Índice.

1 Introducción.

El clima ha sido siempre objeto de estudio por parte del hombre. Su asombrosa complejidad hace que incluso durante gran parte de la historia se atribuyesen los cambios del clima a algo divino y sobrenatural.

Hoy en día, se puede dar respuestas a gran parte de los procesos climáticos que actúan en el planeta. El uso de nuevas tecnologías ha ayudado en gran manera a avanzar en el estudio de este campo. Aun así, todavía hay muchos otros que continúan siendo incomprensibles para el hombre.

Uno de los últimos hallazgos es el hecho de que el clima a nivel general no es un proceso estable, si no que cambia en el tiempo. Ese tiempo en el que oscila el clima a nivel global es del orden de decenas o incluso centenas de miles de años, lo que hace que la humanidad tal y como la conocemos hoy este viviendo por primera vez esta alteración climática.

Lo que conocemos hoy en día como cambio climático es un proceso que tiene un efecto grandísimo en el ser humano. Por eso, y por ser algo tan complejo, ha de ser estudiado a consciencia y con la suficiente antelación como para poder llevar a cabo acciones que hagan disminuir sus efectos sobre la humanidad y el planeta.

Así pues, se ha de utilizar todas las herramientas disponibles para elaborar estudios sobre el cambio climático. Una de estas herramientas, que por su gran potencia puede ser muy útil, es la simulación y representación por computador.

1.1 Objetivos.

El objetivo principal de este proyecto es diseñar e implementar una aplicación web que sea capaz de representar como quedan las zonas de la tierra cercanas a la costa después de que aumente el nivel del mar. Con el propósito de que pueda ser utilizada para representar los resultados de un proceso de simulación del cambio climático que calcule a cuanto asciende el nivel del mar debido al deshielo de los polos producidos por el aumento de la temperatura global del planeta.
Hoy en día, ya se han realizado experimentos de simulación de este tipo, pero no existe ninguna aplicación que permita su representación visual lo que hace que el análisis de los resultados sea mucho más pesado.

Otro objetivo es que la aplicación se explote y se continúe su desarrollo por otras personas en el futuro, lo cual implica que se ha de realizar con la más reciente tecnología. Es por ello que se requiere que la tecnología usada sea vigente durante un largo periodo de tiempo.

1.2 Motivaciones.

Este proyecto es muy atractivo desde diferentes puntos de vista.

Desde el punto de vista conceptual, una de las grandes motivaciones es que todo esto del cambio climático es algo que esta muy de actualidad y del cual se comienzan a saber cosas muy interesante. A de más, las consecuencias del aumento global de la temperatura pueden cambiar de una manera drástica el mundo tal y como lo conocemos, lo que hace que su estudio sea aun más interesante.

Desde el punto de vista tecnológico, la realización de este proyecto permite la utilización y aprendizaje de tecnologías totalmente novedosas, como es el caso de Microsoft Silverlight, y otras tecnologías de rabiosa actualidad como C# o Virtual Earth.

Por otra parte, este proyecto es muy visual. No es la típica aplicación que calcula un resultado numérico que es necesario interpretar, si no que representa ese resultado de manera entendedora para cualquier persona. Esto es algo que hace que su desarrollo sea más atractivo y menos pesado.

2 Conceptos previos sobre el Cambio Climático.

Se entiende por cambio climático al cambio en el estado del clima que puede ser identificado por variaciones en la media y en la variabilidad de las propiedades del clima (temperatura, nubosidad, precipitaciones,...), persistiendo estas variaciones durante un periodo de tiempo relativamente largo (décadas, siglos,...).

Cualquier tipo de cambio climático además implica cambios en otras variables. La complejidad del problema y sus múltiples interacciones hacen que la única manera de evaluar estos cambios sea mediante el uso de modelos computacionales que intentan simular la física de la atmósfera y del océano y que tienen una precisión limitada debido al desconocimiento del funcionamiento de la atmósfera.

La radiación emitida por el Sol es la fuente principal de energía de la Tierra. Así pues, podemos decir a gran escala que, el clima de la Tierra depende del balance de radiación solar. Es decir, depende de la cantidad de energía provinente del Sol que la Tierra absorbe y de la cantidad que es capaz de reflejar por la superficie terrestre y la atmósfera.

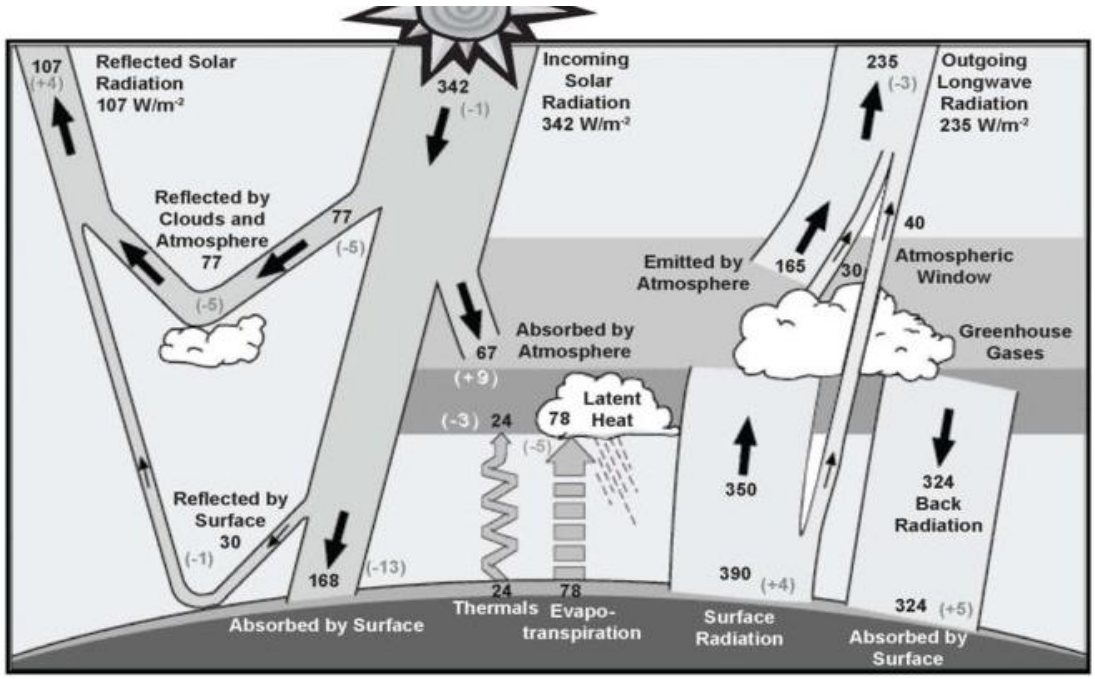


IMAGEN 2.1, RESUMEN DE LOS DIFERENTES PROCESOS INTERNOS.

La energía que recibimos del Sol y que llega a la parte alta de la atmósfera se compone de radiación ultravioleta, luz visible y radiación infrarroja. Para cuando esta energía solar llega a la superficie de la Tierra (corteza terrestre y océanos), ya ha sido absorbida en parte por el ozono, el vapor de agua y otros componentes de la atmósfera, de manera que la energía que realmente llega a la superficie terrestre suele ser en un 49% radiación infrarroja, en un 42% luz visible y un 9% es radiación ultravioleta.
Alrededor de un 30% de la energía que recibe la Tierra se refleja y devuelve al espacio, mientras que el 70% restante se absorbe de manera no uniforme. Estas diferencias producen fenómenos de convección, corrientes atmosféricas que transportan calor, evaporación, condensación…, que producen el clima.

Según la cantidad de radiación infrarroja que emite la Tierra (240 W.m2), sabemos que su temperatura debería rondar los -18 ºC. Pero lo cierto es que la Tierra tiene una temperatura media de de 15ºC. La diferencia entre la energía a la que equivalen estos 15 ºC y la realmente emitida es la que se devuelve al espacio más lentamente porque queda atrapada por las nubes y ciertos gases atmosféricos como el dióxido de carbono, el metano y óxidos de nitrógeno, por lo que estos gases reciben el nombre de gases de efecto invernadero. Así que el efecto invernadero es un fenómeno natural y necesario, ya que es responsable de estos 33 grados de diferencia tan beneficiosos para la vida en el planeta. Por otra parte, la acción del hombre tiene un efecto potenciador sobre este efecto invernadero lo que amplifica las variaciones en el clima, aunque estas no son íntegramente producidas por el ser humano.

Así pues, estas variaciones o cambios son producidos por dos grandes fuentes: procesos internos y fuerzas externas.
Entre las fuerzas externas, se encuentran procesos naturales, como los cambios en la radiación solar o las erupciones volcánicas, y procesos generados por la acción humana sobre el planeta como el cambio en la composición de la atmosfera terrestre producido por el incremento de emisiones de gases que se ha producido a raíz de la Revolución Industrial.

Los procesos internos son generados por componentes dentro del sistema climático como los océanos, la criósfera y la atmósfera. Estos componentes generan procesos complejos interactuando entre ellos.

A continuación explicaremos algunos de los procesos más importantes que hacen variar el balance energético de la Tierra, tanto procesos internos del clima como externos a este.

2.1 Procesos Internos.

Existen procesos climáticos capaces de interferir y modificar el propio sistema climático. Estos procesos son del todo naturales y se dan en la Tierra desde sus inicios. Por ejemplo, los océanos son capaces de contener mucho calor lo que hace el clima mas templado, el hielo terrestre refleja un índice alto de la energía solar, la atmosfera refleja componentes dañinos de la luz solar y a su vez mantiene calor en la Tierra, etc.
A simple vista, estos procesos parecen sencillos pero su complejidad radica en la interacción entre estos tres elemento: océanos, criósfera y atmosfera.
A continuación se explican procesos importantes que se dan dentro de dos de estos tres elementos: la criósfera y la atmosfera.

2.1.1 La criósfera.

La criósfera incluye partes del sistema de la Tierra en donde el agua se encuentra en forma congelada (sólida). Esto incluye: nieve, hielo marino, placas de hielo y glaciares, bloques de hielo(inlandsis[1] y mesetas de hielo[2]) y suelos permafrost[3].

1 Inlandsis: u hoja de hielo, denominación correspondiente a todo gran campo glaciar del tipo llamado *hielo continental* localizado en latitudes elevadas y con extensiones realmente continentales .
2 Meseta de hielo: plataforma gruesa que flota en el océano, originada cuando un glaciar o inlandsis se extiende por la costa hasta llegar al océano
3 Permafrost: o permagel, capa de hielo permanentemente congelada a niveles superficiales del suelo en regiones muy frías.

Comprende las regiones cubiertas por nieve, hielo, en la tierra, el mar, incluyendo la Antártida, el Océano Glacial Ártico, Groenlandia, el Norte de Canadá, el Norte de Siberia y la mayor parte de las cimas más altas de cadenas montañosas del mundo.

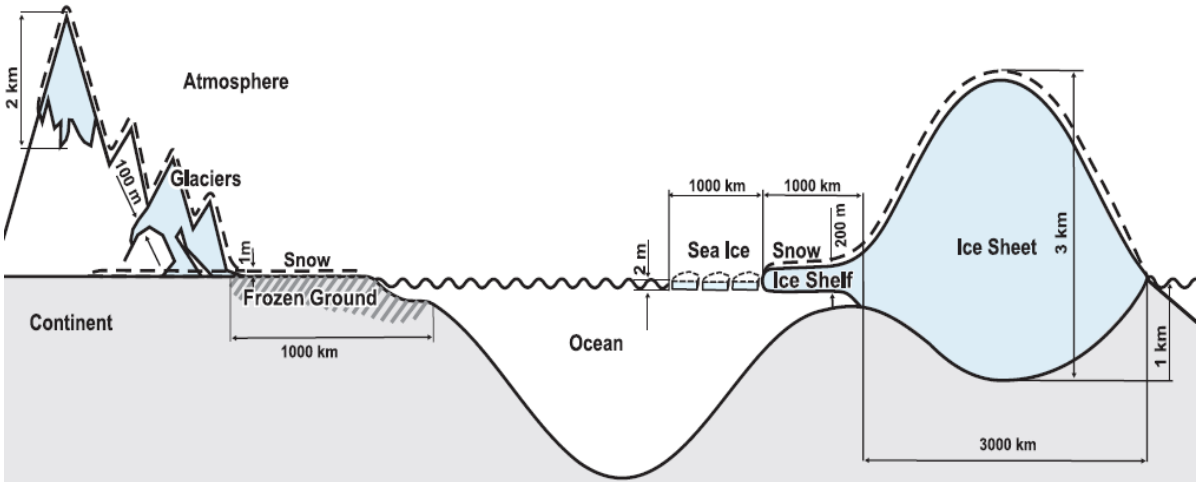


IMAGEN 2.2, FORMAS DE HIELO EN LA TIERRA.

Aproximadamente, tres cuartos del agua dulce del mundo está contenida en la criósfera. Esto significa claramente, que su presencia afecta de manera considerable en el volumen de los océanos y de los niveles globales del mar.

A de más de ser el principal contenedor de agua dulce del planeta, la criósfera tiene mucha relevancia dentro del sistema climático de la Tierra. De hecho está considerado como el segundo elemento más grande en el sistema climático, solo superado por los océanos.

Se conoce, que recientes cambios en la criósfera han supuesto un impacto considerable sobre el clima global del planeta. Esto es debido a que la criósfera es una parte importante del sistema de la Tierra, y que está sumamente interconectada con otros elementos de este sistema como son la hidrosfera y la atmosfera.

Menos hielo en tierra significa que aumenta el nivel del mar.

La relación con la hidrosfera viene dada por el hecho de que todo el hielo que se derrite y por tanto deja de formar parte de la criósfera, pasa a formar parte de la hidrosfera. Es decir, a menor criósfera, mayor hidrosfera, o en términos más simples contra menos hielo en la Tierra más alto será el nivel del mar.

Componente	Superficie (10^6 km^2)	Volumen (10^6 km^3)	Potencial de Elevación del Nivel del Mar (m)
Nieve	1.9 – 45.2	0.0005 - 0.005	0.001 – 0.01
Hielo marino	19-27	0.019 – 0.025	~0
Placas de hielo y glaciares	0.51-0.54	0.05-0.13	0.15-0.37
Inlandsis (hoja de hielo)	14.0	27.6	63.9
- Groenlandia	1.7	2.9	7.3
- Antártida	12.3	24.7	56.6
Permafrost	22.8	0.011-0.037	0.03-0.10

TABLA 2.1, ÁREA, VOLUMEN Y EQUIVALENCIA EN NIVEL DEL MAR.

La vinculación con la atmosfera ocurre mediante dos relaciones:

El derretimiento del hielo origina más calentamiento.

El **albedo** es la relación, expresada en porcentaje, de la radiación que cualquier superficie refleja sobre la radiación que incide sobre la misma. Las superficies claras tienen valores de albedo superior a las oscuras, y las brillantes más que las mates.

El albedo medio de la Tierra es del 30-32%, mientras que el albedo de la nieve o el hielo es de entre un 80 y un 90%. Es decir, debido a que la nieve y el hielo son de color claro, la criósfera refleja mayor cantidad de energía solar de vuelta al espacio. Cuando la nieve se derrite y el color oscuro del suelo queda expuesto, la superficie de la Tierra absorbe mayor cantidad de energía la cual luego es irradiada hacia la atmósfera y de esta manera, hace que la atmósfera sea más caliente.

En resumidas cuentas podríamos decir que el calor genera más calor y el frío más frío, debido a esta capacidad de reflexión de la luz que posee el agua en su estado sólido.

Por lo tanto, al subir las temperaturas, y debido al descenso que esto provoca en la superficie ocupada por la criósfera, la temperaturas tenderían todavía más a subir. Esto provoca un claro bucle retroalimentado.

El permafrost que se derrite libera gases invernadero.

El permafrost submarino, que se encuentra congelado bajo las aguas del mar y sobre todo de los lagos contiene una elevada cantidad de metano, uno de los gases de efecto invernadero.

El permafrost que se derrite libera restos ricos en carbono de plantas y animales. Estos restos se hunden hasta el fondo de los lagos, se descomponen y producen metano que burbujea hacia la superficie y a la atmósfera.

Según el IPCC, aunque se sabe del alto potencial de metano contenido en el permafrost, no se tiene todavía constancia de como afecta al clima, debido a que las observaciones disponibles no permiten asegurar la teoría. Aun así, el liberamiento de gas metano, debido o no a la descongelación del permafrost, provoca un aumento de la temperatura debido al efecto invernadero, lo que provocaría un deshielo todavía mayor de permafrost. Con lo que genera otro bucle retroalimentado.

2.1.2 La atmósfera.

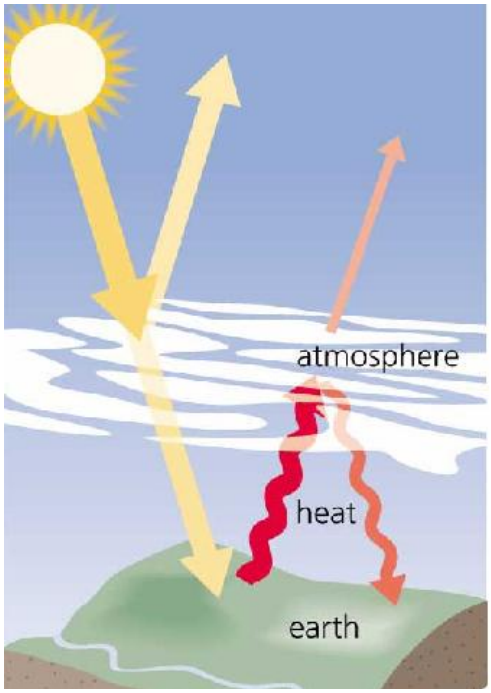


IMAGEN 2.3, PROCESOS DE LA ATMOSFERA.

La atmósfera envuelve a la Tierra y nos protege de los peligrosos rayos del Sol. La atmósfera es una mezcla de gases cuya densidad va disminuyendo a medida que ascienden y, eventualmente, llegan hasta el espacio. Está compuesta de Nitrógeno (78%), Oxígeno (21%), y otros gases (1%).

Podríamos decir, que en la atmosfera es una parte fundamental en el balance energético de la Tierra, ya que en el interviene tanto la composición de la atmosfera como procesos que se dan en esta.

La parte alta de la atmosfera se encarga de no dejar pasar parte de los rayos solares (capa de ozono- rayos ultravioletas). La parte baja de la atmosfera, es la encargada de contener calor. Sin ella la Tierra estaría totalmente congelada. Esta contención del calor se debe a los gases de efecto invernadero. Lo relevante es que no todos son producimos por el efecto del hombre sobre la Tierra.

Aunque pueda sorprender, el vapor de agua está considerado como el más relevante de los gases llamados de efecto invernadero. De hecho es el gas de efecto invernadero con más concentración en la atmósfera y por tanto el más importante de la Tierra, ya que provoca un efecto invernadero del orden de +(30±5)ºC, por lo que sin él, la Tierra estaría totalmente helada. Es decir, el efecto invernadero es algo natural, propio de la Tierra aunque en estos momentos se vea alterado por la acción humana.

La evaporación del agua origina más calor.

Una característica importante a tener en cuenta es que presenta en si mismo un bucle retroalimentado. Al calentarse la superficie terrestre, se genera una mayor cantidad de vapor de agua, que a su vez provoca un aumento de las temperaturas, con lo que acelera la formación de más vapor de agua.

La condensación del agua origina menos calor.

A su vez, el vapor de agua en la atmosfera genera nubes. Las nubes son un elemento complejo del clima, ya que presentan dos efectos opuestos. Por una banda, El albedo de las nubes es de un 50%, superior al de la media terrestre, por lo tanto, reflejan un índice mayor de la energía proveniente del Sol. Es decir, a más nubes menos calor. Pero por otra parte, las nubes dificultan el enfriamiento de la corteza terrestre ya que impiden que esta se desprenda de manera óptima del calor que contiene. De este modo, a más nubes más calor.

El balance entre estos dos efectos depende de muchos factores, incluyendo propiedades físicas de las nubes. Hoy en día, no se sabe con total exactitud como afecta en el clima. Pero se calcula que su acción de calentamiento por efecto del aumento invernadero supone unos 30 Wm^{-2} , mientras que su acción de enfriamiento por el reflejo de parte de la radiación es del orden de 50 Wm^{-2} , lo que supone un efecto neto de enfriamiento de unos 20 $Wm^{-2.}$

La cantidad de vapor de agua así como su distribución vertical son claves en el cálculo de esta

retroalimentación. Los procesos que controlan la cantidad de vapor en la atmósfera son complejos de modelar y aquí radica gran parte de la incertidumbre sobre el calentamiento global.

2.2 Fuerzas Externas.

Las fuerzas externas al clima que influyen en este son muy variadas. La mayoría de estas fuerzas externas son de origen natural, aunque las más relevantes en la explicación del actual cambio en el clima son las originadas por la acción del ser humano, las emisiones de gases de efecto invernadero.

Los forzamientos externos naturales normalmente actúan de forma sistemática sobre el clima, como las erupciones volcánicas, aunque también los hay aleatorios como es el caso de los impactos de meteoritos. Son capaces de alterar el clima drásticamente pero su manera de manifestarse se rige por sistemas caóticos, con lo cual es imposible su estudio en el área de la simulación.
Entre los forzamientos externos naturales se pueden destacar las emisiones de gases de efecto invernadero producidas por la erupciones volcánicas, el incremento o decremento de la emisión de radiación solar debido a las manchas solares o incluso el impacto de meteoritos sobre el planeta puede cambiar la composición atmosférica o incluso variar la órbita o la inclinación terrestre modificando así la cantidad de energía solar que llega a la Tierra.

Pero como ya se ha dicho anteriormente, estas fuerzas externas de origen natural no son las que explican el cambio climático que se está llevando acabo en estos momentos en la Tierra, si no que son las originadas por diferentes acciones del hombre las que verdaderamente están afectando en le clima de manera negativa. Entre estas acciones que realiza el ser humano y que son capaces a largo plazo de cambiar el clima del planeta destaca la emisión de gases invernadero.

2.2.1 Gases de efecto invernadero.

Se denomina gases de efecto invernadero a los gases cuya presencia en la atmósfera contribuye al efecto invernadero. Los más importantes están presentes en la atmósfera de manera natural, aunque su concentración puede verse modificada por la actividad humana.

El más famoso entre estos gases es el dióxido de carbono (CO_2), aunque el gas más relevante es el vapor de agua. Si los ordenamos por su efecto de manera decreciente obtenemos la siguiente clasificación:

1. Vapor de agua (H2O)
2. Dióxido de carbono (CO2)
3. Metano (CH4)
4. Óxidos de nitrógeno (NOx)
5. Ozono (O3)
6. Clorofluorocarburos (artificiales).

Si bien todos ellos, excepto los compuestos del flúor, son naturales, en tanto que existen en la atmósfera desde antes de la aparición del hombre, a partir de la Revolución Industrial, y debido principalmente al uso intensivo de combustibles fósiles en las actividades industriales y el transporte, se han producido sensibles incrementos en las cantidades de óxidos de nitrógeno y dióxido de carbono emitidas a la atmósfera. Se estima que también el metano está aumentando su presencia por razones antropogénicas (debidas a la actividad humana). Además, a este incremento de emisiones se suman otros problemas, como la deforestación, que ha limitado la capacidad regenerativa de la atmósfera para eliminar el dióxido de carbono, principal responsable del aumento antropogénico del efecto invernadero.

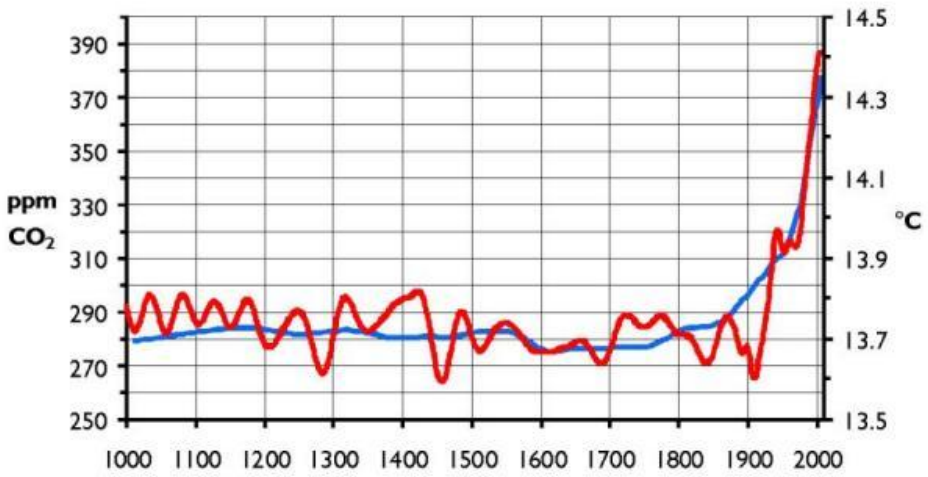


IMAGEN 2.4, CONCENTRACIÓN DE CO_2 EN LA ATMÓSFERA TERRESTRE (AZUL) Y LA TEMPERATURA MEDIA GLOBAL (ROJO), EN LOS ÚLTIMOS 1000 AÑOS.

Respecto a la relación entre la emisión de estos gases y el aumento de la temperatura global del planeta el IPCC indica que "*La mayoría de los aumentos observados en las temperaturas medias del globo desde la mitad del siglo XX son muy probablemente debidos al aumento observado en las concentraciones de GEI antropogénicas*.".

3 Tecnologías Utilizadas

A continuación se explicarán algunos conceptos sobre las tecnologías más relevantes utilizadas en la realización de este proyecto.

3.1 Microsoft Silverlight.

Silverlight es una nueva tecnología de presentación web creada por Microsoft para su ejecución en distintas plataformas. La primera versión de Silverlight fue presentada en septiembre de 2007 y actualmente existe la versión 1.1 también llamada Silverlight 2.
Es el predecesor de WPF (Windows Presentation Fundation) y como este, utiliza el lenguaje **XAML** (e**X**tensible **A**pplication **M**arkup **L**anguage).

Pero se ha de saber que Silverlight no es solo un atractivo lienzo para mostrar contenido web y multimedia. Es también una potente y a la vez que ligera plataforma para desarrollar aplicaciones web portables y multiplataforma. A de más, Silerlight da la posibilidad de construir interfaces para aplicaciones web muy superiores comparadas con las tradicionales.

Como se ha comentado antes, actualmente existen dos versiones diferentes de la plataforma Silverlight:

- **Silverlight 1.0.** Cuenta con las características de presentación propias de Silverlight y utiliza JavaScript para el código procedural.
- **Silverlight 2.** Combina las características de presentación de Silverlight 1.0 con librerías Silverlight y capacidades propias del Framework .Net.

