SISTEMAS INFORMÁTICOS

Curso 2005/2006

Simulación Distribuida Sobre Redes de Servicios Web/Grid:

Simulador de trayectorias de aviones

Autores:

Enrique Aguilera Sanz
Sergio Cuesta Boluda
A. Julián Martínez de la Casa Santos

Director de proyecto:

José Jaime Ruz Ortiz – DACYA

Universidad Complutense de Madrid

Facultad de Informática

PRESENTACIÓN

Este proyecto consiste en el diseño e implementación de un simulador de trayectorias de aviones dentro de escenarios, haciendo uso de la tecnología de los Servicios Web mediante la plataforma .NET.

Dos son los principales objetivos que se intentan alcanzar. Por un lado, la programación del simulador, y por otro, su descomposición en módulos independientes que podrían ser distribuidos por distintos computadores comunicándose a través de la red mediante Servicios Web.

En ese sentido, se ensayará una partición cliente-servidor, con dos módulos ejecutables en el cliente como son el Generador de Escenarios y el Visualizador de los Resultados, mientras que el sistema de Cálculo de Trayectorias y Simulación podrán encontrarse alojados en un servidor para ser accedido desde el cliente.

La simulación realizada se centra en la generación de trayectorias de aviones hasta un objetivo, de forma que se esquiven obstáculos y se eviten radares en la medida de lo posible. El núcleo central es un algoritmo A Estrella.

ABSTRACT

This project consists in the design and implementation of a planes paths simulator for scenarios, using the Web Services technology of the .NET platform.

There are two main goals to achieve. First, the programming of the simulator, and then, its decomposition into independent modules to allow the distribution of the system among different computers which use Internet Web Services to communicate.

An example of client-server partition will be tested, with two modules to be run in the client, as the Scenarios Generator and the Results Viewer, while the Paths Calculator and Simulation system will be installed in a server and will be called by the client.

The developed simulation involves calculating planes paths to a target, avoiding balks and radars if possible. The core of this module is an A Star algorithm.

ÍNDICE

1 TÉRMINOS, ACRÓNIMOS Y ABREVIATURAS

A*: algoritmo Astar para el cálculo de caminos óptimos.

C# : C-Sharp

DTD: Document Type Definition

XML: eXtensible Markup Language

UAV: vehículos aéreos no tripulados (concretamente los aviones).

2 INTRODUCCIÓN

Este proyecto plantea el diseño e implementación de un entorno para la simulación distribuida entre diferentes computadores conectados a Internet utilizando Servicios Web.

Al tratarse de una simulación de un sistema real, al crecer el tamaño del problema, la complejidad de los cálculos, o la precisión de las soluciones, la potencia de cálculo podría convertirse en un aspecto crítico y la necesidad de distribución de los procesamientos en una necesidad potencial. Por ello será importante tener presente la división del proceso de simulación en módulos independientes, con el objetivo de facilitar por un lado la mejora de procesos sin interferir en los demás , y por otro la futura posibilidad de distribución en diversos computadores empleando concurrencia.

La simulación consiste en el cálculo de trayectorias para diversos elementos móviles, en concreto aviones, dentro de un escenario y con el fin de alcanzar un objetivo. Aunque el núcleo central sea la búsqueda de la trayectoria óptima, una parte importante de este módulo será el procesado posterior de la trayectoria para hacerla más realista, con métodos como interpolado o la generación de la dinámica del avión con ángulos de trayectoria y de giro. En la búsqueda de la trayectoria óptima dos factores influirán. Por un lado, la distancia del recorrido, y por otra la probabilidad de que los aviones sean detectados por posibles radares definidos en el escenario.

Por otro lado, el sistema también incluirá un proceso de simulación en el que cada elemento móvil se dirigirá hacia el objetivo según sus propias trayectoria y dinámica previamente calculadas, pudiendo ser detectado por los radares del escenario, e incluso derribado por sus baterías de misiles asociadas.

Finalmente, para facilitar la creación de escenarios y visualización de resultados, se plantea el desarrollo de dos módulos de interfaz gráfica como son el generador de escenarios y un módulo visualizador, ambos bajo DirectX 9.0.

2.1 OBJETIVOS

A continuación, se definen los objetivos u requisitos a conseguir en el proyecto. Posteriormente se encontrarán más en detalle en cada sección correspondiente. También se reflejarán las posibles mejoras que se han

tenido en cuenta pero no implementado, y los defectos que han quedado pendientes de solucionar.

El proyecto está dividido en cuatro fases bien diferenciadas:

1. Definición del escenario de actuación, es decir, la parte correspondiente al editor, donde podremos elaborar los escenarios que consideremos adecuados.

2. Cálculo de las trayectorias de los móviles (aviones, misiles) para conseguir sus objetivos, en el caso del avión, su objetivo será llegar a un determinado objetivo, en el caso del misil, derribar el avión detectado por un radar que tiene asociado dicho misil.

3. Simulación de los cálculos realizados, para que podamos evaluar el éxito o fracaso de las trayectorias calculadas. Para una misma trayectoria calculada, habrá ocasiones en que logre su objetivo mientras en otras el UAV será derribado, dependiendo de la amenaza de los radares en el recorrido.

4. Visualización de los resultados en un entorno 3D, para que podamos comprobar visualmente los resultados calculados.

Las fases 2 y 3 que son las que requieren mayor carga computacional y se prepararán para poder ejecutarse en el computador servidor, además desde local. Las fases 1 y 4 se ejecutarán en el cliente siempre. La comunicación cliente-servidor se realizará con servicios WebXML.

2.1.1 ENTRADA Y SALIDA DEL SISTEMA

Si vemos la aplicación total, incluyendo los 4 módulos principales, podemos definir las siguientes entradas y salidas globales:

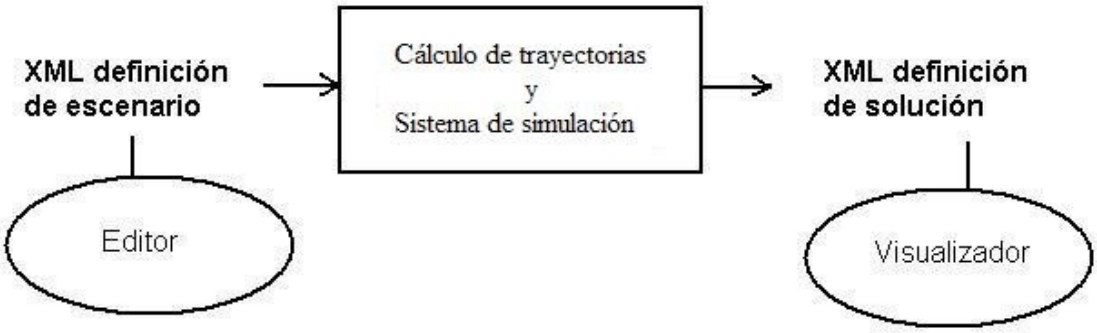
Entrada: fichero .XML de escenario. Es el fichero generado por el Editor de escenarios, conteniendo toda la información de posibles objetos dentro del mismo, como por ejemplo las coordenadas del objetivo dentro del mapa, los obstáculos, etc...

Este fichero también podría ser generado por otra aplicación, o incluso manualmente, siempre que se respeten sus definiciones con la correspondiente DTD.

En el caso de uso de servicios web, será el XML que se envíe al servidor.

Salida: fichero .XML con la solución. Contendrá las trayectorias seguidas por los aviones junto con los instantes de tiempo, la de los misiles, etc... Es decir, la simulación final de la entrada. Este fichero está preparado para ser utilizado en el módulo visualizador.

En el caso de uso de servicios web, será el XML recibido desde el servidor tras todos los cálculos.



2.2 ARQUITECTURA HARDWARE Y SOFTWARE

2.2.1 HARDWARE

Para el desarrollo del proyecto se han empleado equipos trabajando sobre Microsoft Windows, programándose en C# con la plataforma de desarrollo *Visual Studio .NET Enterprise Architect Edition*. El servidor para alojar los servicios web trabaja sobre Windows 2003 con IIS 6.0.

2.2.2 SOFTWARE

En cuanto la arquitectura software, cada módulo creado se corresponde con un proyecto de Visual Studio. La distribución de todos estos módulos del sistema se encuentra bajo una misma ruta en la versión local, junto con otro directorio con los datos que requieren.

El prototipo ofrece la posibilidad de emplear la versión local del cálculo de trayectorias y simulación, o la versión con uso de servicios web. Para este segundo caso, los módulos de software correspondientes (servicios web) se encuentran alojados en el servidor.

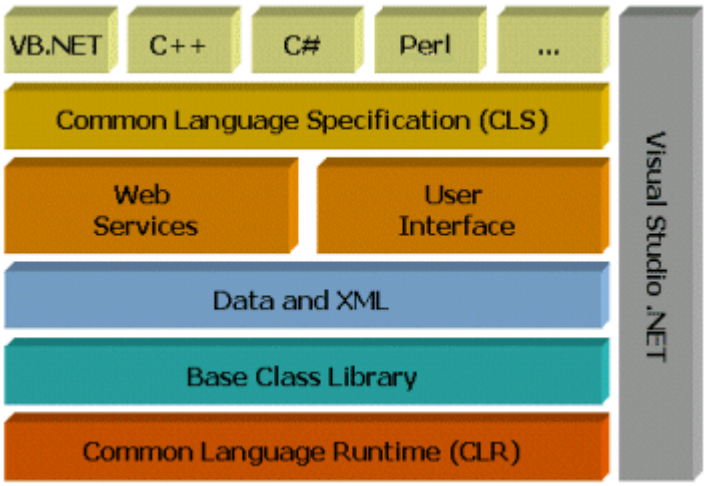
Los requisitos del prototipo implementado son los siguientes:

- Equipo con sistema operativo Windows, con las siguientes instalaciones: .NET Framework, DirectX 9.0 y librerías de TrueVision 3D (para estas últimas se proporciona un instalador).

- Conexión a Internet si se desean emplear los servicios web.

TrueVision3D es un motor gráfico y entorno de desarrollo 3D que emplea las librerías de DirectX. Tanto el Editor como el Visualizador emplean este software. Se puede consultar más información y documentación en http://www.truevision3d.com/.

A continuación se muestra una imagen esquemática de la estructura de la plataforma de Visual Studio .NET. En este proyecto se hará uso intensivo de las capas de XML y Servicios Web.



2.2.3 CONCEPTO DE SERVICIO WEB

Los *Servicios Web* permiten la comunicación programática entre computadores a través de la infraestructura de Internet. Utilizan un mecanismo de llamada remota a funciones residentes en diferentes nodos de la red basado en HTTP y XML. Este mecanismo de comunicación se ajusta a estándares básicos aprobadas por W3C (SOAP, WSDL y UDDI) así como a estándares más complejos demandados por las actuales aplicaciones (WS-Security, WS-Notification, etc.), lo que garantiza la interoperabilidad entre *Servicios Web* soportados sobre diferentes plataformas: J2EE, .NET, etc.

La principal razón para usar servicios Web es que se basan en HTTP sobre TCP en el puerto 80. Muchas empresas se protegen mediante firewalls que filtran y bloquean gran parte del tráfico de Internet. Por ello se cierran casi todos los puertos salvo el 80, porque es el que usan los navegadores. Los servicios Web se realizan por este puerto y ello los hace muy convenientes.

Otra razón es que antes de que existiera SOAP no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red. Las que había eran *ad hoc* y poco conocidas. Una tercera razón por la que los servicios Web son muy prácticos, y que se centra en nuestro caso, es que pueden aportar un débil acoplamiento entre una aplicación que usa el servicio Web y el propio servicio. De esta forma los cambios que cada uno realice con el tiempo no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir las aplicaciones grandes a partir de componentes distribuidos más pequeños es cada día mayor. Esto lo hace muy conveniente en un sistema de simulación como el que nos ocupa, en el que el proceso global puede ser fragmentado en subprocesos independientes entre sí, salvo por las entradas y salidas de cada uno.

2.3 DISEÑO

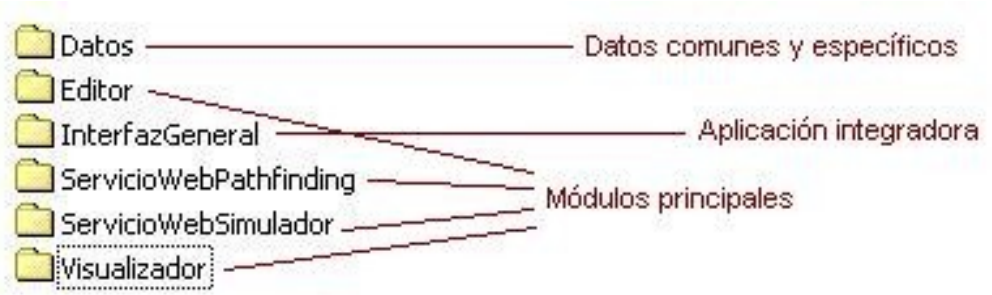
Esta sección comprende el diseño de la aplicación implementada, desde un punto de vista general. Más adelante, en cada apartado se profundizará más en los aspectos estudiados.

El diseño del proyecto ha estado guiado por la necesidad de particionar el proceso principal en módulos más o menos independientes. La separación del sistema en cuatro módulos ya comentada es la consecuencia más evidente. Sin embargo, para el caso del módulo de cálculo de trayectorias y el de simulación, también hay subprocesos claramente diferenciados, y que en un fututo podrían ser fácilmente distribuidos. Cada uno de estos subprocesos, por ejemplo el interpolado de trayectorias o el generado de la dinámica de los aviones, serán comentados en detalle más adelante.

Cada uno de los 4 módulos se corresponde con un proyecto de Visual Studio, siendo sus entradas y salidas de tipo XML, lo que hace que fácilmente puedan operar de manera independiente. Sin embargo, para el caso que nos ocupa, todos se encuentran integrados en una misma ruta.

Así, se dispone de otra aplicación Visual Studio que actúa como el integrador de los módulos, de forma que a través de su interfaz se facilita el acceso a los mismos. Así mismo, para facilitar el uso, todos los datos necesarios para los diversos módulos se encuentran centralizados en una misma ruta compartida. Esta carpeta por lo tanto contiene datos específicos

(cómo las texturas que sólo se usan en el Editor y el Visualizador) y datos comunes (por ejemplo los ficheros XML que son al mismo tiempo salida de un módulo y entrada de otro).



A la hora de ofrecer el uso de los servicios web, se plantearon dos opciones, hacer una nueva interfaz de usuario distinta a la local, o integrar las dos posibilidades en la misma. La segunda opción fue la elegida. El objetivo es permitir al usuario escoger entre lanzar la ejecución de la versión local o la versión de servicio web tanto del cálculo de trayectorias como del cálculo de la simulación.

Consideraciones de rendimiento

Dentro de la implementación, dos procesos han sido identificados como los más problemáticos en cuanto al rendimiento. Por un lado, la ejecución del algoritmo AStar, y por otro, el tiempo en guardar los ficheros XML generados. Más adelante se comentarán las opciones elegidas para intentar minimizar el tiempo de ejecución de ambos.

Resumen

Luego resumiendo, dentro del diseño del proyecto la división inicial, cuyos elementos se estudian en detalle en las siguientes páginas, nos encontramos con:

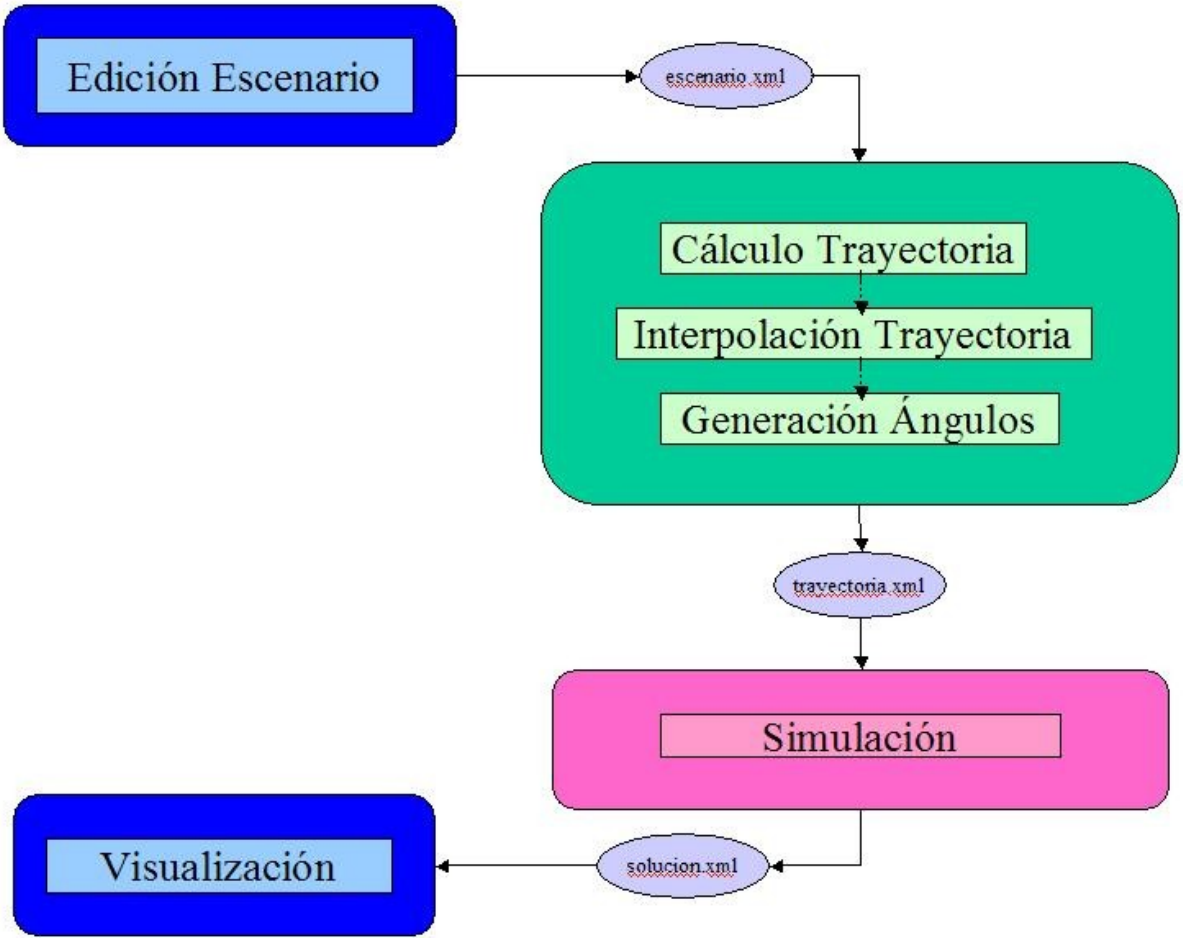
1. Módulo general: implementa el interfaz de usuario para interactuar con los demás elementos de forma sencilla y centralizada.

2. Módulo Editor: aplicación para generar escenarios en 3D.

3. Módulo Pathfinding: aplicación encargada del cálculo de trayectorias a partir de los escenarios generados en el módulo 2.

4. Módulo Simulador: aplicación para el cálculo de la simulación a partir de las trayectorias generadas en el módulo 3 para los escenarios del módulo 2. Como simulación se entiende la generación de ejemplos de simulación de las trayectorias calculadas, de forma que según las probabilidades un avión en ocasiones será derribado por misiles.

5. Módulo Visualizador: aplicación para visualizar en 3D los resultados finales de todo el proceso, es decir, comprende para cada instante de tiempo la trayectoria de los aviones, de los posibles misiles y diversa información junto con eventos o sucesos (lo obtenido en los 3 módulos anteriores).



3 DESARROLLO

3.1 MÓDULO PRINCIPAL (INTERFAZ GENERAL)

3.1.1 DESCRIPCIÓN

El módulo principal simplemente es una aplicación para dar acceso a todas las funciones del proyecto de una manera centralizada. Como se ha comentado, los cuatro módulos diferentes creados están preparados para trabajar de forma independiente, pero gracias a la interfaz común creada, su uso se facilita.

Como ya hemos visto, los 4 módulos son:

1. Editor
2. Cálculo de trayectorias o también denominado "Servicio Web Pathfinding"
3. Cálculo de simulaciones o también "Servicio Web Simulación"
4. Visualizador.

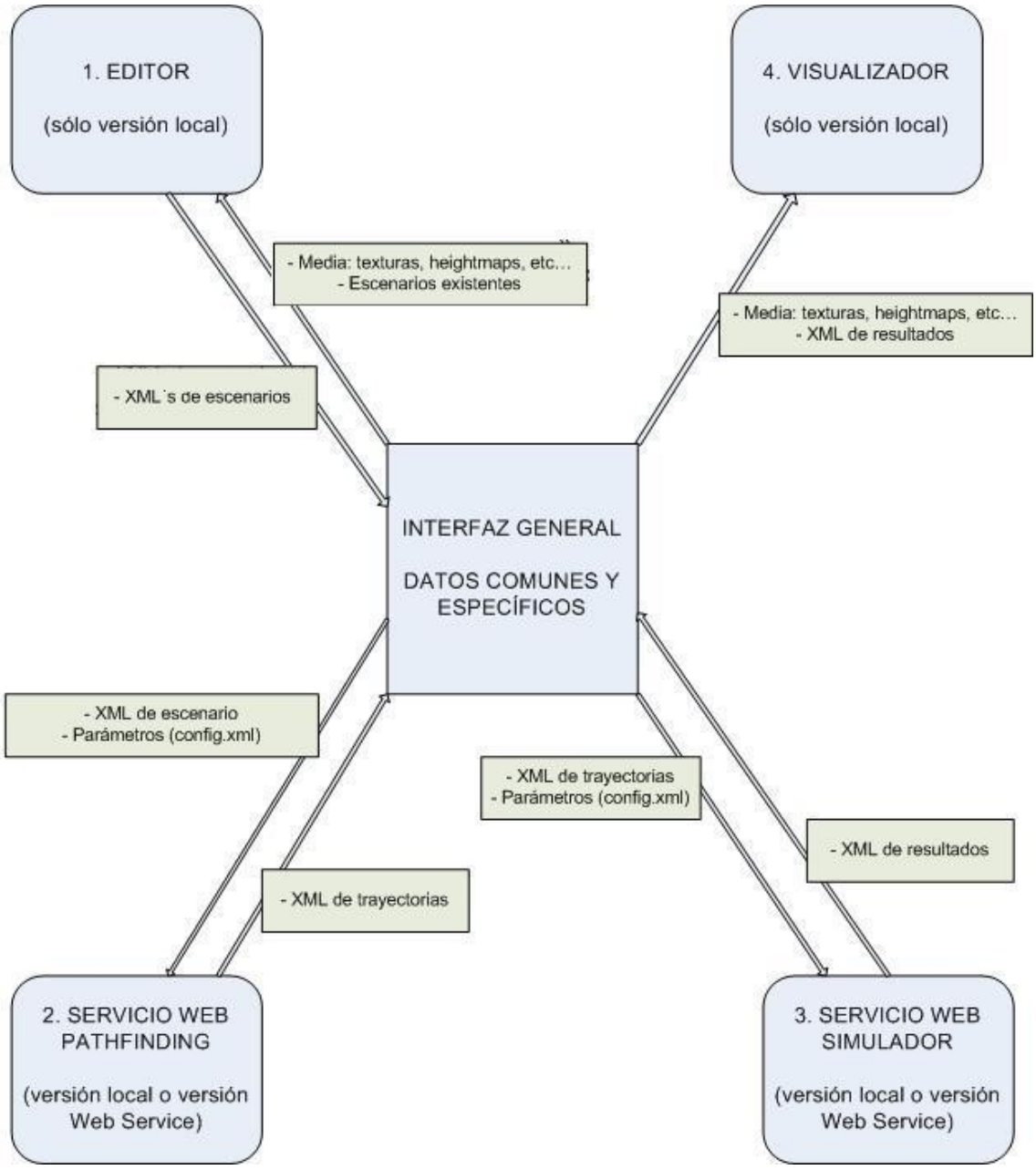
NOTA: se ha decidido nombrar los módulos 2 y 3 como "Servicio Web", porque aunque sea su versión local, se diseñaron para permitir una migración casi inmediata a su versión de servicio web y posteriormente se crearon sus versiones en el servidor.

3.1.2 ARQUITECTURA

Poco es necesario comentar de la arquitectura de este módulo, ya que se trata simplemente de una interfaz de usuario con un único formulario principal.

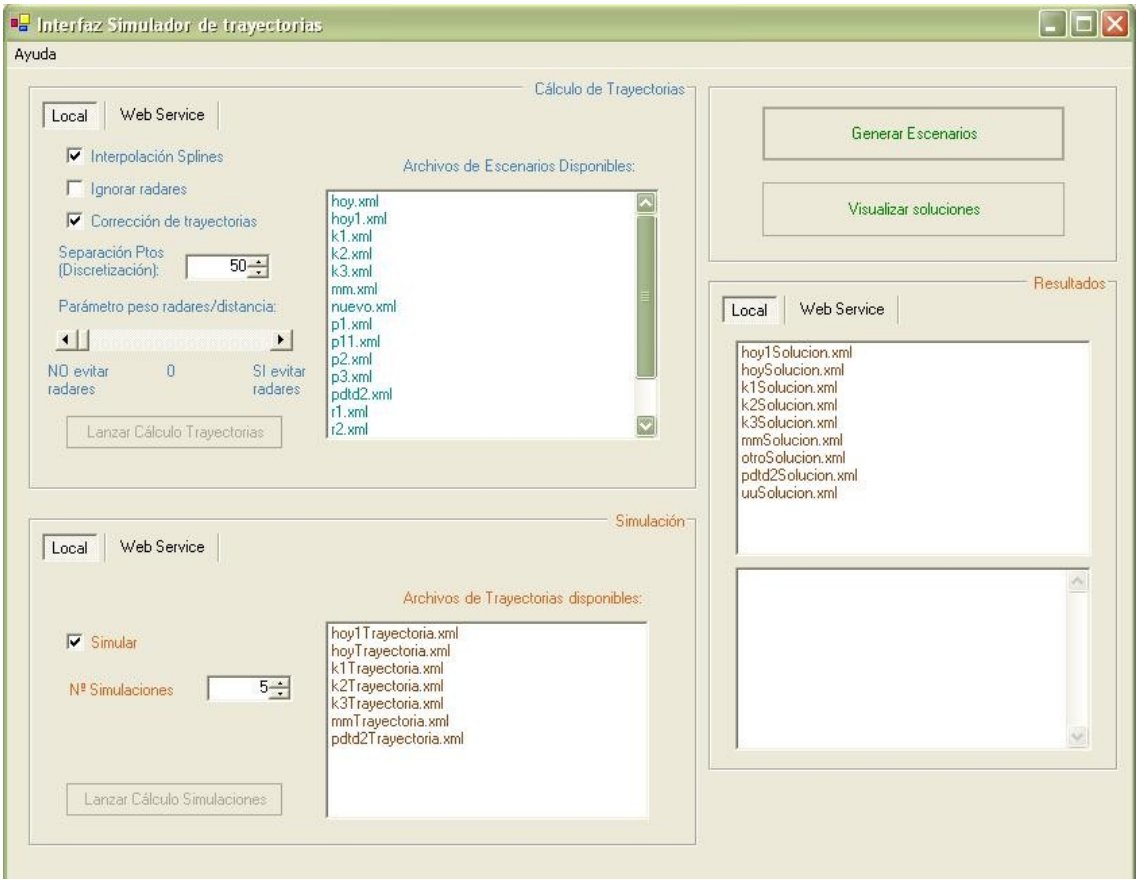
No existe una entrada y salida. De hecho, se puede decir que la entrada y salidas principales del sistema son los ficheros XML de escenario que recibe el Servicio Web Pathfinding y los ficheros XML de resultado que produce el Servicio Web Simulador.

A parte de el interfaz, se ha creado un directorio que engloba todos los datos de los distintos módulos. De esta forma, todos acceden a la misma ruta. El siguiente esquema ilustra los principales ficheros usados por los módulos y los que suben a la ruta de datos tras el procesado.



3.1.3 INTERFAZ GENERAL

El único y formulario principal es el siguiente:



El formulario está dividido en 4 partes para acceder a todas las funciones. Para lanzar el cálculo de trayectorias o el de simulaciones se debe primero seleccionar uno de los ficheros XML de sus respectivas listas, tanto trabajando o en local o seleccionando su lanzamiento en el servidor. En el segundo caso es necesario refrescar manualmente las listas de ficheros.

Los ficheros de resultados se muestran en la lista de la derecha. Seleccionando uno con doble 'click', se muestra un resumen de los mismos en el panel inferior. Para visualizarlos, se debe cargar desde el interfaz del visualizador, que por defecto apuntará a la ruta donde se encuentran: /Datos/Resultados/.

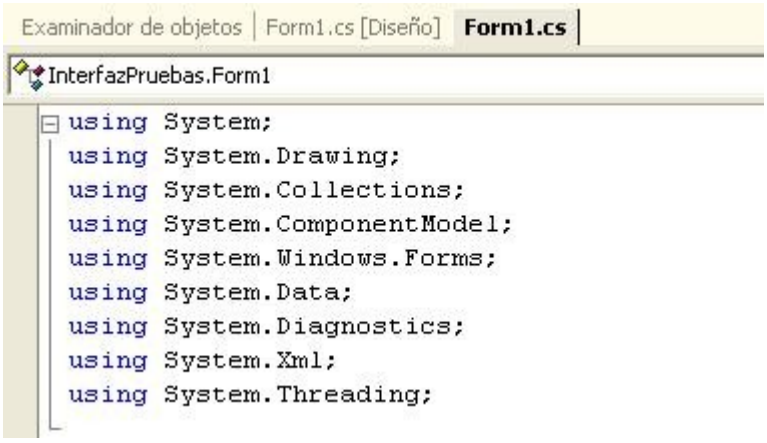
3.1.4 DETALLES DE IMPLEMENTACIÓN: USO DE THREADS

Para cargar los distintos módulos, se emplean subprocesos. Es decir, el módulo principal crea un proceso llamando a los ejecutables de los módulos.

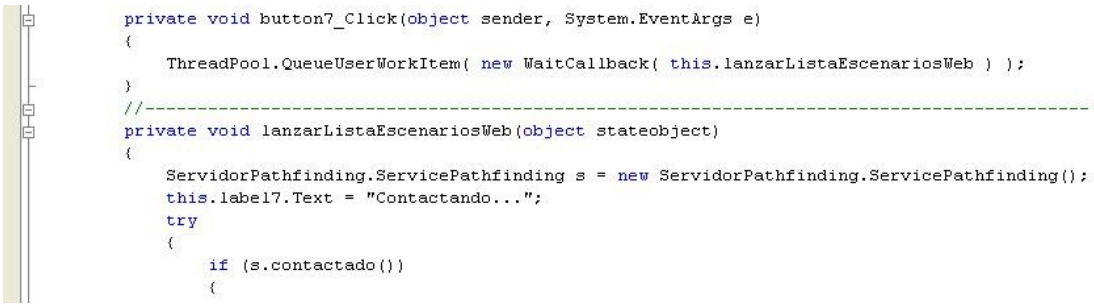
Para evitar que los procesos Editor y Visualizador impidan, por sus características gráficas y consiguiente necesidad de CPU, el trabajar con otras aplicaciones simultaneas, se emplean threads.

Un thread es un proceso que se ejecuta concurrentemente con los demás. Con esto se consigue que la aplicación no se quede esperando a que finalice el proceso que queremos realizar.

Para ello, utilizamos el paquete System.Threading del Visual Studio .NET 2003.



El proceso que queremos ejecutar en un hilo (o thread) debemos encapsularlo en un método aparte y lanzar un thread indicando el método en cuestión como parámetro.



```
private void button7_Click(object sender, System.EventArgs e)
{
    ThreadPool.QueueUserWorkItem( new WaitCallback( this.lanzarListaEscenariosWeb ) );
}
//-----------------------------------------------------------------------------------------
private void lanzarListaEscenariosWeb(object stateobject)
{
    ServidorPathfinding.ServicePathfinding s = new ServidorPathfinding.ServicePathfinding();
    this.label7.Text = "Contactando...";
    try
    {
        if (s.contactado())
        {
```

3.1.5 POSIBLES MEJORAS Y ERRORES CONOCIDOS

- Posibilidad de seleccionar un XML de escenarios o de resultados y que se cargue automáticamente al abrir el Editor o el Visualizador.
- Que se muestre información al usuario del proceso de ejecución cuando se utilizan las versiones de servicio web.

3.2 MÓDULO EDITOR

3.2.1 DESCRIPCIÓN

El módulo de edición se encargará de permitir al usuario de la aplicación editar un escenario para su posterior simulación.

Al ser el primer módulo por el que hay que pasar para realizar una simulación completa, producirá un fichero descriptor de escenario en XML.

Consideraciones

En el espacio tridimensional en el que se edita, se considera el eje *y* como la altura, debido al convenio predefinido del motor gráfico (como se verá en el siguiente apartado).

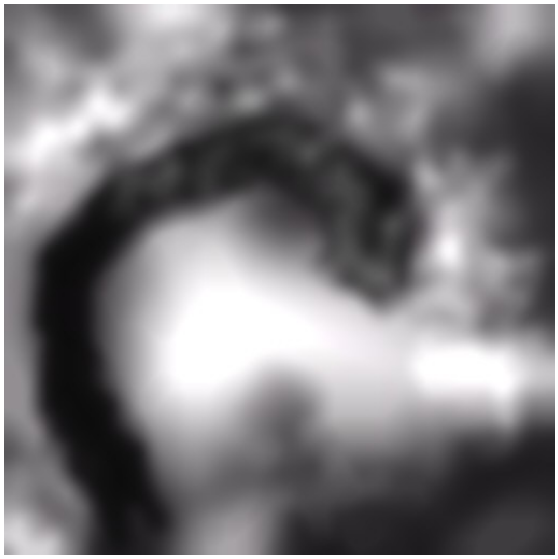
Se podrán ubicar libremente en el escenario:

- Aviones
- Obstáculos
- Un objetivo
- Radares
- Misiles asociados a radares

El terreno se define en base a un *heightmap* (mapa de alturas).

Heightmap

Es un fichero imagen (en formato jpg, bmp, etc.) que, mediante escala de grises, asocia un nivel cromático a una cifra que se corresponderá a la altura describiendo un relieve. En los heightmaps utilizados el blanco es el punto más alto y el negro el más bajo. Pueden crearse manualmente con cualquier aplicación tipo Paint o con algún generador de heightmaps.



Fichero descriptor de salida

Tendrá la siguiente estructura:

```
<Terreno>
   <Anchura> </Anchura>
   <Profundidad> </Profundidad>
   <HeightMap> </HeightMap
</Terreno>
<Aviones>
   <Avion> </Avion>
   ...
</Aviones>
<Radares>
   <Radar> </Radar>
   ...
</Radares>
<Obstaculos>
   <Obstaculo> </Obstaculo>
   ...
</Obstaculos>
<Objetivo> </Objetivo>
```

Cada objeto tiene su propia estructura XML que se describirá más adelante.

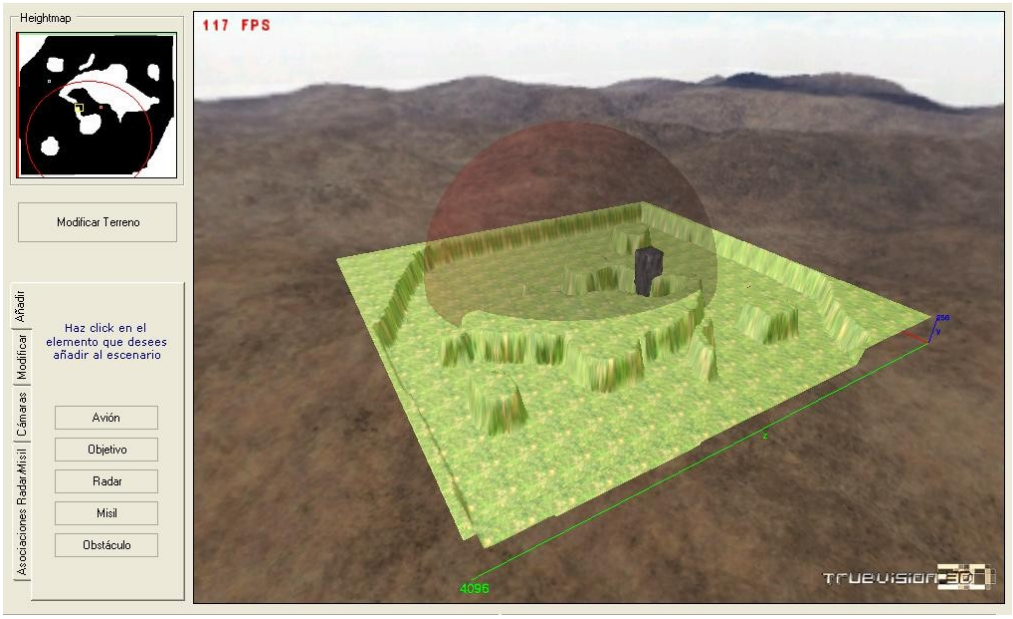
Medidas del terreno

Las dimensiones del terreno que se proporcionarán al editor corresponden al número de "cachos" de terreno.

