a poco ha ido creciendo el uso de la realidad virtual; nos estamos introduciendo en el mundo de los entornos virtuales, del diseño 3D y de los simuladores, los cuales encuentran su aplicación en los diferentes procesos de aprendizajes a los cuales se somete el ser humano. La Universidad de las Ciencias Informáticas, específicamente la Facultad 5 se especializa en todo lo referente a este tema con la creación de varios proyectos, que desarrollan y explotan cada uno de los elementos de esta línea de trabajo.

Por mucho tiempo a la hora de recrear un entorno virtual uno de los grandes problemas ha sido la forma en que se comportarán los elementos que formarán parte del mismo, de forma que ejecuten de la manera más real posible el papel que les tocó representar, ya sea en un juego o en un Simulador, permitiendo que el usuario puede incursionar creativamente, hasta donde el límite de su imaginación se lo permita.

Específicamente, en un entorno de ciudad los carros deben respetar las leyes del tránsito, se puede hablar de señales como los ‘pare’, los ‘ceda el paso’, ‘semáforo’, señales que limitan la velocidad máxima, entre otras muchas. Sin embargo este es solo la primera parte, la más visible; los autos siempre se mueven (en la mayoría de los países) por la senda derecha, cuestión que no se puede olvidar en un trabajo como este. Pero los autos no siempre van uno detrás del otro, en determinados momentos algún auto que lleva mayor velocidad que otro lo adelanta, para lograr esto se requiere de un chequeo del entorno, saber si se encuentra en un lugar que puede realizarse dicha maniobra, pero aunque el auto esté en un lugar que pueda hacerlo, pudiera ser que se acerque un auto por la senda contraria y entonces tampoco puede adelantar al que está delante.

Actualmente no se ha podido lograr un sistema inteligente que permita que los vehículos se desplacen por el entorno de forma autónoma cumpliendo con los requerimientos antes mencionados. De aquí surge la siguiente interrogante, constituyendo el **problema científico** a resolver, ¿Cómo elaborar un módulo que permita el comportamiento autónomo de los autos en una ciudad virtual?

Como **objeto de estudio** se trabaja con los comportamientos de vehículos en Entornos Virtuales. El **campo de acción** se basa en el estudio de los comportamientos de vehículos para mundos urbanos virtuales. El **objetivo** principal que propone este proyecto es presentar un módulo que permitan el movimiento autónomo de los vehículos en un entorno virtual de ciudad.

A continuación se plantean un grupo de **tareas** que permitirán satisfacer los objetivos, y que se pueden resumir en las siguientes:

- Realizar un estudio del problema y de la situación actual del tema.
- Dominar el uso de los *Steering Behaviors*, para darle solución al problema.
- Elaborar la documentación necesaria vinculada a la investigación.
- Formular una propuesta que constituya la base científica para la implementación del módulo que dará solución al problema científico.
- Implementar la solución del problema.

El presente trabajo está estructurado de la siguiente manera: Resumen, Introducción, tres Capítulos de Contenido, Conclusiones, Recomendaciones, Referencias Bibliográficas, Bibliografía Consultada y el Glosario de Términos. A continuación se hace una breve descripción del contenido de los capítulos.

- **Capítulo 1. Fundamentación Teórica.** Inteligencia Artificial y Vida Artificial, principales técnicas y características, problemas en los que son más frecuentemente utilizados, *Behaviors* de Craig Reynols; Agentes Autónomos, características, condiciones, aplicaciones.

- **Capítulo 2. Propuesta de Solución.** Descripción de la solución propuesta. *Behaviors* creados. Descripción de las metodologías y herramientas empleadas. Definición de los procesos, casos de uso del negocio, diagramas de clases del modelo de objetos del negocio; requisitos funcionales y no funcionales; actores y casos de uso del sistema.

- **Capítulo 3.Características del Sistema.** Descripción del diseño a través de diagramas de clases, análisis y diseño de la aplicación, se realiza la descripción de las clases implicadas en el proceso de desarrollo, se incluyen los diagramas de Componente y de Despliegue.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## Introducción

En este capítulo se brinda una explicación sobre la Inteligencia y la Vida Artificial, sus técnicas y donde son más comúnmente utilizadas; se presenta también un estudio sobre los agentes autónomos, donde se incluyen sus aspectos más relevantes; es fundamental consultar lo investigado en este capítulo ya que del entendimiento de lo expuesto en el mismo es de donde surge la propuesta de solución de este trabajo.

## 1.1 Inteligencia Artificial

La Inteligencia Artificial (IA) se define como la inteligencia presentada por un ente artificial, generalmente, una computadora [1]. Ésta surge como resultado de dos áreas del conocimiento muy importantes: la psicología cognitiva y la lógica matemática; en la actualidad se ha enfocado su estudio hacia la explicación del trabajo mental y el desarrollo y construcción de algoritmos para la solución de problemas de propósito general. En esta medida, la Inteligencia Artificial puede definirse como el resultado de la combinación entre las ciencias de la computación, la fisiología y la filosofía en pro de la creación de entes capaces de generar pensamientos, es decir, entes que puedan pensar.

En otras palabras, la Inteligencia Artificial puede entenderse como una disciplina científico-técnica que intenta generar sistemas artificiales con la capacidad de realizar comportamientos, que de ser hechos por seres humanos, se diría que requieren de inteligencia, por lo que la Inteligencia Artificial no es más que un intento por reproducir la forma en que los hombres identifican, estructuran y resuelven problemas de la vida diaria.

La Inteligencia Artificial se diferencia de otras disciplinas de la computación por varias características, como por ejemplo la introducción del uso de símbolos no matemáticos. Sin embargo la diferencia más clara se halla en el comportamiento de los programas, pues éste se caracteriza por no ser descrito explícitamente por el algoritmo sino que la secuencia de pasos que sigue es influenciada por el problema particular presente. De esta forma es el programa el que especifica cómo encontrar la secuencia

necesaria para resolver un problema determinado. En contraste a esta situación se encuentran los programas que no son de Inteligencia Artificial que siguen un algoritmo definido, en el cual se especifica explícitamente cómo encontrar las variables de salida a partir de las variables de entrada [2].

Actualmente, las investigaciones en esta área se encuentran aplicadas en la robótica, ya sea en los empleados en la industria, utilizados en trabajos de montaje, ensamblaje y control, como en los modelos creados para representar los cinco sentidos básicos de los seres humanos (la visión, la audición, el tacto, el olfato, la degustación), en la medicina, ayudando a los médicos a hacer diagnósticos, supervisar la condición de los pacientes, administrar tratamientos y preparar estudios estadísticos; en los satélites artificiales, en los grandes aeropuertos, en las empresas como prototipos informáticos de los sistemas de apoyo a la decisión, de comunicación, de seguridad, entre otros, en el trabajo con Lenguajes Naturales, reconociendo las palabras, no como simples caracteres sino por el significado que tienen en el mundo de los seres humanos, en los entornos virtuales, donde es ampliamente utilizada en simuladores y en juegos, en el control de sistemas, en el reconocimiento de fallas, en la planificación automática, en aplicaciones que realicen la función de consultores, por su habilidad de responder a diagnósticos y preguntas de consumidores [3].

Es claro que dentro del ámbito de las ciencias de la computación la Inteligencia Artificial es una de las áreas que más expectación causan, debido a que busca comprender los mecanismos de la inteligencia, tema que es también de gran interés para otras áreas del conocimiento porque constituye una de las “incógnitas” más estudiadas a lo largo de los años. Dentro de los muchos campos que abarca esta disciplina, lo que más excitación ha generado es el aprendizaje de máquinas por ser un elemento vital dentro del proceso para emular los comportamientos inteligentes. La idea de que los sistemas puedan mejorar su comportamiento a partir de la experiencia, que puedan reconocer un error y que puedan evitarlo o corregirlo, resulta apasionante y atrayente.

### 1.2 Técnicas de Inteligencia Artificial

La Inteligencia Artificial se divide en dos corrientes principales: la denominada Inteligencia Artificial Convencional y la conocida como Inteligencia Computacional. La primera de estas corrientes tiene que ver con los métodos que actualmente se conocen como máquinas de aprendizaje y se caracteriza por el

formalismo y el análisis estadístico. Dentro de esta corriente las técnicas que se encuentran son: los sistemas expertos (capacidad de razonamiento para lograr conclusiones), el razonamiento basado en casos, la red Bayesiana (representación del conocimiento basado en la teoría de probabilidad) y la Inteligencia Artificial basada en comportamientos. Por su parte, la Inteligencia Computacional implica el desarrollo o aprendizaje iterativo que se realiza en base a datos empíricos. Algunas técnicas de esta rama son: redes neuronales (grandes capacidades de reconocimientos de patrones), sistemas difusos (razonamiento bajo incertidumbre) y computación evolutiva (conceptos de la biología aplicados a la solución de problemas).

### 1.2.1 Sistemas Basados en Reglas

Una **regla** es definida como un modo de representación estratégica o técnica, la cual es apropiada cuando el conocimiento con el que deseamos trabajar proviene de la experiencia o de la intuición, y por tanto carece de una demostración física o matemática

Los **sistemas basados en reglas** trabajan mediante la aplicación de reglas, comparación de resultados y aplicación de las nuevas reglas basadas en la nueva situación. También pueden trabajar por inferencia lógica dirigida, bien empezando con una evidencia inicial en una determinada situación y dirigiéndose hacia la obtención de una solución, o bien con hipótesis sobre las posibles soluciones y volviendo hacia atrás para encontrar una evidencia existente (o una deducción de una evidencia existente) que apoye una hipótesis en particular [4].

Como en todos los sistemas de este tipo las reglas pueden estar *encadenadas*, de forma que el consecuente de una regla pasa a ser el antecedente de la siguiente. Las reglas que no son encadenadas, se les denominan *reglas paralelas*.

### 1.2.2 Sistemas Expertos

Los sistemas expertos se basan en la simulación del razonamiento humano. El razonamiento humano tiene para ellos, un doble interés: por una parte, el del análisis del razonamiento que seguiría un experto humano en la materia a fin de poder codificarlo mediante el empleo de un determinado lenguaje

informático; por otra, la síntesis artificial, de tipo mecánico, de los razonamientos de manera que éstos sean semejantes a los empleados por el experto humano en la resolución de la cuestión planteada [5].

Los sistemas expertos son, por lo tanto, intermediarios entre el experto humano, que transmite sus conocimientos al sistema, y el usuario de dicho sistema, que lo emplea para resolver los problemas que se le plantean con la competencia de un especialista en la materia y que, además, puede adquirir una destreza semejante a la del experto gracias a la observación del modo de actuar de la máquina. Los sistemas expertos son, pues, simultáneamente, un sistema de ejecución y un sistema de transmisión del conocimiento. Asimismo, los sistemas expertos se definen mediante su arquitectura; obtienen, por lo tanto, una realidad palpable.

### 1.2.3 Redes Bayesianas

Las Redes Bayesianas sirven para evaluar sistemáticamente el riesgo asociado a una tarea; son las representaciones gráficas de variables "inciertas" ("nodos") y sus dependencias cuantificadas (“arcos”), lo que constituye una manera práctica y compacta de representar el conocimiento incierto [6].

Utilizando modelos computarizados, se puede cuantificar sistemáticamente todas las variables relevantes para una evaluación global de la situación. Cada "estado" discriminado en un nodo es definido como: "Favorable", "Neutro" o "Desfavorable" para la "Actitud del Ente Regulador". Esta cuantificación por lo general nos lleva a una evaluación global de la situación más precisa y "objetiva". El modelo es dinámico ya que los parámetros y por lo tanto la evaluación global pueden ser actualizados luego de haber obtenido nueva información.

### 1.2.4 Lógica difusa

La lógica difusa es una rama de la Inteligencia Artificial que se funda en el concepto: "Todo es cuestión de grado", lo cual permite manejar información vaga o de difícil especificación en caso de que se quiera hacer cambiar con esta información el funcionamiento o el estado de un sistema especifico [7]. Es entonces

posible con la lógica difusa gobernar un sistema por medio de reglas de 'sentido común' las cuales se refieren a cantidades indefinidas.

Básicamente es un tipo de lógica que reconoce más que simples valores verdaderos y falsos. Con lógica difusa, las proposiciones pueden ser representadas con grados de veracidad o falsedad. Por ejemplo, la sentencia "hoy es un día soleado", puede ser 100% verdad si no hay nubes, 80% verdad si hay pocas nubes, 50% verdad si existe neblina y 0% si llueve todo el día.

La Lógica Difusa ha sido probada para ser particularmente útil en sistemas expertos y otras aplicaciones de Inteligencia Artificial. Es también utilizada en algunos correctores de voz para sugerir una lista de probables palabras a reemplazar en una mal dicha. La Lógica Difusa actualmente está relacionada y fundamentada en la teoría de los Conjuntos Difusos. Según esta teoría, el grado de pertenencia de un elemento a un conjunto va a venir determinado por una función de pertenencia, que puede tomar todos los valores reales comprendidos en el intervalo [0,1].

### 1.2.5 Algoritmos Genéticos

Un **algoritmo genético** es una técnica de búsqueda utilizada en informática para encontrar soluciones aproximadas a problemas de optimización y búsqueda. Los algoritmos genéticos son una clase particular de algoritmos evolutivos que utilizan técnicas inspiradas por la evolución biológica cómo herencia, mutación, selección, y cruce [8].

Los algoritmos genéticos se implementan típicamente como una simulación informática en la cual una población de representaciones abstractas (denominadas cromosomas) de soluciones candidatas (denominadas individuos) a un problema de optimización evoluciona hacia mejores soluciones. Tradicionalmente, las soluciones se representan como series binarias de 0s y 1s, pero las codificaciones diferentes son también posibles. La evolución empieza desde una población de individuos completamente aleatorios y pasa por diferentes generaciones. En cada generación, la adecuación de la población entera se evalúa, se seleccionan múltiples individuos de manera estocástica de la población actual (basada en su

adecuación), y se modifican (haciéndolos mutar) para formar una nueva población. La nueva población se utiliza en la siguiente iteración del algoritmo.

### 1.2.6 Redes Neuronales

Una Red Neuronal es usada para aprender patrones y relaciones de datos. Los datos pueden ser el resultado del esfuerzo de una investigación de mercado, el resultado de un proceso de producción dando variación a las condiciones de operación, o las decisiones de un prestamista dado un conjunto de aplicaciones de préstamo, utilizando una Red Neuronal es una salida considerable parecida a un enfoque tradicional. Tradicionalmente un programador o un analista especifican "códigos" de cada faceta del problema en orden para la computadora pueda "entender" la situación. Las Redes Neuronales no requieren el código explícito del problema [9].

Aunque su estructura varía según el tipo de red, lo más usual es que haya tres capas de neuronas, una de entrada, que recoge los estímulos, otra oculta, que procesa la información, y otra de salida, que ejecuta la respuesta. La figura siguiente muestra esta disposición:

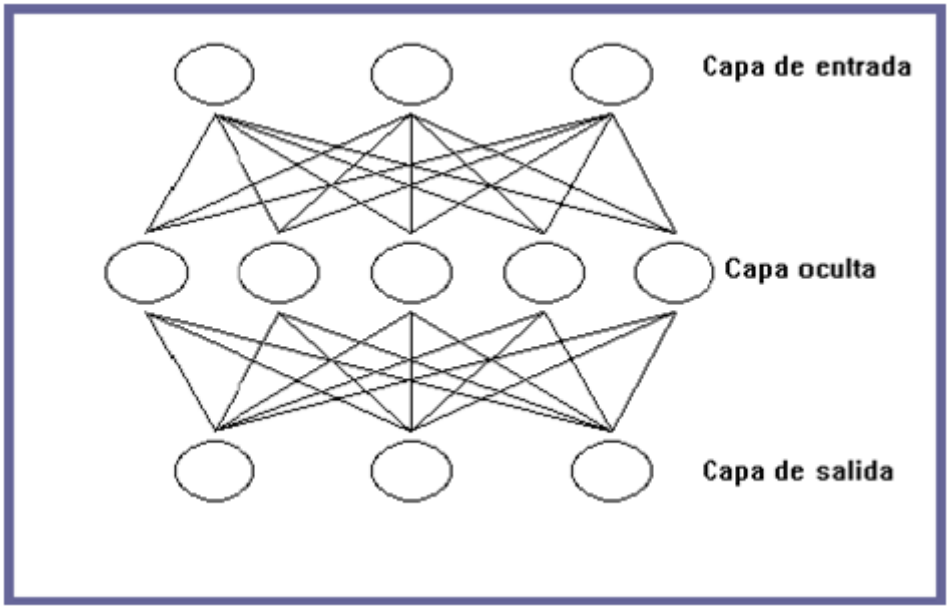


**Figura 1 Red Neuronal**

Su ventaja más importante está en resolver problemas que son demasiado complejos para tecnologías convencionales, problemas que no tienen un algoritmo de solución o que su algoritmo de solución es muy difícil de encontrar. En general, a causa de su abstracción del cerebro biológico, las Redes Neuronales son aptas para resolver problemas que la gente puede resolver, pero que las computadoras no pueden. Estos problemas incluyen reconocimiento de patrones y pronósticos (los cuales requieren el reconocimiento de tendencias de datos).

