UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
VICERRECTORÍA DE FORMACIÓN
DIRECCIÓN DE FORMACIÓN POSTGRADUADA

"Control del tránsito en un simulador de conducción de auto"

Colección de artículos presentados en opción al título de Máster en Informática Aplicada.

Autor: ***Lic. Yuniesky Coca Bergolla***

Tutor: ***Dr C. José I. Guzmán Montoto***

Ciudad de la Habana, julio de 2007

SÍNTESIS

La realización de un simulador de conducción de auto es un hecho en nuestro país, para darle un enfoque instructivo, de entrenamiento y con el realismo necesario se requiere controlar el tránsito en el entorno virtual que se recrea en dicho simulador, haciendo diferencias entre la detección de las infracciones del conductor y el movimiento autónomo de los autos y demás elementos en la ciudad virtual.

A partir de una sucesión de ponencias presentadas y publicadas, el autor realiza un análisis crítico de cada una de ellas, mostrando en cada caso su aporte individual y algunas ideas nuevas que enriquecen el contenido referenciado.

Se brinda una panorámica al lector de los contenidos abordados en los trabajos previos y los principales resultados obtenidos, además de realizar una propuesta final como consolidación de toda la investigación llevada a acabo por el autor, lo cual permite consolidar un producto para el entrenamiento y aprendizaje de la conducción de auto con un alto nivel de realismo y satisfacción de los usuarios.

Índice

INTRODUCCIÓN

La Universidad de las Ciencias Informáticas de manera conjunta con la empresa SIMPRO desarrolla desde hace algunos años un simulador de conducción de auto, la universidad es la encargada del software que forma parte de dicho simulador, se ha desarrollado por parte de un gran equipo de estudiantes y profesores un trabajo constante y con resultados concretos. Para el desarrollo del entorno virtual y lograr un alto realismo que exige una aplicación de esta envergadura se dividió el proyecto en distintos módulos, uno de ellos es el de Inteligencia Artificial, que fue dividido a su vez, en tres etapas fundamentales:

1. La detección de infracciones del conductor.
2. El movimiento autónomo de los autos del entorno.
3. La incorporación de personas auto-controladas y otros elementos a la ciudad.

El objetivo final es concluir cada uno de los sub-módulos que genera cada etapa e incorporarlo al simulador. Se presenta en este trabajo ya concluido e incorporado el primero de ellos (Detección de infracciones del conductor) y en desarrollo el segundo (Movimiento autónomo de los autos en el entorno).

Este tipo de trabajo no se ha realizado con anterioridad en nuestro país. En el mundo los simuladores de conducción son muy escasos y toda la información referente a los mismos está protegida y no se divulga nada concretamente de cómo llegar a construirlos. Teniendo en cuenta que los simuladores de conducción tienen varias similitudes con los juegos de autos, se ha trabajado a partir de bibliografía que existe sobre ese tipo de aplicaciones, aunque tampoco en nuestro país se ha desarrollado algún juego importante con estas características, sí existen libros que muestran algunas investigaciones y aplicaciones útiles para llegar a los mismos, toda esta bibliografía se concentra en libros especializados sobre videojuegos, generalmente editados en EEUU y Europa.

Los estudios para el desarrollo del módulo de Inteligencia artificial del simulador comenzaron a finales del año 2005 presentándose un primer trabajo en el evento UCiencia 2005, se crea posteriormente el Grupo de Desarrollo de Elementos Virtuales Inteligentes "UVi-Bot" formado por un profesor líder del proyecto y varios estudiantes de tercer año de la facultad 5 "Entornos Virtuales" de la Universidad de las Ciencias Informáticas.

La investigación se ha centrado en una de las técnicas más avanzadas en sistemas de realidad virtual para la autonomía de los elementos virtuales, los *Steering Behaviors*, una idea que surge por el investigador Craig Reynolds en el año 1987 aunque se formula concretamente por el mismo científico en el año 1999. Se basa en la manipulación de los comportamientos de los elementos, logrando los mismo a partir de ejercer una determinada fuerza sobre ellos y aplicándoles la segunda ley de Newton o ley de la fuerza, para obtener la aceleración y la dirección en la cual se deben mover, a partir de ahí se obtiene la velocidad y su nueva posición en el entorno. Además de esta técnica muy importante se han incorporado varias otras que ayudan al cumplimiento de los objetivos.

Análisis científico metodológico.

En el simulador de conducción de auto no existe un módulo para el control del tránsito en la ciudad virtual que recrea. Los autos que se mueven por el entorno tienen una trayectoria predeterminada y el conductor puede realizar cualquier maniobra sin que le sea señalada la infracción que cometió. Para dar solución a este problema el trabajo se centra en el *control de los elementos en entornos virtuales*, para accionar concretamente en los *autos en un entorno virtual de ciudad.*

El principal objetivo del trabajo es:

Proponer un conjunto de técnicas y algoritmos que permitan el control del tránsito en un entorno virtual de ciudad en un simulador de conducción de auto.

Más específicamente se propone:

- Detectar, en tiempo real, las infracciones de tránsito del conductor al realizar una maniobra en el entorno de ciudad.
- Mostrar las infracciones cometidas por el conductor.
- Introducir el tema de los *Steering Behaviors* y su efectividad en este tipo de aplicaciones.
- Proponer las estructuras de datos, técnicas y algoritmos necesarios para lograr el movimiento autónomo de los autos en un entorno virtual de ciudad.
- Proponer variantes para lograr eficiencia en el uso de los recursos computacionales.

Para poder dar cumplimiento a los objetivos propuestos se podrían formular varias interrogantes:

¿Se logrará implementar un módulo para detectar las infracciones cometidas por el conductor, con un bajo consumo de recursos?

¿Con el uso de los *Steering Behaviors* (Comportamientos) se logrará el movimiento autónomo de los autos, logrando que respeten las señales de tránsito y sin afectar considerablemente la eficiencia del sistema?

Las tareas de investigación a llevar a cabo se pueden resumir en las siguientes:

1. Realizar un estudio del problema y la situación actual del tema.
2. Elaborar la documentación necesaria vinculada a la investigación.
3. Implementar el módulo de detección de infracciones e incorporarlo al simulador.
4. Formular una propuesta que constituya la base científica para la posterior implementación del módulo para el movimiento autónomo de los autos en el entorno virtual del simulador.

Impacto de la propuesta.

Para la elaboración de esta primera etapa se emplearon los medios disponibles para la formación de los estudiantes en la universidad, no se requirió de costos adicionales, ya el simulador existe con las características necesarias

para incorporarle este módulo por lo que tampoco se necesitó alguna inversión para llevar a cabo esta investigación. La bibliografía consultada está disponible en Internet, en la biblioteca de nuestra universidad y en la base bibliográfica del proyecto, los software utilizados están disponibles en nuestra universidad, además el trabajo se realizó completamente compatible con software libre, si en algún momento se requiere de la compra de los software se puede transferir el código a GNU/Linux sin mayores afectaciones o trabajar siempre con la misma plataforma en que se desarrolle el simulador.

Las infracciones cometidas por el usuario se visualizan en una de las pantallas del simulador cuando el conductor o el instructor lo deseen, incluso puede estar mostrándose en tiempo real, lo cual tiene un gran impacto en el usuario que puede conocer en cada instante que señal de tránsito violó. Como el mismo usuario puede tener en cada instante las infracciones que va cometiendo, para su entrenamiento no requiere de un instructor que le vaya diciendo sus errores, dedicándose este último a otras actividades importantes.

La calidad del sistema aumenta considerablemente con este trabajo ya que, incluso estando el instructor, puede escapársele alguna infracción, cuando este proceso está automatizado, como en este caso, no hay posibilidad de que se "escape" algún error cometido por el usuario.

El entrenamiento mediante este tipo de aplicaciones puede reportar inmensos ahorros de combustible, el simulador solo requiere de corriente eléctrica y mantenimiento mecánico. No requiere combustible adicional, ni otro tipo de productos, además de tener un mecanismo mucho más sencillo y económico que un auto real, con la consiguiente ayuda al medio ambiente, evitando una gran cantidad de gases contaminantes a la atmósfera.

Evita accidentes sobre todo en usuarios novatos que tengan errores muy básicos y se dispongan a entrenar en un entorno real.

Con el prestigio de la educación en Cuba, tener el simulador con el módulo de detección de infracciones, reporta una gran ventaja para la posible exportación del mismo.

Mantener un tránsito inteligente en el entorno, garantizando que cada vez que el usuario se enfrente al ejercicio se le presenten nuevas situaciones aleatorias, tiene un impacto psicológico importante, ya que al igual que en la realidad tiene que estar atento cada momento a lo que se le pueda presentar, esto redunda en una mayor calidad en el entrenamiento.

Publicaciones y trabajos relacionados con la investigación.

El primer trabajo realizado sobre este tema fue una ponencia presentada en el evento científico UCiencia 2005, ¿Simulador de auto inteligente? La cual fue publicada en las memorias del evento. Al año siguiente se presenta un nuevo trabajo en la segunda edición del mismo evento UCiencia 2006 con el nombre "Algunas ideas a tomar en cuenta para la inteligencia de un simulador de auto". Este trabajo fue propuesto y presentado en la Feria Internacional Informática 2007, en su evento virtual.

En el propio año 2007 se discute una tesis de grado con el título "Detección de infracciones del conductor en un simulador de conducción de auto" donde el autor de este trabajo funge como tutor. El producto que se obtiene de la tesis fue incorporado al simulador que se presentó en el área expositiva de la ya mencionada Feria Internacional Informática 2007.

También se presenta un póster en el evento Provincial "Expo Forjadores del Futuro" organizado por la UJC y se presenta un trabajo en el XV FORUM de base de Ciencia y Técnica, obteniendo premio Relevante.

DESARROLLO

Con el desarrollo de la investigación para crear el módulo de Inteligencia artificial del simulador de conducción de auto se elabora un primer articulo que es publicado en el evento UCiencia 2005 [COC05] [ANEXO 1]. El objetivo de dicho trabajo es brindar una panorámica de las técnicas a utilizar en la construcción del simulador de conducción de auto, sobre todo en el área del control de los elementos virtuales que interactúan en dicho entorno virtual.

El trabajo no presenta una gran profundidad, ni mucha calidad científica, solo se refiere a ideas aplicables y tiene como objetivo llegar a un amplio público que puede no tener muchos conocimientos sobre estos temas; muestra una cronología de aplicaciones desarrolladas en el mundo que tienen alguna relación de semejanza con el trabajo que se propone.

Se debe recalcar que no existe bibliografía disponible directamente vinculada al desarrollo de simuladores. Por lo que la primera contribución que se reconoce, de esta serie de trabajos publicados por el autor, es la adaptación de ciertos elementos, propuestos en la bibliografía para juegos, al desarrollo del módulo de inteligencia artificial del simulador de conducción de auto, sobre todo referido al control del tránsito del entorno.

El artículo al que se hace referencia muestra la importancia de la combinación de técnicas deterministas y no deterministas para alcanzar resultados cercanos a los deseados.

Ya desde este momento se muestra la importancia de encontrar la trayectoria a seguir en cada momento por los elementos y se habla muy superficialmente del tratamiento de los comportamientos de los elementos para lograr movimiento autónomo. Además de la necesidad de la utilización de leyes básicas de la física para lograr los objetivos propuestos.

Se presentan varias situaciones a las cuales se les debe dar respuesta con la implementación del módulo para el control del tránsito. Dejando claro de manera sencilla y abarcadora todo el campo de acción del naciente proyecto.

Se deja la puerta abierta para los próximos trabajos aclarando la necesidad de lograr la detección de infracciones del conductor y del movimiento de los autos y personas en la ciudad virtual.

La bibliografía consultada no fue abundante ni muy actualizada aunque sí la necesaria para los objetivos de dicha ponencia. Además del rigor científico, se muestra deficiencias en cuanto a la profundidad de algunos temas.

En una segunda ponencia presentada y publicada en UCiencia 2006 [COC06] y en Informática 2007 [COC07a] [ANEXO 2] se abordan temas de mucha más profundidad y muy útiles para trabajos desarrollados posteriormente [LOR07] [ANEXO 3] [COC07b] [ANEXO 4].

Se proponen ideas para influir en tres variables fundamentales que interactúan en cualquier sistema con estas características, la extensibilidad, la diversidad de actuación de los elementos y la eficiencia de la propuesta.

Ya dirigido a personas vinculadas a la rama, este trabajo, aborda de manera más profunda y con segmentos de código que esclarecen la comprensión, ideas renovadoras propuestas en bibliografía actualizada y de reconocido prestigio en el tema. A pesar de mantener algunas deficiencias metodológicas, logra influir en el lector de manera positiva para atraparlo en el tema y sentirse interesado a profundizar. El lenguaje es sencillo y concreto.

Dentro de la extensibilidad aborda temas como los autómatas, la división de las responsabilidades entre los elementos del entorno y el envío de mensajes entre ellos. Con relación a la diversidad de actuación se aborda la explotación al máximo de los valores que almacenan los elementos (atributos de las clases 'entidades') y la construcción de una jerarquía de clases. Por último para lograr eficiencia se refiere a la división del entrono, lo cual permite la interacción entre los elementos más cercanos y no entre todos los que existan en el entorno, además para saber en qué área se encuentra un elemento dinámico no se busca en todas las áreas del entorno, sino solo en las adyacentes a la última donde se encontraba, para esto se propone una relación (Grafo) entre las áreas.

Otro tema muy interesante para lograr eficiencia es el espaciamiento del análisis de la Inteligencia artificial, es decir que se actualicen los valores de los elementos dinámicos cada cierto tiempo y no en cada instante.

Detección de infracciones.

Como se muestra en el primero de los artículos publicados por el autor [ANEXO 1], una de las primeras tareas a llevar a cabo por los desarrolladores del módulo de inteligencia artificial del simulador es la detección de infracciones del conductor.

Con el objetivo de desarrollar dicho trabajo se lleva a cabo la elaboración de una base teórica que queda como documentación del proyecto y que sirve de base a dos estudiantes para desarrollar una tesis de grado (ver resumen en [ANEXO 3]) para optar por el titulo de Ingenieros en Ciencias Informáticas y darle cumplimiento a la problemática planteada.

El trabajo de la tesis se centra en construir una aplicación e incorporarla al simulador y que cumpla con los requerimientos necesarios para detectar y mostrar cada instante las infracciones cometidas por el conductor mientras transita por el entorno virtual.

Este es un trabajo completamente nuevo en nuestro país, no existe referencias de un simulador anterior al que se construye en la universidad, ni existe un tipo de aplicación similar que realice una detección de infracciones de chóferes en entornos virtuales urbanos.

Lo más interesante del trabajo realizado por los estudiantes fue ajustarse a las ideas planteadas por el tutor del trabajo en artículos previos y en el documento base para construir la aplicación.

Se respetan aspectos como la división del entorno en áreas no consecutivas y en ocasiones superpuestas. El chequeo de las señales mediante esas áreas. Además se logra un chequeo eficiente de las señales, logrando hacer la búsqueda en las más cercanas a la posición del conductor.

El trabajo cumple todos los objetivos trazados al inicio de la investigación.

Movimiento autónomo de autos.