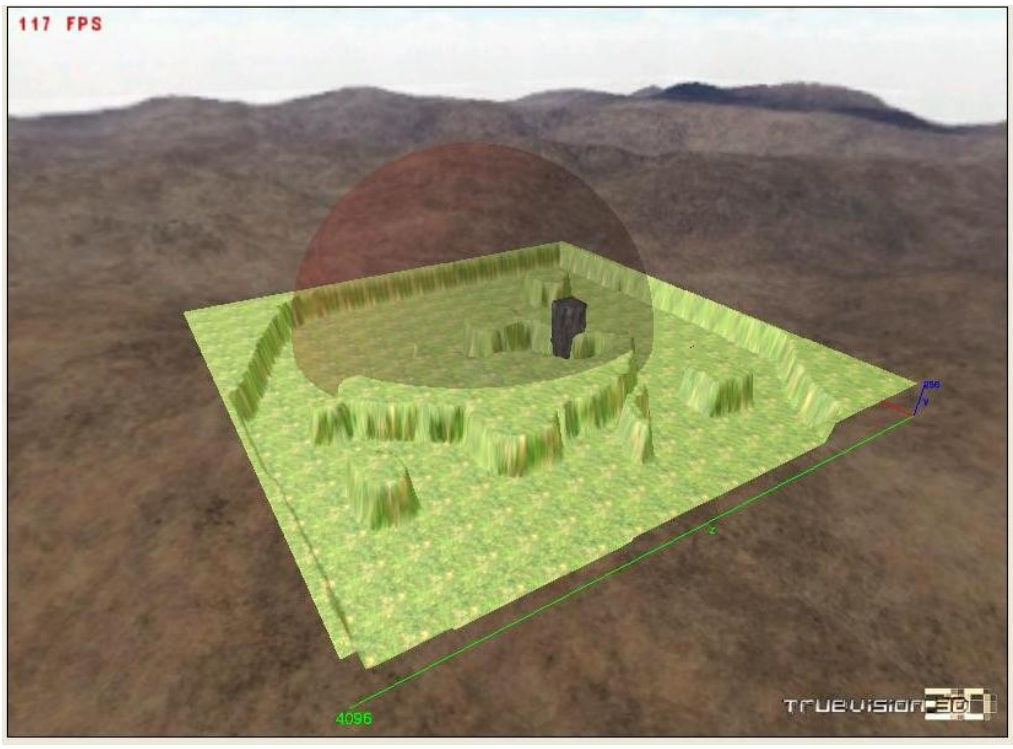
Decimos que un cacho de terreno mide 256x256 píxeles. Esto es consecuencia de la implementación del motor gráfico. Para obtener la dimensión real, basta con multiplicar el número de cachos por 256.

3.2.2 INTERFAZ

Se ha procurado crear una interfaz sencilla y agradable de usar, proporcionando los datos esenciales para la correcta edición de un escenario.



Visor 3D



Es un objeto PictureBox del paquete System.Forms del entorno .NET. Es el encargado de mostrar el estado tridimensional del terreno en tiempo real.

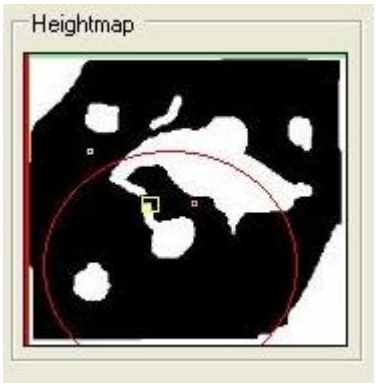
Esto es, se mostrará cualquier objeto que se añada, modifique o elimine en ese momento.

En la esquina superior izquierda se muestran las imágenes por segundo, que servirán para evaluar la fluidez de la representación. En la esquina inferior derecha se muestra la marca de agua del motor gráfico que se está utilizando.

Se pueden observar los tres ejes de coordenadas pegados al objeto terreno, que servirá de guía. El eje z (verde), eje x (rojo) y eje y (azul) muestran en su extremo los píxeles de largo, ancho y altura que tiene el escenario.

Al editar no se permite que se sobrepasen estos límites.

Visor Heightmap



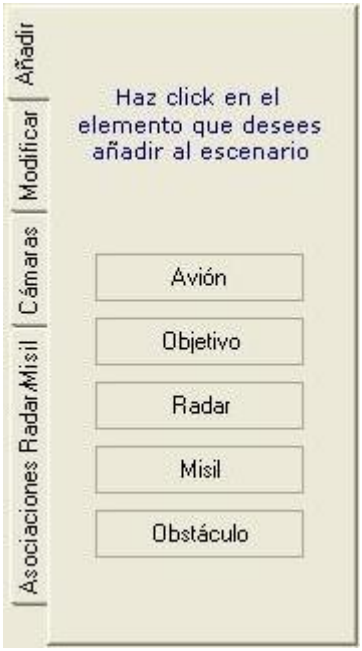
Se muestra la correspondencia del escenario 3D con una perspectiva bidimensional vista desde arriba con el heightmap correspondiente de fondo.
Leyenda:

- Lateral rojo: indicador eje x
- Cima verde: indicador eje z
- Punto amarillo: posición de la cámara
- Punto azúl: posición de un avión
- Punto salmón: posición del objetivo
- Circunferencia roja: radio de acción de un radar
- Punto verde: posición de un misil
- Cuadrado amarillo: obstáculo

Añadir elemento

Se podrá seleccionar qué elemento se quiere añadir al escenario.

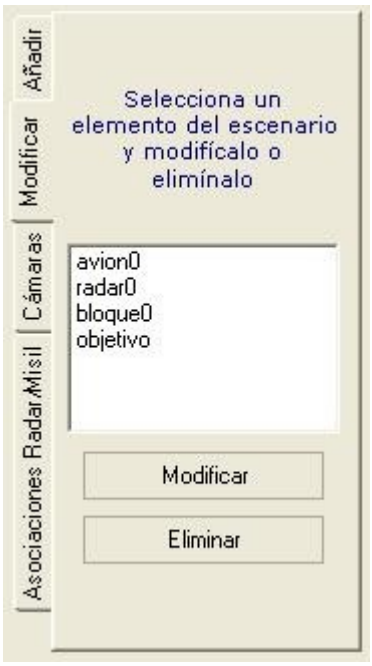
En el manual de usuario se describe cada uno con detalle.



Modificar/Eliminar elemento

Se podrá seleccionar qué elemento del escenario se quiere modificar o eliminar.

En el manual de usuario se describe cada uno con detalle.



Cámaras

Se podrán seleccionar entre dos cámaras predefinidas.

En el manual de usuario se describe su uso con detalle.

Asociaciones Radar-Misil

Se muestran las asociaciones establecidas entre radares y misiles.

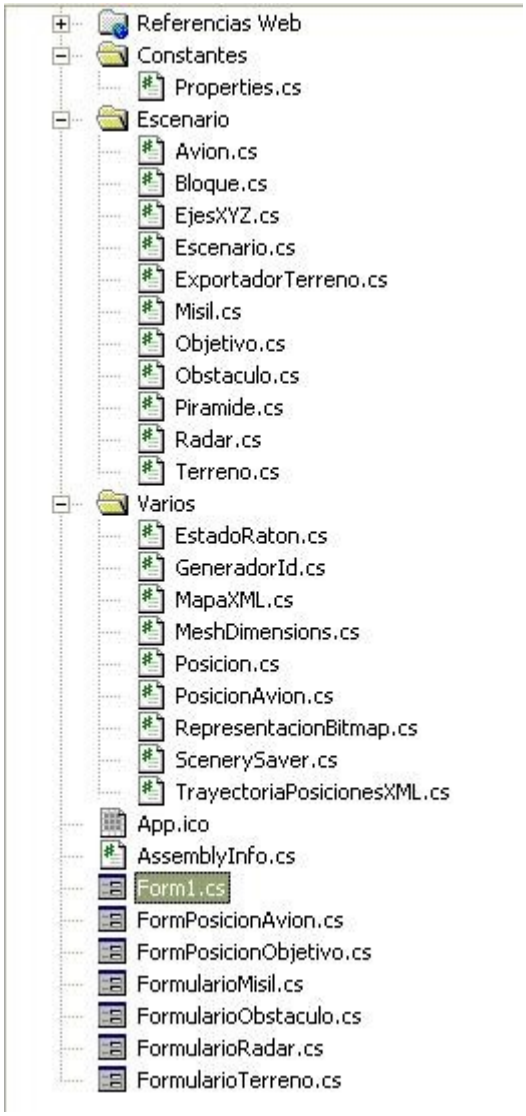
Modificar Terreno

El botón te permite cambiar de heightmap y/o de dimensiones del terreno. Se explica con detalle en el manual de usuario.

Subir terreno al servidor

En el menú Archivo existe una opción para subir un escenario guardado al servidor. Esta es condición indispensable para poder simular un escenario por medio de los servicios web.

3.2.3 CLASES



Namespace Constantes

Properties.cs:
- Define propiedades estáticas que se utilizarán de manera general.

Namespace Escenario

Avion.cs:
- Representa un avión con sus parámetros de posición, escala y mesh (figura 3D) para representar.

Bloque.cs:
- Hereda de la clase Obstáculo. Es un obstáculo de geometría paralelepípeda. Contiene atributos de posición, escala y mesh.

EjesXYZ.cs:
- Proporciona un método estático para representar los ejes de coordenadas en la escena tridimensional.

Escenario.cs:
- Representa el escenario.
- Contiene referencias a todos los elementos del mismo, incluido el terreno, variables globales del motor 3D, etc.
- Proporciona métodos necesarios para retraer información requerida de los mismos.

ExportadorTerreno.cs:
- Permite exportar un terreno discretizado a un fichero de texto.
- Ya no se utiliza.

Misil.cs:
- Representa un misil con sus parámetros de posición y mesh para representarlo.
- El parámetro alcance se utilizará en los módulos posteriores pero se configura en el editor.

Objetivo.cs:
- Representa el objetivo con sus parámetros de posición, escala y mesh para representar.

Obstaculo.cs:
- Interfaz que define a cualquier obstáculo que herede de él (en este caso se utilizan pirámides y bloques). Permitiría añadir más tipos de obstáculos fácilmente.

Piramide.cs:
- Hereda de la clase Obstáculo. Es un obstáculo de geometría piramidal. Contiene atributos de posición, escala y mesh.

Radar.cs:
- Representa un radar con sus atributos de posición y mesh.
- Dispone de un mesh adicional: una esfera que representará su radio de acción.
- El atributo alcance determina el radio de alcance máximo del radar.
- El atributo dmindet determina el radio de mínima detección del radar, que será donde la probabilidad de que un avión sea detectado sea del 100%.
- El atributo timeout determina el tiempo máximo de seguimiento de un avión de manera continuada.

Terreno.cs:
- Representa el terreno.
- Está ligado al objeto TVLandscape del motor gráfico, el cual permite generarlo a partir de un heightmap con suma facilidad.
- Contiene atributos de dimensiones.
- El atributo separacionPuntos indica la separación entre punto y punto correspondiente a la discretización del terreno (explicado en el módulo pathfinding, donde se utilizará).

Namespace Varios

EstadoRaton.cs:
- Representa el estado del ratón en cada instante.
- Ligado al objeto TVInputEngine del motor gráfico.
- Contiene un método para actualizar los valores de posición, botones y scroll.

GeneradorId.cs:
- Permite generar id's para los objetos de un escenario sin que haya repeticiones, puesto que han de ser únicos para asegurar la correcta funcionalidad de la aplicación y eliminar ambigüedades.

MapaXML.cs:
- Representa el terreno discretizado en xml a modo de corte transversal.
- Se utilizará en el módulo de pathfinding.
- Ya no se utiliza en este módulo.

MeshDimensions.cs:

- Permite obtener las dimensiones de un mesh en concreto a partir de su bounding box.

Posicion.cs:
- Representa una posición x,y,z.

PosicionAvion.cs:
- Representa una posición de un avión (x, y, z).
- Además contiene información de los tres ángulos de rotación de un avión (roll, pitch, yaw).

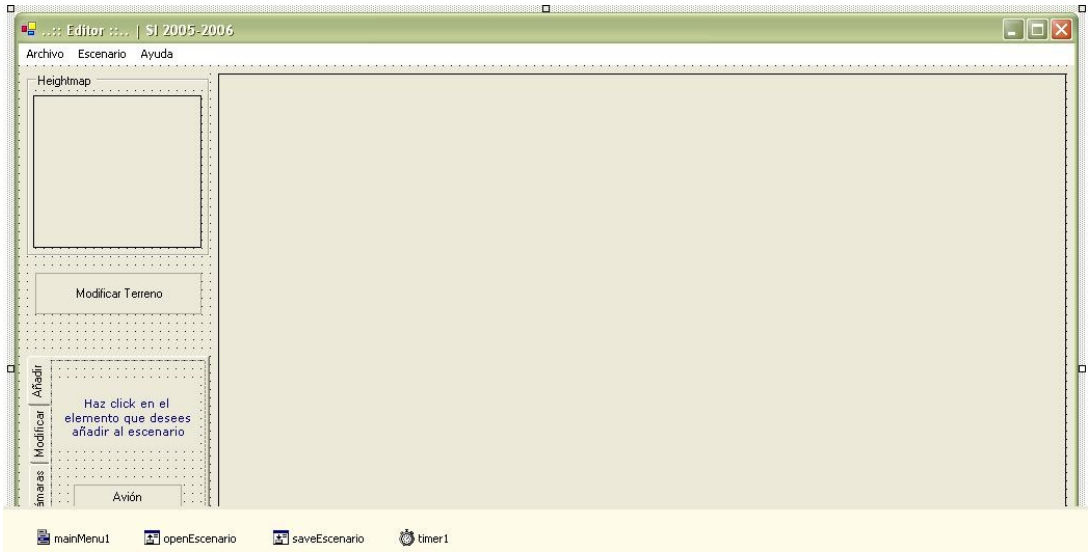
RepresentacionBitmap.cs:
- En un objeto PictureBox dado, se dibuja el estado actual del escenario.
- Se utiliza para refrescar el visor heightmap.

SceneSaver.cs:
- Clase encargada de guardar el escenario en un fichero.
- También ofrece funcionalidad para cargar un escenario.

TrayectoriaPosicionesXML.cs:
- Representará una trayectoria completa en formato XML.
- Sobre cada posición contendrá información sobre posición y estado del avión (inclinaciones y orientación).
- Puede obtenerse en forma de ArrayList y XmlDocument.
- Ya no se utiliza en este módulo por cuestiones de rediseño.

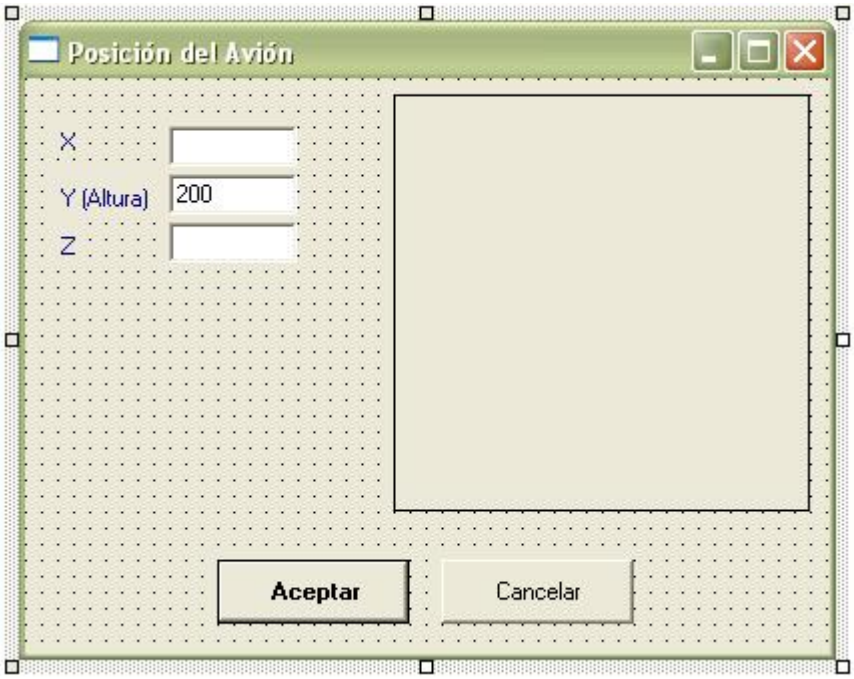
Formularios

Form1.cs:
- Formulario principal (la interfaz).



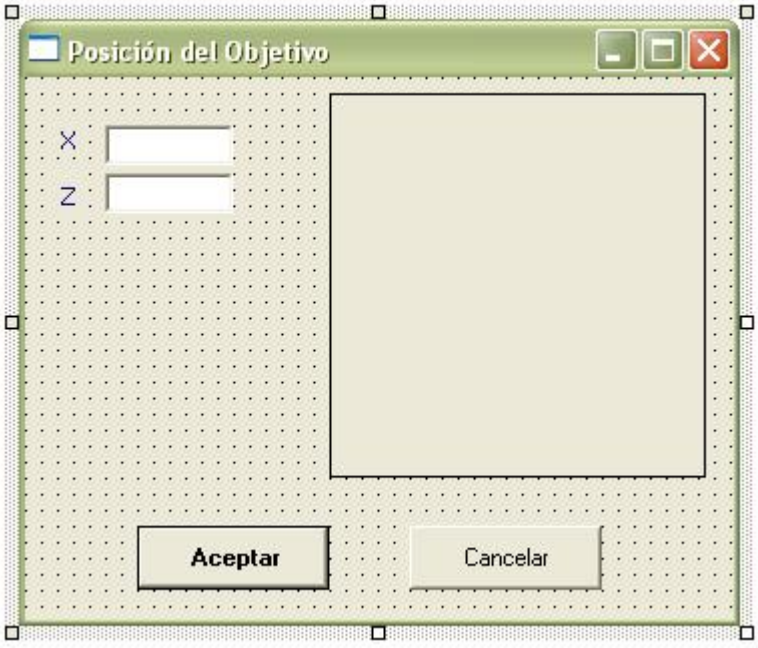
FormPosicionAvion.cs:

- Formulario que permite posicionar un avión en el escenario.



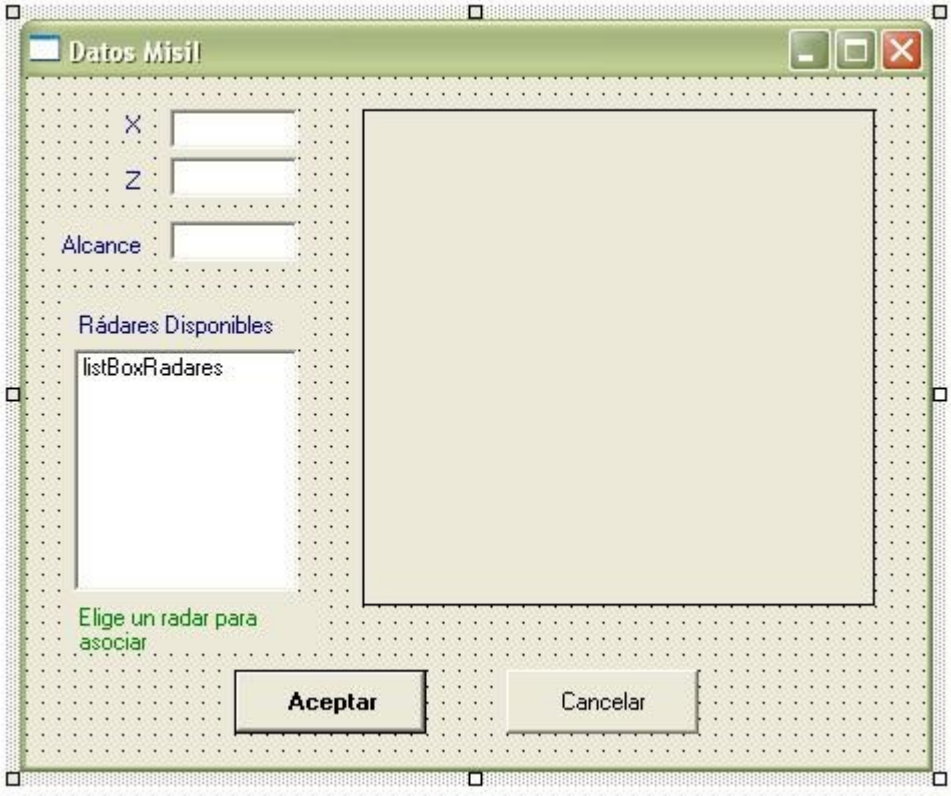
FormPosicionObjetivo.cs:

- Formulario que permite posicionar el objetivo en el escenario.



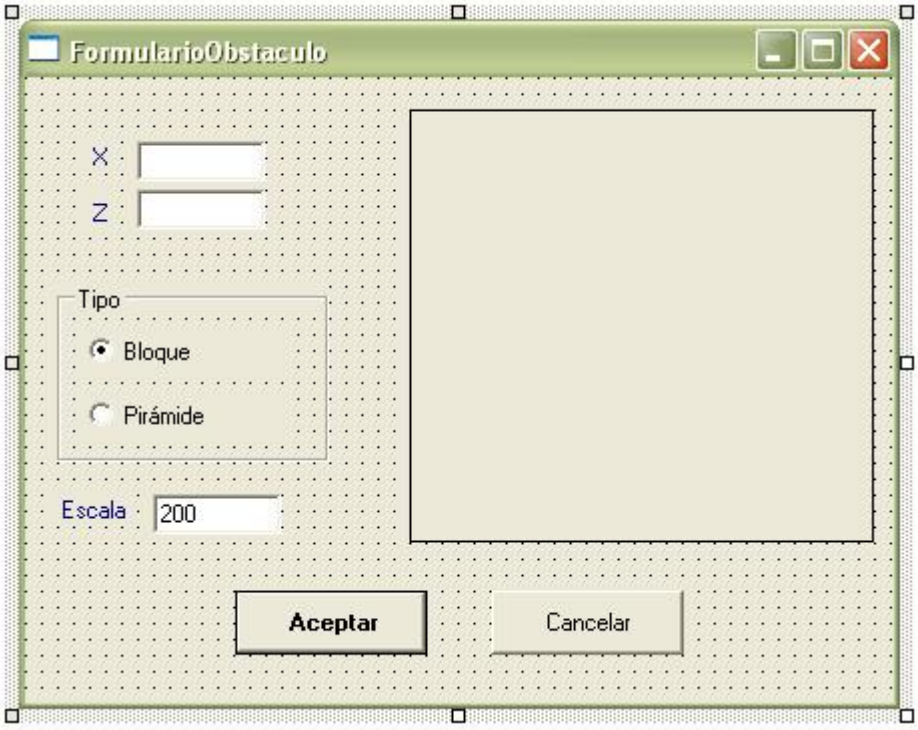
FormularioMisil.cs:

- Formulario que permite posicionar un misil en el escenario y asociarlo a un radar.



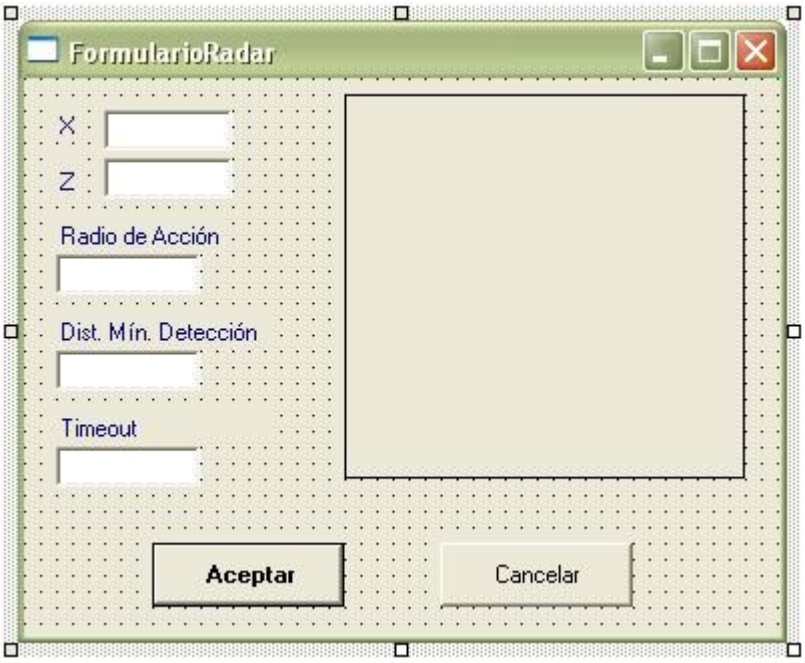
FormularioObstaculo.cs:

- Formulario que permite seleccionar un tipo de obstáculo y posicionarlo en el escenario.



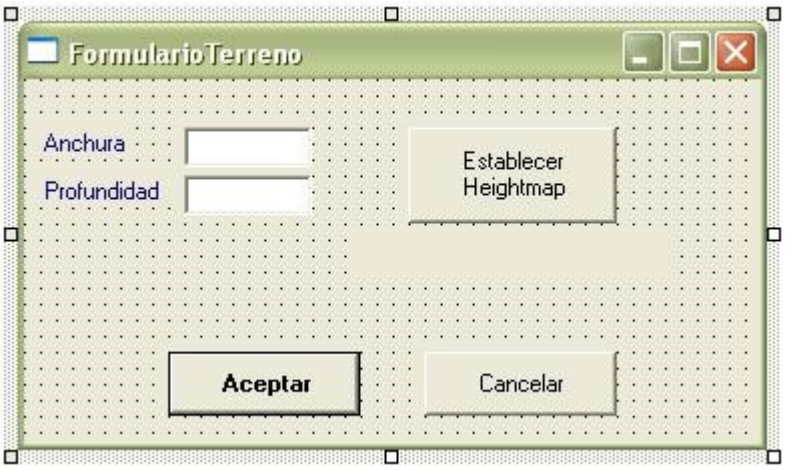
FormularioRadar.cs:

- Formulario que permite configurar las características de un radar y colocarlo en el escenario.

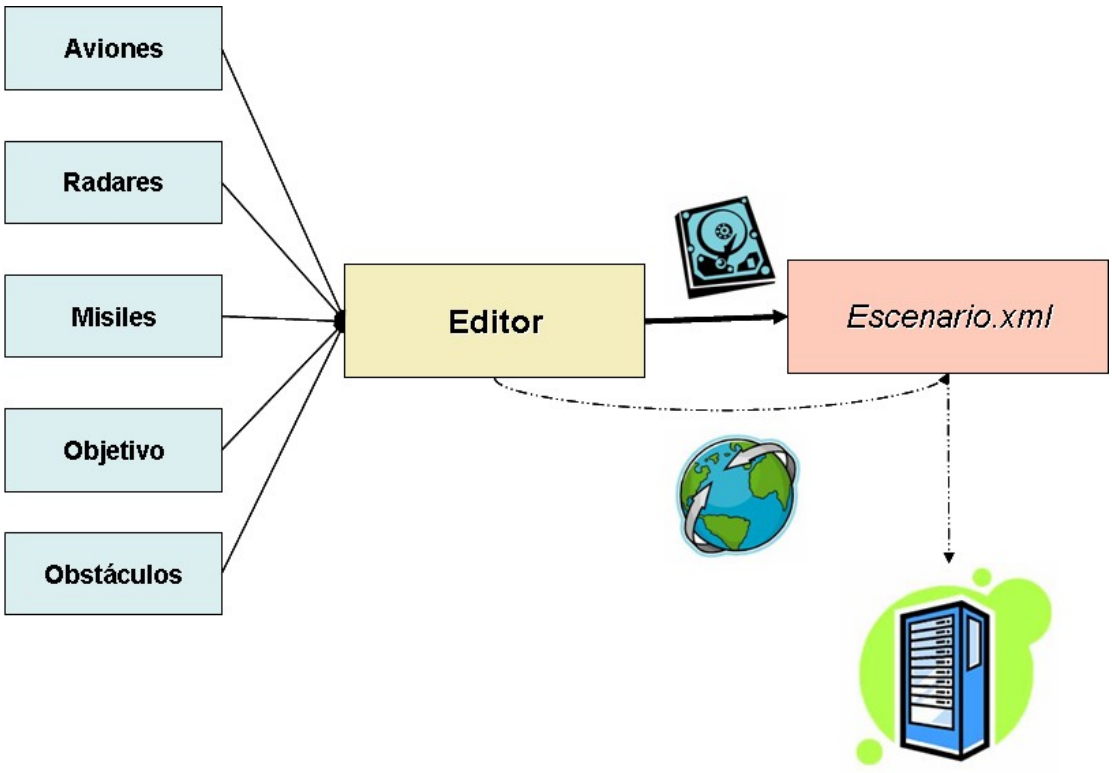


FormularioTerreno.cs:

- Formulario que permite cambiar las dimensiones del terreno y seleccionar un heightmap que configurará su relieve.



3.2.4 DIAGRAMA



3.2.5 DETALLES DE IMPLEMENTACIÓN

Las operaciones de inserción que se llevan a cabo están ligadas al funcionamiento del motor gráfico. En cualquier caso, se ha creado una clase por cada tipo de elemento (Avion, Radar, Misil, etc.) y un objeto Escenario que engloba todos los elementos, terreno, y atributos necesarios.

El proceso de conexión vía servicio web para subir un escenario al servidor se describe en un anexo de esta documentación.

El dibujo sobre el objeto PictureBox se implementa mediante las funcionalidades que proporciona el módulo GDI+ de Microsoft. Ver anexo.

La creación de modelos 3D se detalla también en el anexo correspondiente.

3.2.6 POSIBLES MEJORAS Y ERRORES CONOCIDOS

- Al añadir un elemento, no se actualiza la imagen donde se selecciona la posición hasta que se pincha una vez sobre ella.
- La inserción de elementos en escenario mediante el visor 3D sería una posible mejora. Incrementaría la intuitividad de la aplicación.
- Añadido de más tipos de obstáculos.

3.3 MÓDULO VISUALIZADOR

3.3.1 DESCRIPCIÓN

El módulo de visualización se encargará de permitir al usuario visualizar la secuencia de una simulación, además de proporcionar datos relevantes acerca de la misma.

Es el último módulo en la secuencia del proceso de simulación, así que leerá los ficheros de salida del módulo anterior. Estos serán los ficheros de resultado.

3.3.2 CONSIDERACIONES

Fichero de entrada

El fichero xml que es capaz de abrir este módulo describe lo siguiente:

- Dimensiones del terreno.
- Heightmap que define el relieve del terreno.
- Objetos que intervienen en el escenario.
- Nombre de este fichero.
- Información del éxito de la simulación.
- Trayectorias descritas por los aviones y misiles que intervienen.
- Eventos ocurridos durante la simulación.
- Probabilidades de detección de cada radar a cada avión en cada instante.

Tiene la siguiente estructura:

```
<Datos>
   <Terreno>
      <Anchura> </Anchura>
      <Profundidad> </Profundidad>
      <HeightMap> </HeightMap>
   </Terreno>
   <Aviones>
      <Avion> </Avion>
      …
   </Aviones>
   <Radares>
      <Radar> </Radar>
      …
   </Radares>
   <Misiles>
      <Misil> </Misil>
      …
   </Misiles>
```

```
   <Obstáculos>
      <Obstáculo> </Obstáculo>
      …
   </Obstáculos>
   <Objetivo> </Objetivo>
</Datos>
<Info>
   <NombreFichero> </NombreFichero>
      <PorcentajeLlegadas> </PorcentajeLlegadas>
</Info>
<Solucion>
   <TrayectoriasAviones>
                  <TrayectoriaAvion>
                        <Id>avion0</Id>
                        <TiempoPosicion>
                              <Instante> </Instante>
                              <Posicion> </Posicion>
                  …
                        </TiempoPosicion>
            …
      </TrayectoriaAvion>
      …
   </TrayectoriasAviones>
   <Eventos>
                  <Evento>
                        <Instante> </Instante>
                        <Codigo> </Codigo>
                        <IdAvion> </IdAvion>
                        <IdMisil> </IdMisil>
                        <IdRadar> </IdRadar>
                  </Evento>
      …
   </Eventos>
   <Detecciones>
                  <Deteccion>
                         <InstanteTiempo> </InstanteTiempo>
                        <IdRadar> </IdRadar>
                        <IdAvion> </IdAvion>
                        <PPD> </PPD>
                  </Deteccion>
      …
   </Detecciones>
</Solucion>
```

Reproducción

El visualizador es capaz de generar un entorno 3D idéntico al generado en el editor al configurar el escenario para el fichero dado. A partir de la descripción de trayectorias dadas para cada objeto, es capaz de reproducir la secuencia de manera continua, paso a paso hacia delante, paso a paso hacia atrás y reestablecer el comienzo de la secuencia.

Información de la simulación

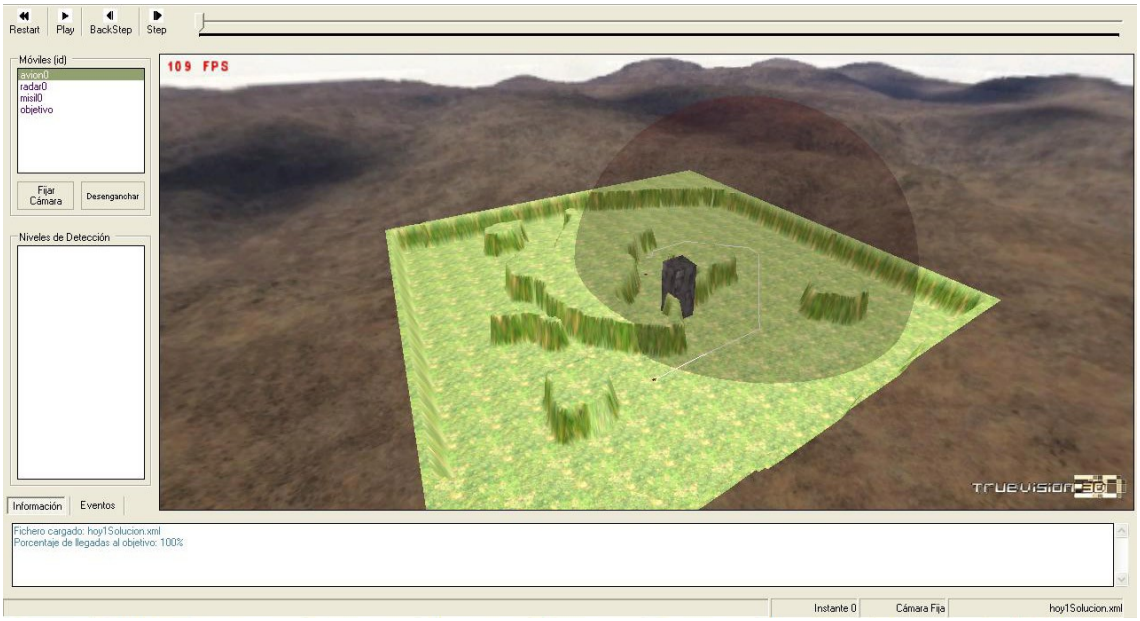
En cada paso de la reproducción se informará al usuario de eventos relevantes de la simulación:

- Detección de un radar
- Seguimiento de un radar
- Disparo de un misil
- Seguimiento de un misil
- Choques de misiles o aviones
- Llegada al objetivo
- Etc.

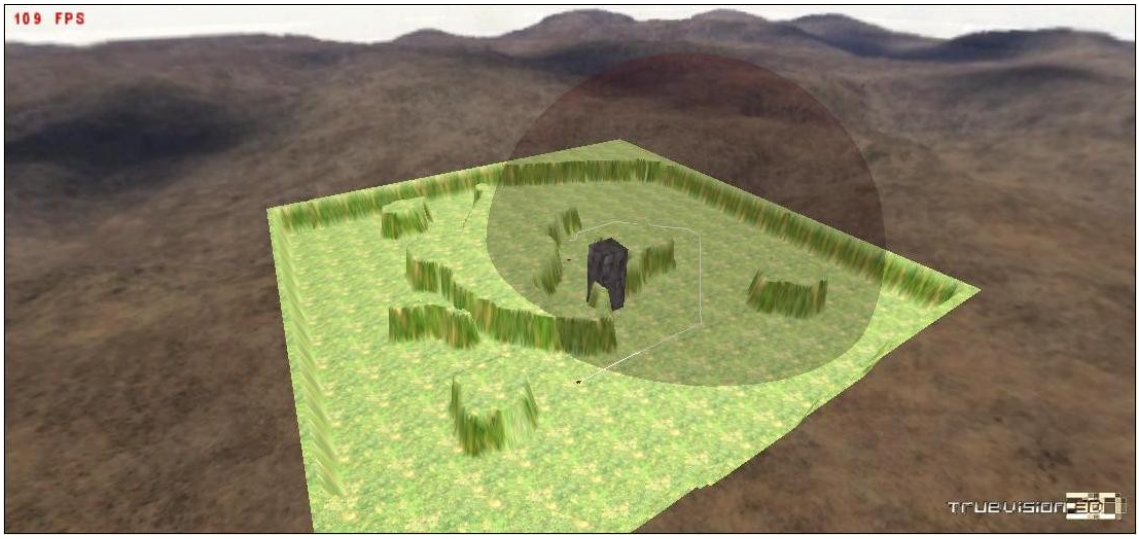
También se informará en cada momento de la probabilidad de detección de cada radar a cada avión.

3.3.3 INTERFAZ

Se ha procurado crear una interfaz sencilla y agradable de usar para la correcta visualización.



Visor 3D



Es un objeto PictureBox del paquete System.Forms del entorno .NET. Es el encargado de mostrar el estado tridimensional de la escena en tiempo real.