Así pues, básicamente se puede definir Silverlight como una plataforma para el desarrollo y diseño de aplicaciones web, cuyas soluciones están formadas por un conjunto de archivos XAML que soportan tras de si código en **lenguajes .Net** (C#, Visual Basic .Net,…) a partir de su versión 1.1, lo cual otorga a esta plataforma una gran potencia y diferentes funcionalidades.

Silverlight se puede ejecutar en todos los entornos: con exploradores y en múltiples dispositivos y sistemas operativos de escritorio. Solamente es necesario instalar un pequeño plug-in de unos 2 MB que se puede instalar cuando el usuario entra en algún sitio web donde se utilice Silverlight.

Una de sus bazas es que su visualización no depende de la plataforma donde se ejecute o del explorador que se use, esto hace que una vez diseñada la aplicación no sufra cambios visuales dependiendo del navegador utilizado. Permite al diseñador web expresar su trabajo en un formato que funciona directamente en web como es el XAML. A de más, permite que el desarrollador incorpore este XAML creado por el diseñador directamente en una página web, de manera que pueda cambiar la apariencia de la aplicación sin tener que compilar, simplemente la próxima vez que se vea la pagina se cargará el nuevo XAML.

El entorno de ejecución en el cliente de la aplicación es simple y compacto, simplemente un archivo **.xap** que se descarga al acceder a la aplicación. Pero la arquitectura de la plataforma de desarrollo no es tan sencilla. Integra un número de características y tecnologías complejas. A grandes rasgos, la arquitectura de esta plataforma consta de dos partes:

1. **El Framework de presentación**. Componentes y servicios orientados hacia la interfaz de usuario y la interacción con este.
2. **El Framework .Net para Silverlight**. Un subconjunto del Framework .NET que contiene componentes y bibliotecas, incluyendo la integración de datos, controles extensibles de ventanas, interconexión, bibliotecas de clases base, y el common language runtime (CLR).

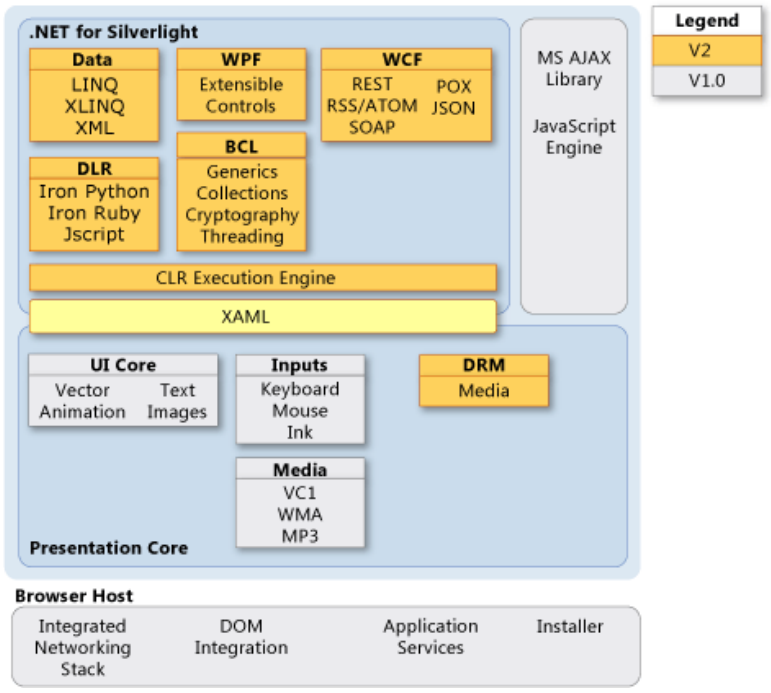


IMAGEN 3.1, COMPONENTES DE LA ARQUITECTURA SILVERLIGHT.

Desarrollar con Silverlight una aplicación sencilla, simplemente con botones, es algo muy sencillo. Para Silverlight 1.0, desarrollar una aplicación que tuviese algo más que botones era algo complejo, debido a que dicha versión no dispone de los widgets propios de cualquier librería gráfica, como pueden ser combo boxes, slayers, radio bottons, etc. Esto supone que si el usuario quiere utilizar este tipo de widgets los ha de programar, cosa que hace mucho más compleja la creación de cualquier aplicación.

En el momento en que salió Silverlight 2, este tampoco contaba con estos componentes gráficos. Pero en Marzo del 2008, poco después de la salida de Visual Studio 2008, Microsoft sacó **Silverlight Tools Beta 1 for Visual Studio 2008**. Un add-on para Visual Studio 2008 con componentes gráficos para Silverlgiht 2 y con previsualizaciones de archivos XAML. Esto supone un gran avance en otorgar a esta tecnología facilidad de uso y más potencia.

Cuando creamos una aplicación Silverlight con Visual Studio, el proyecto, a parte de los archivos de código trasero escritos en lenguajes .Net, incluye por defecto dos archivos xaml: Page.xaml y App.xaml.

El archivo App.xaml se suele usar para declarar recursos, como objetos brocha y estilo que se usarán en toda la aplicación. La clase del código trasero de App.xaml se puede

usar para administrar eventos del nivel de aplicación - como Application_Startup, Application_Exit y Application_UnhandledException.

El archivo Page.xaml es el control inicial que se carga cuando se activa la aplicación. En él podemos usar los controles que definirán nuestra interfaz de usuario, y administrará los eventos de éstos en las clases del código trasero.

Silverlight funciona de la siguiente manera:
La aplicación de Silverlight comienza por invocar el control de Silverlight que se encuentra en una página HTML. Es decir, el navegador llama a dicha página la cual tiene incrustado un objeto o aplicación Silverlight, en ese momento el cliente se descarga un archivo .xap creado al compilar el proyecto y carga la aplicación.
Una vez cargada, el usuario puede interactuar con los elementos, mediante eventos de ratón o de teclado. Estos eventos son recogidos e invocan a diferentes funciones codificadas en scripts, javascripts para la versión 1.0, o codificadas lenguajes .Net como C# para la versión 2. En estos scripts o clases el desarrollador puede escribir los controladores de los diferentes eventos, realizar cualquier tipo de operación e incluso modificar la apariencia del mismo XAML.

3.1.1 XAML.

XAML es un lenguaje declarativo basado en XML, optimizado para describir gráficamente interfaces de usuarios visuales ricas desde el punto de vista gráfico.
Más concretamente, los archivos XAML son archivos XML que usamos para especificar el aspecto de la interfaz de usuario de una aplicación Silverlight o WPF. También podemos usar XAML para representar objetos .NET.

Los archivos tipo XAML se producen con una herramienta de diseño visual, como **Microsoft Expression Blend**, la cual permite crear archivos XAML de manera sencilla para el usuario inexperto ya que es muy visual e intuitivo. De hecho, diseñar XAMLs no demasiado complejos es tan sencillo como dibujar con alguna herramienta como Paint.

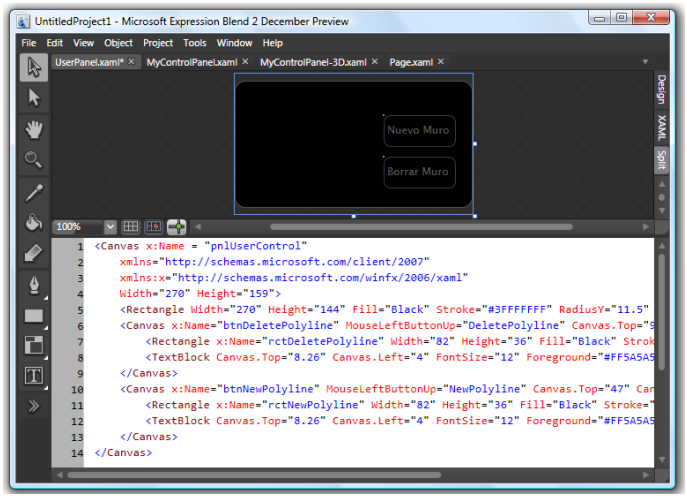


IMAGEN 3.2, MICROSOFT EXPRESSION BLEND.

Con el add-on Silverlight Beta Tools para Visual Studio, los archivos XAML también pueden ser editados mediante el editor Visual Studio 2008, el cual a de más de reconocer el código XAML también cuenta con previsualización.

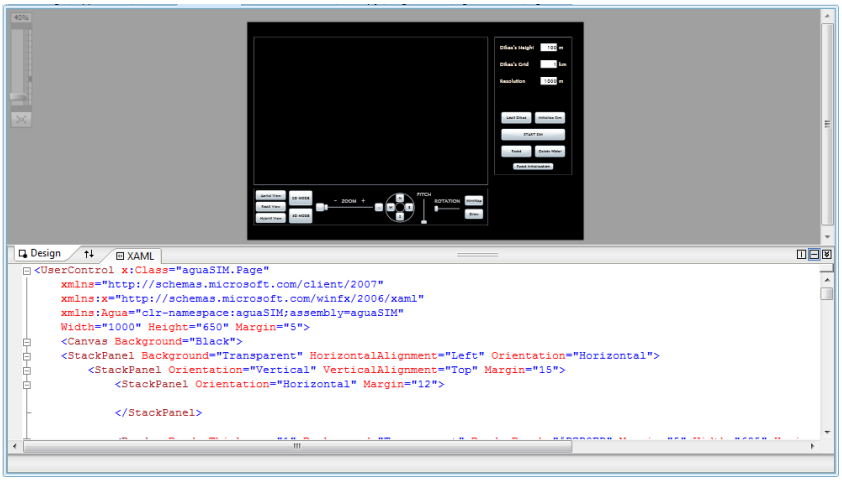


IMAGEN 3.3, MICROSOFT VISUAL STUDIO, EDITOR Y VISUALIZADOR DE XAML.

3.1.2 XAP.

Cuando compilamos un proyecto Silverlight 2, Visual Studio compila el código y el .xaml en un assembli estándar de .NET, y empaqueta todos los recursos estáticos (como imágenes, archivos estáticos) en el archivo .xap. En su manera más básica, el archivo xap contiene los siguientes ficheros:

- ***AppManifest.xaml***
- ***<nombreProyecto>.dll***

- ***System.Windows.Controls.dll***
- ***System.Windows.Controls.Extended.dll***

Los archivos ".xap" (se pronuncia "*zap*") usan el algoritmo estándar .zip para minimizar el tamaño de descarga a cliente. Una aplicación "Holla mundo" en Silverlight (con VB o C#) ocupa unos 4KB. Esto hace que la descarga sea ligera y muy rápida.

Para hostear y ejecutar una aplicación Silverlight 2, podemos añadir el tag <object> en cualquier página html estándar (no hace falta JavaScript) que apunte al archivo .xap. Silverlight descargará el archivo xap, lo instanciará, y lo hosteará en la página del navegador. Esto funciona sin importar el navegador (Safari, Firefox, IE, etc) y sin importar el sistema operativo (Windows, Mac y Linux).

3.1.3 Silverlight Streaming.

Microsoft Silverlight Streaming de Windows Live es un servicio que acompaña a Silverlight . Es un servicio gratuito para personas que desean incluir contenido y aplicaciones basados en Silverlight de manera rápida y sencilla en sus sitios web.

Este servicio ofrece hasta 4 Gigabytes de hospedaje gratuito y un ancho de banda prácticamente ilimitado. Las aplicaciones Silverlight -que pueden ser creadas usando Expression Blend 2, Expression Design, Expression Encoder o Visual Studio 2008- una vez subidas a este servicio, pueden integrarse en otros sitios, permitiendo que el contenido en Silverlight corra en múltiples navegadores y sistemas operativos.
El único límite es de 20 Mb por video (algo así como un video de 10 minutos). Para acceder al servicio solo es necesario contar con una cuenta de Windows Live.

3.2 Microsoft Virtual Earth.

Virtual Earth es una plataforma de mapas geoespaciales on-line creada por Microsoft. Es una plataforma de reciente creación, la primera versión completa salió a mediados de 2007.

Se puede definir como "*Web Maping Service*" o **Servicio Web de Cartografía**, un conjunto integrado de servicios basados en web el cual permite al desarrollador crear aplicaciones on-line integrando en estas componentes cartográficos como mapas o imágenes geoespaciales.

Es el futuro sucesor de la plataforma de Microsoft MapPoint Web Service y el competidor del más conocido Google Maps.

IMAGEN 3.4, VIRTUAL EARTH 2D. ESTADOS UNIDOS.

Desde el punto de vista del usuario, a de más de presentar una buena calidad de imagen y un entorno atractivo, Virtual Earth tiene las siguientes funcionalidades:

- **Diferentes tipos de mapa**: el usuario puede cambiar en cualquier momento de tipo de mapa que quiere ver, pudiendo seleccionar entre Road (mapa de político de carreteras), Aerial (formado por imágenes aéreas), Hybrid (una

mezcla de los dos anteriores) o Bird's Eyes (fotografías realizadas a gran altura pero no de manera perpendicular al suelo).

- **Diferentes modos de visión**: la gran ventaja de Virtual Earth sobre sus competidores, es que permite ver los mapas en modo 3-D en el web browser simplemente instalando un pequeño plug-in. A de más, tiene disponibles las edificaciones en 3D de una serie de ciudades importantes a nivel mundial (sobre todo de EE.UU).
- **Manejo simple de los mapas:** de manera sencilla el usuario puede desplazar el mapa, acercar o alejar el punto de vista y en modo 3-D puede cambiar la inclinación de la cámara. Todas estas acciones se pueden realizar tanto desde el teclado como con la ayuda del ratón.

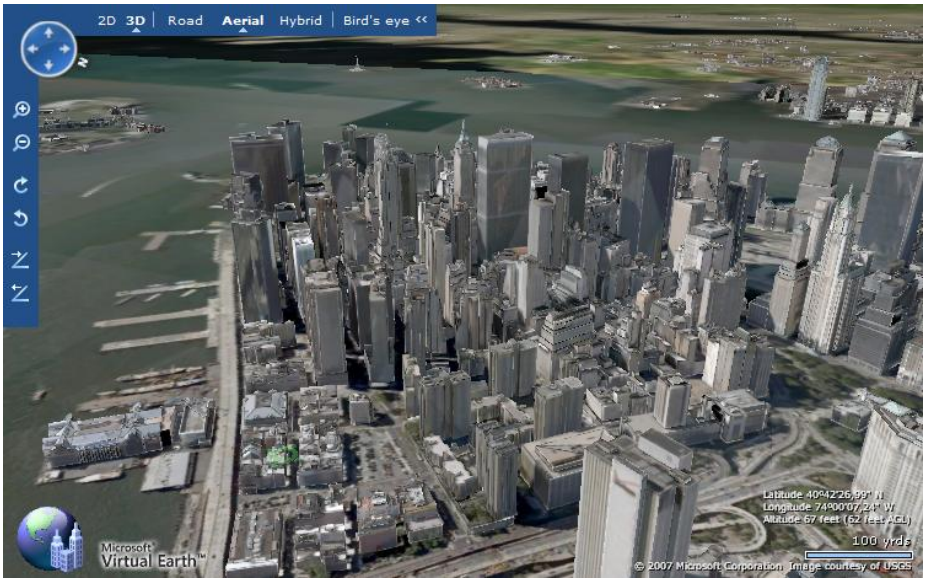


IMAGEN 3.5, VIRTUAL EARTH 3D. NUEVA YORK.

Desde el punto de vista del desarrollador, Virtual Earth consta de una API llamada Virtual Earth Map Control que permite al desarrollador realizar sus peticiones vía **javascript** sobre un mapa que es un objeto **AJAX**. De esta forma no se ha de cargar toda la página cada vez que realizamos algo sobre el mapa, sino que simplemente se vuelve a cargar este. A de más, consta de una buena documentación de todas las clases y funciones disponibles a utilizar, disponible tanto para descargar (VirtualEarthSDK 6.0) como para consultar on-line desde la pagina de Microsoft MSDN.

Licencia.

Virtual Earth forma parte de la plataforma Microsoft Windows Live, que integra también, entre otras cosas, Silverlight Streaming.

Microsoft permite el acceso a todos los servicios de la plataforma mediante una única y sencilla forma de calcular el coste, basándose en el número de usuarios únicos (UU) que acceden a la web.
Virtual Earth tiene una excepción, es totalmente gratuito si no se superan las 3 millones de peticiones por mes. Los sitios que superan los límites establecidos por el servicio gratuito, han de realizar un acuerdo comercial con Microsoft por 0,25 dólares americanos por usuario único al año.

3.3 Sistema de Información Geográfica: Idrisi32.

3.3.1 Concepto de SIG.

El término **SIG** procede del acrónimo de **Sistema de Información Geográfica**, también conocido por sus siglas en inglés **GIS** (Geographic Information System).

Técnicamente se puede definir un SIG como una tecnología de manejo, manipulación, análisis, modelamiento y representación de información geográficamente referenciada, formada por equipos electrónicos (hardware) programados adecuadamente (software) que permiten manejar una serie de datos espaciales (información geográfica) y realizar análisis complejos con éstos.
Es decir, un sistema para almacenar, controlar, manipular, analizar y visualizar datos referidos a lugares.

El elemento diferenciador entre los SIGs y el resto de sistemas de información radica en información geográfica y en la doble vertiente de la información almacenada, por una parte la vertiente espacial y por otra la temática. Es decir, mientras otros sistemas de información contienen solo datos alfanuméricos, las bases de datos de los SIGs contienen la delimitación espacial de cada elemento geográfico.
Por tanto, el SIG tiene que trabajar a la vez con ambas partes de información: con cartografía y con bases de datos, uniendo ambas partes y constituyendo con todo ello una sola base de datos geográfica**.** De esta manera, puede mapear cualquier información almacenada en base de datos que tenga un componente geográfico, permitiendo ver patrones, relaciones y tendencias que no pueden verse en un formato de tabla o lista y dando una nueva perspectiva a la información.

La construcción de una base de datos geográfica implica un proceso de abstracción para pasar de la complejidad del mundo real a una representación simplificada asequible para el lenguaje de los ordenadores. Esto se consigue estructurando la

información en diferentes capas temáticas las cuales se almacenan de manera independiente.

Dependiendo de cómo sea este tipo de estructura para almacenar la información, se pueden diferenciar dos tipos de SIG: raster y vectorial.

- **Raster:** se basan en la relación de vecindad entre los objetos. Para ello divide el espacio en celdas regulares y atribuye un valor numérico a cada una de ellas, lo cual representa su valor temático. Por lo tanto, contra más grandes sean las celdas, se obtendrá una menor precisión.

  Es útil cuando tenemos que describir objetos geográficos con límites difusos donde el contorno no es totalmente nítido, como por ejemplo puede ser la dispersión de una nube de contaminantes.
- **Vectorial:** son aquellos que para la descripción de los objetos geográficos utilizan vectores definidos por pares de coordenadas relativas a algún sistema cartográfico.

  Es adecuado cuando trabajamos con objetos geográficos con límites bien establecidos, como pueden ser fincas, carreteras, etc.

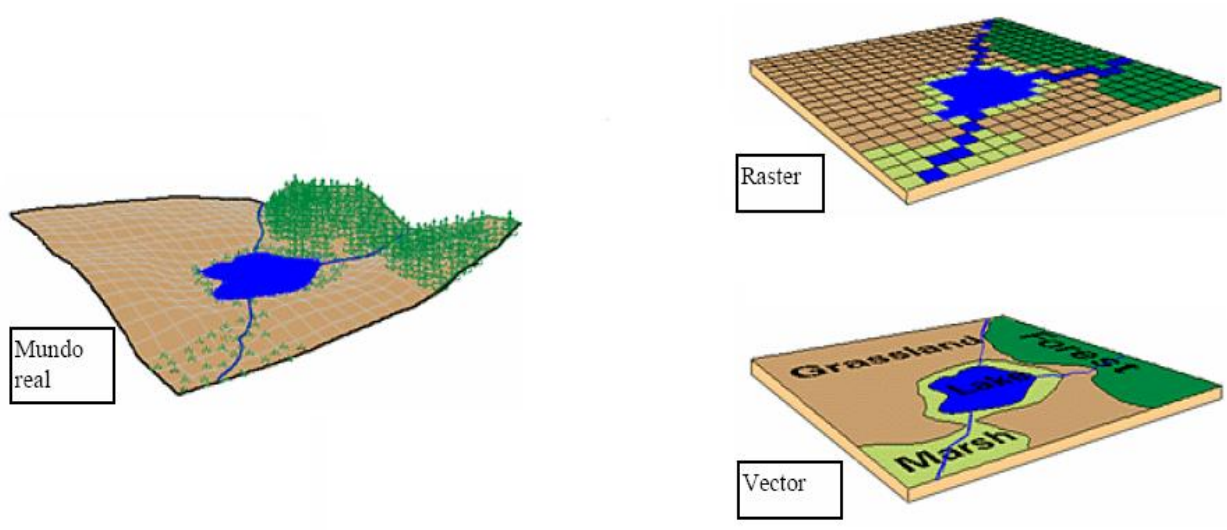


IMAGEN 3.6, REPRESENTACIONES SIG.

Entre las funciones que un SIG es capaz de realizar se pueden destacar las siguientes:

- **Localización:** preguntar por las características de un lugar concreto. (*¿Qué hay en...?*)
- **Condición:** el cumplimiento o no de unas condiciones impuestas al sistema. (*¿Dónde sucede que...?*)
- **Tendencia:** comparación entre situaciones temporales o espaciales distintas de alguna característica. (*¿Qué ha cambiado...?*)
- **Rutas:** cálculo de rutas óptimas entre dos o más puntos. (*¿Cuál es el camino óptimo…?*)
- **Pautas:** detección de pautas espaciales. (*¿Qué pautas existen...?*)
- **Modelos:** generación de modelos a partir de fenómenos o actuaciones simuladas. (*¿Qué ocurriría si...?*)

3.3.2 Idrisi32.

Una de las funcionalidades de este proyecto es el hecho de poder dar al usuario la información generada por el simulador. Para ello se ha decidido exportar esta información a un formato estandarizado de datos SIG. El formato elegido ha sido Idrisi32.

Idrisi32 es un SIG de tipo **raster**. Desarrollado por Clark Labs, perteneciente a la escuela de Geografía de la Universidad de Clark (EE.UU.) y la versión inicial data de 1987.

En el formato de datos de Idrisi32[4] cada capa de información se traduce a dos ficheros:

- **Fichero de datos:** contiene los datos numéricos que representan la temática de esa capa. La estructura del fichero es una simple lista de valores que representan una matriz definida en el fichero de meta-datos. En nuestro caso, por ejemplo, tendremos un fichero de datos donde se nos indique la altura a la que queda el nivel del mar para cada celda del raster.

[4] Mirar Anexo I: Formato Idrisi32 para obtener información más exhaustivo sobre el formato utilizado.

- **Fichero de meta-datos:** en este fichero se escribe la información necesaria para comprender y entender los datos, como por ejemplo las dimensiones de la matriz que representa el fichero de datos.

Estos dos ficheros deben tener el mismo nombre, pro distinta extensión. El fichero de meta-datos tiene extensión *.doc mientras que el de datos tiene extensión *.img .

Idrisi32 también reconoce archivos en **formato vectorial**. De hecho en este proyecto se utiliza dicho formato para permitir al usuario cargar diques desde un archivo plano.
En este formato, los datos se disponen en dos columnas que representan latitud y longitud respectivamente. El vector o línea tendrá tantos puntos como pares de datos se encuentren en el archivo antes de encontrar el par ”0 0” lo que indica final de línea. Dando comienzo a la siguiente o bien a la finalización del fichero.

4 Planificación y Costes.

La planificación es una herramienta imprescindible en cualquier proyecto de ingeniería que se precie, aunque a nivel informático está infravalorada debido a la gran dificultad que supone estimar en este sector el tiempo y los recursos a emplear en las diferentes tareas.

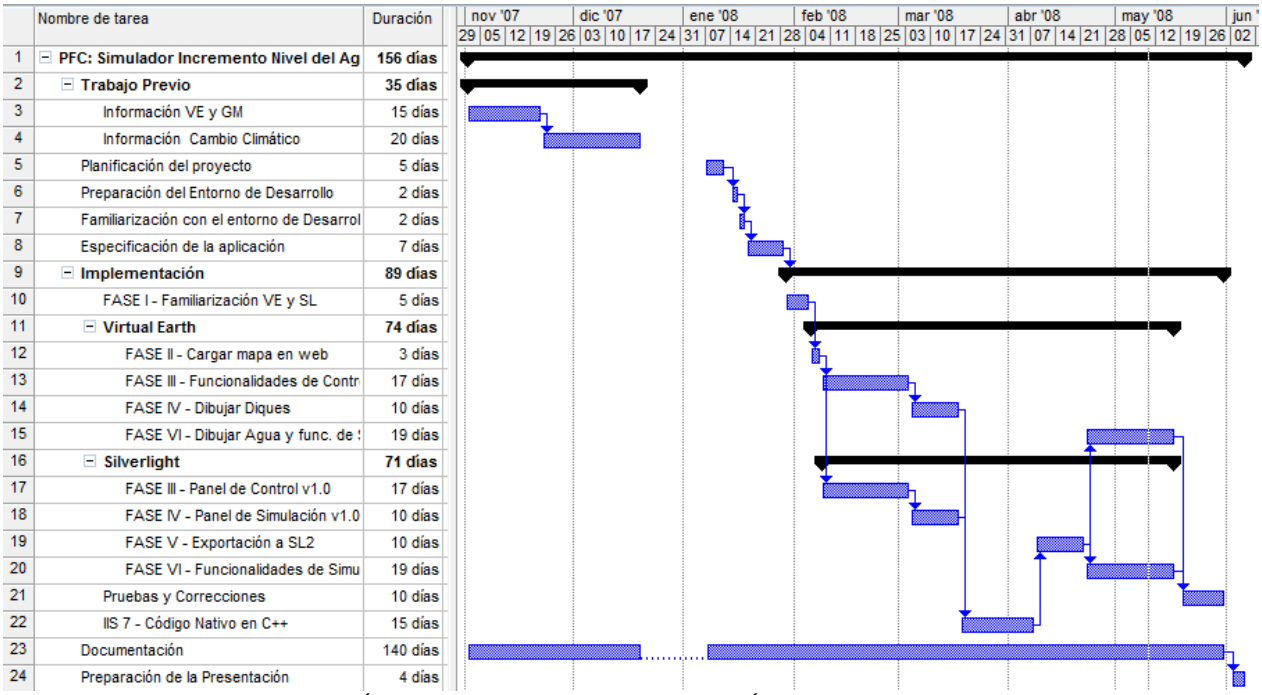
Realizar planificaciones buenas es una faena muy complicada, ya que la persona o conjunto de personas que realicen la planificación han de tener una experiencia amplia en proyectos similares y han de conocer bien las capacidades del personal que va a realizar el desarrollo. Así pues, hacer una planificación precisa es algo que no se da demasiadas veces.