### 1.3 VIDA ARTIFICIAL

La Vida Artificial ha sido definida como “la ciencia que trata de situar la vida "tal como es" dentro del contexto de la vida "tal como podría ser" ” [10]. En los últimos años se han realizado muchos estudios sobre Vida Artificial y este concepto sigue siendo motivo de polémica. La Vida Artificial y la Inteligencia Artificial coexisten de muy buena manera.

La Vida Artificial tiene dos áreas principales, por una parte se ocupa de algoritmos que pueden imitar a la naturaleza, como el Cellular Automata y por otra parte, estudia simulaciones de comportamientos como

por ejemplo el comportamiento de un grupo de hormigas. La Inteligencia Artificial tiende a imitar (o a crear) inteligencia humana. En ambas disciplinas se utilizan técnicas similares como los algoritmos genéticos, pero con el transcurso de los años, se han considerado estudios distintos [11].

Para entender qué es la Vida Artificial, es necesario saber qué es la vida. Hay muchas características que poseen los organismos vivos como: crecimiento, reproducción, respuesta al medio ambiente, adaptabilidad, metabolismo, autonomía, capacidad de reacción, evolución, etc. Sin embargo, hay sistemas además de los vivos que cuentan con algunas de estas características como el fuego, la Tierra y algunos robots.

Hay una característica común de los seres vivientes y esta es la habilidad de categorizar y controlar los eventos en su medio ambiente y la capacidad de almacenar y transmitir estos conocimientos.

Todas estas características deben ser cumplidas por los sistemas de Vida Artificial. Además, deben de seguir las leyes de la física y de la biología. Las leyes de la física son independientes de los organismos, es decir, son universales, se cumplen para todos los seres y objetos (como la gravedad). Por otra parte, las leyes de la biología son diferentes para cada especie y dependen del medio ambiente.

La Vida Artificial es el estudio de sistemas hechos por el hombre que exhiben comportamientos característicos de los sistemas naturales vivientes. Complementa la biología tradicional ocupándose del análisis de organismos vivos intentando sintetizar comportamientos con computadoras y otros medios artificiales. “El significado de la palabra Artificial dentro del término Vida Artificial se refiere a las partes componentes, no a los procesos emergentes. Si las partes de los componentes son implementadas correctamente, los procesos que realicen serán genuinos, entonces, la Vida Artificial será vida genuina, solamente formada por componentes diferentes a los de la vida en la Tierra” [12].

Un sistema de Vida Artificial se compone de organismos que siguen un conjunto de reglas simples y generan un comportamiento a partir de estas reglas. El estudio de este comportamiento es la parte más importante de la Vida Artificial. Este comportamiento es impredecible y pequeños cambios en el ambiente

generan comportamientos completamente diferentes. Esto significa que los organismos son auto-organizados.

Para que un sistema sea auto-organizado debe "ser capaz de mejorar su desempeño mientras persigue sus metas, sin ayuda externa". Un comportamiento auto-organizado forma estructuras, patrones o conductas a partir de condiciones iniciales aleatorias, es conocido como sinergia y los sistemas son conocidos como sistemas dinámicos. Sin embargo, estas condiciones iniciales también pueden dar lugar a un comportamiento caótico y el área de interés de la Vida Artificial se encuentra entre la transición, el caos y el orden y es conocida como un tipo de complejidad organizada.

### 1.3.1 Algoritmos de Vida Artificial

**Cellular Automata**

La definición del Cellular Automata es: Conjunto regular de células cada una de las cuales puede tener cualquier número finito de estados. El estado de todas las células en el conjunto, es actualizado simultáneamente y el estado del conjunto avanza en lapsos discretos de tiempo. El estado de cada célula en el conjunto es actualizado de acuerdo a una regla local que puede depender del estado de la propia célula y de sus vecinos en un lapso de tiempo anterior [13].

Es un sistema dinámico discreto, el cual involucra reglas simples determinadas, como en cualquier sistema los cambios de variables están en función de sus valores predefinidos y se considera una idealización matemática en donde el espacio y el tiempo son caracterizados de manera discreta, así las cantidades relacionadas toman valores discretos.

Una automatización celular consiste en un enrejado uniforme y regular, que generalmente es extenso con una variable discreta para cada sitio, a la cual denominamos "Célula", el valor del sitio de la variable comienza a ser afectado por el valor de una variable que se encuentra en una "vecindad". Las vecindades son los sitios alrededor de cierta célula, las variables de cada sitio están sincronizadas, basadas en los valores de las variables en sus vecindades y prescindiendo del tiempo.

Los ejemplos más conocidos de autómatas celulares son el Cellular Automata de una dimensión (1D CA) creado por Wolfram y el Cellular Automata de dos dimensiones (2D CA) utilizado en el “Conway´s life”.

**Wolfram's 1D-CA:**
Wolfram creó un programa en donde cada una de las células estaban apagadas o prendidas (vivas o muertas). En la configuración inicial el programa comienza con un conjunto aleatorio de células apagadas, entonces las células debajo de la línea inicial son determinadas por la línea anterior. Hay ocho posibles combinaciones para la línea A que determinarán la línea B (000, 001, 010, 011, 100, 101, 110, 111). Los resultados son dependientes del conjunto de reglas (hay 256 posibilidades) [14].

**Conway's Life**
Un buen ejemplo de Vida Artificial es este programa. Las reglas de Conway´s Life son simples (cada píxel representa una célula). Una célula no puede vivir con más de tres o con menos de dos células adyacentes a ella. Si el número de células alrededor de un espacio vacío es igual a tres, entonces nace una nueva célula. Estas simples reglas llevan a un comportamiento complejo. El sistema eventualmente se estabilizará, ya sea muriendo por completo o porque encuentra un punto de equilibrio [15].

Hay un patrón de células llamado Glider. Estas células, después de unos cuantos ciclos, se repetirán a sí mismas, pero en una posición diferente. Los Gliders pueden acomodarse de tal forma que puedan llevar a cabo operaciones lógicas como AND, OR, NOT, XOR, etc.

**Scripts de Arena**
La primera investigación sobre organismos auto-organizados fue realizada por Per Bak en 1987. Creó un modelo sobre el comportamiento de un montón de arena. Los organismos eran los granos de arena y el montón crece hasta llegar a un punto en que no puede crecer más, entonces se generan pequeñas avalanchas. No se sabe si el siguiente grano de arena que llegue al montón generará un movimiento en éste, ni tampoco se sabe de qué tamaño será este movimiento. Sin embargo, se sabe que todos los montones de arena tendrán este comportamiento al llegar a un tamaño determinado [16].

En el sistema creado por Per Bak, se demuestra que la frecuencia de los movimientos de la arena depende del tamaño del movimiento. Es decir, un movimiento diez veces mayor que el movimiento más

pequeño se produce diez veces menos frecuentemente. Si se dibuja esto en una escala logarítmica, el número de movimientos de cada magnitud ocurridos aparece en una línea recta. Esto es conocido como una ley potencial. Entonces, aunque es imposible predecir el tamaño del siguiente movimiento, se puede predecir que con el paso del tiempo se van a producir un número determinado de movimientos de una magnitud específica.

**La Hormiga de Langston**

Este algoritmo consiste en un plano cuadriculado que tiene cuadros blancos y negros. La hormiga camina sobre este plano utilizando dos reglas muy sencillas:

- Si la hormiga está en un punto negro, lo pinta de blanco, avanza hacia adelante después hacia la derecha.

- Si la hormiga está en un punto blanco, lo pinta de negro, avanza hacia adelante y después hacia la izquierda.

Al principio, pareciera que la hormiga tiene un comportamiento aleatorio. Sin embargo, después de un tiempo se descubre que tras ejecutar estas reglas muchas veces, comienza a surgir un patrón. La hormiga se encuentra en una secuencia de la que no puede escapar. Además, independientemente del conjunto inicial de puntos en el que comience la hormiga, siempre acaba cayendo en la misma secuencia [17].

### 1.4 *Behaviors* de Craig Reynolds

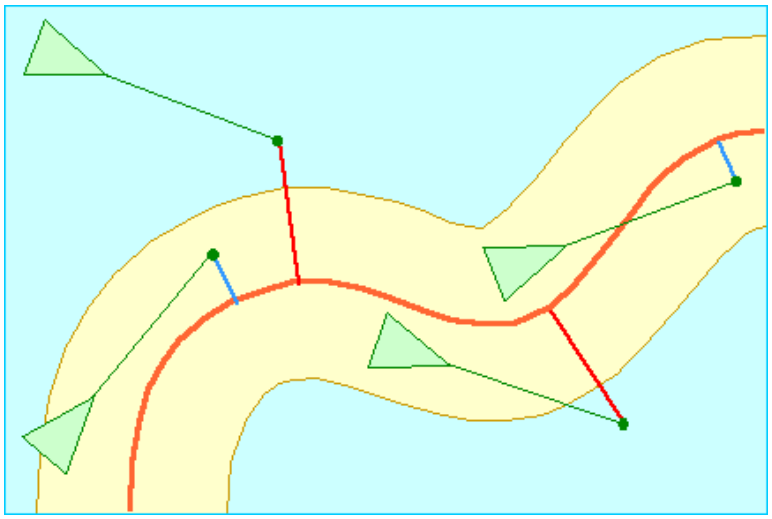

**Figura 2 Representación de un entorno regido por los *Behaviors***

Fue ideado en 1986 por Craig Reynolds y ya que en aquel entonces no se utilizaba todavía el concepto de Vida Artificial, se conoce a estos algoritmos como uno de los precursores. En aquellos años, los conceptos de organismos auto-organizados apenas se estaban definiendo. Estos algoritmos estudian el comportamiento emergente de los conjuntos, basándose fundamentalmente en los comportamientos observados en los animales a partir de tres reglas sencillas.

Craig Reynolds consideraba su algoritmo como parte de los sistemas de partículas. Un sistema de partículas es un conjunto de un gran número de partículas, en el cual cada una de ellas tiene su propio comportamiento. Las partículas tienen un ciclo de vida. Ellas se crean, crecen y mueren, además, pueden tener cambios en el color, localización y velocidad. Los sistemas de partículas se utilizan para modelar humo, nubes, olas, fuego, etc. y se clasifican en:

- Partículas que no interactúan entre sí: En estos sistemas, como el caso del humo y el fuego, las partículas conocen su posición pero no saben cuál es la posición de las partículas vecinas.

- Partículas que sí interactúan entre sí: Estas partículas sí conocen la posición de las vecinas y su movimiento depende de la posición de las demás.

Entonces, lo que hizo Craig Reynolds fue un sistema de partículas donde estas interactúan entre sí, en donde cada partícula en lugar de ser solamente puntos o figuras simples, son objetos que tienen una forma geométrica, un comportamiento y una orientación. La forma en como "vivían" estas partículas fue llamada "*Behaviors*".

Un primer software fue implementado en Flavors, un lenguaje orientado a objetos extensión de Lisp. El sistema fue llamado Stanley & Stella: In Breaking the Ice y consistía en el comportamiento de unas aves y unos peces. En aquellos años, es posible que Craig Reynolds no supiera el impacto que iba a causar su programa en el campo de la animación por computadora y de la Vida Artificial.

En una entrevista realizada por la revista electrónica Generation 5, Reynolds explicó el por qué de su interés en el tema y la manera en como se le ocurrió este algoritmo:
*"En la universidad estuve interesado en el concepto de "caracteres autónomos". Tenía la corazonada de que muchos de los comportamientos complejos podían derivarse de simples reglas y las parvadas de pájaros parecían un buen ejemplo de esto. Asumí que los pájaros no podían pensar mucho para moverse de esa forma, primero porque no son animales muy inteligentes y porque sus movimientos ocurren en tiempo real a una velocidad tan rápida que no hay tiempo para pensar profundamente. Me di cuenta de que debía ser una experiencia muy diferente ser miembro de una parvada que observarla desde afuera. Entonces intenté ponerme en el lugar de un miembro e imaginarme qué tendría que hacer para volar con los demás"[17].*

De esta forma, Reynolds desarrolló tres reglas sencillas para explicar el comportamiento de las parvadas de pájaros:

1. **Cohesión:** Esta regla se refiere a que los miembros de la parvada deben intentar mantenerse lo más cerca del centro posible. Esto se logra manteniéndose cerca de sus vecinos más próximos [18].

2. **Alineación:** Esta regla se refiere a que cada miembro de la parvada debe mantenerse en la misma dirección que el grupo completo [19].

3. **Separación:** Se refiere a evitar chocar con los otros miembros de la parvada y se logra conociendo la distancia que hay entre ellos [20].

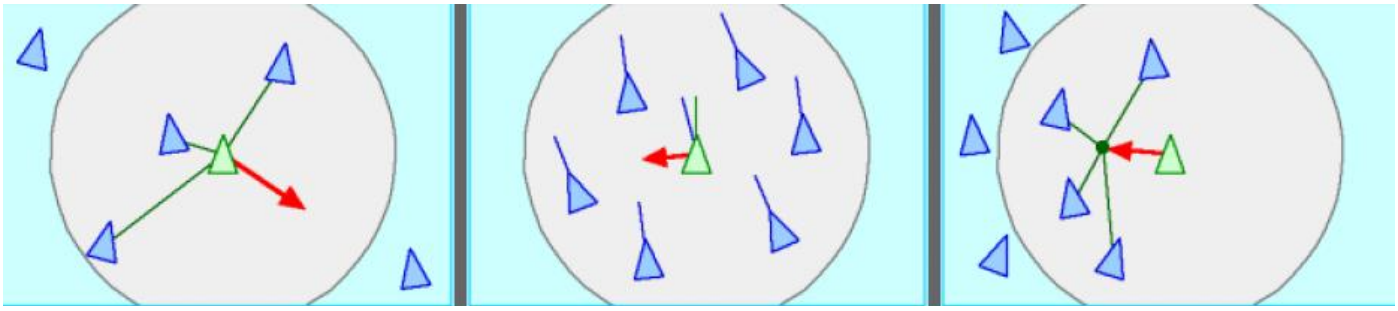

**Figura 3 Separación, Alineación y Cohesión**

Como se mencionó anteriormente, Craig Reynolds trataba de explicar el comportamiento de una parvada, que aunque a simple vista nos pareciera un movimiento caótico, con los *Behaviors* se demuestra que no lo es del todo.

Así que si se puede modelar una parvada, también es posible modelar un cardumen o banco de peces. El movimiento es muy similar al de las parvadas, ya que también se mueven en grupo. Una de las diferencias principales es la forma alargada de los peces y la fricción que ejerce el agua es distinta a la del aire.

En el caso de un grupo de animales terrestres, o sea, una manada. Estos, al igual que las aves y los peces, se agrupan y tienen un comportamiento similar. La diferencia principal entre estos y los anteriores es que las manadas tienen un movimiento limitado a dos dimensiones.

El algoritmo se extiende para permitir a los *Behaviors* los siguientes comportamientos:

- **Evitar obstáculos:** Al estarse moviendo los organismos deben ser capaces de evitar chocar contra los objetos que encuentren en su camino. De la misma forma en que evitan chocar con un compañero, los *Behaviors* pueden medir la distancia entre ellos y los obstáculos y de esta forma evitarlos [21].

- **Huir de un cazador:** Este comportamiento se realiza dándole la capacidad a los *Behaviors* de distinguir entre sus compañeros y el cazador y además cuidar la distancia entre ellos y el cazador. Así mismo, el cazador debe intentar acercarse lo más posible a los *Behaviors* [22].

- **Perseguir una presa:** Al igual que el comportamiento anterior, los *Behaviors* deben ser capaces de distinguir entre sus compañeros y una presa y de esta forma intentar que la distancia entre ellos sea mínima mientras la presa realiza lo contrario [23].