El movimiento autónomo de los autos es el tema fundamental de investigación que propone el autor en la colección de trabajos presentada, se realizará en este epígrafe un análisis de cómo se ha ido consolidando la idea de la aplicación de este tema en la problemática a resolver. Se mostrará de manera simple el funcionamiento de dicha técnica, sus orígenes y los principales trabajos realizados en el mundo sobre la misma. Por último se realiza una propuesta concreta del autor para incorporar estas ideas y dar solución al objetivo general propuesto en este trabajo.

Comportamientos en elementos virtuales.

Todo entorno virtual ya sea para un juego, un simulador o similar, necesita de elementos que dentro de ese entorno se muevan, chequeen el entorno y actúen de manera independiente, es decir, sin ser controlados en cada instante por el programador o algún usuario del entorno. Un concepto de elemento o agente autónomo pudiera ser:

Es un sistema situado en un entorno que lo chequea constantemente y actúa en consecuencia para lograr un objetivo.

Este objetivo puede cambiar a medida que el elemento interactúa con el entorno, por ejemplo puede existir un auto virtual que su objetivo es moverse constantemente por el entorno. En un momento determinado encuentra un semáforo con la roja, tiene que cambiar su objetivo, ahora será detenerse hasta que cambie la luz del semáforo; cuando vuelva la verde, su objetivo volverá a ser moverse por el entorno. Este proceso que llevan a cabo los elementos autónomos puede dividirse en tres partes fundamentales [BUC05].

1. **Seleccionar la acción a realizar:** Se encarga de encontrar la meta o decidir el objetivo del elemento. Es la encargada de decir "vete para allá", "Haz esto y después haz esto otro".

2. **Calcular la fuerza y la dirección de la misma:** Se encarga de calcular la trayectoria requerida para satisfacer la selección de la meta u objetivo seleccionado. *Steering behaviors.*

3. **Movimiento:** Esta es la parte *mecánica* del movimiento, es decir, la que se encarga de como se va a mover el elemento en el entorno, aplica las leyes de la física para, a partir de la fuerza, llegar a los nuevos valores de las variables del agente.

Para seleccionar la acción a realizar se aplican las llamadas *metas*, que no es centro de las investigaciones en este momento. La segunda parte del proceso es la base de la propuesta realizada por este trabajo, el cálculo de la fuerza y su dirección para ser aplicada a los autos en un entorno de ciudad. También se muestra la manera de actualizar los valores del elemento mediante el cálculo de valores físicos, como la aceleración, la velocidad y la posición del elemento en el entorno, fundamentalmente.

Todas estas ideas han sido abordadas en varios trabajos desde que en 1987 Craig Reynols publicó un primer artículo donde crea un modelo de aves usando una técnica nueva hasta ese momento los *Steering behaviors* [REY87] que para simplicidad se le denominará por este autor en lo sucesivo 'Comportamientos'.

A partir de ese momento, esa técnica, se empezó a utilizar en tres grandes campos, la robótica, la inteligencia artificial y en el área de la vida artificial. Estas dos últimas se entrelazan formando un gran campo de acción en aplicaciones de realidad virtual en los que es muy utilizada en la actualidad.

Algunos de los trabajos más importantes que han sido presentados aplicando esta técnica y relacionados con autos, son el de Gary Ridsdale [RID87] quien creó elementos capaces de improvisar movimientos complejos evadiendo obstáculos estáticos y otros elementos móviles. En 1990 Jane Wilhelms y Robert Skinner [WIL90] investigan una arquitectura para elementos que tengan comportamientos similares a vehículos.

Thalmann creó animaciones de comportamientos de elementos que navegaban bajo corredores y bordeando obstáculos [THA90]. Michiel van de Panne trabajó sobre controladores para tareas como el aparcamiento de automóviles usando la búsqueda de espacios de estado. [VAN90].

En [BLU94] Bruce Blumberg describe un detallado mecanismo para la selección de acciones complejas. James Cremer y otros colegas crearon chóferes autónomos que sirvieran como 'extras' creando un ambiente de tráfico interactivo en un simulador de conducción de automóviles [CRE96]. Para ello hacían uso de una máquina de estados finita jerárquica. Posteriormente presentan otro articulo [CRE97] en el que la máquina de estados finita es usada para generar escenarios interesantes de tráfico compuesto por microscópicos vehículos autónomos simulados. Sukthankar [SUK97] introduce métodos para controlar comportamientos tácticos de vehículos en autopistas, centrándose en la selección de las acciones a realizar en cada instante de tiempo.

El trabajo más reciente sobre estos temas es un artículo publicado en el 2004 [WAN04] y perfeccionado en el 2005 [WAN05] por 4 importantes investigadores de estos temas dentro de los que se encuentra James Cremer ya citado anteriormente y aborda la necesidad de tener 5 comportamientos que se pueden aplicar a autos en entornos urbanos, centrándose en los comportamientos necesarios para las intercepciones entre calles en una ciudad. Los divide en modificadores de la aceleración y modificadores de ángulo de giro.

Es importante señalar que no se ha publicado ningún trabajo que muestre o proponga los comportamientos a usar en cada situación para el control de tráfico en un entorno de ciudad, cómo vincular cada uno de ellos y cómo relacionarlos con otras técnicas para lograr un ambiente realista y eficiente, tarea que intenta realizar el autor de esta memoria.

Desarrollo de la propuesta.

Siguiendo cada artículo se puede apreciar el desarrollo y el aumento de la calidad y profundidad con que se abordan los temas por el autor.

En el primer artículo [ANEXO 1] se comienza hablando de los comportamientos de las entidades en un simulador, la posibilidad de mezclar las técnicas deterministas y no deterministas para lograr el movimiento que se desea, además de hacer referencia a la búsqueda de caminos, encontrar hacia donde moverse acercándose o alejándose de otro elemento del entorno. Además muestra la necesidad de evadir obstáculos, que incluso pueden ser móviles, Todo esto es la base para formalizar en siguientes artículos la manera computacional de lograrlo.

Más concretamente y llegando a hacer referencia al primer trabajo presentado sobre la temática de los comportamientos por su creador, se habla de los movimiento grupales para aplicarlos a animales u otras entidades en el entorno del simulador.

Se menciona el uso de una máquina finita de estados, pero con el uso de probabilidades, para lograr un movimiento adecuado y realista en el entorno.

El siguiente artículo [ANEXO 2] se enmarca en otras técnicas necesarias para poder aplicar la propuesta principal. Primeramente se hace referencia a la necesidad de manejar cada comportamiento aparte, y dividir la inteligencia del entorno, cuestión perfectamente lograda con la propuesta final del autor.

Los mensajes no se incorporan directamente como se propone en ese trabajo pero se logra la interrelación de los elementos de una manera eficiente con la aplicación de otras técnicas.

Perfectamente lograda fue la propuesta de variabilidad de acción mencionada en ese mismo artículo. Con el uso de los comportamientos, los valores se van modificando según las condiciones; en cada momento, cada auto tendrá sus propios valores permitiendo una gama de actuaciones diferentes entre todos los elementos del entorno.

Otra idea muy adecuada y lograda es la propuesta de mantener un intervalo de tiempo para actualizar los valores de cada auto, incluso pudiéndose variar ese tiempo en dependencia de la velocidad actual de cada auto. Tomando en cuenta que mientras más rápido vaya un auto más rápido tendrá que responder a las señales de tránsito, se realiza una relación entre la velocidad y el tiempo de

respuesta de cada auto. Esta es una idea completamente nueva y perfectamente lograda en los últimos trabajos del autor.

De la misma manera se logra la interrelación entre los autos y entre los autos y las señales que más cercanas están entre sí y no entre todos los elementos del entorno.

El último de los trabajos presentados por el autor [ANEXO 4], presentado como ponencia en el XV FORUM de Ciencia y técnica, es la consolidación e integración de cada una de las ideas explicadas en los artículos precedentes. Comienza con una breve panorámica sobre la temática de los comportamientos y una muestra de los más recientes artículos vinculados al desarrollo de autos en entornos urbanos y a la determinación de las señales de tránsito a respetar por los autos del entorno. Deja claro además como una de las propuestas innovadoras del autor, la división del entorno en áreas que no cubren todo el entorno y pueden estar superpuestas una a la otra para lograr el chequeo eficiente del entorno.

Un aparte tiene en dicho trabajo la detección de infracciones del conductor como también lo tuvo este trabajo (ver epígrafe 1).

La ponencia continúa introduciendo el modelo físico-matemático de los autos, los atributos de la clase correspondiente y las principales fórmulas físicas utilizadas para lograr el objetivo de cada elemento. Se muestra un ejemplo de código de las principales partes de esta clase.

Seguidamente se explica el funcionamiento de los comportamientos, con ejemplos concretos de algunos de ellos y segmentos de código para ayudar a la comprensión de los mismos. Muy importante es el sub-epígrafe dedicado a la combinación de los comportamientos, este es otro de los aportes del autor, proponiendo una manera de combinar los comportamientos según su importancia. Se hace una diferencia entre comportamientos prioritarios y no prioritarios, se analizan los no prioritarios solo si los prioritarios no incorporan alguna fuerza. Dentro de cada uno de estos conjuntos hay un orden que se debe seguir para analizar cada comportamiento, los cuales se van analizando solo si no se ha cubierto la máxima fuerza posible a aplicar a los elementos, cuando

esto ocurre dejan de analizarse los comportamientos, logrando eficiencia en la propuesta. Se debe aclarar que esto no afecta el realismo del entorno ya que como se realiza este análisis a cada instante, en un tiempo relativamente pequeño se analizan tanto los comportamientos prioritarios como los no prioritarios. Un cuidado especial requiere determinar estos grupos de comportamientos, ya que un comportamiento mal ubicado puede llevar a resultados no deseados de los elementos del entorno.

El principal aporte del autor está en los dos siguientes epígrafes donde describe la manera de incorporar los comportamientos a los autos virtuales del simulador vinculando para esto otras técnicas que se aplican en otras áreas del mismo simulador y otras que no se han aplicado con anterioridad.

Los principales aspectos propuestos, son la manera de tratar las señales de tránsito como objetos que interactúan directamente con los autos, este aspecto es completamente innovador en entornos de este tipo. La interrelación entre el conductor (usuario del simulador) y los autos es completamente nueva, como se había dicho no existe bibliografía concreta de cómo realizar un simulador donde estos dos personajes diferentes y a la vez tan semejantes interactúan de manera tan peculiar. Además muy importante es la vinculación y la propuesta realizada sobre la incorporación de técnicas de optimización a la ya descrita de los comportamientos de los elementos, no existe bibliografía sobre la vinculación de estas técnicas al desarrollo de la Inteligencia artificial de los elementos virtuales. Se logra así la consolidación de un paquete potente y eficiente para lograr el objetivo propuesto.

Comportamientos para entornos de ciudad.

A continuación se muestra una propuesta concreta de algunos de los comportamientos más importantes a incorporar a los autos en determinadas situaciones. En la implementación de esta propuesta se trabaja en la actualidad.

Los comportamientos generales que deben tener los autos son:

Control de la aceleración: Permite mantener una aceleración adecuada en cada instante aunque no estén actuando fuerzas en el auto.

Seguir al líder: No siempre está activo pero sí hay que chequear cada momento si algún auto que vaya delante tiene menos velocidad, en ese momento se activa para seguir al que va delante, garantizando no chocar con él, puede incluso llegar a detenerse si el 'líder' lo hace.

Evadir obstáculos: Mantiene un chequeo constante del entorno, garantizando no aproximarse demasiado a la acera o a otro tipo de obstáculos que se pueda incorporar al entorno. Debe llevar incorporada la posibilidad de mantener la senda derecha de la calle, si todas son de dos vías; si en el entorno hay calles de distinta cantidad de vías se debe tener un comportamiento aparte para controlar este aspecto.

Otros comportamientos que se activan en cierto momento son:

Adelantar: Se encarga de tomar decisiones de adelantar o no a un auto que va delante, dejaría de analizar el seguir al líder e incorpora este.

Toma de decisiones: Analiza cuando está próximo a una esquina que dirección tomar.

Para respetar cada una de las señales de tránsito muchas veces se necesita incorporar más de un comportamiento a los autos, a continuación se muestran algunas de las señales y los comportamientos necesarios a incorporar a cada auto y en qué momento hacerlo.

Pare:

Detenerse: Detiene al auto en el punto señalado si no hay autos en el medio, de haberlo incorpora "seguir al líder", que ya fue explicado. Es importante aclarar que mantiene el comportamiento 'detenerse' hasta que lo hace en el lugar establecido.

Intercepción: es activado cuando el auto está detenido, permite analizar si tiene derecho de vía. Para esto analiza en las rejillas cercanas si se acercan autos.

Seda el paso:

Intercepción: Ya fue explicado, lo activa desde que se acerca, si no tiene derecho de vía para en el lugar establecido, si no continúa.

Semáforo:

Similar al seda el paso pero analiza la luz y no si vienen autos.

Velocidad máxima limitada:

Reducción de velocidad máxima: Reduce ese valor temporalmente. Para esto tiene que guardar el valor original para restablecerlo al desactivar el comportamiento.

Estos son los principales comportamientos que se incorporan en cada momento, es justo señalar que cada uno de ellos lleva un análisis matemático para lograr el resultado esperado en cada instante de tiempo.

CONCLUSIONES

El empleo de los comportamientos en elementos dinámicos, es fundamental en la actualidad para lograr entornos realistas y eficientes, por demás es una técnica relativamente fácil de implementar y de entender, siendo la base de la investigación llevada a cabo por el autor.

Con esta colección de trabajos se logra dar una panorámica general de cómo modelar todo un sistema de control de tráfico para un simulador de conducción de auto. Mostrando variantes para la optimización de la propuesta.

La memoria recoge un análisis crítico de cada uno de los trabajos haciéndose énfasis en los aspectos más importantes y en el aporte del autor en cada uno de ellos.

Se incorpora como novedad a la memoria una propuesta de comportamientos a incorporar a los autos en determinadas situaciones.

RECOMENDACIONES

Perfeccionar cada uno de los comportamientos de los autos.

Publicar algunos trabajos que muestren el procesamiento matemático que requieren los comportamientos.

Implementar e incorporar al simulador el módulo de movimiento autónomo de los autos.

Desarrollar comportamientos de autos en nuevos escenarios, como pistas de carreras y entornos abiertos.

Trabajar en la tercera etapa del proyecto, la incorporación de vida a una ciudad virtual.

REFERENCIAS:

[BLU95] Blumberg B. amd T. Galyean. *Multi-level direction of autonomous creatures for real-time virtual environment.* En ACM Siggraph, pages 47–54, Aug 1995.

[BUC05] Buckland. Mat. *Programming Game AI by Example* Wordware Publishing 2005.

[COC05] Coca Bergolla, Yuniesky. *Simulador de auto inteligente.* Conferencia Científica de la Universidad de las Ciencias Informáticas. UCiencia 2005.

[COC07a] Coca Bergolla, Yuniesky. *Algunas ideas a tomar en cuenta para la inteligencia de un simulador de auto* Evento Virtual de la Feria Internacional Informática 2007.

[COC07b] Coca Bergolla, Yuniesky. *Control del tránsito en un simulador de conducción de auto.* XV FORUM de Ciencia y Técnica. Universidad de las Ciencias Informáticas. 2007.

