SimulationTools
Un pequeño framework para simuladores de entrenamiento

Autor: Sr. Santiago José María Maraggi

Abstract—A lo largo del desarrollo de distintos simuladores en CITEDEF se ha ido resolviendo de manera reiterativa distintas soluciones a la *medida de cada proyecto. SimulationTools surgió como una síntesis de distintas experiencias recabadas en algunos proyectos de manera de ofrecer soluciones flexibles y poderosas a estos problemas. SimulationTools se ofrece como una colección de bibliotecas desarrolladas en C++. Para su desarrollo se puso especial cuidado en las mejores prácticas del paradigma de programación orientada a objetos, de gestión de proyectos y documentación. Proyectos que sigan esta línea de trabajo podrán disponer de servicios para configuración flexible de aplicación mediante archivos XML, establecimiento de conexiones, roles y comunicaciones para aplicaciones distribuidas, captura simplificada de dispositivos de entrada como joysticks, volantes* y otros específicos desarrollados a medida, gestión de eventos de simulación, agentes autónomos con comportamiento de alto nivel incorporados a un motor de simulación flexible, distribución de eventos entre múltiples nodos y sonido espacial con efecto Doppler consistente entre los distintos nodos de la aplicación, sin tener que lidiar con cuestiones de implementación relacionadas a estas funcionalidades.

Index Terms— Simulación, agentes autónomos, aplicaciones distribuidas, diseño de software

1 Introducción

SimulationTools[1] es un conjunto de módulos de software implementados en bibliotecas C++ que prestan servicios de utilidad comunes y reutilizables para múltiples proyectos de distintos tipos de simuladores. Para una mejor concreción de los objetivos propuestos se trabajó con los siguientes principios:

1. **Independencia Interna**. Entre los servicios ofrecidos, de manera que el Proyecto cliente pueda tomarlos independientemente entre sí según sus necesidades.
2. **Independencia Externa**. Respecto de módulos de software externos y dependencias aisladas mediante interfaces y buenos principios de diseño de software.
3. **Simplicidad de uso**. Interfaces simples y prácticas para los proyectos que utilicen los servicios.
4. **Ocultamiento de cuestiones de implementación**. Cuestiones técnicas como liberación de memoria, concurrencia y administración de recursos se hacen transparentes al usuario tanto como es posible.
5. **Programación Orientada a Objetos**. Patrones de diseño[2] y criterios de buen diseño de software[2] [3] [4], facilitando la extensibilidad, adaptabilidad y reutilización de los servicios.
6. **Estándar de código y documentación**. Estilo de código uniforme y documentación online en los archivos de encabezado de código y por separado en formato HTML.
7. **Testing**. Modular e integral para garantizar el correcto funcionamiento de los componentes individualmente y en forma integrada.

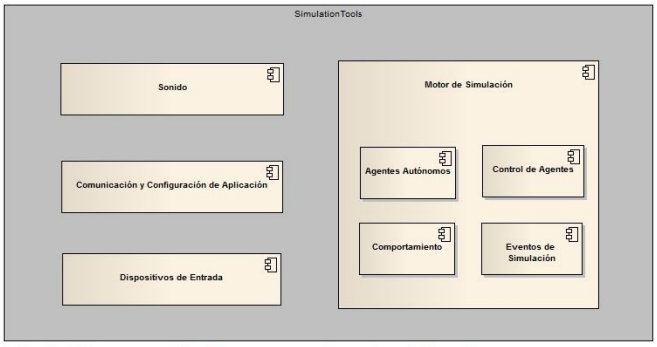

Fig.1 Diagrama de Componentes simplificado que enumera los servicios ofrecidos por SimulationTools.

El desarrollo de bibliotecas comunes facilita a través de los proyectos el mantenimiento, potencia las capacidades, ahorra retrabajo y finalmente costos de mano de obra por proyecto, incrementando la calidad final del software desarrollado. En vez de mantener y asignar recursos a módulos de software diferentes y específicos de cada proyecto, para resolver problemas similares se puede concentrar los mismos en soluciones comunes, aumentando la eficiencia y eficacia de la inversión. Esta unificación de soluciones facilita la articulación de un programa de proyectos cada vez más interrelacionados en la medida en que se potencian módulos que trascienden a

• *El autor es agente RPIDFA III-F-2 de CITEDEF, Instituto de Investigaciones Científicas y Técnicas para la Defensa, B1603ALO, San Juan Bautista de La Salle 4397, Villa Martelli, Buenos Aires, Argentina.*
E-mail: smaraggi@citedef.gob.ar