### 1.4.1 Aplicaciones

Este algoritmo ha sido aplicado en muchos campos. Uno de los principales es la animación por computadora. Craig Reynolds es reconocido internacionalmente por su trabajo con los *Behaviors* [24].

Otro campo en donde se ha aplicado este algoritmo es en la administración. Se ha creado toda una teoría en la cual se pueden aplicar los conceptos de conducta emergente siguiendo las reglas del movimiento de una parvada [25].

El objetivo de esto es obtener el mejor trabajo de la gente y ocasionar que se sientan orgullosos de trabajar en equipo para alcanzar un nivel que los haga confiables y les asegure un mejor futuro para ellos y para el equipo.

También, se han hecho estudios sobre tráfico, en donde los carros son los *Behaviors* y se aplica este algoritmo con algunas consideraciones como el hecho de que los carros deben conducirse por los carriles [26].

Sin duda, una de las aplicaciones más interesantes sobre este algoritmo es la utilización de algoritmos genéticos. Se han construido aplicaciones en las cuales se modela el algoritmo de Presa-Cazador y en donde los *Behaviors* son capaces de evolucionar para aumentar las probabilidades de su existencia. También hay algoritmos en donde toda la manada persigue a una sola presa. Se ha descubierto que con la evolución, los *Behaviors* son capaces de generar técnicas de caza para ser más efectivos [27].

Algunas otras aplicaciones que se le ha dado a este algoritmo son [28]:

- Para mejorar el realismo en juegos computacionales e industrias de ambiente virtual.
- Para publicidad, utilizando vehículos aéreos muy pequeños en eventos públicos.
- Fuegos artificiales inteligentes.
- Instantáneo monitoreo de información aérea en grandes volúmenes atmosféricos.
- Mapeo automatizado de terrenos en 3D con multi-sensores.
- Control espacial de vehículos aéreos sin tripulación.

### 1.4.2 *Steering Behaviors*

A continuación se hace una breve explicación sobre algunos *Behaviors*; los aquí expuestos son de los más sencillos y pueden definirse como las bases que sentarán los movimientos básicos que debe cumplir cualquier organismo que se desplace ya sea solo o en grupo, también son los que tomamos como cimiento para darle solución a este trabajo, pero no son ni de cerca los más complejos o completos que se puedan crear, ya que los *Behaviors* pueden ser tan variados como la situación a recrear así lo necesite, por lo que pueden considerarse como un recurso del que queda mucho por explotar.

### 1.4.3 *Seek and Flee behaviors*

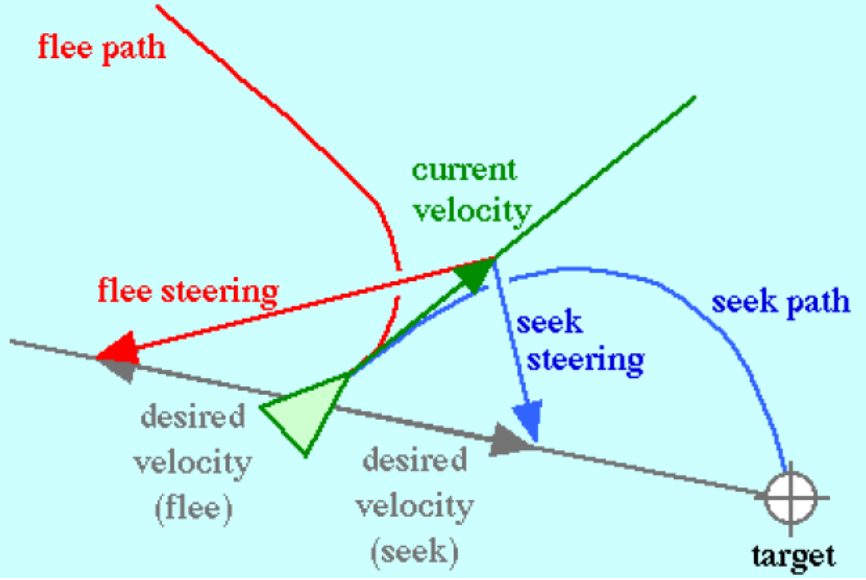


**Figura 4 *Behavior Seek* y *Flee***

Los *Behaviors* (comportamientos) *Flee* (huir) y *Seek* (buscar) implementan patrones de comportamiento complementarios. La lógica implícita en ambos comportamientos es la misma. La única diferencia es la fuerza de dirección resultante. *Seek* se usa para dirigir un vehículo hacia el objetivo especificado. Un ejemplo sería un predador dirigiéndose directamente hacia su alimento. El objetivo puede ser un punto fijo o en movimiento, ubicado en el espacio o en otro vehículo. El comportamiento *Flee* se utiliza para simular una sencilla forma de evasión. Como es el caso del *Seek*, la meta puede o no estar en movimiento [29].

La dirección de la fuerza se calcula sobre la base de los valores de la velocidad actual del vehículo y la posición del objetivo especificado. En primer lugar, la posición del objetivo se expresa como un vector entre el objetivo y el vehículo. Esto representa la velocidad deseada (*desired speed*). La dirección de la fuerza es el resultado de la diferencia entre el vector "*desired speed*" y el vector "velocidad actual".

En el comportamiento *Seek* la velocidad actual del vehículo se le resta a la velocidad deseada. Para obtener la dirección de la fuerza para el comportamiento *Flee*, la velocidad deseada se le resta a la velocidad actual. Con el uso de estos sencillos cálculos el vehículo puede ser conducido hacia su destino o puede ser obligado a evitarlo. Al igual que con todos los comportamientos, la fuerza calculada no debe superar la fuerza máxima del vehículo.

Ya que el comportamiento *Seek* genera una fuerza de dirección continuamente, en la mayoría de los casos pudiera suceder un efecto no deseado. Si el objetivo es estático o se mueve a una velocidad inferior a la del vehículo que realiza la búsqueda, la fuerza de dirección guiará al vehículo hacia la meta en primer lugar, pero este seguirá viajando en la misma dirección hasta que haya sobrepasado al objetivo. Ahora, una fuerza en sentido contrario es calculada y el vehículo se moverá hacia el camino desde donde proviene, dando como resultado un movimiento similar al de una mariposa bailando alrededor de una luz, lo que provocará que nunca llegue al objetivo o que no realice el comportamiento de la forma esperada [30].

### 1.4.4 *Arrive Behavior*

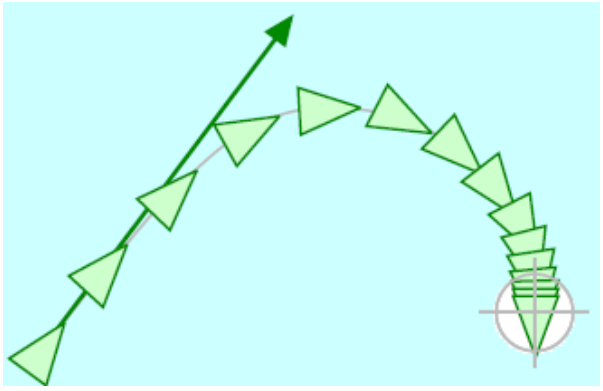

**Figura 5 Arrive *Behavior***

El comportamiento Arrive es una extensión del comportamiento Seek. Al igual que el comportamiento Seek se utiliza para dirigir el vehículo a un determinado objetivo. La diferencia importante es la forma en que llega a su destino. El comportamiento Seek se dirige hacia el destino a toda velocidad, sobrepasa su

destino y vuelve a por él a máxima velocidad y esto se repite hasta que logre acercársele del todo. Esto normalmente no es el tipo de resultado que se espera. A diferencia del Seek el comportamiento Arrive va disminuyendo progresivamente la velocidad del vehículo de forma controlada (de acuerdo a la especificación hecha por el usuario) hasta detenerse en la posición deseada [31].

El cálculo de la fuerza de dirección se realiza de la misma manera que en el comportamiento Seek. El vector resultante es la diferencia entre la velocidad deseada y la velocidad actual.

Hasta ahora no hay ninguna diferencia entre los dos comportamientos. Mientras que la meta esté fuera de la distancia de activación especificada, no se generará ninguna fuerza de dirección. Dentro de esta área la fuerza se modifica por la distancia hasta el objetivo dividido por el número de pasos deseados. Esto hace que el vehículo ralentice su movimiento de acuerdo a la distancia actual del destino. El resultado es un vehículo que se detiene en el destino especificado.

Hay un posible problema con este algoritmo. Al utilizar la distancia dividida por el número de pasos necesarios para alcanzar la meta, el vehículo se acerca a su destino sólo asintóticamente. Normalmente nunca llega exactamente a la posición de destino. Pero desde el punto de vista de los usuarios no hay ninguna diferencia visible si la posición exacta es alcanzada, o si el vehículo se desplaza a una velocidad cada vez menor hasta el punto de destino. En un cierto punto en los cálculos, el vehículo puede, de repente, hacer cambios en la orientación, incluso si se ve detenido todavía por un tiempo. Una posible solución sería que cese la actualización de la orientación del vector de dirección si la fuerza está por debajo de un determinado valor. Una segunda posible solución, utiliza los tres vectores de orientación a la media de orientación en los últimos fotogramas de la animación. Esto suprime los cambios rápidos de dirección y el titilado [32].

### 1.4.5 *Wander Behavior*

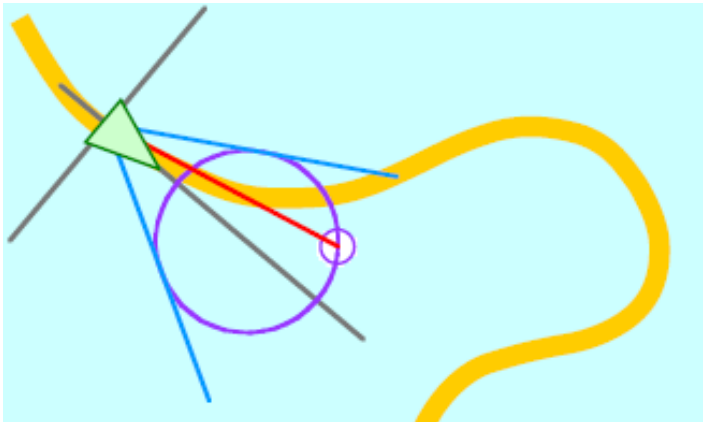

**Figura 6 *Wander Behavior***

El comportamiento Wander permite que los vehículos se muevan de forma autónoma y sin un determinado objetivo a través de la escena. Es comparable al movimiento de las hormigas en busca de alimentos. La mayoría de los animales cuando se encuentran en busca de algo que no se encuentra en sus alrededores, caminan de forma aleatoria e indefinida y ese es el propósito de este comportamiento [33].

Los movimientos en las líneas rectas como se ha visto en otros comportamientos, como, por ejemplo, Seek o Arrive, pueden estar ligeramente distorsionados si se usan conjuntamente con el comportamiento Wander. Las influencias externas como el viento o las características del terreno como se ha visto en la realidad se pueden simular de esta manera.

Básicamente el comportamiento Wander se realiza utilizando dirección de fuerzas aleatorias. Para crearlas hay varias posibilidades. La forma más sencilla sería la de añadir un vector al azar a la velocidad para cada cuadro de la animación. El movimiento resultante puede que no tenga un aspecto muy realista, ya que la dirección de fuerza va a cambiar de dirección casi instantáneamente. Por lo tanto, un algoritmo que impida que esto suceda debe ser utilizado [34].

Al igual que todos los otros comportamientos, el Wander debe calcular un vector fuerza resultante. En este caso el vector sólo tiene un cierto grado de libertad al restringir la punta del mismo a un círculo colocado en la parte delantera del vehículo. Este vector se transforma en coordenadas del mundo y se utiliza como la dirección de fuerza de este comportamiento.

Para cada iteración de la simulación un vector al azar es añadido a la actual dirección de fuerza. La longitud de este vector está limitada a un valor específico. Este atributo, por lo tanto, controla la diferencia máxima en la dirección de fuerza. El vector resultante de la adición del vector aleatorio a la anterior dirección del vector es de nuevo obligado a restringirse en el círculo antes mencionado.

El comportamiento se puede ajustar a las necesidades de la simulación mediante el uso de diferentes valores para el radio del círculo, la posición del círculo y el máximo cambio de dirección. Si el círculo se coloca cerca del vehículo, los cambios en la dirección del movimiento serán más rápidos. Si se coloca a cierta distancia, el movimiento resultante será más lineal.

### 1.4.6 *Separation Behavior*

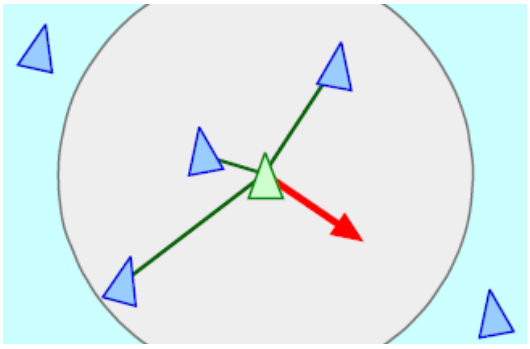

**Figura 7 *Separation Behavior***

El comportamiento Separation se utiliza para obtener que los vehículos mantengan una cierta distancia entre sí. Esto ayuda a prevenir que colisionen entre ellos. Este es un comportamiento inherente a todas las criaturas vivientes. Si se está paseando en un lugar donde haya mucha gente, siempre se trata de evitar golpear a otras personas. Sólo las personas que se encuentran a corta distancia son tomadas en

cuenta a la hora de realizar esta acción, generalmente, nos fijamos en los caminantes que se encuentran por delante.

Las bases aplicadas para la implementación de este comportamiento son las teorías de partículas con carga eléctrica que evitan el hacinamiento rechazándose las unas a las otras. Lo primero que hay que hacer es encontrar a todos los vehículos que se encuentren a cierta distancia de nuestro vehículo. Esto se hace utilizando el objeto de simulación “Vecindario”. De acuerdo a la distancia entre los dos objetos, la fuerza será más fuerte mientras más pequeña sea la distancia. Todas las fuerzas calculadas se suman y forma la dirección de fuerza [35].

Como también son considerados los vehículos que vengan detrás del vehículo que aplica este comportamiento, en algunos casos, se pueden observar movimientos extraños; una persona que conduzca un automóvil sólo considerará a los vehículos que se encuentren delante de él. Para simular esto, se verificará que solo existan vehículos en el frente del vehículo. Esto hace que los vehículos que se encuentren detrás solo sean tomados en cuenta en caso de una aproximación muy cercana, casi en el momento de colisionar [36].

### 1.4.7 *Obstacle Avoidance Behavior*

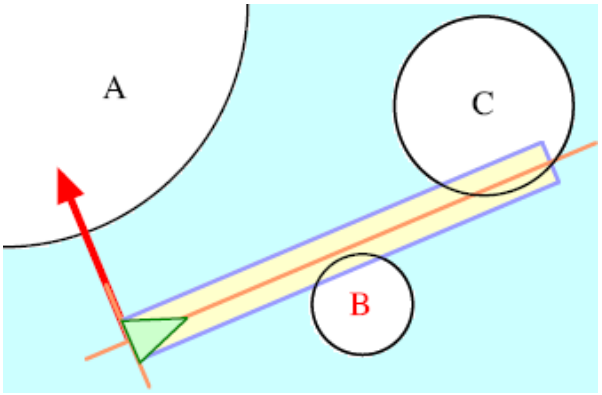


**Figura 8 *Obstacle Avoidance Behavior***

Para dirigir un vehículo a través de un paisaje predefinido de forma realista, es necesario moverse de la misma manera que el usuario espera. Por lo tanto, es necesario definir un comportamiento que permite la manipulación de los obstáculos.

En la vida cotidiana los seres humanos y los animales evitan chocar con los obstáculos. Cuando alguien siente el ansia de una cerveza fría, no se dirige hacia la nevera en línea recta, inconscientemente elige un camino que lo llevará alrededor de todos los obstáculos que se interpongan entre él y la cerveza, como mesas, sillas y otras cosas que conformen el entorno donde se encuentre.

El comportamiento Obstacle Avoidance implementa este comportamiento inconsciente de evitar obstáculos predefinidos. El primer paso es conocer que objetos se encuentran en las inmediaciones del vehículo, después se debe calcular el ángulo de intersección y la distancia que separa al vehículo del obstáculo [37].