[CRE95]J. Cremer, J. Kearney, and Y. Papelis. *Hcsm: A framework for behavior and scenario control in virtual environments.* ACM Transactions on Modeling and Computer Animation, (3), 7 1995.

[CRE97] J. Cremer, J. Kearney, and P. Willemsen. *Directable behavior models for virtual driving scenarios.* Transactions of the society for computer simulation international, (2), 6 1997.

[LOR07] Lores Caignet, Luís F. Marty Consuegra, Yunior M. *"Detección de infracciones del conductor en un simulador de auto".* Trabajo para optar por el título de Ingeniero en Ciencias informáticas. Universidad de las Ciencias Informáticas. Ciudad de La Habana, 2007.

[REY87] Reynolds, C. W. Flocks, Herds, and Schools: *A Distributed Behavioral Model.* Computer Graphics. 1987

[RID87] Ridsdale, Gary. *The Director's Apprentice: Animating Figures in a Constrained Environment*, Ph.D. thesis, School of Computing Science, Simon Fraser University. 1987

[SUK97] R. Sukthankar. *Situation Awareness for Tactical Driving.* PhD thesis, Robotics Institute, Carnegie Mellon University, 1997.

[THA90] Thalmann, Daniel; Renault, Olivier and Magnenat-Thalmann Nadia *A Vision-Based Approach to Behavioral Animation*, Journal of Visualization and Computer Animation, John Wiley & Sons, 1(1) pages 18-21. 1990

[VAN90] van de Panne, M., Fiume, E., and Vranesic, Z. G., *Reusable Motion Synthesis Using State-Space Controllers*, Proceedings of SIGGRAPH `90, In Computer Graphics Proceedings, ACM SIGGRAPH, pages 225-234. 1990

[WAN04] Wang. Hongling, Kearney. Joseph K., Cremer. James, Willemsen. Peter. *Steering Autonomous Driving Agents Through Intersections in Virtual Urban Environments.* International Conference on Modeling, Simulation and Visualization Methods (MSV'04). 2004

[WAN05] Wang. Hongling, Kearney. Joseph K., Cremer. James, Willemsen. Peter. *Steering Behaviors for Autonomous Vehicles in Virtual Environments.* Proceedings of the IEEE Virtual Reality 2005 (VR'05).

[WIL90] Wilhelms, Jane and Skinner, Robert *A Notion for Interactive Behavioral Animation Control*, IEEE Computer Graphics and Applications, 10(3), pages 14-22. 1990.

Anexo 1.

[COC05] Coca Bergolla, Yuniesky. *Simulador de auto inteligente.* Conferencia Científica de la Universidad de las Ciencias Informáticas. UCiencia 2005.

¿Simulador de auto, inteligente?

Yuniesky Coca Bergolla

Grupo SIMPRO, ycoca@uci.cu , Universidad de Ciencias Informáticas, Carretera San Antonio Km 2 ½.La Habana, Cuba

Resumen:

La historia de los simuladores en Cuba apenas comienza y el uso de técnicas de la Inteligencia Artificial es un punto clave a investigar en el desarrollo de este tema. Pretendemos con este trabajo dar una breve panorámica de cómo usar algunas de estas técnicas en función de un simulador de auto que ya se desarrolla entre la empresa SIMPRO y la Universidad de las Ciencias Informáticas.

Abstract:

The history simulator's in Cuba is just beginning, and the application of Artificial Intelligence techniques is essential in investigating any development in this field. With this work we intend to give an overall view on the implementation of some techniques when used in a car simulator; presently in the phase of production between SIMPRO Enterprise and the University of Informatics Sciences.

Introducción.

La Inteligencia Artificial (IA) en los simuladores se encuentra de varias formas, son múltiples las estrategias o vías para hacer de ellos sistemas verdaderamente creíbles y cercanos cada vez más a la realidad. Las técnicas pueden ir desde un simple sistema basado en reglas o una máquina finita de estados, hasta sistemas de redes neuronales o algoritmos genéticos. Es importante aclarar que en simuladores del tipo que nos mueve, la velocidad en la ejecución es muy importante, por lo que la búsqueda de algoritmos rápidos es la primicia, sin perder la necesidad de hacer que los personajes de nuestro sistema se comporten de forma inteligente. Los sistemas que trabajan en tiempo real requieren de las técnicas más avanzadas en cuanto a eficiencia, es decir, lograr un comportamiento inteligente en fracciones de segundo.

Los juegos son muy similares a los simuladores y su historia es mucho más rica que la de estos últimos, es por eso que incluso la bibliografía se refiere siempre a la IA para juegos y no para simuladores, se requeriría de un trabajo investigativo solo para comparar las formas de ver la IA desde los dos puntos de vista. Nuestro trabajo esta dirigido al uso de algoritmos ampliamente empleados en juegos, para la construcción del simulador de auto.

Debemos tomar en cuenta que los simuladores son programas con muy largo período de desarrollo y una complejidad extrema. Reúnen los elementos más complejos de

algunos productos sobre todo de algunos juegos: La compleja inteligencia artificial de los juegos de estrategia, la calidad gráfica de los juegos de acción, la potencia de cálculo que requiere un juego de ajedrez, la espectacularidad de un juego deportivo, la interactividad de una aventura gráfica y la profundidad en reglas y organización de un juego de rol.

1. Los inicios.

Los simuladores nacieron con los primeros sistemas visuales por computadoras, ya en los años ochenta podíamos encontrar versiones de simuladores civiles con algún nivel de 'inteligencia'.

Desde esos años en el mundo se viene trabajando en esta dirección, sin embargo no se han logrado grandes avances en cuanto a simuladores de auto, los más avanzados han sido los de vuelo.

A continuación mostramos una pequeña cronología de los hechos más importantes en cuanto a simuladores se refiere. [Dar03] [Cam04]

–**Morton Heilig**, "**Sensorama**", 1962: Recorrido simulado en motocicleta por N.Y. Tuvo todos los elementos de la Realidad Virtual excepto la interactividad,es decir, viento, vibración, película estéreo, sonido estéreo y olores. –**Ivan Sutherland**, "**Theultimatedisplay**", 1965: Escribe sobre la interactividad,los dispositivos de realimentación y efectores sensoriales. –**Ivan Sutherland**, "**AHead Mounted 3D Display**", 1968: Describe un dispositivo de representación estéreo con dos pantallas que superponían imágenes del mundo real con imágenes por computador. –**VIVED** (*Virtual Visual Environment Workstation*, NASA), 1984. Sistema para astronautas. –**VIEW** (*Virtual Interactive Environment Workstation*, NASA), 1986. Sistema completo para astronautas.
-**Microprose Falcon 4.0,** 1998. Es el más realista y el que proyecta un futuro más prometedor. En cuanto a complejidad basta decir que el manual de vuelo del F16 real y el del simulador son el mismo.
-**CFS2** (Microsoft Combat Flight Simulator 2), 2000. Es un simulador muydivertido y ameno.
-**LOMAC** (Lock On: Modern Air Combat), 2003. La joya del combate aéreo moderno.

Como podemos ver muchos de estos simuladores se hacen con el objetivo más que instructivo de entretener y la mayoría se inclinan por los simuladores de avión ya que son más atractivos al usuario.

En Cuba, una de las empresas que ha impulsado el desarrollo de los simuladores, es el "Centro de Investigación y Desarrollo #2" (CID2), conocido en el mercado como **SIMPRO** (Simuladores Profesionales). Esta empresa comenzó como proyecto desde el año 1994, y oficialmente existe con el nombre de SIMPRO desde el año 2000. Cuenta con numerosos premios a las investigaciones y calidad de sus productos desde su fundación, entre ellos:

Premio de la Academia de Ciencias a las investigaciones más destacadas del año en dos ocasiones. (1996, 1998)

Dos premios relevantes en Forum Nacional de Ciencia y Técnica. (1996, 1998)

Premios en la Feria Internacional del Transporte a la calidad del producto. (1999)

En este momento está enfrascada junto a la UCI en la construcción de un simulador de auto, ya se tiene una primera versión y se trabaja en la perfección del mismo, que incluye dotar de un comportamiento 'inteligente' a los elementos que intervienen en su entorno virtual.

2. Las técnicas.

Tanto personas como animales y entidades no vivas requieren de un movimiento lógico en nuestro mundo. Las personas tienen que lograr diversos comportamientos que puede encontrar el conductor en su vida como chofer. Tanto los conservadores que se cuidan de no tener un accidente, hasta los irresponsables que pueden provocar accidentes del transito, fatales para la vida; pasando por algún descuidado que trate de cruzar una calle sin darse cuenta que está el semáforo con la verde o algún niño que va tras una pelota, cuando jugaba en una acera.

Los animales pueden jugar un papel importante en el sistema de simulación de auto, desde una bandada de aves que pueden disociar al chofer, hasta un pequeño gorrión que pasa frente al parabrisas o un perro que intenta cruzar la calle.

En nuestro mundo también entidades no vivas tienen que tener comportamiento lógico, Los autos que circulan en la ciudad, no nos podemos quedar con un movimiento autómata, tenemos que lograr que representen lo que realmente ocurre en la vida de una ciudad, aglomeración en un semáforo, posibles infracciones de autos.

No existe actualmente algún sistema reconocido tanto de simuladores como de juegos, que no tenga elemento combinados de técnicas deterministas y técnicas no deterministas.

Las técnicas deterministas son aquellas que su comportamiento es especificado y predecible. Este tipo de técnicas son esenciales en sistemas que trabajan en tiempo real. Son técnicas rápidas en su ejecución y fáciles de implementar, de entender y de comprobar su efectividad. Sin embargo a pesar de ser muy importantes, no satisfacen todos los requerimientos para los simuladores, ya que el comportamiento es automático, es decir, no reproducen lo que sucede realmente en la vida, sino lo que el desarrollador especificó que sucediera en cada situación. [Bou04]

Las no deterministas son las que su comportamiento es, hasta cierto punto, impredecible, lógicamente esto depende del método usado. Estas técnicas permiten el aprendizaje de los elementos que intervienen en el mundo y movimientos impredecibles y diferentes para cada vez que se ejecuta una misma situación. Quiere esto decir que no se tiene de antemano codificado explícitamente el comportamiento de cada personaje en una situación determinada como ocurre con las técnicas deterministas. [Bou04] Un problema de esta técnica es que es muy difícil de probar. Imaginen probar todas las posibles variantes de acción de un personaje o tratar de saber cuando haría una determinada acción.

Como podemos ver un buen sistema debe estar dotado de ambas técnicas, por lo

regular se aplican las no deterministas en casos específicos que realmente requieran de una inteligencia autónoma, pero en la mayoría de las situaciones se aplican técnicas deterministas, la principal cuestión es encontrar ese punto; o sea, cuando usar cada una.

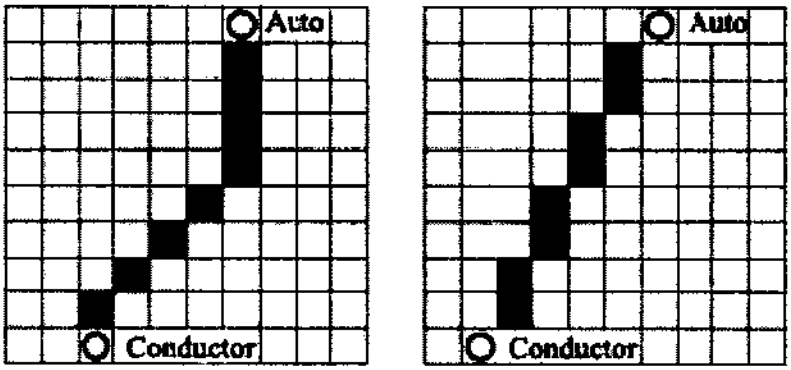
3. IA contra Situaciones.

Algunos desarrolladores de juegos toman la búsqueda de caminos como parte de la inteligencia artificial, otros autores no lo consideran de la misma forma; pero sin lugar a dudas todo comienza con un movimiento adecuado de cada elemento que participa en el mundo que nos encontramos, ya sea para llegar a un lugar o para alejarse de él.

Existen algoritmos muy sencillos de búsqueda de trayectorias para personajes que intervienen en escena. No hacen más que encontrar la mejor forma de llegar a un lugar determinado a partir de una posición inicial. Hay varias variantes posibles en las que se puede ver involucrado un personaje.

Supongamos un peatón que desea llegar a un auto que esta estacionado. Según Pitágoras el camino más cerca entre dos puntos es la línea recta que los une, por tanto ese seria nuestro objetivo final, tratar de llegar directamente hasta nuestro objetivo o de la forma más lineal posible. Existen varias vías para solucionar este problema, los algoritmos pueden variar y se han ido modificando para encontrar la solución más eficiente para este problema. [Bou04].

La **Figura 1** muestra las dos formas más sencillas (computacionalmente hablando) de llegar desde el peatón hasta el auto. La profundización en el código, ventajas y desventajas se verán en trabajos posteriores.



a b

Figura 1.

a) Búsqueda simple
b) Búsqueda en línea de foco.

Puede suceder también que el auto al que queremos llegar esté en movimiento y el peatón sabe que se va a detener en algún lugar más adelante de la cuadra, pero el lugar

es indeterminado en ese momento. Si seguimos la misma estrategia entonces tenemos que estar calculando constantemente la posición del auto para mover al personaje hasta él.

Otra variante que puede ser más eficiente, es tratar de interceptar al auto, para esto como es lógico necesitamos calcular la velocidad con que se mueve y sabiendo además la del peatón pues lograríamos hacer un cálculo del punto exacto donde interceptarlo, de seguir tanto el auto como la persona a la misma velocidad de traslación.

Pero estos algoritmos pueden complicarse, podemos tener obstáculos para llegar a nuestro destino, obstáculos que pueden estar estáticos o pueden estar en movimiento. [Bou04]. Ejemplos claros de esto es que el peatón tenga el auto parqueado en la otra calle de la manzana y deba cruzar a través de los edificios,
o que halla un poste de teléfono en la acera. O que esté en el otro lado de la calle y tenga que cruzar para llegar a su destino o simplemente pueden existir otras personas que se mueven por la misma acera.

Los algoritmos A* son actualmente los más usados en este tipo de problemas de búsqueda de caminos; pero no siempre genera buenas soluciones, depende del tipo de problema que se quiera solucionar.

Lo que hace que los algoritmos A* sean los más usados es que garantizan la mejor solución entre un punto de partida y uno de llegada, siempre y cuando exista ese camino. Es relativamente eficiente, pero para algunos casos puede convertirse en muy costoso, pudiera ser por ejemplo cuando se busca un camino entre dos puntos que no tienen obstáculos, en ese caso es mucho más rápido un simple algoritmo de movimiento directo como ya comentábamos anteriormente. Además cuando se tienen que encontrar caminos de varios elementos a la vez se hace muy costosa la corrida del algoritmo. Otro problema del algoritmo es que a principiantes en el tema, les cuesta mucho trabajo entender la forma en que trabaja.

