2059 – Entorno gráfico para simulaciones de sistemas biológicos

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
David Rel Grau
i dirigit per
Diego Javier Mostaccio Mancini
Bellaterra, 8 de Febrer de 2010

Universitat Autònoma de Barcelona

etse

Escola Tècnica Superior d'Enginyeria

El sotasignat, Diego Javier Mostaccio Mancini

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en David Rel Grau

I per tal que consti firma la present.

Signat: Diego Javier Mostaccio Mancini

Bellaterra, 8 de Febrer de 2010

1. INTRODUCCIÓN 7

1.1 MOTIVACIÓN 7

1.2 FINALIDAD Y OBJETIVOS DEL PROYECTO 8

1.2.1 Finalidad del proyecto 8

1.2.2 Objetivos 8

1.3 ESTADO DEL ARTE 9

1.4 ESTRUCTURA DE LA MEMORIA 11

2. VIABILIDAD Y PLANIFICACIÓN 13

2.1 INTRODUCCIÓN 13

2.2 MERCADO AL QUE VA DIRIGIDA 13

2.3 MODELO DE DESARROLLO DEL SOFTWARE 15

2.4 ALTERNATIVAS DE DESARROLLO 17

2.4.1 API gráfica 17

2.4.2 Lenguaje de programación 19

2.4.3 Otras api's – CEGUI 20

2.5 RECURSOS MATERIALES Y COSTES. 22

2.6 ANÁLISIS DE RIESGOS. 23

2.7 PLANIFICACIÓN TEMPORAL 26

2.8 VIABILIDAD DEL PROYECTO 27

3. REQUERIMIENTOS Y ANÁLISIS FUNCIONAL 29

3.1 REQUERIMIENTOS 29

3.2 ANÁLISIS FUNCIONAL 32

3.2.1 Menú principal: 32

3.2.2 Gui de la simulación: 33

3.2.3 Configuración simulación: 36

3.2.4 Configuración gráficos: 37

4. DISEÑO 41

4.1 ESTRUCTURA DE LA APLICACIÓN 41

4.1.1 Managers – <EscenaMgr> y <MenuMgr> 41

4.1.2 Agentes - <Individuo> 42

4.1.3 Raiz - <rootsingleton> 42

4.1.4 Máquina Estados - <máquinaestados> 42

4.1.5 Aplicación principal – <simulacion3dapp> 43

4.1.6 Bucle de renderizado - <MiFrameListener> 43

4.2 Diagrama de componentes 44
4.3 Diagrama de clases 45
4.4 Diagrama de secuencias 46
4.4.1 Modificación de propiedades de configuración. 46
4.4.2 Carga de visualización de simulación 46
4.4.3 Interacción con la GUI 48

5. IMPLEMENTACIÓN 51

5.1 Conceptos básicos sobre las API's 51
5.1.1 OGRE 3D 51
5.1.1.1 Root 51
5.1.1.2 RenderSystem 51
5.1.1.3 SceneManager 52
5.1.1.4 ResourceGroups 52
5.1.1.5 Meshes (Entity) 53
5.1.1.6 SceneNodes 53
5.1.1.7 Materials 54
5.1.1.8 Animations 55
5.1.1.9 FrameListener 56
5.1.2 CEGUI 56
5.1.2.1 OgreCEGUIRenderer 56
5.1.2.2 System 56
5.1.2.3 Windows 57
5.2 Implementación de la aplicación 58
5.2.1 Algoritmo general 58
5.2.2 Aplicación de visualización de simulación 3d 60
5.2.2.1 Inicialización OGRE: 60
5.2.2.2 Crear Escena: 61
5.2.2.3 Carga de la visualización de simulación: 62
5.2.2.4 Creación CEGUI y control de menús: 65

6. PRUEBAS Y CONCLUSIONES 75

6.1 Pruebas 75
6.1.1 Medidor – Frames por segundo 75
6.1.2 Medidor – Memoria RAM consumida. 80
6.1.3 Requisitos mínimos 83
6.2 Problemas encontrados 84
6.3 Posibles mejoras 85
6.4 Conclusiones 86

7. BIBLIOGRAFÍA 89

8. ANEXOS 93

8.1 ANEXO 1 - MANUAL DE USUARIO 93

8.1.1 Instalación del visualizador de simulaciones 93

8.1.2 Configuración del visualizador de simulaciones 94

8.1.2.1 Incorporar los ficheros de salida de las simulaciones 94

8.1.2.2 Primera ejecución 96

8.1.3 Manejo de la aplicación 97

8.1.3.1 Menú principal: 97

8.1.3.2 Configuración visualización: 97

8.1.3.3 Configuración gráficos: 98

8.1.3.4 GUI entorno de visualización: 98

8.1.4 “Shortcuts” o atajos con el teclado 102

8.2 ANEXO 2 - EXPLICACIÓN DETALLADA DE LAS CLASES 103

8.2.1 Invididuo: 103

8.2.2 BaseManager: 104

8.2.3 EscenaManager: 104

8.2.4 MenuManager: 107

8.2.5 RootSingleton: 113

8.2.6 MáquinaEstados: 113

8.2.7 MiFrameListener: 116

8.2.8 Simulacion3DApp: 118

1. Introducción

Este capítulo enseñará las motivaciones, finalidad y objetivos del proyecto. Es un punto importante ya que se especificarán las metas a las cuales se quiere llegar y mostrar, en su conjunto, para qué servirán.

1.1 Motivación

La motivación principal de este proyecto ha sido sumergirse en el mundo de la visualización tridimensional en tiempo real intentando dar el mayor realismo posible e investigar sobre las distintas técnicas gráficas que permitan realizar esto.
Estas técnicas han sido aplicadas para implementar un visualizador de simulaciones, en este caso de agentes biológicos. Éste es un tema interesante a tratar por el gran abanico de posibilidades que existen dentro del campo de la simulación. La toma de decisiones se lleva a cabo mediante pautas de comportamiento barajando las acciones que los agentes van a tomar en un entorno determinado. Aquí no se tratará la simulación en sí, sino la representación gráfica tridimensional de los datos ya generados por el simulador.

Otra de las motivaciones ha sido la adquisición de experiencia en las distintas fases de desarrollo de un software. Realizar un buen diseño puede facilitar en gran medida el desarrollo de éste con lo que se podrá realizar un mejor mantenimiento y posibles ampliaciones requeridas en un futuro sin grandes complicaciones.

1.2 Finalidad y objetivos del proyecto

1.2.1 Finalidad del proyecto

La finalidad del proyecto es la realización de un software realista que permita una visualización tridimensional post-mortem de una simulación de agentes biológicos en función de los datos generados por el simulador (independientes a éste proyecto).
Se quiere puntualizar con la palabra software, no solo a la aplicación, sino a toda la espiral de desarrollo de éste creando documentos de requerimientos, análisis, diseño, desarrollo, pruebas y conclusiones.

1.2.2 Objetivos

Aquí se enumeran los objetivos del proyecto en base a la finalidad del mismo.

- Elección de un motor gráfico 3D con licencia GPL que se adapte a los requerimientos eficientemente.

- Diseño orientado a objetos con sistema modular, independiente y fácilmente modificable previsto para posibles modificaciones y/o ampliaciones en un futuro.

- Realismo en la representación de la simulación mediante técnicas gráficas especiales.

- Gráficos y visualización de la simulación adaptables a los requisitos de la máquina en la que se ejecuta para poder acelerar los FPS (frames por segundo) y poder mostrar fluidez en la representación.

- Personalización de la representación de la simulación adaptándola a conveniencia del usuario en función del tipo de agentes biológicos que utilicemos y tipo de representación que queramos mostrar.

- Interacción con el mundo 3D modificando en tiempo real la visualización de la simulación para mayor interactividad.

- Uso de una interfaz de usuario gráfica amigable y de fácil manejo que permitirá el uso de la aplicación a usuarios con nivel básico de informática, sin la necesidad de la lectura de manuales ni el aprendizaje de combinaciones de teclas complicadas.

1.3 Estado del arte

Los gráficos son una parte muy importante en el mundo actual y sobretodo en el ámbito de las tecnologías.
El 83% de la información la recibimos por la vista. Estamos acostumbrados a ver gráficos por todos los lados sin saber realmente el impacto que tienen sobre nosotros. En un intervalo muy pequeño de tiempo somos capaces de absorber mucha más información gráficamente que usando los cuatro sentidos restantes que tenemos. [19]
El mundo de los gráficos 3D ha sido el determinante y un punto y aparte en nuestra escalada en la era tecnológica.
La diferencia entre un gráfico 2D y otro 3D es la manera en que éste ha sido generado. Estos gráficos se originan mediante unas reglas y procesos matemáticos que son capaces de describir analíticamente el mundo en el que nos movemos. Una vez tenemos estos modelos matemáticos se plasma una proyección como si a un papel fuese desde un punto de vista determinado dentro del modelo, es decir, como una fotografía. Esto permite recrear mundos con un nivel de detalle cada vez mayor y poderlos usar para multitud de aplicaciones. [20]
La aparición de las GPU (*Graphic Process Unit*) permitió ejecutar programas (*shaders*) en ellas incrementando así la capacidad de cómputo de los gráficos. Estos pequeños programas se aplican por cada píxel o vértice y ayudan a dar mayor realismo a la escena.