En la esquina superior izquierda se muestran las imágenes por segundo, que servirán para evaluar la fluidez de la representación. En la esquina inferior derecha se muestra la marca de agua del motor gráfico que se está utilizando.

Se puede observar una línea blanca que representa la trayectoria descrita por el avión de la captura de pantalla. Para las trayectorias de misiles se utilizan líneas de color rojo.

Móviles (id)

Se muestran los identificadores de diversos elementos relevantes del escenario. El botón Fijar Cámara ajusta la cámara detrás del objeto seleccionado, y el botón Desenganchar "suelta" la cámara de detrás de un avión y la deja fija en el escenario manteniendo el foco en el avión. Se detalla en el manual de usuario.

Niveles de Detección

Informa de la probabilidad de detección de un radar respecto de un avión en un instante determinado. Se actualiza dinámicamente durante la visualización.

Información

Informa del fichero cargado y el porcentaje de llegadas al objetivo.

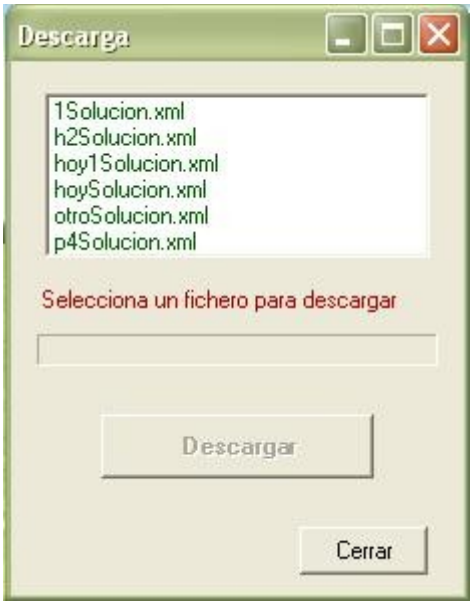
Eventos

Informa de eventos de relevancia acontecidos durante la simulación. Se actualiza la lista para cada instante de tiempo (entre paréntesis).

Controles de Reproducción

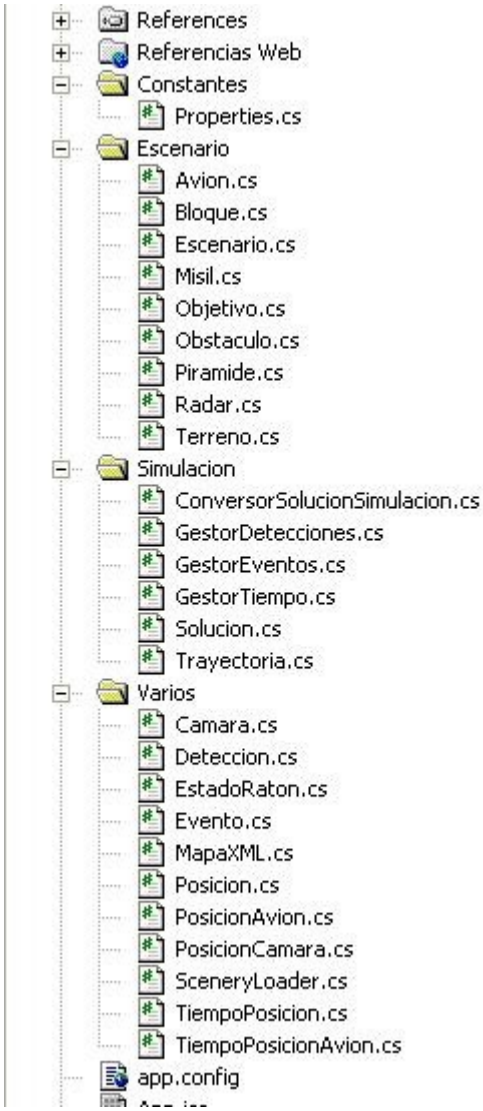
Permite reproducción continua, paso a paso hacia delante y hacia atrás, rebobinado al comienzo y desplazamiento manual de la secuencia.

Descarga de resultados



En el menú Archivo existe la opción de descargar ficheros de solución almacenados en el servidor.

3.3.4 CLASES



Namespace Constantes

Properties.cs:

- Define propiedades estáticas que se utilizarán de manera general.

Namespace Escenario

Avion.cs:

- Representa un avión con sus parámetros de posición, escala y mesh (figura 3D) para representar.

Bloque.cs:

- Hereda de la clase Obstáculo. Es un obstáculo de geometría paralelepípeda. Contiene atributos de posición, escala y mesh.

Escenario.cs:
- Representa el escenario.
- Contiene referencias a todos los elementos del mismo, incluido el terreno, variables globales del motor 3D, etc.
- Proporciona métodos necesarios para retraer información requerida de los mismos.

Misil.cs:
- Representa un misil con sus parámetros de posición y mesh para representarlo.

Objetivo.cs:
- Representa el objetivo con sus parámetros de posición, escala y mesh para representar.

Obstaculo.cs:
- Interfaz que define a cualquier obstáculo que herede de él (en este caso se utilizan pirámides y bloques). Permitiría añadir más tipos de obstáculos fácilmente.

Piramide.cs:
- Hereda de la clase Obstáculo. Es un obstáculo de geometría piramidal. Contiene atributos de posición, escala y mesh.

Radar.cs:
- Representa un radar con sus atributos de posición y mesh.
- Dispone de un mesh adicional: una esfera que representará su radio de acción.

Terreno.cs:
- Representa el terreno.
- Está ligado al objeto TVLandscape del motor gráfico, el cual permite generarlo a partir de un heightmap con suma facilidad.
- Contiene atributos de dimensiones.

Namespace Simulación

ConversorSolucionSimulacion.cs:
- Convierte las soluciones (trayectorias de aviones y misiles) de ArrayList a XML y viceversa.
- Su propósito es facilitar la tarea a la hora de cargar un XML de este tipo o guardarlo.
- Es más fácil trabajar sobre ArrayLists.

GestorDetecciones.cs:
- Proporciona información acerca de los grados de detección de los radares.

GestosEventos.cs:
- Proporciona información sobre los eventos ocurridos en una simulación en función del instante.

GestorTiempo.cs:
- Modifica cambios en las posiciones de los móviles del escenario según la solución obtenida del escenario cargado.
- Se utilizará para facilitar y simplificar la reproducción de la secuencia.

Solucion.cs:
- En este objeto se carga la solución de la simulación dada.
- Contiene trayectorias, eventos y niveles de detección, así como información adicional.

Trayectoria.cs:
- Dibuja las trayectorias en pantalla mediante el objeto TVScreen2DImmediate.

Namespace Varios

Camara.cs:
- Ligado al objeto TVCamera del motor gráfico.
- Almacena información de posición y rotación de la cámara.

Deteccion.cs:
- Representa una detección ocurrida en un instante dado de un radar a un avión.

EstadoRaton.cs:
- Representa el estado del ratón en cada instante.
- Ligado al objeto TVInputEngine del motor gráfico.
- Contiene un método para actualizar los valores de posición, botones y scroll.

Evento.cs:
- Representa un evento ocurrido durante la simulación en un instante determinado.

MapaXML.cs:

- Representa el terreno discretizado en xml a modo de corte transversal.
- Se utilizará en el módulo de pathfinding.
- Ya no se utiliza en este módulo.

Posicion.cs:
- Representa una posición x,y,z.

PosicionAvion.cs:
- Representa una posición de un avión (x, y, z).
- Además contiene información de los tres ángulos de rotación de un avión (roll, pitch, yaw).

PosicionCamara.cs:
- Representa una posición de la cámara (x, y, z).
- Además, contiene información de los tres ángulos de rotación (roll, pitch, yaw).
- También hace constar las coordenadas donde mantiene fijo el foco (x, y, z).

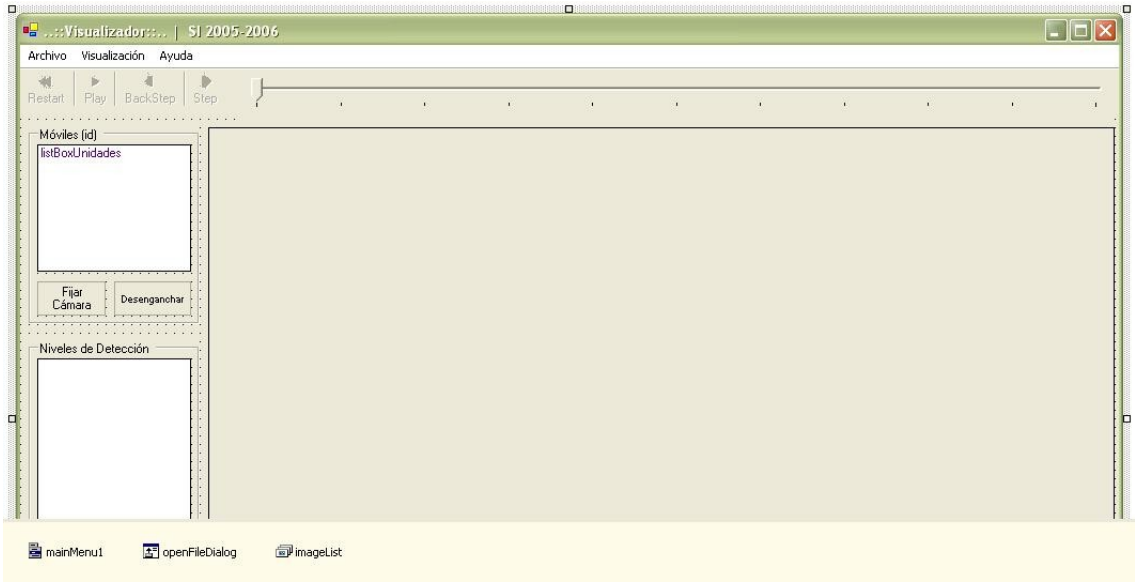
SceneryLoader.cs:
- Carga el escenario a partir del fichero dado.

TiempoPosicion.cs:
- Asocia una posición con un instante de tiempo.

TiempoPosicionAvion.cs:
- Asocia una posición de un avión con un instante de tiempo.

Formularios

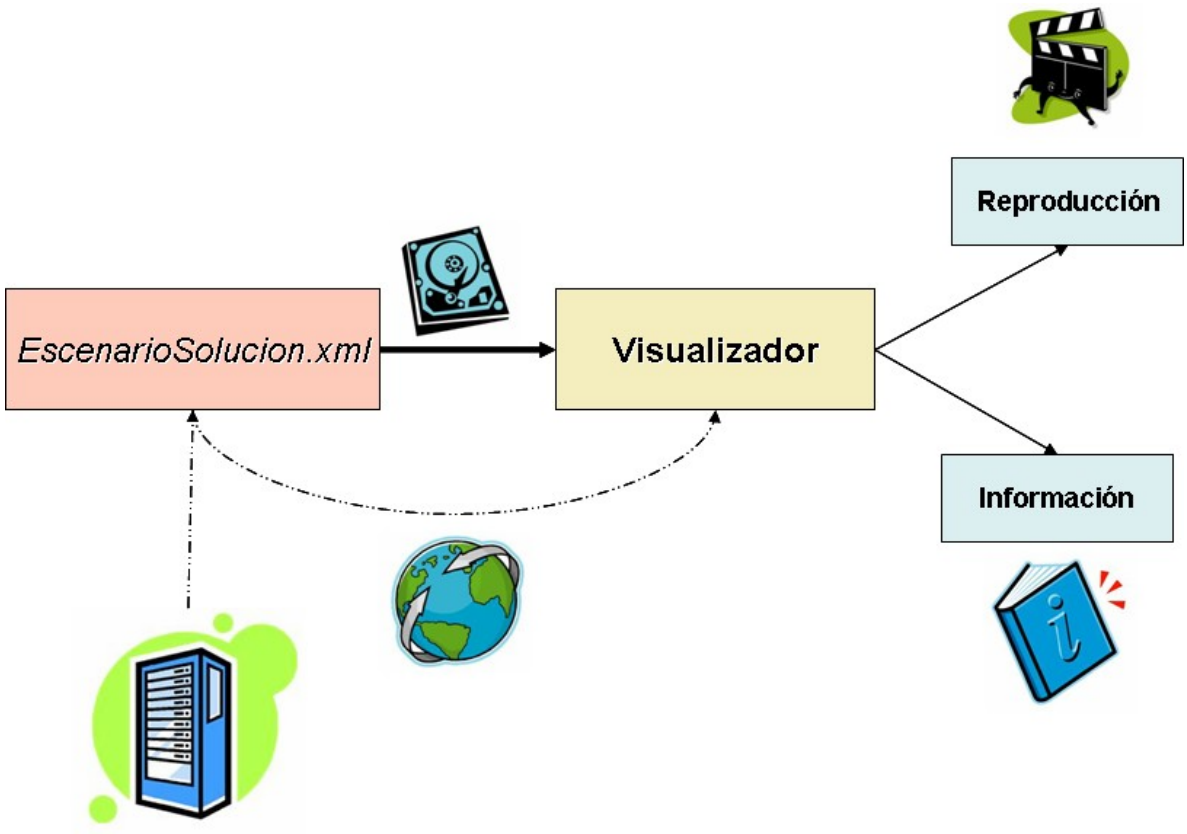
Form1.cs:
- Formulario principal (la interfaz).



FormDescarga.cs:

- Formulario que permite descargar un xml descriptor de solución de una simulación.

3.3.5 DIAGRAMA



3.3.6 DETALLES DE IMPLEMENTACIÓN

Caben reseñar dos procesos que son la base de este módulo:

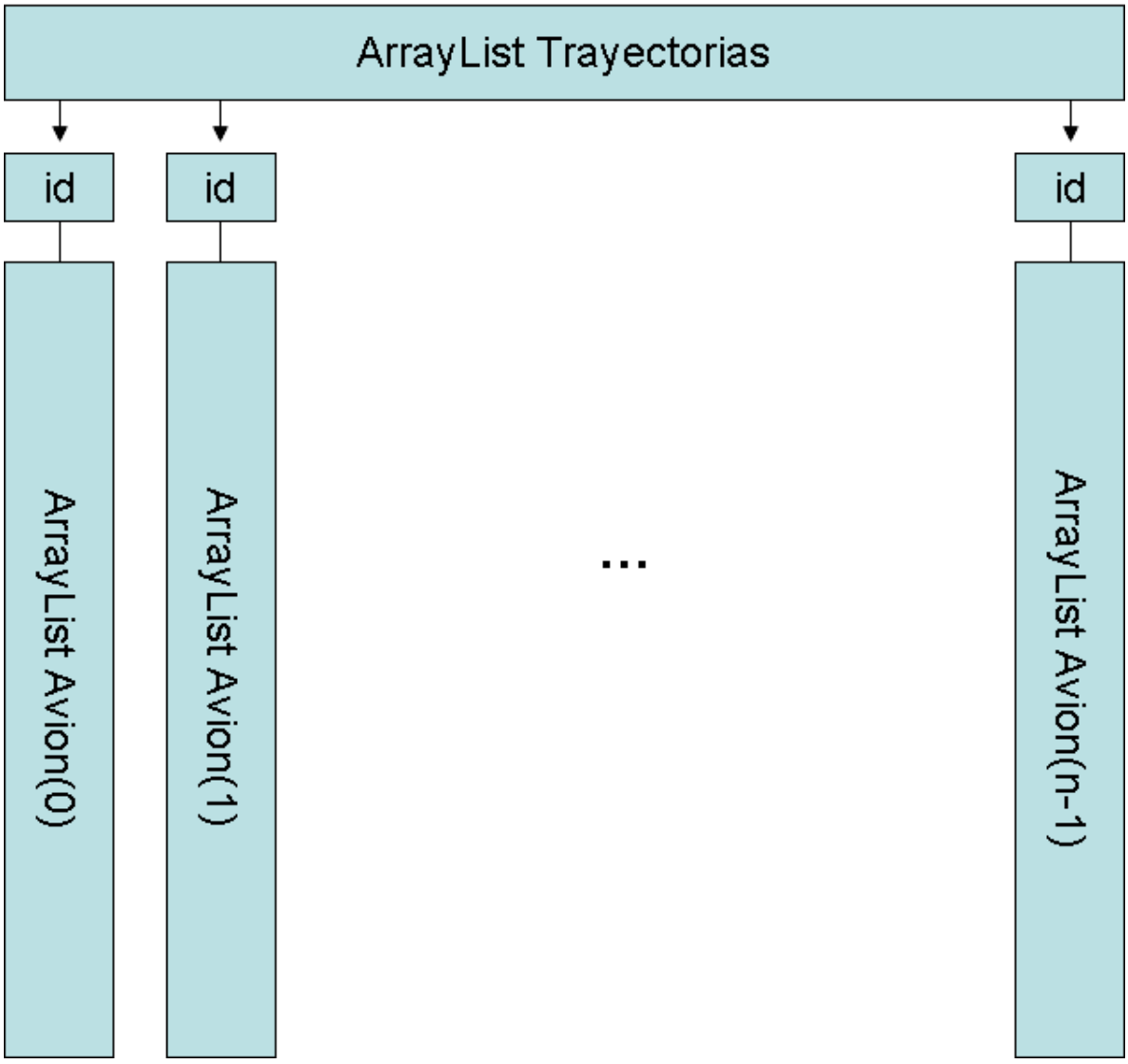
- La reproducción y
- La muestra de información

Reproducción

Para poder llevar este proceso a cabo se parte del hecho que se ha cargado un fichero descriptor de resultado en el objeto Solución por medio del objeto SceneryLoader. En él se disponen de diversos ArrayList con las trayectorias de los aviones y misiles involucrados y, además, con un ArrayList de contiene objetos de tipo Evento y otro que contiene objetos de tipo

Deteccion. En este apartado nos centramos en los ArrayLists de las trayectorias descritas por aviones y misiles si los hubiere.

El ArrayList de trayectorias de avión contiene a su vez objetos de tipo ArrayList (uno por cada avión), que contendrán objetos de tipo TiempoPosicionAvion, y el de trayectorias de misil, análogamente con objetos de tipo TiempoPosicion, puesto que al misil no se le ha dotado de rotación.



La acción de reproducción está basada en el objeto GestorTiempo. Se hace uso del bucle principal de refresco de la escena (ver anexo TrueVision 3D) para actualizar la escena reubicando los móviles involucrados en sus nuevas posiciones. Esto se producirá mientras el botón Play esté pulsado.

Para las funciones Step y Backstep, se hace uso de los método de la clase GestorTiempo, donde se podrán actualizar los móviles a un instante de tiempo posterior o anterior. Para la función Restart, se actualizan los móviles a su primer instante de tiempo.

Muestra de Información

Para mostrar los eventos ocurridos y probabilidades de detección, se hace uso de la clase Solución, nuevamente. Ahí están contenidos loas ArrayList de Eventos y Detecciones (albergan objetos de tipo Evento y Detección, respectivamente).

Cada objeto de tipo evento o detección lleva asociado un instante de tiempo. El procedimiento consiste tan solo en retraer los eventos y probabilidades de cada ArrayList para cada instante de tiempo durante la reproducción mediante los objetos GestorEventos y GestorDetecciones. Después de ello se muestran donde se consideren oportunas. En nuestro caso, los eventos se van acumulando en sendos TextBox.

3.3.7 POSIBLES MEJORAS Y ERRORES CONOCIDOS

- Inclusión de una opción para grabar la secuencia en formato AVI o WMV.
- Posible mejora en la muestra de información.
- Mejoras en el apartado gráfico (no relevante).

3.4 MÓDULO CÁLCULO DE TRAYECTORIAS

3.4.1 DESCRIPCIÓN

Con los datos del escenario de actuación (aviones, radares, obstáculos) se calcularán las trayectorias de los vehículos aéreos para conseguir sus objetivos. El cálculo se realizará en las siguientes fases:

1. Trayectoria "aproximada" utilizando el algoritmo Astar.

$$Ta=[(X_a,Y_a),(X_b, Y_b),\ldots]$$

2. Corrección de trayectorias (evitar el zigzag producido por el algoritmo Astar.

3. Trayectoria fina utilizando interpolación con splines cúbicos naturales, a partir de la trayectoria Ta.

$$Ts = [(X_0\ Y_0), (X_1\ Y_1),\ldots.., (X_{i-1}\ Y_{i-1}), (X_i, Y_i), (X_{i+1}\ Y_{i+1}),\ldots.]$$

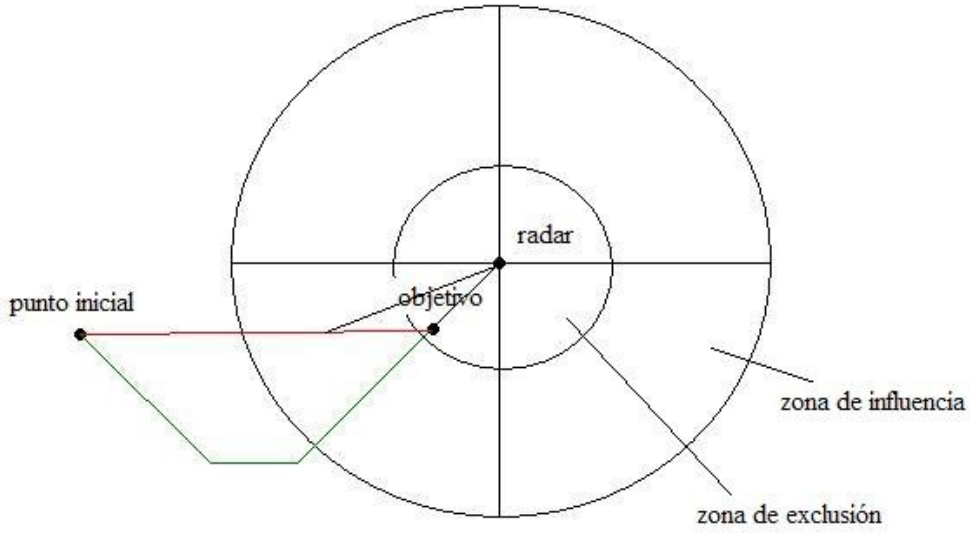
i = índice de tiempo

4. Determinación de la actitud del vehículo aéreo a lo largo de su trayectoria (ángulos de orientación del avión).

3.4.1.1 Requerimientos del cálculo de trayectorias

En las proximidades de un sistema Radar-Misil un UAV está expuesto a ser detectado por el radar y a continuación a ser seguido por el mismo (tracking), es decir, conocer su posición a lo largo del tiempo. Esta información puede ser utilizada para guiar un misil lanzado contra el UAV. Será pues de vital importancia que cuando un UAV entre en la zona de influencia de un radar lo haga de tal manera que disminuya la probabilidad de ser detectado y seguido por el mismo. Dos son los factores más determinantes en la detección y seguimiento de un UAV por un radar: el RCS (Radar Cross Section) y el rango (distancia del UAV al radar). El RCS depende de la actitud del UAV con respecto a la línea de vista del radar. Sin entrar en detalles cuantitativos, podemos decir que un UAV estará más expuesto a la detección y seguimiento del radar cuanto mayor sea la sección que presenta a la línea de vista del radar. Es decir, si nos limitamos a una trayectoria plana (altitud constante) cuanto más perpendicular sea la trayectoria del UAV a la línea de vista del radar mayor será la probabilidad de ser detectado.

En la siguiente figura se representan dos trayectorias desde un punto inicial a un objetivo que se encuentra en las proximidades del radar. La trayectoria de color rojo sería la que generaría un algoritmo de tiempo (distancia) mínimo, por el ejemplo el A* común, ignorando el radar. La trayectoria de color verde tendría en cuenta la presencia del radar, y trataría de minimizar su exposición con una trayectoria cuya dirección se mantiene coincidente con la línea de vista del radar el mayor tiempo posible.



Contexto de utilización

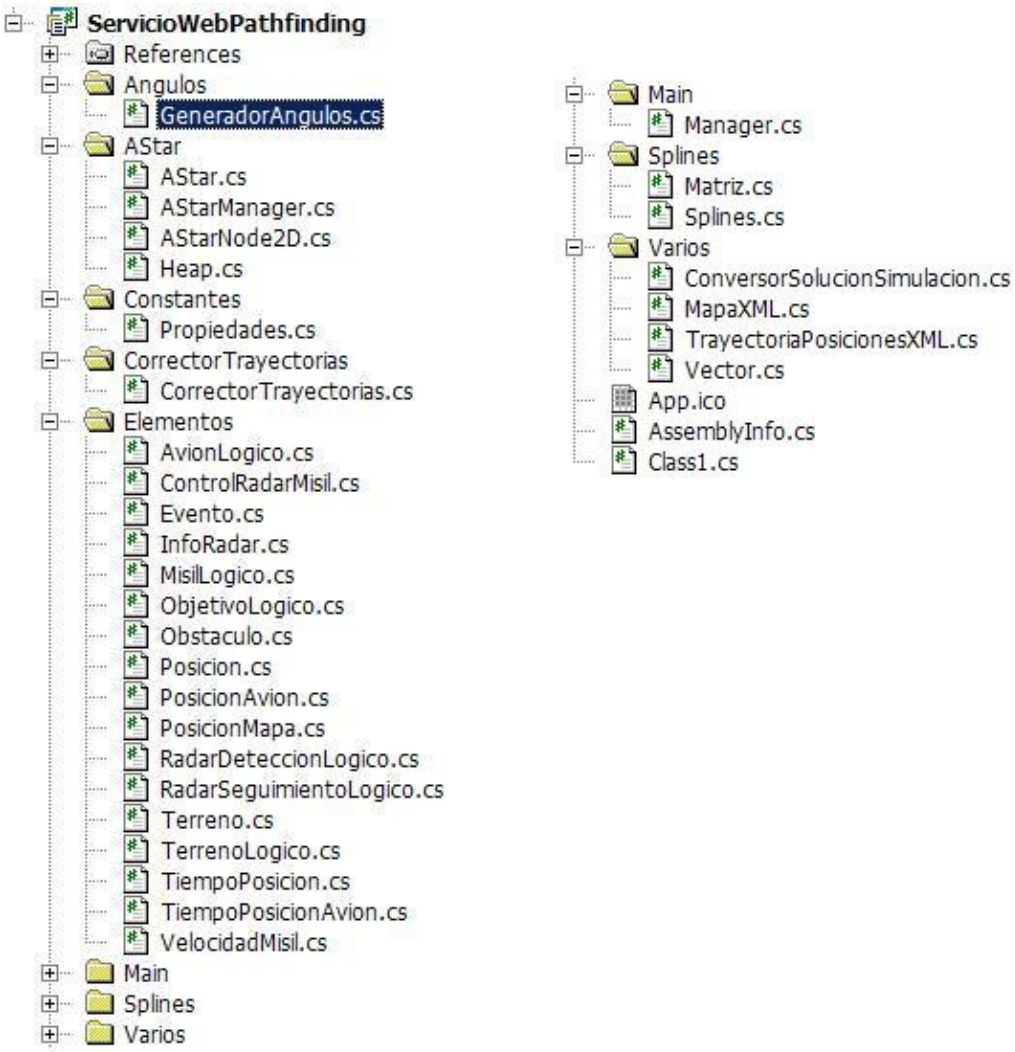
Consideremos dos zonas en torno al radar: la zona de exclusión, que es la zona circular más próxima al radar y que es evitada por el algoritmo A*, y la zona de influencia, zona circular en torno al radar, que incluye a la de exclusión, y que se considera necesaria para la reelaboración del tramo de trayectoria con menor probabilidad de amenaza con respecto al radar.

Las trayectorias generadas por el algoritmo A* no atravesarán la zona de exclusión a menos que no haya otra posibilidad por estar el objetivo dentro, pero sí podrán atravesar la zona de influencia.

En función de un parámetro configurable (parámetro k o de peso de radar-distancia) los UAV evitarán los radares en mayor o menor medida.

Más adelante, en el apartado *3.4.6.1. Algoritmo A Estrella* empleado se describe en detalle las particularidades para conseguir los objetivos mencionados. Por otro lado, en el apartado *4. Ejemplos de ejecución* se pueden consultar ejemplos de lo conseguido claramente explicados.

3.4.2 ARQUITECTURA, PAQUETES Y CLASES



Namespace Angulos

GeneradorAngulos.cs:

- Recibe la lista de posiciones del avión ya suavizado y genera los ángulos.

Namespace AStar

AStar.cs:

AStarNode : Icomparable
- Clase base para los nodos del pathfinding que almacena información sobre el mapa. Nuestro clase de nodos (AstarNode2D), específica para las condiciones del pathfinding concreto, hereda de está.

AStar
- Clase que implementa el AStar pathfinding. El método "FindPath" implementa el algoritmo.

AStarManager.cs:
- Clase que gestiona las llamadas a la búsqueda con AStar. Funciona como adaptador de entradas-salidas al AStar.

AStarNode2D.cs:
- Para búsquedas en un mapa bidimensional. En nuestro caso, además contiene todas las particularidades del A Estrella de este proyecto como por ejemplo la heurística modificada por los radares. Implementa las funciones "AddSucessor" y "Calculate", en la segunda es donde se hacen todos los cálculos heurísticos que se explican en el apartado *3.4.6.1. Algoritmo A Estrella.*

Heap.cs:
- Estructura de datos preparada para mantener una lista ordenada como sea necesario.

Namespace Constantes

Propiedades.cs:
- Contiene los parámetros del sistema. Su diseño es tal, que pueden ser accedidos de manera estática, a la vez que se pueden modificar dinámicamente en ejecución.

Namespace CorrectorTrayectorias

CorrectorTrayectorias.cs:
- Se encarga de minimizar el efecto de zigzag devuelto en las trayectorias del Astar.

Namespace Elementos

AvionLogico.cs:

- Representación lógica del avión, ya que en este punto no es necesarios los datos para una representación 3D.

ControlRadarMisil.cs:
- Lleva el control del par radarDeteccion - radarSeguimiento y de una serie de misiles.

Evento.cs:
- Eventos con sus instantes de tiempo e identificadores de los elementos implicados.

InfoRadar.cs:
- Clase de los objetos que contendrán diversa información para una casilla de tipo PosicionMapa para los cálculos heurísticos en el AStar.

MisilLogico.cs
- Representación lógica de los misiles, con sus posiciones.

ObjetivoLogico.cs
- Representación lógica del objetivo, para los cálculos en el pathfinding.

Obstaculo.cs

Posiciones.cs

PosicionAvion.cs

PosicionMapa.cs
- Representar cada posición del mapa del AStar. Contendrá la información necesaria para indicar que hay en cada "casilla" discreta del mapa, por ejemplo si hay o no obstáculo.

RadarDeteccionLogico.cs

RadarSeguimientoLogico.cs

Terreno.cs
- Representación del terreno empleado en el escenario, usado para generar el mapa discreto que emplea el Astar.,

TerrenoLogico.cs
- No usado, sustituido por Terreno.

TiempoPosicion.cs
- Asocia una posición con un instante de tiempo.

TiempoPosicionAvion.cs
- Asocia una posición de un avión con un instante de tiempo.

VelocidadMisil.cs
- Velocidad del misil en el eje X, Y y Z, con un método para obtener el módulo.

Namespace Main

Manager.cs
- Eje central del servicio web. Realiza varias fases: recibe los datos (toda la información sobre el mapa) y llama al A*. Se modifica la trayectoria, evitando el zigzagueo producto del A*. Llama a realizar el suavizado a partir de la trayectoria obtenida y la lista de instantes de tiempo. Llama a realizar el cálculo de los ángulos de vuelo del avión con la trayectoria suavizada obtenida. Finalmente llama al simulador para que genere las trayectorias de todos los móviles y hace lo que sea con el resultado.

Namespace Splines

Matriz.cs

Splines.cs
- Implementa el algoritmo de interpolado de trayectorias mediante el algoritmo splines cúbicos naturales.

Namespace Varios

ConversorSolucionSimulacion.cs
- Esta clase servirá para convertir estos objetos a XML y viceversa.

MapaXML.cs
- Representa el mapa a una altura dada en XML. No usado.

TrayectoriaPosicionesXML.cs
- Representa una trayectoria completa en formato XML. Sobre cada posición contendrá información sobre posición y estado del avión (inclinaciones y orientación). Puede obtenerse en forma de ArrayList y XmlDocument.

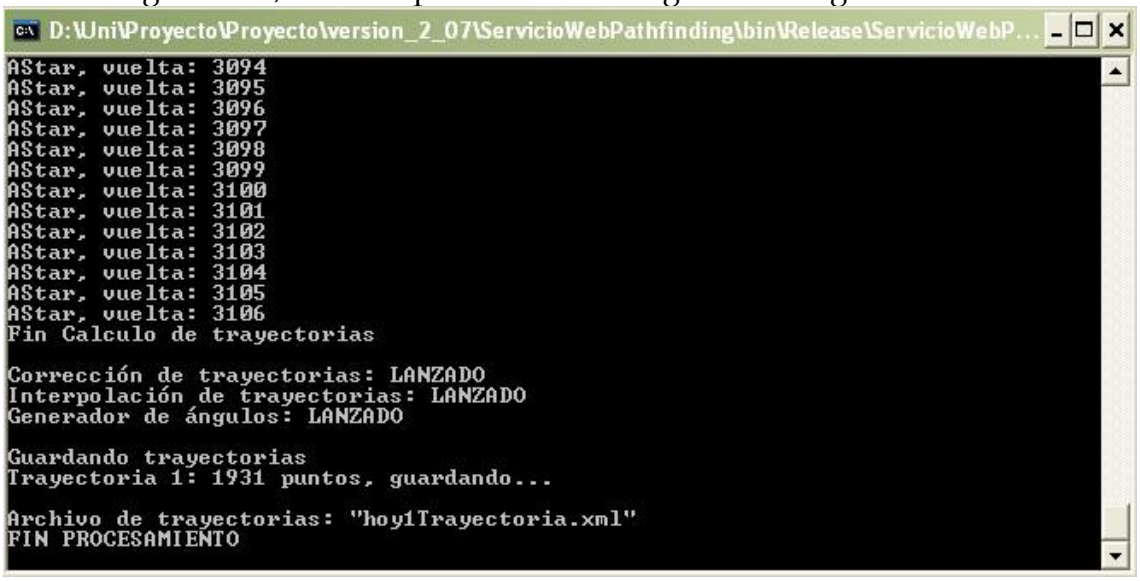
Vector.cs

3.4.3 CONSIDERACIONES DE EFICIENCIA

(Ver anexo 5, Consideraciones de Eficiencia).

3.4.4 INTERFAZ

En este caso, la interfaz es sencilla y de tipo consola. No hay interacción con el usuario, ya que todas las opciones se seleccionan en la interfaz principal. Simplemente se muestra un resumen de los procesos realizados, y el fichero generado, como se puede ver en la siguiente imagen.



3.4.5 ALGORITMOS EMPLEADOS

En este apartado se explica en detalle cada uno de los algoritmos principales utilizados en este módulo.

3.4.5.1 Algoritmo A Estrella

Algoritmo A Estrella común

El algoritmo A Estrella es probablemente el más popular para el cálculo de trayectorias óptimas, por ser eficiente y flexible. La razón de que trabaje tan bien, es que favorece los puntos que están cercanos tanto al de salida como al de llegada, mediante el coste acumulado y la heurística (coste estimado al objetivo) respectivamente.

Los movimientos posibles son en 8 posiciones en este caso, y la heurística inicial la distancia Euclidea (distancia diagonal del punto al objetivo). Está

heurística por si sola es admisible, lo que conlleva que el algoritmo siempre encuentra la solución óptima.

Internamente el algoritmo A Estrella tiene dos listas, la de "nodos abiertos", que almacena todos los nodos que son posibles caminos al objetivo, y la de "cerrados" que almacena todos los nodos que se han expandido.

Para mantener baja la complejidad, se utiliza un ArrayList siempre ordenado, denominado como Heap, y que implementa los interfaces IList y ICloneable.

Existe más información sobre el A Estrella genérico en el *Anexo 7: Algoritmo A Estrella.*

Algoritmo A Estrella modificado

Variante del algoritmo A* en el que los destinos posibles desde un punto del espacio están limitados por la orientación a 3 (giros de 0 o 45 grados).

Como en cualquier algoritmo A*, la función de coste se representa como F = G+H, siendo H la distancia del punto actual al objetivo (Euclidea en nuestro caso) y G la distancia del punto inicial al actual. Pero en este caso será una distancia con peso, es decir, además de la distancia física tendrá en cuenta la dirección de cada uno de los pasos con respecto a la línea de vista de los radares que afecten a ese punto, así como el valor del rango. Entonces:

G = Fg(D, V, R)

- D: distancia acumulada.
- V: valor integral (acumulado) de las direcciones de los pasos de trayectoria con respecto a la línea de vista del radar.
- R: rango del punto actual respecto a las coordenadas del radar.

La función Fg empleada es:

$Fg = Ksen(Beta)/R^4$

- K: parámetro adjustable por el usuario, que otorga más o menos peso a la presencia de radares en el cálculo de las trayectorias.
- Beta: ángulo que forma la orientación del avión con el vector dirección del radar al avión. El peor caso estará cuando valga 90 grados ya que la incidencia es máxima, mientras que 0 grados será lo más favorable (el avión muestra el mínimo flanco a la línea del radar).
- R: rango o distancia del avión al radar.

Estructura de la solución en la implementación

Veamos donde se ha aplicado todo lo explicado desde un punto de vista práctico o de código.