El vector de dirección se basa en los valores resultantes. Para lograr esto, el vector que es sometido a prueba es primeramente ampliado por la distancia hasta la intersección y después es proyectado verticalmente hacia atrás hasta el borde del vehículo, este vector se utiliza para la dirección de fuerza.

### 1.4.8 Combinando *Behaviors*

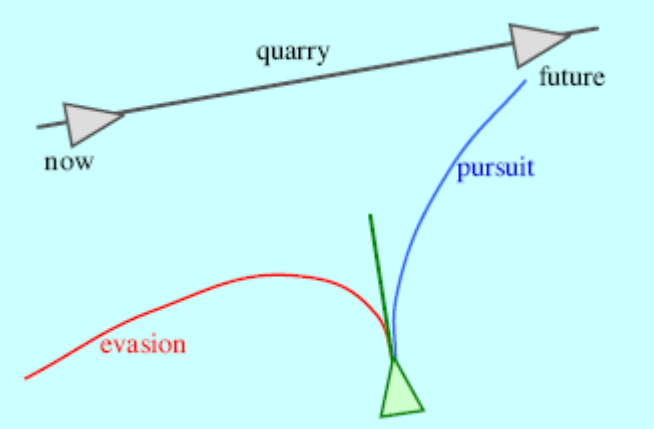


**Figura 9 Combinación de los *Behavior Evasion* y *Pursuit***

Los *Steering Behavior*s descritos anteriormente sirven para la construcción de *Behaviors* más complejos, mediante la combinación de los mismos. La combinación de *Steering Behaviors* puede usarse para implementar resultados de un nivel más alto (Ej.: muévete libremente mientras evitas obstáculos, sigue este camino y únete al grupo de caracteres). A menos que un carácter autónomo forme parte de un mundo muy simple, sería muy extraño que tuviera solo un comportamiento

La combinación de *Steering Behaviors* puede ocurrir de dos formas:

1. Un personaje puede cambiar de forma secuencial entre los modos de comportamiento como las circunstancias cambian en su mundo. Por ejemplo, imagine un rebaño de cabras pastando en un prado cuando de repente sienten que se aproximan lobos. Este evento da lugar a un cambio de comportamiento. Todos los pensamientos (que en este caso sería el comportamiento (*Behavior*)) referentes al pastoreo son olvidados y el rebaño se dispone a huir (*Behavior Flee*) de los depredadores. No hay tendencia a la combinación de estos *Behaviors*: una cabra no ralentizará su carrera durante la huida de un lobo con el fin de apoderarse de otro bocado de comida.

2. Por otra parte, algunos tipos de *Behaviors* son mezclados, actuando efectivamente en paralelo. Por ejemplo, al huir por el bosque, las cabras, tienen que combinar los *Behaviors Evasion* y *Obstacle Avoidance*, esta combinación les permitirá huir de los lobos y esquivar los árboles y a sus propios compañeros. En este caso, las cabras no pueden darse el lujo de ignorar un *Behavior*, ya que solo la unión de estos podrá evitar que sean atrapadas por los lobos y que choquen con algún obstáculo

La mezcla *Behaviors* se puede lograr de varias maneras. La más sencilla es simplemente para calcular cada uno de los componentes del *Behavior* y sumarlos luego todos juntos, posiblemente con un factor de ponderación para cada uno de ellos. (Tenga en cuenta que los vectores de dirección son especialmente fáciles de combinar, otros tipos de comportamientos, que producen otros tipos de valores (por ejemplo, *Behaviors* de conversación) podrían ser mucho más difíciles de combinar.) Esta sencilla combinación lineal a menudo funciona bien, pero tiene al menos dos deficiencias: no es la más eficiente

computacionalmente, y a pesar de ajustar los pesos, los *Behaviors* combinados se pueden cancelar los unos a los otros en momentos inoportunos.

### 1.5 Agentes

El término 'agente' es muy controvertido, son varios los autores que han dado su propia interpretación del término 'agentes', siendo en la mayoría de los casos, el contexto de la situación el que defina lo que el término significa; en la actualidad no hay un criterio bien definido. Un primer pronunciamiento de agente en el año 1996 nos dice: "Un agente como una entidad que percibe y actúa sobre un entorno" (Russell, 1996), a pesar de ser muy sencillo encierra la esencia del tema.

Según diccionarios de la lengua española, en su primera acepción, un agente es una *"Persona que trabaja en una agencia prestando determinados servicios"*. Llevando esta definición al mundo de la computación y viéndolo de manera genérica se puede sustituir el término 'persona' por 'entidad', la frase 'trabaja en una agencia' por 'actúa en un entorno' y la frase 'prestando determinados servicios' por 'transformando dicho entorno'; elaborando un poco el nuevo concepto se puede concluir que un agente es una *entidad que actúa 'de manera autónoma' en un entorno, transformándolo mediante la interrelación con otras entidades. [38]*

Son varias las características atribuidas a los agentes, sin embargo hay tres principales en las que se ha llegado a consenso, que son indispensables para considerar que estamos en presencia de un agente.

***Reactivo:*** El agente debe ser capaz de responder a cambios en el entorno en que se encuentra situado. Actúa como resultado de esos cambios.

***Pro-activo:*** El agente debe ser capaz de intentar cumplir sus propios planes u objetivos. Debe ser capaz de controlar sus propios objetivos a pesar de cambios en el entorno. Cumplir con su tarea a toda costa.

***Social:*** El agente debe comunicarse con otros agentes mediante algún tipo de comunicación.

Estas características deben ser definitorias, son los axiomas principales, en los elementos del entorno para saber si el sistema está elaborado mediante el paradigma orientado a agentes, lo cual no quiere decir que todos los elementos cumplan con estas características. Como se decía anteriormente este paradigma puede aparecer mezclado con otros como sucede en otros casos, el ejemplo más común es la mezcla existente entre el estructurado y el orientado a objetos. [39]

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 (Franklin, 1996), algunas de estas características son:

***Continuidad Temporal:*** Se considera un agente un proceso sin fin, ejecutándose continuamente y desarrollando su función.
***Autonomía:*** Un agente es autónomo si es capaz de actuar basándose en su experiencia, siendo capaz de adaptarse a cambios en el entorno.
***Racionalidad:*** El agente siempre realiza «lo correcto» a partir de los datos que percibe del entorno.
***Adaptatividad:*** El agente es capaz de aprender y de alguna manera cambiar su comportamiento basándose en ese aprendizaje.
***Movilidad:*** Es la capacidad de un agente de trasladarse a través de una red (Internet, intranet u otro medio).
***Veracidad:*** Un agente no comunica información falsa a propósito.
***Benevolencia:*** Un agente está dispuesto a ayudar a otros agentes si esto no entra en conflicto con sus propios objetivos.

Los principales 'tipos' de agentes que hasta hoy se han trabajado son:

***Agentes de interfaz:*** Ayudan a enseñar determinadas materias en un entorno virtual, o a dirigir el trabajo en un software.
***Agentes móviles:*** Se mueven a través de un entorno, para esto deben estar soportados sobre alguna plataforma común, la Web fundamentalmente, cumpliendo algún rol específico.

***Agentes de Internet o de información:*** Se encargan fundamentalmente de seleccionar textos interesantes o que le sean solicitados por los usuarios, sobre temáticas, de sitios específicos.
***Agentes robóticos:*** Incorporan todas estas estrategias a objetos reales, su explotación lleva muchos recursos. Se han logrado resultados interesantes en países altamente desarrollados.
***Agentes creíbles:*** (virtuales). Simulan el comportamiento de humanos, animales u otros objetos en un entorno virtual. 40]

### 1.5.1 Aplicaciones

Un agente, tal como se ha definido anteriormente, puede ser usado de múltiples maneras según el entorno. Algunas de las aplicaciones son:

**Ayuda al cliente**: Un agente que ayude al cliente, además de ser cacofónico, podría “escuchar” el problema, buscar en varios medios diferentes y finalmente dar una respuesta al mismo. Este agente puede residir tanto en la red como en el ordenador del usuario, y además, puede "conocer" los fallos más habituales, tanto en general como del usuario en particular para tratar de proponer soluciones a los mismos antes de que sucedan.

**World Wide Web**: Estos agentes aprenderían de los hábitos de búsqueda del usuario, mirando las páginas que se visitan, la frecuencia con que se hace y el tiempo que se gasta en cada una, para proponer nuevas páginas y avisar cuando se ha creado una página interesante.

**Busca gangas**: Utilizados en entornos donde se puede comprar directamente de los productores, para comprar tickets de viajes en avión o en un entorno de telecomunicaciones no reguladas, un agente podría buscar el objeto que desee su usuario al mejor precio, comprar un billete de avión en las mejores condiciones o enrutar una llamada telefónica usando los portadores con mejor precio o calidad. Actualmente hay muchos busca-gangas en la Internet, aunque no se podrían calificar como agentes.

**Agentes charlatanes**: En un mundo estigmatizado por la soledad de las personas, y donde el ocio se convierte en la industria número uno, un agente charlatán puede dar conversación a su usuario, tomando la apariencia deseada o jugar a un juego con él o ella.

**Trabajo en grupo**: Quien haya tratado de convocar una reunión "que le venga bien a todo el mundo" sabe que es un proceso de negociación con restricciones duras, otras menos duras, en muchos casos, largo y de difícil solución. Los agentes se están usando ya para negociar este y otros problemas de trabajo y coordinación de un grupo.

**Autónomos**: Son los agentes que se mueven en mundos artificiales, estos agentes sienten su entorno, actúan sobre él, cambian de estado interno, aprenden y evolucionan, persiguiendo sus propios objetivos de forma que afecte lo que se sentirá en el futuro, el afectar lo que se percibirá en el futuro implica inteligencia, al menos en el sentido de que se aprende de los fallos para no volver a cometerlos en el futuro, lo cual implica adaptabilidad, y además implica un bucle percepción/procesamiento de información/acción, para volver otra vez al principio, es decir, un comportamiento similar al de cualquier ser vivo medianamente inteligente; sin embargo, estos agentes no tienen porqué hacer nada útil, sino simplemente simular algún aspecto del mundo siendo el entorno el que les proveerá las herramientas necesarias que permitirán que realicen la función para la cual fueron creados.

El que tengan percepción y acción implica que exista comunicación, la cual puede realizarse con programas no-agentes o con otros agentes, lo cual puede dar lugar a sociedades de agentes.

Un agente autónomo, se sale de la definición clásica término ‘programa’; es decir, no se puede llamar ‘programa’ en el mismo sentido que un sistema operativo no se puede llamar ‘programa’. Una de las características que los distinguen de los programas es la autonomía, lo cual implica dos cosas:

1. **Son pro-activos**: Que no sólo actúan respondiendo a una acción del usuario, sino que también actúan siguiendo sus propios objetivos.

2. **Son persistentes**: Que no se pueden "apagar"; incluso aunque el usuario no esté interactuando con ellos, los agentes siguen funcionando, recolectando información, aprendiendo y comunicándose con otros agentes.

# CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

## Introducción

En este capitulo se presenta una visión más clara del sistema que se va a desarrollar; se podrá encontrar la descripción de la propuesta de solución, todo lo relacionado con los *Behaviors* que se crearon para darle solución al proyecto, así como también las metodologías y herramientas empleadas, además se muestra una panorámica desde el punto de vista del dominio donde se definen las características y las funcionalidades que tiene que cumplir el sistema para poder cumplir los objetivos trazados.

## 2.1 Descripción de la propuesta de solución

Hasta el momento se han definido los principales conceptos de los algoritmos de IA y de los Agentes Autónomos, pero solo eso, ya que aún no se ha dado una visión clara del camino que pretende seguir el trabajo, definir ese camino es el objetivo de este epígrafe.

Teniendo en cuenta todo lo relacionado con la Inteligencia y la Vida Artificial, las técnicas que las conforman, sus características y sus diferentes campos de aplicación, decidimos que para darle solución al problema utilizaríamos la técnica de los *Steering Behaviors*, por las funcionalidades que brindan cuando de simulación de comportamientos se trata; ya en los ejemplos de *Behaviors*, que aparecen cuando describimos esta técnica, se pueden observar algunos comportamientos que son de uso obligatorio cuando se quiere simular el movimiento tanto de animales, vehículos o personas, pueden tomarse por ejemplo los *Behaviors Seek* o *Flee*, ya que la creación de los mismos está basada en la observación de lo que ocurre en la vida real; para solucionar el problema científico crearemos nuestros propios *Behaviors*, de los que ya hablaremos más adelante.

Se asignarán los *Behaviors* a los autos del entorno virtual según la situación en la que estos se encuentren, con lo que lograremos que dichos autos posean “cierta” autonomía; cierto es que, en nuestro proyecto estos autos no serán agentes autónomos del todo, ya que, por ejemplo ellos no seleccionarán el camino por el cual van a transitar (lo tendrán definido) y por ende no van a aprender o a evolucionar; ellos solamente simularán el movimiento que realizan en una ciudad los autos que se mueven por la misma,

aplicando la inteligencia que le proporcionarán los *Behaviors* creados, demostrando así un grado de autonomía, ya que su actitud ante una situación dada va a modificarse.

### 2.2 *Behaviors* creados

Teniendo en cuenta los objetivos del proyecto, los Casos de Uso planteados y los requisitos funcionales que se definieron, se llegó a la conclusión de que para darle solución al proyecto debíamos crear *behaviors* que se acoplaran a las situaciones propias del entorno; hay que destacar que lo *behaviors* creados tienen cierta "diferencia" con los *behaviors* antes expuestos en el capítulo 1, la diferencia reside en que los *behaviors* mencionados en el capítulo 1 trabajan con un vector 'fuerza' y a partir de este calculan la velocidad con la que se desplazará el ente inteligente, mientras que los *behaviors* creados trabajan directamente con la velocidad, la aceleración y la distancia; esta diferencia se debe principalmente a que el motor gráfico seleccionado trabaja con la velocidad para aplicarle movimientos a los objetos y los mismos se mueven a través de un camino ya predefinido.

A continuación se brinda una explicación en la que se describe todo lo referente a los *behaviors* que conforman la propuesta de solución de nuestro trabajo.

#### 2.2.1 *Behavior* Movimiento Libre

Este *behavior* provee a los vehículos de la escena el movimiento básico con el que se desplazarán, consiste en que el auto va a moverse siguiendo una ruta predefinida, a una velocidad específica (nunca mayor que 80 Km/h, que es la velocidad máxima permitida dentro de una ciudad) mientras no exista algún factor (señales del tránsito u otros vehículos) que obliguen al vehículo modificar su comportamiento.

#### 2.2.2 *Behavior* Seguimiento

Si se tienen dos autos que se mueven por una misma calle y esta es de una sola senda, en el caso de que el vehículo que va delante tenga una velocidad menor que la del vehículo que va detrás, en algún momento los dos autos van a encontrarse, a no ser que a una distancia específica el segundo auto disminuya su velocidad hasta que iguale la del primer vehículo y así mantener una separación entre ellos, previendo así que colisionen; esa es precisamente la función del *behavior* Seguimiento, realizar ciertas

operaciones para que el auto que se encuentra a la espalda de otro no colisione con el mismo sino que disminuya paulatinamente su velocidad hasta igualar la del auto que va delante.

### 2.2.3 *Behavior* Intersección

Este *behavior* será el que se encargará de guiar a los vehículos cuando se encuentren frente a una intersección (un cruce de vías); para la realización del mismo se tuvo en cuenta la siguiente condición:

1. Si la calle que se encuentra perpendicular al vehículo es de un solo sentido o de dos.

Cuando el vehículo entra a la intersección chequea si hay alguna señal del tránsito (Semáforos, Ceda el Paso o Pare) que esté rigiendo el nivel de prioridad de los autos en el cruce, en caso de que esta exista analiza su tipo; si lo que hay es un Semáforo chequea el color del mismo (cruzará con la luz verde), en el caso de un Ceda el Paso o un Pare verificará si la calle que se encuentra perpendicular a él es de un solo sentido o de dos, una vez hecho esto verifica que en los puntos cercanos a las esquinas no se encuentren vehículos en ese instante y es entonces que cruzará. En el caso que no exista una señal, se tomará que el vehículo posee la prioridad para cruzar y por lo tanto no se analizarán ninguna de las condiciones antes mencionadas.

### 2.2.4 *Behavior* Paso Peatonal

Este *behavior* será el que se encargará de guiar a los vehículos cuando se encuentren frente a un Paso Peatonal. Cuando el vehículo entra en la zona del Paso Peatonal, se detiene, acto seguido verifica si en la zona definida para el Paso Peatonal se encuentra algún Peatón transitando, en caso de que exista, esperará hasta que no haya nadie y continuará su movimiento. Actualmente en el entorno virtual no hay personas moviéndose, por tanto lo que está implementado es que el vehículo cuando llegue a un paso peatonal se detenga, espere cinco segundo y continúe su movimiento.