Por eso lo que se tiene que tener siempre en mente es que la planificación no es algo estricto y sólido que se ha de cumplir a raja-tabla, si no que ha de ser la herramienta que nos sirva de guía para llegar a conseguir nuestro objetivos en el tiempo estimado, marcando hitos tanto a corto como a medio o largo plazo. Para ello, la planificación ha de ser algo dinámico, capaz de ser adaptada a las condiciones de cada momento.

En el caso de este proyecto, como era de esperar, la planificación inicial no se ha seguido estrictamente, sino que se ha ido adaptando a lo largo del desarrollo de este.
Debido a la aparición de dificultades en diferentes tareas, a la aparición de nuevos objetivos en el proyecto y a la falta de experiencia tanto en lo que a planificación se refiere como a la hora de desarrollar en algunas de las tecnologías utilizadas, la planificación ha sufrido cambios, adaptándose para marcar los nuevos objetivos y desestimar los obsoletos, eliminando ciertas tareas e incorporando otras tantas que no se previeron en la primera planificación.

4.1 Desarrollo del Proyecto.

La planificación comprende de principio de Enero a finales de Mayo. La idea de hacer una planificación muy ajustada para acabar antes de Junio se hizo con la premisa de que, debido a la inexperiencia y a que el proyecto a desarrollar es novedoso, era de esperar que alguna tarea se alargara más de lo convenido y había una gran probabilidad de que surgiesen problemas que hiciesen modificar aspectos del proyecto. Para que no se alargase más allá de los plazos estipulados de lectura de proyectos, se creyó conveniente ajustar la planificación. De esa manera, aunque se han presentado problemas, se ha tenido el suficiente tiempo de margen para reaccionar y llegar no demasiado apurado a la entrega.



	Nombre de tarea	Duración
1	**PFC: Simulador Incremento Nivel del Ag**	**156 días**
2	**Trabajo Previo**	**35 días**
3	Información VE y GM	15 días
4	Información Cambio Climático	20 días
5	Planificación del proyecto	5 días
6	Preparación del Entorno de Desarrollo	2 días
7	Familiarización con el entorno de Desarrol	2 días
8	Especificación de la aplicación	7 días
9	**Implementación**	**89 días**
10	FASE I - Familiarización VE y SL	5 días
11	**Virtual Earth**	**74 días**
12	FASE II - Cargar mapa en web	3 días
13	FASE III - Funcionalidades de Contr	17 días
14	FASE IV - Dibujar Diques	10 días
15	FASE VI - Dibujar Agua y func. de !	19 días
16	**Silverlight**	**71 días**
17	FASE III - Panel de Control v1.0	17 días
18	FASE IV - Panel de Simulación v1.0	10 días
19	FASE V - Exportación a SL2	10 días
20	FASE VI - Funcionalidades de Simu	19 días
21	Pruebas y Correcciones	10 días
22	IIS 7 - Código Nativo en C++	15 días
23	Documentación	140 días
24	Preparación de la Presentación	4 días

IMAGEN 4.1, VISIÓN GENERAL DE LA PLANIFICACIÓN FINAL DEL PROYECTO.

Aunque se realizaron algunas tareas del proyecto durante los meses de Octubre y Diciembre del 2007, no fue hasta finales de Enero del 2008 cuando verdaderamente se comenzó a trabajar constantemente.

El tramo comprendido entre Octubre del 2007 y Diciembre del 2007 se dedicó a hacer **tareas previas** asociadas al proyecto. En concreto dos:

1.- Información sobre el Cambio climático.

Esta tarea consistió en informarse sobre el Cambio Climático, que es la razón principal por la cual nuestro proyecto tiene sentido.

2.- Información sobre *Web Mapings Services.*

Esta tarea, mucho más imprescindible e importante que la primera ya que marcará el transcurso del proyecto. Consistía en informarse sobre los principales WMS y después de conocerlos y compararlos elegir el más conveniente para el desarrollo del proyecto.

Estas dos tareas se realizaron de una manera muy lenta debido a que fue un tramo de tiempo de transición entre las clases y la dedicación al proyecto.

En Enero de 2008 se retoma el proyecto y lo primero que se realiza es una primera versión de la planificación. Esta **primera planificación** se realiza durante una semana, intentando ver los puntos más conflictivos en el desarrollo y las tareas que podrían ocasionar más problemas. Aunque se hizo con tiempo y teniendo en cuenta muchos aspectos, pronto quedaría obsoleta.

Tras la realización de la planificación, se dio paso a la **preparación del entrono y familiarización** con este, que duraría 4 días. Esto constaba de la instalación y utilización de Windows Vista, Microsoft Visual Studio 2008, Microsoft Expression Blend, Microsoft Silverlight y Microsoft Virtual Earth 3D.

Durante la siguiente semana y media se realizó una primera versión de la **especificación** de la aplicación a desarrollar. En esta especificación se relataron los principales casos de uso y los requisitos funcionales y no funcionales del sistema. Posteriormente, durante las últimas semanas del proyecto se retomó este escrito para completarlo con los casos de uso que no se previeron en su momento.

A finales de Enero se comienza con el grupo de tareas principal del proyecto, la **implementación**. Desde un primer momento ya se sabía que en estas tareas surgirían problemas y por ello ya se estimo unos tiempos para el desarrollo largos. Aun así los

tiempos que se estimaron en muchos casos fueron incorrectos. Llegando al punto en que algunos la gran parte de las tareas han cambiado con respecto a la primera planificación realizada.

La primera tarea a hacer en la implementación fue la **familiarización con las tecnologías** que se iban a usar. En este caso, se dedicaron 5 días para mirar ejemplos de aplicaciones realizadas con Silverlight o Virtual Earth, para mirar sus respectivos SDKs y para realizar algo de código de ejemplo.

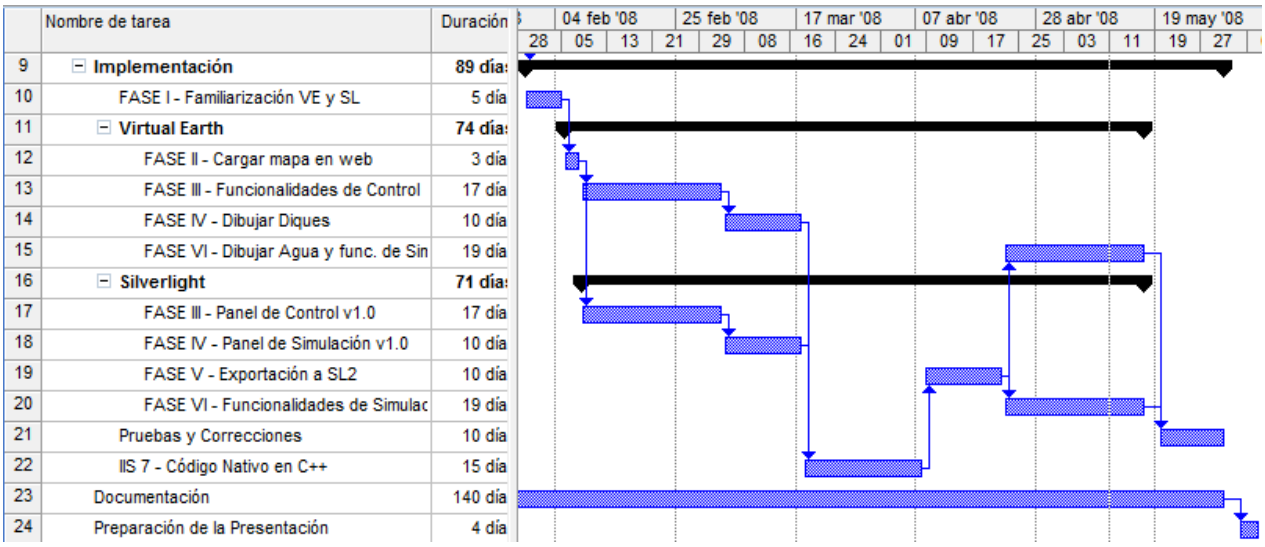


IMAGEN 4.2, FASES DE LA IMPLEMENTACIÓN.

Seguidamente se realizó la **Fase II**, donde se debía crear una web donde hubiese un mapa de Virtual Earth. Era lo primero que se realizaba con Virtual Earth en el proyecto y aunque era una tarea muy sencilla fue costosa. En su momento se tardaron casi 3 días, mientras que a día de hoy se tardaría un par de horas.

La **Fase III** comprendía una parte mucho más compleja que lo que se había realizado hasta el momento. Se debía realizar un panel de control con Silverlight que controlase los movimientos sobre el mapa Virtual Earth. Esta tarea fue muy costosa sobre todo por la parte de Silverlight, ya que se utilizó Silverlight 1.0 el cual no tiene ningún control implementado, así que para crear botones se tuvieron que dibujar y programar su funcionamiento. Se emplearon 17 días hábiles entre crear este panel en Silverlight y realizar todas las funciones de control sobre el mapa en javascript.

La **Fase IV** consto también de partes con Silverlight y con Virtual Earth. Se creo otro panel de botones donde se controlarían los procesos de la simulación, entre ellos el hecho de dibujar diques de contención. Aunque a primera vista podía resultar más complejo que la fase anterior, gracias a la experiencia ya adquirida su tiempo de desarrollo fue inferior a la fase III. Aun así, se emplearon 10 días en vez de los cinco que se habían planificado.

Una vez se llegó a este punto, la intención era de realizar unas funciones en C++ para que llevasen a cabo todo el proceso de simulación, incluido un motor de simulación. Para ello se había pensado en conectar la aplicación web con una dll en C++ alojada en un servidor mediante ISAPI. Al constatar que ISAPI esta obsoleto, se decidió utilizar la alternativa que sucede a ISAPI, que es realizar código nativo para **Internet Information Server 7**. En esta tarea se emplearon quince días sin obtener ningún resultado positivo, debido a la falta de conocimientos y tiempo. Por ello se decidió cambiar de estrategia. Se determinó que se cambiaría la aplicación web, exportándola a la versión 2 de Silverlight, que aun estando en fase beta, dispone de la capacidad para realizar tras la capa visible código en c#. Así se aprovecharía este C# para realizar las funciones previas a la simulación, y de mientras se intentaría mirar de encontrar al solución para avanzar en el código nativo para ISS7, cosa que poco más adelante se descartó por completo dentro del ámbito de este proyecto, debido a que era preferible realizar una aplicación consistente con Silverlight 2 que no dejarla en Silverlight 1.0 y realizar el motor en C++.

Aquí se puede ver, como la falta de experiencia y conocimiento condiciona la planificación y el correcto desarrollo de un proyecto, ya que esto se debería de haber previsto cuando se realizo la primera planificación o incluso al realizar la especificación de la solución. Sin en cambio este hecho hizo perder tres semanas de trabajo.

En la **Fase V** se cambió toda la aplicación de la versión 1.0 a la versión 2 de Silverlight. Esta versión ya contaba con controles implementados, con lo cual se podía obtener una solución muy superior visualmente. A parte permitiría poner por detrás código en

c# y no solo javascript, otorgando una gran potencia. Antes de empezar se había estimado unos 15 días para su realización, pero solo se necesitaron 10. La verdad, es que teniendo la base de haber implementado en la versión anterior y teniendo disponible el plug-in para Visual Studio Silverlight Tools Beta 1, la realización de esta tarea fue mucho más sencilla de lo esperado.

La siguiente fase, la **Fase IV**, implicó también trabajar con ambas tecnologías, VE y SL2.
En la parte de VE, lo que se debía de hacer era que fuese capaz de representar el agua. Esto fue algo muy sencillo, debido a que se había realizado algo similar cuando se programó el dibujo de los diques. Así que este apartado se realizó en un tiempo mínimo, 1 día. También se tenía que dividir el mapa en celdas, cosa que no fue tan sencilla. En esto se emplearon unos 5 días.
Sin en cambio la parte de SL2 no fue tan fácil, debido al desconocimiento de C# y a la cantidad de cosas que se tuvieron que hacer, como el uso de isolated storage, la utilización de un web service asíncrono y la representación de la información del mapa en formato idrisi32. Para todo esto se utilizaron 14 días, sumando un total para la Fase IV de 19 días hábiles.

Los últimos diez días de la tarea de implementación se dedican a realizar **pruebas** para verificar el correcto funcionamiento de la aplicación y corregir los errores que surgen. Esto implica retoques en todas las partes de la implementación, ya que aunque se fueron probando progresivamente durante el desarrollo, no fue hasta este punto cuando se hicieron unas pruebas más exhaustivas.

La **documentación** se ha ido realizando a medida que se avanzaba el proyecto, de manera continuada. Aun así, las últimas semanas se le ha dado un énfasis muy importante a la realización de esta. Es importante el hecho de haber realizado parte de la documentación durante todo el proyecto ya que si se hubiese dejado para escribir toda la en la fase final no se hubiese llegado a tiempo a la entrega.

4.2 Valoración económica.

El número de horas empleadas en la elaboración del proyecto no ha sido la misma cada día, debido a compromisos externos al proyecto.

Durante la primera parte del proyecto, en la que se realizaron trabajos previos, se dedico una media de 2 horas diarias, debido a que entre clases y trabajo no se disponía de más tiempo.
Así pues, 35 días a 2 horas, haciendo un total de 70 horas empleadas.
Durante el resto de proyecto la dedicación fue muy superior. Como no se disponía de suficiente tiempo durante los días laborables, lo que se hizo fue trabajar en le proyecto 4 horas diarias entre semana y dedicar 10 más durante el fin de semana. Esto supone una media de 6 horas por día laborable, dando un total de 630 horas trabajadas durante esta parte (105 días x 6 h/día).

Esto da un total de **700 horas** aproximadamente dedicadas a la elaboración de dicho proyecto.

Para sacar un precio cercano al real del trabajo realizado, de este total de horas, se debería diferenciar entre que rol se estaba desempeñando en cada momento del proyecto. En este caso, para obtener un valor aproximado se procede a ponderar las horas realizadas según el rol:

- Analista: Se ha desempeñado rol de analista en el apartado de la planificación y especificación del proyecto. A parte se ha podido tener este rol para dar soporte en cualquier momento del mismo. Así se estima que el rol de analista ha sido un 30% del tiempo total.
- Programador: El programador ha sido el rol principal durante la fase de implementación y parte de la documentación. Se estima que se ha tenido este rol durante un 70% del tiempo del proyecto.

Suponiendo que un analista cuesta 45€/h y un programador cuesta 30€/h, el coste por mano de obra del proyecto asciende a:

(30€/h x 700h x 0,65) + (45 €/h x 700h x 0,35) = 24.675

A este coste, se le tiene que añadir los costes de las licencias para utilizar el software necesario:

- Microsoft Office 2007 Professional: 429 €.
- Microsoft Visual Studio 2008 Professional Edition: 1.072 €.

Todo esto hace un **total de 26.176€.**

5 Especificación.

La especificación es una herramienta fundamental a la hora de desarrollar cualquier tipo de aplicación informática. Para llegar a la fase de implementación con una idea más concreta de lo que se espera de la aplicación es vital realizar de manera satisfactoria la especificación de esta.

La especificación se dividirá en dos puntos: Requisitos y Casos de Uso.
En los Requisitos, por una parte se definirán las diferentes funcionalidades que tendrá la aplicación y por otra se enumerarán todos los aspectos que vienen predefinidos de por sí que se han de tener en cuenta para el correcto desarrollo de la aplicación.
En los Casos de Uso se definirán las funciones de la aplicación, quien interviene en y que fin tendrá cada una de estas funciones.

5.1 Requisitos.

5.1.1 Requisitos No Funcionales:

1. La aplicación se mostrará bajo una página web. Cosa que la hará muy portable. Se ha de garantizar su correcto funcionamiento sobre los principales brosers web: Internet Explorer y Mozilla Firefox.
2. El apartado visual se llevará a cabo con Microsoft Silverlight. De esta manera se garantiza que la aplicación tenga el mismo comportamiento independientemente del navegador utiliza.
3. La aplicación utilizará los mapas y datos proporcionados por el WMS Microsoft Virtual Earth.
4. Para los datos referentes a la altura de un punto respecto al nivel del mar, los cuales no son proporcionados por el WMS elegido, serán obtenidos mediante el Web Service *Elevation-SRTM3* de geonames.org.

5. Diseño en capas, para obtener un acoplamiento mínimo, y de manera que las funcionalidades referentes a la simulación o a la inicialización de esta puedan ser reutilizadas.
6. Uso sencillo e intuitivo de las diferentes funcionalidades.
7. Se ha de garantizar un tiempo de respuesta óptimo.
8. Para el desarrollo se utilizará el editor Microsoft Visual Studio 2008.
9. Se seguirá la guía de estilo *"Guia d'estil de programación GEA4D.SDK001 versió 1.0".*

5.1.2 Requisitos Funcionales:

1. La aplicación ha de permitir visualizar correctamente los mapas de Virtual Earth. A de más, ha de ser capaz de realizar las principales funcionalidades que presente este como por ejemplo pan, zoom o cambio de tipo de vista de manera sencilla para el usuario.
2. La aplicación ha de permitir al usuario dibujar polilíneas sobre los mapas con la utilización del ratón, para simular la construcción de diques de contención.
3. La aplicación ha de permitir la inserción de diques a partir de la carga de archivos planos.
4. La aplicación ha de permitir al usuario seleccionar la altura a la que quiere que se eleven los diques.
5. La aplicación ha de permitir al usuario elegir la resolución de la malla en la que se dividirá el mapa antes de realizar la simulación.
6. Una vez realizada la simulación, la aplicación ha de "pintar" las zonas que se vean inundadas por el incremento del nivel del mar.
7. La aplicación ha de permitir visualizar los resultados de la simulación también en la vista 3D de Virtual Earth de manera satisfactoria.
8. La aplicación ha de exporta la información previa a la simulación a archivos con formato idrisi32 para ser utilizados por el simulador.

5.2 Casos de Uso.

Podemos dividir las funcionalidades de nuestra aplicación en dos grupos. Un grupo sería el formado por todas aquellas funcionalidades que se realizan estrictamente sobre los mapas de Virtual Earth, como puede ser los diferentes movimientos sobre el mapa que se pueden realizar (pan, zoom, rotación,…) o los distintos modos de visualización que se da a elegir.

Por otra parte, tenemos las propias de nuestra aplicación, más relacionadas con la lógica de la simulación que se quiere llevar a cabo, como puede ser el dibujo de diques, la inicialización previa a la simulación, etc.

5.2.1 Funcionalidades Virtual Earth:

Estas funcionalidades son las relacionadas estrictamente con los mapas Virtual Earth. La mayoría de ellas son relacionadas con el desplazamiento o la visión de los mapas y presentan unos casos de uso muy sencillos.

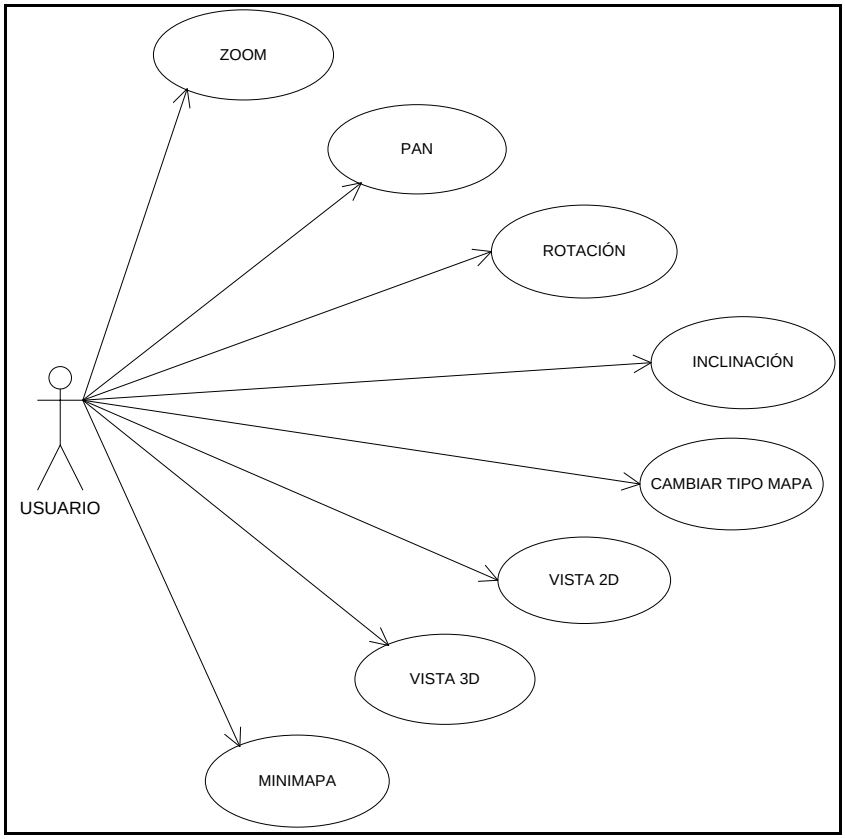


IMAGEN 5.1, DIAGRAMA CASOS DE USO 1.

Casi todas estas funcionalidades se pueden realizar con el ratón pulsando sobre el mapa o bien utilizando los controles creados en la interficie de la solución para gestionar cada uno de estos eventos.

1. Zoom:

Caso de uso: Zoom.

Actores: Usuario.

Objetivo: Acercar o alejar el punto de vista del usuario respecto al mapa para observar el mapa con más detalle o para tener una visión más general de este.

Descripción: El usuario realiza la acción zoom+ o zoom– para aproximar la cámara al mapa o alejarla.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón de zoom pertinente: + o -. O desliza el slider zoom. O doble clic izquierdo para + y CTRL + doble clic izquierdo para menos.	2. El sistema muestra el mapa con el detalle elegido.
3. Fin del caso de uso.	

2. Pan:

Caso de uso: Pan.

Actores: Usuario.

Objetivo: Desplazar el punto de vista del usuario respecto al mapa: Norte, Sur, Este U oeste para ver diferentes zonas del mapa

Descripción: El usuario desplaza el mapa hacia arriba, abajo, derecha o izquierda.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**

1. El usuario pulsa el botón de pan pertinente: arriba, abajo, izquierda o derecha. O botón izquierdo del ratón pulsado y arrastrar hacia donde se desea.	2. El sistema muestra el mapa desplazado.
3. Fin del caso de uso.	

3. Rotación:

Caso de uso: Rotación del mapa.

Actores: Usuario.

Objetivo: Rotar el mapa, solo en modo 3D.

Descripción: El usuario, para visualizar el mapa desde otra perspectiva, pudiendo rotar el mapa en sentido horario o antihorario.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario desplaza el slider de rotación hasta conseguir el punto que desea. O Ctrl. + desplazamiento de ratón a izquierda o derecha.	2. El sistema muestra el mapa rotado.
3. Fin del caso de uso.	

4. Inclinación:

Caso de uso: Inclinación del punto de vista.

Actores: Usuario.

Objetivo: Inclinar la cámara, solo en modo 3D para ver el mapa desde otra perspectiva.

Descripción: El usuario inclina el punto de vista hacia arriba o hacia abajo.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**

1. El usuario desplaza el slider de inclinación hasta conseguir el punto que busca. O Ctrl. + desplazamiento de ratón hacia arriba o abajo.	2. El sistema muestra el mapa con la inclinación de cámara pedida.
3. Fin del caso de uso.	

5. Cambiar Tipo de Mapa:

Caso de uso: Cambiar Tipo Mapa.

Actores: Usuario.

Objetivo: Cambiar el tipo de mapa. Puede pasar a tipo Road, Aerial o Hybrid.

Descripción: El usuario elige el tipo de mapa que desea ver. Cuenta con tres alternativas: Road, Aerial o Hybrid.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón del tipo de mapa que de sea ver, pudiendo elegir entre Road (de carreteras), Aerial (vista aérea) o Hybrid (híbrido)	2. El sistema muestra el mismo mapa pero en el modo seleccionado, respetando las coordenadas que estaba observando el usuario con anterioridad, el nivel de zoom y en caso de 3D el nivel de inclinación y de rotación.
3. Fin del caso de uso.	

6. Cambiar Tipo de Vista a 2D:

Caso de uso: Cambiar Tipo Vista a 2D.

Actores: Usuario.

Objetivo: Cambiar entre de vista en 3D a vista en 2D.

Descripción: El usuario decide ve el mapa en 2D.

Curso típico de acontecimientos:

Acciones del usuario	Respuestas del sistema
1. El usuario pulsa el botón de vista en 2D. O pulsa el número “2” del teclado, tiendo el foco sobre el mapa.	2. El sistema muestra el mapa en el tipo de vista 2D, respetando las coordenadas vistas con anterioridad y el nivel de zoom. A de más, se habilitan las funcionalidades propias de la simulación, como son la inicialización, la simulación o la carga de diques. Se deshabilitan las funcionalidades propias del modo 3D como la inclinación y la rotación.
3. Fin del caso de uso.	

7. Cambiar Tipo de Vista a 3D:

Caso de uso: Cambiar Tipo Vista a 3D.

Actores: Usuario.

Objetivo: Cambiar entre de vista en 2D a vista en 3D.

Descripción: El usuario decide ve el mapa en 3D.

Curso típico de acontecimientos:

Acciones del usuario	Respuestas del sistema
1. El usuario pulsa el botón de vista en 3D. O pulsa el número “3” del teclado si tiendo el foco sobre el mapa.	2. El sistema muestra el mapa en el tipo de vista 3D, respetando las coordenadas vistas con anterioridad y el nivel de zoom. A de más, se deshabilitan las funcionalidades propias de la simulación, como son la inicialización, la simulación o la carga de diques. Se habilitan las funcionalidades propias del modo 3D como la inclinación y la rotación.
3. Fin del caso de uso.	

8. Mostrar/Ocultar Minimapa:

Caso de uso: Mostrar o ocultar minimapa.

Actores: Usuario.

Objetivo: Hacer visible o invisible en la esquina superior izquierda un pequeño mapa que nos indique la zona del globo donde nos encontramos. Este presenta el mismo centro que el mapa normal pero con un nivel de zoom inferior, lo que hace que se tenga una visión más general de la zona observada.