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

los proyectos y sus especificaciones particulares.
La combinación de servicios independientes también está prevista, de manera de posibilitar resultados integradores de alto nivel y simplificar el lanzamiento de aplicaciones de simulación, permitiendo que el equipo de desarrollo se pueda enfocar más en los requerimientos particulares del proyecto como interfaces de usuario, políticas de desarrollo de la acción e implementación de reglas particulares que en aspectos tecnológicos de bajo nivel.
Desafíos técnicos como el modelado de sonido atenuado en tres dimensiones, efecto Doppler de un avión que hace una pasada rasante o fuentes de sonido con un oyente en movimiento, utilización de dispositivos de entrada como volantes, joysticks u otros construidos a medida y su conexión al modelo de simulación, el establecimiento de conexiones entre nodos en sistemas distribuidos con distintos roles, control de agentes autónomos, como vehículos, que pueden entender órdenes de alto nivel, como perseguir a un enemigo o huir de él, marchar en formación, evadir obstáculos, seguir rutas preestablecidas o todos estos comportamientos y otros superpuestos al mismo tiempo para mayor riqueza del comportamiento, el procesamiento y la generación de eventos de simulación relevantes para el simulador o para los agentes autónomos (que también generan y consumen eventos), consistencia del sonido para simuladores distribuidos entre los distintos nodos mediante difusión de eventos, entre otras facilidades, dejan de ser un problema a abordar desde el inicio para proyectos que aprovechen los servicios de SimulationTools.

2 DESARROLLO

2.1 SimulationTools

Las distintas bibliotecas de SimulationTools son en general independientes entre sí, por lo que cada proyecto puede tomar de estas herramientas las que particularmente vaya a utilizar. Los servicios implementados que serán descritos con más detalle a continuación son los siguientes:

1. **Dispositivos de entrada**
2. **Configuración de aplicación**
3. **Comunicación**
4. **Sonido**
5. **Motor de simulación**
6. **Eventos**
7. **Agentes autónomos**

2.2.1 Dispositivos de entrada

Los dispositivos se describen en este módulo genéricamente como una colección de ejes y botones, cuyos valores reflejan los eventos del hardware correspondiente. Los ejes se modelan mediante un valor real entre un mínimo y un máximo y los botones a través de un estado binario: soltado-presionado. El módulo resuelve internamente la concurrencia entre hilos, captura de eventos del hardware, actualización de estado e inyección de eventos al software mediante interfaces genéricas y funciones que se pueden configurar dinámicamente de una manera computacionalmente eficiente y práctica. El módulo uniformiza el trato para todo tipo de dispositivos a la vez que presenta servicios sencillos, absolutamente flexibles y prácticos para el usuario. Mediante la instanciación de clases se puede corresponder hardware de distintas características con los eventos que serán levantados desde el software. También desde luego está contemplada la instanciación de varios dispositivos de entrada simultáneamente, de manera que se pueden utilizar volantes, pedaleras y botoneras con forma conjunta. En casos de aplicación concretos se ha utilizado este módulo levantando varios joysticks asignados a controladores de aplicación con los resultados esperados.

2.2.2 Configuración de aplicación

Este módulo posibilita al programa cliente especificar los distintos parámetros y posibles valores que espera levantar de un archivo XML de configuración de aplicación. El módulo se encarga de leer el archivo, identificar todos los valores, validar el tipo y los valores de las variables leídas y finalmente configurar la aplicación. El módulo también implementa un rearmado de una estructura de directorios, de manera que permite definir direccionamiento indirecto de archivos tanto hacia niveles superiores como inferiores en la jerarquía respecto del directorio de ejecución del programa, de manera de establecer la estructura de directorios de la configuración de entorno local del programa en forma integral. Los programas cliente con muy pocas líneas especifican la configuración que desean levantar de un archivo mediante un formato XML estandarizado e interpretación parametrizada.

2.2.3 Comunicación

Este módulo permite especificar la configuración de las distintas conexiones y roles de cada nodo de una aplicación mediante archivos XML. Cada puesto de trabajo o nodo de la aplicación interpretará su archivo de configuración de conexiones y se conectará según los parámetros establecidos en los mismos.
Se implementó para este módulo un modelo cliente-servidor, pero que aún permite la definición de múltiples roles para cada nodo. Estos roles luego se pueden corresponder con otras conexiones que no tengan que ver con el rol principal de cliente servidor.
El módulo brinda los servicios de conexión, manejo de excepciones, envío y recepción de mensajes y brinda interfaces genéricas para que la aplicación dé el curso deseado a las distintas situaciones, así como también y especialmente a la facilidad para definir tipos de mensajes propios. El módulo permite además definir estados de los puestos y ofrece interfaces con los proxy locales según los distintos estados de conexión. Para cada nodo o puesto se define un nombre y se especifica por configuración XML su dirección IP y puerto para el establecimiento de las conexiones.
Estas comunicaciones se establecen por el momento sobre el protocolo TCP/IP, de manera de garantizar el funcionamiento de protocolos de control y la confir-