Partiendo del algoritmo A Estrella genérico con una clase *AstarNode* que no realiza cálculos en relación con las coordenadas, se ha creado una heredera de la anterior denominada *AStarNode2D* y que contendrá las particularidades de nuestro caso.

El algoritmo genérico se corresponde con las clases *AStar* (que contiene el método que implementa el algoritmo, denominado *FindPath*), *AStarNode* y *Heap*.

Para adaptar el algoritmo a las necesidades concretas se han creado *AStarManager* y *AStarNode2D*. La primera clase simplemente gestiona las llamadas a la búsqueda con Aestrella, funcionando como adaptador de las entradas y salidas al AStar desde el manager principal del módulo de cálculo de trayectorias. El segundo requiere más atención, ya que implementa los métodos que aportan las particularidades ya explicadas:

- *IsSameState():* decide si un nodo coincide con otro según su estado, que viene dado por la posición y la orientación o yaw concreta.

- *GetSuccessors():* obtiene todos los posibles sucesores, teniendo en cuenta que no haya obstáculos y que según la orientación actual podemos obtener 3 posibles sucesores (no se permiten giros de 90 grados, sí de 45).

- *Calculate():* calcula el *GoalEstimate* (la heurística) de acuerdo a la información del estado y el nodo objetivo. Aquí se realizan los cálculos de la heurística referentes a los radares.

3.4.5.2 Proceso de corrección de trayectorias

El primer y más básico paso en hacer más realista una trayectoria devuelta por un algoritmo AStar es eliminar los efectos de zigzag que produce, como se puede ver en la figura de más abajo. Este efecto es causado por el hecho de que en el algoritmo A Estrella estándar desde una posición se puede sólo a las 8 colindantes, para luego continuar de forma similar.

El resultado es muy negativo para una trayectoria como la buscada en este proyecto, que debe ser lo suficientemente fina para coincidir con la de un avión real.

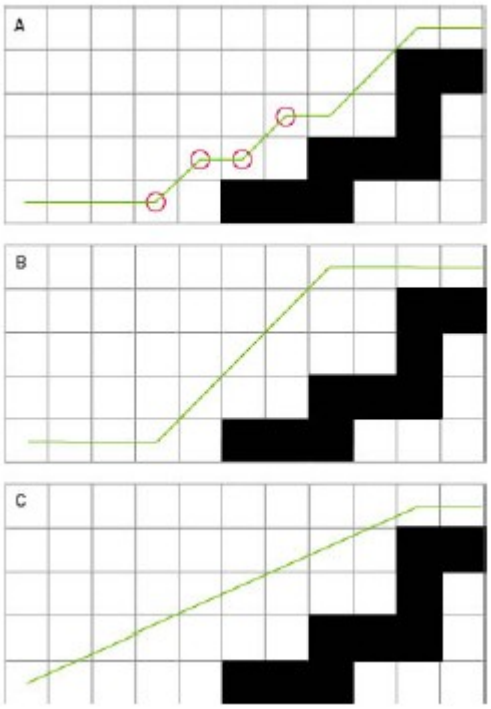


FIGURA
(a) Ilustra el efecto de zigzag producido por el algoritmo AStar.
(b) una modificación con un efecto todavía no deseable.
(c) La ruta más directa y deseada. Para lograrla, se eliminan los puntos intermedios resaltados en rojo en la figura (a).

Un método sencillo para reducir el número de giros sería realizar la siguiente modificación al algoritmo AStar: añadir una penalización de coste cada vez que un giro es tomado, favoreciendo rutas que tienen la misma distancia, pero hacen menos giros, como se observa en la figura (b).

Sin embargo, está simple solución no es muy efectiva, entre otros motivos porque todos los giros todavía son de 45 grados y no se logra el realismo adecuado.

Entonces, deseamos conseguir una ruta del tipo de la mostrada en la figura (c), ya que toma la ruta más directa, con unos ángulos cualquiera. Esto se puede lograr con un simple pero efectivo algoritmo de suavizado o de corrección de trayectorias, que está implementado en este proyecto en "CorrectorTrayectorias.cs" del namespace "CorrectorTrayectorias".

El algoritmo hace uso de una función "walkable(posicionA, posicionB), que muestrea puntos en una línea que va de la posición A a la B (ambas posiciones de la trayectoria del A Estrella) con un determinado intervalo para comprobar en cada uno si se encuentra en una zona bloqueada por obstáculo (en el caso que nos ocupa, además de los obstáculos se tienen en cuenta las zonas de influencia de los radares). Devuelve "cierto" si no hay ningún elemento entre medias.

El siguiente fragmento, muestra el método implementado, en forma de pseudocódigo:

```
checkPoint = starting point of path
currentPoint = next point in path
while (currentPoint->next != NULL)
if Walkable(checkPoint, currentPoint->next)
// Make a straight path between those points:
temp = currentPoint
currentPoint = currentPoint->next
delete temp from the path
else
checkPoint = currentPoint
currentPoint = currentPoint->next
```

El algoritmo simplemente va comprobando de punto en punto (waypoints) a través de la ruta, intentando eliminar los intermedios cuando es posible.

3.4.5.3 Proceso de interpolado, mediante splines cúbicos naturales

Consideraciones y definiciones previas

Podemos ver un spline como una curva definida "a trozos" mediante polinomios. Es por ello que en muchas ocasiones en análisis numérico denominamos interpolación segmentaría a la realizada con este tipo de técnica, fundamentada en el uso y correcta unión de segmentos de polinomios, en lugar de utilizar solamente un polinomio.

Este modo de interpolación mediante splines ha demostrado ser muy fino y preciso, soliendo obtener resultados muy similares a los originales para los problemas de interpolación, requiriendo únicamente la utilización de polinomios de pequeño grado, y al mismo tiempo que se evitan las siempre indeseables oscilaciones que resultan de realizar interpolaciones con polinomios de alto grado.

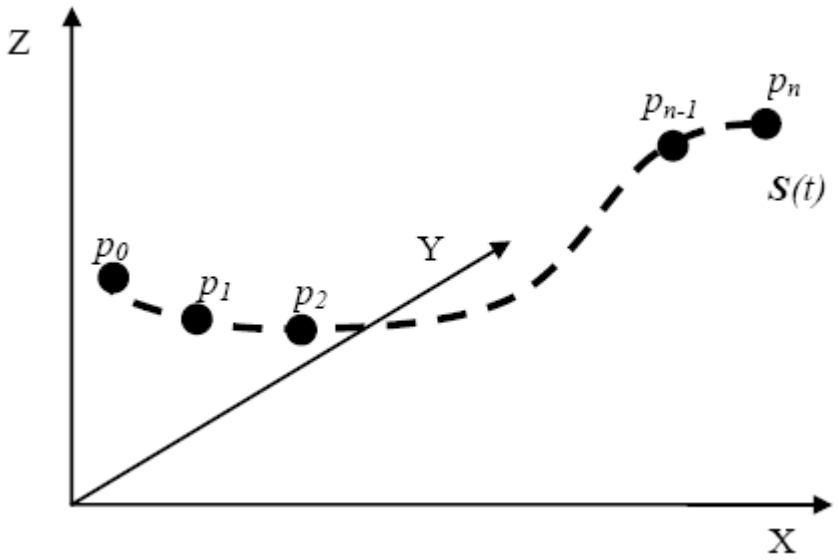
Por tanto, se podría decir que una función spline está formada por varios polinomios, cada uno definido en un intervalo determinado y unidos entre

si bajo una serie de condiciones de continuidad, las cuales exponemos a continuación.

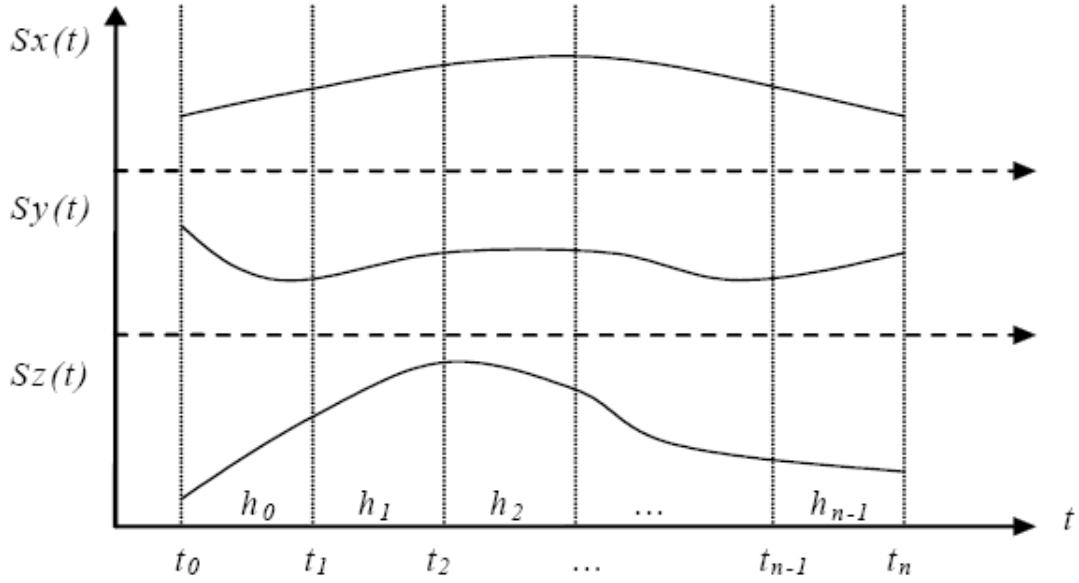
Definimos interpolación como un proceso matemático que permite, a partir de un conjunto discreto de puntos dados, por ejemplo una trayectoria, la generación de una nueva serie de puntos más precisa en su resolución. Informalmente, en una trayectoria, se consigue un efecto de "suavizado".

Como parece lógico, pueden existir distintos tipos de funciones que interpolen los mismos datos; por ejemplo, funciones trigonométricas, funciones exponenciales, funciones polinomiales, etc. En este proyecto se emplea el método de splines cúbicos naturales.

Vamos a construir la trayectoria de interpolación S(t) para un móvil que pasa por las posiciones espaciales p_0, p_1, p_2, ….,p_{n-1}, p_n en los instantes de tiempo t_0, t_1, t_2, …, t_{n-1}, t_n. S(t) estará definida para todas las posiciones del espacio correspondientes a todos los valores de t entre t_i y t_{i+1} para i=0,…,n, es decir, para t ε R.



La trayectoria de interpolación será una función paramétrica tridimensional S(t) de parámetro t que pasa por los puntos de la tabla de interpolación. Las tres dimensiones de S(t) las aproximaremos por unas funciones Sx(t), Sy(t) y Sz(t). Cada función Sj(t) se construye con un conjunto de polinomios de grado 3, uno para cada intervalo $h_i = t_{i+1} - t_i$ (intervalos de tiempo) con i = 0,…,n-1. Aunque en nuestro caso, sólo harán falta dos, ya que la altura es constante.

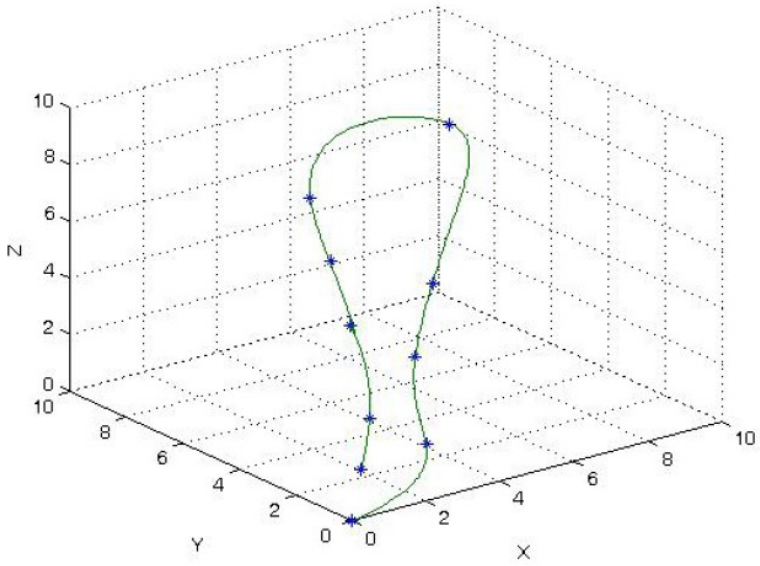


Dos ejemplos ilustrativos

Ejemplo 1:

Tenemos los siguientes puntos, con sus respectivas coordenadas x, y, z, queremos construir una trayectoria que pase por esos puntos. Serán necesaria una gran cantidad de puntos para obtener una trayectoria.

	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9
x	0	2	4	6	8	5	4	3	2	1
y	0	0	3	5	7	8	6	4	2	1
z	0	2	3	4	8	6	5	4	2	1
t	0	4	6	7	11	15	16	21	28	30

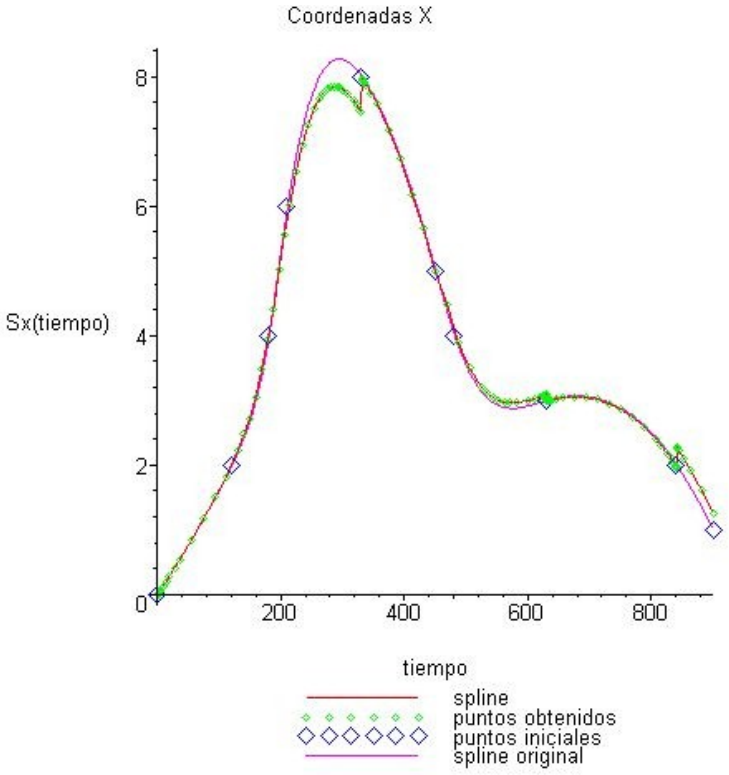


Ejemplo 2:

Partiendo de las siguientes posiciones x, junto con sus correspondientes intervalos de tiempo,

px = [0,2,4,6,8,5,4,3,2,1]
t = [0,4,6,7,11,15,16,21,28,30]

Obtenemos este gráfico usando splines cúbicos naturales:



La leyen los puntos obtenido

A continuación se muestra la parte del código en la que una vez construido el polinomio correspondiente, se le da valores para obtener las posiciones en los instantes de tiempo que se desea:

```
//Construye la tabla de valores interpolados w, es el factor
//de densidad del parámetro t

int w = PropertiesManager.properties.WSPLINES;
for (int j=0; j < h.Count; j++)
{

    for (int i=w*Convert.ToInt32(t[j]); i<=w*Convert.ToInt32(t[j+1]); i++)
    {

        double iw = (double)i/(double)w;

        double sum1 = (double)(Convert.ToDouble(ct3[j])*
            Convert.ToDouble(Math.Pow(iw,3)));
        double sum2 = (double)(Convert.ToDouble(ct2[j])*
            Convert.ToDouble(Math.Pow(iw,2)));
        double sum3 = (double)(Convert.ToDouble(ct1[j])*
            Convert.ToDouble(iw));
        double sum4 = (double)Convert.ToDouble(ct0[j]);
        double valor = (double)(sum1+sum2+sum3+sum4);

        s.Add(valor);

    }

}

return s;
```

Para cada intervalo de tiempo t[j] y t[j+1] se multiplica por el parámetro w, que es el factor de densidad del parámetro t, y se obtienen los puntos para esos instantes de tiempo. Cuanto más grande sea el factor de densidad (w), más puntos se generarán entre los puntos iniciales. Una densidad con un número grande de puntos de partida puede paliar el efecto de "no suavidad" en cierta medida.

Conclusión

El método de Splines es un método muy recomendable cuando se busca un equilibrio sencillez-resultados. En este proyecto se han obtenido resultados muy satisfactorios en el interpolado de las trayectorias devueltas por el algoritmo AStar.

Es importante resaltar que se ha buscado un equilibrio entre resolución de la trayectoria y rendimiento, ajustando el parámetro w. Recordemos que al incrementarlo, aumenta la precisión de la trayectoria interpolada, siendo la

consecuencia un mayor tiempo de guardado de los ficheros XML por existir más puntos.

3.4.5.4 Proceso de generación de ángulos

En este punto, ya se ha calculado la trayectoria del avión y se ha interpolado. El siguiente paso lógico es dotar de una dinámica al modelo del avión en forma de ángulos de giro y balanceo para lograr unos movimientos realistas. Recordemos que se trabaja con altura constante y por lo tanto no es necesario un cálculo del cabeceo.

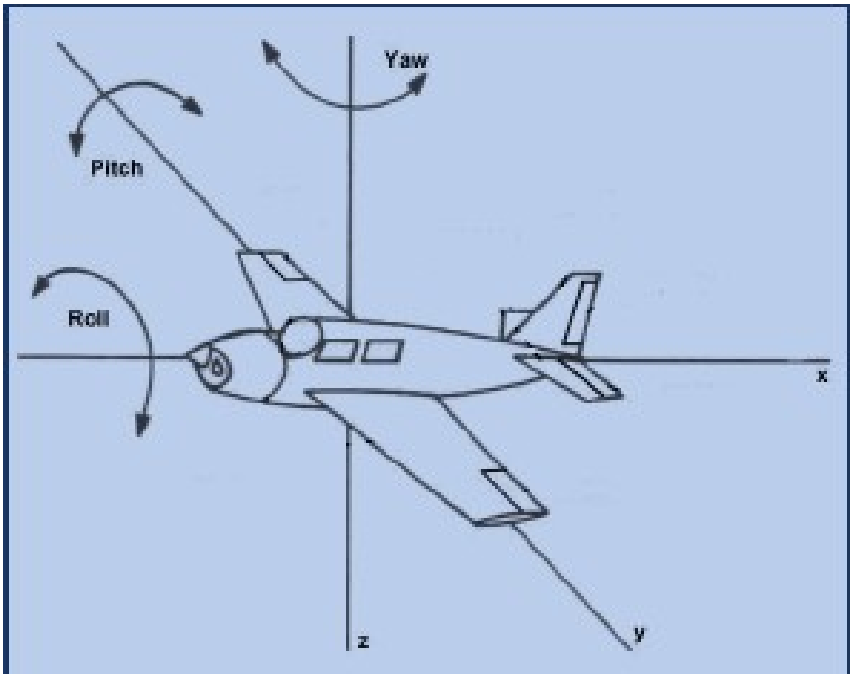
El proceso es el siguiente:

1. Cálculo de la guiñada (yaw) para los diferentes puntos de la trayectoria.

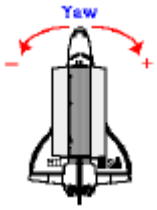
2. Cálculo del balanceo (roll) para los diferentes puntos de la trayectoria, a partir de las guiñadas calculadas previamente

Un avión tiene seis grados de libertad, es decir, seis movimientos independientes. Tres de ellos corresponden desplazamientos, y los otros tres a giros en torno a sus ejes principales. Los desplazamientos no tienen nombre específicos, el del eje X produce un avance del avión sobre su dirección de vuelo; el del eje Z un ascenso o descenso del avión (pero en nuestro caso, ya que consideramos que la altura es constante, la Z, será siempre la misma), y el del eje Y un movimiento lateral. Este último son muy pocos los aviones que lo pueden controlar de forma individual. Los giros del avión en torno a sus ejes si están designados con nombres específicos porque representan maniobras comunes para guiar el avión. Son tres, uno en torno a cada uno de los ejes principales, aunque en el caso de Pitch (cabeceo), no se considera, ya que al ser la altura constante no tiene sentido.

Veámoslo gráficamente:



Guiñada (ψ) (*Yaw*): es un giro en torno al eje vertical (Z) del avión, y produce un cambio en la dirección horizontal de su vuelo.

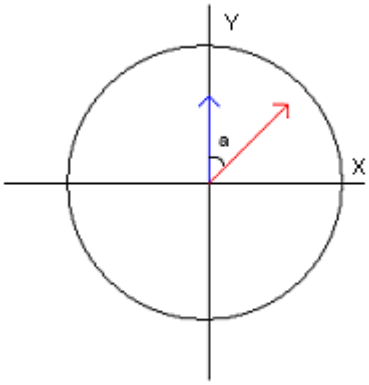


Para calcular esté ángulo se distinguen varios casos, como se verá a continuación. En cada posición el yaw se calculará a partir de las coordenadas x e y de la posición anterior y posterior.

Nombraremos las posiciones x e y posterior de un punto dado como xPos e yPos respectivamente, y las posiciones anteriores como xAnt e yAnt. Nuestro punto actual será siempre el (x,y).

El método empleado es sencillo. En primer lugar se calcula el vector trayectoria en la posición (x,y), que será el que va del punto (xPos,yPos) al punto (xAnt, yAnt). Este vector a continuación se usa como referencia.

Consideremos el siguiente gráfico:



Imaginemos que el avión avanza hacia unas posiciones xPos y yPos mayores que xAnt e yAnt.

El vector de trayectoria formado por el punto (xPos,yPos) y (xAnt,yAnt) es el que aparece representado en rojo, mientras que el vector azul será siempre el vector de referencia, siendo a el ángulo que forman ambos.

Ahora distinguiremos los casos:

1. Caso 1 (xPos>xAnt) y (yPos>yAnt) 'Primer Cuadrante': calculamos los dos vectores, y el ángulo yaw sería igual al ángulo que forman los dos vectores multiplicado por (-1) y sumado 90 grados.

2. Caso 2 (xPos<xAnt) y (yPos>yAnt) 'Segundo cuadrante': al igual que antes, calculamos los dos vectores (el de referencia siempre es el mismo). El yaw se calcula igual que antes, ángulo que forman multiplicado por (-1) y añadirle 90 grados.

3. Caso 3 (xPos<xAnt) y (yPos<yAnt) 'Tecer cuadrante': siguiendo el patrón anterior, pero en este caso el yaw es igual al ángulo que forman los dos vectores y añadiendo 90 grados.

4. Caso 4 (xPos>xAnt) y (yPos>yAnt) 'Cuarto cuadrante': este caso se calcula como el Caso 3.

5. Casos especiales: casos en que una coordenada posterior xPos o yPos coincide con una coordenada anterior xAnt e yAnt. En esto caso el yaw es igual al yaw anterior, ya que el avión sigue la misma trayectoria.

Balanceo (Φ) (*Roll*): es un movimiento en torno al eje longitudinal del avión (eje X) y se aprecia como un desplazamiento lateral del mismo.

Su cálculo se hace en función del Yaw previamente calculado. El método empleado se basa en cada punto determinado en el yaw de la posición anterior y el yaw de la posición posterior. Nombrando como yawAnt y yawPos, al yaw de la posición anterior y posterior respectivamente, el roll se calcularía con la diferencia del yawAnt con el yawPos.

Se ha fijado un ángulo de balance máximo del roll, para ajustarse a la realidad en la que un avión cuando gira toma como máximo un ángulo de aproximadamente 90^{o}, y se ha tenido que fijar una serie de casos límite ya que cuando se trabaja con ángulos hay que tener en cuenta los signos de los mismos, para saber en que cuadrante de la circunferencia estaría.

En el caso de que la diferencia de ángulos yaw fuera 0, el roll para esa posición, sería el roll de la posición anterior.

Cabeceo (θ) (*Pitch*): este giro es realizado en torno al eje de las alas del avión (eje Y) y su efecto se aprecia como un ascenso o descenso del morro. No es considerado por ser la altura especificada de los aviones constante.

3.4.6 POSIBLES MEJORAS Y ERRORES CONOCIDOS

- La generación de ángulos de los aviones podría ser más precisa y científica.
- Los ángulos generados tienen algún pequeño error en puntos aislados.
- Podrían utilizarse métodos más complejos para generar trayectorias más realistas a partir de las obtenidas por el AStar.

3.5 MÓDULO SIMULACIÓN

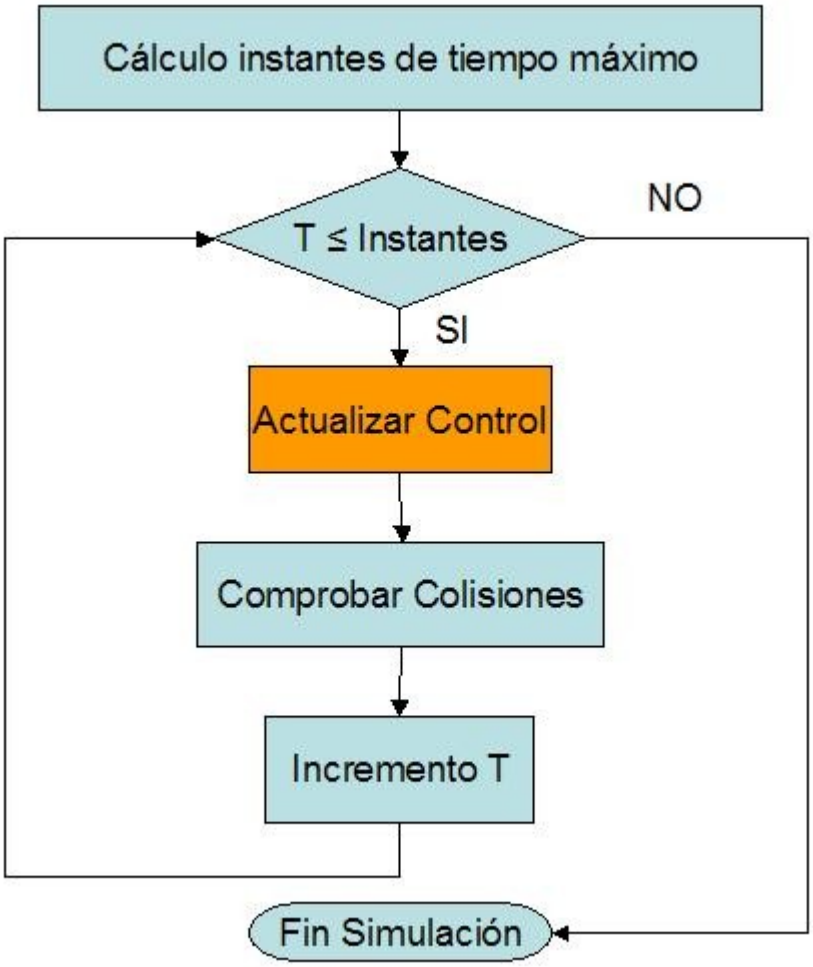
3.5.1 DESCRIPCIÓN

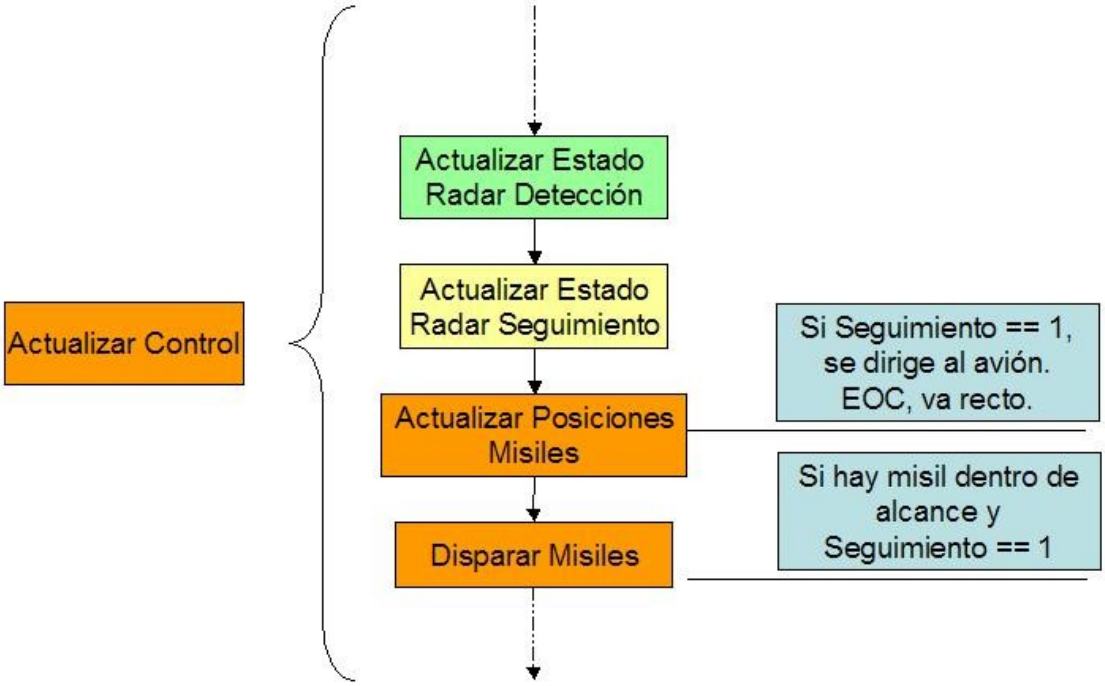
El módulo de simulación es el encargado de recibir el fichero descriptor de trayectorias procedente del módulo de pathfinding y simular el comportamiento de los radares y misiles con respecto a los aviones involucrados en el escenario.

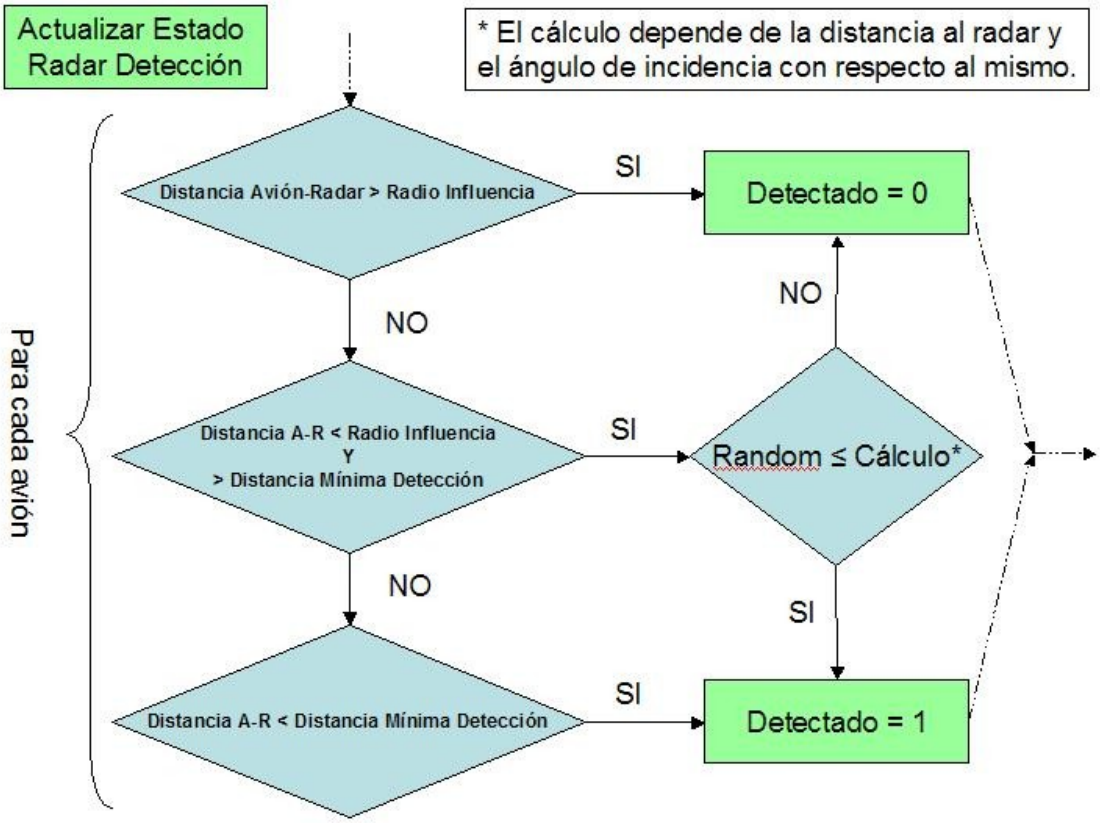
Se realiza en los siguientes pasos:

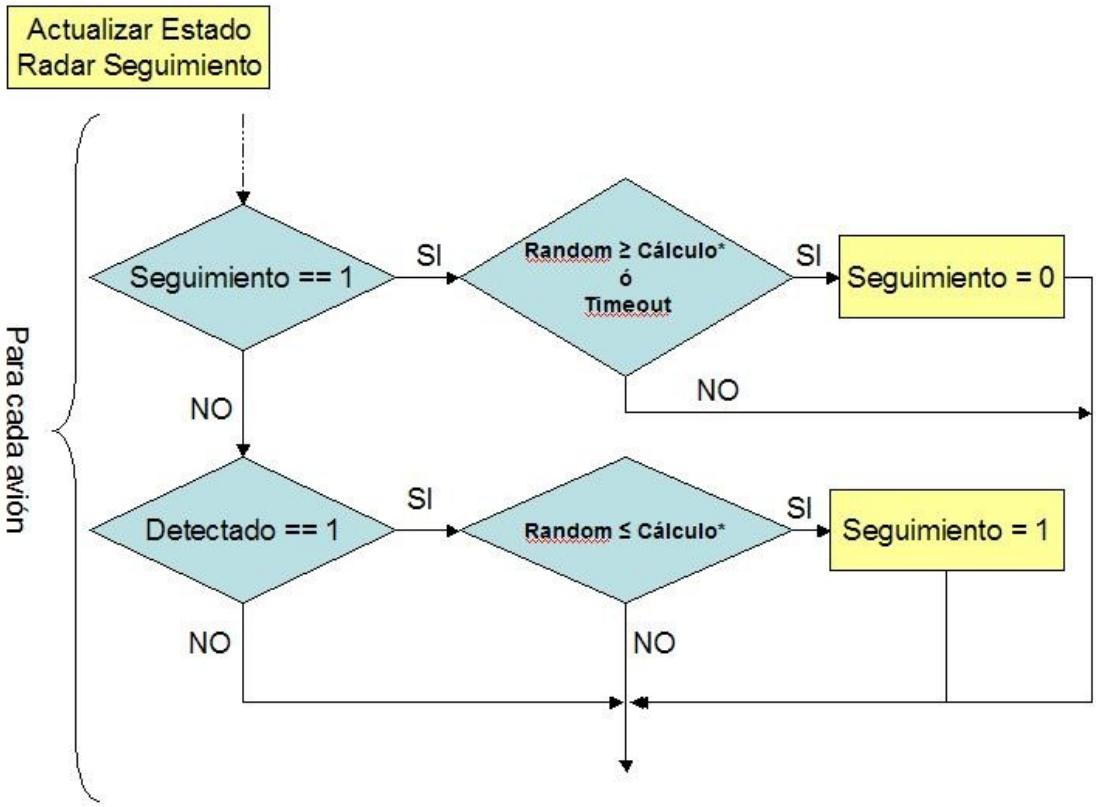
- Cargar el fichero descriptor de escenario dado.
- Cargar el fichero descriptor de trayectorias dado, con respecto al de escenario cargado.
- Llevar a cabo la simulación un número de veces dado.
- Se recoge la distancia alcanzada por cada simulación. La que se repita un número mayor de veces, será la "simulación más probable". En caso de empate, se coge la de menor distancia.
- Se guardan las trayectorias finales de todos los móviles en un fichero descriptor xml de solución, que será la entrada del módulo de visualización.