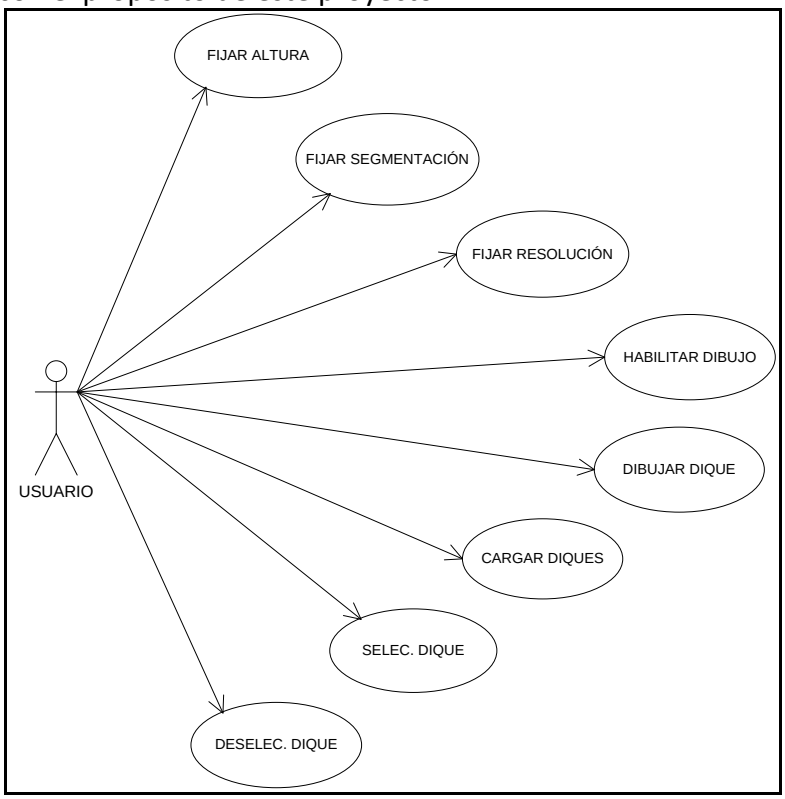
Descripción: El usuario, como carácter informativo, puede hacer visible un pequeño mapa para saber en que parte del globo terráqueo esta. No es posible su realización en modo 3D.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón para habilitar/deshabilitar el minimaza. O pulsa el número "9" del teclado teniendo el foco en el mapa.	2. El sistema muestra un mapa pequeño.
3. Fin del caso de uso.	

5.2.2 Funcionalidades Propias

Este conjunto de funcionalidades son las propias de nuestra aplicación, más relacionadas con el propósito de este proyecto.



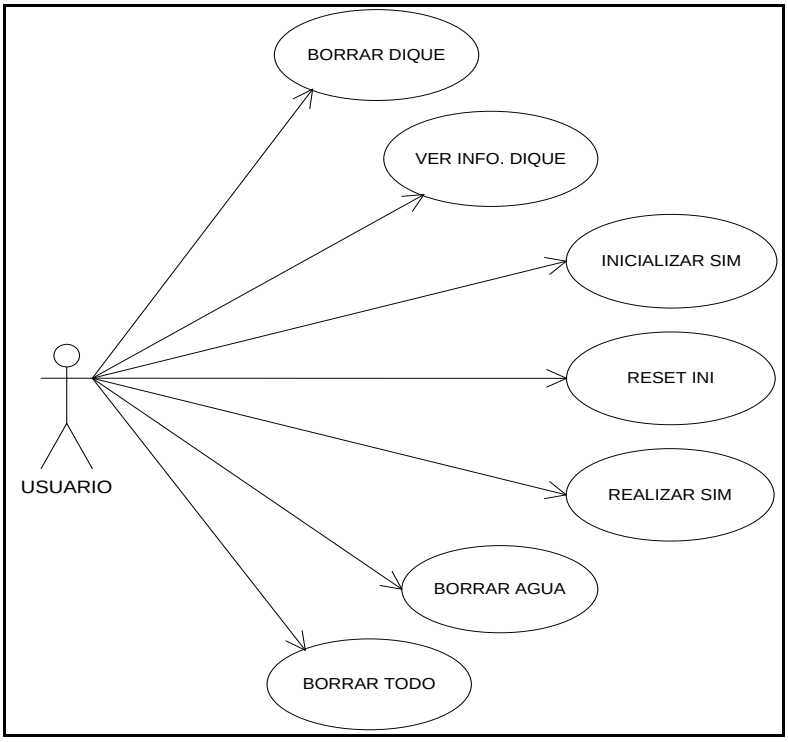


IMAGEN 5.2, DIAGRAMAS CASOS DE USO 2.

1. Fijar Altura de los Diques:

Caso de uso: Fijar altura de los diques de contención.

Actores: Usuario.

Objetivo: Seleccionar la altura a la que el usuario desea dibujar los diques de contención. Los siguientes muros dibujados tendrán esta altura.

Descripción: El usuario rellena el campo de la aplicación referente a la altura de los diques. Al cambiar de valor este se almacena para su posterior uso.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario introduce un valor en el control textedit referente a la altura de los diques.	2. El sistema almacena esa altura, para dibujar los siguientes muros con el valor introducido en el parámetro altura.
3. Fin del caso de uso.	

2. Fijar Segmentación de los Diques:

Caso de uso: Fijar distancia en la que segmentar los diques de contención.

Actores: Usuario.

Objetivo: Fijar un valor para dividir los muros en segmento de ese tamaño. Los siguientes muros dibujados estarán divididos en trozos con longitud igual al valor introducido.

Descripción: El usuario rellena el campo de la aplicación referente a la segmentación de los diques. Al ser cambiado el valor, es almacenado por la aplicación.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario introduce un valor en el control textedit referente a la segmentación de los diques.	2. El sistema almacena ese valor, para dibujar los siguientes muros segmentados cada tantos metros como indique le valor especificado.
3. Fin del caso de uso.	

3. Fijar Resolución:

Caso de uso: Fijar distancia en la que segmentar los el mapa.

Actores: Usuario.

Objetivo: Fijar un valor para dividir lo que se ve del mapa en celdas con dicho valor de longitud. En cierta manera, se fija el valor del nivel de resolución que tendrá la simulación. Cuando se realice la inicialización de la simulación, este valor será utilizado para dividir el mapa en celdas de este tamaño.

Descripción: El usuario rellena el campo de la aplicación referente a la resolución de la simulación. Al ser cambiado el valor, es almacenado por la aplicación.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario introduce un valor en el control textedit referente a la resolución.	2. El sistema almacena ese valor, segmentar la parte visible mapa en la inicialización de la simulación.
3. Fin del caso de uso.	

4. Habilitar/deshabilitar dibujo de diques:

Caso de uso: Habilitar o deshabilitar lo funcionalidad de dibujar diques.

Actores: Usuario.

Objetivo: Que le usuario pueda decidir si quiere habilitar o deshabilitar el modo dibujar, ya que el estar o no en este modo implica un cambio en la realización de algunas otras funcionalidades relacionadas con el mapa, como puede ser el zoom o el desplazamiento realizado con el ratón sobre este.

Descripción: El usuario habilita o deshabilita la funcionalidad de dibujar diques.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón correspondiente a la habilitación de la	2. El sistema almacena controla el evento cambiando de estado la habilitación de la

funcionalidad dibujar. O pulsa el número "4" del teclado teniendo el foco sobre el mapa.	funcionalidad dibujar e informará al usuario mostrando el botón de habilitar/deshabilitar como pulsado o no.
3. Fin del caso de uso.	

5. Dibujar Dique:

Caso de uso: Dibujar dique de contención.

Actores: Usuario.

Objetivo: Dibujar una línea sobre el mapa que represente un muro de contención.

Descripción: El usuario dibuja la representación de un dique para no dejar pasar el agua en caso de que la zona se vea inundada. El usuario puede dibujar líneas con tantos vértices como quiera. Como precondición, ha de estar habilitada la función de dibujar.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón izquierdo del ratón sobre la posición donde quiere colocar el primer punto de la linea.	2. El sistema almacena este punto.
3. El usuario pulsa de nuevo el botón izquierdo del ratón sobre el punto del mapa donde desea poner el siguiente punto.	4. El sistema recoge el punto y dibuja la línea correspondiente.
5.a (Vuelta al punto 3). 5.b Fin del caso de uso.	

6. Cargar Diques desde archivo:

Caso de uso: Carga diques mediante la escritura de estos en un archivo de texto plano.

Actores: Usuario.

Objetivo: Poder realizar la carga de los diques sin tener que dibujar en el mapa, sino de una manera mucho más rápida y precisa. Cargando un archivo donde están escritas las coordenadas geográficas en las que se quiere colocar cada punto de los diferentes muros que se desean dibujar.

Descripción: El usuario indica que archivo cargar. Previamente habrá tenido que editar dicho archivo con los puntos donde quiere dibujar los muros escritos en un formato determinado.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón correspondiente a la carga del archivo con los diques.	2. El sistema abre un *Open File Dialog* con el sistema archivos del entorno cliente.
2. El usuario elige el archivo que desea cargar.	3. El sistema lee el archivo y dibuja los diques representados en este.
4. Fin del caso de uso.	

7. Seleccionar Dique:

Caso de uso: Seleccionar un dique de entre todos los dibujados.

Actores: Usuario.

Objetivo: Seleccionar un dique de entre los dibujados, para realizar con el alguna funcionalidad como dibujarle mas puntos, borrarlo o ver la información sobre este.

Descripción: El usuario selecciona un muro pulsando con le ratón sobre este, siempre y cuando no este habilitado el modo dibujar.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa con el botón izquierdo del ratón sobre el dique que desea seleccionar.	2. El sistema marca como seleccionado este dique y lo dibuja de otro color para resaltarlo del resto. Si anteriormente había un dique seleccionado lo marca

	como no seleccionado y lo dibuja del color por defecto.
3. Fin del caso de uso.	

8. Deseleccionar Dique:

Caso de uso: Deseleccionar el dique.

Actores: Usuario.

Objetivo: Deseleccionar el dique anteriormente seleccionado sin tener que seleccionar otro.

Descripción: El usuario deselecciona el muro que estaba marcado como seleccionado.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa con el botón izquierdo del ratón sobre el dique seleccionado anteriormente. O pulsa la tecla “6”.	2. El sistema almacena que no hay ningún dique seleccionado. El antiguo seleccionado es pintado en el color por defecto.
3. Fin del caso de uso.	

9. Borrar Dique:

Caso de uso: Borrar un dique de contención.

Actores: Usuario.

Objetivo: Seleccionar eliminar del mapa y de los datos de la aplicación uno de los diques anteriormente dibujados.

Descripción: El usuario, habiendo seleccionado previamente el dique, procede a eliminarlo.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa la tecla “5”.	2. El sistema el sistema elimina todo

	rastro del muro que se encontraba seleccionado. No deja ningún muro como seleccionado.
3. Fin del caso de uso.	

10. Ver información Dique:

Caso de uso: Ver información sobre el Dique.

Actores: Usuario.

Objetivo: Visualizar algunos datos sobre el dique como puede ser su longitud o su identificador.

Descripción: El usuario sitúa el puntero del ratón sobre le centro del dique del cual quiere visualizar la información. Seguidamente se le mostrará el identificador de este dique y su longitud total en kilómetros.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario sitúa el puntero del ratón sobre el dique del cual quiere visualizar la información.	2. El sistema muestra un bocadillo con la información del dique.
3. Fin del caso de uso.	

11. Inicializar Simulación:

Caso de uso: Inicializar simulación.

Actores: Usuario.

Objetivo: Realizar una serie de cálculos previos a la realización de la simulación, como por ejemplo dividir el mapa en celdas o crear los ficheros con los datos necesarios para la realización de la simulación.

Descripción: El usuario pulsa el botón referente a la inicialización de la simulación.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**

1. El usuario pulsa el botón de inicialización de la simulación.	2. El sistema realiza el los trabajos necesarios para dividir el mapa en celdas y para generar los ficheros con la información necesaria en la simulación. El sistema bloquea la aplicación durante el proceso. Una vez finalizado el proceso, el sistema desbloquea la aplicación pero el mapa y sus controles permanecen bloqueados para que no se pueda mover hasta después de simular. A de más, habilita la funcionalidad de Simular.
3. Fin del caso de uso.	

12. Reset Inicialización:

Caso de uso: Resetear inicialización.

Actores: Usuario.

Objetivo: Colocar la aplicación en el estado anterior a la realización de la inicialización previa a la simulación.

Descripción: El usuario pulsa el botón de resetear la inicialización para volver al estado inicial, es decir, mapa desbloqueado y la funcionalidad de simular deshabilitada. En el caso en que se quiera simular después de realizar este evento se deberá volver a realizar la inicialización.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón referente al reseteo de la inicialización previa.	2. El sistema desbloquea el mapa y sus controles. Deshabilita la funcionalidad de simular.
3. Fin del caso de uso.	

13. Realizar Simulación:

Caso de uso: Simular.

Actores: Usuario.

Objetivo: Realizar la simulación de la elevación del nivel del mar y llevar a cabo su representación grafica

Descripción: El usuario, una vez ha realizado la inicialización previa, decide realizar la simulación.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón para iniciar la simulación.	2. El sistema calcula la altura a la que ha de subir el nivel del mar y muestra el mapa alterado, de manera que marca de color azul aquellas zonas inundadas por la subida del mar. El mapa y sus controles son desbloqueados. Pasando la aplicación al estado previo a la incialización.
3. Fin del caso de uso.	

14. Borrar Agua:

Caso de uso: Borrar Agua.

Actores: Usuario.

Objetivo: Eliminar del mapa todos aquellos polígonos que representan la subida del nivel del mar.

Descripción: El usuario pulsa el botón de borrar agua eliminando todo polígono dibujado tras la simulación.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón referente al borrado del agua.	2. El sistema borra todos los polígonos que representan la subida del nivle del

	mar.
3. Fin del caso de uso.	

15. Borrar Todo:

Caso de uso: Borrar Todo.

Actores: Usuario.

Objetivo: Eliminar todo aquello dibujado sobre los mapas por el usuario o por la aplicación, con el objetivo de realizar otra simulación con otros diques.

Descripción: El usuario pulsa el botón de borrar todo eliminando todo polígono dibujado tras la simulación y todas las líneas que simulan los diques.

Curso típico de acontecimientos:

Acciones del usuario	**Respuestas del sistema**
1. El usuario pulsa el botón referente al borrado total	2. El sistema borra todos los polígonos que representan la subida del nivel del mar y todos los diques de contención.
3. Fin del caso de uso.	

6 Arquitectura e Implementación.

Tras la explicación de la especificación de la aplicación se procede a detallar como se ha desarrollado el software presentado.

En este capítulo se expondrá la estructura de la solución final. Es decir, se comentarán las diferentes partes de las que consta la aplicación, entrando en detalle en las partes más complejas, y como interactúan entre ellas.

Cabe destacar, que para la implementación se ha intentado seguir en la medida de lo posible la metodología explicada en el documento *"Guia d'estil de programación GEA4D.SDK001 versió 1.0*".
Esta metodología esta pensada para proyectos en C++. Nuestro proyecto consta de partes en C# donde la aplicación de esta metodología ha sido bastante útil y seguida, pero también consta de una parte importante en javascript donde ha sido algo más complicado llevar a cabo dicha metodología.

6.1 Arquitectura de la solución.

A continuación se muestra un esquema de la arquitectura global de la solución para entender el funcionamiento de la aplicación en un entorno real. En este subapartado se explicará la solución de manera sencilla sin entrar en detalles. Todos los detalles de las diferentes partes de la aplicación se explicarán en el subapartado Implementación.

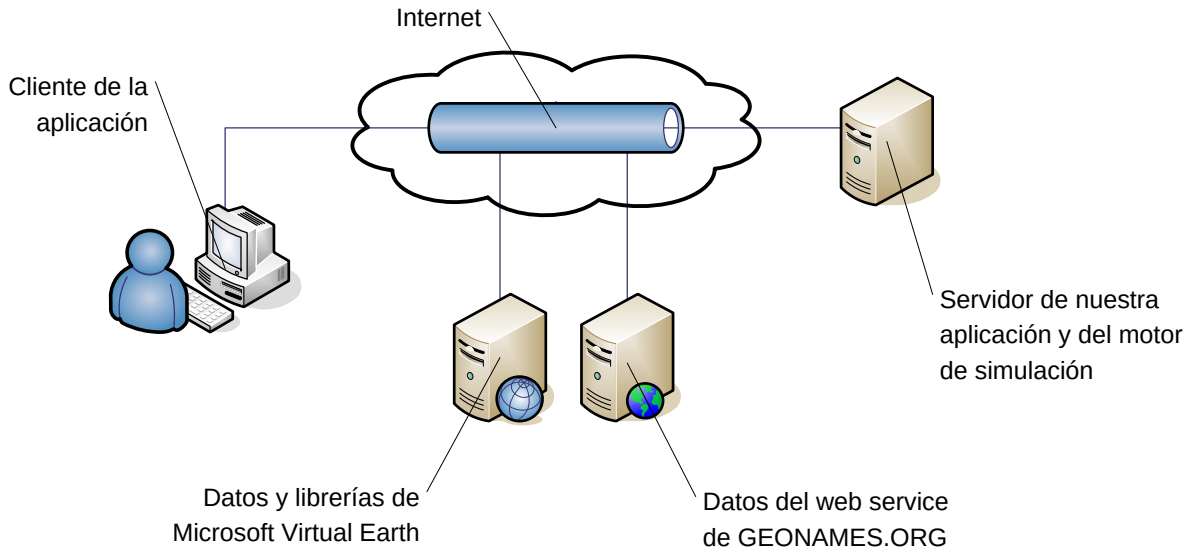


IMAGEN 6.1, ESQUEMA ARQUITECTURA DE LA SOLUCIÓN.

En este esquema se pueden ver los siguientes componentes:

- Cliente: es el equipo mediante el cual se conecta el usuario a la aplicación.
- Servidor: es el servidor donde estará alojada nuestra aplicación. También contendrá el motor que realice la simulación.
- Servidor Virtual Earth: es el servidor donde están alojados los mapas y las librerías de Virtual Earth.
- Servidor Geonames: es el servidor al que nos referimos cuando necesitamos información relacionada con la altura del suelo respecto al nivel del mar.
- Internet: la red de redes es la conexión entre los diferentes puntos del sistema.

El usuario, mediante un ordenador con conexión a Internet y con un browser web accede a la página que contiene nuestra aplicación.

Al entrar en ella, se descarga una archivo .zap que contiene la parte de la solución de realizada con Silverlight. De esta manera, todas las operaciones que realice con Silverlight no afectarán al rendimiento de nuestro servidor, si no que se ejecutarán en el cliente. Lo que hace que nuestro servidor no trabaje más de lo estrictamente necesario, reservando su esfuerzo para acciones más críticas como puede ser la realización de la simulación.

A sí pues, desde el cliente, la aplicación pedirá la información necesaria para visualizar los mapas de Virtual Earth.
Cuando llegue el momento de preparar la simulación, la aplicación se pondrá en contacto con un webservice el cual le devolverá información de manera asíncrona sobre la altura del suelo respecto al nivel en un punto determinado, información necesaria para realizar la simulación.

Esta información se almacenará en un conjunto de archivos en formato Idrisi32, los cuales serán utilizados por la aplicación que implemente el motor de simulación, la cual estará alojada en nuestro servidor.

Una vez el servidor haya acabado la simulación, devolverá una serie de valores a la aplicación que representan la altura a la que se eleva el nivel de mar. La aplicación se encargará de dibujar el agua según la altura indicada.

6.2 Implementación de la aplicación.

Seguidamente se presenta un esquema visual de los diferentes componentes de la aplicación para aportar un primer punto de vista de las partes de las que consta la solución a nivel de implementación. Después se explicarán algunos detalles de implementación que pueden resultar interesantes.

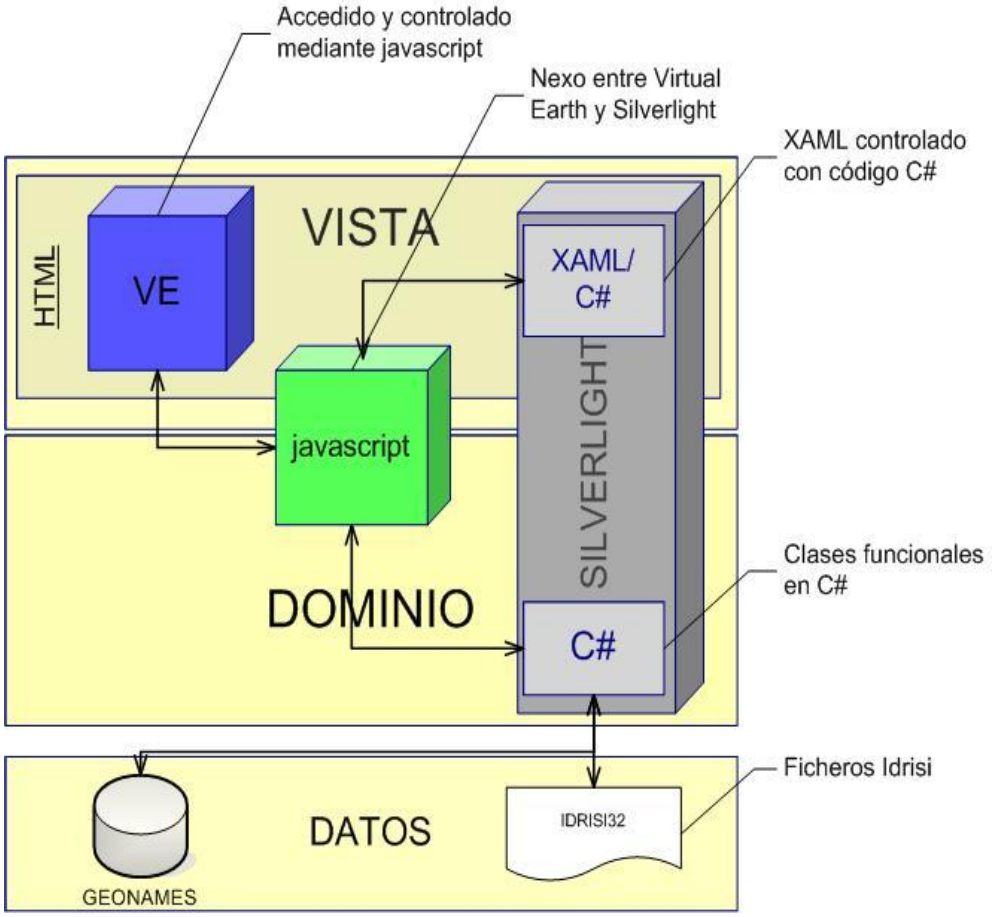


IMAGEN 6.2, ESQUEMA DE LA SOLUCIÓN.

Como se puede ver en el esquema, el proyecto está formado por partes de tecnologías diversas.

Por un lado, tenemos la parte visual, donde interviene el XML de Silverlight, los mapas de Virtual Earth, todo junto sobre una web en HTML.

Por otro lado, tenemos lo que podríamos considerar el dominio de la aplicación, donde están implementadas las diferentes funcionalidades de la aplicación. Esta parte esta realizada en clases en C#, ayudadas por funciones en javascript para recoger información del mapa.

La aplicación no consta de capa de datos, ya que no es necesario almacenar datos de manera masiva ni persistente. Solamente necesitamos los datos geoespaciales específicos para realizar una simulación en particular. Estos datos se representarán en archivos en formato idrisi32 para que el simulador los pueda interpretar correctamente.

6.2.1 Vistas

Como se ha dicho anteriormente podríamos dividir esta capa en dos grandes partes, la parte referente a los mapas de Virtual Earth y la parte referente a la interfaz de la aplicación realizada con Silverlight.

HTML.

El hecho de que ambas partes descansen sobre una página HTML es estrictamente necesario desde el instante en que se necesita añadir el mapa. Ya que, actualmente, es imposible encapsular los mapas de Virtual Earth en los XAMLs de Silverlight, la alternativa utilizada ha sido mostrar la aplicación Silverlight y encima poner el mapa, todo esto sobre un HTML que nos hace de nexo.

```
<body onload="GetMap();">
    <!-- mapa virtual earth -->
    <div id='myMap' style="position: absolute; width: 690px; height: 440px;"></div>
    <div id="silverlightControlHost">
        <object data="data:application/x-silverlight," type="application/x-silverlight-2-b1"
            width="100%" height="100%" id="SL">
            <param name="source" value="ClientBin/aguaSIM.xap" />
            <param name="onerror" value="onSilverlightError" />
            <param value="Transparent" name="background" />
            <param value="True" name="windowless" />
            <param value="True" name="enableHtmlAccess" />
            <a href="http://go.microsoft.com/fwlink/?LinkID=108182" style="text-decoration: none;">
                <img src="http://go.microsoft.com/fwlink/?LinkId=108181" alt="Get Microsoft Silverlight"
                    style="border-style: none" />
            </a>
        </object>
        <iframe style='visibility: hidden; height: 0; width: 0; border: 0px'></iframe>
    </div>
    <!-- Runtime errors from Silverlight will be displayed here.
    This will contain debugging information and should be removed or hidden when debugging is completed -->
    <div id='errorLocation' style="font-size: small; color: Gray;"></div>

</body>
```

Así pues, este HTML tiene dos únicas divisiones:

1. La división del mapa.
2. La división de la aplicación en Silverlight 2.

En este punto encontramos un acoplamiento entre ambas divisiones. Como el mapa no está dentro de la aplicación Silverlight, si no que esta encima, debemos colocarlo de acuerdo al sitio que dejemos libre en la pantalla realizada con Silverlight. A de más, debemos darle el tamaño adecuado para que este no oculte parte de la aplicación tras de sí.

Por otra parte, hemos de definir el objeto Silverlight de manera que permita HTML. Esto se consigue de manera muy sencilla poniendo como cierto la propiedad "enableHtmlAcces" al definirlo. El resto de propiedades del objeto Silverlight no son tan importantes, pero si no se inicializan correctamente puden suponer un problema, sobre todo la propiedad "source", en la cual hemos de instanciar el archivo .xap que será generado al compilar el proyecto Silverlight.
La propiedad "background" define el color tras el objeto y la en propiedad "onSilverlightError" se indica la función a ejecutar en caso de que el objeto Silverlight de un error al cargar. Todas estas propiedades solo se pueden tocar al crear el objeto. Una vez creado no son modificables.

Otro aspecto importante en la definición del objeto Silverlight es su identificador (id), ya que nos servirá para referenciarlo desde funciones en javascript.

VIRTUAL EARTH (javascript)

El mapa Virtual Earth está controlado mediante funciones implementadas en javascript. Este archivo javascript, llamado *VEMap.js,* está vinculado con la página html, de manera que al cargarse esta lo primero que hace es llamar a la función GetMap() para que esta cargue el mapa de la manera indicada.

En esta función se realiza la carga del mapa, y entre otras cosas, se define el modo y el tipo de mapa que mostraremos por defecto y se asocian las funciones que controlan los diferentes eventos sobre este que no vengan por defecto.

Aunque hay algunas funciones presentes en VEMap.js que tienen que ver más con aspectos del dominio de la aplicación de las cuales se hablará más adelante, la mayoría de las funciones presentes controlan aspectos gráficos del mapa o interactúan con la parte gráfica de Silverlight. Todas estas funciones se llevan a cabo utilizando las funcionalidades del Virtual Earth SDK 6.0.

Entre estas funciones podemos destacar tres grupos, las que realizan funciones de control sobre el mapa, las que dibujan formas en el mapa y las que recogen eventos del mapa.

Las primeras son las más sencillas. Simplemente realizan la función deseada sobre el mapa, como cambiar el nivel del zoom, rotar el mapa o cambiarlo de modo mediante funciones propias del SDK de Virtual Earth.