mación de los envíos. Estas conexiones están pensadas para ser utilizadas a nivel aplicación y no comprenden la visualización distribuida; tal funcionalidad se relega a un motor de visualización como podría ser el PGS / P3D desarrollado en CITEDEF [7] o algún otro que contemple la distribución de la visualización (OpenSceneGraph, Vega Prime de PRESAGIS, etc.), y para lo cual por lo general se utiliza una conexión UDP.
A través de la herencia, la aplicación cliente puede disponer de todas estas funcionalidades con muy poco código y distribuir un archivo de configuración de conexiones a cada puesto, manteniendo el formato provisto, para configurar todas las conexiones, que serán establecidas de manera automática por el módulo "EnlaceLib".
Se eligió el protocolo TCP/IP porque las latencias de conexiones de red local permiten solicitar una conexión con garantía de entrega como ofrece este protocolo sin afectar el rendimiento de la aplicación. Como algunos mensajes de SimulationTools pueden afectar la lógica del programa cliente según cómo se utilice, se prefirió a este nivel utilizar un socket TCP en vez de UDP como podría ser más habitual para otras aplicaciones en que algunas pérdidas son tolerables. Reemplazar este socket por uno UDP es una alternativa abierta a futuras consideraciones.

2.2.4 Sonido

La biblioteca SoundLib ofrece una interfaz sencilla e intuitiva para manipular sonidos. El servicio contempla la reproducción de sonidos con distintos modelados de atenuación espacial, posición y velocidad de fuentes de sonido y del oyente propiamente dicho, efecto Doppler y administración de buffers de sonido centralizado, de manera de lograr eficiencia en la utilización de recursos para múltiples fuentes de sonido que reproducen un mismo archivo.
La biblioteca SoundLib envuelve los servicios de la biblioteca OpenAL e incorpora servicios de paralelismo, administración de recursos y sobretodo interfaces muy simples cuyos métodos no precisan mayor referencia a documentación de biblioteca mencionada. La centralización en el manejo de recursos, en particular de los buffers e hilos de procesamiento, simplifica la tarea respecto de una utilización directa de la biblioteca de más bajo nivel OpenAL y logra cumplir los servicios planteados a un costo computacional bastante bajo.
SoundLib permite manipular fuentes de sonido como posicionales o como ambientales, siendo que en este segundo caso no se considera la posición ni movimiento de la fuente de sonido ni la del propio del oyente. OpenAL no prevé la actualización de la posición de las fuentes de sonido en función de su velocidad, sino que la velocidad vectorial es utilizada solamente a los efectos del cálculo del desplazamiento de frecuencia de reproducción debida al efecto Doppler respecto del oyente. SoundLib implementa un hilo paralelo a la aplicación que se encarga de actualizar las posiciones de todas las fuentes de sonido y del oyente mismo. Este servicio de mantenimiento y actualización de las fuentes de sonido y de la posición del oyente se integra con el motor de simulación que se describe en el apartado siguiente. La dependencia entre ambos módulos es a través de una interfaz genérica, por lo que el módulo de sonido se puede integrar con cualquier otra herramienta que se necesite adaptar a la misma. SoundLib incorpora simplicidad mediante interfaces, orientación a objetos y nuevas funcionalidades a la biblioteca OpenAL, manteniendo a la vez la riqueza de esta poderosa biblioteca, pudiendo por ejemplo elegirse entre distintos modelos de atenuación sonora y configurando coeficientes que alteran el modelado del sonido y su relación con el espacio, etc.
SoundLib se puede invocar independientemente de otras bibliotecas de SimulationTools y el servicio de actualización posicional es opcional. SoundLib es, por lo tanto, viable para utilizar incluso en pequeñas aplicaciones o pruebas de lo más triviales, además de brindar servicios aptos para aplicaciones más demandantes.
El usuario puede administrar la carga y descarga de los buffers de sonido en todo momento, así como delegar la limpieza final en SoundLib, que al cerrarse descargará todos los buffers y concluirá hilos que hayan podido quedar abiertos. La aplicación cliente de esta manera se descarga de bastantes tareas administrativas, a veces tediosas, a través de SoundLib en lo que a manejo de sonido respecta, pero manteniendo el poder de decisión sobre la carga y descarga de recursos para cuando fuere necesario una administración meticulosa.
SoundLib no prevé la implementación de sonido para múltiples nodos o puestos de simulación, sino que brinda los servicios de sonido para un único oyente. Para lograr este cometido se debe combinar los servicios de SoundLib con el motor de simulación (descrito a continuación), y la administración distribuida de eventos de simulación. Mediante la combinación de estos servicios se logra que cada entrenando o participante de la acción en el curso de una simulación pueda percibir el sonido acorde a su situación particular en relación con el modelo de simulación: distinta posición del oyente, retardos de propagación del sonido, efecto Doppler diferente, descarte de sonidos lejanos, etc.