## 2.3 Metodologías y herramientas a utilizar para llevar a cabo la propuesta de solución

### 2.3.1 Visual Studio 2003

El Visual Studio 2003, es una herramienta que ofrece un entorno de desarrollo completo e integrado para la construcción, depuración y despliegue de aplicaciones, que incluye la integración de varios lenguajes de

programación (C++ entre ellos) y de una interfaz amigable al usuario que permite encontrar e integrar los componentes necesarios de forma muy rápida. Posee una excelente detección y corrección de errores. Permite a los desarrolladores crear rápidamente aplicaciones de alto rendimiento, o lo que es lo mismo, desarrollar aplicaciones para soluciones integrales. Trae incluido una serie de librerías donde aparecen muchos códigos pre-compilados que ayudan a que la creación de cualquier producto sea más sencilla y rápida.

### 2.3.2 Visual Paradigm 3.1

Visual Paradigm para UML es una herramienta profesional fácil de utilizar, que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Ayuda a una más rápida construcción de aplicaciones de calidad a un menor coste. Permite dibujar todos los tipos de diagramas de clases, permite la realización de ingeniería tanto directa como inversa, generar el código desde diagramas y generar la documentación automáticamente en varios formatos como Web o Pdf, permite el control de versiones. Además, la herramienta es colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto. Proporciona también abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Además es robusta y portable.

### 2.3.3 Scene ToolKit 2.3

**SceneToolKit** es una herramienta de apoyo a los programadores de aplicaciones finales, que se encuentra acorde con la tendencia mundial de desarrollo de los llamados "*engines*" o motores gráficos, es una de las herramientas brindadas por el Proyecto de Herramientas de Desarrollo para Sistemas de Realidad Virtual, tiene como objetivo básico agrupar las funcionalidades comunes a cualquier sistema de realidad virtual, de manera que se les facilite el trabajo a los programadores de juegos y simuladores a través de la reutilización de código.

Esta herramienta, en desarrollo actualmente, permite no solamente la visualización de los entornos sintéticos sino además la aplicación de leyes físicas y matemáticas, animaciones, etc., usando las librerías

gráficas OpenGL y DirectX, sobre plataforma Windows y Linux. A la vez, se retroalimenta con las necesidades de los programadores que las utilicen, así como de sus investigaciones.

### 2.3.4 UML

El **Lenguaje Unificado de Modelado** es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico, ya que cuenta con un grupo de diagramas, los cuales muestran diferentes aspectos de las entidades representadas y que se utilizan para visualizar, especificar, construir y documentar un sistema de software, en cada una de las etapas por las que tiene que pasar el mismo.

Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

### 2.3.5 RUP

El **Proceso Unificado Racional** (*Rational Unified Process* en inglés, habitualmente resumido como **RUP**) es un proceso para el desarrollo de un proyecto de software que define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Sus características esenciales son:

1. **Está dirigido por los Casos de Uso:** que orientan el proyecto a la importancia para el usuario y lo que este quiere.
2. **Está centrado en la arquitectura:** que Relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden.
3. **Es iterativo e incremental:** donde divide el proyecto en mini proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada

Además, incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso). Así se puede determinar exactamente qué se está realizando en cada Fase del proceso de desarrollo y quién lo está realizando.

### 2.3.6 Lenguaje C++

C++ es la evolución de C adaptada a la programación orientada a objetos. Se encuentra entre los más usados para desarrollar aplicaciones gráficas en 3D, por ser los que con más velocidad ejecutan el código, en general puede llegar a ser un lenguaje tan rápido como C (el más rápido después del Lenguaje Ensamblador), además tiene algunas cuestiones más pulidas como un control más estricto en el manejo de tipos de datos, y otras características que ayudan a la programación libre de errores.

Entre las principales ventajas del C++ se encuentran:

1. C++ es un lenguaje de propósito general, o sea, que con él se puede programar cualquier cosa, desde sistemas operativos y compiladores hasta aplicaciones de bases de datos y procesadores de texto, pasando por juegos, etc.
2. Los programas escritos en C++ tienen la ventaja de que podrán ejecutarse en cualquier máquina y bajo cualquier sistema operativo.
3. Es un lenguaje multi-nivel, es decir, se puede usar para programar directamente el hardware o para crear aplicaciones tipo Windows.

## 2.4 MODELO DE DOMINIO

Se propone la realización de un modelo de dominio porque los procesos de negocio que dan surgimiento al problema científico no son palpables en toda su extensión, o sea, no son visibles del todo porque no se

encuentran bien definidos y sus fronteras no están correctamente establecidas, además, el trabajo no es de una gran complejidad estructural, no vamos a desarrollar un producto de software de gran magnitud que implique la realización de un Modelo de Negocio, que es más extenso y más profundo que un Modelo de Dominio.

Con la realización del mismo permitiremos que se tenga una visión clara de todos los procesos que formaron parte del desarrollo del producto.

### 2.4.1 Reglas del Negocio

El programador que utilice el módulo debe tener conocimientos básicos de programación.

El programador que utilice el módulo debe tener experiencia en el trabajo con el motor de visualización.

### 2.4.2 Clases Conceptuales

Se le denominará ***usuario*** al sistema que utilice o interactúe con el módulo. Estos sistemas pueden ser los de un juego, un simulador o cualquier aplicación a la que sea incorporado el módulo.

Se le denominará ***escena*** a la visualización que se mostrará como resultado de las acciones del usuario.

Se le denominará ***vehículo*** a la entidad que va a representar un vehículo. Estas entidades estarán definidas por un comportamiento (*behavior*) en cada momento.

Se le denominará ***entorno*** al paisaje que se muestra en la escena.

Se le denominará ***behavior*** al conjunto de restricciones y condiciones del sistema que definirán las acciones validas en los vehículos.

Se le denominará ***señal*** a la entidad que va a representar una señal del tránsito.

### 2.4.3 Modelo de Objetos

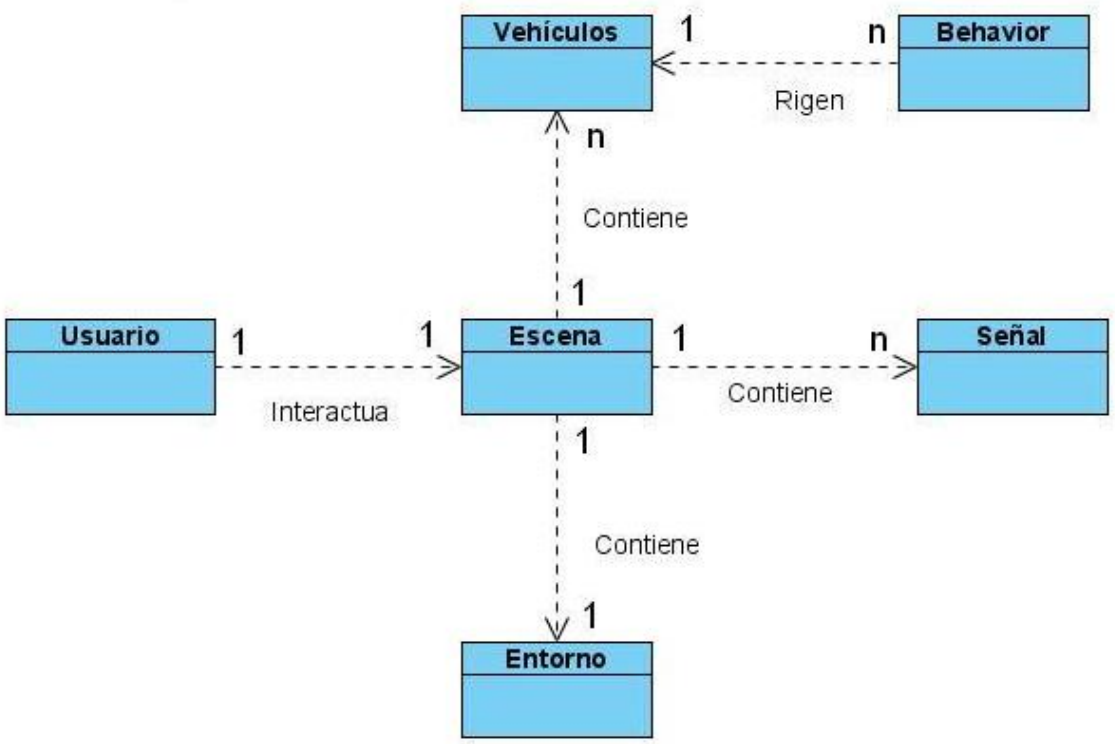


**Figura 10 Modelo de Objetos**

## 2.4 REQUERIMIENTOS

### 2.4.1 CAPTURA DE REQUISITOS

El propósito fundamental de la captura de requisitos es guiar el desarrollo del software hacia la obtención del producto esperado, ya que es aquí donde se describen los requerimientos del sistema, entiéndase, las condiciones o capacidades que el sistema debe cumplir.

### 2.4.2 REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales son capacidades, condiciones y acciones que el sistema debe cumplir, por lo que es de vital importancia que su definición sea lo más clara posible.

A continuación se muestran los requerimientos funcionales definidos para nuestro proyecto de tesis; los mismos se presentan asociados a los Casos de Uso del sistema:

| **CU 1** | **Movimiento Libre** |
|---|---|
| **RF 1** | Cumplir los Límites de Velocidad |
| El vehículo debe respetar la velocidad máxima establecida para el entorno en el que se esté moviendo. | |
| **CU 2** | **Seguimiento** |
| **RF 2** | Respetar la Distancia entre dos vehículos |
| Al mantenerse siguiendo a otro vehículo, debe respetar la distancia establecida para esta maniobra. | |
| **CU 3** | **Intersección** |
| **RF 3** | Respetar las Señalizaciones de Pare, Ceda el Paso y Semáforo |
| Al encontrarse frente a una intersección el vehículo tendrá en cuenta las señales de Pare, Ceda el Paso y Semáforo. | |
| **CU 4** | **Paso Peatonal** |
| **RF 4** | Respetar la Señalización de Paso Peatonal |
| El vehículo debe respetar esta señal y realizar las maniobras que denoten un buen comportamiento en la vía. | |

**Tabla 1Requisitos Funcionales**

### 2.4.3 REQUERIMIENTOS NO FUNCIONALES

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Estos están vinculados a los requerimientos funcionales, ya que una vez se conoce lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser.

A continuación se presentan los requerimientos no funcionales definidos para la aplicación a desarrollar.

- **Usabilidad:** Los futuros usuarios del sistema serán programadores con conocimientos básicos de C++ y del Scene ToolKit.
- **Rendimiento**: Debe ser una aplicación que se ejecute en tiempo real, por lo que debe realizar todas las operaciones a una alta velocidad.
- **Portabilidad**: Debe ser compatible con la plataforma Windows y Linux.

## 2.5 Modelo de Casos de Uso

### 2.5.1 Determinación y justificación de los actores del sistema

| Actor | Justificación |
|---|---|
| Sistema | Sistema capaz de crear y asignar (según sea conveniente) los comportamientos, de manera que se pueda observar como la aplicación de los mismos influyen en la escena. |

**Tabla 2 Definición de actores**

### 2.5.2 Diagrama de casos de uso

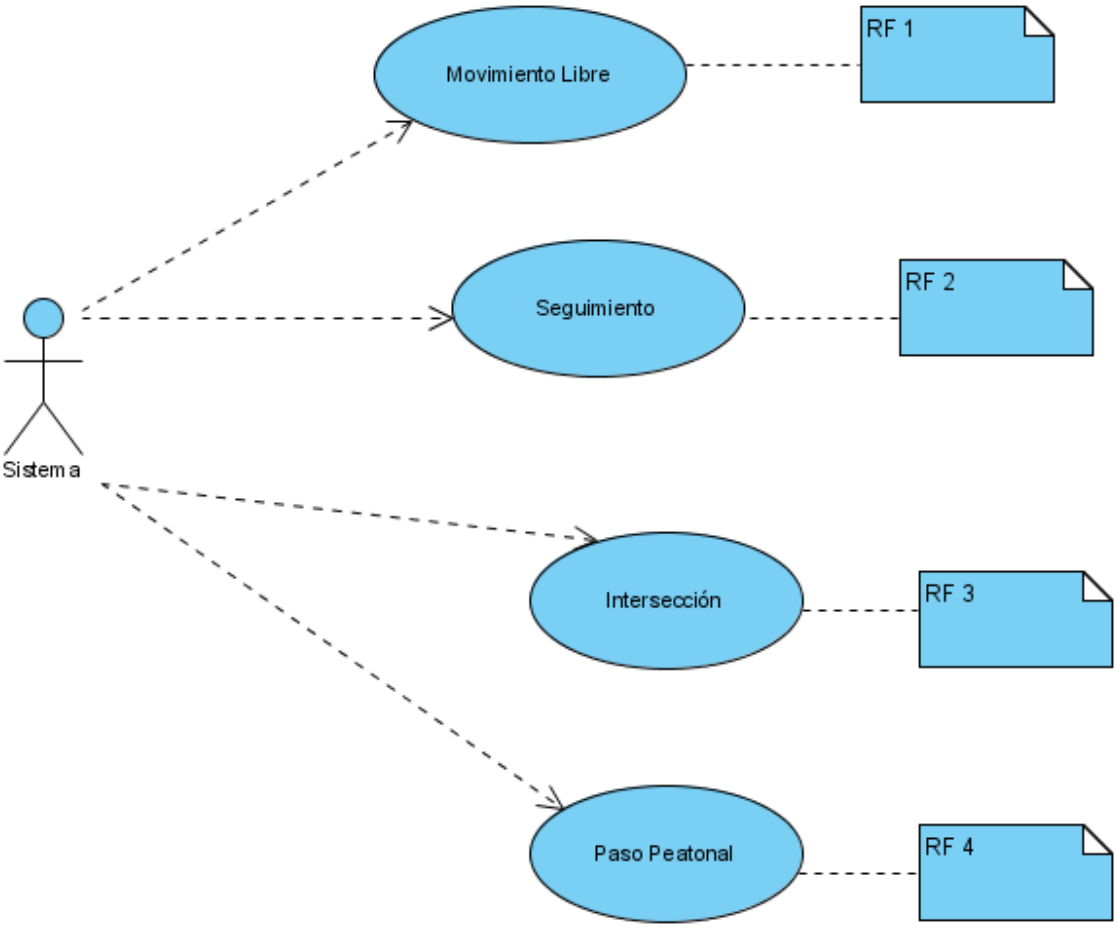


**Figura 11 Diagrama** de **Casos de Uso**

### 2.5.3 Descripción de los CU en formato expandido

| **Nombre del caso de Uso** | Movimiento Libre |
|---|---|
| **Actores** | Sistema |
| **Propósito** | Permitir el movimiento libre de los vehículos de la escena. |
| **Resumen** | El caso de uso se inicia cuando el sistema decide asignar este comportamiento a un(os) vehículo(s) perteneciente(s) a la escena. |
| **Precondiciones** | Se está ejecutando la aplicación. |
| **Poscondiciones** | Se halla asignado el comportamiento |
| **Curso Normal de los eventos** | |
| **Acciones del Actor** | **Respuesta del Sistema** |
| 1 El sistema decide dadas las condiciones del ambiente asignar este comportamiento | 1.1 El sistema genera a partir de las clases definidas el comportamiento solicitado y lo asigna. |
| **Curso Alternativo de los eventos** | |
| **Acción X** | |
| **Prototipo de Interfaz de usuario** | |
| **Prioridad** | Crítico |

**Tabla 3 Descripción del CU Movimiento Libre en formato expandido**

| **Nombre del caso de Uso** | Seguimiento |
|---|---|
| **Actores** | Sistema |
| **Propósito** | Permitir que un vehículo pueda seguir a otro. |
| **Resumen** | El caso de uso se inicia cuando el sistema decide asignar este comportamiento a un(os) vehículo(s) perteneciente(s) a la escena. |
| **Precondiciones** | Se está ejecutando la aplicación. |
| **Poscondiciones** | Se halla asignado el comportamiento |
| **Curso Normal de los eventos** | |
| **Acciones del Actor** | **Respuesta del Sistema** |
| 1 El sistema decide dadas las condiciones del ambiente asignar este comportamiento | 1.1 El sistema genera a partir de las clases definidas el comportamiento solicitado y lo asigna. |
| **Curso Alternativo de los eventos** | |
| **Acción X** | |
| **Prototipo de Interfaz de usuario** | |
| **Prioridad** | Crítico |