Las que dibujan formas son un poco más complejas. Dibujar un polígono o una recta es algo bastante sencillo en Virtual Earth, simplemente necesitamos crear una nueva instancia de la clase VEShape indicándole que tipo de forma queremos y los puntos de esta. A parte de crear la forma, se le dan diferentes propiedades, como el color, el ancho de la línea, la altura a la que se dibuja y el modo de la altura, relativo o absoluto.

Las funciones que recogen eventos son también bastante sencillas. Para su correcta ejecución, hay que relacionar la función con el evento mediante la función AttachEvent("<nombreFuncion>", <tipoEvento>) sobre la instancia del mapa.
Algunas de estas funciones interactúan con la parte de vista de Silverlight invocando funciones de C#.

```
function OnDoubleClickEventHandler(e)
{
    document.getElementById("SL").Content.Bridge.ZoomUpdateJS(e.zoomLevel);
}
```

En este caso, invocamos la función *ZoomUpdateJS* implementada en C#. Para ello recoge el contenido del elemento del documento que contiene el objeto Silverlight. En el objeto Silverlight se deberá registrar la clase a la que se quiere acceder como scriptable, en este caso se registra con el nombre "Bridge".

Una de las principales dificultades encontradas en el momento de implementación ha sido que el mapa en 2D no se comporta igual que el mapa en 3D y en algunos aspectos el comportamiento del modo 3D no es adecuado.
Por ejemplo, a la hora de dibujar un muro, la implementación realizada juega con latitudes y longitudes transformándolas en pixels, esta transformación se realiza perfectamente en el modo 2D, pero esto no es así en el modo 3D, donde la aproximación de pixel a latitud/longitud no la hace adecuadamente teniendo dificultades para dibujar los muros una vez se segmentan.

Otra dificultad muy remarcable es el hecho de la falta de precisión la altura a la que se dibujan los polígonos o polilíneas. Para determinar la altura a la que se dibuja una forma, Virtual Earth tiene dos modos, el absoluto y el relativo. Si estamos en modo relativo, la altura indicada será referente a la altura por encima del suelo, mientras que si estamos en modo absoluto el valor entrado hará referencia a la altura por encima del nivel del mar. Por lo tanto para nuestro interés, el modo elegido es el absoluto, donde la altura con respecto al nivel del mar se obtiene realizando una aproximación de la forma de la Tierra con un elipsoide, cosa que hace que en algunos puntos del globo no resulte ser nada exacta la altura dada.[5]

Por último, también es destacable la lentitud a la hora de realizar cualquier movimiento sobre el mapa (zoom, pan, rotación) cuando se ha dibujado una cantidad considerable de polígonos.

[5] Ver Anexo VI: Problema Geoid-Elipsoide WGS84

SILVERLIGHT (XAML + C#)

Como ya se ha dicho anteriormente, la capa de vista en la parte de Silverlight consta de archivos XAML, que realizan la representación gráfica de las pantallas, controlados por código en C#. En nuestro caso, debido a que solo tenemos una pantalla, tenemos un único XAML llamado *Page.xaml* y una clase C# que controla los diferentes eventos que se producen, bajo el nombre de *Page.xaml.cs*.

La implementación del XAML se ha realizado con los controles facilitados en el plug-in Visual Studio Silverlight Tools Beta 1, lo que ha facilitado mucho el trabajo y ha permitido una apariencia mucho más cuidada.

El primer tag que se encuentra al ver el archivo XAML de cualquier aplicación Silverlight es el tag UserControl, donde se define el nombre de la clase a la que hace referencia el XAML, el schema que utiliza y el namespace y assembly de la aplicación. Toda esta información es autogenerada si utilizamos Visual Studio.

```
<UserControl x:Class="aguaSIM.Page"
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:Agua="clr-namespace:aguaSIM;assembly=aguaSIM"
    Width="1000" Height="650" Margin="5">
    <Canvas...>
</UserControl>
```

A parte de este código autogenerado, también se pueden definir otros aspectos como la altura y el ancho de la pantalla que estamos implementando. Dentro de UserControl definimos el tag Canvas (lienzo) donde se colocarán el resto de los controles y formas de la aplicación.

La colocación de controles sobre el Canvas se pude realizar de diferentes maneras.

Si se trata de una pantalla simple con pocos controles se puede optar por poner directamente a cada control el sitio donde ha de estar. Esto se consigue mediante las propiedades “Canvas.Top” y “Canvas.Left” a las cuales les asignamos un valor entero que representa el pixel donde se sitúa la esquina arriba-izquierda del control que queremos colocar.

Pero si se implementa de esta manera una pantalla compleja puede resultar catastrófico, debido no solo a las veces que se deberían poner estas dos propiedades si no a la complejidad que tendría el código XAML ya que no seguiría ningún orden, teniendo todos los controles directamente colgando del tag Canvas. Como solución, hay dos alternativas, utilizar el control Grid o utilizar el control StackPanel.

Al utilizar el control Grid, este divide el Canvas en celdas del mismo tamaño en las cuales se colocan los diferentes controles. La desventaja de esta solución es que todo queda demasiado cuadrado.

Con el control StackPanel se define una pila de controles. Esta pila puede ser horizontal o vertical, de manera que se pueden declarar diferentes controles dentro de el StackPanel los cuales se iran añadiendo a la izquierda del primero o bajo este. A de más, dentro de un StackPanel podemos definir otro, de manera que podemos tener todo ordenado tanto horizontal como verticalmente.

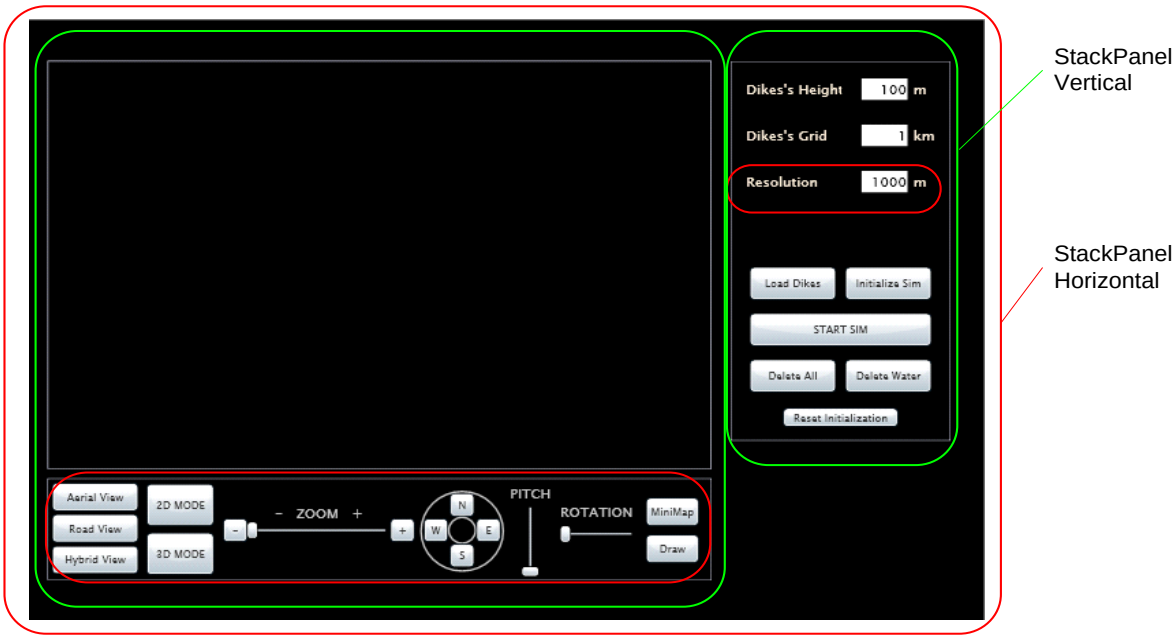


IMAGEN 6.3, VISTA PRELIMINAR DE PAGE.XAML.

Para recoger un evento de un control determinado lo que se hace es poner como valor de la propiedad del evento la función que queramos que lo recoja. Cada tipo de control tiene propiedades de eventos diferentes. Por ejemplo la más utilizada es el evento de picar un botón, este evento se recoge poniendo en la propiedad Click del botón la función deseada.

```
<Button x:Name="btnDeleteWater" Click="btnDeleteWater_Click"
        Content="Delete Water" Width="90" Height="35"
        Margin="5" TextAlignment="Center"></Button>
```

En este ejemplo se puede ver como se asigna la función btnDeleteWater_Click implementada en la clase Page.xaml.cs al botón btnDeleteWater.

Algunas de estas funciones que controlan los eventos interactúan con el mapa mediante funciones en javascript. Para ello es necesario incluir la librería System.Windows.Browser y mediante el objeto HtmlPage se invoca la función previamente implementada en el javascript vinculado en el HTML de la aplicación, como se pude ver en el siguiente ejemplo.

```
private void btnDeleteAll_Click(object sender, RoutedEventArgs e)
{
    HtmlPage.Window.Invoke("DeleteAllShapes");
}
```

También es necesaria la interacción en el sentido contrario, es decir desde código javascript llamar a funciones en C#, para modificar componentes de la pantalla cuando se realice algún evento sobre el mapa. Para permitir la interacción, se deben realizar los siguientes pasos:

1. Declarar la clase como scriptable.

```
[ScriptableType]
public partial class Page : UserControl...//class
```

2. Registrar la clase a la cual se quiere acceder como scriptable.

```
HtmlPage.RegisterScriptableObject("Bridge", this);
```

   Se debe registrar la clase como un objeto scriptable dándole un nombre determinado por el cual accederemos a la clase desde javascript.
   En nuestro caso el nombre será “Bridge”, ya que lo que se hace es crear un puente entre las dos tecnologías, y lo definiremos en la función creadora del objeto Page.

3. Declarar cada función que se tenga que llamar desde javascript como scriptable.

```
[ScriptableMember]
public void ZoomUpdateJS(int level)...
```

Habiendo realizado estos pasos solamente hay que llamar a las funciones de la manera que se ha explicado en el subapartado anterior.

La principal dificultad encontrada en esta parte se vio al realizar la primera versión de la pantalla. Entonces no se contaba con el plug-in para Visual Studio y se realizaba en Silverlight 1.0 con lo cual no se contaba con los controles y componentes, solo con canvas, rectángulos y cuados de texto. Esta primera implementación fue más costosa y daba como resultado una aplicación muy poco atractiva.

Otra complicación remarcable es el hecho del refresco de la pantalla. Si desde una función se realiza un cambio en algún componente de la pantalla, este cambio no tiene efecto hasta que finaliza la función y se devuelve el foco a la vista. Esto ha implicado un comportamiento no deseado en algún momento, como por ejemplo a la hora de realizar los procesos de inicialización de la simulación.

Este proceso es costoso, y hay que dar un feedback al usuario informándole que la aplicación está trabajando. Para ello se muestra un mensaje informativo, pero este mensaje no aparece hasta que el foco es devuelto a la pantalla, de tal manera que es inapreciable. Para corregir este hecho se pensó en utilización de threads, consiguiendo un resultado aproximado al requerido. Es aproximado por el hecho de que si no se está en el UI thread (*User Interface Thread*) no es posible modificar ningún componente de la pantalla e incluso no es posible invocar a las funciones presentes en el javascript.
Se ha intentado también utilizar la propiedad “Dispatcher” de los controles que se querían modificar desde un thread que no fuese el UI, pero tampoco se ha conseguido el efecto esperado, así que finalmente solo se ha utilizado threads.

Mediante la propiedad Dispatcher se puede modificar los controles desde otro thread que no sea el UI, mediante el código que veremos a continuación, pero el comportamiento es el mismo que antes, no vemos el cambio hasta que el foco vuelve a la pantalla.

```
public void Nada(object sender, DoWorkEventArgs e) {
    rtgBlock.Dispatcher.BeginInvoke(new Visible2(this.Visible));
}
private delegate void Visible2();
public void Visible() {
    rtgBlock.Visibility = Visibility.Visible;
}
```

En este código destacan dos cosas, la utilización de la propiedad Dispatcher del objeto rgtBlock. Este, mediante la función BeginInvoke() crea un objeto Visible2. Este objeto ha sido declarado con la propiedad *delagate,* y lo que hace es que nos permite invocar a la función que le pongamos entre los paréntesis desde un thread que no sea el UI.

6.2.2 Dominio.

Esta capa se basa en gran parte en las funciones implementadas en C# en la clase CFuncionabilidad, aunque algunas de estas operaciones necesitan consultar información en el mapa, con lo cual llaman a funciones implementadas en VEMap.js, realizando su invocación de la manera descrita anteriormente.

El dominio de la aplicación tiene un propósitos muy marcado que no es otro que el de crear los archivos en formato Idrisi32 necesarios para realizar la simulación. Por otra parte, también tiene que leer los datos recibidos de la simulación para que la capa de vistas sea capaz de dibujar los polígonos que interpretan el agua.

SILVERLIGHT (C#)

Como se acaba de decir, se han de crear los ficheros necesarios con la información que requiere el simulador. Silverlight, al estar diseñado para la creación de aplicaciones web ejecutadas en el cliente, no permite la manipulación del sistema de archivos, ya que esto supondría una falta alarmante de seguridad. Para ello existe la clase IsolatedStorage, presente en el framework .Net del que dispone Silverlight 2.0.

IsolatedStorage es un mecanismo de almacenaje que proporciona aislamiento y seguridad, definiendo maneras estándares de asociar la aplicación con los datos guardados. Con esto se consigue crear localizaciones seguras en el sistema de archivos, donde los datos almacenados están protegidos de otras aplicaciones, sin tener que definir estas localizaciones. Es decir, no es necesario hardcodear la información que indica el sitio donde la aplicación guarda los datos, ya que esto se hace de manera interna, definiendo un sitio para cada aplicación, usuario y assembly. Esto es muy útil para aplicaciones web, donde la política de seguridad no permite el acceso al sistema de archivos mediante el mecanismo estándar de entrada y salida.

Mediante esta clase, podemos crear todo tipo de archivos desde una aplicación web, pero solo teniendo el acceso restringido a la carpeta que nos generé IsolatedStorage. Es la única manera posible de realizar este proceso con esta tecnología, y aunque es positivo el hecho de que estos datos no puedan ser accedidos desde ninguna otra aplicación asegurando su confiabilidad, cuenta con una gran desventaja.

Esta desventaja es que desde nuestra aplicación podemos acceder a los datos pero no podemos conseguir el path que nos lleve hasta ellos, es decir, la aplicación sabe donde se encuentran pero no se puede mostrar el camino hasta ellos al usuario. A esto se le une otro inconveniente, y es que la ruta donde se generan los almacenes de IsolatedStorage es dependiente del sistema operativo donde corre la aplicación, cosa que hace todavía más complicado el intento del usuario de recoger los datos generados por la aplicación.

SO	PATH
Windows 98, Windows Me - user profiles not enabled	Roaming-enabled stores = <SYSTEMROOT>\Application Data NonRoaming stores = WINDOWS\Local Settings\Application Data
Windows 98, Windows Me - user profiles enabled	Roaming-enabled stores = <SYSTEMROOT>\Profiles\<user>\Application Data Nonroaming stores = Windows\Local Settings\Application Data
Windows NT 4.0	<SYSTEMROOT>\Profiles\<user>\Application Data
Windows NT 4.0 - Service Pack 4	Roaming-enabled stores = <SYSTEMROOT>\Profiles\<user>\Application Data

	Nonroaming stores = <SYSTEMROOT>\Profiles\<user>\Local Settings\Application Data
Windows 2000, Windows XP, Windows Server 2003 - upgrade from NT 4.0	Roaming-enabled stores = <SYSTEMROOT>\Profiles\<user>\Application Data Nonroaming stores = <SYSTEMROOT>\Profiles\<user>\Local Settings\Application Data
Windows 2000 - clean install (and upgrades from Windows 98 and NT 3.51)	Roaming-enabled stores = <SYSTEMDRIVE>\Documents and Settings\<user>\Application Data Nonroaming stores = <SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data
Windows XP, Windows Server 2003 - clean install (and upgrades from Windows 2000 and Windows 98)	Roaming-enabled stores = <SYSTEMDRIVE>\Documents and Settings\<user>\Application Data Nonroaming stores = <SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data
Windows Vista	Roaming-enabled stores = SYSTEMDRIVE>\Users\<user>\AppData\Roaming Nonroaming stores = <SYSTEMDRIVE>\Users\<user>\AppData\Local

TABLA 6.1, RESUMEN DIRECTORIOS DE ISOLATEDSTORAGE SEGÚN SOS DE MICROSOFT.

En nuestro caso, nos interesa que el usuario pueda recoger los ficheros creados por la aplicación con el objetivo de introducirlos en un Sistema de Información Geográfica. Para ello se pensó que aunque la ruta la definiese la aplicación, se podía mostrar al usuario donde encontrar los archivos, cosa que se ha visto resulta imposible actualmente con Silverlight. Y digo actualmente con Silverlight, porque otras tecnologías que cuentan con el framework .Net si que pueden mediante la técnica conocida como Reflection. Reflection es un mecanismo utilizado para descubrir información de las clases solamente en tiempo de ejecución. Por desgracia en el framework preparado para Silverligth Reflection, tiene muchísimas limitaciones, pensadas para respetar la seguridad del sistema que actúe como cliente.

Para utilizar IsolatedStorage necesitamos incluir la librería System.IO.IsolatedStorage. Los pasos son los siguientes.

1. Declarar el objeto IsolatedStorageFile. Esto recogerá el IsolatedStorage de la aplicación.

```
var isf = IsolatedStorageFile.GetUserStoreForApplication();
```

2. Declarar el objeto IsolatedStorageFileStream. Se define el archivo que se abrirá dentro del directorio de isolatedStorage. También se define con que tipo de permisos se hace.

```
var docAltura = new IsolatedStorageFileStream
                ("altura.doc", System.IO.FileMode.Create, isf);
```

3. Por último se define un StreamReader o StreamWriter para el archivo especificado, dependiendo si se quiere leer o escribir.

```
m_swAltura = new StreamWriter(m_imgAltura);
```

A partir de aquí se utiliza como un StreamReader o StreamWriter normal, mediante las funciones como por ejemplo readLine o writeLine. Al final es necesario cerrar tanto el StreamWriter como el isolatedStorageFileStream, se hace mediante la función close() sobre cada uno de los objetos.

La carpeta que crea IsolatedStorage permite almacenar hasta cierto tamaño de datos. Por defecto el tamaño esta en 100kb, pudiéndose incrementar la cuota a 1Mb, 5Mb, 10Mb o tamaño ilimitado. Esto se consigue mediante la función TryIncreaseQuotaTo(<int tamaño>). Antes de realizarse por primera vez esta operación, aparece una ventana de dialogo en la cual el usuario ha de aceptar para que el tamaño de la cuota se incremente. Sin el permiso del usuario no es posible incrementarlo.

A parte de crear los ficheros, necesitamos obtener la información para rellenarlos. Esta información se consigue de dos fuentes diferentes:

- Del mapa: la información referente a los diques y a las coordenadas geográficas.
- Del webservice srtm3JSON de Geonames.org: la información sobre la altura del suelo respecto al nivel del mar.

La información referente a coordenadas y diques se calcula en funciones implementadas en javascript, ya que se ha de tener acceso al mapa. Estas funciones

facilitan el numero de celdas en las que se divide le mapa, la longitud en pixels de cada celda y si en una celda hay o no un dique.

Sabiendo cuantas celdas hay y que longitud tienen se realiza un bucle en C# para recorrerlas, pidiendo al mapa la referencia latitud y longitud del punto medio de cada una de las celdas. Con las coordenadas se monta un string que se utilizará para obtener la información de la altura del suelo.

La información referente a la altura del suelo respecto al nivel del mar, debido a que los mapas de Virtual Earth no disponen de esta información, se consigue mediante la invocación a un webservice.
Este webservice pertenece a la web *www.geonames.org*, que se dedica a dar servicios web sobre información geográfica. Tiene diferentes webservices dependiendo de la información que se busque. En nuestro caso nos interesa el srtm3JSON, que mediante una coordenada geográfica compuesta por latitud y longitud nos devuelve la altura sobre el nivel del mar en metros al que se encuentra ese punto de la Tierra.

Desde C# hay diferentes formas de invocar a un webservice, destacando dos grupos: de manera asíncrona y de manera síncrona. La diferencia entre ambas es que al utilizar la síncrona, una vez se realiza la llamada, la aplicación espera hasta que el webservice le proporciona la información requerida. Mientras que utilizando la asíncrona, la aplicación sigue su curso y la recepción de la información solicitada se lleva a cabo más adelante desde otro thread.
A nivel de programación, la recepción síncrona es mucho más sencilla de implementar que la asíncrona.
Aunque C# cuenta con las dos formas diferentes de recepción, el framework preparado para Silverlight 2 solo cuenta con la recepción asíncrona.

Para realizar la petición asíncrona de un webservice se necesita incluir la librería System.Net. Esta nos habilita la utilización de los tipos de objeto WebClient y DonwloadStringCompletedEventHandler.

```
WebClient client = new WebClient();
client.DownloadStringCompleted +=
    new DownloadStringCompletedEventHandler(WSAsyncronousReception);
client.DownloadStringAsync(new Uri(url));
```

Como se ve en el código de ejemplo, lo primero a hacer es crear un WebClient. Esta clase proporciona métodos comunes para enviar o recibir datos de cualquier recurso local, intranet o de internet identificado por un URI.

Seguidamente, a la propiedad DonwloadStringCompleted se le asigna un objeto DonwloadStringCompletedEventHandler, el cual tiene como parámetro la función que se ejecutará cuando la aplicación reciba la respuesta del servicio web.

Por último se llama al método DowloadStringAsync del objeto WebClient. Este método descarga el recurso especificado por la URI que se le asigna, sin bloquear el thread que realiza la petición. Es decir, el objeto URI ha de contener la dirección web del servicio que se invoca con los parámetros pertinentes. En nuestro caso tenemos el siguiente string: “http://ws.geonames.org/srtm3JSON?lat=<LAT>&lng=<LONG>”

Cuando el webservice da respuesta a la petición, desde otro thread se ejecuta la función que se le ha identificado en el objeto DonwloadStringCompletedEventHandler. Esta función se ha de definir con dos parámetros, el primero de tipo object y el segundo de tipo DownloadStringCompletedEventArgs. En este último se encontrará la información pedida.

```
void WSAsyncronousReception
        (object sender, DownloadStringCompletedEventArgs e)...
```

El principal problema encontrado en este apartado es que claramente el tiempo de ejecución de nuestra aplicación depende en gran medida de las respuestas del webservice. Si el webservice falla la aplicación queda a la espera o recibe información invalida. A primera vista no hay solución aparente a esto, aunque una posible vía sería montar una base de datos con los datos que nos proporciona el servicio web. Estos datos se pueden conseguir mediante geonames.org, pero es una compilación de información enorme, lo cual implica un gasto de tiempo importante para construir la base de datos.

JAVASCRIPT

En el javascript VEMap.js se realizan dos tareas importantes para la capa de dominio. La primera es calcular el número de celdas en las que se divide el mapa y la segunda es comprobar si en una celda especifica existe o no presencia de diques.

Para obtener la información del mapa lo primero que se hace es dividir este en celdas, la longitud de las cuales se le pide al usuario. Esta longitud esta expresada en metros. Las coordenadas de referencia del mapa están expresadas en latitud y longitud, las cuales no tienen una transformación directa a metros, ya que los metros de 1 grado de longitud dependen de la altitud a la que están. Por esta razón, se decide hacer todo este tipo de cálculos en pixels aprovechando unas funciones implementadas en Virtual Earth que pasa de pixel a latitud/longitud y viceversa.
De esta manera, se calcula la longitud total del mapa en metros, se divide entre la longitud en metros que ha de tener cada celda, y sabiendo que el mapa de largo tiene 690 pixels, se calcula cuantos pixels corresponden a cada celda.

Por otra parte, por cada celda se mira si hay o no existencia de dique. Esta comprobación la realiza una función javascript, la cual monta un cuadrado con las 4 esquinas de la celda y mira si alguno de los diques dibujados en el mapa corta alguna de las 4 rectas que forman los lados del polígono mediante la utilización de ecuaciones de la recta.

7 Manual de uso.

A continuación se verá de manera ilustrativa las diferentes funcionalidades de la aplicación, y como usar esta para asegurar su correcto funcionamiento.

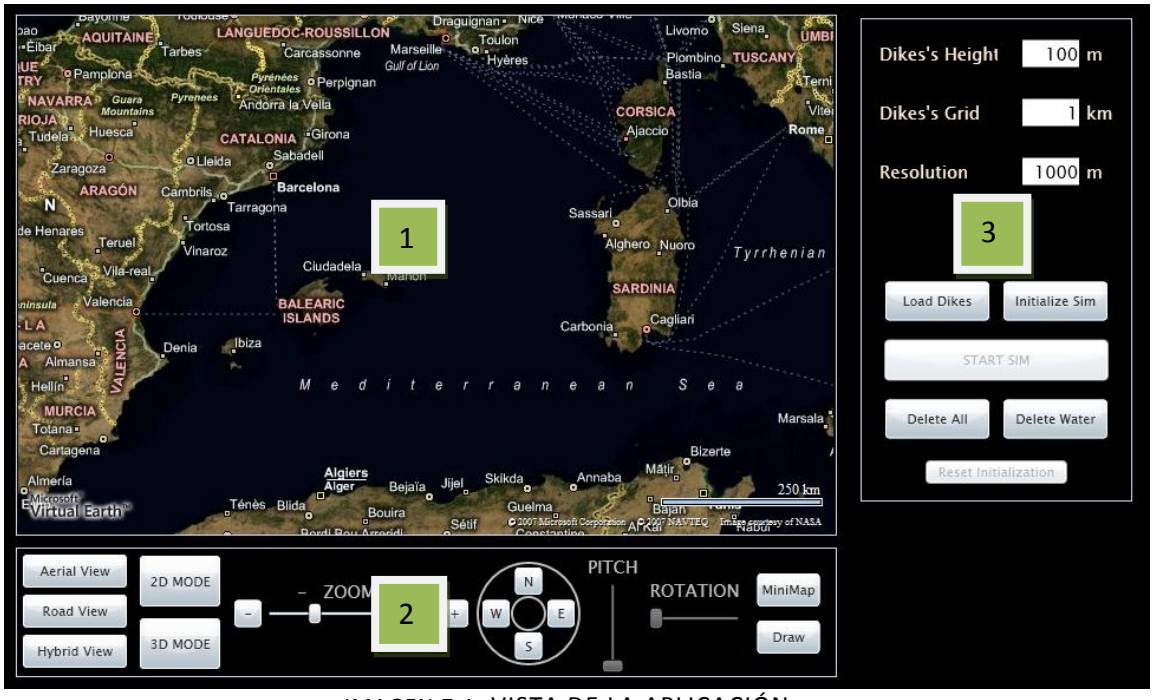


IMAGEN 7.1, VISTA DE LA APLICACIÓN.