2.2.5 Motor de simulación

Este módulo es la base para la implementación principalmente de otros dos servicios de más alto nivel y que se describen a continuación: agentes autónomos y eventos de simulación.
Para no condicionar al programa cliente se mantuvo el motor al mínimo, se aislaron las dependencias por interfaces y se logró de esta manera proveer la fun-

cionalidad de dar vida a los agentes autónomos que pueblan un caso de simulación. El motor se puede poner en marcha, detener, y permite incorporar o eliminar nuevos agentes a pedido de la aplicación cliente en cualquier momento, así como también definir operaciones particulares adicionales. El motor de simulación es el nexo entre las entidades (móviles o estáticas), el escenario en que se desarrolla la acción y los eventos que ocurren.

Este módulo ofrece interfaces para consulta e interacción con las distintas entidades que participan de la simulación, garantiza la concurrencia segura en la ejecución de la simulación y la sincronización con todos los eventos externos, sean estos operaciones del usuario, mensajes de aplicación o surgidos del procesamiento de eventos propios de la simulación. El motor tiene un paso propio que determina la evolución de la acción, aunque esta iteración no está ligada a la generación de los cuadros por segundo de la visualización. La sincronización entre el modelo y el engine gráfico se basa en una consulta al engine gráfico con una frecuencia determinada. Con esta implementación no se han apreciado saltos ni incongruencias entre los dos modelos en los ensayos realizados, así como tampoco en el proyecto en el que se ha utilizado este módulo de software.

Un submódulo del motor permite su integración automática con la plataforma P3D / PGS anteriormente mencionada [7], y es aplicable análogamente para su integración con otros motores gráficos. Esto se logra mediante la implementación concreta de interfaces genéricas en un módulo específico a tal fin.

El motor provee modelos básicos para entidades, entidades móviles y vehículos, siendo sucesivamente estas clases una heredera de la otra y a su vez más ricas en comportamiento. Los vehículos por ejemplo tienen asociada una fuente de sonido, tal como se las define en la biblioteca SoundLib mencionada en el apartado anterior.

Si bien este servicio es central para un simulador que lo utilice, se tuvo el especial cuidado de no ser invasivo respecto de las incumbencias más particulares que lógicamente pueda tener cada proyecto, tratando de mantener el servicio y sus clases al mínimo. El comportamiento de todas las clases es sobradamente extensible y los servicios se brindan a través de interfaces genéricas. SimulationTools es el nombre de la biblioteca que implementa el servicio de motor de simulación y tiene el mismo nombre que la colección completa de todos los servicios que componen este trabajo.

2.2.6 Eventos de simulación

Los agentes autónomos a los que anima el motor de simulación pueden reaccionar a eventos que ocurren en la simulación, según sean o no de su interés. A su vez, los agentes autónomos en su curso de acción como agente autónomo, configurar y manipular pueden generar eventos que sean significativos para la aplicación (concreción de un objetivo) o para otros agentes (amenaza enemiga) y la aplicación misma o el usuario del sistema pueden generar también eventos en sus distintas interacciones. Este módulo ofrece el servicio para creación, difusión y consumo de los eventos de simulación de la aplicación de una manera sencilla e integrada al motor de simulación descripto previamente.

En caso de ser utilizado en una aplicación distribuida, este módulo está diseñado para su sencilla integración con el módulo de comunicaciones “EnlaceLib”.

Este módulo se denomina “SimulationEvents” y resuelve todos los aspectos técnicos relacionados con la gestión de eventos, así como también brinda facilidades para que la aplicación pueda definir sus propios eventos particulares. Los eventos de aplicación pueden contener información de interés para su procesamiento y son directamente serializables por herencia para su envío mediante el módulo de comunicaciones.

El motor de simulación utiliza un evento periódico para actualización del estado de aplicación en nodos remotos cuando se activa el mismo para funcionar de esa manera. Como se mencionó en el apartado de SoundLib, los sonidos también se gestionan en aplicaciones distribuidas mediante emisión y consumo de eventos específicos. Así mismo determinados eventos como inicio de simulación, destrucción de entidad y varios más son provistos directamente por SimulationEvents a la aplicación cliente. La aplicación cliente puede, desde luego, extender estos eventos y generar otros nuevos. La política de gestión de eventos tiene mucha correlación con lo que la aplicación defina como reglas de la simulación, reglas que son determinadas por los requerimientos particulares de cada simulador. Utilizando SimulationTools el equipo de desarrollo puede centrarse en estas políticas desde el inicio y no perder tiempo en aspectos técnicos de más bajo nivel.

SimulationEvents además permite definir los eventos como reportables o no reportables, para que la aplicación los defina de forma que al concluir un ejercicio de simulación pueda confeccionar un reporte con los eventos significativos.