**Tabla 4 Descripción del CU Seguimiento en formato expandido.**

| **Nombre del caso de Uso** | Intersección |
|---|---|
| **Actores** | Sistema |
| **Propósito** | Permitir que un vehículo pueda gestionar las intersecciones del entorno. |
| **Resumen** | El caso de uso se inicia cuando el sistema decide asignar este |

| | |
|---|---|
| | comportamiento a un(os) vehículo(s) perteneciente(s) a la escena. |
| **Precondiciones** | Se está ejecutando la aplicación. |
| **Poscondiciones** | Se halla asignado el comportamiento |
| **Curso Normal de los eventos** | |
| **Acciones del Actor** | **Respuesta del Sistema** |
| 1 El sistema decide dadas las condiciones del ambiente asignar este comportamiento | 1.1 El sistema genera a partir de las clases definidas el comportamiento solicitado y lo asigna. |
| **Curso Alternativo de los eventos** | |
| **Acción X** | |
| **Prototipo de Interfaz de usuario** | |
| **Prioridad** | Crítico |

**Tabla 5 Descripción del CU Intersección en formato expandido**

| | |
|---|---|
| **Nombre del caso de Uso** | Paso Peatonal |
| **Actores** | Sistema |
| **Propósito** | Permitir que un vehículo pueda gestionar los pasos peatonales del entorno. |
| **Resumen** | El caso de uso se inicia cuando el sistema decide asignar este comportamiento a un(os) vehículo(s) perteneciente(s) a la escena. |
| **Precondiciones** | Se está ejecutando la aplicación. |
| **Poscondiciones** | Se halla asignado el comportamiento |
| **Curso Normal de los eventos** | |

| **Acciones del Actor** | **Respuesta del Sistema** |
|---|---|
| 1 El sistema decide dadas las condiciones del ambiente asignar este comportamiento | 1.1 El sistema genera a partir de las clases definidas el comportamiento solicitado y lo asigna. |
| **Curso Alternativo de los eventos** | |
| **Acción X** | |
| **Prototipo de Interfaz de usuario** | |
| **Prioridad** | Crítico |

**Tabla 6 Descripción del CU Paso Peatonal en formato expandido**

## CAPÍTULO 3: CARACTERÍSTICAS DEL SISTEMA

### Introducción

A continuación se presentarán los diagramas de clases de análisis y sus diagramas de interacción por cada caso de uso, así como el diagrama general de clases de diseño y la descripción de las clases implicadas, así como los diagramas de Componente y de Despliegue.

### 3.1 Diagramas de Análisis e Interacción por Caso de Uso

Los diagramas clases del análisis asociados con sus correspondientes diagramas de interacción son un paso importante para lograr que los desarrolladores tengan un entendimiento profundo de cada uno de los procesos del sistema, es importante destacar que son la base de la arquitectura que se le vaya a aplicar al software, sin tener que depender de los lenguajes de programación