Podemos diferenciar tres sectores distintos dentro de la pantalla de la aplicación.

El sector 1 es el referente al mapa, sobre él el usuario puede llevar a cabo diferentes funciones, como dibujar diques de contención y desplazarse sobre este.

En el sector 2 se encuentran los controles de la aplicación que se encargan de la gestión del mapa, movimientos del mapa, de la cámara o cambio de modo de vista.

El sector 3 agrupa los controles que podríamos denominar de simulación. Permiten la introducción de los parámetros necesarios para inicializar la simulación y gestionan las funcionalidades propias de la simulación como la carga de diques o el reseteo de los resultados de la simulación.

7.1 Control del mapa.

Produciendo eventos de ratón o teclado sobre el mapa podemos realizar las siguientes operaciones:

Eventos de teclado:

1. **Pulsar tecla "2":** se cambia el modo de visión, pasando de 3D a 2D. Si ya se encuentra en modo 2D no realiza ninguna acción.
2. **Pulsar tecla "3":** se cambia el modo de visión, pasando de 2D a 3D. Si ya se encuentra en modo 3D no realiza ninguna acción.
3. **Pulsar tecla "4":** habilita o deshabilita la opción de dibujar diques de contención en el mapa.
4. **Pulsar tecla "5":** borra del mapa el dique que se encuentra seleccionado (dibujado en color rojo). Si no hay ningún dique seleccionado no hace nada.
5. **Pulsar tecla "6":** deselecciona el dique seleccionado sin tener que seleccionar otro. Si no hay ningún dique seleccionado no hace nada.
6. **Pulsar tecla "7":** borra los resultados de la simulación, es decir, borra los polígonos que representan la subida del nivel del mar.
7. **Pulsar tecla "8":** borra tanto los diques como el agua, dejando el mapa en su estado inicial.
8. **Pulsar tecla "9":** muestra o esconde un pequeño mapa en la esquina superior izquierda que indica el lugar donde que estamos viendo con un nivel de zoom menor para tener una visión más amplia de la zona.

Eventos de ratón:

1. **Doble click sobre el mapa:** acerca la cámara, es decir, realiza la función de ampliar el zoom.
2. **CTRL + Doble click sobre el mapa:** aleja la cámara. Disminuye el nivel de zoom. Solo posible en modo 2D.

3. **Scroll:** dependiendo si se gira hacia arriba o hacia abajo, disminuye o amplia el nivel de zoom.
4. **Mantener clickado y arrastrar:** arrastra el mapa hacia la dirección donde se desplaza el ratón. Es decir, si se desplaza el ratón hacia abajo, se realizará una acción similar a un pan hacia el norte.
5. **CTRL + desplazar ratón en sentido vertical:** modifica la inclinación de la cámara. Desplazando hacia arriba se consigue mayor perpendicularidad con respecto al suelo. Solo realizable en modo 3D.
6. **CTRL + desplazar ratón en sentido horizontal:** rota el mapa. Si se desplaza hacia la derecha la rotación será en sentido horario, si por el contrario se desplaza a izquierda la rotación será en sentido horario. Solo realizable en modo 3D.
7. **Click de ratón teniendo habilitada la opción dibujar:** dibuja el segmento del dique comprendido entre el último punto dibujado del dique seleccionado y el punto donde se ha picado con el ratón. Si es el primer punto, simplemente se almacena. Solo habilitado en modo 2D.
8. **Click de ratón sobre dique:** deselecciona o selecciona el dique picado, dependiendo de si estaba o no previamente seleccionado.
9. **Colocar puntero sobre chincheta de dique:** se muestra un mensaje con el número que identifica al dique y su longitud total en kilómetros.

7.2 Panel de control del mapa.

Los controles de esta sección realizan funciones en el mapa. Los podemos dividir en tres apartados, los controles de modo de mapa y vista, los controles de cámara y otros.

IMAGEN 7.2, PANEL DE CONTROL DEL MAPA.

Controles de modos de mapa y vista:

1. **Aerial View:** Cambia el modo del mapa a modo "Aéreo", formado solo por imágenes, sin ninguna información adicional.
2. **Road View:** Cambia el modo a modo "Carretera", presentado un mapa de político con información de carreteras y calles.
3. **Hybrid View:** Cambia el modo a modo "Híbrido", compuesto por la información conjunta de los modos "Aéreo" y "Carretera". Es el modo de mapa por defecto.
4. **2D MODE:** Cambia el tipo de vista a modo 2D. Es el modo de vista por defecto.
5. **3D MODE:** Cambia el tipo de vista a modo 3D.

Controles de cámara y desplazamiento:

1. **ZOOM+:** Amplia el nivel de zoom, acerca la cámara al suelo.
2. **ZOOM -:** Disminuye el nivel de zoom, aleja la cámara del suelo.
3. **ZOOM Slider:** Desplazándolo hacia la izquierda se consigue alejar la cámara del suelo, mientras que hacia la derecha se acerca.
4. **N, S, E, W:** Realiza un desplazamiento de la cámara hacia la dirección seleccionada (N Norte, S Sur, E Este o W Oeste). En modo 3D, el mapa puede estar rotado, lo que implica que el desplazamiento Este no es hacia el este si no hacia la derecha de la pantalla.
5. **PITCH Slider:** modifica la inclinación de la cámara. Desplazando hacia abajo se consigue mayor perpendicularidad con respecto al suelo. Solo realizable en modo 3D.
6. **ROTATION Slider:** rota el mapa. Si se desplaza hacia la derecha la rotación será en sentido horario, si por el contrario se desplaza a izquierda la rotación será en sentido horario. Solo realizable en modo 3D.

Otros Controles:

1. **MiniMap:** muestra o esconde un pequeño mapa en la esquina superior izquierda que indica el lugar donde que estamos viendo con un nivel de zoom menor para tener una visión más amplia de la zona. Si el minimapa esta visible el botón MiniMap se encontrará ligeramente oscurecido.
2. **Draw:** habilita o deshabilita la opción de dibujar diques de contención en el mapa. Si la opción está activa, el botón permanece oscurecido.

7.3 Panel de control de la simulación.

Este panel contiene los controles relacionados con la simulación. Por una parte tiene una serie de parámetros que el usuario ha de entrar al comenzar a utilizar la aplicación, por otra parte consta de una serie de botones mediante los cuales nos desplazamos por los diferentes estados de la aplicación.

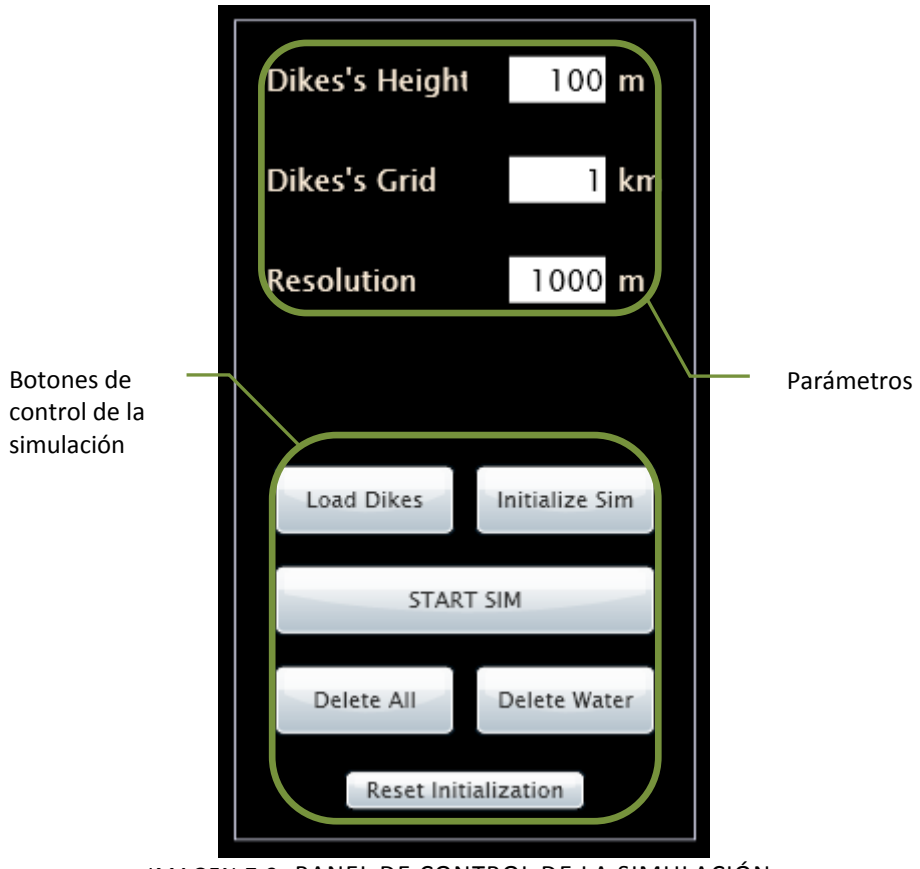


IMAGEN 7.3, PANEL DE CONTROL DE LA SIMULACIÓN.

Parámetros:

1. **Dike's Height:** Marca la altura respecto al elipsoide WGS84 del dique que se está dibujando. Valor en metros.
2. **Dike's Grid:** Determina cada cuanto se segmenta un dique. Valor en kilómetros.
3. **Resolution:** Especifica las dimensiones de cada celda de la malla en la cual se divide el mapa para realizar la simulación. Valor en metros.

Controles de la simulación:

1. **Load Dikes:** Al pulsar este botón se abre una ventana para seleccionar un archivo mediante el cual cargar diques. Este archivo ha de estar en formato Idrisi32 vectorial. No es posible cargar diques entre la inicialización y la simulación.
2. **Initialize Sim:** Inicialización previa a la simulación. Crea los ficheros necesarios para llevar a cabo la simulación. También bloquea el mapa para que no se pueda mover hasta después de haber finalizado la simulación.
3. **START SIM:** Ejecutar la simulación. Solo accesible cuando se haya realizado correctamente la inicialización previa. Una vez finalizada la simulación se volverá a tener control sobre el mapa, y se podrá preparar para otra posible simulación.
4. **Delete All:** Elimina toda forma sobre el mapa creada por el usuario o por la aplicación. Es decir, elimina las polilíneas que representan los diques y los polígonos que representan el agua, dejando el mapa en su estado inicial.
5. **Delete Water:** Elimina los polígonos dibujados por la aplicación que representan la subida del nivel del mar. Es decir, como s no se hubiese realizado ninguna simulación.
6. **Reset Initialization:** Desbloquea el mapa bloqueado por la inicialización previa. Debido a esto, deshabilita la opción de simular hasta que se vuelva a realizar la inicialización.

8 Trabajos Futuros.

Este proyecto, como todos los proyectos, tiene un inicio y un fin. La finalización de este PFC solo supone la finalización del inicio de este ambicioso proyecto. Solamente se ha puesto la primera piedra en la construcción del simulador del agua.
Hay muchas cosas por implementar o incluso por mejorar en el simulador, que debido a la limitación de tiempo no se han podido realizar. Quedando así pendientes para futuros desarrollos.

A continuación se enumeran las diferentes tareas que se deberían realizar en futuras implementaciones, así como posibles cambios sobre lo realizado para mejorarlo.

Nuevas Implementaciones: El motor de Simulación.

Debido a la falta de tiempo y a la complejidad que implicaba su desarrollo, no se ha llegado a realizar el motor de simulación.

Es una tarea muy costosa. La simple especificación de este motor puede ser algo muy complicado de realizar debido a la cantidad de procesos que entran en juego a la hora de hablar sobre el cambio climático y a la complejidad de algunos de estos procesos.
Para ello, se debería hacer un estudio sobre estos procesos más amplio que el que se ha realizado en este PFC, pudiéndose coger el pequeño estudio realizado aquí como inicio. Se deberán utilizar herramientas de simulación como diagramas SDL para realizar una correcta especificación.

Por otra parte, en lo que a implementación se refiere, el motor será una pieza clave de la solución final y sus tiempos de ejecución se deberán tener muy en cuenta. Debido a la más que posible complejidad de este, se debería implementar en un lenguaje que otorgue ejecuciones rápidas como es el caso de C++.

También se ha que tener en cuenta que el simulador es una aplicación web. Esto supone que el motor ha de ser accesible desde la aplicación actual. Para ello habrá que publicar el motor en un servidor. Durante este PFC se ha intentado dar el primer paso para realizar esta tarea, todo hay que decirlo, sin mucho éxito.
La idea inicial era utilizar las librerías de C++ ISAPI preparadas para realizar desarrollos web. Estas librerías han caído en desuso con la nueva versión de Internet Information Server (IIS 7). En la actualidad, los desarrollos web con C++ se realizan con código nativo para IIS 7. Este código nativo se puede subir como un módulo del ISS y puede ser ejecutado desde una web. Es una tecnología bastante espesa y novedosa, pero con la dedicación apropiada se llegar a realizar correctamente la implementación.

Otra posible solución consistiría en implementar el simulador en C#. Las clases del simulador se podrían unir a las creadas actualmente y formarían parte del código de la aplicación Silverlight. La ventaja con la anterior sería sin duda su facilidad de implementación, mientras que constaría con múltiples desventajas como el hecho de que se ejecutaría en el cliente, teniendo que pedir los datos que se dispongan almacenados vía web a nuestro servidor, y que no tendría la velocidad de C++ ni tampoco la potencia de C# debido a que las librerías de Silverlight están capadas en muchos aspectos.

Modificaciones y Mejoras.

Hay varios aspectos de lo actualmente implementado que no acaban de convencer. La mayoría de estos aspectos son provocados por limitaciones de las tecnologías utilizadas, siendo estas limitaciones culpa probablemente de lo innovadoras que son algunas de las tecnologías utilizadas.

Uno de los aspectos a mejorar es el feedback con el usuario cuando se está realizando la simulación. Este proceso será largo y el usuario deberá esperar. Actualmente se muestra un mensaje que le indica que aguarda mientras se realizan los cálculos pertinentes, pero este mensaje sale tarde. Esto es debido a que no aparece hasta que se le devuelve el foco a la aplicación, y este momento coincide con la finalización de la

simulación. Como ya se ha explicado, se ha intentado que el comportamiento sea diferente utilizando threads pero no se ha conseguido todo lo esperado.

Referente también a Silverlight, otro punto que no se ha llegado a conseguir es que el usuario al menos sepa el lugar donde se han generado los archivos en formato idrisi por si desea exportarlos a un sig. Como se ha explicado, en C# se puede mediante reflection pero no así en C# de Silverlgiht debido a que esta limitado. Es posible que en futuras versiones esté disponible o incluso haya alguna otra manera de solucionar este problema, aunque debido a los problemas de seguridad que podría causar es complicado que se permita manejar el sistema de archivos más allá de lo que permite isolatedStorage.
Si se pudiese hacer que el usuario tuviese acceso a los ficheros idrisi de manera sencilla, se podría implementar una nueva funcionalidad que consistirá en la carga de estos archivos sobre el mapa, es decir, se podrían dibujar diferentes diques, guardarlos en un archivo y cargarlos desde este en otra ocasión. Esto se podría realizar ya, pero no tiene sentido si el usuario no es capaz de encontrar el archivo que se ha generado para cargarlo más adelante.

Habrá que estar atentos a nuevas versiones de Silverlight para ver como va incrementando su potencial y si podemos obtener soluciones a nuestros problemas.

Sobre a la generación de los ficheros Idrisi, la recogida de los datos referentes a la altura respecto al nivel del mar de un punto de la Tierra es demasiado lenta. El hecho de utilizar un servicio web relentiza mucho la ejecución ya que la aplicación ha de esperar hasta que el webservice le dé respuesta. Además, si se le realizan muchas peticiones seguidas se colapsa devolviendo la petición sin respuesta, lo que imposibilita el hecho de realizar simulaciones sobre zonas muy grandes o con resoluciones altas.

No se ha encontrado ninguna alternativa a este problema. Lo ideal sería que esta información la facilitase el propio mapa. Se debería estar atento a nuevas versiones de Virtual Earth, o incluso se podría plantear el hecho de tener una base de datos propia

con esta información, aunque probablemente el resultado sería el mismo ya que los datos tendrían que viajar de nuestro servidor al cliente que es donde se está ejecutando la aplicación.

El mapa Virtual Earth es sin duda uno de los puntos a los que hay que estar atentos a nuevas versiones, no solo por la información de la altura, sino también por el hecho de que no dibuje los polígonos a la altura realmente especificada. A esto se le une el tema de que el modo 3D no realiza del todo bien el pase de coordenadas Latitud/Longitud a pixels lo que hace que no sea capaz de dibujar los diques de manera satisfactoria en modo 3D, para corregir este tema se optó por la segmentación de los diques, lo que hace que mejore mucho a la hora de dibujar líneas de distancias grandes, ya que estas quedaban ocultadas por la forma esférica del mapa 3D.
Por otra parte, si se realiza una simulación en la cual se han de pintar muchos polígonos el mapa en modo 2D se relentiza demasiado, cosa que impide hacer simulaciones con mucha precisión o de dimensiones muy grandes.

9 Conclusiones.

Aunque no se ha conseguido todo lo que en un primer momento se propuso, se está completamente satisfecho con todo lo conseguido y con el trabajo realizado, ya que se ha hecho todo lo posible para conseguir los objetivos en el plazo determinado.

No se ha podido realizar la implementación ni la especificación del motor de simulación, pero después de ver su complejidad es cierto que el objetivo inicial del proyecto era demasiado ambicioso para abarcarlo en un desarrollo de cuatro meses.

Por otro lado, la realización de este proyecto ha permitido la utilización de tecnologías muy actuales, como puede ser C#, y de tecnologías totalmente novedosas, como Silverlight. Esto marca dos aspectos positivos a nivel personal: la adquisición de nuevos conocimientos y la capacidad para buscar soluciones sobre aspectos que poca gente conoce.

Realizando este proyecto no solo se han adquirido conocimientos técnicos o informáticos, sino también conocimientos culturales de ámbito general, destacando mucho aspectos medioambientales referentes al cambio climático, los procesos que intervienen y sus efectos. Lo que conlleva el tener una visión diferente a la que se tenía anteriormente sobre este grave problema mundial.

10 Bibliografía

- **Cambio Climático.**

  - *Findings of the IPCC Fourth Assessment Report: Climate Change Science* http://www.ucsusa.org/global_warming/science/ipcc-highlights1.html
  - http://www-atmo.at.fcen.uba.ar/materias/CLASE%201-2007_clima.pdf
  - IPCC Working Group 1. *Climate Changes: The Physical Science Basis.* (2007) http://ipcc-wg1.ucar.edu/wg1/wg1-report.html
    - Chapter 4. Observations: Oceanic Climate Change and Sea Level.
    - Chapter 5. Observations: Changes in Snow, Ice and Frozen Ground.
    - Chapter 8. Climate Models and their Evaluation.
    - Chapter 10. Global Climate Projections.
  - IPCC Data Distribution Center. http://www.mad.zmaw.de/IPCC_DDC/html/ddc_gcmdata.html
  - Norma Sanchez Santillán, Rubén Sanchez Trejo, René Garduño López y Alfonso Esquivel Herrera. *El cambio climático relacionado con los mecanismos de retroalimentación entre la atmósfera, la superficie terrestre y la criósfera.* (2004) http://www.izt.uam.mx/contactos/n55ne/c-clima.pdf

- **Silverlight.**

  - Laurence Moroney. *Getting Started with Silverlight.* (2007) http://msdn.microsoft.com/en-us/library/bb404300.aspx
  - Silverlight en Microsoft Developer Network. http://msdn.microsoft.com/en-us/library/bb188743.aspx
  - Página oficial Silverlight. (tutoriales y foro) http://silverlight.net/

- Silverlight demos.

  http://www.vectorform.com/silverlight/

- Blogs de desarrolladores.
  - http://blogs.msdn.com/expressate/default.aspx
  - http://www.wilcob.com/Wilco/Silverlight.aspx
  - http://timheuer.com/blog/
  - http://weblogs.asp.net/mschwarz/default.aspx
  - http://simplesilverlight.wordpress.com/
- Ejemplos.

  http://www.silverlightexamples.net/

- **Virtual Earth.**
  - Virtual Earth en msdn.

    http://msdn.microsoft.com/en-us/library/aa905677.aspx
  - Foro virtual Earth en msdn.

    http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=537&SiteID=1
  - Virtual earth interactive SDK.

    http://dev.live.com/virtualearth/sdk

- **Otros.**
  - Servicios Web de información geoespacial.

    http://www.geonames.org/export/web-services.html
  - Formato Idrisi32.

    http://karnak.upc.es/~pau/Docencia/IDRISIFormat.pdf
  - Internet Information Server, página oficial.

    http://www.iis.net/
  - Problemática del geoid.

    http://www.esri.com/news/arcuser/0703/geoid1of3.html
  - Métodos de rasterización.

    http://www3.uji.es/~ribelles/Docencia/E26-F27/Docs/9-Raster.pdf

ANEXO I: Formato Idrisi32

En este anexo se describe el formato propio de los ficheros de Idrisi32, tanto los ficheros .doc y .img de tipo raster como los de tipo vectorial.

Raster:

En el tipo de fichero raster .img se expresan los datos geoespaciales en formato de matriz. Pero la representación en el fichero es algo tan sencillo como una larga columna de datos, donde cada elemento de la columna representa la información de una de las celdas de la malla en la que se ha dividido el mapa.

18
25
30
42
15
25
34
50
08
20
17
30

Pero la columna con la información, por si sola no representa nada, ya que no sabemos el tamaño de la matriz, ni el número de filas y columnas, ni al espacio geográfico a donde pertenece. Para todo ello es necesario el archivo de metadatos.

En nuestro caso el archivo de metadatos contendrá el siguiente formato:

```
TITLE: Dikes's Altitude
DATA TIPE: REAL
COLUMNS: 4
ROWS: 3
REF.SYSTEM: LAT/LONG
REF.UNITS: DEGREES
MIN.X: 2,02457427978515
MAX.X: 2,26146697998048
MIN.Y.: 41,4324311284618
MAX.Y: 41,3190756229514
POS'N ERROR: Unkown
RESOLUTION: 1000
COMENTS: Altura en metros de diques de contención.
```

1. TITLE: Título del archivo.
2. DATA TYPE: Tipo de dato en le que esta representada la información.
3. COLUMNS: Número de columnas de la matriz representada por los datos.
4. ROWS: Número de filas de la matriz de datos.
5. REF.SYSTEM: Sistema de referencia geográfica utilizado. En nuestro caso Latitud/Longitud.
6. REF.UNITS: Unidades de referencia. En nuestro caso grados
7. MIN.X: X mínima. En nuestro caso latitud mínima
8. MAX.X: X máxima. En nuestro caso latitud máxima.
9. MIN.Y: Y mínima. En nuestro caso longitud mínima
10. MAX.Y: Y máxima. En nuestro caso longitud máxima.
11. POS'N ERROR: Error geográfico. En nuestro caso desconocido.
12. RESOLUTION: Tamaño de cada celda. En este caso 1000 metros.
13. COMENTS: Comentarios sobre el archivo.

Teniendo en cuenta la información proporcionada por el archivo .doc, ahora podemos representar la tira de valores en como la siguiente matriz.

18	25	30	42
15	25	34	50
8	20	17	30

Vectorial:

El formato vectorial consta de dos columnas, una representando la latitud y la otra la longitud. Cada fila de esas dos columnas representa un punto por donde pasa ese vector. El vector será tan largo como pares latitud/longitud haya en le fichero hasta encontrar el par "0 0". El par de datos siguiente a esta serán coordenadas pertenecientes al siguiente vector.

41.12	2.17
39	3.90
39	1.00
0	0
45	5.73
46	4.21
47	6.51
0	0

ANEXO II: Elección del servidor de mapas on-line.

Para el buen desarrollo del proyecto, una parte vital es la correcta elección del servidor de aplicaciones de mapas a utilizar, ya que una parte importante del proyecto se desarrolla sobre dicha herramienta.

Este paso es un punto de inflexión en la elaboración del proyecto debido a que una vez elegida la aplicación no habrá marcha atrás. Por lo tanto, ha de ser una decisión tomada bajo unos criterios sólidos y objetivos.

Partiendo de la premisa que sabemos donde queremos llegar, debemos de ser capaces de elegir el camino con menos dificultades y el que más se adapte a nuestras expectativas.

Los servicios denominados *"Web Maping Services"* , cuya traducción al castellano sería "*Servicios de Cartografía en la Web*", son servicios de visualización o provisión de datos geoespaciales vía web. En la actualidad hay varios de estos servicios online disponibles de manera gratuita que constan de una buena calidad. Si destacamos los tres mas conocidos, en orden cronológico de aparición, son:

1. - NASA World Wind (2003).
2. - Google Maps (2005).
3. - Microsoft Virtual Earth (2007 versión Beta).

A primera vista, los tres presentan una buena calidad en las imagines, pero eso no es el requisito principal que estamos buscando. Uno de los requisitos fundamentales es que pueda funcionar sobre un web browser, cosa que cumple a la perfección Google Maps y Virtual Earth pero que no cumple NASA World Wind, quedando totalmente descartado de nuestra elección.

Así, contamos con dos herramientas muy similares disponibles para nuestra elección.

Definición de Criterios.

Una vez tenemos acotado el número de herramientas entre las cuales realizar la selección y antes de seguir estudiándolas y comparándolas, para ser capaces de diferenciarlas y ver cual nos es más útil y encaja mejor con nuestros propósitos, debemos definir los diferentes criterios a tener en cuenta para la correcta elección.

A. **Portabilidad:**

   A.1 Portabilidad a nivel de SO. El criterio a seguir es, obviamente, que contra mas sistemas operativos diferentes o mas versiones de los mismos soporte mejor valoración obtendrá el WMS.

   A.2 El WMS a utilizar ha de funcionar en un web browser. Como mínimo, el WMS deberá ser compatible con Internet Explorer y Mozilla Firefox. A partir de aquí, se valorará positivamente la compatibilidad con otros browsers como Opera.

B. **Funcionabilidad** [1]**:** Hay que examinar que funcionalidades plantea cada uno de los WMS. Lo mínimo es que nos permita movernos libremente sobre los mapas, tanto de manera horizontal como vertical (calidad del zoom). A partir de ahí, puede presentar otros tipos de funcionabilidades como visores en 3D, ortofotos,... .

   Para ver la calidad del zoom en las diferentes partes del globo terráqueo utilizaremos la aplicación web de la siguiente página en la cual se nos presenta ambos WMS a la vez, y al interactuar con uno el otro también lleva a cabo el mismo evento.

   http://www.jonasson.org/maps/

C. **Usabilidad:**

   C.1 A nivel de usuario. Intuitividad y facilidad de uso de la interficie de usuario que presenta el WMS. También es destacable si la interficie es amable al usuario.