La simulación de altas prestaciones es una tendencia que está en auge en este nuevo siglo, pudiendo predecir comportamientos aproximados de sistemas

reales realizando un estudio de comportamiento de éstos basados en reglas. [21]

Se está aplicando actualmente en muchos ámbitos como la medicina [22], biología [23] o meteorología [24].
Se utilizan entornos distribuidos para poder realizar los cálculos más rápidamente, ya que si no sería imposible en según qué aplicaciones. Posteriormente, los datos que se generan por separado, se fusionan de determinada forma que no hayan ambigüedades para poderlos interpretar correctamente.

Hace poco tiempo se tenían que interpretar cantidades de datos prácticamente ilegibles para quienes tenían que hacerlo.

En estos últimos años se han mezclado los datos con los gráficos para facilitar su interpretación. Es mucho más fácil para una persona observar cómo realmente se está moviendo un agente que observando una ristra de datos casi infinita.

La simulación gráfica en 3D se ha hecho generalmente con pocos agentes y en tiempo real de ejecución. La inteligencia artificial de los videojuegos actuales es el mejor ejemplo de simulación en tiempo real. Juegos tales como Half Life, Sims y Flight Simulator contemplan una cantidad baja de agentes.
Por ejemplo, en los Sims se utiliza una máquina de estados finita, con un alto número de estados mezclado con técnicas de vida artificial propias del motor de los Sims.
Half-life por ejemplo está basado en distancias y en probabilidades aleatorias para dar más o menos realismo a las acciones que va a tomar el enemigo. [25]

Dado que la realidad contempla un alto número de condiciones y la capacidad de cómputo actual aun no es la suficiente, la tendencia ha sido paralelizar esas aplicaciones y generar sus datos, que posteriormente podrán ser simulados.
Tenemos multitud de proyectos, por ejemplo uno de ellos es “Computación de altas prestaciones sobre entornos grid en aplicaciones biomédicas”. En este

caso los recientes incrementos en el ancho de banda de las redes de comunicaciones han propiciado la idea de unir recursos computacionales geográficamente distribuidos y por lo tanto poder paralelizar los cálculos entre ellos. Posteriormente, éstos, son procesados gráficamente analizando la actividad eléctrica cardiaca y el diseño de proteínas [26].

1.4 Estructura de la memoria

La memoria está estructurada en seis capítulos a los que se les ha añadido los anexos y la bibliografía.

El primer punto será una introducción del proyecto. Aquí se contemplará, a grandes rasgos, qué ha motivado a la elaboración de éste, su finalidad y los objetivos concretos.

En el segundo apartado se hará un análisis previo a la elaboración del proyecto contemplando si es viable su realización analizando las distintas opciones o caminos, sus costes, establecimiento del tiempo del proyecto, etc.

Después del capítulo de análisis de viabilidad se escogen los requerimientos del software y se realizará un análisis funcional indicando qué funciones podrá utilizar el usuario dentro de la aplicación.

El cuarto punto trata sobre el diseño de la aplicación. Aquí se analizará la estructura de las clases y funcionalidades internas y externas creando una estructura eficiente para la posterior ampliación o mantenimiento del código. También se podrá contemplar el acceso o interfaces de comunicación entre clases.

El quinto punto es el de implementación. Se dará una breve visión de la estructura de las API's y de su funcionalidad interna. Siguientemente se muestra la explicación de cómo se ha implementado la aplicación internamente.

En el sexto punto se da una visión de los resultados obtenidos sobre la visualización de simulaciones dadas (análisis de rendimiento) y posibles mejoras sobre el software en un futuro.

Finalmente se muestra la bibliografía utilizada y todos los anexos referentes al proyecto.

2. Viabilidad y planificación

Este capítulo mostrará cómo se va a enfocar el proyecto. Se establecerán los cimientos analizando si éste puede ser viable o no. Se tratarán puntos como el modelo de desarrollo, la elección de las API's y material, los posibles riesgos dentro de éste, la planificación temporal y finalmente su viabilidad.

2.1 Introducción

Como ya se ha dicho anteriormente, la finalidad del proyecto es la creación de un software de visualización de simulaciones 3D.

Esto conlleva a definir correctamente las herramientas a utilizar combinándolas entre ellas y a mantener una correlación adecuada entre calidad, coste y tiempo. Estos tres factores serán determinantes para definir hacia dónde se quiere enfocar el proyecto barajando distintas opciones y aplicándolas en todo el periodo de desarrollo del software.

Una vez se han seleccionado las distintas opciones y finalizado el estudio, se debe asegurar que el proyecto podrá finalizarse ajustándose, con un margen de error pequeño, a los factores anteriormente mencionados.

2.2 Mercado al que va dirigida

Como se ha mencionado anteriormente el entorno de la simulación está latente y se prevé una gran utilización de estas técnicas por parte de la ciencia.

El abaratamiento que puede suponer a sus usuarios puede ser muy grande ya que la representación del sistema es lógica (informáticamente) y no es físicamente real.

El software servirá solo para visualizar los datos generados, es decir, no contendrá el simulador. Los simuladores serán aplicaciones externas a la nuestra y solo se dedicarán a generar los datos que el visualizador utilizará. [Figura 1]

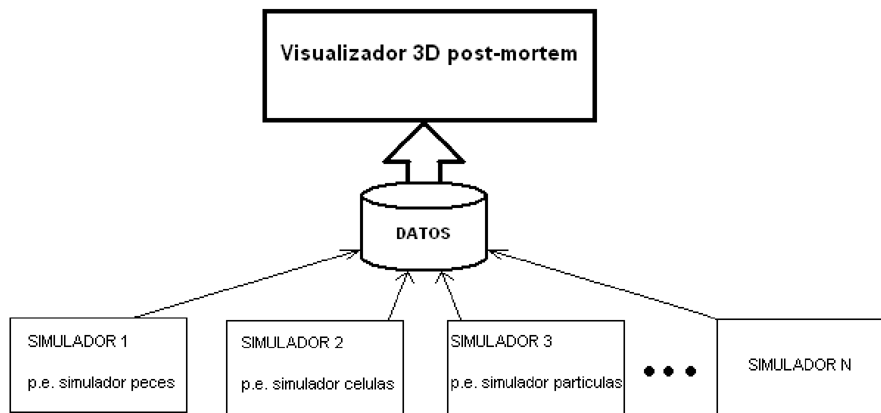


Figura 1

Esto tiene sus ventajas y desventajas:

Ventajas	Desventajas
Uso de distintos simuladores sobre un único software de visualización.	Los datos de entrada al visualizador deben de tener el mismo formato.
Abaratamiento del coste.	
Familiarización con un único software.	

Figura 2

Esta independencia entre simulador y visualizador puede facilitar la distribución en el mercado y da la posibilidad de mejorar el visualizador sin que sea necesario cambiar el simulador, facilitando la gestión del cambio a cualquiera de nuestros usuarios.

El software irá orientado a toda la rama de la ciencia que se vea en la necesidad de tener visualizar una representación de posiciones de agentes/individuos en el mundo tridimensional, bidimensional o unidimensional respecto del tiempo.
Esto indica que sus usuarios no tienen porque tener altos conocimientos de informática (biólogos, químicos, matemáticos,…) con lo que la aplicación debe de ser familiar y fácil de usar a la hora de su personalización.

2.3 Modelo de desarrollo del software

En el proceso de desarrollo del software se debe de mantener una filosofía o paradigma a la hora de evolucionar en el proyecto, gestionando por partes y marcando metas para mantener un control sobre este.
Tenemos varias opciones o filosofías ya estudiadas y testeadas en toda la historia del desarrollo del software, incluidas en la ciencia de la ingeniería del software:

- Modelo en cascada (modelo tradicional) – Marcada por las etapas del ciclo de vida del software, donde cada etapa debe de esperar rigurosamente la finalización de la anterior.
- Modelo en espiral (modelo evolutivo) – Las actividades no están fijadas a priori sino que se fijan en función del análisis de riesgos que se hace en cada iteración.
- Modelo de prototipos (modelo evolutivo) – Iniciamos las iteraciones con los objetivos globales y a cada iteración vamos profundizando más hacia la concreción.

El software vendrá marcado por unos requisitos especificados inicialmente, seleccionando exactamente lo que se quiere al inicio de la primera iteración de desarrollo del software. Claro está, estos requerimientos pueden ir variando, pero son claros y detallados.
Con el resto de etapas de desarrollo del software ocurre exactamente lo mismo. Una vez se sabe cómo se quiere que funcione el sistema, se realizará un diseño del software en base a ello y así sucesivamente con todas las etapas. Este modelo requiere que una etapa haya finalizado totalmente antes de la finalización de la anterior, con lo que el modelo de desarrollo de software que encaja mejor y va a ser utilizado será el modelo de desarrollo en “Cascada”. Se puede ver su evolución en la Figura 3.