Otra de las situaciones que se nos presentan en los simuladores, son los comportamientos de movimiento de grupos [Cra87], o sea, un grupo de personas a cruzar una calle, una fila de carros que se siguen en una calle, un grupo de autos al salir de un semáforo. Etc. Para dar credibilidad y realismo al simulador podemos usar esta estrategia, por ejemplo en una emigración de aves que pasa por el lugar, eso puede incluso disociar al conductor y llevarlo a descuidar su trabajo. Un modo de comprobar la eficiencia del conductor es poniéndole un obstáculo o una situación difícil en ese mismo momento.

Lo más impresionante es que este comportamiento se logra con tres sencillas reglas: Cohesión, alineación y separación entre cada unidad y sus vecinos. O sea, calcular estas tres variables para cada elemento tomando en cuenta algunos de los demás elementos del conjunto, en la **figura 2** mostramos que objetos se analizarían para un determinado elemento **E0**.

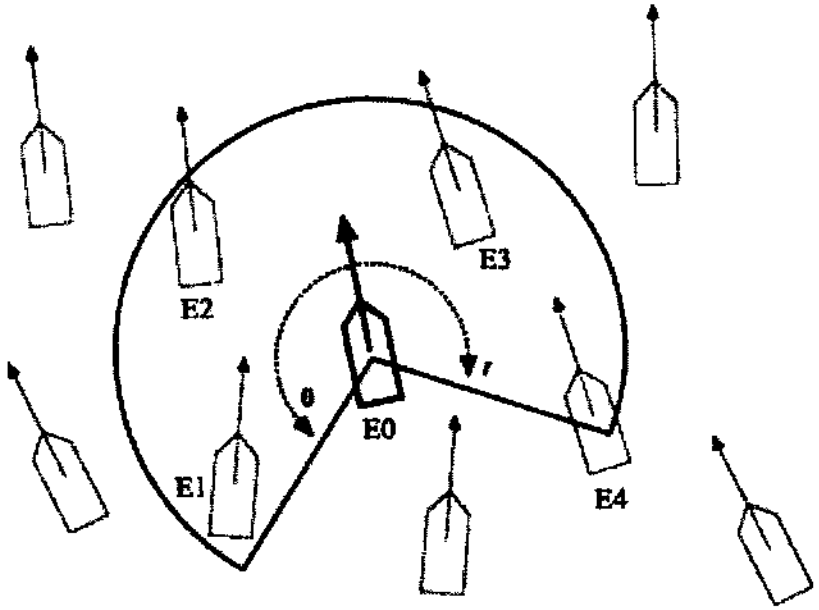


Figura 2

La física siempre ha estado vinculada a las demás ramas de la ciencia. En la computación es sin dudas un puntal para el desarrollo de varios tareas y si se trata de la simulación de procesos es imprescindible acercarnos a ella para la solución de problemas. La función potencial Lenard-Jones [Bou04] es usada en varias implementaciones de juegos y nos puede ser muy útil en la solución de problemas para nuestro simulador. La fórmula, que es muy sencilla, le asigna a cada elemento un valor de cada una de sus variables y a través de esos valores se les da una distribución en la escena. O sea, como en la física, sobre los elementos actúan la fuerza de atracción y la de repulsión según la cercanía entre ellos. La función es:

$$U = -\frac{A}{r^n} + \frac{B}{r^m}$$

'**U**' representa la energía potencial que es inversamente proporcional a la separación '**r**' entre los elementos. '**A**' y '**B**' son parámetros al igual que '**n**' y '**m**'. La parte negativa puede representar en nuestro sistema la fuerza de repulsión y la parte positiva la fuerza de atracción. Los parámetros '**A**' y '**B**' podemos tomarlos como el valor exacto de cada fuerza y los valores de '**n**' y '**m**' la atenuación de estas fuerzas. ¿Como verlo en la práctica? Es indudable que si tenemos un auto en movimiento una persona no debe acercarse demasiado, al igual que un auto en movimiento no puede acercarse a un peatón, los autos mientras transitan deben guardar una distancia entre ellos etc. Estos problemas son fácilmente solubles mediante esta fórmula.

Otra de las herramientas de inteligencia artificial muy usada en estos sistemas son las máquinas de estado. Estas a través de un conjunto de estados y de funciones de transición manejan el comportamiento de una entidad en un sistema. Esta herramienta

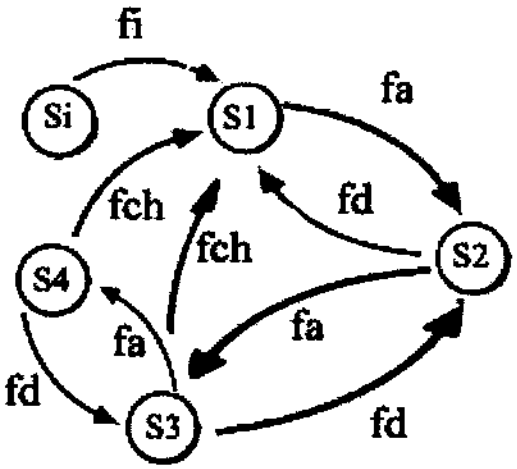
puede ser usada, por ejemplo, en la detección de infracciones del conductor. Tomamos cada estado del usuario que se evalúa (Posición del carro) como momentos posibles en su evaluación. Los estados pueden ser: Detenido, Mov_correcto, Mov_cometiendo_infracción. La transición a cada uno está determinada por una cantidad de variables que se analizan dentro de cada función de transición. Pero para lograr un resultado eficiente debemos hacer uso de otra de las herramientas de la inteligencia artificial, la lógica difusa.

Esta técnica se encarga de representar computacionalmente algunos problemas de la misma manera que los soluciona el hombre.

En varias ocasiones se resuelven problemas de una manera imprecisa, esto sucede si no se tiene toda la información necesaria o se quiere hallar una solución rápida sin entrar en muchos detalles que complicarían en nuestro caso el trabajo y por ende demorarían el sistema. Otro ejemplo puede ser a la hora de trabajar la velocidad del auto, la podemos tomar como 'cero', 'lenta', 'rápida', 'muy_rápida'. Teniendo en cuenta estos valores podemos solucionar nuestro problema. El uso de las reglas combinado con esta estrategia puede hacer que el sistema funcione mucho más eficiente, tomando en cuenta que no siempre los sistemas basados en reglas tienen que comparar valores enumerativos, enteros o booleanos, sino que con el uso de cadenas podemos incluso permitir la facilidad y posibilidad de incrementar nuevas reglas y adicionarlas al sistema sin mucho esfuerzo de conversión de datos. Como vemos estamos combinando las máquinas de estado con la lógica difusa y las reglas, para dar solución a nuestro problema. **Figura 3.**

Esta combinación será usada sobre todo para la búsqueda de infracciones del conductor en el sistema. Infracciones del tránsito que pueden llevarlo a suspender el examen.

Este tema es el más apremiante en el simulador actualmente por lo que es prioridad una profundización en el mismo.



Si: Estado inicial: Auto apagado.
S1: Velocidad cero
S2: Lento.
S3: Rápido
S4: Muy rápido.

fi: Función de arranque
fa: Función de aceleración.
fd: Función de desaceleración
fch: Función de choque..

Figura 3.

Estamos hablando de un simulador de autos donde, como ustedes pueden suponer, el tránsito no es algo caótico, pero si es impredecible el movimiento de cada entidad, tanto las personas como los autos. Para hacer del sistema un conjunto inteligente, no siempre que se evalúa un usuario puede aparecer una misma secuencia de movimientos para cada elemento. Entramos en el mundo de las probabilidades, y los movimientos aleatorios.

Es imprescindible el uso de estas técnicas para hacer creíble y no caducable nuestro sistema, suponiendo que el usuario desaprueba en su primera oportunidad, cuando venga a la segunda, sin lugar a dudas, va a encontrar las mismas situaciones con las mismas características y ya puede estar preparado para solucionar la situación. En consecuencia nuestro sistema no cumpliría el objetivo de evaluar la verdadera creatividad y conocimientos del usuario.

A partir de las probabilidades nuestro sistema puede incluso 'aprender', supongamos que vamos tomando las infracciones que va cometiendo el usuario y las relacionamos con algunas situaciones previas. Ejemplos: Se lleva un semáforo cuando acaba de adelantar a un auto, Viola las señales del tránsito (Cualquiera de ellas) cuando hay elementos que lo disocien, por ejemplo personas tratando de cruzar la calle, autos de frente, algún auto que lo intenta adelantar, alguna violación de otro auto. Tomando estadísticas podremos sacar la probabilidad que tiene de hacer una infracción en un momento determinado y a partir de ahí podremos insistir en ciertas situaciones.

Conclusiones:

Estas son solo algunas de las más sencillas formas de vincular algoritmos y estrategias de los juegos en el simulador de auto. Tanto las técnicas deterministas como las no deterministas han sido usadas en la solución de problemas reales del simulador de auto. La búsqueda simple de caminos, máquinas de estado, lógica difusa y el uso de reglas, las pudiéramos clasificadas como técnicas deterministas. Mientras la búsqueda de caminos por los algoritmos A*, los comportamientos de grupo, la función potencial y la búsqueda de probabilidades nos permiten darle un poco más de inteligencia autónoma a los bots. Existen varios algoritmos y técnicas que no se trataron en este artículo, y que sin dudas pueden ser incorporadas al simulador de auto. Hasta aquí solo hemos tratado de dar una panorámica del uso de estas técnicas, que históricamente han sido incorporadas a los juegos. Ahora pueden ser utilizadas, con pequeñas transformaciones, en la creación de simuladores profesionales, que tendrán un objetivo mucho más instructivo y serio.

Referencias:

[Dar03] "Darkness" Campomanes, Iñaki "Simuladores de vuelo: videojuegos por todo lo alto" -http://www.msdox.com/reportajes Octubre de 2003.

[Cam04] Camacho Román, Yanoski Rogelio, Jiménez López, Fernando. "Biblioteca Gráfica Para Sistemas de Realidad Virtual". Trabajo para optar por el Título de Ingeniería en Informática. La Habana. Julio de 2004.

[Bou04] Bourg, David M, Seeman, Glenn "AI for Game Developers" O'Reilly. Julio de 2004.

[Cra87] Craig W. Reynolds, **"Flocks, Herds, and Schools: A Distributed Behavioral Model"** *Computer Graphics*, **21**(4), Julio de 1987.

Bibliografía:

"Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem" Por C. L. Forgy, Artificial Intelligence, 1982.

http://www.gameai.com
http://www.aboutai.net
http://www.aaai.org/AITopics/html/expert.html
http://www.igda.org/ai/
http://www.red3d.com/cwr/papers/1987/boids.html
http://www.dedalus-software.com.ar/topic-tutoriales.php
http://ai-depot.com/Features/Tutorials.html.

Anexo 2.

[COC07a] Coca Bergolla, Yuniesky. *Algunas ideas a tomar en cuenta para la inteligencia de un simulador de auto* Evento Virtual de la Feria Internacional Informática 2007.

"Algunas ideas a tomar en cuenta para la inteligencia de un simulador de auto."

Autor: Lic. Yuniesky Coca Bergolla.
ycoca@uci.cu

Universidad de las Ciencias Informáticas.
La Habana Cuba
Junio 2006.

Resumen.

Los simuladores han sido de las primeras líneas investigativas y productivas de la Universidad de las Ciencias Informáticas. Lograr un simulador competente en el mercado mundial es nuestro objetivo principal. Para ello debe lograr una interacción inteligente con el usuario. El simulador de auto es el más avanzado en este sentido en nuestra universidad, sin embargo le falta realidad al entorno virtual, aplicar la IA mínima que logre la credibilidad y calidad de respuesta del entorno virtual es nuestro objetivo principal. Este trabajo propone algunas de las ideas fundamentales, que se pueden generalizar, para incorporar inteligencia, de forma eficiente, a un simulador de autos.

Introducción.

Nuestro país está envuelto en la reanimación de la economía que fue fuertemente afectada con el período especial. Una de las estrategias fundamentales es potenciar el desarrollo informático y de la computación de forma general. Llegar a convertirnos en una potencia en la construcción de software. Nuestra Universidad de las Ciencias Informáticas está enmarcada en esta estrategia y dentro de ella los simuladores fueron una de las principales líneas de investigación y producción. El simulador de auto SIMPRO, ya ha alcanzado prestigio y aceptación dentro y fuera del país. Sin embargo está lejos de ser el simulador que aspiramos. Con la convicción de esto y el espíritu de superación, proponemos este trabajo para brindar las ideas esenciales para la incorporación de "Inteligencia" a este simulador, que perfectamente pudiera ser incrementada de manera sencilla y generalizada para otros simuladores en entornos urbanos fundamentalmente.

En la bibliografía existente no se habla de las características concretas de la implementación de los simuladores. Sin embargo los juegos han tomado un lugar preferencial en las publicaciones de los expertos en estos temas. Sin ser lo mismo, la relación existente entre los simuladores y los juegos es mucha, se crean entornos virtuales en los que se simula la realidad, los primeros con un objetivo recreativo, los segundos, por lo general, con fines sociales o incluso con fines militares y de entrenamiento. Por lo tanto estos últimos deben ser mucho más "serios". Sin embargo las técnicas a usar son prácticamente las mismas. Se intenta diseñar un entorno 3D lo más cercano a la realidad, con texturas y efectos visuales que hagan adentrarse al usuario en un verdadero mundo *"real-virtual"*. Este punto juega un papel fundamental en lograr el objetivo final tanto en juegos como en simuladores, pero no es este el tema que vamos a tratar en este artículo, nos vamos a centrar en el diseño de los elementos inteligentes, llamados en la bibliografía para juegos *"non-player character"*, personajes no dirigidos por el jugador, es decir, controlados completamente por el sistema, y en buscar una interacción inteligente de todo ese entorno 3D, asumiendo que se ha logrado representar la realidad visual y animada con la calidad requerida.

Existen estudios para tratar de hacer módulos inteligentes genéricos, como los realizados por el grupo ***SOAR*** (http://sitemaker.umich.edu/soar). O por ***EngenuityTechnologies, Inc.*** (http://www.biographictech.com) es decir, que sean aplicables a todos los juegos que tengan una misma línea temática. Sin embargo los principales expertos expresan que eso es imposible, que cada entorno virtual que se intente construir debe incorporársele la inteligencia de manera particular [TOZ02]. De la misma manera que existen los expertos en cada rama de la ciencia o que incluso en un juego determinado hay jugadores que se especializan y siempre buscan un estilo determinado para jugar, así mismo se debe especificar o implementar para cada entorno virtual, cómo se van a crear las estrategias y modelos de IA. Sin embargo hay determinadas ideas que si son aplicables en todos los casos. Estudios y juegos ya creados que han demostrado su poder de inteligencia y credibilidad visual

e interactiva y de los cuales se pueden sacar varias ideas perfectamente aplicables y muy útiles para la construcción de un entorno virtual ya sea de un juego o un simulador o cualquier otra variante similar.

Desarrollo

Extensibilidad.