2.2.7 Agentes autónomos

Son los agentes que actúan de manera autonómica, siendo capaces de tomar decisiones independientemente y realizar acciones para satisfacer objetivos internos basados en el entorno que perciben. SimulationTools ofrece a las aplicaciones que la utilicen el servicio de agentes autónomos con comportamiento configurable a alto nivel. Con el motor de simulación en marcha una aplicación puede instalar un vehículo

viamente o en tiempo de ejecución su comportamiento con órdenes como perseguir a un enemigo o huir de él, dirigirse a un lugar esquivando obstáculos, marchar en formación siguiendo a un líder y esquivando obstáculos, seguir una ruta preestablecida o todo tipo de combinación entre estos y otros comportamientos, de manera que se enriquezca el resultado final. Con SimulationTools la aplicación tiene una gama amplia y rica de posibilidades para generar comportamientos con significado según sus requerimientos, a la vez que retiene el control total sobre el mismo a través de órdenes directas a los agentes autónomos mediante una sencilla e intuitiva interfaz. Con los servicios de agentes autónomos de SimulationTools se puede modelar desde una vaca pastando, camiones en marcha, bandadas de pájaros o aviones volando en formación, hasta un misil guiado que persigue a su objetivo.

El motor de simulación antes descrito es el que permite la movilidad de las entidades de simulación. El servicio de agentes autónomos es el que ofrece el servicio de entidades autónomas con comportamiento propio, con un sentido semántico concreto y delegando el control real en la aplicación que los utiliza.

Hasta la implementación de la clase *Vehiculo* en la biblioteca SimulationTools el servicio que se proveía dejaba librado a la aplicación que los utiliza a movilizar desde el código y al más bajo nivel a los vehículos, es decir, manipulando atributos como posición y velocidad de manera individual, y librado a su capacidad para gestionar un ejercicio de prestidigitación semejante. El servicio de agentes autónomos de SimulationTools viene a paliar esa situación, reemplazando la necesidad de manipular vehículos en extremo detalle por la posibilidad de dar órdenes de alto nivel, directas y significativas, que el agente interpreta y ejecuta de forma autónoma.

Cada agente autónomo tiene una instancia de *SteeringManager*. Esta clase provee una interfaz intuitiva para gobernar el comportamiento de los agentes autónomos, desligando a la aplicación que utilice este servicio de su manipulación. Para implementar este servicio se usó en parte el diseño propuesto por Mat Buckland [5] y se tomaron elementos del trabajo de Creig Reynolds [6]. La técnica de Steering Behaviors está ampliamente difundida desde hace tiempo en la industria de los videojuegos y simuladores. Aunque este modelo tiene algunas limitaciones que es preciso tener en cuenta, en particular respecto del realismo físico del modelo cinemático de vehículos terrestres y aéreos, se pueden lograr comportamientos muy aceptables, al menos para una amplia gama de requerimientos, para entidades que no sean directamente manejadas por un usuario, configurando y calibrando correctamente los distintos comportamientos y parámetros relacionados.

El modelo de Steering Behaviors plantea un comportamiento como la sumatoria de fuerzas de distintos comportamientos atómicos que se pueden superponer para dar por resultado una fuerza de comportamiento resultante que es la que movilizará finalmente al vehículo. Por ejemplo, la orden de dirigirse desde la ubicación actual hacia un lugar A generará una fuerza de volanteo (Steering Force), con orientación desde el agente hacia el punto A e intensidad determinada por el tipo de vehículo o algún parámetro si lo hay. Si además el agente debe esquivar obstáculos, ante la aparición de un obstáculo colisionable en el camino se generará una componente de fuerza lateral, tal que la sumatoria de fuerzas lo sigue acercando al punto A al que debe dirigirse, pero con una nueva fuerza lateral causada por la percepción del obstáculo que debe evitar, que determinará un desvío respecto de la ruta original. La superposición de estos comportamientos atómicos permite lograr con relativa sencillez comportamientos compuestos en apariencia de gran complejidad. Para mayor uniformidad y claridad se mantuvieron los nombres en inglés de las steering forces o fuerzas de volanteo. A continuación se describen sus nombres, significados y los parámetros que utiliza cada una:

- **Seek(A)**. Dirigirse al punto A.
- **Arrive(A, K)**. Llegar al punto A deteniéndose suavemente con una constante de frenado K.
- **Flee(A, D)**. Huir del punto A en tanto se esté a una distancia menor a D (D es opcional).
- **Wander (R, D)**. Merodear. Se modela mediante una esfera de radio R a una distancia D por delante del agente autónomo, en la que se sortea la oscilación del punto destino imaginario al que se dirige el agente.
- **Obstacle Avoidance (K, L)**. Evadir obstáculos con constante de frenado K y largo de detección L.
- **Evade(V)**. Evadir agente autónomo V (huir de otro vehículo).
- **Persuit(V)**. Perseguir a otro agente autónomo V (persecución).
- **Offset Persuit(V, P)**. Seguir a otro agente autónomo manteniendo una posición relativa P (muy utilizado para vuelos de aeronaves en formación, o bandadas de pájaros, por ejemplo).