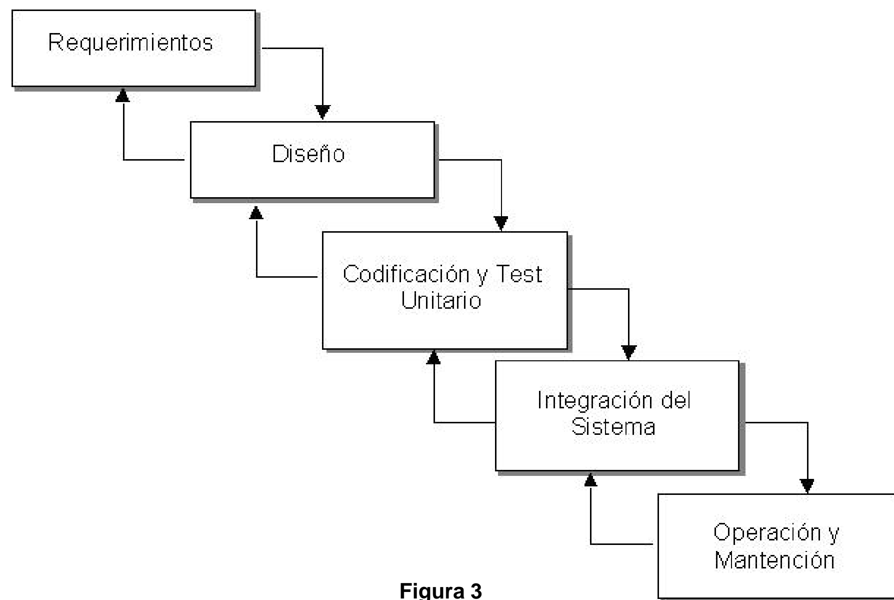


Figura 3

A continuación se comentan brevemente las etapas del proyecto dando una idea general de lo que se va a realizar en cada una de ellas:

- Requerimientos: Se especifican todos los requerimientos (funcionales y no funcionales) sobre el desarrollo del software para la correcta interpretación de qué se quiere y cómo se quiere en la aplicación.
- Diseño: Se realiza el diseño de nuestra aplicación: Estructuras, datos y movimiento de datos en nuestro sistema.
- Codificación y test unitario: Se traspasa el diseño a código y se realizan las pruebas básicas de funcionamiento de los distintos submódulos que componen la aplicación.
- Integración del sistema: Se integran los submódulos enlazando sus entradas y salidas para montar el sistema global.
- Operación y mantención: Se realizará el testing del sistema global para encontrar el mayor número de fallos posibles y así volver a iterar el ciclo desde cualquiera de las fases anteriores mejorando la calidad final del software.

2.4 Alternativas de desarrollo.

2.4.1 API gráfica

Como ya se ha visto anteriormente nuestro software se basará en un entorno de representación tridimensional. Esto obliga a escoger una API o motor gráfico 3D que se adapte a estas necesidades.

Las dos API's más extendidas e importantes del mercado son OpenGL y DirectX.

OpenGL tiene licencia GPL, es decir, software libre y por lo tanto totalmente usable por el proyecto. Esta API está disponible para los sistemas operativos Windows, Linux y MAC, con lo que permitiría ejecutar la aplicación en multiplataforma siempre que tengamos instalados los SDK (Software Development Kit) de esta API en la plataforma que se quiera utilizar. [27]

DirectX tiene licencia gratuita y también totalmente usable por el proyecto. El contra de esta API es que solo está disponible para Windows. [28]

La desventaja de estas dos API's es la complejidad de su uso a la hora de tratar los gráficos y de insertar realismo en la escena. Esto hace barajar la opción de utilizar un motor gráfico simplificando todos estos inconvenientes.

A continuación se presenta un listado de motores gráficos 3D que pueden ser de utilidad en nuestro proyecto:

- <u>PhyreEngine</u>: Es uno de los motores que han aparecido recientemente con novedosas técnicas gráficas, soportando todas las nuevas tecnologías aportadas por la consola PlayStation 3. [29]

  Tiene dos desventajas muy claras:

  - Directamente orientada a la programación de videojuegos.
  - Gran dificultad existente en la programación.

Figura 4

- Crystal Space: Este motor contiene un gran número de técnicas gráficas de gran potencia, pero de difícil uso como para que sea de utilidad en nuestro desarrollo. [30]

Figura 5

- Ogre3D: Es de fácil implementación, y tiene mucha información de ayuda que puede respaldar el trabajo que se quiere realizar. Tiene un sistema avanzado de materiales, animaciones, sistemas de partículas, meshes, escenas,… Tiene completa compatibilidad con oFusion (que en este caso no se usará porque es para 3D Studio Max), un plug-in que exporta directamente al formato de Ogre.

  El principal problema es exportar de blender (aplicación de modelado 3D) al formato Ogre ya que los plugins que hay hasta el momento no son muy buenos, sobre todo con la exportación de materiales. La cantidad de información desordenada es otro de los inconvenientes, ya que es difícil encontrar la información que realmente se está buscando. [3]

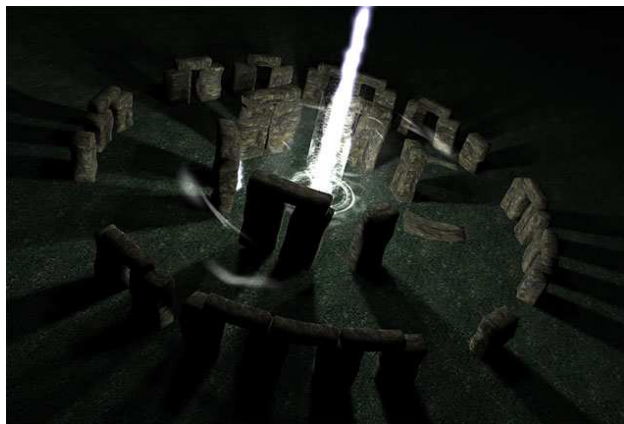


Figura 6

Finalmente se ha elegido Ogre por la cantidad de técnicas gráficas que se pueden utilizar. Su sistema de scripting de materiales es muy atractivo, ya que permite incluir shaders en él e incluso aplicarlos en las proyecciones (el renderizado o imagen que vemos en la pantalla) con el nombre de compositors. El sistema jerárquico de representación de los datos en la escena es de tipo grafo, siendo éste muy claro y orientativo. Las funciones típicas son muy parecidas a las de OpenGL o DirectX, aunque facilitando la lógica de control de éstas. También ofrece la posibilidad de escoger la API (OpenGL o DirectX) con la que se quiere renderizar, siendo exportable la aplicación a cualquier plataforma con pocos cambios e incluyendo la adaptación a los tres sistemas operativos globales: Windows, Linux y Mac/OS.

2.4.2 Lenguaje de programación

Escogiendo Ogre como motor gráfico existe otro abanico de opciones para elegir el lenguaje de programación a utilizar:

Nombre API	Lenguaje programación	Soporte oficial
Ogre SDK C++	C++	SI
Python-Ogre	Python	NO
MOGRE	C#	NO
Ogre4J	Java	NO

Figura 7

La opción escogida ha sido el lenguaje C++. Una de las razones es porque es un lenguaje de alto nivel, aunque más cercano al bajo nivel que el resto. Esto implica aumento de velocidad de ejecución que para el desarrollo de gráficos 3D resultará fundamental.
Otra de las razones de peso es que es la única que tiene soporte oficial de la comunidad Ogre, ya que el resto son proyectos que se han decidido realizar a parte. El soporte oficial está implicado directamente en el desarrollo de versiones nuevas periódicamente con nuevos complementos, corrección de bugs, mayor documentación e información, etc.

2.4.3 Otras api's – CEGUI

Uno de los objetivos a conseguir en el software es: "*Uso de una interfaz de usuario gráfica amigable y de fácil manejo*".

Esto lleva a pensar en utilizar en el software una GUI (Graphic User Interface). Una GUI es una interfaz gráfica en la que el usuario puede interactuar gráficamente con la aplicación y cambiar su estado. Ésta tiene que ser intuitiva y ofrecer un lenguaje visual al usuario simple y claro.

A continuación se muestra un ejemplo entre UI (User Interface, Figura 8) y GUI (Figura 9) para apreciar la diferencia de facilidad de interacción del usuario con la aplicación:

```
Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/P] [/W] [/A[[:]attribs]] [/O[[:]sortord]]
    [/S] [/B] [/L] [/C[H]]

  [drive:][path][filename]   Specifies drive, directory, and/or files to list.
  /P      Pauses after each screenful of information.
  /W      Uses wide list format.
  /A      Displays files with specified attributes.
  attribs   D  Directories   R  Read-only files         H  Hidden files
            S  System files  A  Files ready to archive  -  Prefix meaning "not"
  /O      List by files in sorted order.
  sortord   N  By name (alphabetic)       S  By size (smallest first)
            E  By extension (alphabetic)  D  By date & time (earliest first)
            G  Group directories first    -  Prefix to reverse order
            C  By compression ratio (smallest first)
  /S      Displays files in specified directory and all subdirectories.
  /B      Uses bare format (no heading information or summary).
  /L      Uses lowercase.
  /C[H]   Displays file compression ratio; /CH uses host allocation unit size.

Switches may be preset in the DIRCMD environment variable.  Override
preset switches by prefixing any switch with - (hyphen)--for example, /-W.

C:\>_
```

Figura 8

Figura 9

Para crear una GUI en la aplicación se ha valorado la opción de utilizar directamente las Overlays (Capas) de Ogre3D, que son zonas delimitadas con textura pegadas sobre el renderizado 3D. Esto complicaba bastante el desarrollo ya que se tenía que desarrollar un motor o manager para la GUI que gestionase todo esto de manera sencilla.