C.2 A nivel de desarrollador:

C.2.1 Posibilidad de realizar aplicaciones. Es imprescindible que el WMS facilite la realización de aplicaciones sobre él. Es decir, el WMS ha de poder soportar aplicaciones hechas por usuarios, no solo ha de facilitar los datos cartográficos.

C.2.2 Existencia de Kit de Desarrollo de Software y Documentación sobre este. Se valorará mucho la existencia de APIs o SDKs igual que la documentación existente sobre ellos. Es importante comprobar que el SDK tenga métodos que nos permitan llevar a cabo nuestro propósito.

C.2.3 Requisitos para desarrollar. Otro aspecto a tener en mente es qué requisitos necesitamos a nivel de desarrollador para poder implementar aplicaciones para el WMS (licencia, registro, software especial...).

D. Confiabilidad [6]: es importante que el WMS elegido no presente demasiados casos de error. Se ha de comportar de manera lo más estable posible.

E. Eficiencia [1]: Cuanto más rápido sea capaz el WMS de cargar las distintas imágenes mejor. Miraremos la carga al hacer pan y zoom en las diferentes formas de mapas que cada WMS posee.

Para ver la velocidad a la hora de cargar mapas en 2D se utilizara la misma aplicación web que en el apartado de funcionabilidad.

Microsoft Virtual Earth.

A. Portabilidad.

A.1 A nivel de SO.

Funciona correctamente bajo Windows excepto la funcionalidad de vista en 3D. Es decir, Todas sus funcionalidades 2D (mapa, ortofoto e híbrido) funcionan a la

6 Solo se han utilizado los browsers Mozilla Firefox e Internet Explorer.

perfección bajo cualquier versión del sistema operativo Windows a partir de su versión '98, pero no es así para las funcionalidades 3D. Para que el WMS aguante la visualización en 3D es necesario como mínimo Windows XP con Service Pack 2.0, Windows Server 2003 o Windows Vista debido a que es necesario la instalación de una serie de librerías para su garantizar su funcionamiento.

En el resto de Sistemas Operativos (Linux y MAC OS) las funcionalidades 2D funcionan correctamente pero las 3D no.

A.2 A nivel de Web Browser.

Funciona correctamente bajo Internet Explorer y bajo Mozilla Firefox, aunque como ya se ha comentado para las funcionalidades en 3D se necesita la instalación de librerías, tanto para hacerlo funcionar en Explorer 6.0 y superiores como en Firefox 1.5 y superiores.

En otros browsers como Opera o Safari no soporta algunas funcionalidades.

B. Funcionabilidad.

La gran baza de Virtual Earth es que dispone de vista en 2D y en 3D. Cada uno de los diferentes modos de mapas se pueden ver tanto planos como de manera tridimensional, lo cual otorga bastante espectacularidad.

Cuenta con tres modos de ámbito general, más otros 2 restringidos al zoom o al lugar geográfico.

Entre los tres modos generales tenemos el modo “Road” donde dependiendo de la aproximación tenemos la información pertinente a un mapa político (continentes, países, regiones, ciudades) o un mapa callejero (carreteras, calles, sitios de interés...). Otro modo es el modo “Aerial” el cual se basa en ortofotografías a diferentes resoluciones, sin ningún tipo de información adicional. El tercero es el modo “Hybrid” que no es más que las imágenes de “Aerial” con la información de “Road”.

A parte de estos tres modos, también dispone de un modo “Traffic” restringido al ámbito estadounidenses en el que se muestran las principales arterias de comunicación y su estado, y el modo “Bird's Eyes” restringido a una cierto zoom y a

poblaciones, en el cual se nos muestra el mapa de la ciudad que queramos mediante fotografías en perspectiva. Depende de la importancia de la localidad que veamos, tendremos o no este modo de igual manera que tendremos imágenes desde 1 a 4 puntos de vista (norte, sur, este i oeste).

En los mapas en vista 2D se nos permite movernos con total libertad en horizontal y en vertical (zoom y pan) tanto con el ratón como con el teclado. A más a más, en modo 3D la libertad es aun mayor ya que también se nos permite el desplazarnos en profundidad y hacer rotaciones sobre el mapa. Esto hace que podamos observar los mapas desde cualquier ángulo dotándolo de una visión muy realista.

Por otra parte, Virtual Earth nos permite agregar nuestros propios objetos en 3D, previamente diseñados con una herramienta que nos facilita pero que funciona por separado.

C. Usabilidad.

C.1 A nivel de usuario.

Las diferentes funcionalidades se pueden realizar con eventos del ratón sobre el mapa, con eventos de teclado o bien mediante el click sobre los diferentes botones que presenta la interficie. Para cambiar de modo de mapa se realiza simplemente clickando sobre los botones identificados por el nombre del modo: 2D, 3D, Road, Aerial, Hibrid, Bird's Eyes.

El uso de los botones para realizar los desplazamientos es sencillo e intuitivo. Botones con los signos “+” y “-” para realizar el zoom, en el caso del pan dispone de un círculo que al clickar dentro de este, se realiza un pan hacia la dirección que se ha picado dentro del círculo. Para el modo 3D, existen 2 grupos de botones más. Dos botones que permiten rotar el mapa y otros dos que permiten modificar la inclinación de la cámara.

Todo esto, también se puede realizar con el ratón de manera simple: doble click izquierdo o girar la rueda para zoom, mantener picado el botón izquierdo y desplazar el ratón para el pan. Para las funcionalidades específicas de 3D se

requiere la tecla "*Control*", manteniendo esta tecla y el botón izquierdo picados si desplazamos en el eje x realizamos una rotación y si lo desplazamos en el eje z realizamos una inclinación de cámara.

Por otra parte, la interfície que presenta es bastante atractiva para el usuario, con un estilo moderno.

C.2 A nivel de desarrollador.

C.2.1 Posibilidad de realizar aplicaciones.

Microsoft Virtual Earth esta capacitado para soportar aplicaciones realizadas por el usuario.

C.2.2 Existencia de Kit de Desarrollo de Software y Documentación sobre este.

Virtual Earth cuenta con un SDK, disponible de forma gratuita. En estos momentos se encuentra en su sexta versión. También hay disponible bastante documentación: una versión interactiva y online del SDK (http://dev.live.com/virtualearth/sdk/), un foro para desarrolladores (http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=537&SiteID=1) incluso una comunidad que cuenta con una wiki y otro foro (http://viavirtualearth.com/vve/Dashboard/Default.ashx).

El SDK es capaz de trabajar con polígonos y polilíneas. Aunque en el modo 3D existen algunas complicaciones para poder colocar estos y otros elementos a la altura adecuada. También plantea complicaciones a la hora de conseguir la altura de un determinado punto definiendo este mediante latitud y longitud.

C.2.3 Requisitos para desarrollar.

No es necesaria ninguna clave ni licencia para desarrollar sobre Virtual Earth, lo único que se nos pide es un límite de transacciones al día (100.000).

D. Confiabilidad.

Suele tener un comportamiento bastante estable, aunque muy de vez en cuando no da servicio o es incapaz de cargar el modo 3D. Esto se supone que es debido a que se encuentra todavía en versión Beta. Aun así, parece bastante robusto y no presenta ningún comportamiento anómalo en las funcionalidades 2D.

E. Eficiencia.

El pan se realiza casi de manera inmediata, esto es capaz gracias a que no recarga todo el mapa sino simplemente la parte que antes no se veía.

El zoom es un poco más lento, pero su tiempo de carga sigue siendo bueno.

Los tiempos de carga, desde el punto de vista del ojo humano, no son diferenciables entre las funcionalidades "*Road*", "*Aerial*" o "*Hibrid*". La funcionalidad "*Bird's eyes*" es una poco más lenta, es de suponer que las imágenes que carga son más pesadas y por ello tarda un poco más, aunque sigue siendo un tiempo aceptable.

En la vertiente 3D, en las ciudades que no presentan edificios los tiempos de carga son iguales que en 2D. Ahora bien, en las ciudades con edificios, el WMS carga el suelo rápidamente como en el resto de ciudades y después de manera algo más lenta va cargando los diferentes edificios. Por lo tanto el tiempo de carga depende de la masificación de edificios que haya en la ciudad que carguemos. Es tiempo si que es apreciable al ojo humano, y hace que el usuario se tenga que esperar unos segundos para ver la imagen completa.

Goggle Maps.

A. Portabilidad.

A.1 A nivel de SO:

Una de las grandes ventajas de Google Maps es que el único requisito necesario para su funcionamiento es que el sistema operativo disponga de web browser. Aunque no se ha podido probar, en la página de Google Maps se da constancia de que también funciona en MAC OS.

A.2 A nivel de Web Browser:

Funciona correctamente con los principales browser Internet Explorer, Mozilla Firefox.

Actualmente Google Maps admite los siguientes navegadores web:

- IE 6.0 o superior.
- Firefox 0.8 o superior.
- Safari 1.2.4 o superior.
- Netscape 7.1 o superior.
- Mozilla 1.4 o superior.
- Opera 8.02 o superior .

B. Funcionabilidad.

Google Maps presenta diferentes modos de visionado de los mapas, algunos de ellos de ámbito general mientras que otros están restringidos a partir del lugar que estemos viendo o del nivel de zoom al que nos encontremos. En este aspecto muy similar a virtual Earth.

Entre los generales, el primero, bajo el nombre de “Mapa”, es un mapa normal y corriente, en el cual dependiendo del zoom se nos muestran los diferentes países, ciudades, poblaciones, carreteras y calles. El segundo modo es el denominado “Satélite” en el cual se nos muestra el mundo a base de fotografías, sin ningún tipo

de información adicional, simplemente las imágenes. Aquí podemos apreciar desde las más importantes cordilleras hasta las calles de las diferentes poblaciones del mundo dependiendo del nivel al que estemos. El tercero se llama “Híbrido” y es una superposición de la información del primero sobre las imágenes del segundo.
Entre los modos restringidos encontramos dos restringidos al ámbito de EE.UU. El primero es un modo “Tráfico” en el que se nos indican el estado de las principales vías de servicio. El segundo, es el modo “Calle”, solo posible en algunas ciudades estadounidenses en las cuales se nos muestra fotografías de la calle que seleccionemos en las cuales podemos movernos en 360º.
A diferencia que Virtual Earth, Google Maps no dispone de visor 3D. Para ello Google posee una herramienta especial, llamada Google Earth pero que funciona aparte y sin browser. De los que dispone Maps es de una vista en pseudo-3D en algunas ciudades americanas para el modo Mapa, pero no deja de ser una imagen en 2D con los edificios dibujados en perspectiva.

En los modos generales, se nos permite desplazarnos en horizontal (pan) y en vertical (zoom). Tanto con el ratón como con el teclado. La calidad del zoom depende de la posición geográfica que queramos ver, sobre todo en el modo “Satélite” e “Híbrido”. En los Estados Unidos suele haber más precisión, en el resto del mundo, cerca de las ciudades importantes también la precisión aumenta. En cualquier caso, el nivel de zoom para nosotros es bueno en cualquier punto del planeta.

Una de las grandes cosas de Google Maps, es el hecho que es muy sencillo agregar mapas o aplicaciones realizadas por otros usuarios, así el aspecto funcional es muy amplio.

C. Usabilidad.

C.1 A nivel de usuario.

La interficie es más simple y bastante menos atractiva, pero tiene casi la misma funcionalidad e intuitividad.

Para cambiar de modo hay botones con el nombre de cada modo (Mapa, Satélite, Híbrido). Para realizar zoom encontramos una barra desplazable o también se puede realizar con un doble click del ratón.

Para hacer pan encontramos una cruceta de flechas o también podemos realizarlo con el ratón manteniendo pulsado el botón izquierdo del mismo y moviéndolo. Mediante la cruceta el inconveniente es que no podemos realizar un desplazamiento en diagonal.

C.2 A nivel de desarrollador.

C.2.1 Posibilidad de realizar aplicaciones.

Google Maps esta diseñado para facilitar el desarrollo por parte de usuarios. Pudiendo hacer desde algo tan simple como marcar ubicaciones en mapas como aplicaciones más complicadas.

C.2.2 Existencia de Kit de Desarrollo de Software y Documentación sobre este.

Google Maps cuenta con una API totalmente gratuita. Esta API cuenta con documentación online (http://www.google.es/apis/maps/documentation/index.html) y con un foro para desarrolladores (http://groups.google.com/group/Google-Maps-API). También tiene más documentación fuera del sitio de google pero accesible desde este, como un tutorial bastante completo sobre como utilizar la API para hacer aplicaciones web (http://www.econym.demon.co.uk/googlemaps/) o una wiki (http://mapki.com/wiki/Main_Page).

Por otra parte, la API presenta métodos para utilizar imágenes, polígonos y polylíneas, necesarios para llevar a cabo nuestra tarea. No presenta ninguna función para obtener la altura de un punto del suelo con respecto al nivel del mar.

C.2.3 Requisitos para desarrollar.

Es necesario tener una clave de google que se nos concede al darnos de alta en una cuenta google. Esta clave es necesaria para poder hacer funcionar la

API que facilita google para poder mostrar contenido de Google Maps es una sitio web.

D. Confiabilidad.

No presenta ningún tipo de fallo o de mal funcionamiento. Es un producto más maduro que Virtual Earth y debido a eso se supone que ha de presentar menos bugs o errores funcionales.

E. Eficiencia.

Al igual que en Virtual Earth, la carga de mapas al hacer pan es más rápida que al hacer zoom. Los tiempos en ambos sentidos no son diferenciables a los de la anterior herramienta.

La carga de imágenes en modo “Satélite” también es algo más lenta que en el modo “Mapa”.

Hay que decir que, al presentar los edificios de algunas ciudades en un pseudo-3D esto no influye en el tiempo de carga ya que forman parte de la misma imagen. Al no ser un añadido la carga es inmediata, ya que se realiza al mostrar la imagen ya que es parte de la misma.

Resultado.

A nivel de portabilidad y eficiencia, ambos WMS están más o menos a la misma altura. También lo están en lo que a usabilidad del usuario se refiera.

La gran ventaja de Virtual Earth es que dispone de modo 3D. Esto es muy útil de cara a ver la simulación de manera más realista.
Por otra parte, el punto más débil es que se encuentra en versión beta. Esto hace que de vez en cuando su comportamiento no sea el esperado, pero también hace que todavía no haya mucha gente desarrollando para Virtual Earth y por lo tanto no resulte tan sencillo encontrar respuestas a algunas preguntas. Por ejemplo, una de las cosas que se ha observado es que al parecer, por lo que comentan algunos desarrolladores, hasta la versión 6 no se podía poner objetos a una altura diferente de la del suelo. Ahora en la sexta versión esto es posible, pero hay problemas para colocarlos a la altura adecuada porque no tiene en cuenta la altura respecto al nivel del mar sino que utiliza la relativa con la altura del suelo.

Google Maps cuenta como gran baza que se encuentra en un estado más maduro y que hay mucha gente detrás desarrollando y muchas aplicaciones hechas Esto significa que ya hay mucha gente que se ha encontrado con algunos problemas y por ello será más sencillo encontrar la solución. La debilidad de Google Maps es que visualmente no es tan atractivo como Virtual Earth y que no tiene el modo 3D integrado para ser utilizado en el browser.

ANEXO III: Gantt del proyecto.

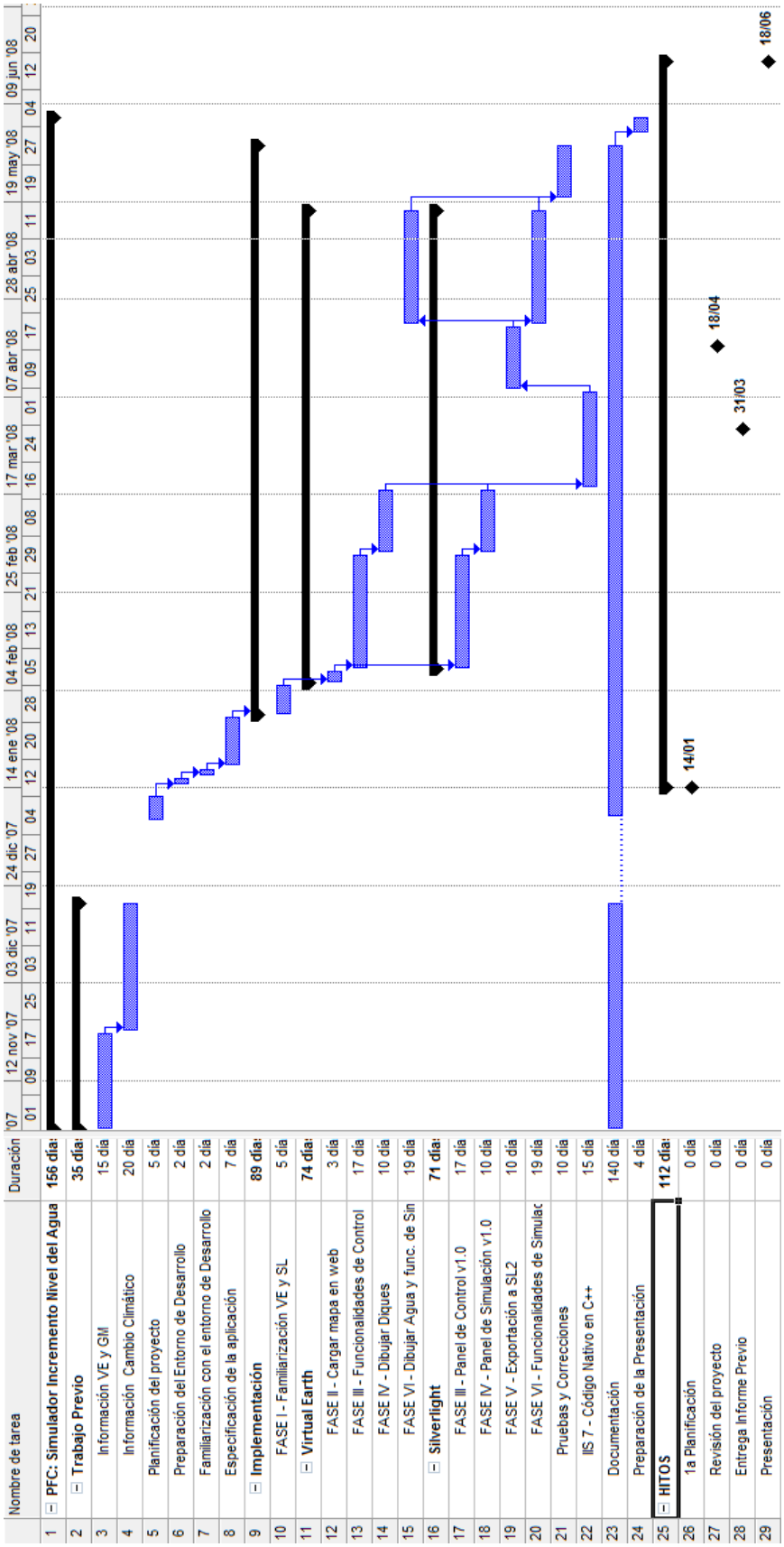


IMAGEN A3.1, DIAGRAMA DE GANTT DEL PROYECTO.

ANEXO IV: Documentación Clases C#.

A continuación se presenta la documentación de las clases implementadas en C#. Esta documentación ha sido autogenerada por el compilador de visual Studio al realizar los comentarios con tags XML. A este XML creado se le ha añadido un xls que le da formato.

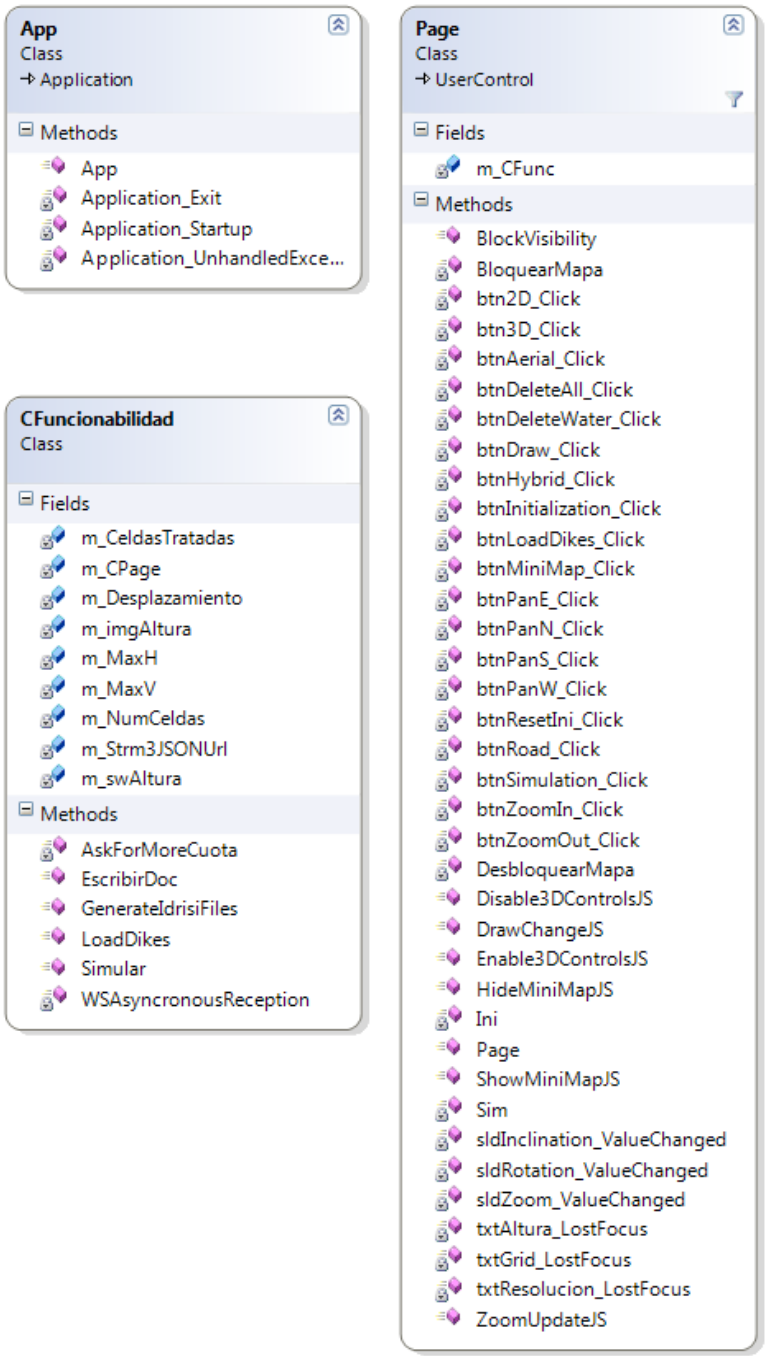


IMAGEN A4.1, DIAGRAMA DE CLASES, GENERADO POR VISUAL STUDIO.

aguaSIM.App

Remarks
Main class of the application. Auto-generate code by Visual Studio 2008.

Method	**App()**
Return	void.
Description	Constructor of the class. This function initializes the object. It defines the functions that handler the Startup event, the Exit event and the UnhandledException event.

Method	**Application_Startup(object, StartupEventArgs)**
Return	void.
Description	Function executed when the Startup event is throw. Create a instance of the class Page and sets it as the main application UI.

Method	**Application_Exit(object, EventArgs) method**
Return	void.
Description	Function executed when the Exit event is throw.

Method	**Application_UnhandledException(object, ApplicationUnhandledExceptionEventArgs)**
Return	void.
Description	Function executed when the UnhandledException event is throw**.**

aguaSIM.Page

Remarks
Class that contains the backstage of the application. This class implements the different functions and behaviour of the application's controls implemented in Page.xaml. Also, this class is defined as ScriptableType because it's necessary to invoke some of its methods from a javascript source.

Field **m_CFunc**
Description Instance of the class CFuncionabilidad. This class stores some relevant data that must persist for the next time we invoke one of its methods, so it's necessary to keep the instance and have only one.

Method **Page()**
Return void.
Description Constructor of the class. Creates the instance of CFuncionabilidad, initializes the diferents controls and registers the page as ScriptableObject in order to permit the access from a javascript source.

Method **txtAltura_LostFocus(Object,RoutedEventArgs)**
Return void.
Description This function handlers the losing of focus of the field "Grid's Hight". When "Dike's Hight" lose the focus, this function pass the value of the textedit to a javascript source, in order to the javascript source could draw the dikes with this altitude.
Parameters *sender:* Identifies the Silverlight object that generated the event.
e: References event data for a specific event.

Method **txtGrid_LostFocus(Object,RoutedEventArgs)**
Return void.
Description This function handlers the losing of focus of the field "Dike's Grid". When "Dike's Grid" loses the focus, this function pass the value of the texedit to a javascript source, in order to the javascript source could split the dike in diferents pices of this lenght.
Parameters *sender*: Identifies the Silverlight object that generated the event.
e: References event data for a specific event.

Method	**txtResolucion_LostFocus(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the losing of focus of the field "Resolution". When "Resolution" loses the focus, this function pass the value of the texedit to a javascript source, in order to the javascript source could grid the map in "Resolution"x"Resolution" of area cells's.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnInitialization_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnInitialization". When this button is clicked, the map is blocked and the program starts the initialization of the simullation, creating the files with the necessary information to begin the simulation.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnSimulation_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnSimulation". When this button is clicked the simulations begins. Before of this, the simulation must be initializated.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnLoadDikes_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnLoadDikes". When this button is clicked an openFileDialog is prompted. The user must to select a .img file that contains a unlimited number of dikes expresset in the Idrisi32 vectorial format. Then the dikes will be drawed.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnResetIni_Click(Object,RoutedEventArgs)**

Return void.
Description This function handlers the Click event over the button "btnResetIni". When this button is clicked the initialization of the simulation is canceled and the map is unblocked again.
Parameters *sender*: Identifies the Silverlight object that generated the event.
e: References event data for a specific event.

Method **btnDeleteAll_Click(Object,RoutedEventArgs)**
Return void.
Description This function handlers the Click event over the button "btnDeleteAll". When this button is clicked all the shapes of the map are automaticaly deleted. In oder words, all dikes and the polygons that represent the water will be deleted. To do that, the function invokes to a javascript source.
Parameters *sende*r: Identifies the Silverlight object that generated the event.
e: References event data for a specific event.

Method **btnDeleteWater_Click(Object,RoutedEventArgs)**
Return void.
Description This function handlers the Click event over the button "btnDeletewater". Polygons that represents the water are deleted. Also, to do that, the function invokes to a javascript source.
Parameters *sender*: Identifies the Silverlight object that generated the event.
e: References event data for a specific event.

Method **btnAerial_Click(Object,RoutedEventArgs)**
Return void.
Description This function handlers the Click event over the button "btnAerial". This function invokes to a javascript function that changes the style of the map to Aerial style.
Parameters *sender*: Identifies the Silverlight object that generated the event.
e: References event data for a specific event.