- **Path Following(C)**. Seguir el camino C (colección de puntos).

De esta manera, al *SteeringManager* de un *Vehiculo* se le puede dar simultáneamente todas estas órdenes: dirigirse a un lugar, evitar un lugar, evitar obstáculos, huir de un agente, arribar suavemente a un lugar, perseguir o escoltar a otra entidad móvil o seguir una ruta preestablecida. Todos estos comandos son muy ricos en significado para generar situaciones de todo tipo, en tanto la aplicación también retiene el control total sobre los agentes a través de estas órdenes y puede darles instrucciones precisas en todo momento. Se presta el servicio de agentes autónomos con gran riqueza semántica y delegando en la aplicación la política de gobierno de estos agentes, de manera que pueda ajustar todo a los requerimientos particulares de la mejor manera para generar situaciones y comportamientos deseados sin tener que manipularlos a bajo nivel.

3 Resultados obtenidos

Todos los módulos de software desarrollados en SimulationTools fueron probados extensivamente. En todas las pruebas se terminó obteniendo el resultado esperado, y además de los resultados directamente observables, tanto en calidad visual como en fluidez del curso de acción, se constató el correcto cierre y reapertura de todos los módulos sin *memory leaks*. Para constatar este hecho se utilizó la herramienta Visual Leak Detector en el IDE Visual C++. Todas las pruebas y aplicaciones que corrieron sobre SimulationTools utilizaron el sistema operativo Microsoft Windows 7 con equipos de al menos entre unos dos a tres años respecto de la fecha de esta publicación, siendo la única distinción mencionable que se utilizaron placas de video de gama media de aquella época, es decir, hardware absolutamente corriente y disponible.

Posteriormente a ser probados todos los módulos individualmente y su integración conjunta con casos de prueba desarrollados ad-hoc, SimulationTools fue incorporado para dar soporte al proyecto SITARAN II SECCIÓN del Ejército Argentino, desarrollado en CITEDEF. Este proyecto modela la operación de cañones antiaéreos en distintos puestos, por lo que es una aplicación distribuida, en la que se utilizan distinto tipo de vehículos aéreos y terrestres que deben ser batidos o no por los operadores de las piezas. Para este proyecto se diseñaron otros componentes propios que exceden a este trabajo, como ser un módulo de inteligencia artificial que opera sobre los servicios de agentes autónomos ofrecidos por SimulationTools. La facilidad para extender, adaptar y personalizar los servicios ofrecidos por SimulationTools fue uno de los objetivos primarios de este trabajo.

Aparte de las pruebas atómicas de rigor, hubo tres casos de prueba principales en que se integraron las distintas funcionalidades de SimulationTools.

Fig.2 Vehículos terrestres rompen temporalmente la formación mientras evaden un obstáculo, siguiendo una ruta predefinida. Otro par de vehículos está en modo persecución y evasión respectivamente.

En el primer caso de prueba se instanció un vehículo terrestre en un terreno plano con un gran obstáculo esférico en el medio. El vehículo recorre el pequeño terreno de un lado a otro esquivando el obstáculo cuando éste se encuentra a la distancia de detección configurada.

En un segundo caso de prueba se instanciaron más vehículos con distinto tipo de comportamientos. Un grupo de tres vehículos terrestres marcha en formación siguiendo un circuito preestablecido y esquivando obstáculos, uno de los cuales se encuentra justo en el camino que tienen preestablecido. El resultado fue óptimo: los vehículos marchan en formación siguiendo la ruta preestablecida y cuando se encuentran el obstáculo el líder primero tuerce el rumbo para esquivarlo y luego hacen lo propio los escoltas; la formación se desarma parcialmente durante el proceso de esquivado del obstáculo pero se rearma cuando el mismo queda atrás y los vehículos ya quedan libres de continuar su rumbo. Otros vehículos terrestres prueban los comportamientos de persecución y evasión satisfactoriamente también. Se incluyó un grupo de tres aeronaves que vuelan también en formación, lanzándose en picada ofensiva sobre el obstáculo que obstruye el circuito de los vehículos terrestres.