Finalmente se ha decidido utilizar una API llamada CEGUI debido a la posibilidad de integración sobre el motor Ogre3D y el buen aspecto gráfico que genera.

Ésta puede ser ejecutada sobre OpenGL o DirectX indistintamente y tiene licencia GPL con lo que se puede utilizar sin problemas.

Lo que permite esta API es añadir, por encima del renderizado de la escena, una capa que incrusta la GUI y los elementos que la componen.

Existen una serie de componentes ya establecidos que podemos usar y asignarles una función determinada en función de los eventos invocados sobre éstos. En la figura 10 se puede observar un ejemplo que utiliza CEGUI:

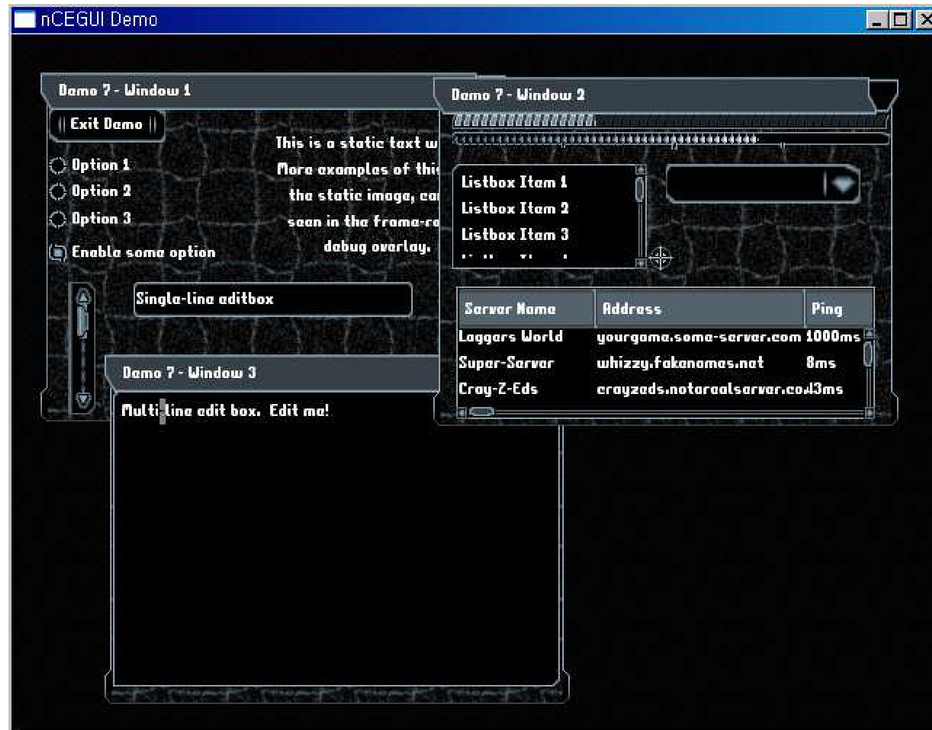


Figura 10

2.5 Recursos materiales y costes.

Distinguiremos básicamente entre dos tipos de recursos materiales en este proyecto. Los de software y los de hardware.

1. Recursos software :
    - Microsoft Visual Studio 2005 -
        - Con licencia de estudiante → Tasas de matrícula
        - Caso de compra → 1.199 $
    - Motor gráfico OGRE 3D
        - Licencia LGPL → Gratuita
    - API CEGUI
        - Licencia LGPL → Gratuita
    - Blender.
        - Licencia GPL → Gratuita

2. Recursos hardware :
    - Ordenador Intel Pentium IV Core 2 Duo a 2,0 GHz con 2 GB de RAM y una tarjeta gráfica aceleradora.
        - Usando los Pc's de la universidad → Tasas de matrícula
        - Comprando → 1200 $
    - El grid para generar los ficheros de datos
        - Usando el de la universidad → Tasas de matrícula
        - Comprando → Inviable

Tenemos dos vías:

1. **Estudiante universidad** → Solo haremos el pago de la matrícula teniendo derecho a un PC y licencia del MVS.

2. **Proyecto aparte** → 1199$ (de MVS 2005) + 1300 $ (del PC) = **2499 $**

En este caso, al tratarse del PFC, el director y yo tendremos un coste de tiempo, aunque infinitamente más económico que el pago por horas de un empleado en un proyecto aparte.

Por lo tanto, por estas razones, se ha elegido la vía de “Estudiante de universidad”.

2.6 Análisis de riesgos.

Hay varios tipos de riesgos los cuales el proyecto podrá encontrar durante todo su periodo de desarrollo:

1. El hecho de no conocer detalladamente todas las funciones del motor gráfico y los límites que estos tienen.

2. Otro problema puede venir dado consecuentemente de la utilización conjunta de las dos API’s, pudiendo existir cierta dificultad en la fusión y propagando errores entre ambas.

3. El riesgo más destacado es típico en todo trabajo de desarrollo de software: estancarse en algún punto puede incrementar altamente el tiempo de desarrollo, retrasando la planificación temporal.

4. Motores gráficos muy grandes y de mucha complejidad. Aprender las estructuras y funciones más típicas es relativamente fácil. El problema viene cuando se requiere de la utilización de técnicas más avanzadas, como por ejemplo, la utilización de shaders o programas para GPU.

5. El motor gráfico y la otra API son LGPL y están desarrollados por voluntarios. Esto puede implicar que existan bugs no detectados dentro del sistema, aunque éstos ya lleven bastantes años corriendo incluso en aplicaciones profesionales.

6 Según la metodología de desarrollo utilizada, el fallo en una etapa propaga los errores a las siguientes. Por lo tanto, hay que ir con cuidado antes de la finalización de cada etapa y revisándolas con detenimiento.

7 Fallo de hardware de almacenamiento y perdida de la información.

A continuación se presenta una tabla de impacto (Figura 11) de los riesgos anteriormente enumerados con las posibles soluciones proactivas para evitarlos:

Riesgo	Prob.	Impacto	P*I	Coste	Solución
Desconocimiento de los limites de las API's	0.2	Muy Alto(10)	2.0	Puede provocar la pérdida del proyecto	Documentarse bien de lo que es capaz de realizar la API en el proceso de alternativas de desarrollo
Problemas en la integración entre las API's	0.2	Alto(7.5)	1.5	Puede provocar retrasos importantes cambiando a otras API's	Realizar pruebas básicas de integración e informarse si hay compatibilidades.
Estancamiento en un punto de desarrollo	0.8	Bajo(2.5)	2	Pequeños retrasos ya previstos en la planificación	Buscar la solución lo antes posible o cambiar la planificación retrasando la tarea para avanzar en las demás.
Pérdida de tiempo probando técnicas avanzadas.	0.5	Medio(5)	2.5	Puede retrasar bastante el proyecto	Buscar documentación y ejemplos que permitan entender bien dichas técnicas. No excederse en el tiempo ya que son complementos.
Bugs en las API's	0.1	Muy Alto(10)	1	Puede provocar gran pérdida de tiempo buscando soluciones alternativas incluso la pérdida del proyecto.	Descargarse las versiones más estables de las API's ya testeadas y probadas durante tiempo.
Fallo en alguna etapa del ciclo del software	0.5	Alto(7.5)	3.75	Un fallo en las primeras etapas puede retrasar mucho el desarrollo del proyecto.	Centrarse mucho en las dos primeras etapas: "Requerimientos" y "Diseño" principalmente.
Fallo hardware de dispositivo de almacenamiento	0.3	Muy Alto(10)	3	Pérdida absoluta del proyecto	Duplicar información del proyecto periódicamente en dispositivos separados físicamente.

Figura 11

Suma de P*I	Nº de riesgos	**IMPACTO MEDIO DEL PROYECTO**
15.75	7	**2.25**

Figura 12

Podemos ver que se ha definido una escala de impacto con cuatro niveles:

Muy Alto = 10

Alto = 7.5

Medio = 5

Bajo = 2.5

Realizando los cálculos pertinentes, se ha llegado a la conclusión de que tendremos como media un impacto de 2.25 puntos (sobre 10 puntos) a lo largo del proyecto (ver Figura 12), con lo que el impacto es inferior al nivel “Bajo” que hemos establecido con 2.5.