Method **btnRoad_Click(Object,RoutedEventArgs)**
Return void.
Description This function handlers the Click event over the button "btRoad".

	This function invokes to a javascript function that changes the style of the map to Rode style.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e:* References event data for a specific event.

Method	**btnHybrid_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnHybrid". This function invokes to a javascript function that changes the style of the map to Hybrid style.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btn2D_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btn2D". This function invokes to a javascript function that changes the mode of the map to 2D Mode. Also disables the specific controls of the 3D Mode(Pitch and Rotation) and enables the simulations ones.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btn3D_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btn3D". This function invokes to a javascript function that changes the mode of the map to 3D Mode. Also disables the simulation controls and enables the 3d Mode controls(Pitch and Rotation).
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnZoomOut_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnZoomOut". Decrease by 1 the value of the zoom slider.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e:* References event data for a specific event.

Method	**btnZoomIn_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnZoomIn". Increase by 1 the value of the zoom slider.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**sldZoom_ValueChanged(Object,Double)**
Return	void.
Description	This function handlers the ValueChanged event over the button "sldZoom". Invokes to a javascript function that set the zoom of the map with the new value of the slider.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnPanS_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnPanS". Invokes to a javascript function that pan the map to the south.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnPanE_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnPanE". Invokes to a javascript function that pan the map to the east.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnPanW_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnPanW". Invokes to a javascript function that pan the map to the west.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnPanN_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnPanN". Invokes to a javascript function that pan the map to the north.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**sldRotation_ValueChanged(Object,Double)**
Return	void.
Description	This function handlers the ValueChanged event over the button "sldRotation". Invokes to a javascript function that set the heading of the map with the new value of the slider. A value of 0 is true north, and a value of 180 is true south. Values less than 0 and greater than 360 are valid and are calculated as compass directions.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**sldInclination_ValueChanged(Object,Double)**
Return	void.
Description	This function handlers the ValueChanged event over the button "sldInclination". Invokes to a javascript function that set the pitch of the map with the new value of the slider. A value of 0 is level and a value of -90 is straight down. Values less than -90 or greater than 0 are ignored, and the pitch is set to -90.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnMiniMap_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnMiniMap". This function calls to a javascript function that shows or hide a minimap in the top left corner of the map.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**btnDraw_Click(Object,RoutedEventArgs)**
Return	void.
Description	This function handlers the Click event over the button "btnDraw". This function enables the drawing dikes functionality clicking over the map.
Parameters	*sender*: Identifies the Silverlight object that generated the event. *e*: References event data for a specific event.

Method	**DrawChangeJS()**
Return	void.
Description	Function that set as checked/unchecked the check button btnDraw. It is called from a javascript source when the enablesDraw event occurs over the map.

Method	**ShowMiniMapJS()**
Return	void.
Description	Function that set as checked the check button btnMiniMap. It is called from a javascript source when the show minimap event occurs over the map.

Method	**HideMiniMapJS()**
Return	void.
Description	Function that set as unchecked the check button btnMiniMap. It is called from a javascript source when the hide minimap event occurs over the map.

Method	**Enable3DControlsJS()**
Return	void.
Description	Function that enables the 3d mode controls and disables de simulation controls. It is called from a javascript function when the map mode change to 3d mode by a map event.

Method	**Disable3DControlsJS()**
Return	void.
Description	Function that disables the 3d mode controls and enables de simulation controls. It is called from a javascript function when the map mode change to 2d mode by a map event.

Method	**ZoomUpdateJS(Int32)**
Return	void.
Desfription	Function that updates the value of the zoom slider. It is called from a javascript source when the zoom level is changed by a mouse event over the map.
Parameters	*level*: Int that represents the level of the zoom

Method	**Ini(Object,RunWorkerCompletedEventArgs)**
Return	void.
Description	This function calls to the function that implements the generation of the idrisi32 files representing the dikes and the terrain altitude, placed in the CFuncionabilidad Class. Also, this function blocks the map in order to not chage the view until the simulation take place.
Parameters	*sender*: *e*: An EventArgs that contains the event data.

Method	**Sim(Object,RunWorkerCompletedEventArgs)**
Return	void.
Description	This function calls to the m_CFunc function that calls the simulation motor. Also controls the unblocking of the map.
Parameters	*sender* *e*: An EventArgs that contains the event data.

Method	**BlockVisibility()**
Return	void.
Description	This function disables the controls of the right panel when the application is doing some processing.

Method	**DesbloquearMapa()**
Return	void.
Description	This function enables the map's controls and the map's mouse events.

Method	**BloquearMapa()**
Return	void.
Description	This function disables the map's controls and the map's mouse events.

aguaSIM.Funcionalidad.CFuncionabilidad

Remarks
Class that contains the functions related to the simulation and the creation of the idrisi32 format files.

Field **m_CPage**
Description Instance of the class Page.

Field **m_Strm3JSONUrl**
Description Constant string that contains the uri of the webservice called to find the terrain altitude. See geonames.org

Field **m_NumCeldas**
Description Total number of cells of the map grid.

Field **m_CeldasTratadas**
Description Number of cells that have been processed.

Field **m_Desplazamiento**
Description Number of pixels of long of each cell of the map grid.

Field **m_MaxV**
Description Number of columns of the map grid.

Field **m_MaxH**
Description Number of rows of the map grid.

Field **m_swAltura**
Description StreamWriter of the file altura.img. It is needed to write asyncronously in this file while receiving the data from an asyncronous web service.

Field **m_imgAltura**
Description IsolatedStorageFileStream of the file Altura.img. It is needed to write asynchronously in this file while receiving the data from an asynchronous web service.

Method **EscribirDoc(String,StreamWriter)**
Return void.
Description Function that writes the files .doc. This files are the metadata for idrisi32 format.
Parameters *info*: String with the information to write.
swDoc: StreamWriter associated with the file where the function writes.

Method **Simular()**
Return void.
Description Calls to the simulation motor.

Method **GenerateIdrisiFiles(Page)**
Return void.
Description Function that creates the Idrisi32 files with the information of the dikes and the altitude of the terrain.
The function grids the map, and goes through each cell of the grid, checking if there is any dike and invoking a webservice to get the altitude of the terrain. This webservice is asynchronous so the file with the altitude is writing in the fucntion "RecepcionAsincronaWS".
The files are stored in the IsolatedStorage of the application.
Parameters *CPage*: Instance of the class Page. It is needed to unblock when the asynchronous reception is finish.

Method **WSAsyncronousReception(Object,DownloadStringCompletedEventArgs)**
Return void.
Description This function receives the data from a asynchronous webservice from www.geonames.org.
Asynchronously, this function receives the data representing the altitude of the terrain and writes it in the altura.img file.
Parameters *sender*: The source of the event.
e: A DownloadStringCompletedEventArgs that contains event data.

Method	**LoadDikes(StreamReader)**
Return	void.
Description	This function reads a file that represents dikes in Idrisi32 vectorial format, and draw this dikes into the map.
Parameters	*reader*: StreamReader of the file that contains the dikes to draw in Idrisi32 vectorial format

Method	**AskForMoreCuota(IsolatedStorageFile)**
Return	void.
Description	This function asks to the user for increase the cuota for isolatedstorage. It would fix the cuota in 10mb
Parameters	*store*: represents the Isolated Storage of the application

ANEXO IV: Documentación Funciones javascript.

A continuación se presenta la documentación de las funciones implementadas en javascript referentes a la manipulación de los mapas Virtual Earth. A diferencia de la documentación para C#, esta ha tenido que ser escrita y no es autoimplementada.

VEMap.js

Remarks
VEMap.js is a javascript file that contains all the functions needed in the project that involves a Virtual Earth map or component. For information about Virtual Earth functions see VirtualEarthSDK60.

Method	**GetMap()**
Return	void.
Description	This is the main function of the file. This function creates and initializes the map object. Defines the functions that handler the different types of events(onmouseover, onclick, ondoubleclick) and creates the shape layers needed for drawing.

Method	**OnMouseOverEventHandler(EventObject)**
Return	void.
Description	Function executed when the onMouseOver event is throws.
Parameters	*e*: References event data for a specific event.

Method	**OnClickEventHandler(EventObject)**
Return	void.
Description	Function executed when the right button of the mouse is clicked over the map. Is used to select or deselect a dike and also to draw a new one.
Parameters	*e*: References event data for a specific event.

Method	**OnKeyPressEventHandler(EventObject)**

Return void.

Description Function executed when a key of the keyboard is pressed when the focus is on the map. Is used to do events clicking the numbers of the keyboard:

KEY	FUNCTION
2	Change to 2D Mode.
3	Change to 3D Mode.
4	Enable Draw,
5	Delete selected dike.
6	Deselect selected dike.
7	Delete polygons (water)
8	Delete all shapes (water and all dikes)
9	Show/Hide MiniMap

Parameters *e*: References event data for a specific event.

Method **OnFocusEventHandler(EventObject)**

Return void.

Description Function executed when the map gets the focus. It is used to change the mouse icon to a cross when the draw function is enabled.

Parameters e: References event data for a specific event.

Method **OnDoubleClickEventHandler(EventObject)**

Return void.

Description Function executed when the right button of the mouse is doubleclicked over the map. It is used to zoom using only the mouse.

Parameters *e*: References event data for a specific event.

Method **SetZoom(Integer)**

Return void.

Description Function called to change the zoom of the map.

Parameters *newLevel*: References the new zoom level.

Method **PanNorth()**

Return void.

Description Function called to pan to the north in the 2D mode. In the 3D mode, this function pans forward.

Method **PanSouth()**
Return void.
Description Function called to pan to the south in the 2D mode. In the 3D mode, this function pans backward.

Method **PanWest()**
Return void.
Description Function called to pan to the west in the 2D mode. In the 3D mode, this function pans left.

Method **PanEast()**
Return void.
Description Function called to pan to the east in the 2D mode. In the 3D mode, this function pans right.

Method **Rotate(Double)**
Return void.
Description Function called to change the orientation of the map. Only available in 3D mode.
Parameters *heading*: References the compass direction, expressed as a double. A value of 0 is true north, and a value of 180 is true south. Values less than 0 and greater than 360 are valid and are calculated as compass directions.

Method **Inclination(Double)**
Return void.
Description Function called to change the inclination of the map. Only available in 3D mode.
Parameters *pitch***:** References the pitch direction, expressed as a double. A value of 0 is level and a value of -90 is straight down. Values less than -90 or greater than 0 are ignored, and the pitch is set to -90.

Method **ThreeD()**
Return void.

Description Function called to change the view mode to 3D mode.

Method **TwoD()**
Return void.
Description Function called to change the view mode to 2D mode.

Method **Road()**
Return void.
Description Function called to change the view mode to Road mode.

Method **Aerial()**
Return void.
Description Function called to change the view mode to Aerial mode.

Method **Hybrid()**
Return void.
Description Function called to change the view mode to Hybrid mode.

Method **EnableDraw()**
Return void.
Description Function called to change state of the draw function, enabling or disabling the drawing functionality.

Method **ShowMiniMap()**
Return void.
Description Function called to show a small map in the top left corner of the map.

Method **HideMiniMap()**
Return void.
Description Function called to show a small map in the top left corner of the map.

Method **SelectPolyline(EventObject)**
Return void.

Description Function called to select a dike. This function select a dike, changing its colour and initializing the parameters needed if we are going to continue drawing points for this dike. If there is any other dike selected, this dike will be recoloured as a non-selected one.
Parameters *e*: References event data for a specific event. It is necessary to obtain the shape id of the new dike selected.

Method **DrawtPolyline(EventObject)**
Return void.
Description Function called to draw a dike. This function draws a new dike or a new point of an existing dike. To draw a new point of an existing dike this dike must be selected. Every time a new point is drawed, the longitude of the dike is recalculated.
Parameters *e*: References event data for a specific event. It is necessary to obtain the geofrafic reference (latitude/longitude) of the new point.

Method **DeletePolyline()**
Return void.
Description Function called to delete the selected dike from the map.

Method **DeselectPolyline()**
Return void.
Description Function called to deselect selected dike from the map without select another one.

Method **DrawtPolygon(Array, Integer)**
Return void.
Description Function called to draw a polygon that represents the water.
This function draws a new polygon. This polygon will be placed in the coordinates that are stored in the array latLongs.
Parameters *latLongs*: References an array of LatLong objects. Each one of the objects represents a vertex of the polygon.

Method **DeletePolygons()**
Return void.

Description Function called to delete from the map all the shapes drew over the "agua" layer.

Method **DeleteAllShapes()**
Return void.
Description Function called to delete from the map all the shapes drew over the "agua" layer and over the "diques" layer.

Method **GetDistance(Array)**
Return void.
Description Function called to calculate a dike longitude in kilometres.
This function calculate the longitude working with two pairs of latLong.
Tranforms latLong to Radians and then applies the Haversine formula:

R = earth's radius (mean radius = 6,371km)
Δlat = lat2− lat1
Δlong = long2− long1
cos(lat1).cos(lat2).sin²(Δlong/2)
c = 2*atan2(√a, √(1−a))
distance = R*c

Parameters *puntos*: References an array of LatLong objects. Each one of the objects represents a point of the selected dike.

Method **latLongToRadians(latLong)**
Return void.
Description Function called to transform a coordinate in latitude/longitude to radians.
Parameters *point*: References the LatLong object that has to be transformed.

Method **TramarMuro(latLong, latLong, Array)**
Return void.
Description Function called to divide the part of a polyline that goes from p1 to p2 in segments of the same longitude.
This longitude is defined as a global parameter. Initially, ptsM are all the points of the array before segment the new part. Finally, in the array ptsM are the points mentioned before and all the new points created by the segmentation.
Parameters *p1*: LatLong object that starts the part of the polyline.
p2: LatLong object that ends the part of the polyline.

ptsM: Array of LatLong objects that represent the points of the polyline.

Method **txtGridMuros_onChange(Double)**
Return void.
Description Function called from Silverlight's C# code to store the value of the longitude to perform the segmentation of the polylines.
Parameters *gridMuros*: Longitude in kilometres to segment the polylines.

Method **txtAlturaMuros_onChange(Double)**
Return void.
Description Function called from Silverlight's C# code to store the value of the altitude of the dikes.
Parameters *alturaMuros*: Numeric value that represents in meters the altitude of the dikes.

Method **txtResolucion_onChange(Double)**
Return void.
Description Function called from Silverlight's C# code to store the distance used in order to grid the map.
Parameters *resolucion*: Numeric value that represents in meters the distance used to grid the map.

Method **ObtenerDesplazamiento()**
Return *Array*: Array of values that represents the following information:

INDEX	INFORMATION
0	Longitude of each cell (Pixels).
1	Number of horizontal cells.
2	Number of vertical cells
3	Top left corner's Longitude
4	Bottom right corner's Latitude
5	Top left corner's Latitude
6	Bottom Right corner's Longitude
7	Longitude of each cell (Meters).

Description Function called from Silverlight's C# code to calculate the number of pixels that has each cell of longitude.
The function gets the distance in meters represented by the 690 pixels of width that has the map. Then divide this distance between the value of the grid's resoluction to obtain the number of horizontal cells. Finally, calculate the pixels of longitude of each cell dividing the map's width between the number of horizontal cells.
Also calculates the geographic coordinates of the top left and bottom right corners.

Method **txtResolucomprobarInterseccion(Double, Double, Double, Double, Double, Double, Double, Double)**

Return *Array*: Array of values that represents the following information:

INDEX	INFORMATION
0	Boolean that shows if the lines intersect.
1	X component of the crossing point. If there is no point, return 0.
2	Y component of the crossing point. If there is no point, return 0.

Description Function called to calculate if two lines intersect or not. Using line equations, this function determinates if two lines intersect in any point. If it is true, the function returns the crossing point.

Parameters *lineAx1*: X component of the first point of the first line.
lineAy1: Y component of the first point of the first line.
lineAx2: X component of the second point of the first line.
lineAy2: Y component of the second point of the first line.
lineBx1: X component of the first point of the second line.
lineBy1: Y component of the first point of the second line.
lineBx2: X component of the second point of the second line.
lineBy2: Y component of the second point of the second line.

Method **contieneMuro(Double, Double, Double)**

Return *Integer*: Altitude of the dike that is in the cell. If there isn't any dike, return 0.

Description Function called from Silverlight's C# code to calculate if there is any dike in the cell we are studying.
To do that, the function calculates the 4 corners of the cell, making 4 different lines. Then verifies if any of the dikes drew in the map intersects with any of the 4 lines that represent the cell.

Parameters *pMedX*: X Component of the pixel that is in the center of the cell.

pMedY: Y Component of the pixel that is in the center of the cell.
Despl: Pixels of longitude of the cell.

Method	**CargarMuro(string)**
Return	void.
Description	Function called from Silverlight's C# code to draw a dike that was represented in a idrisi32 format file.
Parameters	*String*: String that represents a dike. This string is compose of pairs of values that represent the latitude and the longitude of one point of the dike.

Method	**DibujarAgua(Double, Double, Double, Double)**
Return	void.
Description	Function called from Silverlight's C# code to draw a polygon that represents the new sea level. This function gets the center of the cell in pixels, calculates the 4 corners of the cell, transformer these pixels into LatLongs and calls DrawPolygon function.
Parameters	*pMedX*: X Component of the pixel that is in the center of the cell. *pMedY*: Y Component of the pixel that is in the center of the cell. *despl*: Pixels of longitude of the cell. *altura*: Altitude over the sea level of the polygon.

Method	**ConvertirPixelALatLon(Double, Double)**
Return	*Array*: Array with two values: Latitude and Longitude of the coordinate.
Description	Function called from Silverlight's C# transform a pixel into a coordinate Latitude Longitude.
Parameters	*pMedX*: X Component of the pixel to convert. *pMedY*: Y Component of the pixel to convert.

Method	**DisableMouseEvent ()**
Return	void.
Description	Function called to attach an event handler that does nothing to the mouse event, in order to prevent the interaction with the map when simulating.

Method	**DisableMouseEventHandler ()**
Return	void.
Description	Function attached to mouse events as onmousedown, ondoubleclick, onmousewheel to prevent any move of the map.

Method	**EnableMouseEvent ()**
Return	void.
Description	Function called to detach the event handler that does nothing when mouse event.

ANEXO VI: Problema Geoid vs. Elipsoide WGS84.

La Tierra, aunque se representa como una esfera, su forma geométrica no es ni mucho menos perfecta. Para tratar diferentes aspectos científicos sobre el nivel del mar, se utiliza una compleja aproximación de la tierra llamada geid.

El geoid[7] se define como la superficie de nivel, equipotencial en el campo de la gravedad, que adopta la forma de esferoide irregular tridimensional - debido a que depende de la distribución de masas en el interior de la Tierra, es imposible de representar matemáticamente - para ello se utiliza el elipsoide de referencia que más se le aproxime o ajuste - es coincidente con la superficie del agua en reposo de los océanos, extendida virtualmente por debajo de los continentes, de manera que la dirección de las líneas de plomada crucen perpendicularmente esta superficie en todos sus puntos.

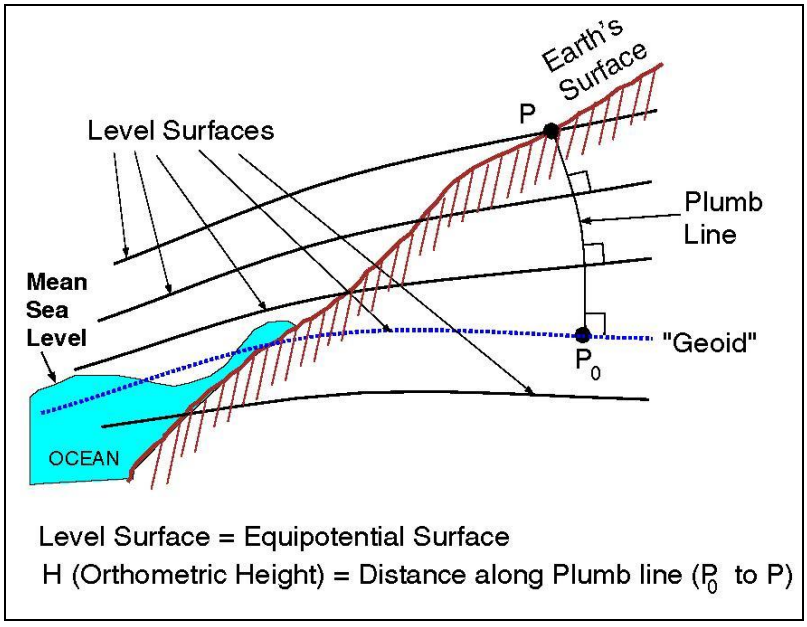


IMAGEN A6.1, ESQUEMA SUPERFICIES DE NIVEL Y GEOID.

El geoide no es plenamente coincidente con la forma de la Tierra y tampoco es coincidente con ninguna forma geométrica. La forma geométrica más parecida y a la

[7] También se puede encontrar por geoide o geodesia.

cual se aproxima en muchas ocasiones para facilitar cálculos es el elipsoide de revolución.

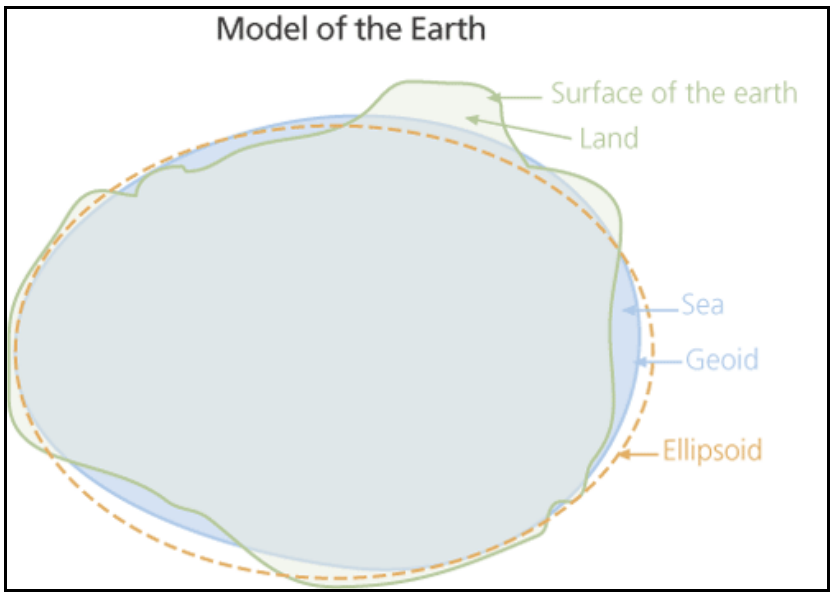


IMAGEN A6.2, ESQUEMA GEOID Y ELIPSOIDE.

Hay diferentes elipsoides que se utilizan para aproximar el geoid. El más utilizado es el elipsoide WGS84, usado por ejemplo por la tecnología GPS.

Como ya se ha comentado, el elipsoide WGS84 no coincide exactamente con el geoide, ya que el primero es una forma geométrica totalmente lisa mientras que el segundo no es liso. Esto hace que existan diferencias sutiles cuando se comparan.

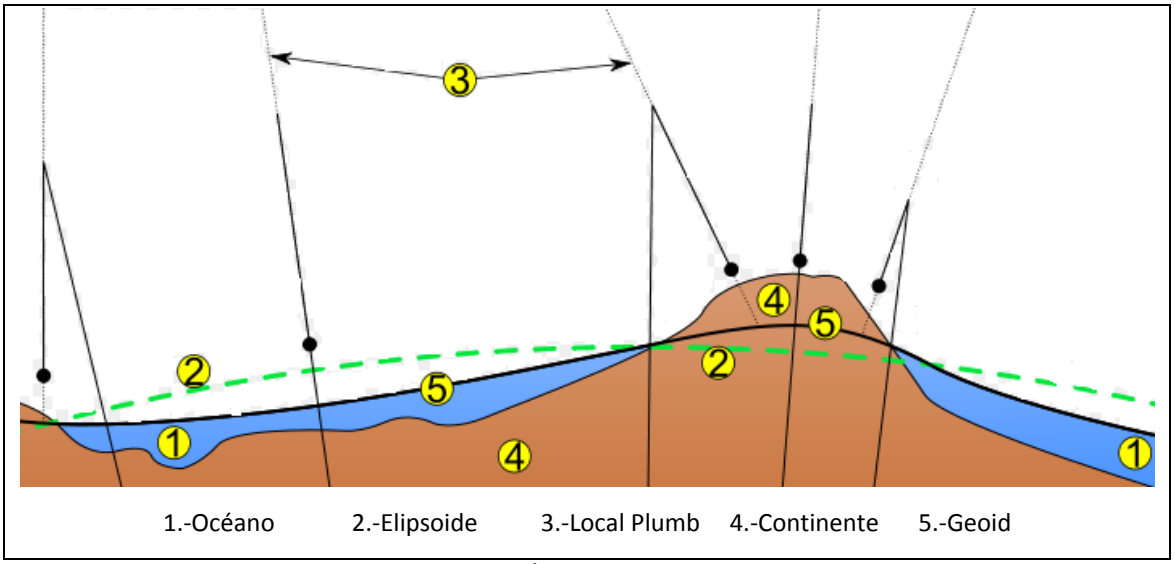


IMAGEN A6.3, OCÉANO, ELIPSOIDE Y GEOID.

En el caso que nos atañe, los mapas Virtual Earth tienen dos métodos para expresar la altura, el dependiente de la altura del suelo y el absoluto, dependiente del nivel del mar. Ahora bien, este nivel del mar lo define mediante el elipsoide WGS84, lo que supone una aproximación no solo a la realidad si no una aproximación al geoid.

Es decir, la altura a la que están dibujados los mapas es la altura del geoid, sin en cambio la altura a la que se dibujan los elementos creados sobre el mapa se representan según el elipsoide WGS84.

Esto hace que al dibujar un polígono a una altura determinada nos podamos encontrar con 3 situaciones diferentes dependiendo del punto de la Tierra en el que nos encontremos:

1. El polígono muestra una altura inferior a la que se ha indicado.

   Esto es debido a que el elipsoide en ese punto se encuentra por debajo del geoide. Imaginemos que dibujamos un polígono a 100 metros por encima del elipsoide, estando este a 47 por debajo del geoide. Nuestro polígono se verá situado a unos 53 metros por encima del nivel del mar. Este el caso por ejemplo de Barcelona.
2. El polígono muestra una altura superior a la que se ha indicado.

   Esto es debido a que el elipsoide esta por encima del geoide. Por ejemplo sucede en Nueva York, donde al dibujar un polígono a 100 metros se verá representado a unos 130.
3. El polígono esta correctamente colocado.

   Encontrar un punto que cumpla esto puede ser todo un milagro.

Aun que la diferencia más importante que me he encontrado es de entorno a los 50 metros, es sabido que la mayor anormalidad entre el geoid y el WGS84 se encuentra en el sudeste de India, situándose el geoid 105 metros por debajo del elipsoide WGS84.

Índice de Imágenes.