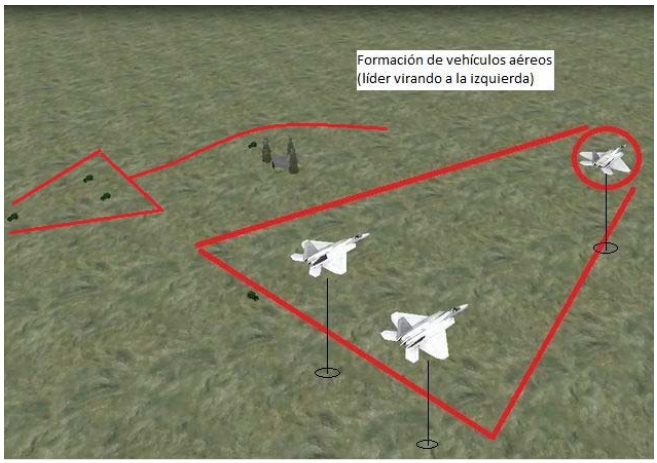

Fig.3 Vehículos aéreos en formación hacen su aparición, mientras que los vehículos terrestres continúan su ruta tras recomponer la formación de marcha preestablecida.

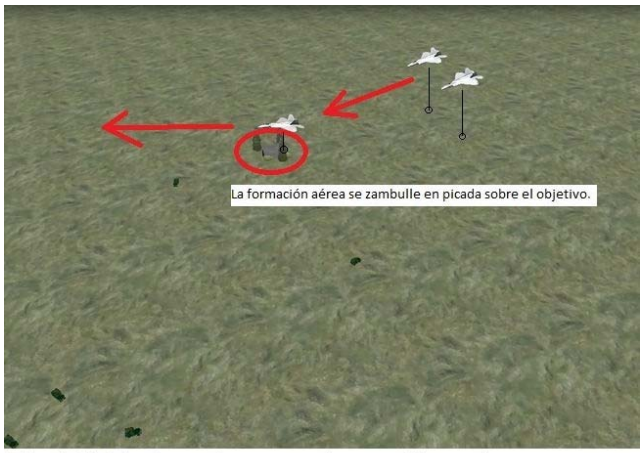

Fig.4 Vehículos aéreos en formación se lanzan en picada sobre el obstáculo del terreno y prosiguen su marcha luego del ataque.

El último caso de prueba es por demás el más interesante. Se modelaron varias formaciones de vehículos, se les asignaron trayectorias y comportamientos, se instanció el módulo de sonido espacial, se incorporó un joystick para controlar la cámara de visualización y se lanzó a correr la escena. De los aviones, se aprecia que realizan pasadas rasantes de tipo ofensivo con un realismo aceptable y el comportamiento esperado. El efecto Doppler enriquece la escena entre las pasadas de aviones veloces y helicópteros cercanos. Un grupo de tanques pasa cerca de los restos de un galpón y esquivan el obstáculo, marchando también en formación. El ensayo logra generar una situación fluida, con vehículos autónomos con comportamiento propio y una sensación inmersiva subjetivamente aceptable, al menos satisfaciendo las expectativas iniciales y funcionando dentro de los parámetros esperables.

Fig.5 Pasaje rasante de aviones en formación y helicópteros. El comportamiento de los vehículos en conjunción con el sonido posicional, efecto Doppler y control de cámara con joystick logra una situación inmersiva dentro de los parámetros esperables.

7 Secciones finales

7.1 Agradecimientos

A mis compañeros de trabajo de CITEDEF en ocasión de poner en común este trabajo por el aliento, los buenos auspicios para presentar este trabajo posteriormente a su puesta en común y comentarios técnicos de suma utilidad, en particular sobre la utilización de la plataforma gráfica P3D / PGS, desarrollada ad-hoc en CITEDEF [7], Ing. Lucas Guaycochea, Ing. Eduardo Calderón, Sr. Federico García del Río, Ing. Javier Luiso y Lic. María Victoria Galán.

7.2 Referencias

Para los distintos ensayos en que se probaron las funcionalidades de SimulationTools se utilizó el motor gráfico PGS / P3D desarrollado ad-hoc en CITEDEF [7].
Esta Plataforma de Desarrollos 3D Distribuidos o Plataforma General de Simulación ofrece servicios gráficos para aplicaciones integrados a una capa de red para garantizar la consistencia de la visualización distribuida.
La dependencia de SimulationTools respecto de esta plataforma está aislada en un set de implementaciones particulares de interfaces generales propias de SimulationTools, de manera de facilitar la integración de estas herramientas a cualquier engine de simulación y aislar, cuando menos, estas dependencias.