Lo primero que nos viene a la mente cuando pensamos en hacer inteligente nuestro entorno es implementar los elementos dinámicos como autómatas que sepan en cada momento qué tienen que hacer, es decir, que funcionen como una máquina de estado finita. Supongamos un carro en nuestro simulador de autos. Que en su método de actualización funcione como sigue:

```
void SP_Auto::ActzarIA(){

       ActualizarPosicion();

      for(int i = 0; i < cantAutos; i++)     Interactuar(GetAuto(i));

      int i_sennal = Determina_sennal();

      switch(i_sennal)
      case: PARE{  EjecutarPare(); break;
      }
      case: SEDAPASO{  EjecutarSedaPaso(); break;
      }
      case: SEMAFORO{  EjecutarSemaforo(); break;
      }

//.................................//


}
```

En el ejemplo se comienza por actualizar la posición del carro en el entorno, luego se hace un ciclo para analizar con cada auto si existe algún tipo de interacción, como puede ser prevenir un choque, adelantarlo o cuestiones semejantes. Por último llamamos a un método que retorne que señal de tránsito debo analizar, es decir cual tengo próxima a mi posición, con este resultado ejecuto lo necesario para cada caso.

De la misma manera pudiéramos adicionarle a este código la interacción con otros tipos de elementos ya sea estáticos o dinámicos, obstáculos en el camino, personas que cruzan la calle, etc. simplemente adicionando código para cada tipo de elemento y como interactuar con el mismo.

Así logramos un buen avance, con código sencillo, fácil de implementar y de entender logramos el primer paso, pero ¿Qué sucede si en un determinado momento deseamos incorporar nuevos elementos al entorno que ya teníamos creado, como una nueva señal de tránsito por ejemplo? Tenemos que volver a implementar cada elemento que teníamos a punto, además si a cada elemento dinámico le implementamos, como se muestra en el ejemplo, qué hacer con cada elemento estático, (una piedra en la calle, un semáforo, un pare), puede sobrecargarse la implementación de su IA, haciéndose más compleja e ilegible. Con este ejemplo además estamos analizando solo un elemento estático a la vez, quiere decir que dos

elementos no serían analizados en un mismo instante de tiempo, habría que esperar al siguiente frame para tratar de encontrarlo, si además se le dice de alguna manera que ya fue analizada cierta señal de tránsito y que aunque siga estando en el radio de acción del auto ya no hay que analizarla. Sin embargo si logramos que el *mundo contenga la IA*, que cada elemento aunque no sea dinámico tenga su propia información y sepa qué tiene que lograr en los demás elementos cuando interactúe con algún elemento dinámico es decir que sepa decirle qué tiene que hacer; estaremos diseñando un entorno mucho más extensible. Así logramos que cuando pongamos un nuevo elemento se encargue de lo suyo y de decirle al resto como actuar con él.

```
void SP_Auto::ActzarIA(){

       ActualizarPosicion();

      for(int i = 0; i < cantAutos; i++)     Interactuar(GetAuto(i));

             SP_Mensaje* po_mens;
      for(int i = 0; i < cantEleEst; i++)
            if(GetSennal(i)->TieneMensaje()){
                     mens = GetSennal(i)->Mensaje();
                     EjecutarAccion(mens);
            }}
```

Ahora tenemos, además de la lista de autos, una lista de elementos estáticos (señales de tránsito en nuestro caso), cada uno guarda información particular, los voy recorriendo y si tengo mensajes de alguno de ellos llamo al método 'EjecutarMensaje' que precisamente ejecuta una acción en dependencia del mensaje que le sea pasado. En el caso de las señales de tránsito constantemente envían su mensaje para cuando algún carro esté cerca. De esa manera tendríamos, al igual que en la variante anterior, que analizar todas las señales, a no ser que tengamos una lista de mensajes y sean estos los que se recorren en un momento dado, más adelante veremos como funciona esta técnica. En el método 'EjecutarMensaje' tendríamos que analizar si el mensaje es para el carro en cuestión o es para otros. Aunque parezca que hacemos lo mismo, la ventaja fundamental es que el código de nuestro carro no hay que modificarlo cuando adicionemos nuevas señales o nuevos elementos de otro tipo, solo lo incluimos en el listado de elementos estáticos y le implementamos como tiene que interactuar con los objetos dinámicos.

Tenemos que lograr una eficiente transmisión de mensajes entre elementos del entorno para ello debemos analizar algunas cuestiones. Lo primero es que cada elemento debe ser capaz de enviar en un "mensaje" toda la información necesaria.

A estos mensajes en los juegos se les conoce como *trigger* [ORK02] que no son más que mensajes que se guardan en una cola con prioridad y contienen toda la información necesaria, por ejemplo, quien lo creó, para quien va dirigido, el tiempo de duración del mensaje, el radio de acción, el momento en el que fue creado, etc. etc. Los *trigger* tienen que ser actualizados en cada momento, analizar si aun existen o si hay que eliminarlos porque ya dejaron de ser útiles. Cada objeto dinámico tiene que recorrer los *trigger* para verificar si alguno es para él, y en consecuencia actuará. Esto se analizará posteriormente para hacerlo de manera eficiente. Primeramente debemos tener un código para cada mensaje, podemos para esto usar las variables enumerativas, así definimos los mensajes como un valor entero que los identifique.

```
enum MSG_Name {MSG_PARE, MSG_SEDAPASO, MSG_VELOCIDAD...}
```

Una posible clase para los mensajes es la siguiente:

```
class SP_MSG{

private:  MSG_Nombre sp_eNombre;
      int sp_iTipo;
      int sp_iPrioridad;
      char* sp_acEmisor;
      char* sp_acReceptor;
      float sp_fRadioDeAcc;
      float sp_fTiempoDeExp;
      bool sp_bRecibido;
      bool sp_bEmisorDin.

public:

  //................................//


};
```

No todos los mensajes van a guardar todos estos atributos por lo que esta idea se puede perfeccionar haciendo una jerarquía de clases, pero aquí para lograr simplicidad incorporamos todas las ideas en una misma clase.

El nombre es en base a los valores enumerativos predefinidos anteriormente, tenemos sin embargo otro atributo para el tipo de mensaje para facilitar la búsqueda del *trigger* cuando esté en una lista. Como habíamos dicho los *trigger* se guardan en una cola con prioridad por lo que tenemos que tener un atributo para guardar esta prioridad. Este valor va en aumento numérico, es decir el valor '1' es el de mínima prioridad, para garantizar que siempre haya opciones para aumentar la prioridad de un posible mensaje nuevo. Debemos guardar el *Id* del emisor y el receptor, pero puede ser que un mensaje no se envíe a alguien en especifico y sea algo para todo el que esté en un radio de acción de ahí el parámetro 'sp_fRadioDeAcc', este mensaje puede tener un tiempo de expiración 'sp_fTiempoDeExp'.

Cuando sea recibido se pone en falso el atributo 'sp_bRecibido' para no volver a ser analizado. Se elimina el *trigger* cuando el receptor sale del radio de acción. En nuestro caso lo elimina el emisor cuando el auto sale del 'área', después hablaremos con más calma de estas áreas.

El último de los parámetros se pondrá en verdadero si el emisor es un objeto dinámico, esto es muy útil porque tendremos que seguir analizándolo por si hay alguna actualización.

En nuestro caso los mensajes de las señales de tránsito pudieran estar constantemente lanzados para cuando un auto se acerca capture ese mensaje y actué en consecuencia, esta idea ya la habíamos comentado. Sin embargo pudiera ser la misma señal quien esté analizando si alguien ha entrado en su radio de acción y de ser así le envía un mensaje al elemento que entra en su radio. Como veremos más adelante esta estrategia es mucho más eficiente y fácilmente aplicable a nuestro caso.

Variabilidad de acción.

Para lograr un entorno lo más cercano a la realidad posible debe existir diversidad, esta diversidad debe estar en función de lograr diferencia de actuación entre elementos del mismo tipo. Si implementamos varios objetos dinámicos, se nos complica el código, se demora más procesar la diversidad de inteligencia de los elementos, hay que diseñar más objetos 3D y sus

animaciones, etc. etc. Sin embargo, si logramos que objetos iguales tengan comportamientos diferentes para iguales o similares situaciones logramos la diversidad que necesitamos en nuestro entorno.

Lograremos esto mediante el máximo de explotación de los datos o variables de cada elemento.

```
class SP_Auto{
private: int sp_iId;int sp_iEstado;     SP_Posicion sp_oPos; int sp_iVelocidad;
    SP_VecDirec sp_oDireccion; int sp_iPrudencia; int sp_iMaxVel; int
    sp_iTiempoResp;     int sp_iIdArea;

        .........


  public:  ............
        };
```

Para lograr la diversidad ¿Que hemos empleado? Además de las variables necesarias, que son muchas más que las que se ven en el segmento de código, utilizamos 'sp_iPrudencia' esta variable hará que cada carro actúe de manera diferente en situaciones similares. Usaremos esta variable, por ejemplo, con el objetivo de que existan carros que se acerquen más a los de adelante, o que se atrevan a cruzar a otros carros aunque venga alguno relativamente cerca de frente, o que cruce un semáforo con la amarilla, o que no pare en un seda el paso cuando viene algún carro relativamente cerca, en fin que usaremos esta variable como una probabilidad de efectuar determinada acción.

Otra de las variables interesantes y que harán diferencia entre los carros es 'sp_iMaxVel' que limitará la velocidad máxima de cada carro. También mostramos en el ejemplo 'sp_iTiempoResp' que no es más que el tiempo que daremos para analizar cada elemento del entorno para no hacerlo cada frame, este tema lo analizaremos más profundamente en la próxima sección.

Las variables más importantes para la entidad en este caso son su posición, la velocidad y la dirección de su movimiento, en base a esto tendremos que actualizar su posición cada frame. Ya se han realizado estudios de cómo lograr la movilidad de los autos en el entorno por lo que no será analizado en este artículo.

Hasta aquí tenemos planteada nuestra principal entidad dinámica, con sus principales características, sin dudas todo esto lo podemos englobar en una jerarquía de clases donde tendríamos una entidad general con algunas de estas características y otras que seria usadas en todos nuestros objetos, las clases hijas serian los autos, las personas y quien sabe si hasta animales en un momento determinado. Posteriormente los autos pudieran separarse en autos ligeros, pesados etc. etc. hasta llegar a una jerarquía de clases donde implementaríamos cuestiones muy específicas para cada nuevo tipo de objeto.

Optimalidad.

El funcionamiento general de los juegos es analizar en cada frame o instante de tiempo cada uno de las cuestiones importantes. Sin embargo siempre hay que buscar estrategias para no hacer cálculos innecesarios. En todos los casos tienen que interactuar los elementos del entorno virtual. Una de las estrategias generalizadas y muy útil es la de dividir el mapa en secciones o áreas para hacer las búsquedas [BIA02]. No es objetivo de este trabajo definir bien cómo seleccionar las áreas y como quedarían distribuidas en todo el entorno, ya que esto requiere un estudio más pormenorizado del tema.

Cada sección, tiene una o varias señales de tránsito u otros elementos estáticos o

dinámicos, estos pondrán los *trigger* pero en una cola de prioridad dentro del área donde se encuentran o sección, para que los elementos dinámicos sean capaces de analizar los *trigger* que están en su área y no los de todo el entorno.

De aquí podemos deducir que la interacción de los elementos no se hará con el resto de los elementos del entorno sino solo con los que están en la misma área o en las áreas adyacentes. Esto se verá mucho más específicamente en próximos trabajos, aquí solo daremos la idea general de cómo tener en cada área un puntero a los elementos dinámicos y una lista con los elementos estáticos que están ubicados dentro del área.

```
class SP_Area{
private: int sp_iId;
        SP_Auto** sp_aoAutos;
        SP_Sennales* sp_oSennales;
        SP_Coordenadas* sp_oCoordenadas;
        SP_Area** sp_aoAdyacentes;

        ..............


};
```

Como podemos observar además de lo ya señalado tenemos un puntero a las áreas adyacentes, que es lo que nos va a garantizar no hacer búsquedas y análisis en todo el entorno, sino solo en un espacio reducido cercano al objetivo.

Otra manera de optimizar el análisis de la IA en los elementos, es no haciéndola en cada frame. Todos sabemos que en la vida real los seres vivos no actúan inmediatamente a las acciones del medio, si implementamos que constantemente se analice cada situación del medio y en consecuencia actúe, será todo perfecto y al instante, algo que sabemos no ocurre así en la vida real. Una manera de encontrar realidad y además hacer más eficiente nuestro módulo es dando un tiempo de análisis a determinadas partes de inteligencia [MCL02]. En juegos por ejemplo se analiza la visión y la audición cada un tiempo que puede oscilar en dependencia del momento o del tipo de elemento en cuestión. Nuestros carros tendrán el análisis de la posición, al igual que en los juegos, cada instante, sin embargo el análisis de los *trigger* (Donde estará guardada la información de cada señal de tránsito o la cercanía de algún otro carro) la haremos cada un tiempo que depende de cada objeto (Aunque sean de un mismo tipo) y estará guardado en la variable "sp_iTiempoResp" como habíamos dicho anteriormente. Esto implica separar la actualización de la IA en métodos diferentes, un método para la actualización constante y otro método para la actualización temporal.

```
void SP_Auto::ActzarCteIA(){

        ActualizarPosicion(); }

        void SP_Auto::ActzarTpralIA(){
       if(tiempo >= sp_iTiempoResp){
         for(int i = 0; i < cantAutos; i++)
               Interactuar(GetAuto(i));
           SP_Mensaje* po_mens;
         for(int i = 0; i < cantEleEst; i++)
             if(GetSennal(i)->TieneMensaje()){
                       mens = GetSennal(i)->Mensaje();
                       EjecutarAccion(mens);
             }}else
      tiempo++; }
```

La variable tiempo pudiera declararse 'static' en el método o considerarse una variable auxiliar dentro de los atributos de la clase. Los valores que tomaría 'sp_iTiempoResp' como habíamos visto serán diferentes para cada objeto. Los valores serán asignados aleatoriamente en el constructor en un rango aceptable de valores posibles, tomando en cuenta que no demore mucho ni que analice demasiado rápido.

Ahora veremos como llamar desde la clase controladora a estos métodos de los objetos dinámicos.

```
void SP_Principal::AnalizarIA{
      for(int i = 0; i < sp_iCantOD; i++){
              sp_aoAutos->ActzarCteIA();
              sp_aoAutos->ActzarTpralIA();
      }}
```

Esta idea tiene un posible problema y es que en algún frame coincidan varios objetos a analizar su IA completa, influyendo negativamente en el rendimiento en un momento dado de la ejecución. Para garantizar que no suceda esto, se analiza un máximo de elementos en cada instante, para esto el método 'ActzarTpralIA()' debe retornar un valor 'bool' que sea verdadero si hubo procesamiento de IA y falso en caso contrario. La implementación quedaría como sigue:

```
      void SP_Principal::AnalizarIA{
      for(int i = 0; i < sp_iCantOD; i++){
              sp_aoAutos[i]->ActzarCteIA(); }
      unsigned int procesados=0;
      unsigned int i = 0;
      while((procesados < sp_iMaxProcPorFrame) && (i
< sp_iCantOD)){
       if(sp_aoAutos[i]-
             >ActzarTpralIA())
             procesados++;   i++;
             }}

        bool SP_Auto::ActzarTpralIA(){
              if(tiempo >= sp_iTiempoResp){
              for(int i = 0; i < cantAutos;
              i++)    Interactuar(GetAuto(i));

         SP_Mensaje* po_mens;
       for(int i = 0; i < cantEleEst; i++)
           if(GetSennal(i)->TieneMensaje()){
                    po_mens = GetSennal(i)->Mensaje();
                    EjecutarAccion(mens); }
       return true;
      }else{  tiempo++;
            return false;
      }}
```

Esta estrategia logra que no se acumulen demasiados carros para actualizarse la IA temporal en un mismo frame y pasen a ser analizados en el próximo, la probabilidad de que en el próximo frame vuelvan a haber muchos carros para actualizarse y que también estén delante en la lista es realmente muy baja. Nótense que incluso sucediendo esto en dos frame sucesivos no afectaría considerablemente la 'inteligencia' del objeto, puede ser incluso que haya otros objetos que por tener mayor 'sp_iTiempoResp' demoren más en analizar nuevamente su IA temporal.