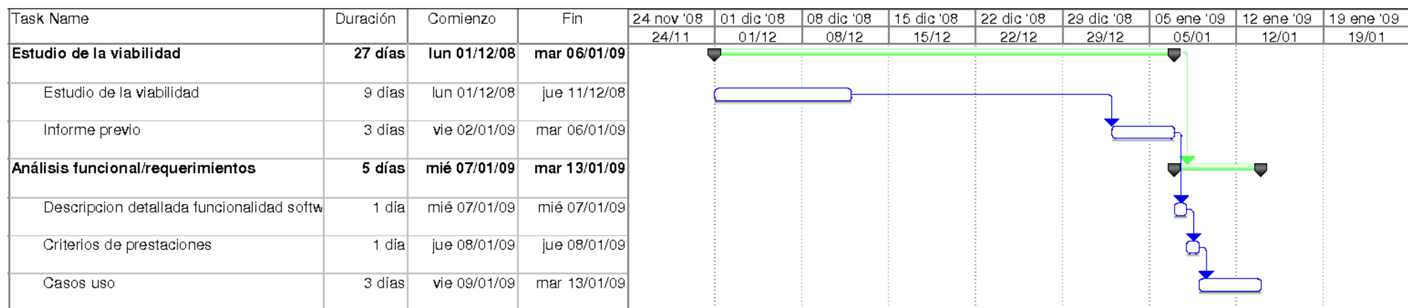
El impacto sobre el proyecto será únicamente temporal con lo que se estipulará que el retraso probabilístico que sufrirá el proyecto será de un 2.25 sobre 10, o lo que es lo mismo, un 22.5% del tiempo total estipulado del proyecto sin riesgo.

Este dato se tendrá en cuenta a la hora de elaborar la planificación temporal dando los márgenes adecuados a las tareas que se enmarquen dentro de los riesgos anteriormente mencionados.

2.7 *Planificación temporal*

La planificación temporal finalmente se ha dividido en dos periodos de trabajo debido al abandono temporal del proyecto por cuestiones externas. Se muestra para cada uno de ellos las tareas realizadas y su tiempo empleado:

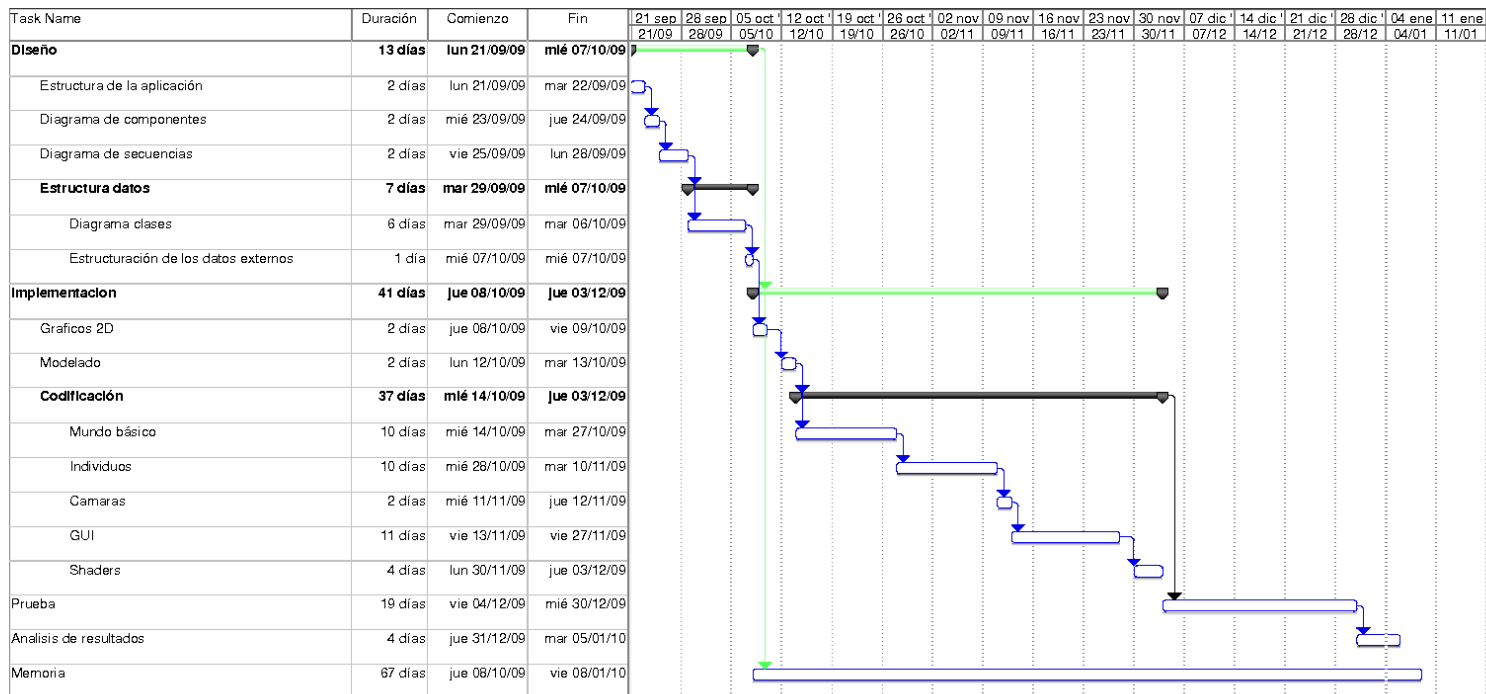
<u>Periodo 1 (2008-2009):</u>



Task Name	Duración	Comienzo	Fin
Estudio de la viabilidad	**27 días**	**lun 01/12/08**	**mar 06/01/09**
Estudio de la viabilidad	9 días	lun 01/12/08	jue 11/12/08
Informe previo	3 días	vie 02/01/09	mar 06/01/09
Análisis funcional/requerimientos	**5 días**	**mié 07/01/09**	**mar 13/01/09**
Descripcion detallada funcionalidad softw	1 día	mié 07/01/09	mié 07/01/09
Criterios de prestaciones	1 día	jue 08/01/09	jue 08/01/09
Casos uso	3 días	vie 09/01/09	mar 13/01/09

Figura 13

<u>Periodo 2 (2009-2010):</u>



Task Name	Duración	Comienzo	Fin
Diseño	**13 días**	**lun 21/09/09**	**mié 07/10/09**
Estructura de la aplicación	2 días	lun 21/09/09	mar 22/09/09
Diagrama de componentes	2 días	mié 23/09/09	jue 24/09/09
Diagrama de secuencias	2 días	vie 25/09/09	lun 28/09/09
Estructura datos	**7 días**	**mar 29/09/09**	**mié 07/10/09**
Diagrama clases	6 días	mar 29/09/09	mar 06/10/09
Estructuración de los datos externos	1 día	mié 07/10/09	mié 07/10/09
Implementacion	**41 días**	**jue 08/10/09**	**jue 03/12/09**
Graficos 2D	2 días	jue 08/10/09	vie 09/10/09
Modelado	2 días	lun 12/10/09	mar 13/10/09
Codificación	**37 días**	**mié 14/10/09**	**jue 03/12/09**
Mundo básico	10 días	mié 14/10/09	mar 27/10/09
Individuos	10 días	mié 28/10/09	mar 10/11/09
Camaras	2 días	mié 11/11/09	jue 12/11/09
GUI	11 días	vie 13/11/09	vie 27/11/09
Shaders	4 días	lun 30/11/09	jue 03/12/09
Prueba	19 días	vie 04/12/09	mié 30/12/09
Analisis de resultados	4 días	jue 31/12/09	mar 05/01/10
Memoria	67 días	jue 08/10/09	vie 08/01/10

Figura 14

2.8 Viabilidad del proyecto

En este apartado se contempla si el proyecto será viable o no en función de los análisis y predicciones que se han realizado en los puntos anteriores, manteniendo la relación, que previamente habíamos mencionado, entre calidad, coste y tiempo.

- **Conocimientos** → La documentación requerida de las API's está disponible, por lo que se podrá investigar y aprender sin problemas. La experiencia previa en el lenguaje de programación C++ hace que por este punto el proyecto sea totalmente viable.

- **Tiempo** → El tiempo es uno de los puntos clave a la hora de que el proyecto sea viable de realizar o no. En función de la planificación temporal realizada y el margen de tiempo introducido en la etapa de desarrollo basándonos en los análisis de riesgos, se puede asegurar que el proyecto será viable temporalmente hablando.

- **Coste material** → El estudio en los puntos de alternativas de desarrollo y análisis de recursos y costes han ayudado a definir el coste total del proyecto, siendo éste el pago de la matrícula anual de estudios. De esta forma el proyecto por este punto es también totalmente viable.

De esta forma se puede asegurar que el proyecto será viable. Contemplando que se podrá finalizar en los tiempos marcados por la planificación y con los costes estipulados anteriormente en el análisis de costes y recursos.

3. Requerimientos y análisis funcional

Este apartado mostrará todos los requerimientos funcionales y no funcionales de la aplicación. Es un punto muy importante para el tipo de modelo de desarrollo que se ha escogido (Cascada) ya que serán los cimientos de nuestra aplicación y hay que definir muy bien lo que se requiere.
Se puede ver la importancia de esta sección documentando en qué punto los proyectos software fallan: [30]

- Requerimientos Incompletos 13.1%
- Falta de Requerimientos 12.4%
- Falta de Recursos 10.6%
- Expectativas no Realistas 9.9%
- Cambio Requerimientos/Especificaciones 8.7%
- Falta de Planeamiento 8.1%
- No se especifico el tiempo adecuado 7.5%

3.1 Requerimientos

Los requerimientos para el software serán:

- Panorama general
  - Este proyecto tiene por objetivo crear un sistema 3D de visualización de simulaciones post-mortem.
- Metas
  - En términos generales la meta es poder visualizar la simulación previamente hecha por el simulador de manera que se pueda interactuar con el mundo 3D y personificarlo a nuestros requerimientos.