El proceso de simulación viene descrito por los siguientes diagramas de flujo:

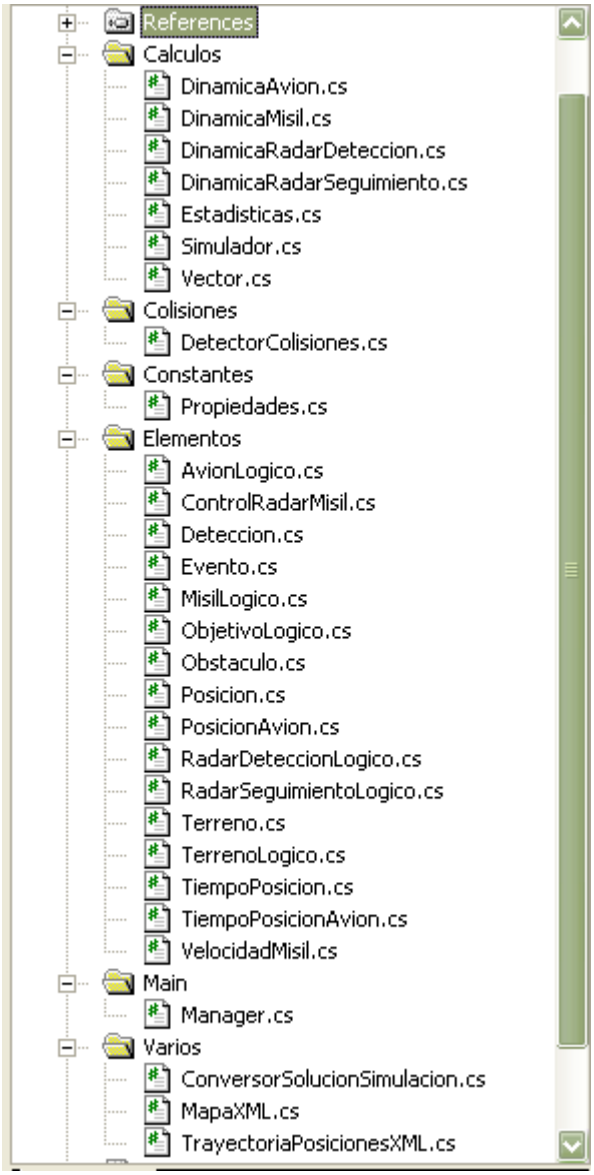

Cálculo instantes de tiempo máximo
T ≤ Instantes
NO
SI
Actualizar Control
Comprobar Colisiones
Incremento T
Fin Simulación


Actualizar Control
Actualizar Estado Radar Detección
Actualizar Estado Radar Seguimiento
Actualizar Posiciones Misiles
Disparar Misiles
Si Seguimiento == 1, se dirige al avión. EOC, va recto.
Si hay misil dentro de alcance y Seguimiento == 1


Actualizar Estado Radar Detección
* El cálculo depende de la distancia al radar y el ángulo de incidencia con respecto al mismo.
Para cada avión
Distancia Avión-Radar > Radio Influencia
SI
Detectado = 0
NO
Distancia A-R < Radio Influencia Y > Distancia Mínima Detección
SI
Random ≤ Cálculo*
NO
NO
SI
Distancia A-R < Distancia Mínima Detección
SI
Detectado = 1



3.5.2 ARQUITECTURA, PAQUETES Y CLASES



Namespace Calculos

DinamicaAvion.cs:

- Implementa la dinámica del movimiento del avión en función del instante de tiempo dado y la lista de posiciones dada.

DinamicaMisil.cs:

- Implementa la dinámica del movimiento del misil en función de la posición anterior, la posición del avión y si el radar de seguimiento asociado está en seguimiento o no.

DinamicaRadarDeteccion.cs:

- Implementa la dinámica de detección del radar. Se basa en un random y en la posición de un avión dentro de la zona de influencia del radar, estableciendo esto una probabilidad de detección.

DinamicaRadarSeguimiento.cs:

- Implementa la dinámica de seguimiento del radar. Se basa en un random para decidir si sigue a un avión dado y, por tanto, le dispara un misil.

Estadisticas.cs:

- Recibe una lista de alcances conseguidos por cada simulación hecha. Determina la trayectoria más probable y el porcentaje de llegadas al objetivo.

Simulador.cs:

- Rige la simulación del escenario.

Vector.cs:

- Representa un vector e implementa operaciones con respecto a otros vectores.

Namespace Colisiones

DetectorColisiones.cs:

- Detecta colisión entre misiles, aviones y terreno.

Namespace Constantes

Propiedades.cs:

- Contiene los parámetros del sistema. Su diseño es tal, que pueden ser accedidos de manera estática, a la vez que se pueden modificar dinámicamente en ejecución.

Namespace Elementos

AvionLogico.cs:

- Representa un avión con sus coordenadas y dimensiones.

ControlRadarMisil.cs:

- Controla un radar de detección, seguimiento y misiles asociados.

Deteccion.cs:

- Representa un nivel de detección de un radar con respecto a un avión.

Evento.cs:

- Representa un evento sucedido en la simulación.

MisilLogico.cs:
- Representa un misil con sus coordenadas.

ObjetivoLogico.cs:
- Representa el objetivo con sus coordenadas.

Obstaculo.cs:
- Representa un obstáculo con sus dimesiones y coordenadas.

Posicion.cs:
- Representa una posición (x, y, z).

PosicionAvion.cs:
- Representa una posición de un avión (x, y, z) y sus ángulos de giro (roll, pitch, yaw).

RadarDeteccionLogico.cs:
- Representa un radar de detección.

RadarSeguimientoLogico.cs:
- Representa un radar de seguimiento.

Terreno.cs:
- Representa el terreno.

TerrenoLogico.cs:
- Representa el terreno.
- Ya no se utiliza, en sustitución de Terreno.cs

TiempoPosicion.cs:
- Asocia un instante de tiempo con una posición.

TiempoPosicionAvion.cs:
- Asocia un instante de tiempo con una posición de un avión.

VelocidadMisil.cs:
- Velocidad del misil en el eje X, Y y Z, con un método para obtener el módulo.

Namespace Main

Manager.cs:

- Realiza todo el proceso paso a paso.

Namespace Varios

ConversorSolucionSimulacion.cs:
- Convierte la simulación en ArrayList a XML

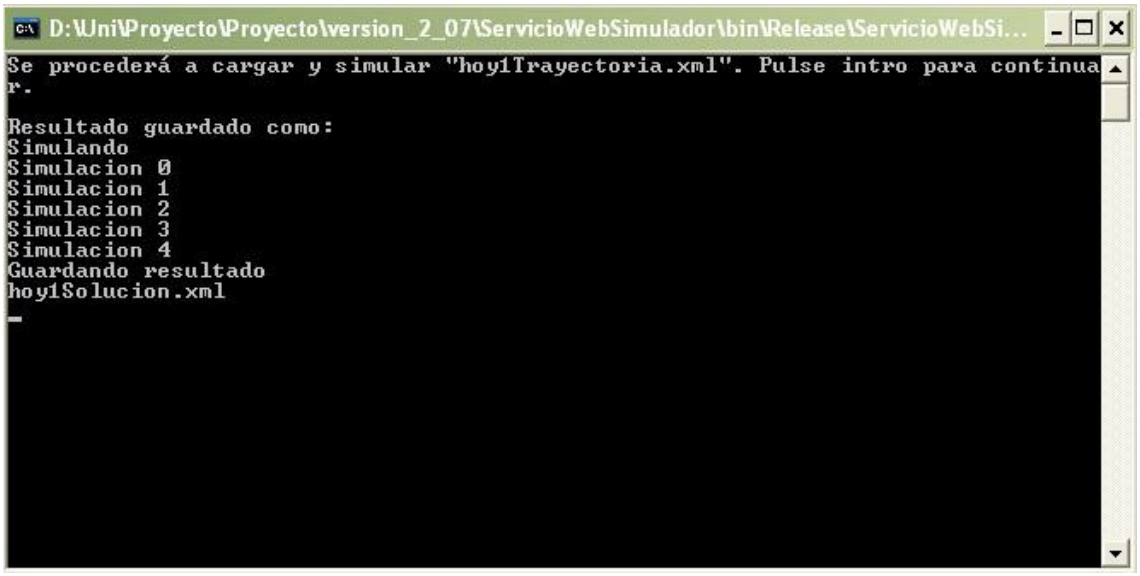
MapaXML.cs:
- Saca un terreno discretizado a un fichero de texto.
- Ya no se usa.

TrayectoriaPosicionesXML.cs:
- Saca una trayectoria en formato XML a partir de un ArrayList.

3.5.3 CONSIDERACIONES DE EFICIENCIA

(Ver anexo de 7.5 Consideraciones de Eficiencia).

3.5.4 INTERFAZ



Si se ejecuta desde la Interfaz General, se muestra el fichero que se va a procesar. Si se ejecuta por separado, se pide el fichero. A continuación, se pide confirmación, y se informa de las simulaciones que se van realizando.

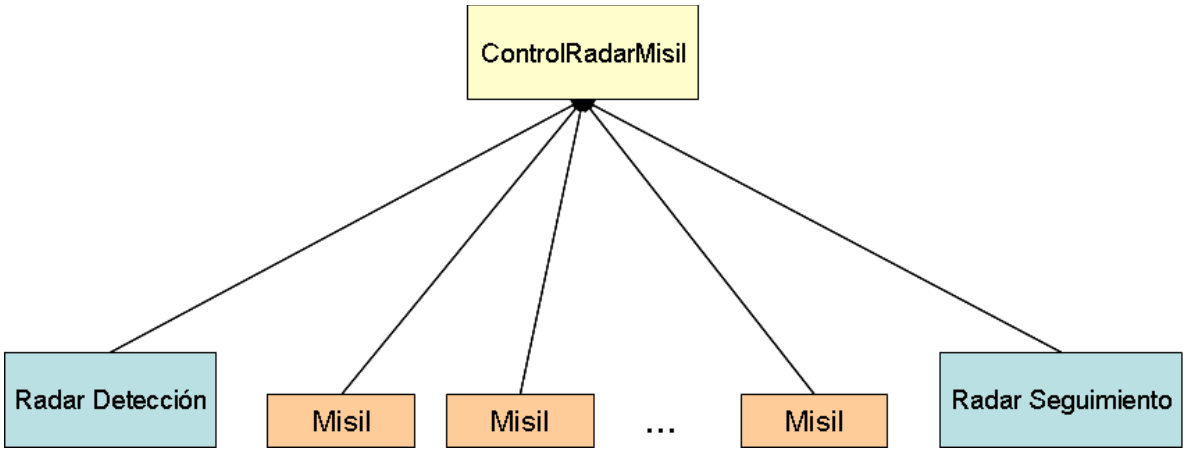
Se confirma al final el fichero de solución guardado.

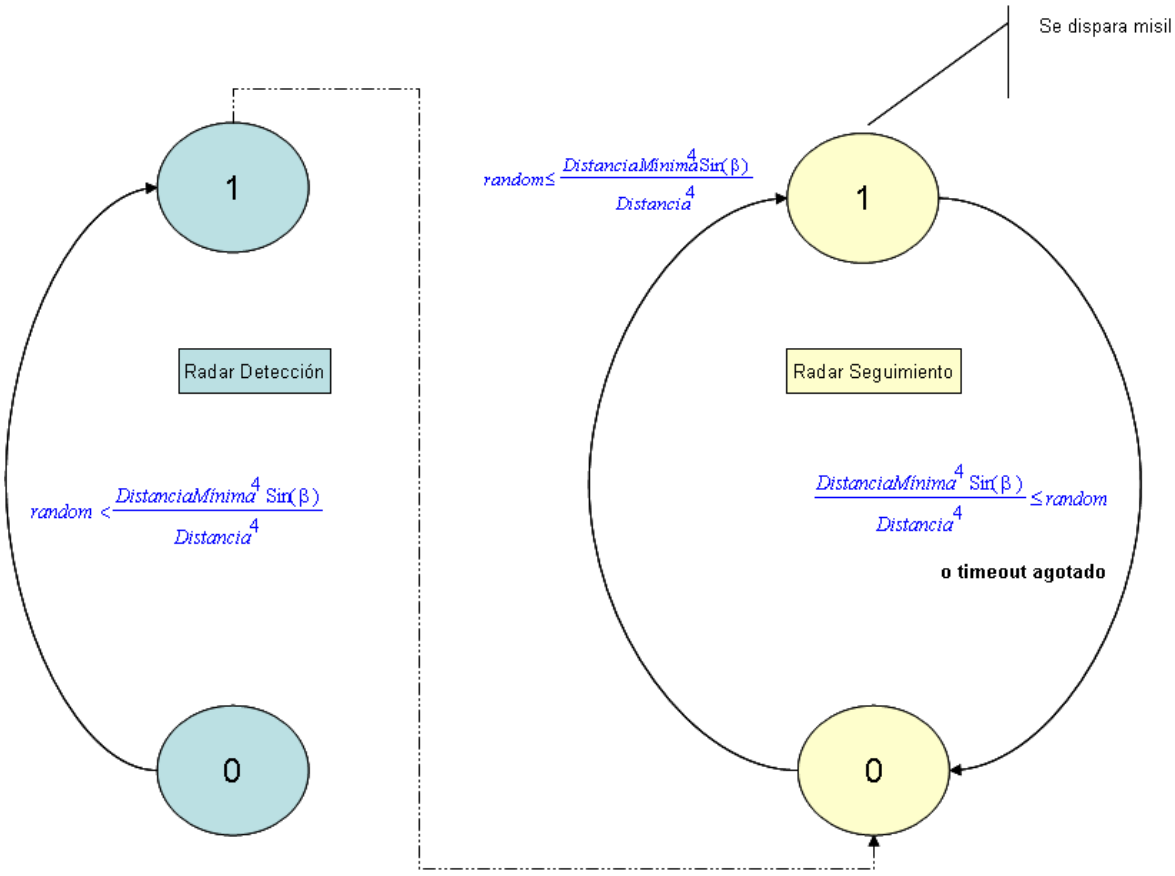
3.5.5 ALGORITMOS EMPLEADOS

Cabe reseñar el proceso de simulación. Se dispone de:

- Aviones
- Radares
- Misiles
- Controles que rigen radares con misiles asociados.

El comportamiento del simulador puede resumirse con el siguiente diagrama:





4 EJEMPLOS DE EJECUCIÓN

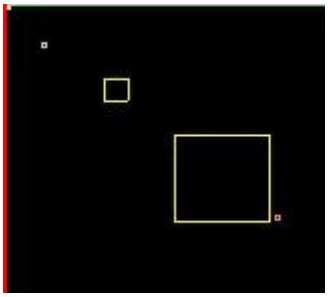
Se ha dividido este apartado en tres categorías de ejemplos.

4.1 EJEMPLOS DE TIPO 1

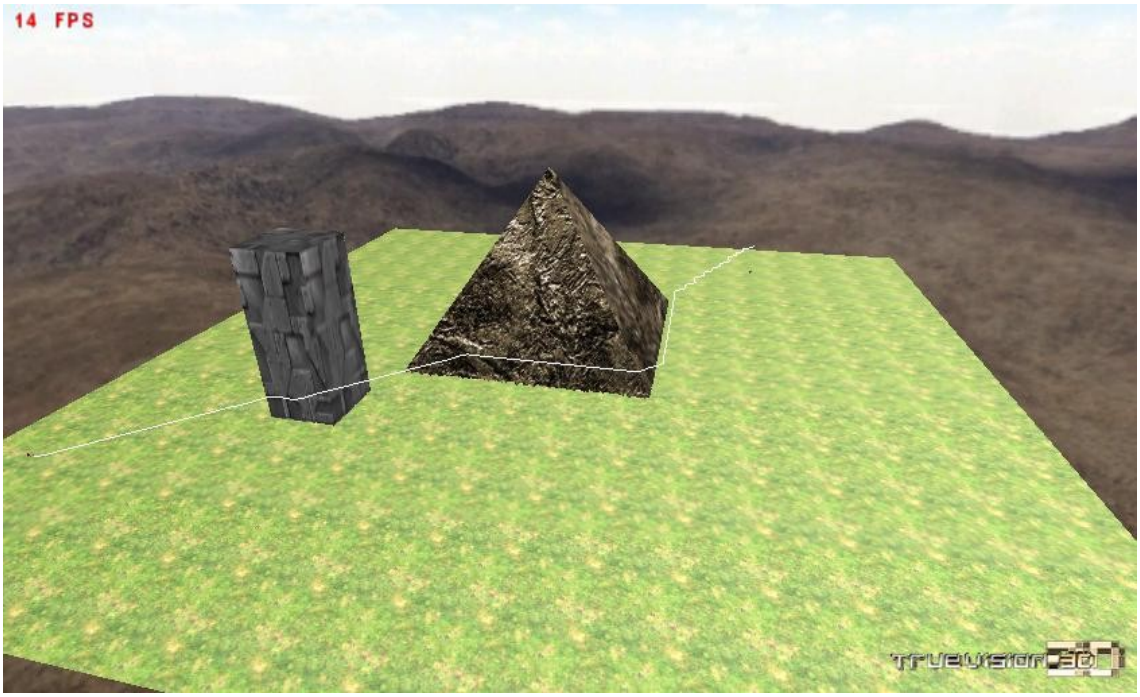
En este apartado mostraremos distintas opciones que podemos realizar, pero obviaremos la cuestión de los radares, que se tratarán en los ejemplos de más abajo.

Ejemplo 1.1

Terreno simple con obstáculos, y sin eliminar el zigzag del avión. Este sería el aspecto del escenario inicial, lo vemos, a través de heightmap.



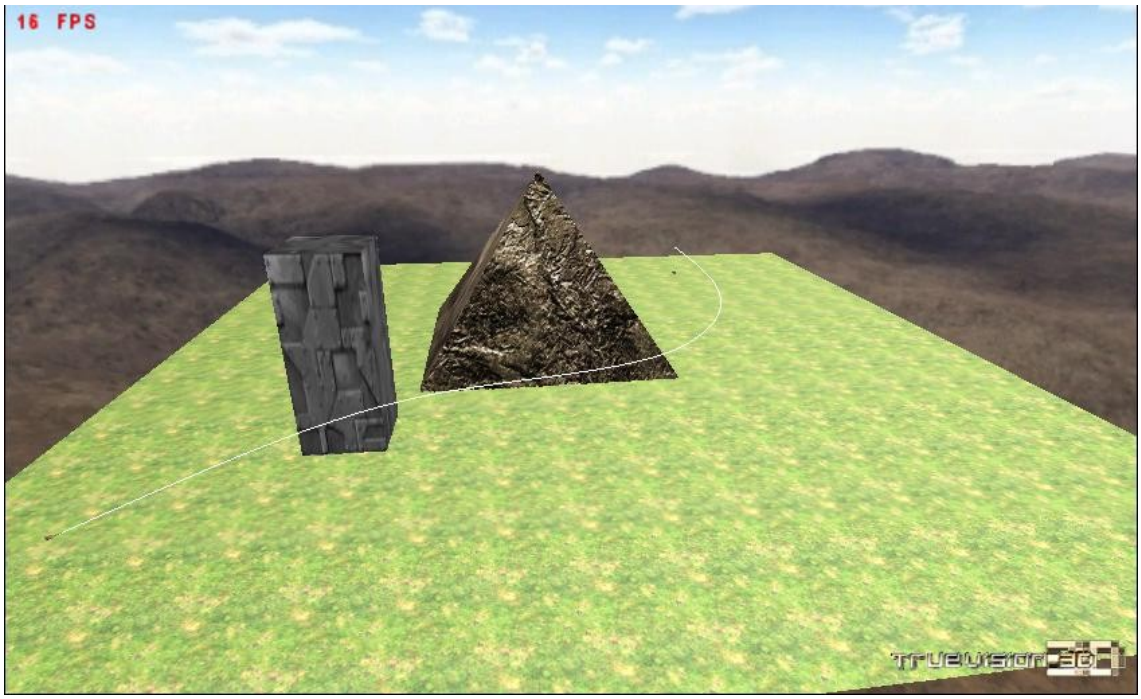
El resultado devuelto es el siguiente:



Como se observa, la trayectoria del avión (línea blanca) sortea perfectamente los obstáculos y llega al objetivo. Como hemos comentado antes, no hemos puesto que elimine el zigzag de la trayectoria, y vemos como en la parte final del objetivo, el avión hace un zigzag constante. En el siguiente ejemplo probaremos a eliminar el zigzag para ver como mejora la trayectoria del avión.

Ejemplo 1.2

Terreno simple con obstáculos (el mismo que antes), y esta vez eliminando el zigzag del avión. La visualización es la siguiente:

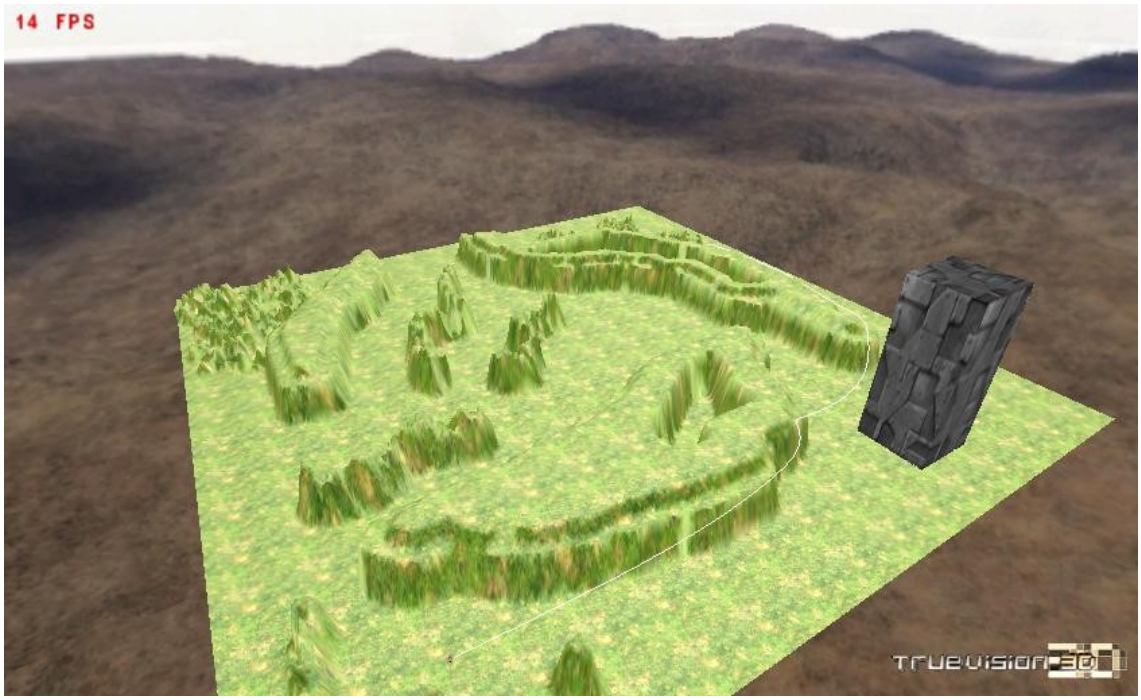


Como observamos, al elegir la opción de 'Corrección de Trayectorias' del menú, se elimina el zigzag del avión, y se obtiene una trayectoria más fina y suave.

Ejemplo 1.3

Ahora consideraremos un terreno más montañoso, para observar como el avión sortea las montañas, también pondremos algún obstáculo para hacerlo más interesante. Se seleccionará la opción de 'Corrección de Trayectorias' del menú para eliminar el zigzag. Este sería el aspecto del escenario inicial (las manchas blancas como sabemos representan montañas).

A continuación se muestra y comenta la solución.



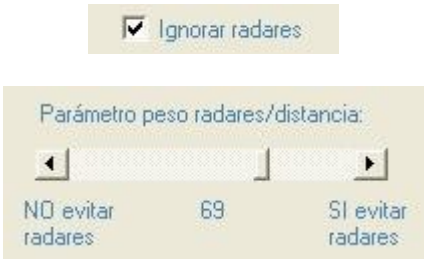
Como observamos la trayectoria del avión, representada por la línea blanca, sortea perfectamente las montañas y el obstáculo, y llega al objetivo.

4.2 EJEMPLOS DE TIPO 2

En este segundo apartado de ejemplos se mostrará la influencia de los radares en la trayectoria calculada por el algoritmo de pathfinding sin estar el objetivo dentro de la zona de influencia de los mismos.

El avión no tripulado se encontrará con radares por el camino. El algoritmo de cálculo de trayectorias intentará esquivar estos radares en la medida de lo posible para evitar ser derribado. Que esquive o no los radares que pueda encontrarse por el camino viene dado por la opción *Ignorar Radares*. Si se activa, el algoritmo de pathfinding no los toma en consideración y calcula la trayectoria como si no estuvieran. El parámetro de peso radares/distancia

(parámetro K, en adelante) influye en este caso sólo si la opción *Ignorar Radares* está desactivada. En este caso, cuanto mayor sea el valor de K, se intentarán evitar los radares de una manera más exhaustiva. Así, si ajustamos K a 100, aseguramos que se intentarán evitar los radares completamente en la medida de lo posible.



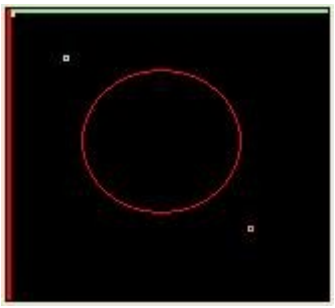
Importante: Los efectos de la variación del parámetro K sobre la trayectoria calculada dependen del tamaño del radar. Si K=0, la trayectoria ignora la influencia del radar, pero sí esquiva la zona de detección mínima del mismo, donde la detección y el disparo de un misil es seguro.

Así, si el radio de acción del radar es muy pequeño, sólo se notarán efectos del parámetro K en diferencias muy pequeñas y con valores bajos, esquivándose totalmente si lo incrementamos un poco (sin necesidad de hacer K=100). Si el radar es grande, el efecto es el contrario. Se necesitan variaciones más grandes para apreciar efectos y necesitará adoptar un valor más cercano a 100 para esquivar el radar completamente.

Ejemplo 2.1 (escenario *prueba21.xml*)

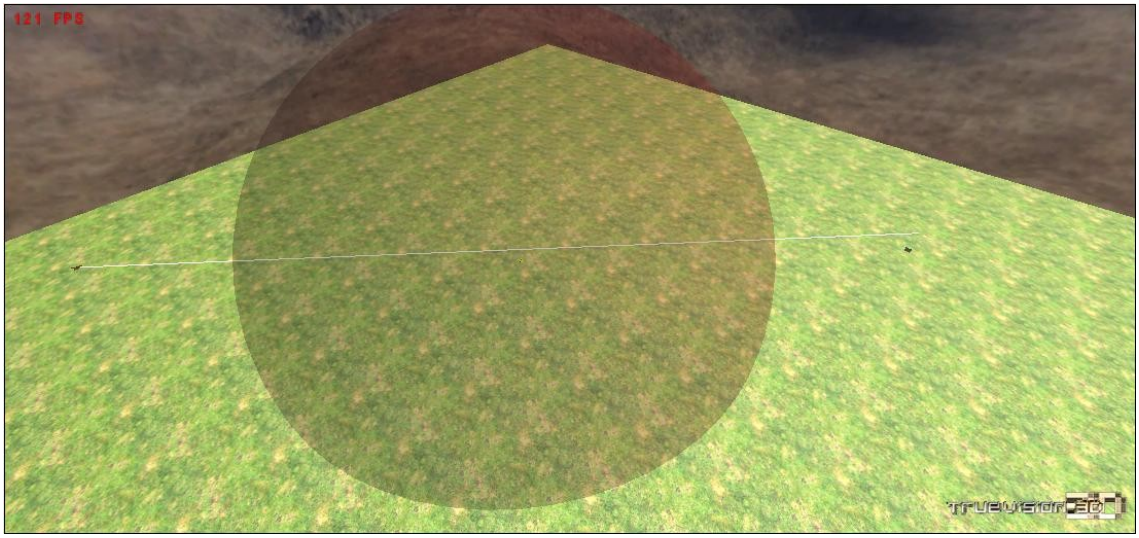
Colocaremos un radar por donde se pueda pensar que pueda pasar la trayectoria del avión. No ubicaremos ningún misil de momento, puesto que sólo queremos ver el comportamiento del cálculo de pathfinding.

La distribución del mapa será la siguiente:



Se tratará de una trayectoria directa al objetivo que se verá influenciada por el radar colocado (circunferencia roja).

Realizamos la prueba con *Ignorar Radares* activado y el resultado es el siguiente:

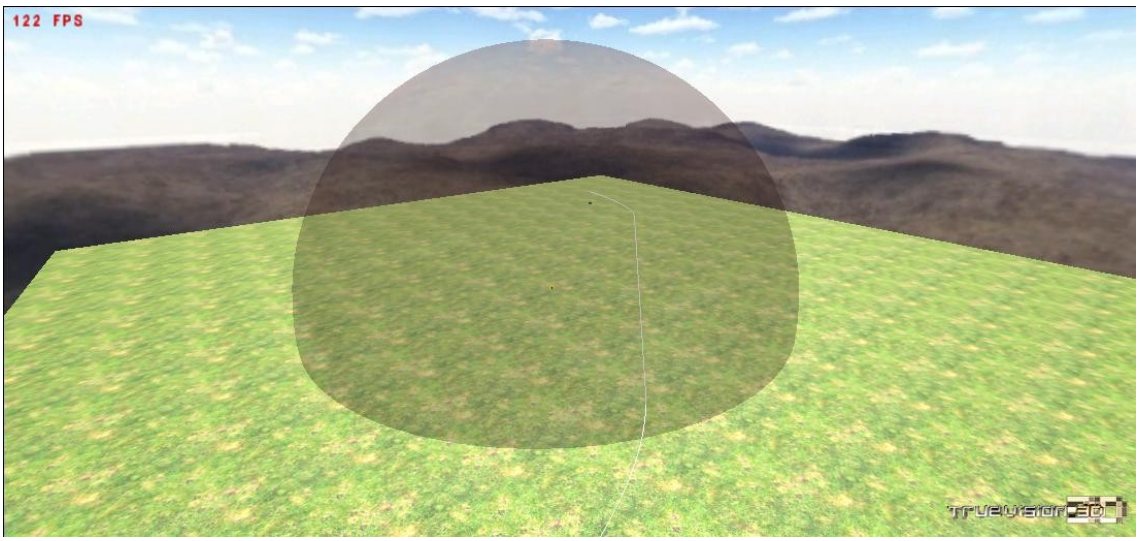


Podemos observar que, efectivamente, el algoritmo pathfinding ha obviado la existencia del radar y ha calculado una trayectoria recta.

Ejemplo 2.2 (escenario *prueba21.xml*)

Ahora variaremos tan solo la opción *Ignorar Radares* para ver cómo cambia el cálculo. Mantenemos K = 0.

El resultado es el siguiente:

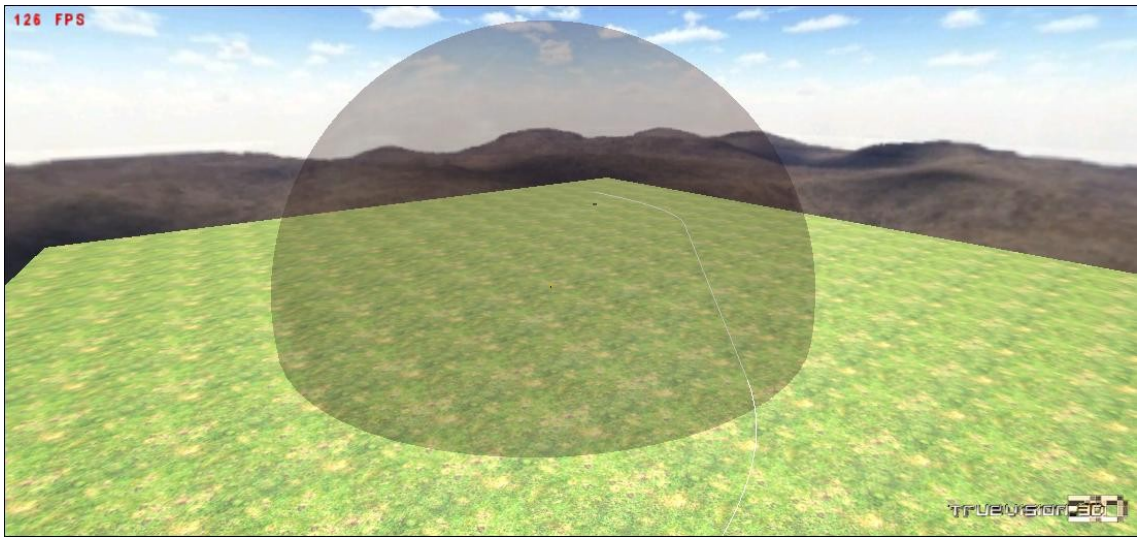


Vemos que la trayectoria ya no es recta e intenta alejarse del radar.

Ejemplo 2.3 (escenario *prueba21.xml*)

Se probará a aumentar el valor del parámetro K (K=6), manteniendo la opción *Ignorar Radares* desactivada. Se espera que pase por el radar de forma más alejada.

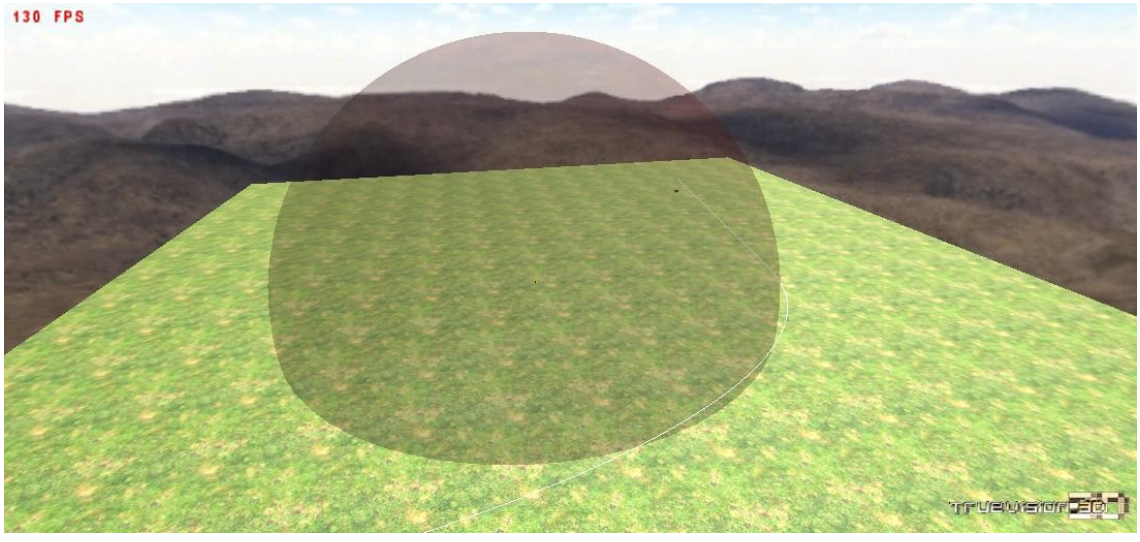
Veámoslo:



Comprobamos que, efectivamente, así es.

Ejemplo 2.4 (escenario *prueba21.xml*)

Misma prueba que la 2.3, pero con K = 100. En este caso se espera que se evite el radar completamente.

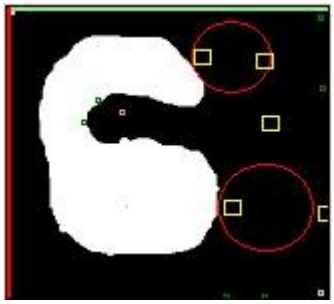


Vemos que lo bordea perfectamente.

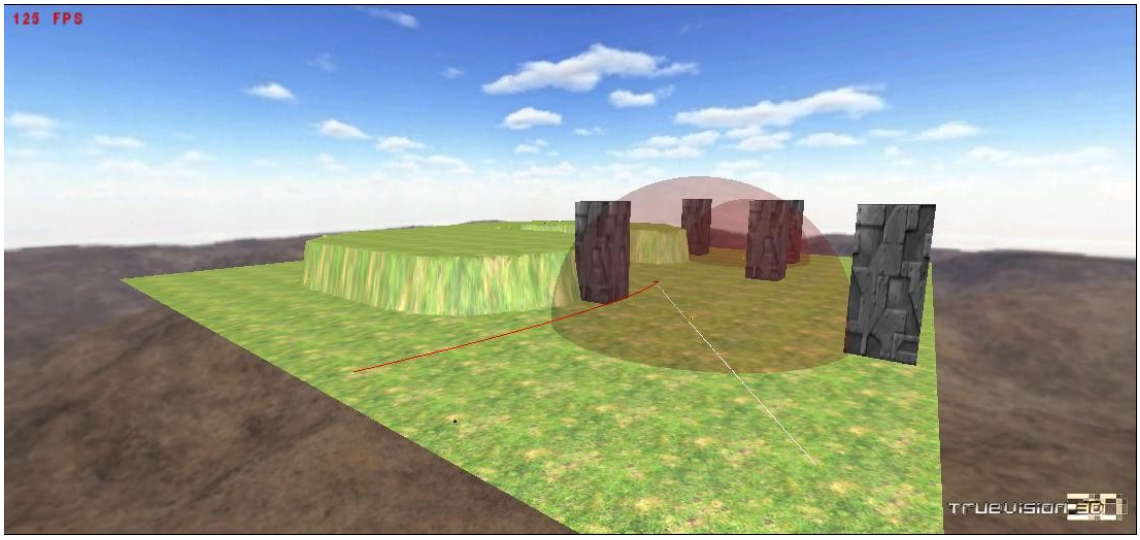
Ejemplo 2.5 (escenario *prueba22.xml*)

Finalmente, para esto tipo de ejemplos radares sin objetivo dentro de ningún área de influencia de radar, ejecutaremos un escenario con radares, montañas, obstáculos y misiles, analizando los resultados de la simulación obtenida.