Conclusiones.

El sistema de IA de los entornos virtuales es complejo y siempre hay que analizar las particularidades de cada uno. Sin embargo estas son algunas ideas que se están llevando a cabo para la implementación del módulo de IA para un simulador de auto y que pueden ser generalizadas a varios sistemas similares. Lograr la extensibilidad del sistema mediante la distribución de la IA y no la concentración en algunos tipos de objetos, lograr la interacción de los objetos mediante el envío de mensajes, lograr la variedad en el entorno mediante variables y sus valores y no aumentando los tipos de objetos, lograr velocidad de procesamiento mediante la división del terreno a evaluar en subsecciones, mediante el análisis temporal y no constante de partes de la IA y definiendo un máximo de objetos a analizar por frame. Este es un tema que conlleva una profundización y la aplicación de otras muchas estrategias para lograr un simulador competitivo en el mercado mundial. Para ello en este momento se desarrollan varias tesis de pregrado que nutrirán en poco tiempo y profundizarán en estos y otros muchos temas dentro del mundo de la IA para simuladores.

Referencias bibliográficas:

[TOZ02] ***Tozour. Paul – Ion Store. "The evolution of game IA"*** AI Game Programing Wisdom. Editado por Steve Rabin. ISBN: 1-58450-077-8.

[ORK02] ***Orking. Jeff "A General-Purpose Trigger System."*** AI Game Programing Wisdom. Editado por Steve Rabin. ISBN: 1-58450-077-8.

[MCL02] ***McLeans. Alex W. "An efficient Ai architecture using prioritized task categories"***. AI Game Programing Wisdom. Editado por Steve Rabin. ISBN: 1-58450-077-8.

[BIA02] **Biasillo. Gari, "Representing a Racetrak for the AI".** AI Game Programing Wisdom. Editado por Steve Rabin. ISBN: 1-58450-077-8.

Bibliografía consultada:

AI Game Programing Wisdom. Editado por Steve Rabin. ISBN: 1-58450-0778.

AI for Game Developers. **Por:** David M. Bourg, Glenn Seeman ISBN: 0-59600555-5

AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. Por: Alex J. Champandard ISBN: 1-5927-3004-3

Anexo 3.

Resumen de:

[LOR07] Lores Caignet, Luís F. Marty Consuegra, Yunior M. *Detección de infracciones del conductor en un simulador de auto.* Trabajo para optar por el título de Ingeniero en Ciencias informáticas. Universidad de las Ciencias Informáticas. Ciudad de La Habana, 2007.

Universidad de las Ciencias Informáticas.
Facultad 5. Entornos Virtuales.

Título de la tesis. "Detección de infracciones del conductor en un simulador de conducción de auto"	
Nombre de los autores	Luís Felipe Lores Caignet Yunior Michel Marty Consuegra
Nombre del Tutor	Lic. Yuniesky Coca Bergolla
Nombre del Vicedecano que avala la calidad de la tesis.	Lic. Lidiexy Alonso Hernández
Nombre del jefe de tribunal	Ing. Alain Hernández Castillo
Nombre del Oponente	Ing. Osmanys Valdés Puga

RESUMEN

Al transitar por una urbe con un tráfico de autos considerablemente grande se provocan fácilmente desastres automovilísticos. Se podría deducir que el principal problema del tráfico en la actualidad al haber tanta circulación en las vías es causado fundamentalmente por la falta de precaución y de destreza al conducir. En el simulador de conducción de auto que se desarrolla de manera conjunta entre la empresa SIMRPO y la Universidad de Las Ciencias informáticas, no se cuenta con un sistema que posibilite la prevención de infracciones del conductor al ingresar en su paseo virtual, todo esto ha traído consigo una falta de realismo importante en el simulador y un futuro descontento de los usuarios que obtendrán los servicios finales de este producto. La mayor importancia de la prevención de infracciones en un simulador de conducción recae en el seguro de vida que representaría para los conductores conocer y

prevenir dichas infracciones, mucho antes de enfrentarse a la práctica real de conducción.

Por tanto se hace necesario desarrollar un sistema que permita darle solución a los problemas mencionados anteriormente, con esta convicción y espíritu de superación, se propone este trabajo.

El objeto de estudio de la tesis son los simuladores virtuales, delimitando así el campo de acción a la detección de infracciones de un conductor en un simulador de autos.

El objetivo general de la tesis es desarrollar un módulo que permita la detección y clasificación de las infracciones cometidas por el conductor en el simulador de de conducción de auto.

Se tiene como objetivos específicos:

- Detectar las infracciones de tránsito cometidas por el conductor en el simulador de auto.
- Clasificar las infracciones de tránsito cometidas por el conductor en el simulador.
- Realizar una propuesta de solución para una futura implementación de las infracciones mecánicas cometidas por el conductor.
- Integrar a la Herramienta del Simulador de auto el Módulo de detección de infracciones de conducción.

El contenido del documento presentado está desglosado en 4 capítulos que se describen a continuación:

Resumen:

Partiendo de una brillante idea de Craig Reynolds, que ha recorrido el mundo y que mantiene impresionante actualidad, la manipulación de los comportamientos (Steering Behaviors) de los elementos virtuales y sus aplicaciones, siguen siendo aplicadas en diversas tareas relacionadas con los entornos virtuales. Con una implementación y fundamento muy sencillo, con un basamento físico-matemático accesible y eficiente, sigue mostrando resultados muy positivos en esta rama. De varios artículos y libros se han conocido con mayor o menor grado la manera de aplicarlos y de llegar a resolver un problema determinado. ¿Se imaginan controlar el tránsito en una ciudad a partir de los Steering behaviors? ¿Se podrá lograr un resultado eficiente de carros auto-controlados en una ciudad? ¿Será aplicable al desarrollo de un simulador de conducción de auto? Se proponen varias ideas para lograr este control del tráfico en un entorno de ciudad de una manera eficiente, haciendo diferencias entre los autos que se autocontrolan y el conductor de un simulador de conducción.

Introducción.

Los comportamientos de los elementos autónomos en cualquier entorno virtual pueden ser perfectamente simulados con el uso de esta técnica bien conocida y actualizada [REY99]. Hay ejemplos muy claros [GO04] [BIA02] [WAN05] de cómo usar esta técnica para el control de autos, el problema puede ser cómo chequear el entorno en una ciudad virtual para que cada auto respete las señales del tránsito. La base de esta propuesta será la misma planteada en otros trabajos [COC06], se hará uso de una clase que tendrá los atributos necesarios para controlar todo el modelo físico-matemático del auto, con la masa, la velocidad, la aceleración y la posición, fundamentalmente. Además se necesita cada uno de los comportamientos que se adicionarán a los autos. Para realizar el chequeo del entorno y las colisiones se realizarán algunas modificaciones a la implementación de los Stering Behaviors propuestos en la bibliografía [BUC05], para lograr los resultados esperados.

Existen varias señales de tránsito que los conductores deben respetar en una ciudad, sin embargo se podrá observar que muchas obligan al conductor (o a los autos en nuestro caso) a actuar de la misma manera, variar su aceleración y por consiguiente su velocidad, efectuar un giro determinado o detenerse en un lugar específico. Esto permitirá el chequeo de nuevas señales de manera rápida, sin tener que adicionar nuevos códigos.

Cada señal de tránsito garantizará actualizar los comportamientos de los autos que entran a su radio de acción, siendo los autos los que chequean en cada momento si entraron o no en un área de acción de alguna señal. Con algunas modificaciones y combinaciones de los comportamientos se logra mantener a los autos a la derecha en las calles y respetar cada una de las señales de tránsito.

Una de las propuestas fundamentales de este trabajo es el uso de una estructura diferente a las planteadas en la actualidad, un entorno dividido en áreas que pueden estar superpuestas o separadas, para la ubicación de las señales de tránsito en el entorno. Esta estructura de datos junto a una rejilla regular que nos permitirá el chequeo del entorno de una manera eficiente, para encuestar sobre los autos, obstáculos y demás objetos del entorno, forman una potente estructura que logra eficiencia y facilidad de comprensión.

Se plantea una propuesta para optimizar el procesamiento mediante la aplicación de un algoritmo de reubicación [NOG07], logrando dinamismo en el entorno con una cantidad mínima de autos circulando. Además se usarán los volúmenes de frontera temporal [NOG07] para realizar todo el análisis de los comportamientos, solo a los elementos que sean visibles en un momento dado, además de realizarse una propuesta de análisis de la "Inteligencia Artificial" de los autos en intervalos de tiempo y no en cada frame [COC06].

Estas ideas no solo son aplicables a los autos virtuales que se mueven por el entorno virtual, también pueden ser muy útiles en la detección de posibles infracciones cometidas, por ejemplo por un jugador o un usuario que hace uso de un simulador de conducción de autos [LOR07].

Desarrollo.

En un entorno de ciudad los carros deben respetar las leyes del tránsito, se puede hablar de señales como los 'pare', los 'seda 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.

La propuesta que aquí se muestra se ha tratado de realizar lo más general posible para ser aplicada no solo al simulador de conducción (donde se prueban estos resultados), sino a otros tipos de aplicaciones que necesiten representar entornos de ciudad. Cuando se realiza alguna propuesta que puede ser aplicada solo a simuladores de conducción se aclara y se trata de brindar una variante para otros tipos de aplicaciones. Un ejemplo de esto es que el simulador muy concretamente tiene un conductor; este es un auto que se mueve por el entorno, no es controlado por la aplicación, es el usuario que se sienta al simulador el que se va moviendo por el entorno, por tanto no hay que decirle lo que tiene que hacer, pero sí hay que saber en cada momento si realizó alguna maniobra incorrecta. Como este simulador tiene un componente instructivo, que es para el entrenamiento, hay que controlar todas las infracciones cometidas por dicho conductor [LOR07] en su ejercicio virtual, se incluyen en la bibliografía las infracciones de tipo mecánicas, es decir, con relación a los cambios de velocidad, de presión del freno, el embrague, etc. pero estas no son objetivo de este trabajo.

Modelo físico-matemático del auto.

Para entender todo el trabajo se debe comenzar por explicar qué es el modelo físico-matemático de los autos.

Primeramente nuestra clase base será la que guarde todas la informaciones necesarias para el auto. Se considera como información que no puede faltar en dicha clase, un identificador, su posición en el entorno y el radio que ocupa en ese entorno. Otras informaciones que se necesitarán son la velocidad, y el sentido para el que se mueve el auto, esto se guarda en dos vectores uno que represente la dirección en la que está el auto y el otro perpendicular al mismo para saber cuál es su izquierda esto garantiza saber cual es el sentido en el que se mueve el auto, estos dos vectores definen un eje de coordenadas para el auto. La dirección del auto siempre será alineada con la velocidad.

Otras de las características necesarias son, la masa, la velocidad máxima que puede alcanzar el auto, la mayor fuerza que se le puede aplicar y el máximo valor de rotación admisible.

Como necesidad esta clase 'vehículo' debe tener un puntero a la clase controladora del entorno, para poder chequearlo en busca de obstáculos o de otros vehículos, cada instante de tiempo, además de un objeto de la clase comportamientos la cual es la encargada, como se verá más adelante, de analizar los comportamientos que se quieran obtener del auto. En el transcurso del trabajo se verá que se debe tener una colección de estos objetos.

Por último se necesita un método que sea el encargado de actualizar, cada instante, todos estos valores del auto.

```
    SVector2D    m_vHeading;
    SVector2D    m_vSide;
    double       m_dMass;
    double       m_dMaxSpeed;
    double       m_dMaxForce;
    double       m_dMaxTurnRate;
    GameWorld*         m_pWorld;
    SteeringBehaviors* m_pSteering;
public:
    void Update(double time_elapsed);
};
```

Lo primero que se hace en la implementación del 'actualizar' es llamar a algún método que calcule la fuerza que se le va a aplicar al auto, a este o estos métodos estará dirigido el próximo epígrafe. Luego de tener este resultado se aplican las leyes de Newton. Lo primero es calcular la aceleración a partir de la masa y la fuerza que ya se había calculado.

Aceleración = Fuerza / masa.

Y después se actualiza la velocidad del vehículo aplicando:

*Velocidad += aceleración * tiempo.*

El tiempo es precisamente el que se pasa por parámetro. Sin embargo esta velocidad no siempre es aceptada por el vehículo, si recuerdan se tenía una velocidad máxima declarada como atributo de la clase, por tanto hay que verificar que la velocidad no exceda esa velocidad máxima.

Ahora sí se pueden actualizar los nuevos valores de las coordenadas del auto en el entorno.

*Posición += velocidad * tiempo.*

Aun falta actualizar el sistema de coordenadas del auto. Es decir el vector dirección y el vector perpendicular a este.

Este sistema de coordenadas se actualiza sólo si la velocidad es mayor que un valor bastante pequeño pero nunca si es cero ya que el cálculo de la normalización haría que se produjera una división por cero.

Solo faltaría verificar que esa posición esté dentro de las coordenadas del entorno, para ello se puede llamar un método con el valor de la posición y teniendo guardado los valores de x y y máximos del entorno, si excede alguno de ellos se reinicia nuevamente en cero esa coordenada, esto permite volver al inicio cuando el auto llegó al final. Si para nuestro trabajo particularmente no se necesita esto porque quizás se tiene bien definido el entorno, se puede crear un borde en el entorno para que el auto nunca pueda salir de este, entonces no habría que hacer este análisis.

```
bool Vehicle::Update(double time_elapsed)
{
  SVector2D SteeringForce = m_pSteering->Calculate();
  SVector2D acceleration = SteeringForce / m_dMass;
  m_vVelocity += acceleration * time_elapsed;
  m_vVelocity.Truncate(m_dMaxSpeed);
  m_vPos += m_vVelocity * time_elapsed;
  if (m_vVelocity.LengthSq() > 0.00000001)
  {
    m_vHeading = Vec2DNormalize(m_vVelocity);
    m_vSide = m_vHeading.Perp();
  }
}
```

Comportamientos o Steering Behaviors.

En este momento se hará referencia al método que quedó planteado anteriormente, ¿Como calcular la fuerza? ¿Cómo saber hacia donde va dirigida la fuerza que se debe aplicar al auto? Todo esto depende de los comportamientos que se quieran obtener del auto. Se puede querer en algún momento que el auto se mueva aleatoriamente esquivando los obstáculos y los otros carros, pero en otro momento se puede querer que siga a otro carro o que se aleje de algún otro auto, que llegue a una esquina y pase si tiene derecho de hacerlo o espere si viene algún auto por la calle perpendicular, quizás simplemente que llegue a un lugar determinado, en fin son varios los casos que se pudieran tener, más o menos complejos. Para comenzar se verá de manera independiente algunos comportamientos. Cada uno de ellos se representará por una clase que herede de la clase genérica '`SteeringBehaviors`' que se vio en la declaración de la clase '`Vehicle`'.