#### 3.1.1 CU Movimiento Libre

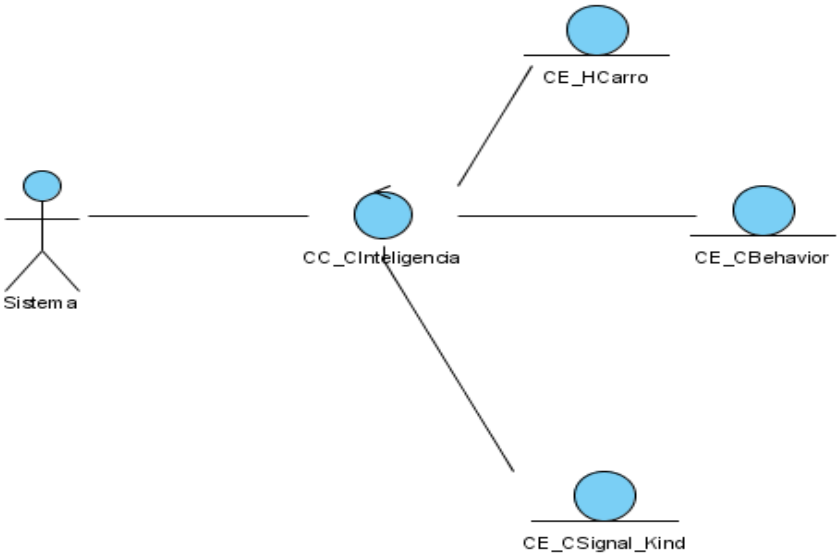


**Figura 12 Diagrama de clases del análisis CU Movimiento Libre**

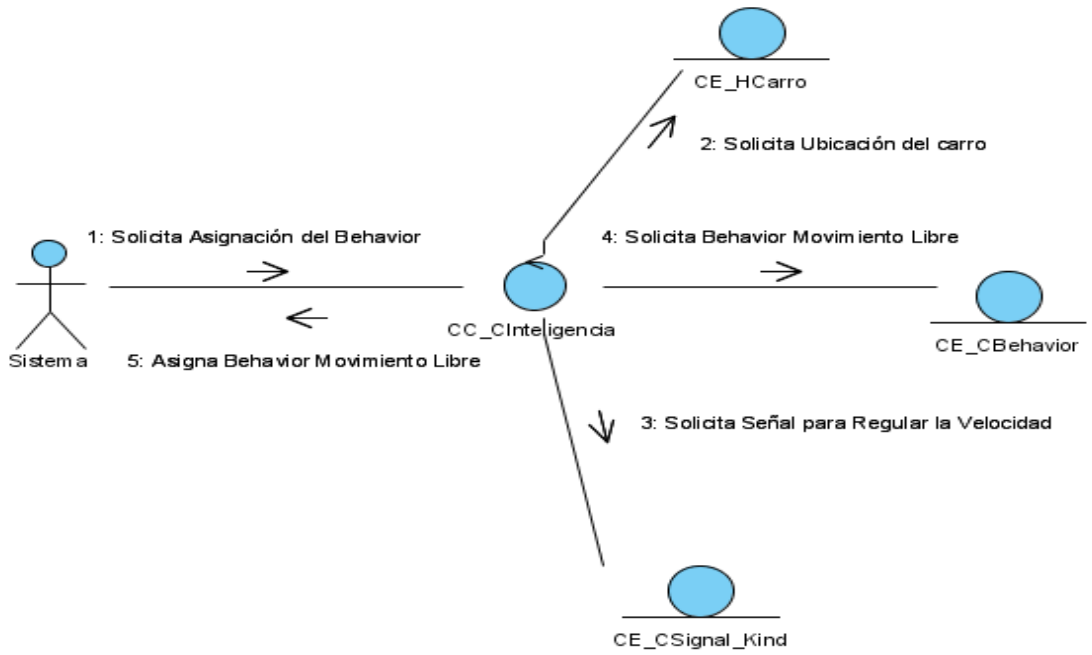


**Figura 13 Diagrama de colaboración del análisis CU Movimiento Libre**

### 3.1.2 CU Seguimiento

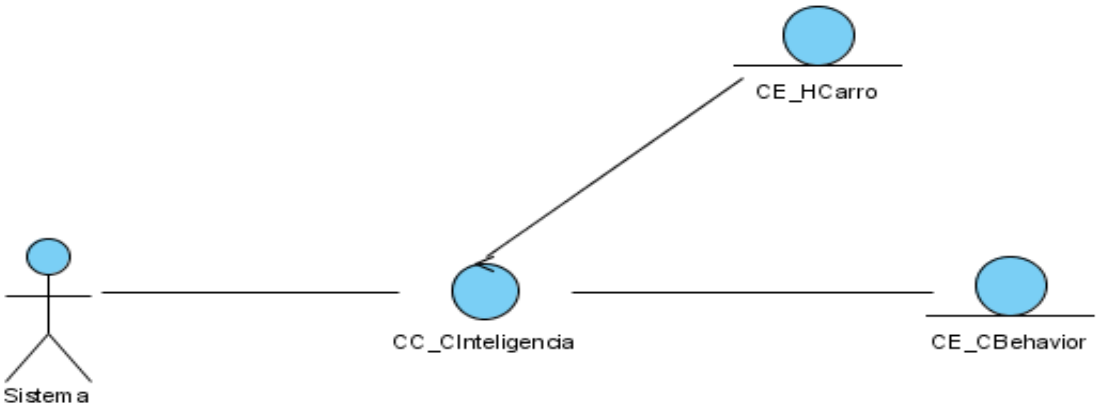


**Figura 14 Diagrama de clases del análisis CU Seguimiento**

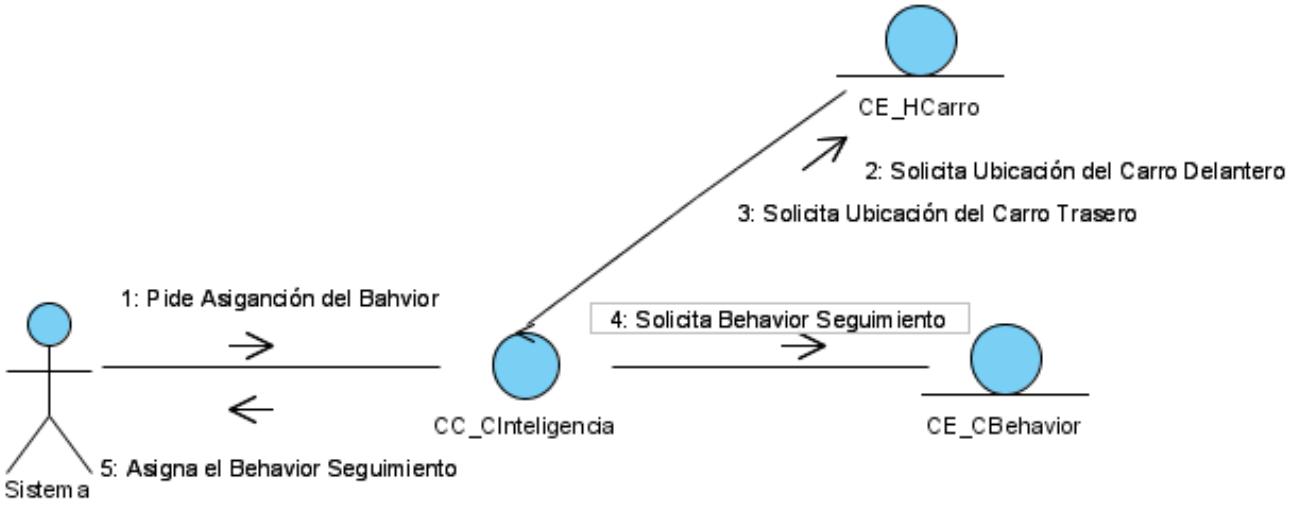


**Figura 15 Diagrama de colaboración del análisis CU Seguimiento**

### 3.1.3 CU Intersección

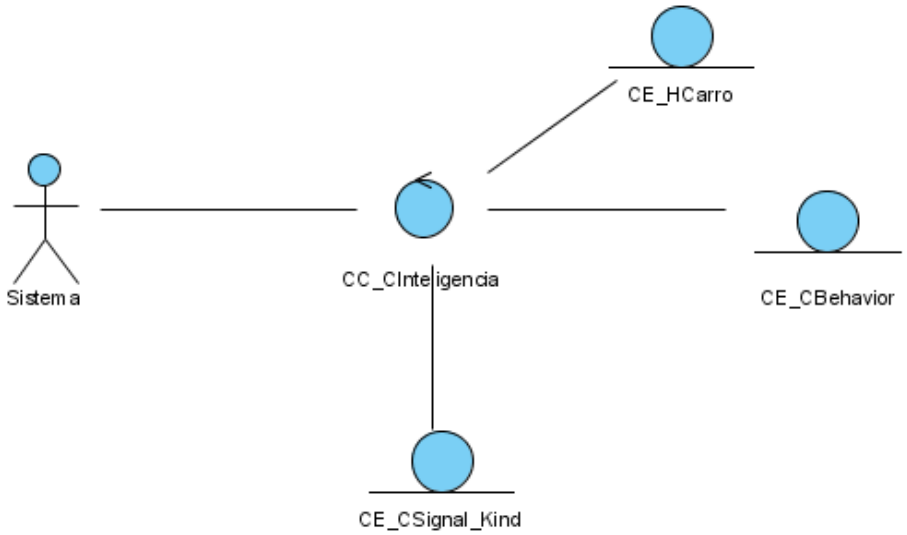


**Figura 16 Diagrama de clases del análisis CU Intersección**

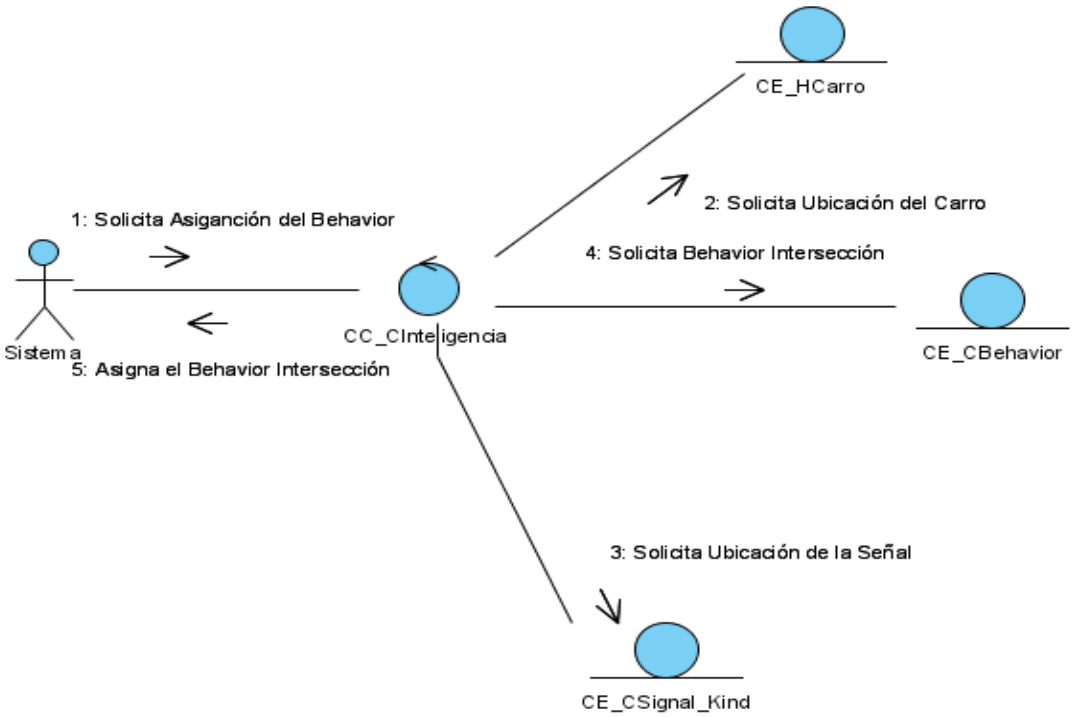


**Figura 17 Diagrama de colaboración del análisis CU Intersección**

### 3.1.4 CU Paso Peatonal

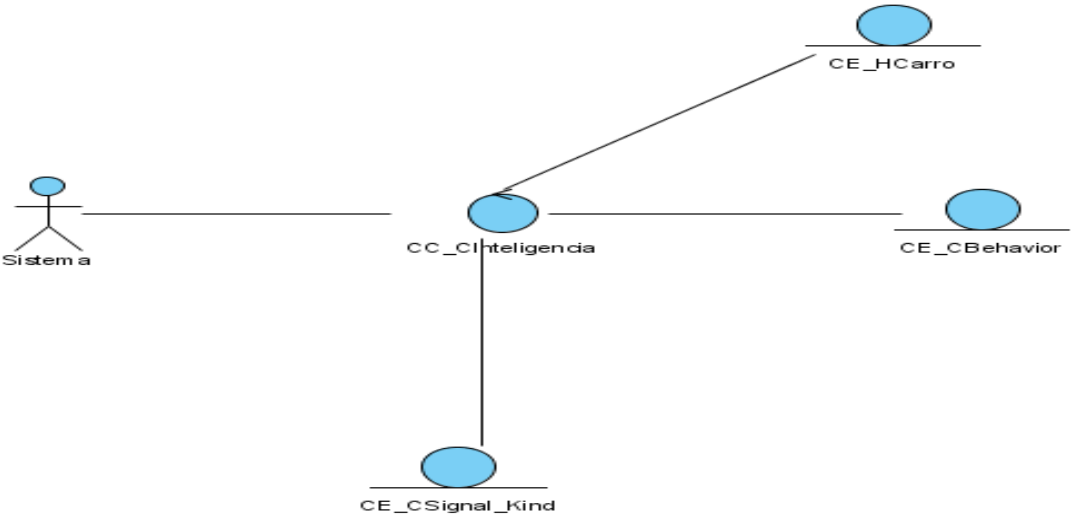


**Figura 18 Diagrama de clases del análisis CU Paso Peatonal**

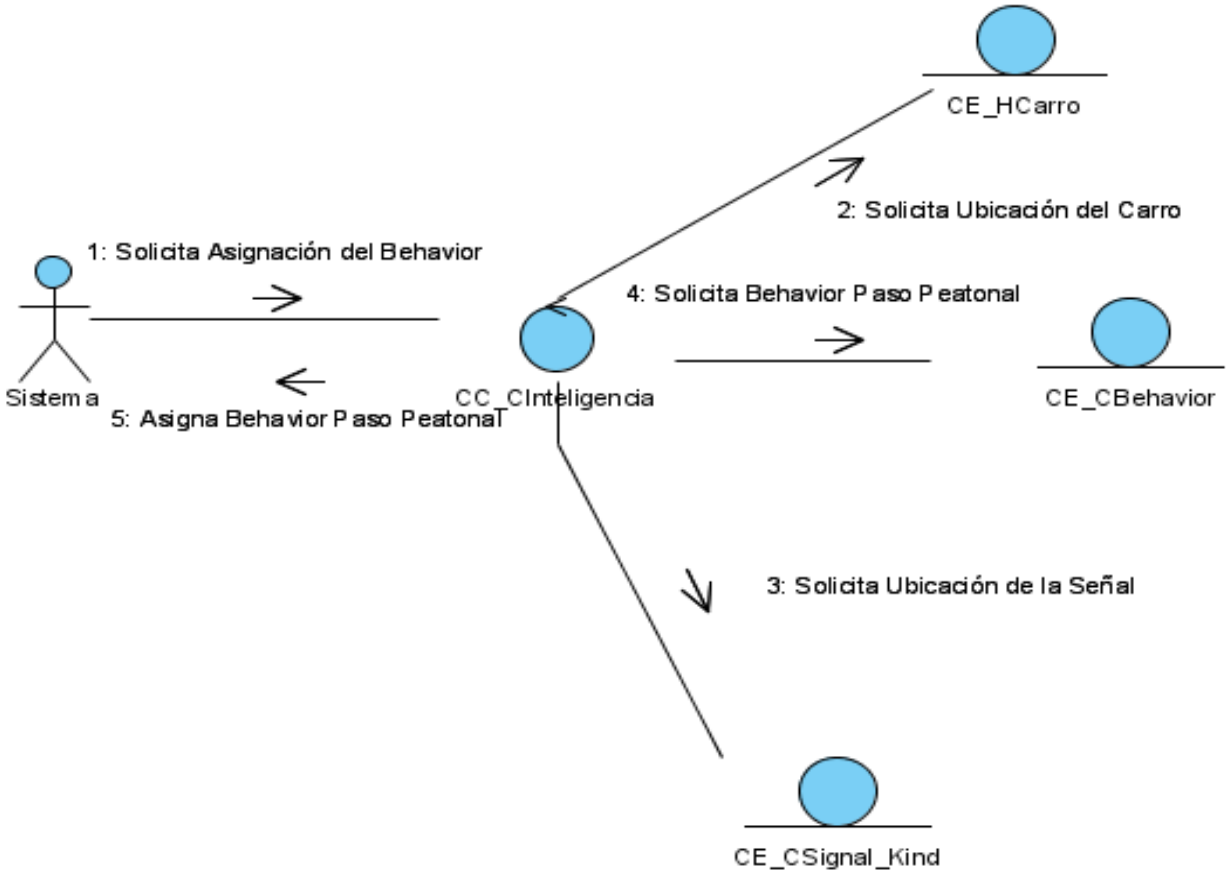


**Figura 19 Diagrama de colaboración del análisis CU Paso Peatonal**

### 3.2 Diagrama de Diseño

Los diagramas de clases de diseño tienen como premisa el análisis del sistema y su objetivo es dejar la estructura de la solución planteada en un lenguaje de programación específico.

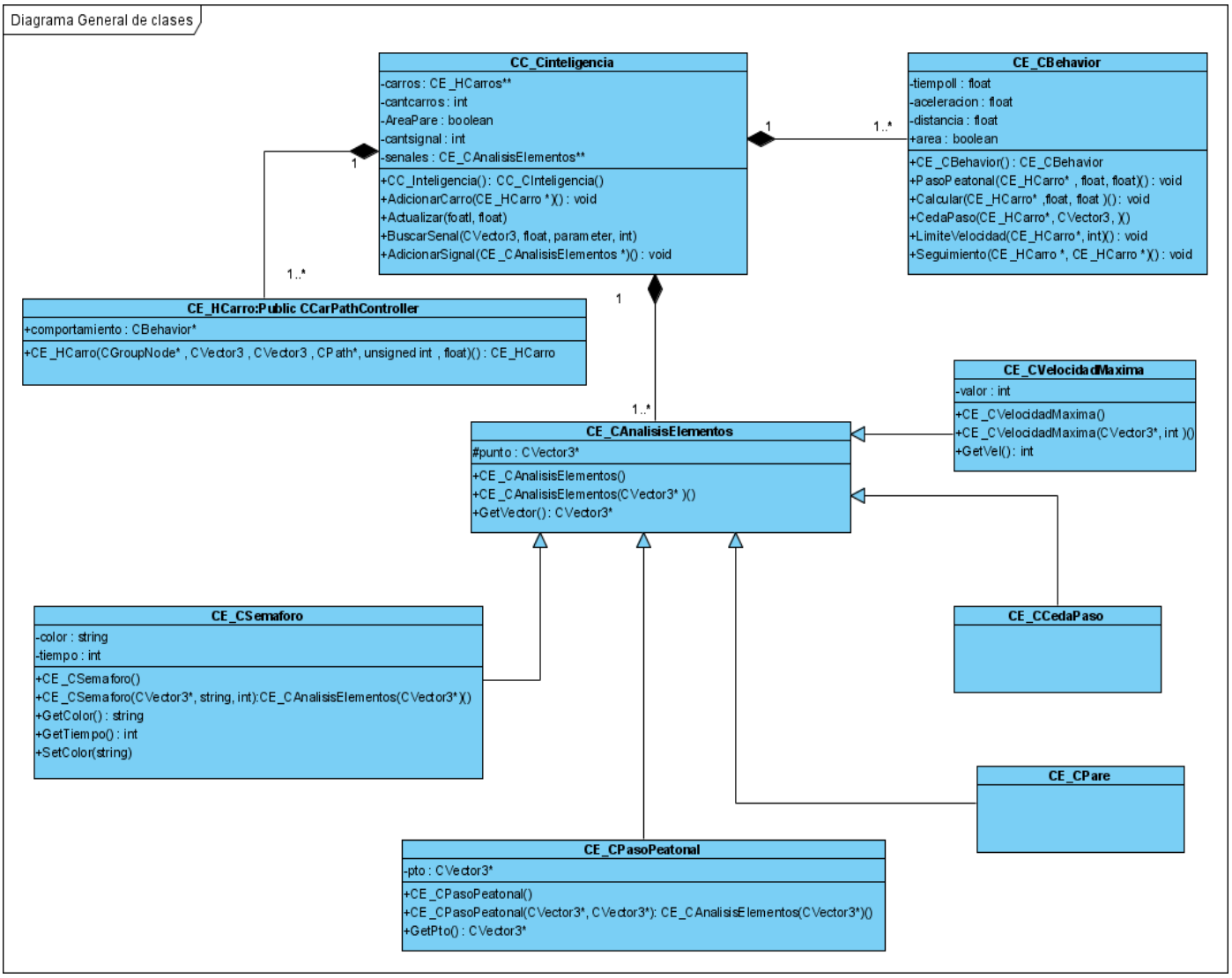


**Figura 20 Diagrama de clases del diseño.**

### 3.3 Descripción de las clases

En este epígrafe se van a describir las clases que conforman el esqueleto de la aplicación así como principales atributos y funcionalidades.

| **Nombre: CE_CBehavior** | |
|---|---|
| **Tipo de clase (Entidad)** | |
| **Atributo** | **Tipo** |
| tiempoll | float |
| aceleracion | float |
| distancia | float |
| area | bool |
| **Para cada responsabilidad**: | |
| Nombre: | CE_CBehavior() |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | PasoPeatonal(CE_HCarro* , float , float) |
| Descripción: | Se encarga de simular el *Behavior* Paso Peatonal, este *behavior* realizará el comportamiento establecido para este tipo de señalización. |
| | |
| Nombre: | Calcular(CE_HCarro* , float) |
| Descripción: | Calcula la velocidad que se le va a impartir al vehículo |
| | |
| Nombre: | CedaPaso (CE_HCarro* , CVector3*) |
| Descripción: | Se encarga de simular el *Behavior* Ceda el Paso, este *behavior* realizará el comportamiento establecido para este tipo de señalización. |
| | |
| Nombre: | LimiteVelocidad(CE_HCarro* , int) |
| Descripción: | Se encarga de simular el *Behavior* Movimiento Libre, este *behavior* realizará el comportamiento establecido para esta acción. |

| | |
|---|---|
| | |
| Nombre: | Seguimiento(CE_HCarro*, CE_HCarro*) |
| Descripción: | Se encarga de simular el *Behavior* Seguimiento, este *behavior* realizará el comportamiento establecido para esta acción. |

**Tabla 7 Descripción de la clase CE_CBehavior.**

| **Nombre: CE_HCarro** | |
|---|---|
| **Tipo de clase (Entidad)** | |
| **Atributo** | **Tipo** |
| comportamiento | CBehavior* |
| **Para cada responsabilidad**: | |
| Nombre: | CE_HCarro(CGroupNode* , CVector3 , CVector3 , CPath*, unsigned int , float)() : CE_HCarro |
| Descripción: | Constructor de la clase. |

**Tabla 8 Descripción de la clase CE_HCarro**

| **Nombre: CE_CAnalisisElementos** | |
|---|---|
| **Tipo de clase (Entidad)** | |
| **Atributo** | **Tipo** |
| punto | CVector3* |
| **Para cada responsabilidad**: | |
| Nombre: | CE_CAnalisisElementos() |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | CE_CAnalisisElementos (CVector3*) |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | GetVector() |

| | |
|---|---|
| Descripción: | Devuelve el vector punto. |

**Tabla 9 Descripción de la clase CE_CAnalisisElementos**

| **Nombre: CE_CSemaforo** | |
|---|---|
| **Tipo de clase (Entidad)** | |
| **Atributo** | **Tipo** |
| color | string |
| tiempo | int |
| **Para cada responsabilidad**: | |
| Nombre: | CE_CSemaforo() |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | CE_CSemaforo(CVector3*, string, int ) |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | GetColor() |
| Descripción: | Devuelve el atributo color. |
| | |
| Nombre: | GetTiempo() |
| Descripción: | Devuelve el atributo tiempo. |
| | |
| Nombre: | SetColor(string ) |
| Descripción: | Cambia el atributo color. |

**Tabla 10 Descripción de la clase CE_CSemaforo**

| **Nombre: CE_CPasoPeatonal** |
|---|
| **Tipo de clase (Entidad)** |

| **Atributo** | **Tipo** |
|---|---|
| Pto | CVector3* |
| **Para cada responsabilidad**: | |
| Nombre: | CE_CPasoPeatonal () |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | CE_CPasoPeatonal (CVector3*, CVector3*) |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | GetPto() |
| Descripción: | Devuelve el atributo pto. |

**Tabla 11 Descripción de la clase CE_CPasoPeatonal**

| **Nombre: CE_CPare** | |
|---|---|
| **Tipo de clase (Entidad)** | |
| **Atributo** | **Tipo** |
| **Para cada responsabilidad**: | |

**Tabla 12 Descripción de la clase CE_CPare**

| **Nombre: CE_CCedaPaso** | |
|---|---|
| **Tipo de clase (Entidad)** | |
| **Atributo** | **Tipo** |
| **Para cada responsabilidad**: | |

**Tabla 13 Descripción de la clase CE_CCedaPaso**

| **Nombre: CE_CVelocidadMaxima** | |
|---|---|
| **Tipo de clase (Entidad)** | |
| **Atributo** | **Tipo** |
| valor | int |
| **Para cada responsabilidad**: | |
| Nombre: | CE_CVelocidadMaxima() |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | CE_CVelocidadMaxima(CVector3*, int) |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | GetVel() |
| Descripción: | Devuelve el atributo valor. |

**Tabla 14 Descripción de la clase CE_CVelocidadMaxima**

| **Nombre: CC_CInteligencia** | |
|---|---|
| **Tipo de clase (Controladora)** | |
| **Atributo** | **Tipo** |
| carros | CE_HCarros** |
| senales | CE_CAnalisisElementos** |
| cantcarros | int |
| AreaPare | boolean |
| cantsignal | int |
| **Para cada responsabilidad**: | |
| Nombre: | CC_Inteligencia() |
| Descripción: | Constructor de la clase. |
| | |
| Nombre: | AdicionarCarro(CE_HCarro *) |

| | |
|---|---|
| Descripción: | Permite adicionar carros al atributo “carros”. |
| | |
| Nombre: | AdicionarSignal(CE_CAnalisisElementos *) |
| Descripción: | Permite adicionar carros al atributo “senales”. |
| | |
| Nombre: | Actualizar(float, float) |
| Descripción: | Actualiza la condición de todos los vehículos en la escena |
| | |
| Nombre: | BuscarSenal(CVector3 , float , float , int ) |
| Descripción: | Busca la señal que se encuentre en un área cercana a un vehículo. |