El mapa a simular es el siguiente:

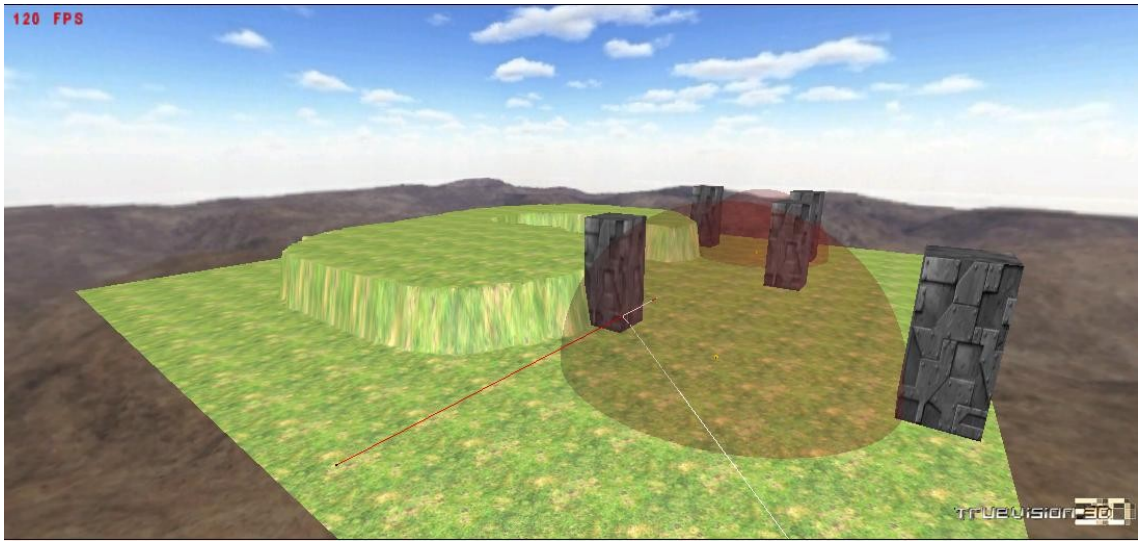


Habilitando la opción *Ignorar Radares* obtenemos el siguiente resultado:



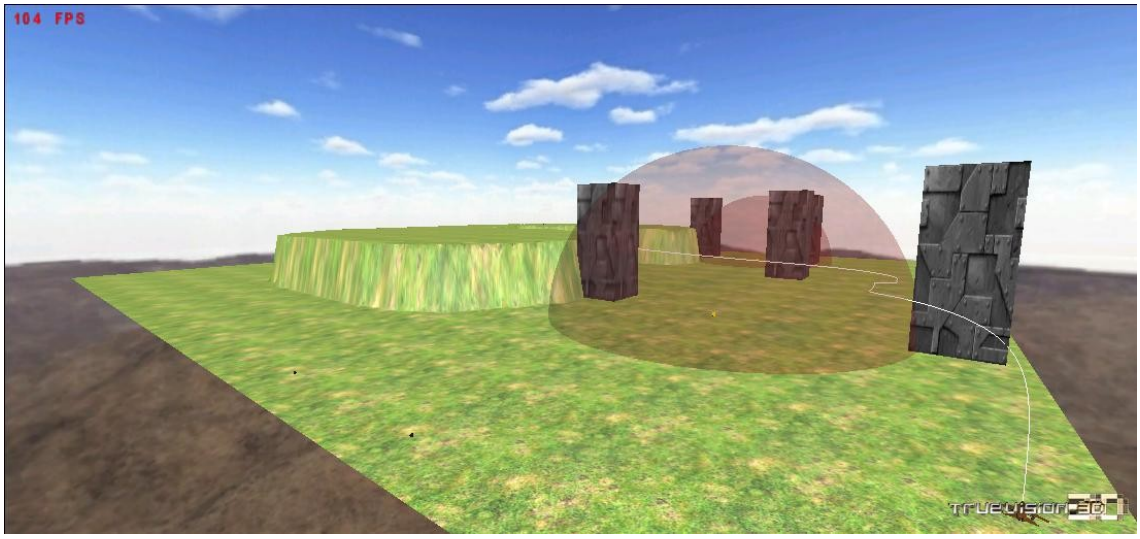
Observamos que el misil intercepta al avión siempre después de 50 simulaciones. Nunca llega al objetivo. El cálculo de trayectorias no tiene en cuenta la zona de influencia y queda expuesto al radar.

Si habilitamos la opción *Ignorar Radares* y mantenemos K=0, obtenemos:



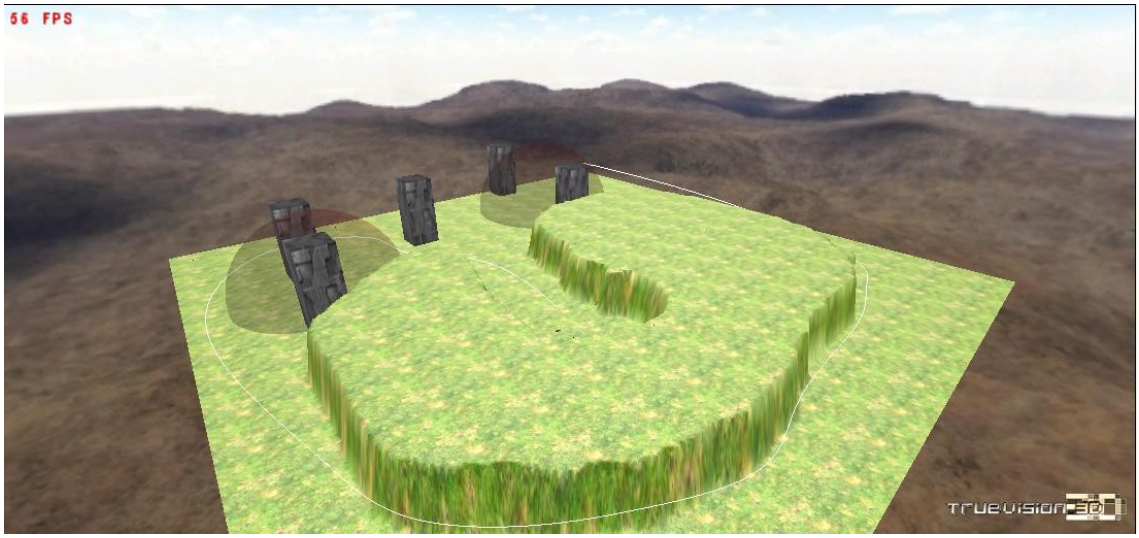
En este caso, el avión bordea la zona de detección mínima del radar (detección segura), pero no tiene en cuenta ángulo de incidencia. Consigue que se tarde más en disparar el radar, pero lo abate de todas formas en el 100% de las ocasiones.

Aumentemos el valor de K para ver si conseguimos reducir las posibilidades de detección:



Hemos fijado K=11. Toma una ruta que equilibra el compromiso entre distancia hasta el objetivo y riesgo de detección. Bordea el radar (por dentro de su zona de influencia) pero consiguiendo que no le dispare ningún misil. Llega hasta el objetivo.

Veamos, por último, qué trayectoria saca si establecemos K=100:



Como era de esperar, esquiva todas las zonas de influencia de los radares existentes, teniendo que bordear el montículo, ya que el primer radar le "bloquea" el paso al avión, junto con los dos bloques.

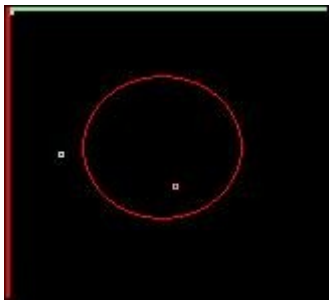
4.3 EJEMPLOS DE TIPO 3

En esta tanda de ejemplos se mostrará la influencia de los radares en la trayectoria calculada por el algoritmo de pathfinding sin estar el objetivo dentro de la zona de influencia de los mismos.

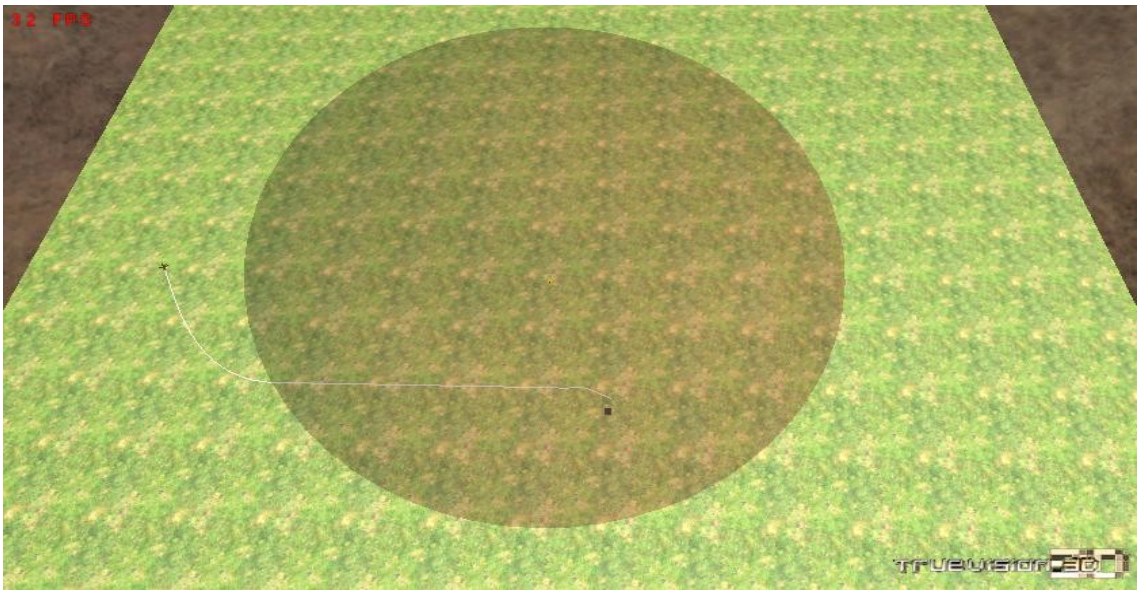
Ejemplo 3.1 (escenario *prueba31.xml*)

Simplemente tenemos un radar, el objetivo dentro de la zona de influencia del radar y un avión.

La distribución del mapa será la siguiente:



Realizamos la prueba con un valor 0 para el peso radares/distancia y el resultado es el siguiente:

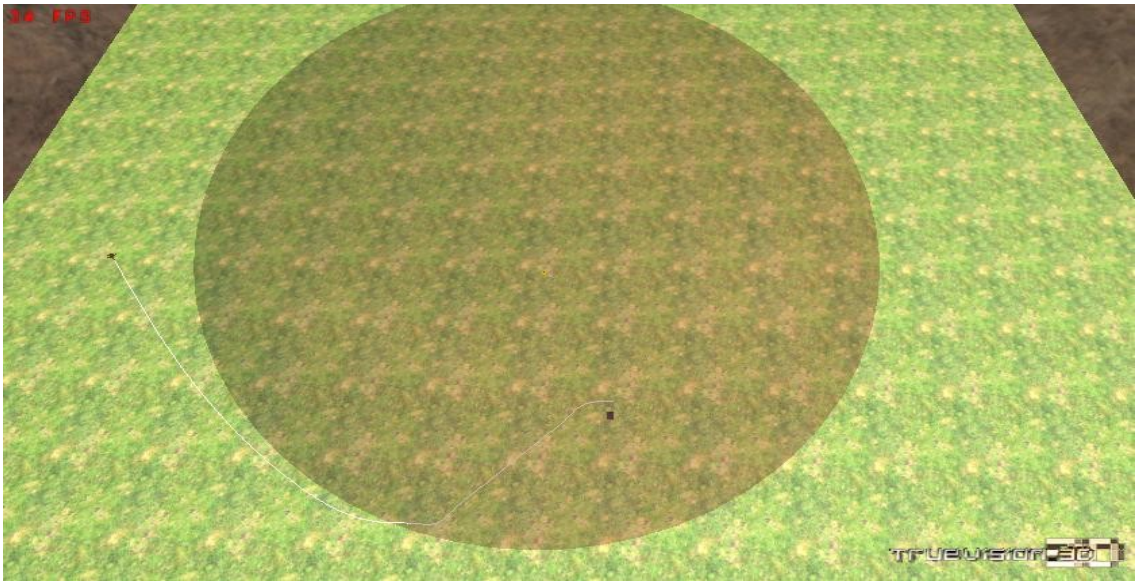


Se puede observar que el avión toma una trayectoria directa dentro de la zona de influencia, y está ofreciendo una orientación muy perpendicular con respecto a la línea del radar, con lo que la probabilidad de detección es grande.

Ejemplo 3.2 (escenario *prueba31.xml*)

Partiendo del escenario anterior, ahora vamos a dar un peso importante a los radares, con un valor del parámetro peso radares/distancia de 50.

El resultado es el siguiente:



Vemos como se ha buscado una trayectoria de forma que el avión entra en la zona de influencia de modo que muestra menos flanco a la línea de vista del radar en todo momento, con lo que las probabilidades de detección se reducen con respecto al caso anterior.

5 CONCLUSIONES

Hemos realizado una simulación de un vuelo en la que intervienen distintos elementos. Este tipo de simulaciones son muy interesantes por la cantidad de costes que se ahorran las empresas, ya que con ellas es posible observar el comportamiento que tendrían distintos móviles (en este caso aviones) en un escenario, interactuando con distintos objetos y acercándonos a lo que sería la realidad.

Una aplicación de esta índole "nunca se acaba", pues siempre es posible mejorar alguno de los procesos con nuevas o más precisas y complejas técnicas. Bajo esa perspectiva, esperamos haber conseguido en efecto una aplicación altamente modular, de forma que el proyecto quede abierto a posibles ampliaciones de una manera sencilla, a partir de lo que consideramos una aplicación que contempla todos los puntos importantes de la simulación requerida.

Hemos trabajado con una herramienta de mucho importancia en la actualidad, el Microsoft Visual Studio.NET y aprendido numerosas funcionalidades que hemos puesto en marcha en el prototipo y que van desde los Sevicios Web, hasta el empleo de un motor gráfico 3D, y pasando por problemas de eficiencia, de diseño de una aplicación de entidad y con necesidad de modularidad, de diversas técnicas para cálculo y mejora de trayectorias, y un largo etcétera.

6 MANUAL DE USUARIO

El propósito de este documento es ilustrar el funcionamiento del prototipo implementado. Para ello, se indica desde la instalación del mismo y todo lo necesario para que funcione, hasta el uso de los distintos interfaces, cómo cambiar distintos parámetros y las consecuencias de estos cambios. Los ejemplos de funcionamiento se ilustran paso a paso.

Índice de contenidos:

6.1 INSTALACIÓN Y PUESTA EN MARCHA

Para la ejecución del prototipo, es necesario lo siguiente:

- Sistema Operativo: Microsoft Windows 2000 o posterior

- DirectX 9.x.

- Microsoft .NET framework.

- Librerías de TrueVision 3D.

- Descargar los .zip del proyecto y descomprimir en una ruta cualquiera.

6.2 ESTRUCTURA DE CARPETAS

Una vez descargada la aplicación, la organización de directorios es la siguiente para la versión con el código fuente:

Los ejecutables se encuentran en /bin/release.
Para cada módulo del programa tenemos un directorio, que se corresponde con el proyecto de Visual Studio .NET. Estos son InterfazGeneral, Editor, ServicioWebPathfinding, ServicioWebSimulador y Visualizador.

Finalmente, se encuentra la carpeta Datos, que reúne todos los archivos necesarios para la ejecución de los distintos módulos:

- dentro de /Configuración encontramos el fichero config.xml de la aplicación.

- en /Escenarios se guardarán los .xml de salida del Editor, que serán a su vez los ficheros de entrada del módulo Cálculo de Trayectorias.

- en /Trayectorias se generarán los ficheros .xml de trayectorias, salidas del módulo Cálculo de Trayectorias. Estos ficheros serán a su vez la entrada del módulo de Simulación.

- en /Resultados se encuentran los .xml listos para cargar en el Visualizador, como salida del módulo Simulador.

- dentro de /Media, se encuentran recursos para la visualización, como texturas, modelos 3D, mapas de alturas, etcétera.

6.3 PARÁMETROS DEL SISTEMA

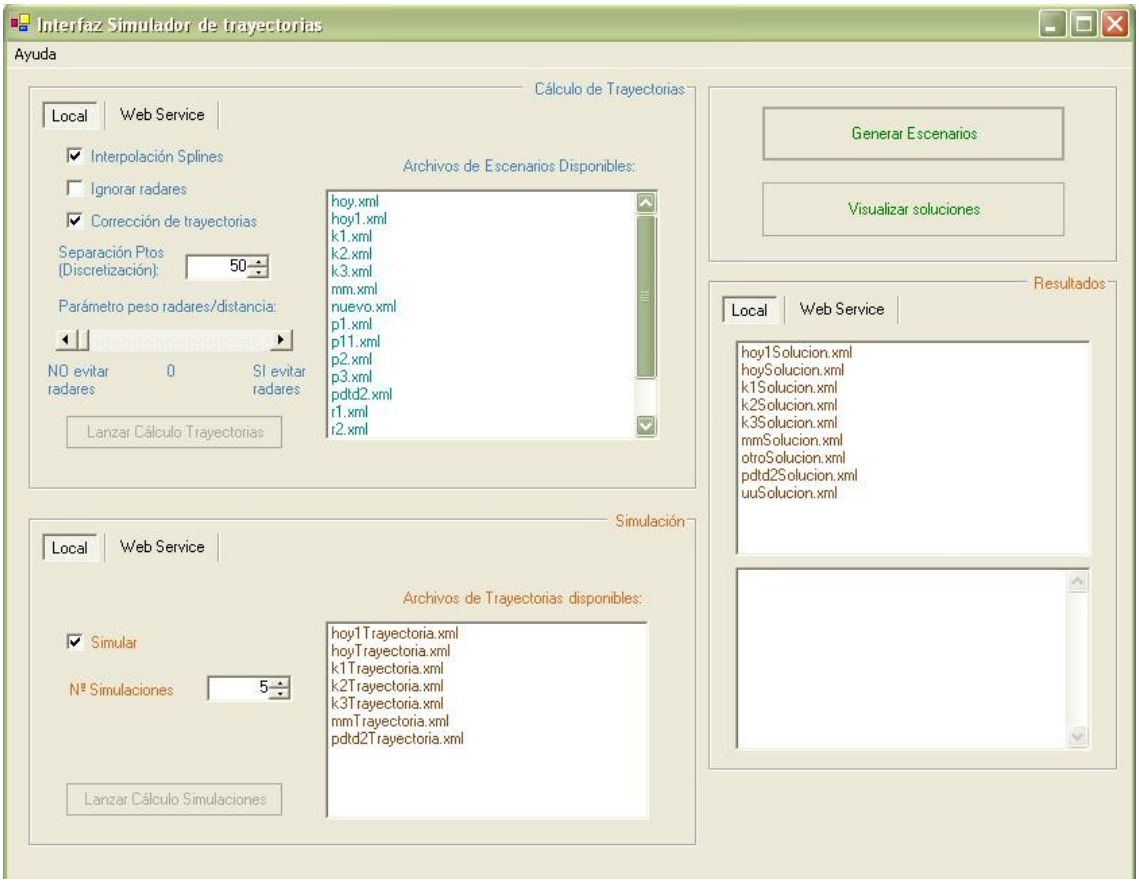
Los parámetros de la aplicación, usados por los distintos módulos, se encuentran guardados en el fichero /Datos/Configuración/config.xml. En este xml se encuentran los valores por defecto, pero no deben ser modificados aquí por el usuario.

Los parámetros que sí son modificables se encuentran en la interfaz de usuario general.

6.4 INTERFAZ GENERAL

Desde este formulario se puede acceder a todas las opciones del sistema, entre ellas modificar parámetros y hacer las llamadas a los distintos módulos, tanto en su versión local como en su versión de servicio web.

También se muestran los distintos ficheros xml (de escenario, trayectorias y resultados) disponibles.



6.4.1 LOCAL VS. WEB SERVICE

En primer lugar existe la opción de elegir el modo de ejecución, con la opción local, o la de Web Service (ver figura de abajo). Para la opción local, los archivos correspondientes al cálculo de la trayectoria, la interpolación, y la simulación deberán estar alojados en el ordenador del cliente (en concreto la aplicación trabaja automáticamente sobre la carpeta Datos). Para la opción de Web Service, estos archivos mencionados serán enviados al servidor para su procesamiento, para que otros se generen allí. Todo esto se realiza de forma transparente para el usuario.

6.4.2 CÁLCULO DE TRAYECTORIAS

Para este primer cálculo tenemos varios opciones para elegir:

1. Interpolación Splines: activa o desactiva la interpolación. La interpolación genera una serie de puntos intermedios en la trayectoria devuelta por el Astar, simulando un "suavizado" de la trayectoria.

2. Ignorar radares: indica si se desea que no se tengan en cuenta los radares a la hora de calcular la trayectoria de el/los avión/es. Desactivarlo provocará que el riesgo de detección sea máximo.

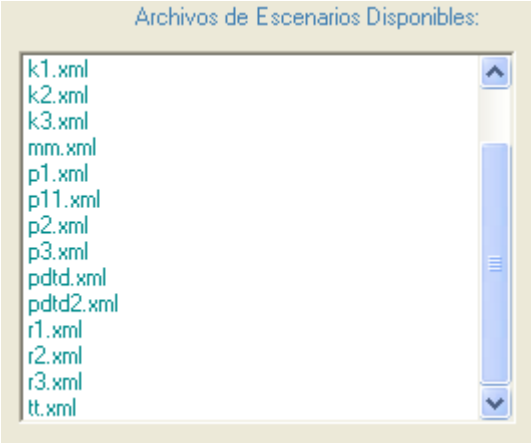
3. Corrección de trayectorias: opción de corrección de trayectoria, para mostrar una trayectoria más lineal (es decir, evitar posibles zigzag en las trayectorias de los aviones).

4. Separación de puntos (discretización): se puede elegir la separación de puntos (discretización) para la cuadrícula que se utilizará dentro del Astar. El valor por defecto es 50. Su disminución implica una mayor resolución en el cálculo de las trayectorias, penalizando a su vez el rendimiento.

5. Peso radares/distancia: barra deslizable que sirve para dar peso a los radares (es decir, la influencia que se desea de los radares en el cálculo de trayectorias). Por ejemplo, con un peso igual a 0, se ignorarán los radares completamente, si por el contrario ponemos un peso 100, los radares serán evitados por completo siempre que sea posible.

6. Por último está una lista de los escenarios disponibles, con dos posibilidades:

    a. Para la opción local, aparecerán los escenarios disponibles en la carpeta local /Datos/Escenarios. Por ejemplo:



    b. Para la opción de Web Service, a diferencia del caso anterior existe un botón refrescar para mostrar los escenarios disponibles en el servidor. Además se informa de la versión en el servidor.

6.4.3 CÁLCULO DE SIMULACIONES

Para este segundo proceso también existen unas intuitivas opciones

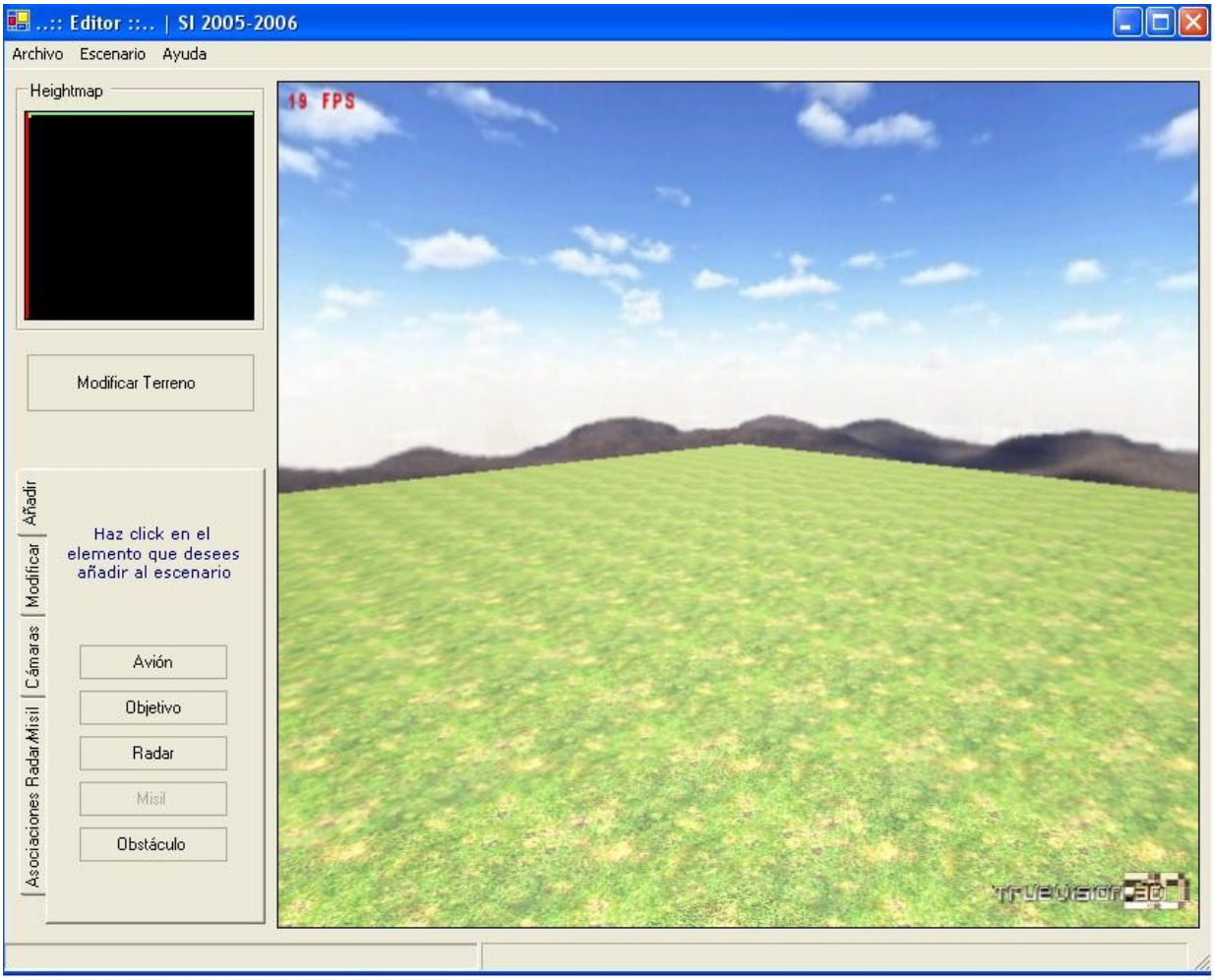
Si la opción de simular está marcada se puede elegir el número de simulaciones que se lanzarán. En el caso de no estar marcada, el programa simplemente calculará la trayectoria sin tener en cuenta los elementos de la simulación (radares, misiles).

Como en el caso anterior, existe la posibilidad de elegir el escenario a simular. Una vez se selecciona uno, se activa el botón "Simular".

NOTA: en caso de no seleccionarse un escenario o una trayectoria, los procesos preguntarán al usuario por el fichero a cargar.

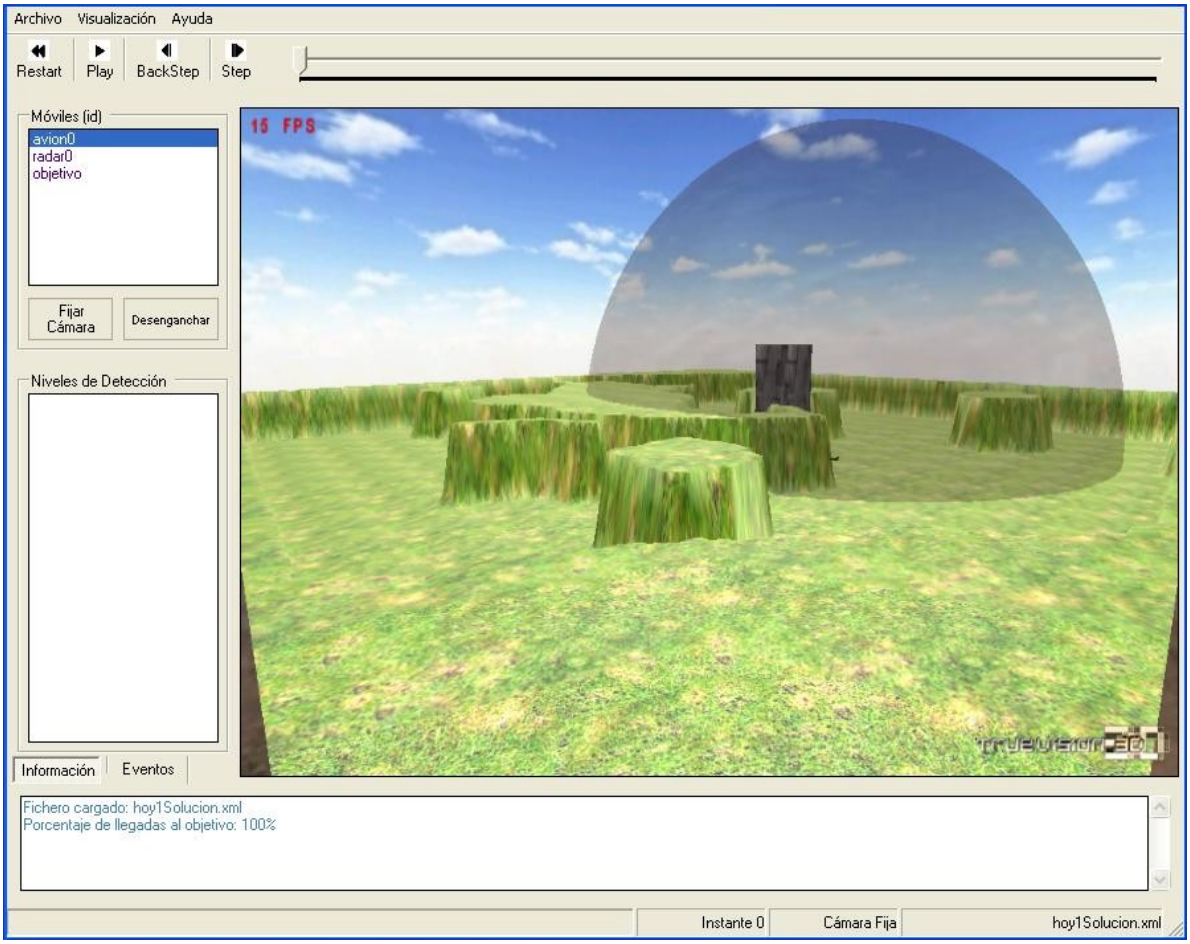
6.4.4 GENERAR ESCENARIOS

Con "Generar escenarios" se lanzará la siguiente aplicación cuyo funcionamiento está indicado más adelante.



6.4.5 VISUALIZAR ESCENARIOS

Con "Visualizar soluciones" se lanzará una aplicación de aspecto similar a la anterior pero con opciones y finalidad totalmente distintas como se verá más adelante.

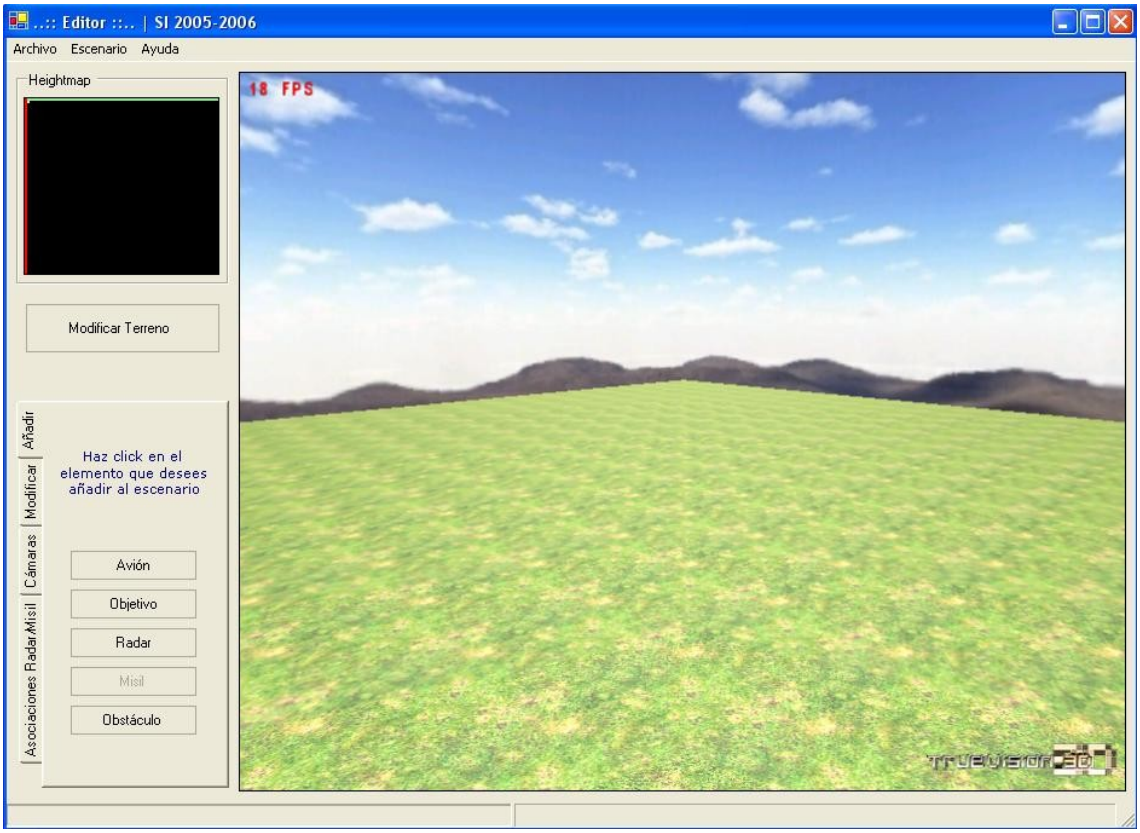


6.4.6 AYUDA

La ayuda consta de un enlace a una versión HTML del manual de usuario y del "Acerca de", con información de la versión y de los integrantes del proyecto.

6.5 INTERFAZ DEL EDITOR

Dentro de este formulario es posible crear un escenario, guardarlo o cargar uno ya existente. Para ello se podrá añadir los distintos elementos como el terreno y los radares, misiles, aviones u obstáculos predefinidos. Además podrán ser eliminados o modificados.



A continuación se detallan las opciones del menú principal:

6.5.1 ARCHIVO

1. Subir el escenario al servidor: una vez creado el escenario como queramos, tenemos la opción de subir el escenario al servidor, y que se quede allí guardado, para los cálculos que deseemos.

2. Salir: esta opción es utilizada para salir del menú actual.

6.5.2 ESCENARIO

1. Salvar: para guardar el escenario creado.

2. Cargar: para cargar un escenario creado anteriormente.

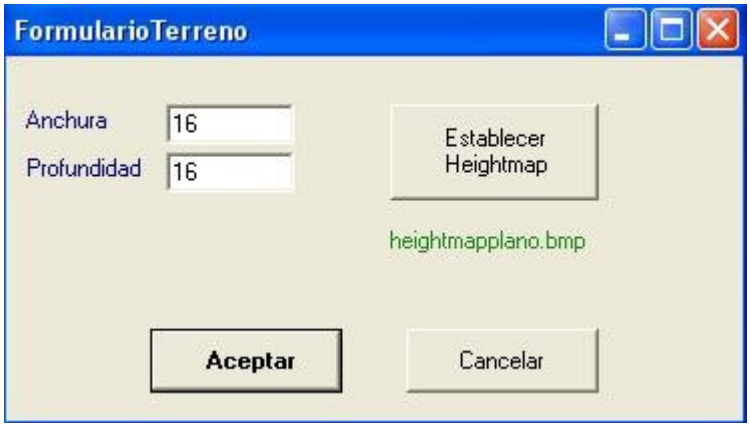
6.5.3 AYUDA

1. Ayuda: abre la versión HTML del manual de usuario.

2. Acerca de: fecha, versión y autores del proyecto.

6.5.4 BOTÓN MODIFICAR TERRENO

Al elegir esta opción se obtiene un formulario como el siguiente:



Con Anchura y Profundidad, se pueden especificar las dimensiones para el terreno o escenario. Estas dimensiones están medidas en píxeles, y se multiplicarán por 256. Por ejemplo, si se indica 16, en realidad el tamaño será 16*256=4096 px. Por otro lado, ese es el tamaño por defecto.

Con el botón 'Establecer Heightmap', se podrá elegir entre los archivos de mapa de altura. Estos deben ser archivos de imagen, normalmente en escala de grises. El programa los interpreta cargando montañas cuya altura depende del nivel del tono de cada píxel.

Pestañas para añadir, modificar elementos al terreno o asociar misiles a los radares:

6.5.5 AÑADIR

Opción para añadir los distintos elementos de los que disponemos: aviones, el objetivo, radares, obstáculos, misiles (siempre que se haya añadido un radar primero, ya que los misiles deben ser asociados a un radar).
Añadir un avión:

La altura es constante, y en las posiciones X y Z se introducirá la posición deseada. También es posible hacerlo directamente pinchando sobre algún punto del rectángulo de 2 dimensiones que representa el escenario. Las coordenadas en ese caso se actualizarán automáticamente.

Si hay otros elementos, aparecerán dibujados en el escenario de dos dimensiones.

Añadir un objetivo:

Caso similar al anterior. Está vez se puede observar el punto que indica la posición de un avión añadido en el mapa. Todos los elementos añadidos aparecerán en el mapa para facilitar al usuario la creación de escenarios.

En este caso, el avión es representado por un punto azul, mientras que los objetivos por un punto rojo.