Alcanzar un objetivo.

Con este comportamiento se logra un movimiento hacia un objetivo, pero como se podrá observar no es muy útil cuando se quiere llegar hasta ese objetivo.

Lo primero es calcular la velocidad deseada, que no es más que el vector que representa la velocidad máxima que puede alcanzar el elemento hacia el objetivo.

Este método retorna la fuerza requerida para ser aplicada al auto para que lo impulse hacia su objetivo. Para lograr esto solo hay que restar el vector de la velocidad actual a la velocidad deseada. El método quedaría de la siguiente manera.

```
Vector2D cSeek_Behavior::Calculate( int ElapsedTime )
{
  Vector2D DesiredVelocity =
     Vec2DNormalize(Target - pEntity->Position())* pEntity->
MaxSpeed();
  return (DesiredVelocity-pEntity->Velocity());
}
```

Llegar a un objetivo.

Este método es muy parecido al anterior pero tiene una llegada mucho más realista ya que va disminuyendo la velocidad del auto cuando se acerca al objetivo. Por supuesto que esto ocurre cuando está lo suficientemente cerca, para mantener la velocidad

constante la mayor parte del camino y solo disminuirla cuando este lo suficientemente cerca. Esto hace mucho más realista este comportamiento.

```
Vector2D cArrive_Behavior::Calculate( int ElapsedTime )
{
   double speed = pEntity->MaxSpeed();
   double dist  = Vec2DDistance(Target, pEntity->Position());
   rep_explosion=false;
   if(dist <= 1000)
  speed = (speed*dist)/1000.0f;
   Vector2D DesiredVelocity =
                    Vec2DNormalize(Target - pEntity->
Position())*speed;
   return (DesiredVelocity-pEntity->Velocity());
}
```

Este comportamiento es igual al anterior cuando el elemento está lejos del objetivo y solo cuando se acerca (Esta distancia se puede variar en dependencia de la situación o de lo que quiera obtener el programador) comienza la deceleración hasta hacer su velocidad cero cuando llega al objetivo.

Alejarse de un objetivo

Este comportamiento es el opuesto al de alcanzar un objetivo. Lo que tenemos que lograr es el vector de la fuerza en el sentido contrario. Esto se logra restándole al vector posición del vehículo la posición del objetivo, para obtener la dirección del vector que representa la velocidad deseada.

```
Vector2D cFlee_Behavior::Calculate (Vector2D TargetPos)
{
    Vector2D DesiredVelocity = Vec2DNormalize(m_pVehicle->Pos()-
TargetPos)* m_pVehicle->MaxSpeed();
    return (DesiredVelocity - m_pVehicle->Velocity());
}
```

Evadir obstáculos.

Este comportamiento es el que va a permitir evadir obstáculos en el entorno. Para hacer más simple la explicación se trabajará con obstáculos que serán círculos. Para detectar estas posibles colisiones se usará un área rectangular delante del carro y que será proporcional a la velocidad que lleve. El ancho del rectángulo es el mismo que el ancho del auto.

Para encontrar si hay una posible intersección o no se proponen 4 pasos fundamentales que se exponen a continuación.

1. Se analizan los obstáculos que estén en el radio de acción del auto, es decir tomando como radio la longitud del área rectangular de chequeo.
2. Se analizan a partir de un sistema de coordenadas con base en el elemento, los obstáculos que quedan con valor de x negativa no se analizarán.
3. De los obstáculos que quedan por analizar se verifica si el área de chequeo intercepta a alguno.
4. Determinar los puntos donde intercepta el área con cada obstáculo.

Después de tener los puntos de intercepción con los obstáculos se puede calcular la fuerza necesaria para alejarse de él, similar al comportamiento ya descrito anteriormente.

Evadir rectas.

Aquí también se evaden objetos, pero en este caso segmentos de líneas, para ello se tienen definidas las líneas y a partir de eso se verifica si existe la posibilidad de sobrepasar la línea y de ser así se retorna una fuerza igual a la distancia que se sobrepasaría la recta pero en el sentido de la normal de la línea.

Path Follow.

Este comportamiento es muy útil, ya que crea una fuerza que hace mover al agente a través de una serie de puntos que forman un camino. Regularmente este camino tendrá un punto de partida y uno de llegada. Otras veces se realiza un ciclo infinito y se mueve de manera circular por los puntos definidos en su camino.

Para lograr este comportamiento se necesita, como es lógico, una clase grafo encargada de tener los puntos a través de los cuales se moverá el agente. Como se verá en el código que se muestra se usarán los comportamientos ***Alcanzar un objetivo*** para lograr un movimiento a través de los puntos y del ***Llegar a un objetivo*** cuando se encuentra el punto de llegada del grafo.

```
SVector2D cFollowPath_Behavior::Calculate()
{
 if(Vec2DDistanceSq(m_pPath->CurrentWaypoint(), m_pVehicle->Pos())
                                        < m_WaypointSeekDistSq)
      m_pPath->SetNextWaypoint();

 if (!m_pPath->Finished())
    return Seek->Calculate(m_pPath->CurrentWaypoint());
 else
    return Arrive->Calculate (m_pPath->CurrentWaypoint());
}
```

Combinación de los comportamientos.

Como se había adelantado, los autos podrán tener varios comportamientos, incluso a la vez se pueden analizar varios posibles comportamientos. En lo primero que se puede pensar es en una especie de sumatoria de las fuerzas, pero esto puede traer resultados no deseados. Para lograr la combinación de los distintos comportamientos y obtener buenos resultados se muestran las siguientes ideas.

Cada auto tiene una lista de comportamientos que debe respetar en un momento determinado.

Cada 'comportamiento' es una clase y cada auto tiene una lista de objetos de esa clase. En el método 'calculateResultForce()' se calcula la fuerza resultante de llamar al método 'calculate()' de cada 'Steering Behaviors'. Teniendo en cuenta que en la clase 'vehículo' se necesita una lista de comportamientos.

```
SVector2D Vehicle::calculateResultForce()
{
  SVector2D SteeringForce;
  std::vector<cSteeringBehavior*>::const_iterator curOb =
```

```
                                                    apSteering.begin();
    while(curOb != apSteering.end())
        SteeringForce += ((*curOb)->Calculate()*(*curOb)->Amount());
    return SteeringForce.Truncate(MAX_STEERING_FORCE);
}
```

El comportamiento final se calcula multiplicando el valor de cada 'comportamiento' por un peso. Esto puede hacerse dentro del mismo método calcular de cada clase 'behavior' o puede ser afuera, teniendo un atributo cada steering que diga cuál es su peso y se calcula afuera llamando a ese método, esto ayudaría si en algún momento hace falta la fuerza sin combinarla con otra.

Un importante inconveniente de este tipo de cálculo de la suma es que si tiene muchos comportamientos el objeto puede ser muy costosa la ejecución de todos ellos, pero además es complejo dar un peso realmente factible a cada comportamiento. Otro problema serio es que en algunos casos puede existir conflicto entre las fuerzas y que realmente haya alguna violación importante que no debía ocurrir.

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

El método quedaría más o menos así:

```
SVector2D Vehicle::calculateResultForce()
{
  m_vSteeringForce.Zero();
  SVector2D force;

  if (On(wall_avoidance))
  {
    force = WallAvoidance->Calculate(m_pVehicle->World()->Walls())
* m_dMultWallAvoidance;
      if (!AccumulateForce(m_vSteeringForce, force))
        return m_vSteeringForce;
  }

  if (On(obstacle_avoidance))
  {
     force = ObstacleAvoidance->Calculate( m_pVehicle->World()->
                          Obstacles()) * m_dMultObstacleAvoidance;
     if (!AccumulateForce(m_vSteeringForce, force))
         return m_vSteeringForce;
  }
     /* ..... */

  return m_vSteeringForce;
}
```

El método 'On' devuelve verdadero si ese comportamiento está en la lista de los comportamientos a analizar por el vehículo.

Como es lógico aquí no se muestran todos los comportamientos, solo algunos de ellos. Otra variante para lograr un análisis mucho más realista, es hacer una diferencia entre comportamientos prioritarios y otros no prioritarios, si la combinación de los

prioritarios determina una fuerza es la que se analiza y no se analizan los otros posibles comportamientos, en el próximo frame sin lugar a dudas les tocará el turno a ellos, esto permite mantener mucho más estable el movimiento de los autos.

Propuesta de solución.

El entorno estará formado por objetos que cumplirá cada uno una tarea específica. Ya se hizo referencia en epígrafes anteriores al auto, en cada momento, o cada un tiempo, tendrá que actualizar sus valores y mantener un movimiento realista por el entorno. Para lograr esto los autos tienen un conjunto de comportamientos que chequean constantemente como puede ser el de evadir obstáculos o el de seguir un camino (Que habrá que actualizar varias veces en el entorno). Además tendrán una lista de áreas en las que esté dentro en ese momento.

También se hizo referencia al conductor en el caso del simulador, toda la información referente al mismo también se maneja aunque se adquiere desde el modelo físico-matemático del auto del simulador, mucho más complejo que el explicado en el primer epígrafe de este trabajo.

Las señales de tránsito también se consideran objetos, cada señal de tránsito deberá guardar la información que necesite entregar a los autos. Para ello se creará una jerarquía de señales.

Las señales de tránsito pueden ser muy variadas pero la mayoría existen con el objetivo que los carros reduzcan su velocidad, paren si se cumplen determinadas condiciones o realicen un giro también en determinadas situaciones. Ejemplos las que obligan a tener una velocidad regulada, las señales de escuelas u hospitales, las de curvas peligrosas. Para que se detenga el auto se tienen las señales de semáforo si hay roja, las de pare, las cebras. Para hacer cumplir con estos comportamientos solo habrá que usar unos pocos Steering Behaviors, además de los que tienen que mantenerse de manera constante en cada auto. Los nuevos y más complejos serán los que permitan una interacción entre los autos, a la hora de pasarse entre sí, realizar maniobras en intercepciones de las calles, en los 'Pare' para saber si puede continuar o debe esperar a que cruce otro auto.

Hay señales que necesitan guardar información específica, pero todas guardan información que es común para todas, para ellos se define una clase general para las señales de la cual heredarán el resto.

La clase general va a guardar un área de acción, aquí es donde influirá la señal, pero además contendrá una colección de elementos, que no es más que un arreglo de objetos que representarán los comportamientos que deben ser incorporados a los autos que entran en su área de acción.

Se creará una clase para cada tipo de señal que hereda de la clase general. De manera específica los tipos de señales guardarán determinados atributos que sean necesarios, como por ejemplo, las áreas de velocidad máxima limitada, deben guardar ese valor.

Una aclaración importante es que las señales pueden tener áreas de acción que se superpongan, esto no entra en contradicción con la propuesta realizada en este trabajo, ya que como se había explicado cada auto puede analizar varios posibles comportamientos a la vez.

Para lograr que cada auto respete las señales cada una debe tener una lista con los comportamientos necesarios para transmitir a los autos que entran a su radio de acción. Una señal de tipo 'pare', debe transmitir un comportamiento de detenerse en una posición determinada. Pero no puede detenerse y salir en cualquier momento, tiene que ser cuando no haya autos que vengan por la calle perpendicular, para esto se

implementa un comportamiento nuevo que se encargue de analizar este tema [WANO5], el cual es usado también en las señales de tipo 'seda el paso', 'paso peatonal', aunque esta última lo que analiza es si están cruzando personas y no autos. Por lo pronto no es objetivo explicar este nuevo comportamiento. Una señal de 'No parqueo' por ejemplo, tendría un comportamiento que permita ejercer una fuerza constante en dirección de la velocidad, pero para evitar que el auto se detenga este comportamiento tiene que tener cierta prioridad sobre otros comportamientos, en la propuesta hecha para combinar los comportamientos, este sería de los primeros en analizarse.

¿Concretamente cómo se logra la actualización de los comportamientos de los autos? Se plantea a continuación un algoritmo para lograr esto.

Para cada auto se chequea.
Si está dentro del área de acción de alguna señal y no estaba desde antes.
Incorpora a sus comportamientos los que tiene esta área.
Adiciona a su lista de áreas en la que se encuentra ahora.
Analiza comportamientos prioritarios
Si devuelve una fuerza distinta de cero.
Actualizo valores del auto.
Sino
Analiza el resto de los comportamientos (Hasta llenar la capacidad máxima de la fuerza o llegar al final de la lista de comportamientos a analizar).
Actualizo valores del auto.
Para cada área de su lista de áreas.
Si el auto ya no está en ella.
La elimina de su lista de áreas.
Elimina los comportamientos relacionados con la misma. (Si no están relacionados con otras áreas)

Cada uno de los pasos lleva una explicación que hacerla aquí se haría demasiado extenso el trabajo. Este es un primer acercamiento a este tema que será abordado más profundamente en trabajos posteriores.

No se puede dejar de mencionar la estructura de datos básica necesaria para el chequeo constante del entorno, para esto se usará una rejilla regular, con la cual se garantiza buscar los autos a analizar solo en las rejillas vecinas, al igual que para el análisis de los obstáculos.

Tampoco se ha hablado de cómo lograr el movimiento de los autos y como guiarlos por el entorno. Pues existe un comportamiento para lograr un movimiento aleatorio de los autos, que unido a la evasión y la separación de los bordes de la calle permite un comportamiento relista y eficiente, en este mismo punto se logra mantener al auto cerca de la orilla derecha pero sin colisionar. Además se puede tener un grafo que guarde como nodos cada esquina de la ciudad y en cada momento se le pasa a los autos un comportamiento 'Alcanzar un objetivo' que sea uno de los vértices, pero que cuando se aproxime a la esquina automáticamente se le asigne otro punto de llegada al auto, de manera muy similar al comportamiento 'Seguir camino' explicado en epígrafes anteriores.

También se mencionará a continuación la manera de analizar la actuación de los carros cuando el posible obstáculo sea otro carro. El análisis de los obstáculos ya se comentó, sin embargo en el caso que un auto se acerque a otro hay que verificar si viene de frente abra que tomar algunas precauciones pero solo para garantizar que no ocurra una colisión, pero cuando lo que quiere es adelantar al carro que va en su mismo sentido

se puede crear un conflicto, al querer adelantarlo por la fuerza que le estamos tratando de poner hacia la derecha de la senda, por tanto la propuesta que se hace por el momento es regular la velocidad del auto hasta la velocidad del que va delante, esto por el momento pudiera atentar contra el realismo del entorno pero como al llegar a la esquina más próxima pueden tomar calles diferentes, no se hará visible esta situación. Para próximos trabajos se recomienda desarrollar un comportamiento para solucionar esta situación y mostrar de manera realista el avance de un auto a otro.

Optimizaciones.