- Visualización realista de la representación de la simulación.
- Interacción con la visualización de la simulación y con el mundo tridimensional.
- Personalización del entorno y agentes.

- Requerimientos funcionales
  - Evidentes:
    - Carga de distintos datos generados por las simulaciones.
    - Reproducción de los datos generados por la simulación que se desee.
    - Distintas cámaras que permitan ver la escena desde puntos de vista diferentes.
    - Vistas en primera y tercera persona para realizar el seguimiento sobre un agente con la previa selección de éste.
    - Posibilidad de marcar a un listado de individuos arbitrário para su seguimiento durante la reproducción de la simulación.
    - Posibilidad de mover a todo el grupo de agentes a la posición inicial que interese dentro del espacio tridimensional. La visualización se reproducirá relativa a esta posición.
    - Inclusión de un menú multimedia para reproducir, pausar y parar la simulación.
    - Inclusión de una barra de desplazamiento que indique el porcentaje de reproducción, con la posibilidad de avanzar o retroceder visualización de la simulación a conveniencia del usuario.
    - Opción de escoger la velocidad de reproducción de la simulación.
    - Opción de escoger el número de individuos que queremos que aparezcan en la simulación.
    - Opción de elegir la malla del agente de la simulación
    - Opción de elegir la animación del agente de la simulación.

        - Opción de elegir el fondo o cielo que queremos en la simulación.
        - Opción de filtrar los colores (Rojo, Verde, Azul) gradualmente del modelo del agente.
        - Opción de escalar la malla del agente.
        - Opción para mostrar el mar o no mostrarlo.
    - Ocultas:
        - Se guardan y se cargan los parámetros de configuración en un fichero para que se guarde el estado de la máquina de estados una vez hemos salido de la aplicación.
        - Carga una lista de carpetas en una carpeta preestablecida que contendrán las simulaciones.
        - Carga una lista de modelos/animaciones que estarán en una carpeta preestablecida.
        - Carga una lista de fondos que estarán en una carpeta preestablecida.

- Requerimientos no funcionales.
    - Sistema de visualización realista.
    - Sistema intuitivo y familiar basado en mouse y no en teclado (puede tener accesos directos por teclado pero no son imprescindibles).
    - Tiempo de respuesta. El tiempo en generar un frame tiene que ser como máximo de 0.04 segundos, es decir, 25 frames por segundo. Esto irá en función de la máquina en la que ejecutemos y en el nivel de detalle de las mallas elegidas, incluso de sus materiales.
    - La plataforma o sistema operativo será Microsoft Windows XP o superior (con los service packs 1 y 2 instalados en caso de que sea el XP) con Direct3D instalado.

3.2 Análisis funcional

Para realizar el análisis funcional se utilizarán los diagramas de casos de uso de UML, siendo éste un diagrama de comportamiento del sistema con la correspondiente interacción funcional que tendrá el usuario a la hora de manejar la aplicación.

Se organizará la funcionalidad de la aplicación en casos más globalizados, es decir, dividiéndola en cuatro bloques:

- Menú Principal.
- GUI de la Simulación.
- Configuración Visualización.
- Configuración Gráficos.

3.2.1 Menú principal:

El caso de uso de la Figura 15, mostrará la posible interacción funcional del usuario en el momento en que entra en la aplicación. Dicho de otra manera, será el punto de partida o raíz para acceder al resto de funcionalidades de la aplicación.
Éste será el caso de uso base y el resto, serán llamados por el mismo.

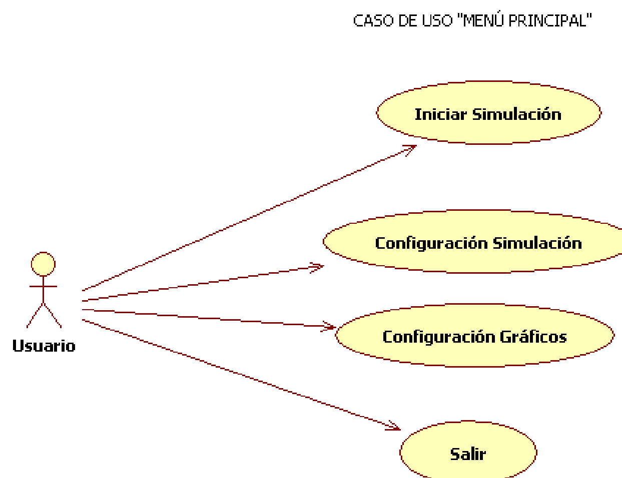


Figura 15

- Iniciar Simulación → Este será el punto de partida de inicio de la representación de la simulación. Primeramente se activa la inicialización de la escena. Aquí se cargarán los ficheros de salida de las simulaciones que se hayan establecido. Se cargarán todas las posiciones de los agentes y sus direcciones en cada posición de tiempo y se generará la interpolación de éstas para crear la animación general. Hecho esto, la simulación está preparada para ser mostrada.

- Configuración Simulación → Se accederá a las propiedades de configuración de la visualización de la simulación especificadas en el caso de uso que se estudia a posteriori.

- Configuración Gráfica → Se accederá a las propiedades de configuración gráfica especificadas en el caso de uso que se estudia a posteriori.

- Salir → Romperá el bucle de renderizado y finalizará la aplicación.

3.2.2 Gui de la simulación:

El caso de uso de la Figura 16, mostrará la funcionalidad una vez tenemos los datos resultantes de la simulación cargados. Será el que interactuará con el mundo tridimensional y el que permitirá cambiar el estado de la visualización en tiempo real.

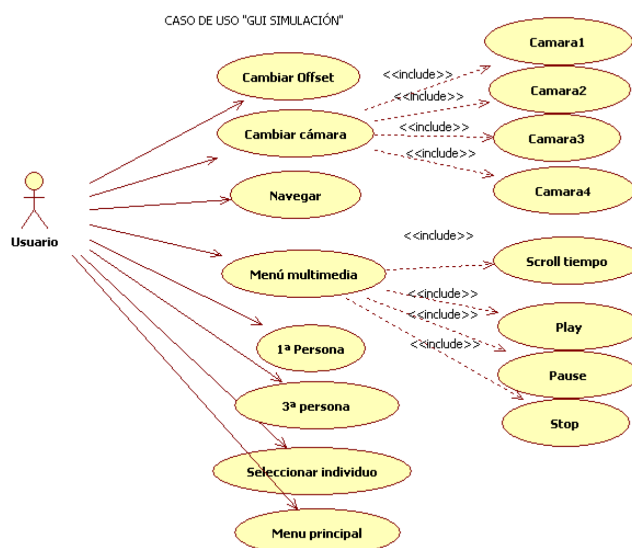


Figura 16

- Cambiar Offset → Funcionalidad que dará la opción de cambiar la posición inicial de todos los agentes. La simulación generará unos datos absolutos respecto una ubicación inicial, con lo que puede interesar cambiar esa posición y reproducir la animación relativa a ésta, colocando los individuos en la parte del mundo que interese. Esto se hará con desplazamientos (X,Y,Z), observando en todo momento dónde se están ubicando los individuos.

- Cambiar Cámara → Funcionalidad que dará la opción de cambiar la cámara que se desea mostrar. Se tendrán un total de cuatro cámaras totalmente independientes. Esto quiere decir que cada una de ellas tendrá sus propiedades y se podrá ubicar u orientar cada una en el espacio tridimensional, mostrando la cámara que más convenga en cada momento de la representación.

- Seleccionar individuo → Funcionalidad que servirá para seleccionar al individuo al que se le quiere realizar un seguimiento. Esto servirá para las tres siguientes funcionalidades:

  - Vista primera persona → Una vez seleccionado el individuo al cual se quiere seguir, podrá sumergirse en la visión de dicho individuo, viendo el mundo como si se observase a través de sus ojos.
  - Vista tercera persona → Una vez seleccionado el individuo al cual se quiere seguir, se apuntará la cámara actual hacia dicho individuo, realizando su seguimiento y pudiendo cambiar la posición de la cámara libremente para mejorar la observación.
  - Selección múltiple → Se podrá seleccionar múltiples individuos añadiendo sus identificadores en una lista para realizar su seguimiento.

- Cambio velocidad → Funcionalidad que permitirá incrementar o decrementar la velocidad de reproducción en una unidad.

- Navegar → Existen dos modos generales de funcionamiento dentro de la representación de la escena:

    - Modo Navegación → El movimiento por el mundo tridimensional será con las teclas y el ratón como si de un juego FPS (First Person Shooter) se tratase, siendo una manera eficiente y fácil de cambiar la posición de la cámara actual por el mundo.

    - Modo Menú → Aparece un cursor evitando el movimiento por el mundo tridimensional, de forma que el enfoque pasa al menú de la GUI con lo que se podrán cambiar sus opciones.