7.4 Trabajos a futuro

En proyectos con algún requerimiento específico para funcionar en múltiples plataformas podrían reemplazarse algunas bibliotecas de servicios externas a SimulationTools desarrolladas ad-hoc que trabajan específicamente sobre la API de Windows.
En la medida que el uso de los agentes autónomos se hiciera repetitivo se podrían mejorar las parametrizaciones de los comportamientos de los agentes. Este calibrado es un trabajo muy común cada vez que se implementan soluciones con Steering Behaviors [5]. Así mismo se podrían implementar algunos comportamientos que al momento no han sido incluidos acabadamente como merodear (“Wonder”) [5], [6]. Previendo esta posibilidad todos los parámetros de Steering Behaviors se abrieron a las interfaces que se brindan al cliente para que pueda ajustar los comportamientos a su necesidad.
Un modelo más preciso de colisiones podría ser implementado sobre los servicios ya ofrecidos. Este servicio debería consultar las geometrías específicas de los distintos objetos, típicamente utilizando algún servicio provisto por el motor gráfico, servicio que en este caso no estaba aún disponible. La inclusión de un servicio de consulta de colisiones más sofisticado o instalación de callbacks podría ser un curso natural para este trabajo, ya que de momento se ofrece una sencilla de consulta de colisiones modelada con una bounding sphere. Por falta de tiempo, esa tarea se realizó directamente en una aplicación cliente, utilizando los servicios de eventos de simulación de SimulationTools para la generación de disparos, lo cual demostró que la infraestructura ofrecida por estos servicios resultó extensible de la manera esperada.
Cuando se puso en común este trabajo al equipo de desarrollo

me han propuesto que la configuración de conexiones entre los nodos de una aplicación distribuida se pudiera realizar mediante un archivo diferente para cada conexión, en lugar de utilizar un único archivo de configuración global, de manera de simplificar los mismos y minimizar los errores al realizar estas configuraciones. Quedó el planteo como una idea razonable para una posterior adaptación si fuera necesario u oportuno llevarla a cabo.

7 Conclusiones

SimulationTools cumplió satisfactoriamente todas las pruebas implementadas para verificar el cumplimiento de los objetivos propuestos en cuanto a funcionalidades y calidad de servicio. Posteriormente el simulador del Ejército Argentino SITARAN II SECCIÓN utilizó satisfactoriamente y sin novedad estas herramientas, aun transfiriendose el material a otro desarrollador para la conclusión del proyecto, en gran parte gracias a las buenas prácticas de documentación y desarrollo implementadas, la relativa sencillez en las interfaces para el consumo de los servicios y los principios de diseños aplicados.

Existen en el mercado otras soluciones similares provistas por productos comerciales muy usadas en desarrollo de videojuegos y también en simuladores, que por lo general implican pagos de licencias o limitaciones de uso y en general poco control sobre aspectos internos, aunque no sean necesarios en la mayoría de los casos. La elección de las herramientas a utilizar es una de las más importantes que tiene a cargo un director de proyecto y es en respeto a esas decisiones que en SimulationTools se implementaron los servicios de manera independiente entre sí tanto como fue posible, de manera de condicionar el diseño del proyecto que utilice alguno de ellos al mínimo.

Agradecimientos

Al Instituto CITEDEF en que me desempeño, al Ing. Horacio Abbate, Jefe de División Computación Gráfica, e Ing. Eduardo Álvarez, Jefe del Departamento de Simulación y Guiado.

Referencias

[1] S. Maraggi, "SimulationTools: una colección de soluciones para simuladores," ISSN 0325-1527, Nota Técnica N° 934, CITEDEF (Instituto de Investigaciones Científicas y Técnicas para la Defensa), Ministerio de Defensa.

[2] E. Gamma, R. Helm, R. Johnson y J. Vlissides, "Design Patterns: Elements of Reusable Object Oriented Software".

[3] M. Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language".

[4] B. Stroupstrup, "The C++ Programming Language".

[5] M. Buckland, "Programming AI By Example".

[6] C. Reynolds, "Steering Behaviors For Autonomous Agents".

[7] Lucas Guaycochea, Javier Luiso, María Victoria Galán, Horacio Abbate., "Plataforma de Desarrollos 3D Distribuidos (P3D)," 45 JAIIO, SIE 2016.

First A. Author Santiago Maraggi se encuentra cursando las últimas materias de la carrera Ingeniería Informática en la Universidad de Buenos Aires. El autor es Profesor Instructor de Tiro categoría "A" (Licencia ITA 5951). Este trabajo ha sido publicado por CITEDEF (Instituto de Investigaciones Científicas y Técnicas para la Defensa) con ISSN 0325-1527, Nota Técnica N°934 y ha sido aplicado al proyecto SITARAN II SECCION del Ejército Argentino. El autor trabajó en proyectos de simulación NeoNahuel II (Caballería, simulador de Tanques TAM), SIMOA III (simulador de Observador Adelantado de artillería), SITARAN II SECCION (artillería antiaérea) y SIMRA II / SIMRA III (simulador de tiro). El autor se desempeña como agente RPIDFA Categoría III-F-2 en el mencionado Instituto en la División Computación Gráfica del Departamento de Simulación y Guiado. El autor además es Coordinador Nacional de Tiro Deportivo en la Federación del Deporte Universitario Argentino y representó al país en Juegos Olímpicos Universitarios en dos ocasiones como tirador.