Hasta aquí se ha mostrado una propuesta de cómo lograr un tráfico adecuado en un entorno virtual de ciudad, pero de todos es conocido que estos sistemas siempre han presentado una contradicción entre calidad visual e inteligencia, el problema es que ambas tareas consumen muchos recursos. Dándole solución a este problema hay que lograr optimizaciones, tratar de hacer en cada frame la menor cantidad de tareas posible afectando lo menos posible el realismo del entorno. Ya en nuestros proyectos se han presentado trabajos sobre este tema.

Siempre que en el entorno exista una única cámara para visualizar se puede aplicar un algoritmo de reubicación combinado con los Volúmenes de Frontera Temporales, TBV por sus siglas en Ingles [NOG07].

Si por el contrario la aplicación es, por ejemplo, un juego en red, donde existen varias áreas del entorno visibles. Se puede aplicar, además del algoritmo de los TBV, y siguiendo la teoría de grafos, una relación entre las señales, permitiendo en cada momento que cada auto no tenga que analizar todas las señales del entorno, sino solo las posibles a transitar desde la posición donde se encuentra. Esto es fácilmente aplicable teniendo en cada señal una lista de las áreas adyacentes o accesibles desde ella, y que cada auto guarde una lista con las áreas accesibles desde el lugar donde se encuentra.

Otra de las optimizaciones que se pueden lograr es no chequeando la IA cada frame, sino cada un tiempo determinado, incluso en los autos que pueden ser visibles, es fácil imaginar que haciendo un sistema que en cada momento se analice cada una de las situaciones, puede mostrar poco realismo, en la vida real las personas al conducir se demoran en tomar decisiones, pueden cometer alguna que otra infracción, en fin que si se deja un tiempo, corto por supuesto, de análisis a cada señal de tránsito se pueden alcanzar resultados mucho más realistas y a la vez se consumen menos recursos cada instante de tiempo. Sin embargo hay otras cuestiones que si hay que analizar cada frame. [COC06].

Detección de infracciones.

Como se ha dicho al conductor del simulador hay que darle un tratamiento especial en el entorno para el control de tránsito. En el simulador el conductor es el protagonista, todo debe girar en torno a él. La estructura básica que hemos mostrado se mantiene, la diferencia está en que en el área de acción de cada señal no se le obliga a tener un comportamiento al conductor sino que se analiza si ha violado esa señal [LOR07]. Con esto se logra el objetivo del simulador de servir como entrenador, en cada momento, o al final, se le muestra al usuario qué infracción cometió. Con esta información se pueden incorporar módulos de evaluación o tomar decisiones sobre posibles situaciones que se pudieran crear para ese mismo usuario u otros usuarios.

Inicialización del entorno.

¿Cómo organizar todas estas estructuras necesarias al inicio para hacer un uso eficiente en tiempo real? En tiempo de diseño se llena un fichero donde se debe guardar la información de cada señal (para facilitar el trabajo se puede crear una pequeña aplicación), esta información incluye el área de acción, el tipo de señal, los comportamientos que debe transmitir a los autos, etc. este fichero se carga en memoria al comenzar la simulación y permanece sin modificación siendo usado cada vez que sea necesario.

Igual sucede con cada estructura básica necesaria para el desempeño de la aplicación, se cargan al inicio de la ejecución y permanecen en memoria para agilizar el proceso de búsqueda.

Conclusiones.

El empleo de los comportamientos en elementos dinámicos, es fundamental en la actualidad para lograr entornos realistas y eficientes, por demás es una técnica muy fácil de implementar y de entender. Con este trabajo se logra dar una panorámica general de cómo modelar todo un sistema de control de tráfico para un simulador de conducción de auto. Mostrando variantes para la optimización de la propuesta.

Algunas de estas ideas ya se han aplicado a dicho simulador, ya está implementada e incorporada la detección de infracciones del conductor, mediante una tesis de grado [LOR07], se presentó este módulo como parte del simulador en la feria internacional Informática 2007, y fueron presentadas parte de estas ideas en el evento virtual de dicha feria mediante una ponencia que también se presentó en el evento nacional UCiencia 2006. Fue publicada en las memorias de ambos eventos. [COC06] [COC07]

Referencias bibliográficas.

[BIA02] Biasillo. Gari, *Representing a Racetrak for the AI.* En: AI Game Programing Wisdom. Editado por: Steve Rabin. ISBN: 1-58450-077-8. 2002.

[BUC05] Buckland, Mat. *"Programming Game AI by Example"* Wordware Publishing 2005.

[CAM04] Camacho Román, Yanoski Rogelio, Jiménez López, Fernando. *"Biblioteca Gráfica Para Sistemas de Realidad Virtual"*. Trabajo para optar por el Título de Ingeniería en Informática. CUJAE. Ciudad de La Habana. Julio de 2004.

[COC06] Coca Bergolla, Yuniesky. *"Algunas ideas a tomar en cuenta para la inteligencia de un simulador de auto"*. II taller de Realidad Virtual. Memorias del evento UCiencia 2006.

[COC07] Coca Bergolla, Yuniesky. *Algunas ideas a tomar en cuenta para la inteligencia de un simulador de auto* [En Línea] Evento Virtual de la Feria Internacional Informática 2007. Disponible en: http://www.informaticahabana.com/evento_virtual/?q=node/248&ev=3er%20Congreso%20Internacional%20de%20Tecnologias,%20Contenidos%20Multimedia

[GO04] Go, Jared. Vu, Thuc. Kuffner, James J. *Autonomous Behaviors for Interactive Vehicle Animations.* [En Linea] Eurographics/ACM SIGGRAPH Symposium on Computer Animation. Editado por: R. Boulic, D. K. Pai 2004. Disponible en: http://gamedev.cs.cmu.edu/vehicle_planning/paper/vehicle_anim_sca2004.pdf

[LOR07] Lores Caignet, Luis F. Marty Consuegra, Yunior M. *"Detección de infracciones del conductor en un simulador de auto"*. Trabajo para optar por el título de Ingeniero en Ciencias informáticas. Universidad de las Ciencias Informáticas. Ciudad de La Habana, 2007.

[MCL02] McLeans. Alex W. *"An efficient AI architecture using prioritized task categories"*. AI Game Programing Wisdom. Editado por Steve Rabin. ISBN: 1-58450-077-8. 2002.

[NOG07] Nogueira Collazo, Mariela. Correa Madrigal, Omar. *"Algoritmos para la manipulación eficiente de objetos dinámicos en escenas virtuales urbanas"* Trabajo para optar por el título de Ingeniero en Ciencias informáticas. Universidad de las Ciencias Informáticas. Ciudad de La Habana, 2007.

[REY99] Reynolds, Craig. *"Steering Behaviors For Autonomous Characters"* [En linea] Game developers conferences. 1999. Disponible en: http://www-eco.enst-bretagne.fr/~phan/complexe/steer/index.htm

[WAN05] Wang, Hongling. Kearney, Joseph K. Cremer, James. Willemsen, Peter. *Steering Behaviors for Autonomous Vehicles in Virtual Environments* [En línea] Proceedings of the IEEE Virtual Reality 2005. Disponible en: http://www.cs.uiowa.edu/~kearney/pubs/SteeringBehaviorsIEEEVR05.pdf

"Control del tránsito en un simulador de conducción de auto"

Autor: Lic. Yuniesky Coca Bergolla.
Coautores: Luís Felipe Lores
Annier José Alfonso

Universidad de las Ciencias Informáticas.
Ciudad de La Habana 2007

Anexo 4.

[COC07b] Coca Bergolla, Yuniesky. Control del tránsito en un simulador de conducción de auto. XV FORUM de Ciencia y Técnica. Universidad de las Ciencias Informáticas. 2007.

Capítulo I. ***Fundamentación teórica:***

En este capítulo se muestra la base teórica que fundamenta la investigación, se tratan conceptos fundamentales relacionados con la misma, se acerca al lector al conocimiento de los simuladores virtuales y su auge en la actualidad. Se concluye que independientemente de la gran variedad de simuladores virtuales que existen en la actualidad, los destinados al entrenamiento y entretenimiento son de los más difundidos en la población mundial, de aquí la necesidad de incorporar la detección de infracciones del conductor en el Simulador de auto, con lo cual se lograría dar cumplimiento al objetivo general de este trabajo, por último se realizó un recuento de la trayectoria de la empresa SIMPRO que trabaja junto a la Universidad en la construcción de dicho simulador.

Capítulo II. ***Módulo Infracciones del conductor:***

Se realiza un análisis profundo del módulo de detección de Infracciones del conductor en el simulador, lo cual permite conocer importantes aspectos que ayudan a dar solución a la problemática a la que se enfrenta el presente trabajo, se aborda todo lo referente a los distintos tipos de señalizaciones de tránsito que son necesarias introducir en el simulador y por consiguiente de las distintas infracciones que pueden surgir, además se hace un análisis de los métodos que se utilizan para la detección de las infracciones cometidas por el conductor en el simulador.

Sobre las Infracciones

En cada país hay señalizaciones diferentes de tránsito por lo que se toma como base algunas posibles infracciones universales de tránsito y que son necesarias incluir en el entorno virtual del simulador, se aglutinan un grupo de infracciones que son las que siempre se pueden cometer, pero es uno de los objetivos propuestos la realización de un módulo genérico con posibilidades de incorporar nuevas infracciones sin mucho esfuerzo, ni teniendo que realizar grandes modificaciones en lo ya implementado.

Sobre los Parámetros.

Para lograr la detección de infracciones de tránsito se necesitan varias informaciones provenientes del entorno virtual, de carácter general informaciones que tienen que ver con el auto y en ocasiones otras informaciones que necesariamente no están relacionadas con él. En cada instante de tiempo es necesario conocer toda la información proveniente del auto, tales como, la posición del auto, la velocidad, si tiene encendido algún intermitente, el ángulo de giro del timón, etc. Para dar solución a lo antes expuesto se crea una clase que tiene como atributos todas las variables necesarias y que tienen que ver precisamente con las informaciones provenientes del auto. Desde la Herramienta del Simulador se reciben los parámetros necesarios para crear un objeto de dicha clase, dicho objeto es utilizado por el método responsable de la detección de infracciones.

Sobre las áreas.

En la mayoría de los casos para juegos y otras aplicaciones similares se divide todo el entorno en áreas, en el caso del entorno virtual del simulador sería la calle, pero no es necesario dividir toda una calle para el chequeo de las infracciones sino solo una parte de estas.

Tomando estas ideas se divide todo el entorno virtual en áreas para determinar dentro de cada una de las mismas, qué infracción analizar. Las áreas están enmarcadas en una porción de una cuadra, o en algunos casos en intercepciones de ellas.

Por otra parte se necesita conocer la posición de cada señal en el entorno, quizás no la posición exacta pero si aproximada, pero con solo la posición de la señal no siempre se resuelve este problema, nuevamente se necesita un área para solucionar esta situación.

Todas las áreas que se necesiten para controlar las infracciones en el simulador, se guardan en un fichero, desde la Herramienta del Simulador de auto se cargan las áreas previamente guardadas en el fichero y se almacenan en una colección, este proceso se realiza cada vez que comienza la simulación virtual.

Capítulo III. *Características del sistema:*

En este capítulo se expone el objeto de estudio que sustenta la investigación, se especifican detalladamente las labores que son objeto de automatización para determinar de forma clara el límite y responsabilidad del sistema a construir así como las Herramientas y metodologías a utilizar. Se realiza una descripción del contexto del negocio expresado en un modelo del dominio, se hace un levantamiento de los requisitos funcionales y no funcionales, los requisitos funcionales se estructuran mediante los Casos de Uso del sistema, de los cuales se ofrece una descripción textual. Además se definen los actores del sistema, donde se incluye una amplia descripción de cada una de sus funcionalidades.

Reglas del negocio.

Las áreas correspondientes a las señalizaciones de tránsito incluidas en el Entorno Virtual de la Herramienta del simulador deben ser definidas por un diseñador.

Las áreas definidas por el diseñador deben ser guardadas en un fichero.

La carga del fichero de las áreas debe realizarse al iniciar la Herramienta del simulador.

Requisitos no funcionales.

1. **Soporte:** En una versión inicial deberá ser compatible con la plataforma Windows, pero debe estar preparado para que con pequeñas modificaciones pueda migrar al Sistema Operativo Linux.
2. **Legales:** Se regirá por las normas ISO 9000.
3. **Software:** Sistema operativo Windows.
4. **Hardware:** Compatibilidad con tarjetas gráficas de la familia NVIDIA (Geforce 3, Geforce 4, FX 5200).
5. **Diseño e implementación:** Se regirá por la filosofía de Programación Orientada a Objetos y el lenguaje de programación a utilizar tiene que ser el C++.

Requisitos funcionales generales.

1. **Cargar fichero.**
2. **Clasificar infracciones de tránsito.**
3. **Clasificar infracciones mecánicas.**
4. **Restablecer infracciones.**

Capítulo IV. *Análisis y diseño del sistema:*

En este capítulo se finaliza la etapa de análisis y diseño del sistema, se muestran los principales artefactos UML obtenidos en los flujos de trabajo de diseño e implementación. Se definen gráficamente los diagramas de secuencia uno por cada caso de uso, los cuales son unos de los artefactos más importantes y que requieren de mayor dedicación en esta etapa, se logra obtener un modelo de clases, donde se exponen las clases, sus relaciones y asociaciones, navegabilidad, roles y multiplicidad. Se finaliza esta etapa con el desarrollo de un diagrama de despliegue y uno de componentes, este último permite describir los elementos físicos del sistema y la relación entre los mismos. Además se muestra el modelo de implementación para una mejor descripción de la solución. La nomenclatura utilizada en los diagramas de clases, se puede consultar en el epígrafe "Estándares de codificación".

Conclusiones.

Con el desarrollo del presente trabajo se da cumplimiento satisfactoriamente a los objetivos propuestos. Se desarrolla un módulo que permite la detección de infracciones de tránsito del conductor en el simulador de conducción de auto y se plantea una propuesta con la cual se abre el camino para que otros desarrolladores e investigadores continúen con el desarrollo de la detección de infracciones mecánicas del conductor en dicho simulador. Se completó el diseño e implementación como estaba previsto. El personal involucrado en el desarrollo

del presente trabajo adquirió experiencia en el conocimiento de las distintas reglas que se definieron para el manejo de las infracciones de tránsito y mecánicas cometidas por el conductor, lo cual será de gran utilidad para la futura definición de reglas para los diferentes agentes inteligentes que intervienen en el simulador.

Recomendaciones.

1. Incluir otros tipos de infracciones de tránsito a medida que se vayan introduciendo señalizaciones de tránsito en el simulador.

2. Estudiar para una futura implementación de las infracciones mecánicas la propuesta presentada en este trabajo, con lo cual, se les exhorta a los desarrolladores del Modelo matemático a que realicen un profundo estudio de las variables necesarias que se necesitan obtener de este Modelo para poder implementar de forma exitosa la detección de infracciones mecánicas en el simulador.

3. Se les exhorta a los desarrolladores implicados en la definición de reglas de tránsito para los distintos Agentes inteligentes que intervienen en el simulador, ha que estudien, analicen y por consiguiente tomen como guía la propuesta desarrollada en el presente trabajo en la definición de reglas de tránsito para el conductor del simulador.

4. Realizar la migración del Módulo a Software Libre, ya que la Herramienta del simulador migrará a Linux para disminuir los costos de producción y poder comercializarlo con mayores ganancias y menos limitantes.