Añadir un radar:

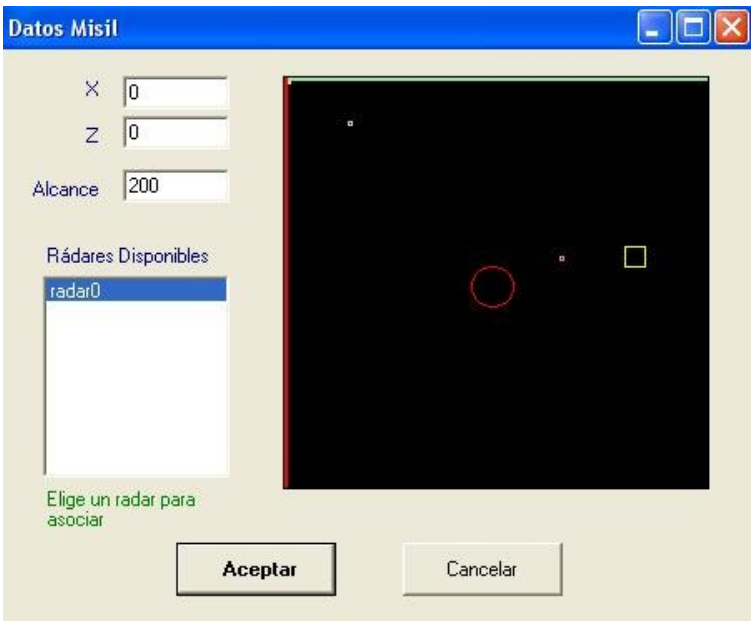
El radio de acción del radar es el radio a partir del cual se pueden detectar aviones (se muestra sobre el fondo negro con un círculo rojo). La distancia mínima de detección representa la distancia a partir de la cual los aviones son detectados con una probabilidad del 100%.
Por último, el Timeout indica el número de instantes en que un avión detectado permanece en seguimiento. Cada vez que se cumple un Timeout se vuelve a calcular la detección del avión por parte del radar.

Añadir un obstáculo:

Al igual que anteriormente, se introducen o bien numéricamente o bien pinchando sobre el fondo negro. Se puede elegir entre dos tipos de obstáculos predefinidos, en forma de pirámide o de bloque. Estos se representan sobre el fondo negro con un cuadrado amarillo.

Añadir un misil:

Para añadir un misil, es necesario haber añadido un radar antes.



En este caso el alcance indica el número de píxeles del escenario que puede recorrer el misil.

Abajo aparece el cuadro Radares Disponibles para asociar los misiles a un radar.

6.5.6 MODIFICAR

Opción para modificar alguno de los elementos creados. Las modificaciones se hacen sobre las mismas interfaces explicadas arriba.

6.5.7 CÁMARAS

1. Libre: la cámara será libre y se manejará mediante el teclado o mediante el ratón. Para ello, se debe mantener pulsado el ratón y luego moverlo libremente como desee.

   Opciones de enfoque de la cámara libre según la tecla pulsada:
   a. Tecla I(Zoom): Acercar
   b. Tecla J: Hacia la izquierda
   c. Tecla K(Zoom): Alejar
   d. Tecla L: Hacia la derecha
   e. Tecla A: Hacia arriba
   f. Tecla Z: Hacia abajo

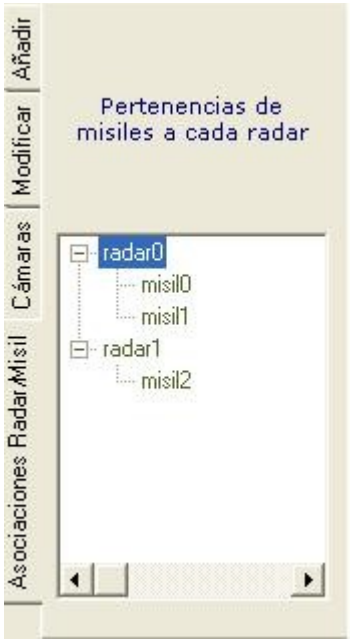
2. Fija: la cámara estará fija, por ejemplo siguiendo un elemento como un avión.

   Opciones de enfoque de la cámara fija según la tecla pulsada:
   a. Tecla I: Hacia arriba
   b. Tecla J: Hacia la izquierda
   c. Tecla K: Hacia abajo
   d. Tecla L: Hacia la derecha
   e. Tecla A(Zoom): Acercar
   f. Tecla Z (Zoom): Alejar

En la interfaz se indican las funciones de las teclas para los movimientos de la cámara.

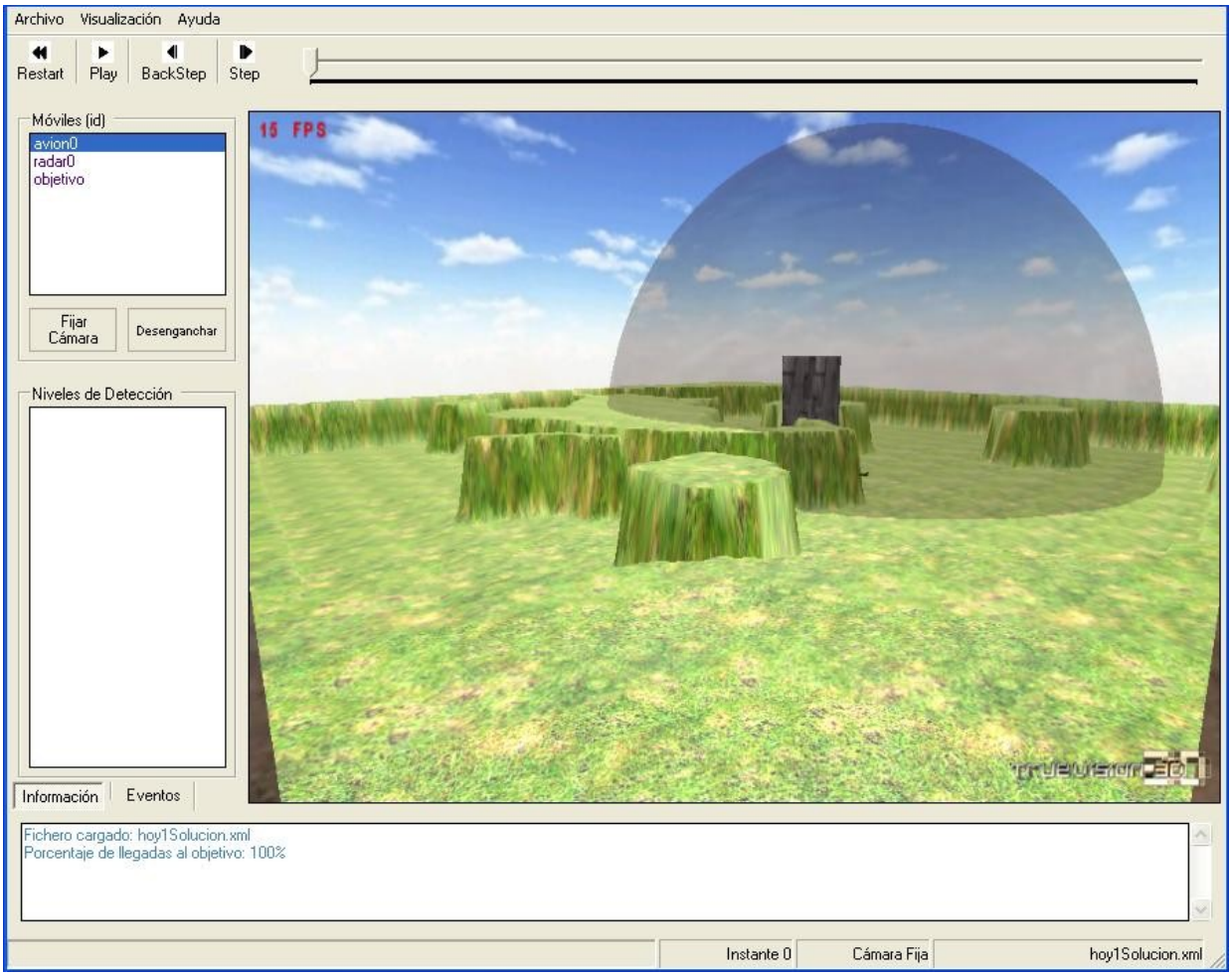
6.5.8 ASOCIACIONES RADAR MISIL

Opción para consultar como están asociados los misiles a los radares. Como se puede apreciar en la imagen siguiente, se trata de un menú jerárquico donde pulsando en el '+' se despliegan las asociaciones.



6.6 INTERFAZ DEL VISUALIZADOR

Dentro de esta aplicación, podremos ver los resultados finales (ficheros XML de solución). Además parte de poder reproducir las trayectorias de los objetos móviles, se mostrarán los eventos que se han producido en cada instante de tiempo y los porcentajes de éxitos o detecciones (estadísticas de las simulaciones realizadas).



Se dispone de las siguientes opciones de menú:

6.6.1 ARCHIVO

1. Descargar mapa: con esta opción puede descargarse alguno de los mapas resultados de una simulación cuando están alojados en el servidor.

2. Mapa por defecto: para cargar un mapa vacío.

3. Cargar mapa: para cargar uno de los ficheros solución. Se podrá elegir entre los que se encuentran en la carpeta /Datos/Resultados.

4. Salir

6.6.2 VISUALIZACIÓN

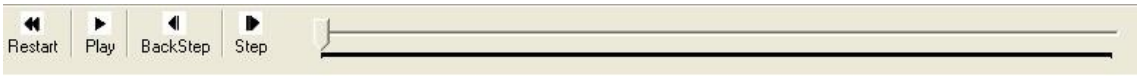
1. Mostrar trayectorias: función para mostrar las líneas que representarán las trayectorias de los elementos móviles.

2. Cámaras: se desplegará un submenú con dos opciones, la de cámara fija y cámara libre. Su funcionamiento está explicado en la sección 6.5.7 Cámaras dentro de 6.5 Editor.

6.6.3 AYUDA

Para acceder a la ayuda en versión HTML y al Acerca de.

6.6.4 FUNCIONES DE REPRODUCCIÓN



Se trata de unos controles de reproducción estándar, comúnmente utilizados en muchas aplicaciones.

Para pausar la ejecución, se debe pulsar el botón "Play" una vez arrancado.

"Step" servirá para el modo "paso a paso" y por último es importante destacar que es posible utilizar la barra de desplazamiento manualmente para situar la reproducción en un punto determinado.

6.6.5 VENTANAS DE TEXTO

Proporcionan diversa información:

- Móviles(id): muestra los móviles que aparecen en la simulación cargada junto con su identificador. Una interesante función es la de seleccionar uno de los móviles y pulsar sobre "Fijar Cámara" para que ésta enfoque directamente al móvil seleccionado. Con "Desenganchar" se logra que la cámara deje de seguir al avión, aunque seguirá enfocándolo.

- Niveles de detección: muestra los distintos instantes por los que va pasando el resultado de la visualización. Además también se refleja si determinado radar (ej. radar0) ha detectado a determinado avión (ej. avion0) y el porcentaje de posibilidad de detección en cada posición del avión.

6.6.6 PANELES DE INFORMACIÓN

Son dos y están situados en la parte inferior:

- Información: proporciona más datos como son el nombre del fichero resultado cargado, y el porcentaje de éxito de llegada al objetivo que se ha obtenido en las distintas simulaciones calculadas.

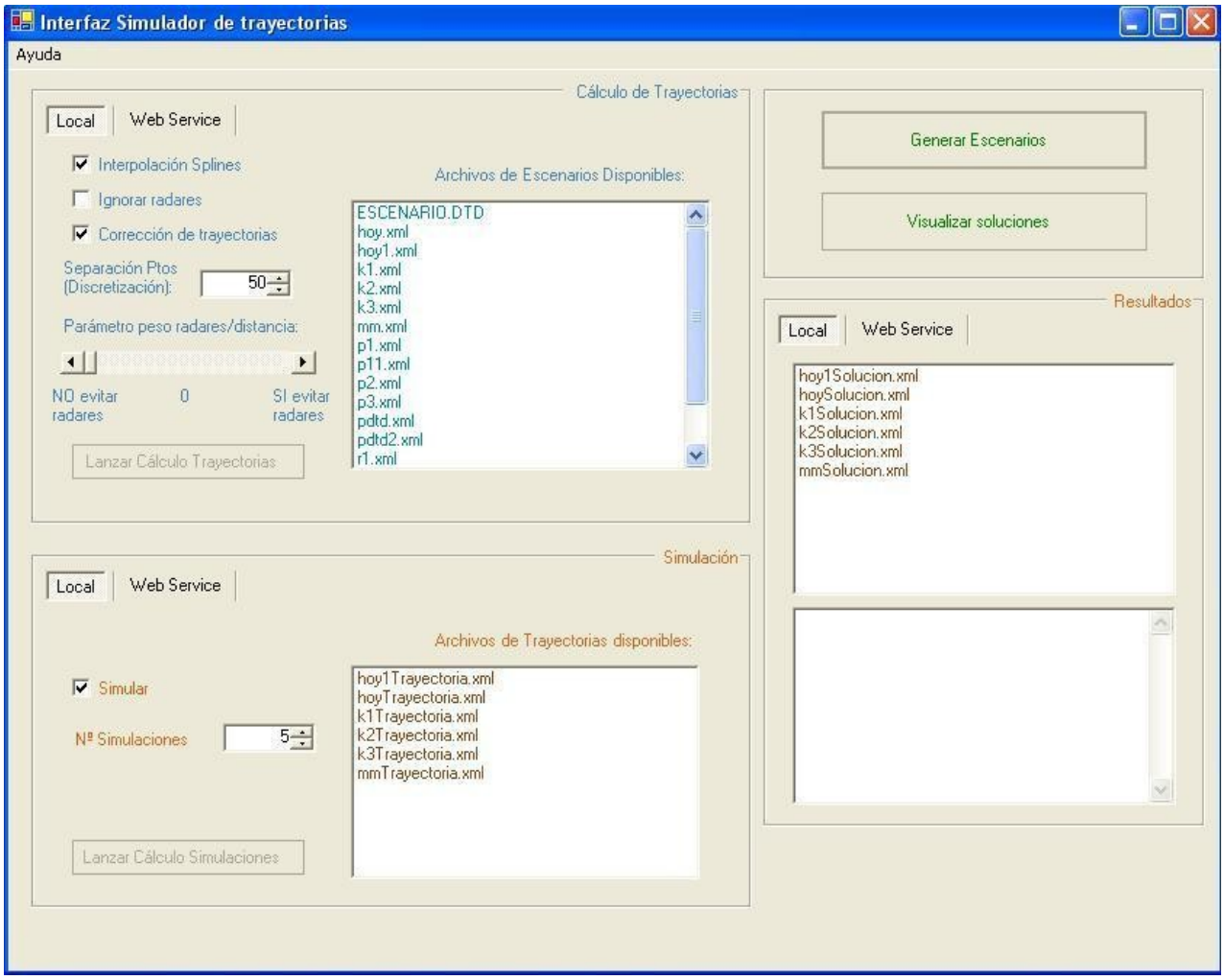
- Eventos: muestra los diversos eventos que se suceden en la reproducción del resultad, resultando de gran ayuda para el seguimiento de lo obtenido. Los eventos que se muestran son de diversos tipos:

    - Avión detectado por radar
    - Avión deja de ser detectado por radar
    - Avión es seguido por cierto radar
    - Avión destruido por cierto misil
    - Avión choca
    - Avión llega al objetivo
    - Misil disparado
    - Misil en seguimiento
    - Misil deja de estar en seguimiento
    - Misil choca
    - Misil destruido por superar alcance

6.7 EJEMPLOS DE EJECUCIÓN PASO A PASO

6.7.1 VERSIÓN LOCAL

Ejecutando el programa aparece la interfaz general:

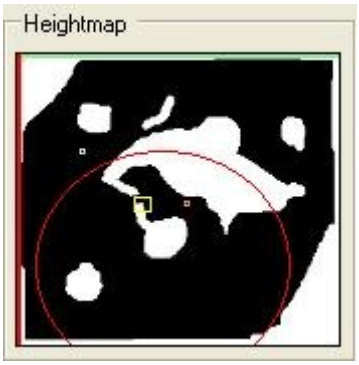


1. El primer paso es la generación de escenarios, que se lanza con "Generar Escenarios". Todas las opciones para ello están explicadas en 6.5 Interfaz del Editor.

   Una vez generado el escenario (es importante generar un objetivo y al menos un avión), éste debe ser guardado con el nombre deseado, por ejemplo "hoy1.xml". Una vez salvado, aparecerá en la lista "Archivos de Escenarios Disponibles" para trabajar con él.

2. Se continúa con el cálculo de trayectorias. Se puede dejar por ejemplo las opciones por defecto y modificar el parámetro peso radares/distancia,

poniéndolo al máximo (100) con lo que se indicará que se eviten los radares siempre que sea posible. En la lista de Archivos de Escenarios disponible, elegir "hoy1.xml" u otro. "hoy1.xml" tiene este aspecto (visible desde el editor):



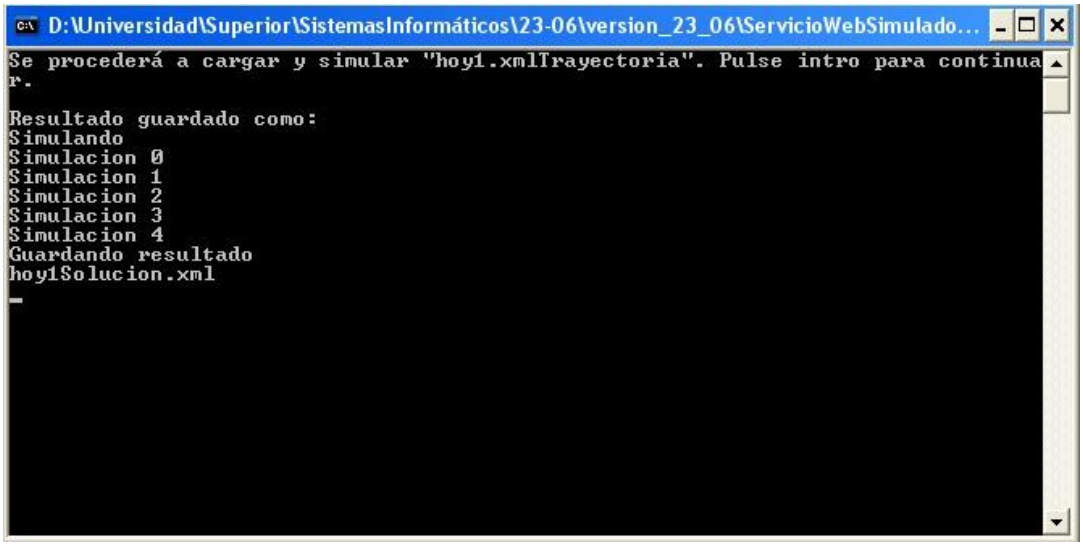
Como se explica en el apartado del editor, el punto azul corresponde al avión, el punto rojo al objetivo, el círculo rojo es el radio de detección del radar, el cuadrado amarillo en este caso es un bloque, y las zonas blancas se corresponden con montañas.

Una vez seleccionado el escenario, se puede lanzar el proceso con "Lanzar Cálculo de Trayectorias". La interfaz de línea de comandos nos informa del proceso de ejecución y el usuario sólo debe de pulsar <Intro> al principio y al final. Si no es seleccionado ningún escenario de la lista, el interfaz pedirá que se escriba manualmente.

Al final se informa al usuario del archivo XML de trayectorias que ha generado. En este caso, es "hoy1Trayectoria.xml". Este nuevo archivo aparecerá en la lista "Archivos de Trayectorias disponibles".

3. Se pasa al proceso de simulación. Dejando las opciones predefinidas, se selecciona el archivo de trayectorias deseado, por ejemplo "hoy1Trayectoria.xml" y se lanza el proceso mediante "Lanzar Cálculo Simulación".

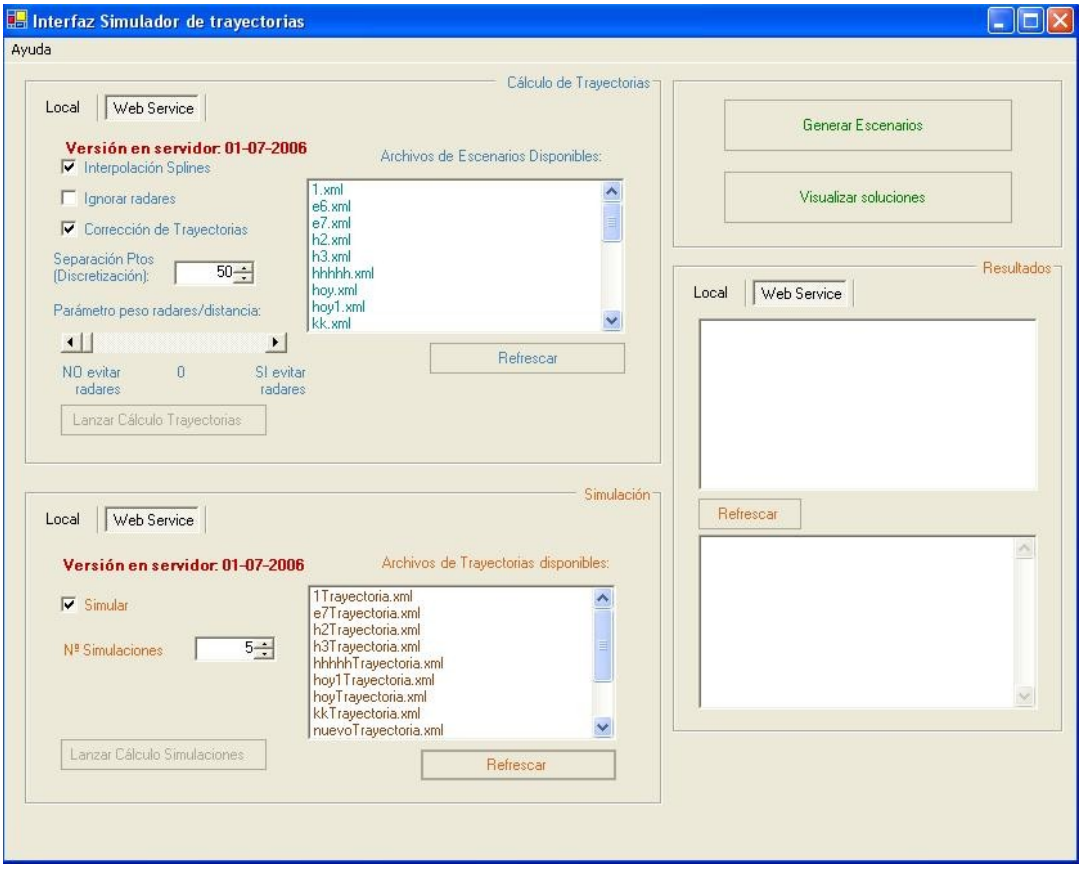
   Una vez más se informara del proceso de ejecución mediante consola y al final aparece el archivo XML de resultados generado. En el caso del ejemplo, "hoy1Solucion.xml". Este archivo aparecerá a su vez en la lista "Resultados" de la derecha del formulario.



4. Por último ya se dispondría de la solución para su visualizado, mediante el botón "Visualizar Soluciones". Desde el visualizador podremos cargar el archivo de soluciones o resultados deseado. El funcionamiento del visualizador está explicado en detalle en 6.6 Interfaz del Visualizador.

6.7.2 VERSIÓN SERVICIO WEB

En este ejemplo se seleccionaran las opciones “Web Service”.



Este es un ejemplo prácticamente similar al anterior, con lo que los pasos son los mismos.

Es importante destacar que no obtendremos información de ejecución por consola, ya que los procesos de cálculo y simulación se ejecutarán en el servidor.

La otra diferencia está en el botón "Refrescar" que debe ser usado cuando sea necesario para obtener los ficheros del servidor, ya que aquí no hay refresco automático de las listas.

7 ANEXOS

7.1 ANEXO1: TRUEVISION 3D

www.truevision3d.com

Para esta aplicación se ha decidido utilizar un motor gráfico 3D basado en DirectX para simplificar la tarea de representación tridimensional, ya que no era el propósito fundamental de la misma.

El motor gráfico elegido por su sencillez de uso, claridad y adecuación a lo requerido es el TrueVision 3D v6.2.

Es utilizado en dos partes de la aplicación; concretamente en el módulo de edición y el de visualización.

7.1.1 ESTRUCTURA DE USO

El uso de este motor gráfico se puede consultar en la documentación del mismo, pero se hará una breve descripción.

La idea fundamental en la que se basa es en que no existe el movimiento, sino los "fotogramas". Es por ello que se hace uso de un bucle principal donde se realizarán los cambios a los objetos involucrados en la escena para simular ese movimiento.

La estructura de un programa básico utilizando el motor gráfico es el siguiente:

Declaración de variables:

```
TVEngine TV;
bool DoLoop = true;
```

TV será la instanciación del motor, propiamente dicho. Y DoLoop no es más que una variable booleana que controlará la continuidad del bucle principal. En cuanto se haga falso, se detendrá el programa.

Inicialización:

```
TV = new TVEngine();
TV.Init3DWindowedMode(this.pictureBox1.Handle.ToInt32(), true);
TV.DisplayFPS = true;
```

Con la primera instrucción instanciamos el motor. Con la segunda inicializamos en modo ventana, asignando el control a un objeto PictureBox. Y con la tercera instrucción le decimos al motor que nos enseñe en todo momento los FPS (imágenes por segundo).

En este ejemplo no se hace, pero aquí se deben inicializar también parámetros de iluminación, texturas, modelos, terreno, el skybox, etc.

```
Main_Loop();
```

Esta es la llamada al bucle principal.

Bucle:

```
while(DoLoop == true)
{
      TV.Clear(false);
      //Aquí irían las modificaciones oportunas
      TV.RenderToScreen();
      System.Windows.Forms.Application.DoEvents();
}
Main_Quit();
```

Mientras la variable booleana sea verdadera, se limpia la pantalla, se realizan las modificaciones oportunas, se renderiza todo en la pantalla y se permite que se ejecuten los eventos de la aplicación.

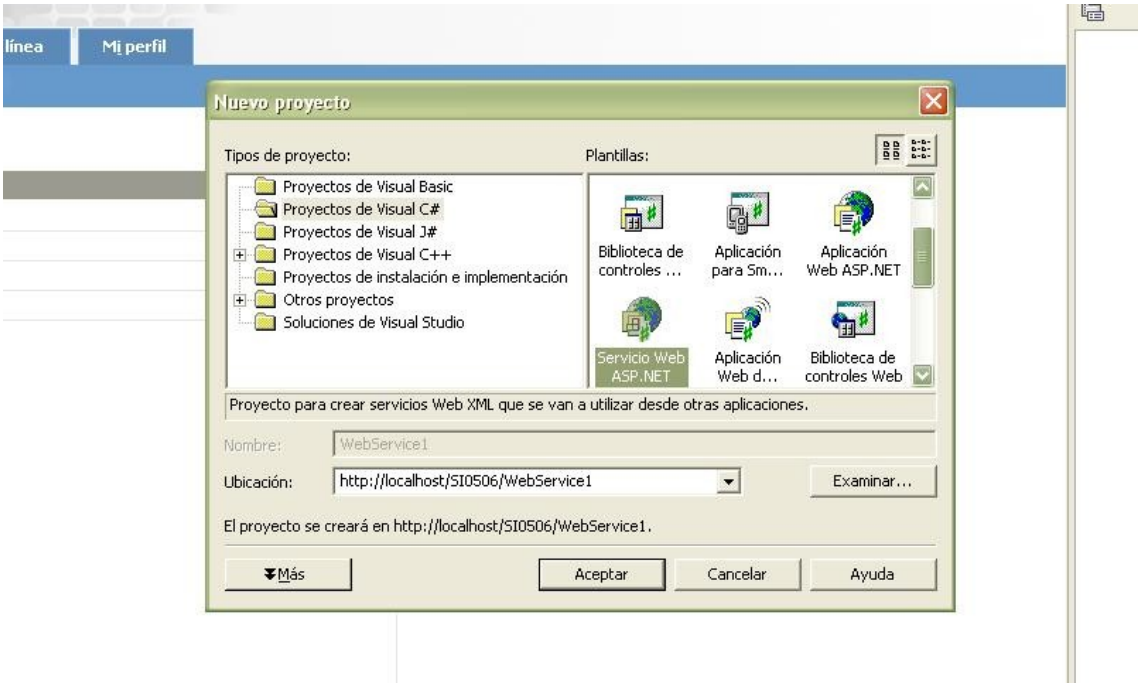
7.2 ANEXO 2: CONEXIÓN WEB SERVICE

7.2.1 CREACIÓN DE UN WEB SERVICE

Para crear un servicio web con Visual Studio .NET 2003 se va a describir un ejemplo paso a paso que servirá para ilustrar cómo se ha desarrollado este apartado.

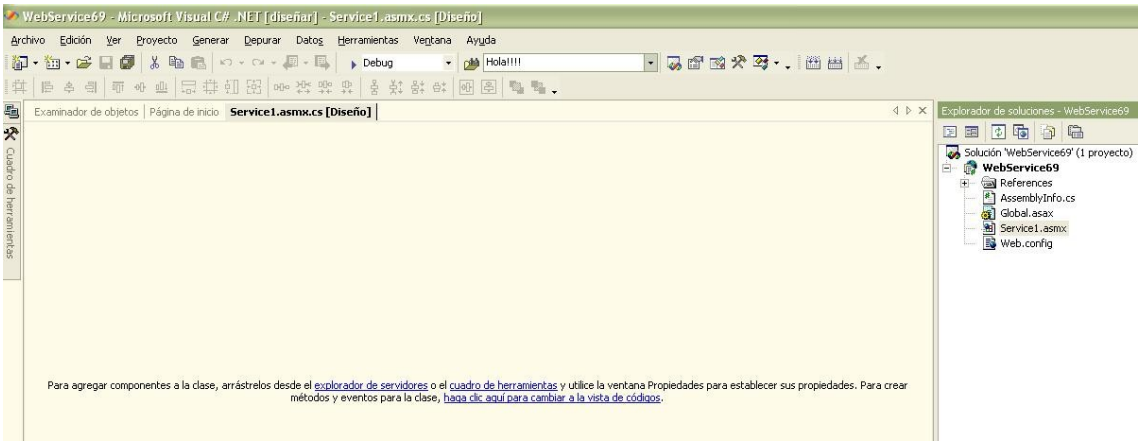
Los pasos son los siguientes:

1. Crear nuevo proyecto "Servicio Web ASP.NET".



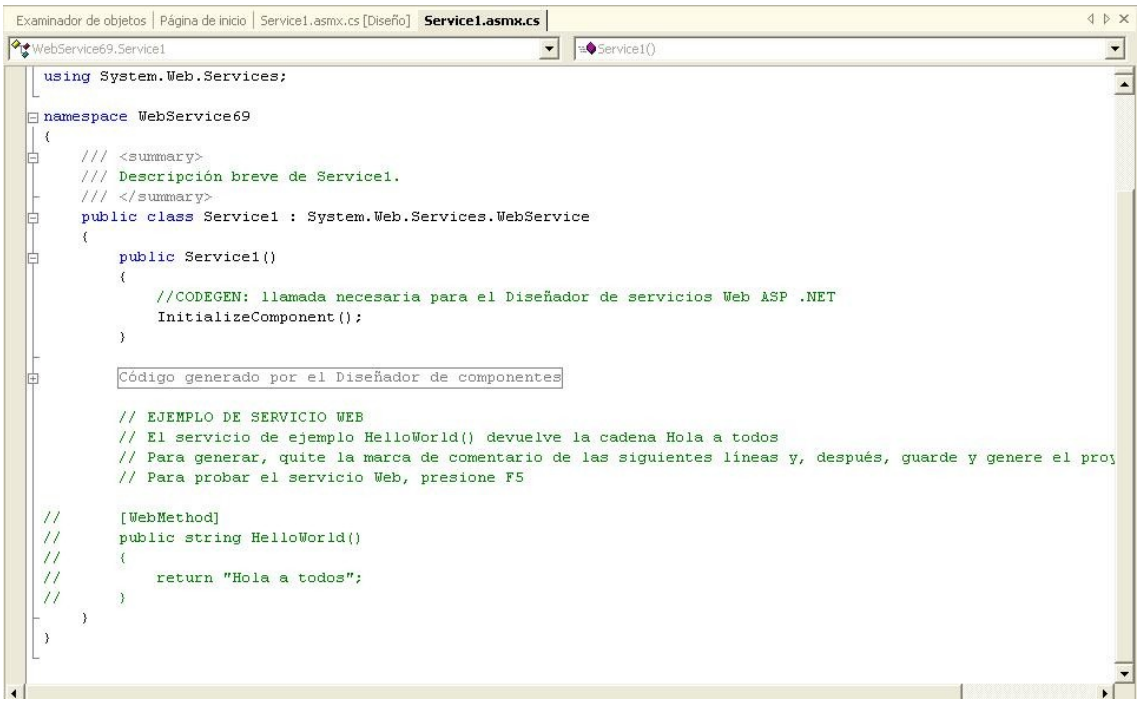
En la ubicación se debe especificar la URL del equipo que va a albergar el servicio. Este equipo ha de tener el servidor IIS de Microsoft instalado. Se recomienda instalarlo en http://localhost/ (el mismo equipo desde donde se está creando – dirección IP: 127.0.0.1) y una vez hecho y probado replicarlo en el servidor donde quedará ubicado. Esto es recomendable por cuestiones de permisos. En el equipo propio no hay ningún tipo de restricción si se está utilizando una cuenta de administrador y no dará ningún quebradero de cabeza.

2. Abrir el fichero *.asmx* que contendrá nuestros servicios.



Para ello pinchar en "haga clic aquí para cambiar a la vista de códigos".

En este punto tenemos la siguiente hoja abierta (*Service1.asmx.cs*):



Nuestro servicio se llama *Service1* y es una clase que hereda de *System.Web.Services.WebService*.

Si nos fijamos hay un método comentado que trae por defecto. Es un método normal y corriente, pero viene precedido de la etiqueta *[WebMethod]*. Esto causará que todo método que esté precedido por esta etiqueta esté disponible a través del servicio web (a modo de "fachada"). Por supuesto, se pueden crear métodos auxiliares, se pueden incorporar clases al proyecto y/o bibliotecas DLL.

Así, el método comentado de la captura estará disponible a través de nuestro servicio web y nos devolverá un "Hola a todos".

Importante: los datos transmitidos a través de estos servicios mediante el protocolo SOAP tienen limitaciones. Se pueden enviar objetos de tipos básicos (int, float, string, bool, …), estructuras que contengan objetos de tipo básico (ArrayList de bool's, etc.) y, puesto que la comunicación está basada en XML, cualquier objeto de tipo XML como por ejemplo, System.Xml.XmlDocument.

A la clase *Service1* se le puede anteponer la etiqueta *[WebService]* y se le pueden añadir ciertos atributos, como por ejemplo lo que sigue:

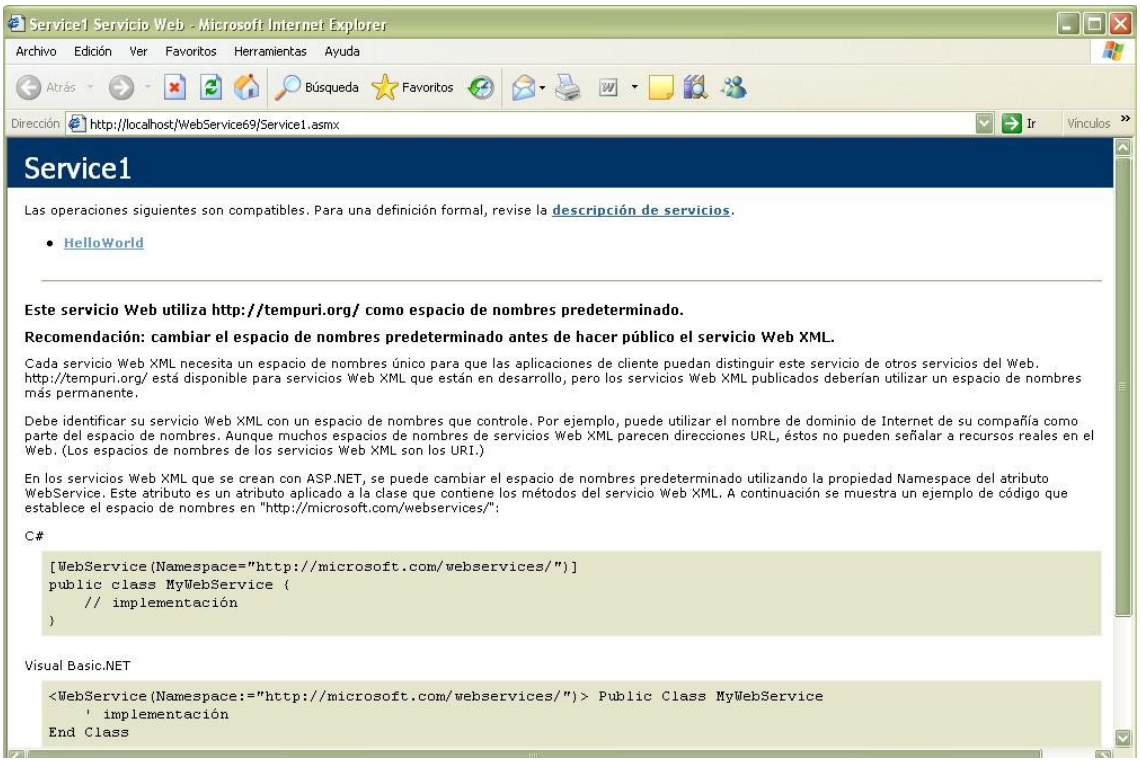
```
[
 WebService(Namespace="http://localhost/WebService69/",
        Name="Ejemplo de Servicio Web en C#.NET",
        Description="Descripción")
]
```

7.2.2 PRUEBA DE UN WEB SERVICE CREADO

Para probar el servicio web creado nos basamos en el ejemplo del apartado anterior.