- Menú Multimedia → Este menú permitirá manejar la variable tiempo de la animación, es decir, manipular la evolución de ésta a nuestra conveniencia. Se contemplan varias funcionalidades:

    - Scroll Tiempo → Existe un scroll que mostrará el porcentaje que llevamos de animación reproducida. Este scroll será modificable y se podrá avanzar y retroceder el porcentaje de la reproducción de la simulación a nuestro antojo como si de un reproductor de video multimedia se tratase.

    - Play → Funcionalidad que permitirá poner la escena en modo reproducción.

    - Pause → Funcionalidad que pausará la evolución de los agentes en algún punto del tiempo de la reproducción.

    - Stop → Funcionalidad que parará la reproducción llevándola al punto de tiempo=0 y pausándola.

- Menú Principal → Esta función volverá a la interfaz del menú principal para poder cambiar las propiedades de la representación de la escena. Esto obligará a que se tenga que cargar de nuevo.

3.2.3 Configuración simulación:

El caso de uso de la Figura 17, mostrará la funcionalidad para personalizar la configuración de la reproducción de la simulación accesible desde el menú principal.

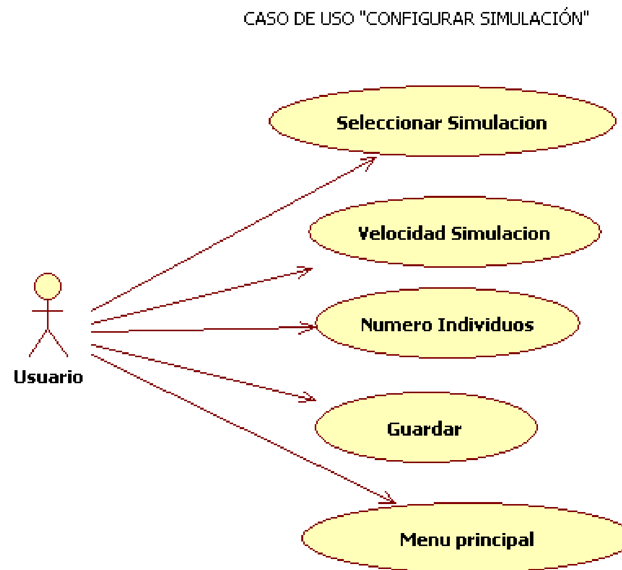


Figura 17

- Seleccionar Simulación → Funcionalidad que permitirá seleccionar el nombre de la carpeta donde tenemos los datos generados de la simulación. El nombre de la carpeta que éste seleccionada será la que se reproducirá al iniciar la reproducción.

- Velocidad Simulación → Funcionalidad que permitirá seleccionar la velocidad de reproducción. Ésta irá de un rango de x0 a un rango de x10, es decir multiplicando el valor del tiempo de la representación por el valor escogido. Por ejemplo, en una simulación de 20 segundos y una velocidad x10, tardaremos 2 segundos en visualizarla completamente

- Número Individuos → Funcionalidad que permitirá escoger un porcentaje de número de individuos respecto al número total de individuos que contienen los datos de la simulación. Esto servirá para reducir el número de individuos en el caso de que nos interese o que la máquina en la que

ejecutamos el programa no sea demasiado potente con la correspondiente mejora de FPS o Frames por Segundo.

- Guardar → Funcionalidad que guardará en disco los valores elegidos para que la próxima vez que se ejecute la aplicación éstos se mantengan.

- Menú Principal → Esta función volverá a la interfaz del menú principal para poder cambiar las propiedades. Esto hará que se tenga que cargar otra vez la escena.

3.2.4 Configuración gráficos:

El caso de uso de la Figura 18 mostrará la funcionalidad del menú de configuración de gráficos, accesible desde el menú principal. Permitirá cambiar las opciones gráficas como agentes, animaciones, fondos, etc.

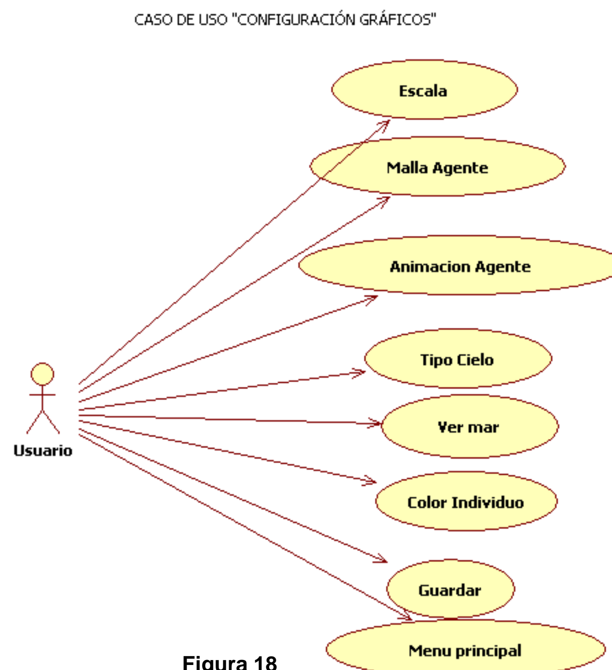


Figura 18

- Escala → Funcionalidad que permitirá reescalar la malla del agente. Esto servirá para adaptarla a nuestra escena ya que el modelo se ha podido modelar a una escala distinta a la que se usa en la aplicación.

- Malla Agente → Funcionalidad que permitirá cambiar el modelo o malla del agente a representar en la escena. Esto ayudará a personalizar la escena pudiendo elegir modelos como: peces, pájaros, moléculas,…

- Animación Agente → Funcionalidad que irá ligada a la selección de la malla del agente. Cada malla puede tener asignadas distintas animaciones o ninguna. Por ejemplo un mismo modelo de una persona puede tener distintas animaciones como caminando, parado, bostezando, etc…

- Fondo → Funcionalidad que permitirá cambiar el fondo o cielo, por ejemplo un cielo nublado, un cielo soleado, un universo,…

- Ver mar → Funcionalidad que permite quitar el mar de la escena. Esto es por si los agentes de la simulación no son individuos acuáticos o simplemente se desea representarla sin él.

- Color Individuos → Funcionalidad que permite filtrar los colores de la textura del individuo quitando los colores primarios (Rojo, Verde, Azul) para así personalizar el color del individuo.

- Guardar → Funcionalidad que guardará en disco los valores elegidos para que la próxima vez que se ejecute la aplicación éstos se mantengan.

- Menú Principal → Esta función volverá a la interfaz del menú principal para poder cambiar las propiedades. Esto hará que se tenga que cargar otra vez de la escena.

Estos serán los cuatro diagramas de casos de uso principales del sistema. En la Figura 19 se presenta el diagrama general unido para ver la accesibilidad funcional y los caminos para acceder a cada función del programa por el usuario.