**Tabla 15 Descripción de la clase CC_CInteligencia**

### 3.4 Diagrama de Componentes

#### 3.4.1 Subsistema Señal

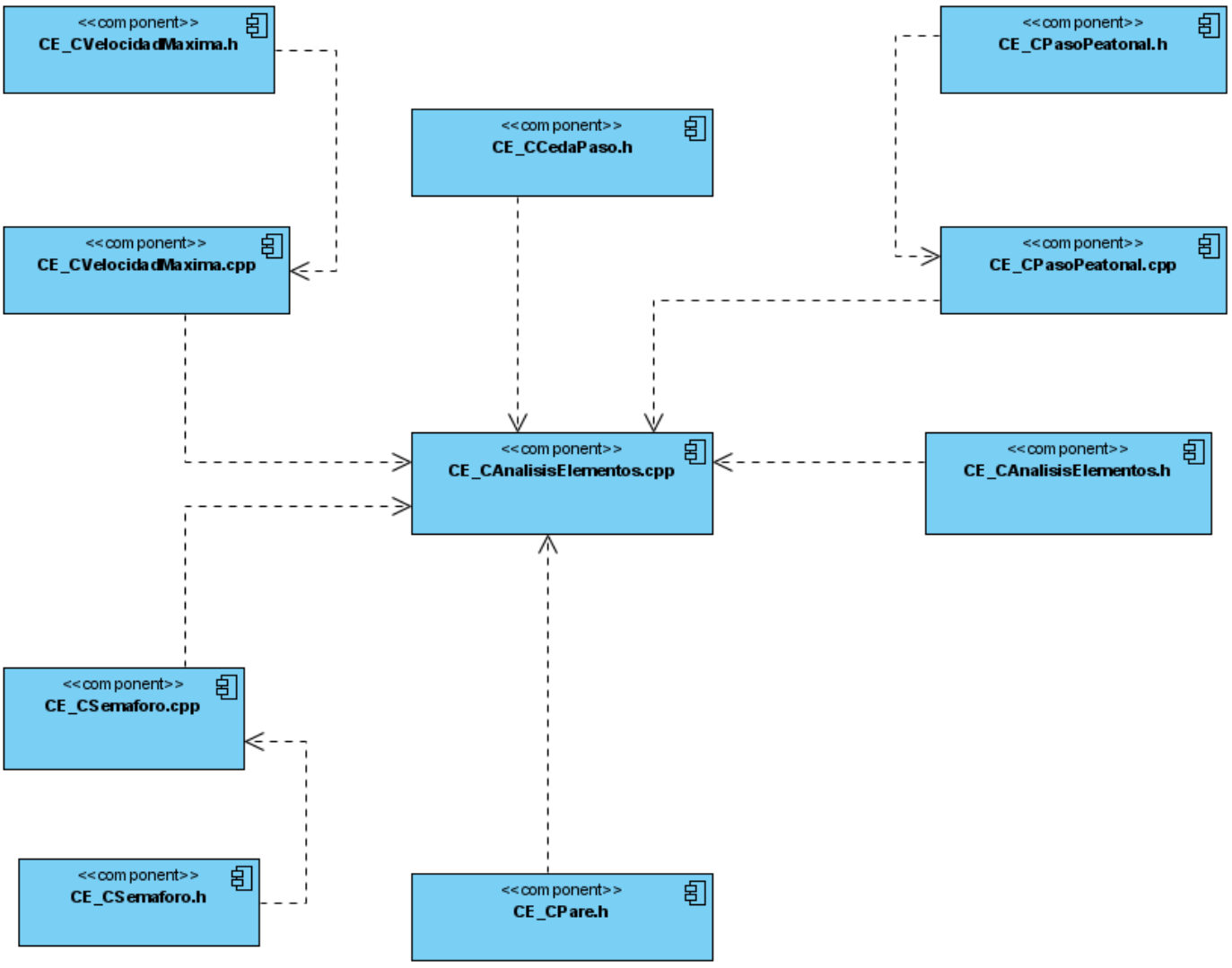


Figura 21 Diagrama de componentes Subsistema Señal

### 3.4.2 Subsistema *Behavior*

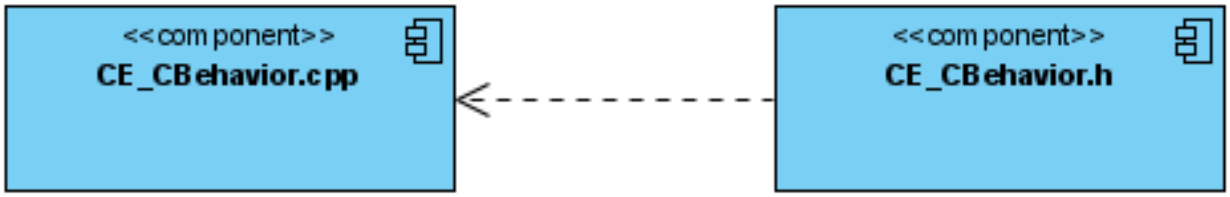


**Figura 22 Diagrama de componentes Subsistema *Behavior***

### 3.4.3 Subsistema Central

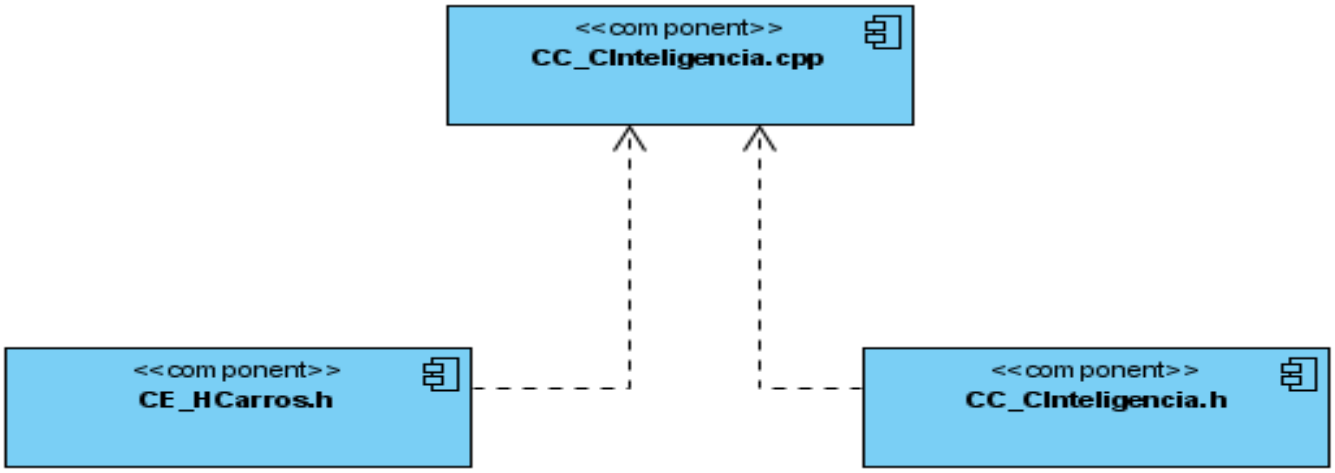


**Figura 23 Diagrama de componentes Subsistema Central**

### 3.4.4 Diagrama de Componentes General

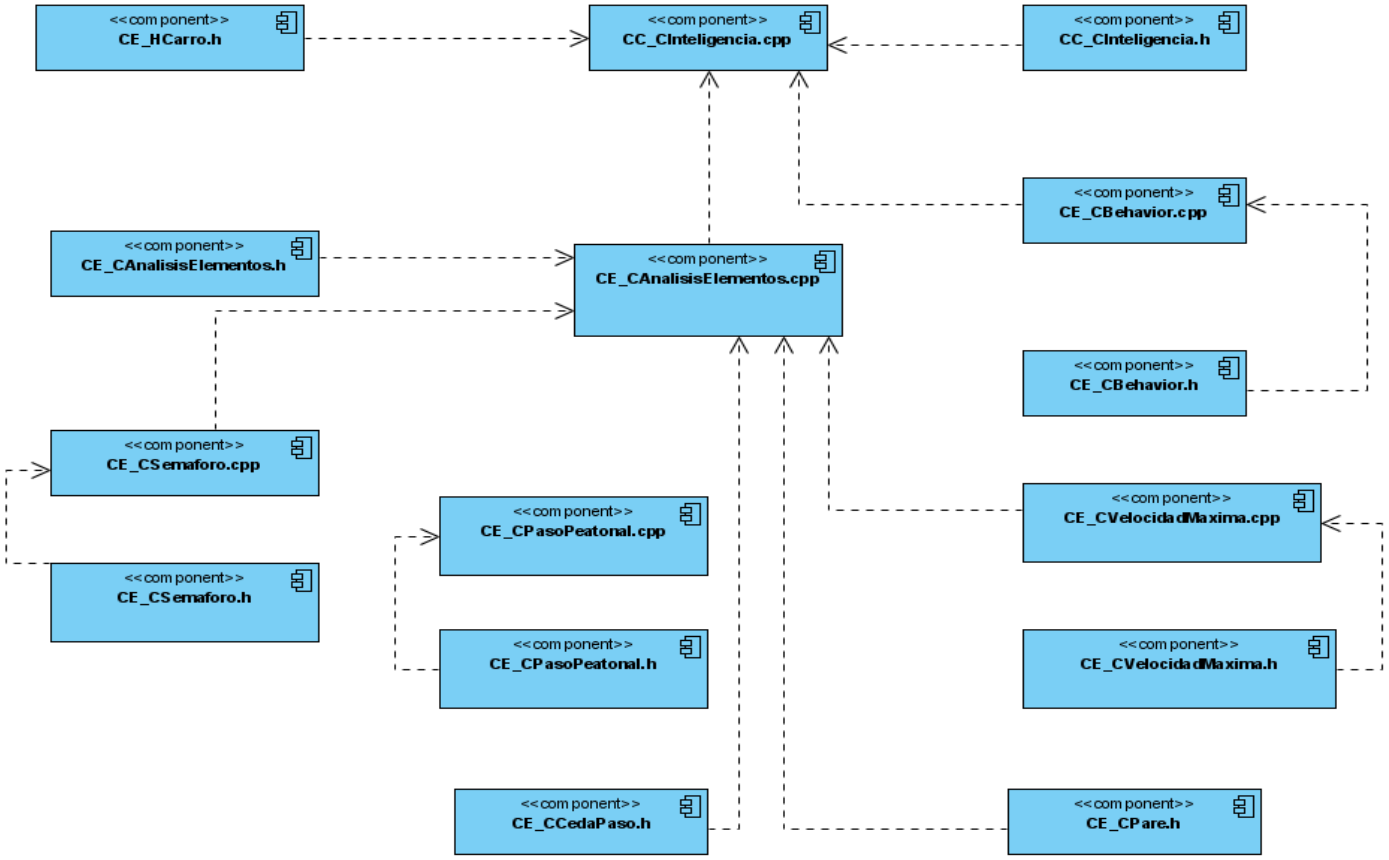


Figura 24 Diagrama de Componentes General

### 3.4.5 Diagrama de Despliegue

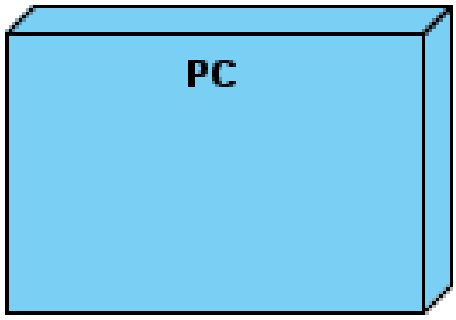


Figura 25 Diagrama de Despliegue

**CONCLUSIONES**

Para el cumplimiento de los objetivos de este proyecto, teniendo en cuenta las características del producto que queríamos lograr y de las herramientas y metodologías necesarias para su realización, fue necesario hacer un estudio de las técnicas y tecnologías actuales implicadas en la creación de *Behaviors*, incluyendo los diferentes algoritmos más comúnmente usados en la conformación de los mismos, esto dio pie a la creación de los *Behaviors* utilizados en el trabajo. Para lograr esto se realizó el proceso de Ingeniería del Software utilizando el Proceso Unificado del Software, con el que se logró plasmar la captura de los requisitos funcionales y no funcionales, la identificación de casos de uso del sistema en conjunto con su descripción en formato expandido, los diagramas que se generan del flujo de trabajo Análisis y Diseño y los diagramas de componentes propios de nuestra solución.

Finalmente, se obtuvo el módulo que cumple con los objetivos del proyecto, proveyendo de esta forma una herramienta para el trabajo con entornos virtuales de ciudad donde interactúen vehículos. Esto posibilita una mayor velocidad a la hora de crear nuevos software que necesiten entornos virtuales de este tipo; esto se traduce en un menor coste de tiempo y recursos, ya que el trabajo se simplifica en un 60%, teniendo como resultado que nuestro trabajo reportará beneficios en el campo de la informática, por la temática desarrollada en la que se aplican conocimientos muy novedosos y de amplia utilización por las posibilidades que ofrecen; en el campo de la economía, ya que menos trabajo trae aparejado un menor gasto de energía, o lo que es lo mismo, de combustible y un menor uso de las computadoras y de las tecnologías implicadas en el proceso de desarrollo, lo que constituye un menor deterioro de las mismas; por último también traerá beneficios a la salud humana, ya que al disminuir el tiempo que emplean los desarrolladores en terminar el producto, estos estarán menos expuestos a padecer de las enfermedades que generalmente afectan a las personas que permanecen mucho tiempo sentadas por la adopción de posturas incorrectas.

**RECOMENDACIONES**

- Seguir trabajando en la implementación de este modulo.
- Acoplar al módulo los elementos necesarios para incluir a las “personas” en el entorno.
- Expandir nuestra propuesta de solución agregando la herencia al subsistema *behavior*, o sea, no implementar todos los *behaviors* en una única clase, sino, implementar cada *behavior* como una clase.

**REFERENCIAS BIBLIOGRÁFICAS**


**1.** http://www.monografias.com/trabajos12/inteartf/inteartf.shtml

**2.** http://www.research.scea.com/research/pdfs/CWR2001_03_20_GDC.pdf

**3.** Ídem a referencia **2**

**4.** http://personales.upv.es/ccarrasc/extdoc/Tema2_2.pdf

**5.** http://www.monografias.com/trabajos10/exper/exper.shtml

**6.** http://www.fi.uba.ar/laboratorios/lsi/p-felgaer-proyectodetesis.htm

**7.** http://personales.ya.com/casanchi/mat/difusa01.htm

**8.** http://www.generation5.org/redir.asp?To=GA

**9.** http://www.generation5.org/redir.asp?To=NN

**10.** http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/suarez_r_pk/capitulo5.pdf

**11.** Ídem a referencia **10**

**12.** Ídem a referencia **10**

**13.** Ídem a referencia **10**

**14.** Ídem a referencia **10**

**15.** Ídem a referencia **10**

**16.** Ídem a referencia **10**

**17**. Ídem a referencia **10**

**18.** http://www.research.scea.com/research/pdfs/CWR2001_03_20_GDC.pdf

**19.** http://www.red3d.com/cwr/steer/

**20.** Ídem a referencia **19**

**21.** Ídem a referencia **19**

**22.** Ídem a referencia **19**

**23.** Ídem a referencia **19**

**24.** http://citeseer.org/bonakdarian98generation.html

**25.** Ídem a referencia **25**

**26.** http://citeseer.org/pyeatt98learning.html

**27.** Ídem a referencia **27**

**28.** Ídem a referencia **10**

**29.** http://www.red3d.com/cwr/steer/SeekFlee.html

**30.** http://fbim.fh-regensburg.de/~saj39122/feisch/Diplomarbeit/theory_eng/steering_eng.html

**31.** http://www.red3d.com/cwr/steer/Arrival.html

**32.** Ídem a referencia **31**

**33.** http://www.red3d.com/cwr/steer/Wander.html

**34.** Ídem a referencia **32**

**35.** http://fbim.fh-regensburg.de/~saj39122/feisch/Diplomarbeit/theory_eng/steering_eng.html

**36.** Ídem a referencia **35**

**37** http://www.red3d.com/cwr/steer/Obstacle.html

**38** **Yuniesky Coca Bergolla.** Agentes Inteligentes. Aplicación a la Realidad Virtual.

**39** Ídem a referencia **38.**

**40** Ídem a referencia **38.**

**BIBLIGRAFÍA CONSULTADA**


- http://www.monografias.com/trabajos12/inteartf/inteartf.shtml
- http://www.research.scea.com/research/pdfs/CWR2001_03_20_GDC.pdf
- http://personales.upv.es/ccarrasc/extdoc/Tema2_2.pdf
- http://www.monografias.com/trabajos10/exper/exper.shtml
- http://www.fi.uba.ar/laboratorios/lsi/p-felgaer-proyectodetesis.htm
- http://personales.ya.com/casanchi/mat/difusa01.htm
- http://www.generation5.org/redir.asp?To=GA
- http://www.generation5.org/redir.asp?To=NN
- http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/suarez_r_pk/capitulo5.pdf
- http://www.research.scea.com/research/pdfs/CWR2001_03_20_GDC.pdf
- http://www.red3d.com/cwr/steer/
- http://citeseer.org/bonakdarian98generation.html
- http://citeseer.org/pyeatt98learning.html
- http://www.red3d.com/cwr/steer/SeekFlee.html
- http://fbim.fh-regensburg.de/~saj39122/feisch/Diplomarbeit/theory_eng/steering_eng.html
- http://www.red3d.com/cwr/steer/Arrival.html
- http://www.red3d.com/cwr/steer/Wander.html
- http://www.red3d.com/cwr/steer/Obstacle.html
- **Yuniesky Coca Bergolla.** Agentes Inteligentes. Aplicación a la Realidad Virtual.

**GLOSARIO DE TERMINOS**

**Variables de salida:** Conjunto solución de un problema, entiéndase, solución implementada de problema informático.

**Variables de entradas:** Conjunto de valores que conforman las premisas de un problema.

**Variables inciertas:** Hipótesis a demostrar pertenecientes a un problema específico.

**Dependencias cuantificadas:** Valores que denotan el peso, la importancia o el costo de una actividad.

**Conocimiento incierto:** Supuesta solución, que está basada en hipótesis sin demostrar.

**Estado:** Entiéndase posible solución (en el epígrafe de Redes Bayesianas).

**Discriminado:** Eliminado (en el epígrafe de Redes Bayesianas).

**Grado:** Nivel, especificidad.

**Herencia:** Algoritmo que simula este proceso biológico.

**Mutación:** Algoritmo que simula este proceso biológico.

**Selección:** Algoritmo que simula este proceso biológico.

**Cruce:** Algoritmo que simula este proceso biológico.

***Steering Behaviors*:** Comportamientos dirigidos (en español)

**Carácter:** Entiéndase persona o animal, en un entorno virtual sería un ente virtual.