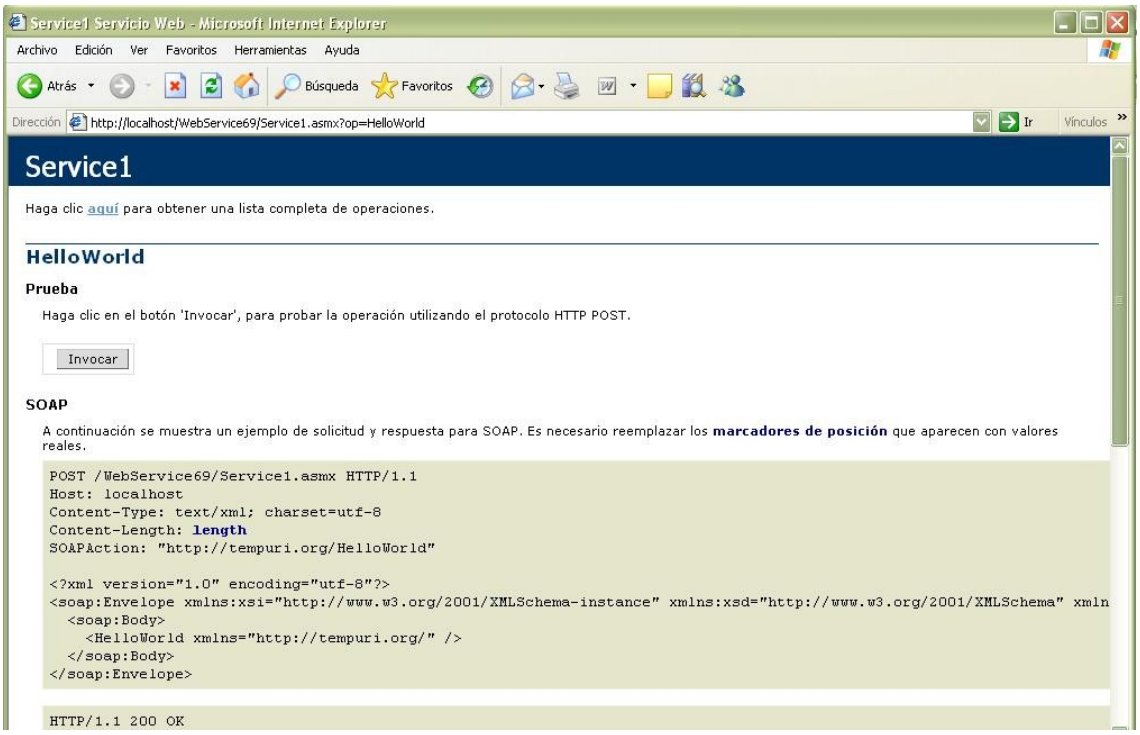
1. Descomentamos el método web *HelloWorld():string.*

2. Compilamos y ejecutamos la aplicación. Se abrirá el explorador web con algo así:

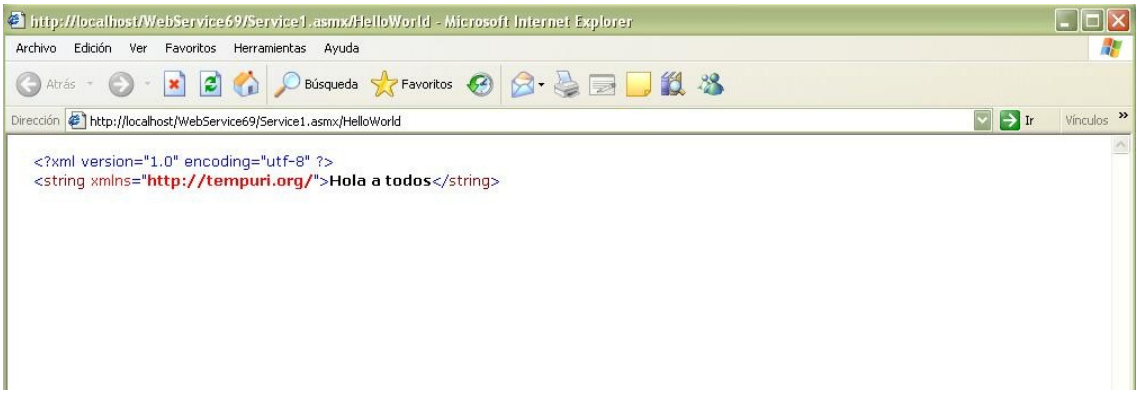


Aparecen listados los métodos web del que dispone el servicio. En este caso, el método *HelloWorld*.

3. Pinchamos en *HelloWorld* y saldrá la siguiente web.



4. Pinchamos en *Invocar* para invocarlo. Sólo funciona bien con métodos no complejos.

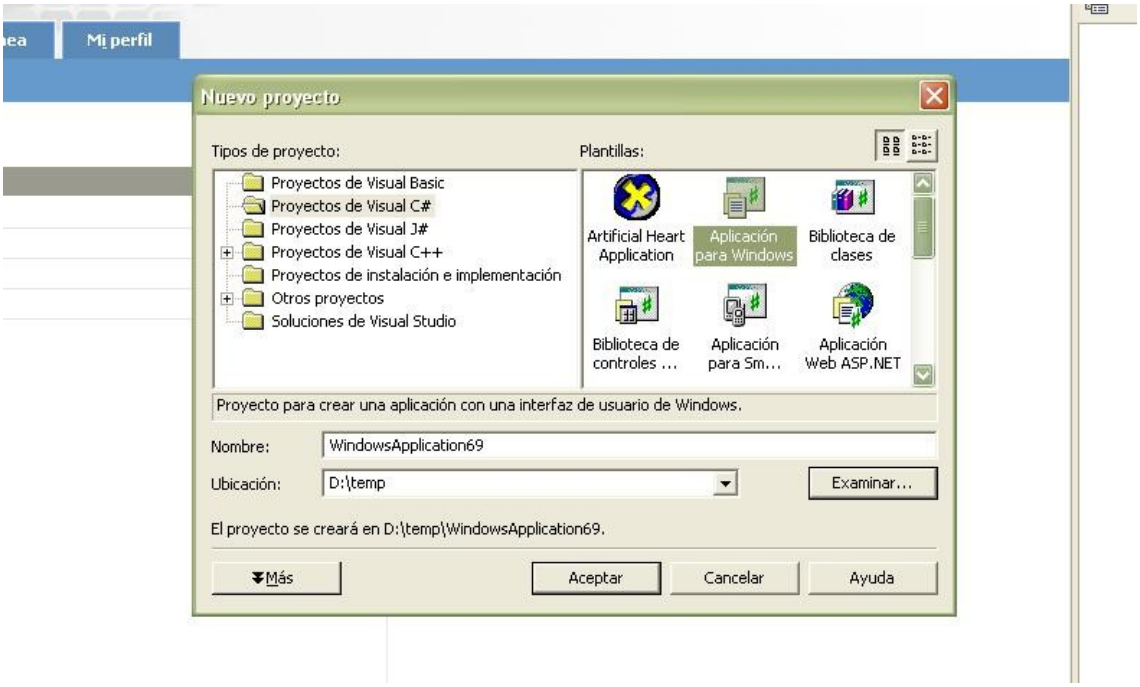


Se muestra lo que se enviaría desde el servicio web hasta la aplicación que lo ha invocado. Se puede observar que se trata de un documento XML.

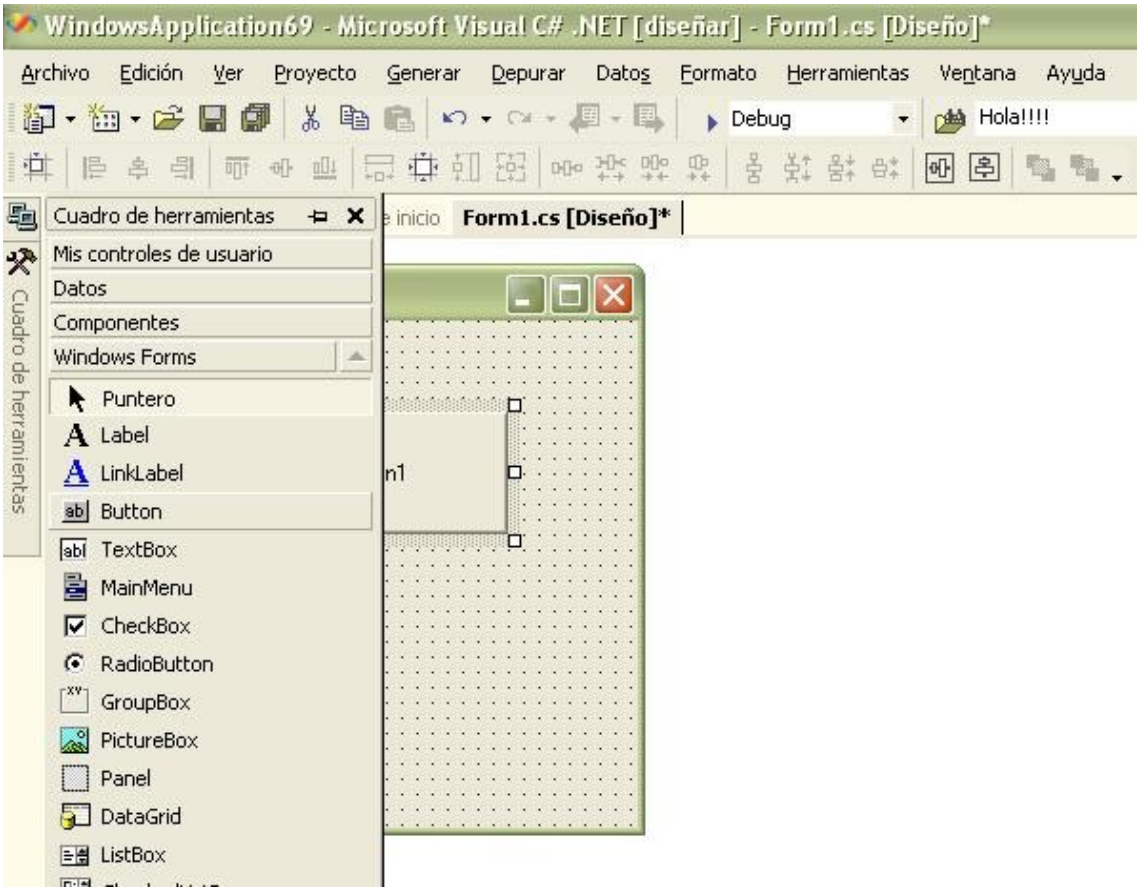
7.2.3 CONEXIÓN CON UN WEB SERVICE DESDE UNA APLICACIÓN

Hemos comprobado que el servicio web funciona. Ahora sólo queda invocarlo desde una aplicación.

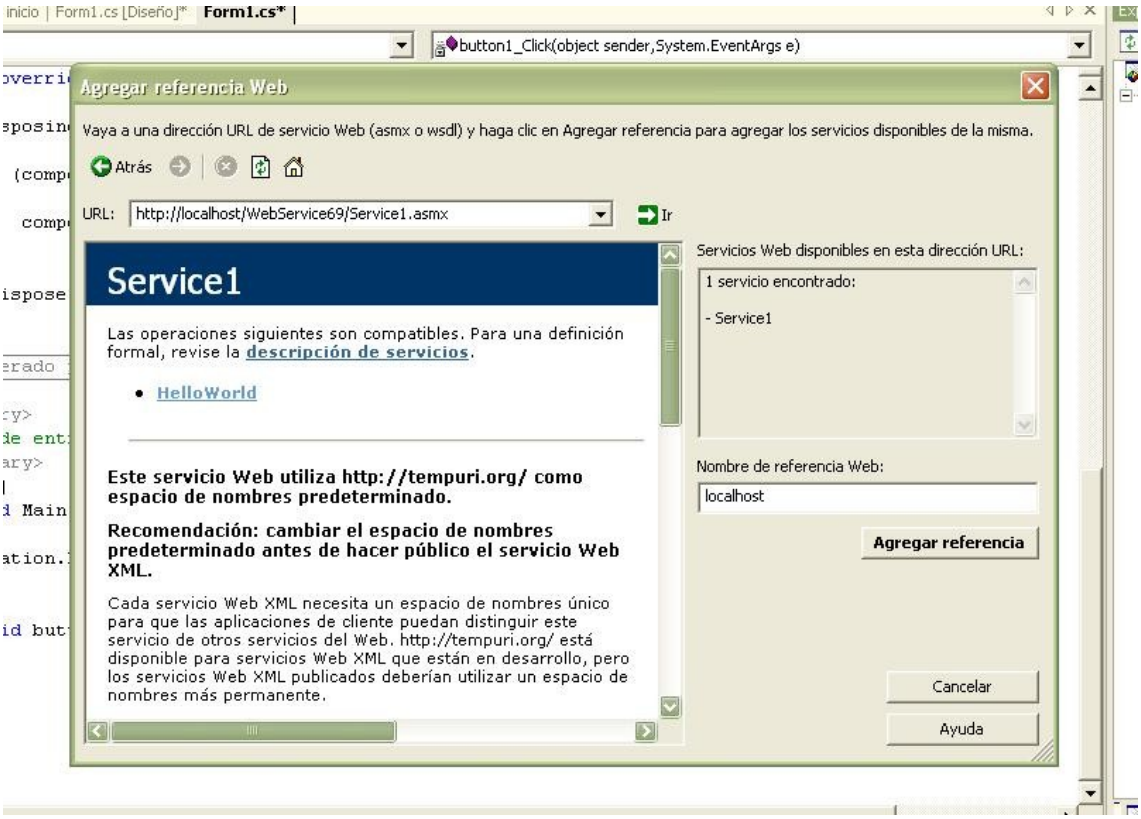
1. Para ello, vamos a crear una aplicación Windows con Visual Studio .NET 2003:



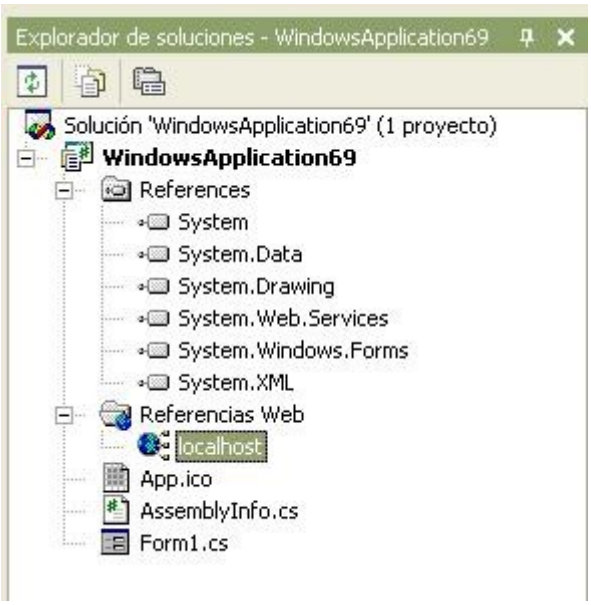
2. Abrimos Form1.cs y añadimos un botón:



3. En el explorador de soluciones creamos una nueva referencia web, seleccionando el servicio que acabamos de crear.



Pinchamos en "Agregar Referencia" y se nos añadirá la referencia web con el nombre "localhost" (tal y como aparece en el área de texto correspondiente).

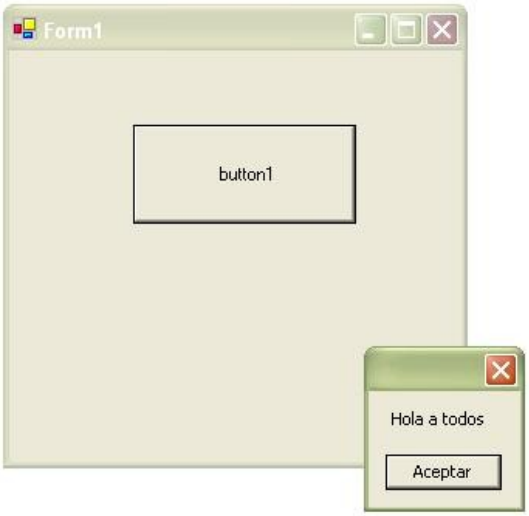


Importante: si se modifica o actualiza el servicio web, se debe actualizar la referencia web de la aplicación.

4. Creamos una instancia del servicio web e invocamos el método *HelloWorld* en el método delegado Click del botón que añadimos en el paso 2 (haciendo doble click sobre el botón en el formulario se crea automáticamente el método).

```
        private void button1_Click(object sender, System.EventArgs e)
        {
            localhost.Service1 s = new localhost.Service1();
            string texto = s.HelloWorld();
            MessageBox.Show(texto);
        }
    }
}
```

5. Si compilamos y ejecutamos la aplicación, podremos comprobar que haciendo click en el botón, efectivamente, contacta con el servicio, retrae el mensaje y lo muestra.



7.3 ANEXO 3: DTD´S DE LOS FICHEROS XML

Escenario.DTD

```
<!ELEMENT Escenario (Terreno, Aviones, Radares, Obstaculos, Objetivo)>
<!ELEMENT Terreno (Anchura, Profundidad, HeightMap)>
<!ELEMENT Aviones (Avion*)>
<!ELEMENT Radares (Radar*)>
<!ELEMENT Obstaculos (Obstaculo*)>
<!ELEMENT Objetivo (Id, X, Y, Z, DimX, DimY, DimZ)>
<!ELEMENT Misiles (Misil*)>
<!ELEMENT Avion (X, Y, Z, Roll, Pitch, Yaw, DimX, DimY, DimZ, Velocidad, Id)>
<!ELEMENT Radar (Id, X, Y, Z, Alcance, DminDet, TimeOut, Misiles*)>
<!ELEMENT Misil (Id, X, Y, Z, Alcance)>
<!ELEMENT Obstaculo (Id, X, Y, Z, Scale, DimX, DimY, DimZ)>
<!ELEMENT Anchura (#PCDATA)>
<!ELEMENT Profundidad (#PCDATA)>
<!ELEMENT HeightMap (#PCDATA)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT Z (#PCDATA)>
<!ELEMENT Roll (#PCDATA)>
<!ELEMENT Pitch (#PCDATA)>
<!ELEMENT Yaw (#PCDATA)>
<!ELEMENT DimX (#PCDATA)>
<!ELEMENT DimY (#PCDATA)>
<!ELEMENT DimZ (#PCDATA)>
<!ELEMENT Velocidad (#PCDATA)>
<!ELEMENT Id (#PCDATA)>
<!ELEMENT Alcance (#PCDATA)>
<!ELEMENT DminDet (#PCDATA)>
<!ELEMENT TimeOut (#PCDATA)>
<!ELEMENT Scale (#PCDATA)>
```

Trayectorias.DTD

```
<!ELEMENT Trayectorias (Trayectoria*)>
<!ELEMENT Trayectoria (IdAvion, PosicionAvion*)>
<!ELEMENT PosicionAvion (X, Y, Z, Roll, Pitch, Yaw)>
<!ELEMENT IdAvion (#PCDATA)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT Z (#PCDATA)>
<!ELEMENT Roll (#PCDATA)>
<!ELEMENT Pitch (#PCDATA)>
<!ELEMENT Yaw (#PCDATA)>
```

Total.DTD (resultados)

```
<!ELEMENT Total (Datos, Info, Solucion)>
<!ELEMENT Datos (Terreno, Aviones, Radares, Misiles, Obstaculos, Objetivo)>
<!ELEMENT Aviones (Avion*)>
<!ELEMENT Radares (Radar*)>
<!ELEMENT Misiles (Misil*)>
<!ELEMENT Obstaculos (Obstaculo*)>
```

```
<!ELEMENT Objetivo (Id, X, Y, Z)>
<!ELEMENT Info (NombreFichero, PorcentajeLlegadas)>
<!ELEMENT Solucion (TrayectoriasAviones, TrayectoriasMisiles, Eventos, Detecciones)>
<!ELEMENT TrayectoriasAviones (TrayectoriaAvion*)>
<!ELEMENT TrayectoriasMisiles (TrayectoriaMisil*)>
<!ELEMENT Eventos (Evento*)>
<!ELEMENT Detecciones (Deteccion*)>
<!ELEMENT Terreno (Anchura, Profundidad, HeightMap)>
<!ELEMENT Avion (Id, X, Y, Z, Roll, Pitch, Yaw)>
<!ELEMENT Radar (Id, X, Y, Z, Alcance)>
<!ELEMENT Misil (Id, X, Y, Z)>
<!ELEMENT Obstaculo (Id, X, Y, Z, Scale)>
<!ELEMENT TrayectoriaAvion (Id, TiempoPosicion*)>
<!ELEMENT TrayectoriaMisil (Id, TiempoPosicion*)>
<!ELEMENT Evento (Instante, Codigo, IdAvion, IdMisil, IdRadar)>
<!ELEMENT Deteccion (InstanteTiempo, IdRadar, IdAvion, PPD)>
<!ELEMENT TiempoPosicion (Instante, Posicion)>
<!ELEMENT Posicion (X, Y, Z, Roll?, Pitch?, Yaw?)>
<!ELEMENT Anchura (#PCDATA)>
<!ELEMENT Profundidad (#PCDATA)>
<!ELEMENT HeightMap (#PCDATA)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT Z (#PCDATA)>
<!ELEMENT Roll (#PCDATA)>
<!ELEMENT Pitch (#PCDATA)>
<!ELEMENT Yaw (#PCDATA)>
<!ELEMENT Id (#PCDATA)>
<!ELEMENT Alcance (#PCDATA)>
<!ELEMENT Scale (#PCDATA)>
<!ELEMENT NombreFichero (#PCDATA)>
<!ELEMENT PorcentajeLlegadas (#PCDATA)>
<!ELEMENT Instante (#PCDATA)>
<!ELEMENT Codigo (#PCDATA)>
<!ELEMENT IdAvion (#PCDATA)>
<!ELEMENT IdMisil (#PCDATA)>
<!ELEMENT IdRadar (#PCDATA)>
<!ELEMENT InstanteTiempo (#PCDATA)>
<!ELEMENT PPD (#PCDATA)>
```

7.4 ANEXO 4: PARÁMETROS DEL SISTEMA

Tanto los módulos cliente como los módulos del servidor utilizan diversas propiedades o parámetros que pueden ser configurados, como por ejemplo el número de simulaciones a realizarse, la separación que indica la precisión empleada en el Astar y que también se empleará para discretizar el escenario, si se debe hacer o no splines (la interpolación de trayectorias) y un largo etcétera.

Ya que algunos de estos parámetros pueden afectar tanto a los módulos cliente, como a los de servidor, es deseable que sólo se necesiten cambiar una vez, en el cliente.

Para ello, se dispondrá de un fichero *Config.xml* en la ruta */Datos/Configuracion*. Los parámetros modificables por el usuario se encuentran disponibles en la interfaz general.

La clase encargada de gestionar estas propiedades es PropertiesManager. Esta clase guarda las propiedades leídas del XML en un objeto Properties estático y de sólo lectura. Así, se puede acceder a las propiedades de forma estática (sin crear instancias de clase) al mismo tiempo que se cargan en ejecución leyendo del fichero XML. Con se consigue una utilización similar a las constantes de un programa pero sin la necesidad de recompilar al cambiarlas, ya que se cambian en el XML.

Ejemplo de acceso a una propiedad desde cualquier punto del programa:

```
PropertiesManager.properties.RUTA_HEIGHTMAPS
```

El siguiente es un fragmento del *Config.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<Config>
 <splines>1</splines>
 <simulacion>0</simulacion>
 <separacionPtos>50</separacionPtos>
 etc...
</Config>
```

7.5 ANEXO 5: CONSIDERACIONES DE EFICIENCIA

Dentro de la implementación, dos procesos han sido identificados como los más problemáticos en cuanto al rendimiento. Por un lado, la ejecución del algoritmo AStar, y por otro, el tiempo en guardar los ficheros XML generados.

AStar

El AStar puede ser un proceso crítico en el rendimiento. Se trata del mejor algoritmo para el cálculo de trayectorias óptimas en el menor tiempo posible, pero sin embargo, para escenarios grandes, puede ser costoso.

Por ello, es importante emplear buenas prácticas de programación aquí. En este caso, se ha empleado una estructura de datos de tipo "Heap" ordenada, de forma que los sucesores en ella se encuentren por orden de menor a mayor estimación+coste de camino recorrido. De esta forma, el obtener los sucesores en el cálculo es inmediato y no hay que buscar el siguiente sucesor a considerar dentro del conjunto.

Guardado en XML´s

En este punto se planteo un problema como es la duración de generado de los ficheros XML.

Al tratarse de ficheros de gran extensión, el problema reside en emplear concatenación sencilla de strings. Para solucionarlo, se sustituyo por el uso de StringBuilder.

Habitualmente se trabaja concatenando diferentes strings o reescribiendo el mismo string una y otra vez. Ambos modos son negativos para el rendimiento de nuestra aplicación, dado que cada vez que se hace se crea una instancia string bastante molesta. StringBuilder ofrece una notabilísima mejora de eficiencia a la hora de concatenar strings.

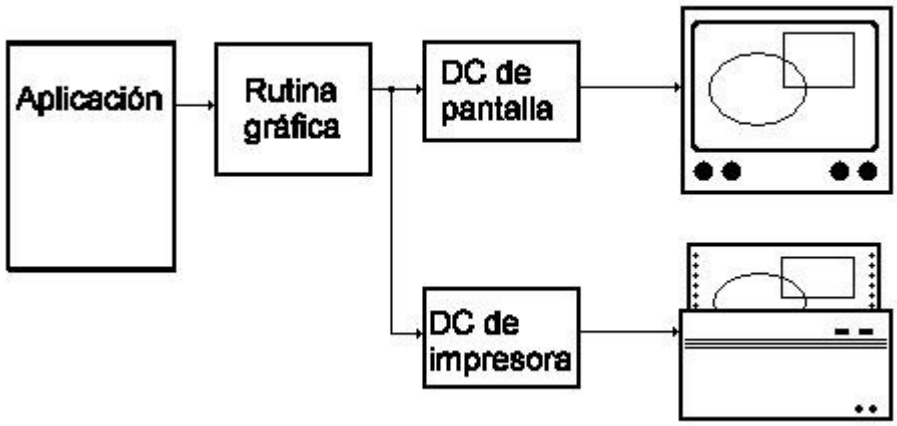
7.6 ANEXO 6: ANEXO GDI

El GDI (Graphics Device Interface), es el interfaz de dispositivo gráfico, que contiene todas las funciones y estructuras necesarias que nos permiten comunicar nuestras aplicaciones con cualquier dispositivo gráfico de salida conectado a nuestro ordenador: pantalla, impresora, plotter, etc.

Veremos que podremos trazar líneas, curvas, figuras cerradas, polígonos y mapas de bits. Además, podremos controlar características como colores, aspectos de líneas y tramas de superficies rellenas, mediante objetos como plumas, pinceles y fuentes de caracteres.

Las aplicaciones Windows dirigen las salidas gráficas a lo que se conoce como un Contexto de Dispositivo, abreviado como DC. Cada DC se crea para un dispositivo concreto.

Un DC es una de las estructuras del GDI, que contiene información sobre el dispositivo, como las opciones seleccionadas, o modos de funcionamiento.



La aplicación crea un DC mediante una función del GDI, y éste devuelve un manipulador de DC (hDC), que se usará en las siguientes llamadas para identificar el dispositivo que recibirá la salida. Mediante el DC, la aplicación también puede obtener ciertas capacidades del dispositivo, como las dimensiones del área de impresión, la resolución, etc.

La salida puede enviarse directamente al DC del dispositivo físico, o bien a un dispositivo lógico, que se crea en memoria o en un metafile. La ventaja de usar un dispositivo lógico es que se almacena la salida completa y posteriormente puede enviarse al cualquier dispositivo físico.

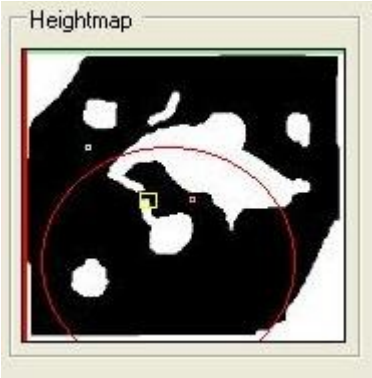
También existen funciones para seleccionar diferentes modos y opciones del dispositivo. Eso incluye colores del texto y fondo, plumas y pinceles de diferentes colores y texturas para trazar líneas o rellenar superficies, y lo que

se conoce como operaciones binarias de patrones (Binary Raster Operations), que indican cómo se combinan las nuevas salidas gráficas con las existentes previamente. También existen varios sistemas de mapeo de coordenadas, para traducir las coordenadas usadas en la aplicación con las el sistema de coordenadas del dispositivo.

Windows usa objetos y manipuladores para acceder a los recursos del sistema, en lo que se refiere al GDI existen los siguientes objetos:

Mapas de bits, pinceles, fuentes, plumas, plumas extendidas, regiones, contextos de dispositivos, contextos de dispositivo de memoria, metafiles, contextos de dispositivo de metafiles, metafiles mejorados y contextos de dispositivo de metafiles mejorados.

Nuestro caso



Utilizamos un objeto System.Windows.Forms.PictureBox para representar el mapa. Hemos de crear un objeto de tipo System.Drawing.Graphics a partir del objeto PictureBox. Sobre este objeto dibujaremos lo que necesitamos con lo métodos proporcionados, como *DrawRectangle*, o *DrawEllipse*.

```
Graphics graph = imagen.CreateGraphics();
```

```
graph.DrawRectangle(
    new Pen(Color.LightBlue,1.0f),
    (int)this.aCoordenadasBitmapX(avion.Posicion.Z,escenario,imagen),
    (int)this.aCoordenadasBitmapY(avion.Posicion.X,escenario,imagen),
    2,2
);
```

```
graph.DrawEllipse(
    new Pen(Color.Red,1.0f),
    (int)this.aCoordenadasBitmapX(radar.Posicion.Z-radar.Alcance,escenario,imagen),
    (int)this.aCoordenadasBitmapY(radar.Posicion.X-radar.Alcance,escenario,imagen),
    (int)this.aCoordenadasBitmapX(2*radar.Alcance,escenario,imagen),
    (int)this.aCoordenadasBitmapY(2*radar.Alcance,escenario,imagen)
    );
```

7.7 ANEXO 7: ALGORITMO A ESTRELLA

El problema de algunos algoritmos de búsqueda en grafos informados, como puede ser el algoritmo voraz, es que se guían en exclusiva por la función heurística, la cual puede no indicar el camino de coste más bajo, o por el coste real de desplazarse de un nodo a otro (como los algoritmos de escalada), pudiéndose dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Es por ello bastante intuitivo el hecho de que un buen algoritmo de búsqueda informada debería tener en cuenta ambos factores, el valor heurístico de los nodos y el coste real del recorrido.

Así, el algoritmo A* utiliza una función de evaluación $f(n) = g(n) + h(n)$, donde $h(n)$ representa el valor heurístico del nodo a evaluar, y $g(n)$, el coste real del camino recorrido para llegar a dicho nodo. A* mantiene dos estructuras de datos auxiliares, que podemos denominar *abiertos*, implementado como una cola de prioridad (ordenada por el valor $f(n)$ de cada nodo), y *cerrados*, donde se guarda la información de los nodos que ya han sido visitados. En cada paso del algoritmo, se expande el nodo que esté primero en abiertos, y en caso de que no sea un nodo objetivo, calcula la $f(n)$ de todos sus hijos, los inserta en abiertos, y pasa el nodo evaluado a cerrados.

El algoritmo es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad: mientras que $h(n)$ tiende a primero en profundidad, $g(n)$ tiende a primero en anchura. De este modo, se cambia de camino de búsqueda cada vez que existen nodos más prometedores.

Propiedades

Como todo algoritmo de búsqueda en anchura, A* es un algoritmo completo: en caso de existir una solución, siempre dará con ella. Si para todo nodo n del grafo se cumple $g(n) = 0$, nos encontramos ante una búsqueda voraz. Si para todo nodo n del grafo se cumple $h(n) = 0$, A* pasa a ser una búsqueda de coste uniforme no informada. Para garantizar la optimalidad del algoritmo, la función $h(n)$ debe ser admisible, esto es, que no sobrestime el coste real de alcanzar el nodo objetivo.

De no cumplirse dicha condición, el algoritmo pasa a denominarse simplemente A, y a pesar de seguir siendo completo, no se asegura que el resultado obtenido sea el camino de coste mínimo. Asimismo, si garantizamos que $h(n)$ es consistente (o monótona), es decir, que para cualquier nodo n y cualquiera de sus sucesores, el coste estimado de alcanzar el objetivo desde n no es mayor que el de alcanzar el sucesor más el coste de alcanzar el objetivo desde el sucesor.

Complejidad computacional

La complejidad computacional del algoritmo está íntimamente relacionada con la calidad de la heurística que se utilice en el problema. En el caso peor, con una heurística de pésima calidad, la complejidad será exponencial, mientras que en el caso mejor, con una buena $h(n)$, el algoritmo se ejecutará en tiempo lineal. Para que esto último suceda, se debe cumplir que

$$h(x) \leq g(y) - g(x) + h(y)$$

donde h* es una heurística óptima para el problema, como por ejemplo, el coste real de alcanzar el objetivo.

Complejidad en memoria

El espacio requerido por A* para ser ejecutado es su mayor problema. Dado que tiene que almacenar todos los posibles siguientes nodos de cada estado, la cantidad de memoria que requerirá será exponencial con respecto al tamaño del problema. Para solucionar este problema, se han propuesto diversas variaciones de este algoritmo, como pueden ser RTA, IDA o SMA.

Implementación en pseudocódigo

Algorimo A*:

```
ABIERTOS := [INICIAL]            //inicialización
CERRADOS := []
f'(INICIAL) := h'(INICIAL)
repetir
   si ABIERTOS = [] entonces FALLO
   si no                          // quedan nodos
      extraer MEJORNODO de ABIERTOS con f' mínima
         // cola de prioridad
      mover MEJORNODO de ABIERTOS a CERRADOS
      si MEJORNODO contiene estado_objetivo entonces
         SOLUCION_ENCONTRADA := TRUE
      si no
         generar SUCESORES de MEJORNODO
         para cada SUCESOR hacer TRATAR_SUCESOR
hasta SOLUCION_ENCONTRADA o FALLO
```

Tratar sucesor:

```
SUCESOR.ANTERIOR := MEJORNODO // nodo padre
g(SUCESOR):= g(MEJORNODO) + coste(MEJORNODO->SUCESOR)
                                  // coste del camino hasta
SUCESOR

caso SUCESOR = VIEJO perteneciente a CERRADOS
```

```
   si g(SUCESOR) < g(VIEJO) entonces // (no si monotonía)
      // nos quedamos con el camino de menor coste
      VIEJO.ANTERIOR := MEJORNODO
      actualizar g(VIEJO) y f'(VIEJO)
      propagar g a sucesores de VIEJO
   eliminar SUCESOR
   añadir VIEJO a SUCESORES_MEJORNODO
caso SUCESOR = VIEJO perteneciente a ABIERTOS
   si g(SUCESOR) < g(VIEJO) entonces
      // nos quedamos con el camino de menor coste
      VIEJO.ANTERIOR := MEJORNODO
      actualizar g(VIEJO) y f'(VIEJO)
   eliminar SUCESOR
   añadir VIEJO a SUCESORES_MEJORNODO
caso SUCESOR no estaba en ABIERTOS ni CERRADOS
   añadir SUCESOR a ABIERTOS
   añadir SUCESOR a SUCESORES_MEJORNODO
   f'(SUCESOR) := g(SUCESOR) + h'(SUCESOR)
```

8 LISTA DE PALABRAS CLAVE

- .NET
- C#
- Web Service
- GDI
- A*
- Interpolación
- Motor Gráfico
- 3D
- UAV
- Trayectoria

9 BIBLIOGRAFÍA

- *Profesional C#.NET*, Robinson, Allen, …, Wrox Press, 2002.

- *Profesional UML with Visual Studio. NET*, Filev, Loton, McNeisk,… ,Wrox Press, 2002.

- http://www.codeproject.com/cs/algorithms/csharppathfind.asp

Podrá encontrar alojado el proyecto en Internet en la página del director del proyecto:

- http://www.fdi.ucm.es/profesor/jjruz/SI/

10 AGRADECIMIENTOS

Sergio

A mis compañeros, por haber hecho un gran trabajo; Raquel, por haberme apoyado durante el desarrollo de la aplicación; a nuestro director de proyecto por habernos ayudado a sacarla adelante.

Julián

A mi familia y amigos, a todos los que me apoyaron y me dieron ánimos, gracias por estar siempre ahí, a mis compañeros por su trabajo y al director del proyecto por su ayuda.

Enrique

En primer lugar a mis compañeros, ha sido un placer trabajar con vosotros; a José Jaime Ruz por haber confiado en nosotros para la realización del proyecto; y a todos mis familiares y amigos porque su compañía hace que todo cobre un sentido.

11 AUTORIZACIÓN A LA UCM

Por la presente se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto esta memoria, como el código, la documentación y el prototipo desarrollado.

Los autores:

Sergio Cuesta Boluda

A. Julián Martínez de la Casa Santos

Enrique Aguilera Sanz