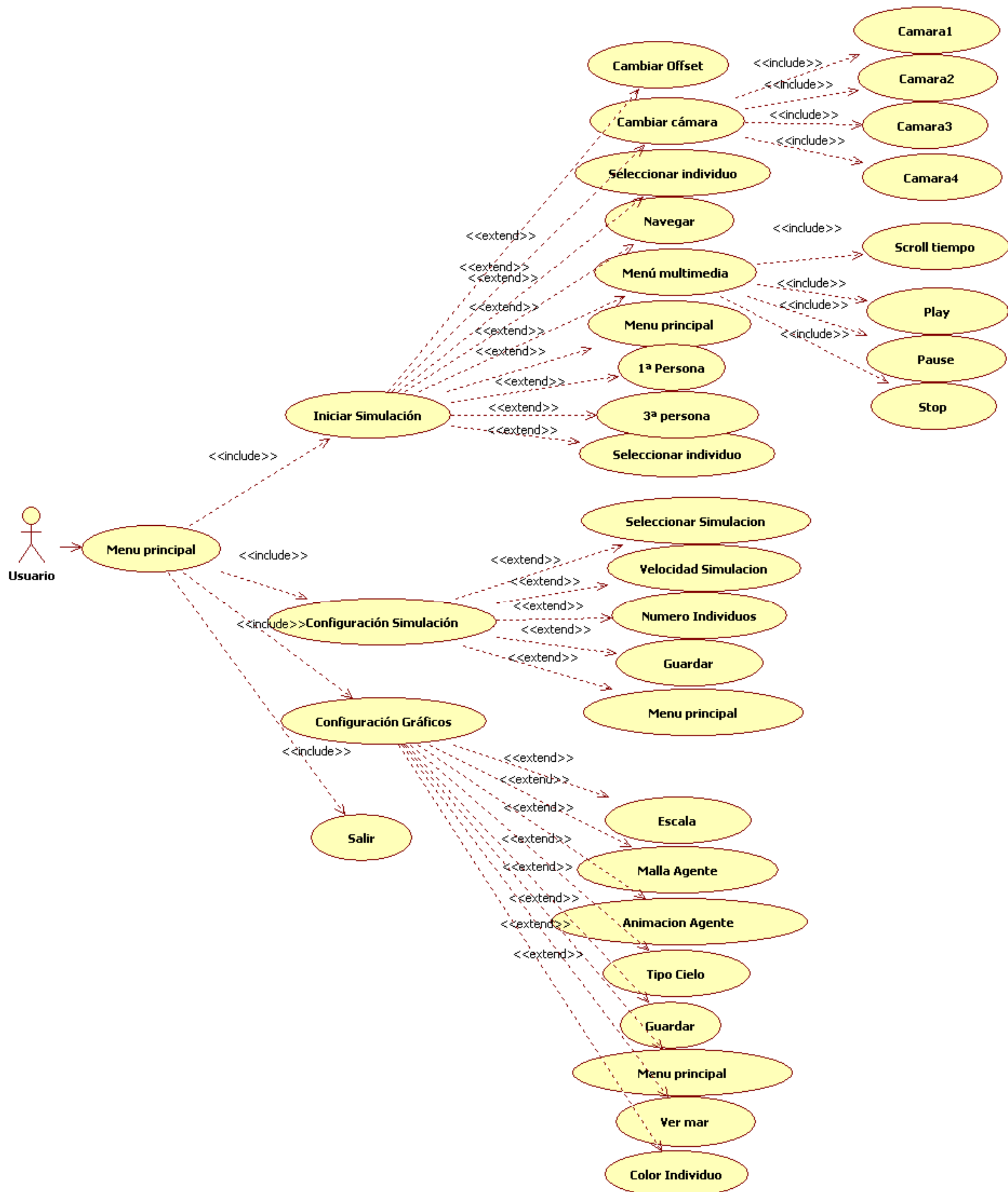


Figura 19

4. Diseño

En este capítulo se mostrará toda la parte de diseño de la aplicación propiamente dicha. Esto incorpora la idea del diseño general, la estructura de las clases y la comunicación entre ellas.

4.1 Estructura de la aplicación

Ante todo, lo que se ha estudiado en este software, es realizar un diseño coherente con una estructura bien definida para poder realizar ampliaciones y mantenimiento sin grandes complicaciones en un futuro. La orientación a objetos que proporciona C++ ha sido de gran utilidad.

La idea general del diseño ha sido dividirlo en módulos independientes, relacionándose mediante un solo objeto raíz que los incorpore y pudiendo así usarlos en cualquier punto del hilo de ejecución de la aplicación.

4.1.1 Managers – <EscenaMgr> y <MenuMgr>

Estos módulos independientes se han llamado “Managers”. Este nombre se refiere a que cada uno de ellos trata un tema específico dentro del programa. La comunicación entre estos Managers no será directa sino por máquina de estados y funciones del propio motor Ogre, ya que en teoría son independientes el uno del otro.

Estos Managers dependen de unos objetos base del propio motor Ogre, por tanto, como todos lo necesitan, se ha creado una clase base que los contiene heredando así sus propiedades. Esta clase base será un patrón de diseño llamado “Singleton” utilizado mucho en la ingeniería del software. El patrón permitirá tener una única instancia de la clase base aunque se tengan varios Managers. El primer Manager que haga referencia a alguna propiedad de la clase base, ésta será instanciada. La próxima vez que se haga referencia, por el mismo o por cualquier Manager, se devolverá el objeto que se había creado anteriormente, usando así la misma instancia en cualquiera de los Managers.

4.1.2 Agentes - <Individuo>

Se necesita también una clase que agrupe al individuo o agente y se puedan realizar modificaciones sobre su estado.

Cada agente estará formado por la malla (Mesh) y por un nodo en la escena (SceneNode). El Mesh contendrá toda la información sobre el modelo 3D como vértices, animaciones, materiales,… En cambio, el SceneNode, estará una capa por encima e independientemente del modelo será un punto en el mundo 3D que definirá la posición, orientación, escalado,… Ambos estarán relacionados.

Se crearán tantos objetos de esta clase como agentes existan en el fichero datos de la simulación.

4.1.3 Raiz - <rootsingleton>

Como se ha dicho anteriormente, se tendrá un solo objeto raíz que incorporará estos Managers. Este objeto raíz (Root) también será “Singleton”, ya que interesa tener solo un objeto raíz con los Managers instanciados una sola vez, es decir, que siempre sean los mismos en cualquier punto del programa.

4.1.4 Máquina Estados - <máquinaestados>

Antes se ha nombrado la Máquina de estados como conductor de información entre Managers. Esta clase contendrá todas las variables de estado y de configuración de la aplicación. También será del tipo “Singleton” ya que solo interesará tener una instancia de la Máquina de estados en cualquier punto de la aplicación para que sus propiedades sean las mismas.

El objeto será accesible desde la aplicación general e incluso en los Managers.

Algunas de las propiedades de configuración de la aplicación, guardadas en ella, serán las que se guardarán en un fichero de configuración para no perder la configuración una vez salimos del programa.

4.1.5 Aplicación principal – <simulacion3dapp>

Existirá una clase que servirá para montar el motor inicial de Ogre. En ésta se inicializarán todos los objetos y estados del motor. Se dividirá esta clase con funciones básicas para agruparlas todas en una única de creación de escena. En la parte de implementación se detallará más a fondo cómo se ha hecho.

4.1.6 Bucle de renderizado - <MiFrameListener>

Se necesita una clase que lleve el control de qué y cuándo se hace algo antes o después de renderizar un frame. Dicho de otra manera, es el lazo o bucle infinito de la aplicación.

Esta clase hereda de la clase <FrameListener> del motor de Ogre y es obligatoria para que la aplicación pueda renderizar lo que se necesite en cada momento.

También se encargará de gestionar la entrada y salida para capturar los movimientos del ratón y del teclado.

4.2 Diagrama de componentes

En el diagrama de la Figura 20 se puede ver, en estructura modular, el diseño de la aplicación a grandes rasgos, viendo la accesibilidad que tiene cada módulo con el resto y la comunicación entre ellos.

Se puede observar que en la capa mas externa tendremos las API's, las cuales serán accesibles desde cualquier punto de la aplicación.

A continuación existe la aplicación base que inicializará todos los componentes de los motores para que estos se pongan en funcionamiento.

Dentro de esta aplicación base está integrado el punto más importante para que la aplicación renderice, el FrameListener, y será el que mantendrá el hilo de ejecución durante todo el ciclo de vida del proceso. Este tendrá absoluta comunicación con el resto de módulos.

Otro de los módulos inmersos en la aplicación base será el RootSingleton y contendrá todos los módulos Manager y los objetos base del motor gráfico Ogre. Éste también tendrá comunicación con el FrameListener, pero no entre Managers.

El último módulo en la aplicación base será el MáquinaEstados y tendrá absoluta comunicación con todos los módulos de la aplicación.

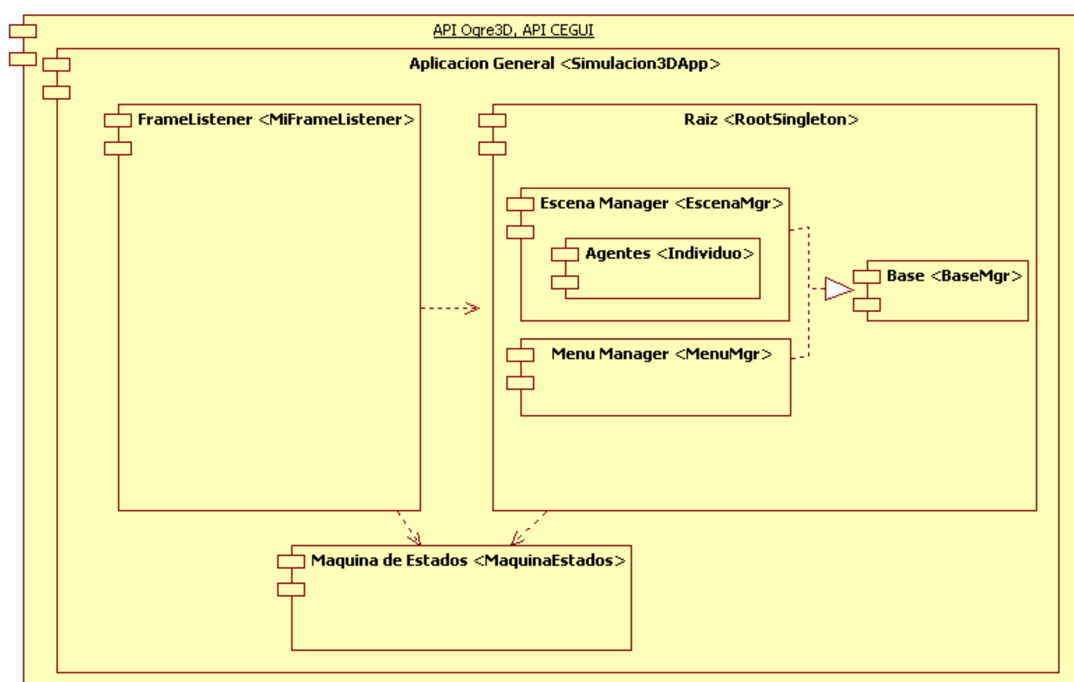


Figura 20

4.3 Diagrama de clases

En la Figura 20 se muestra el diagrama de clases final de la aplicación.

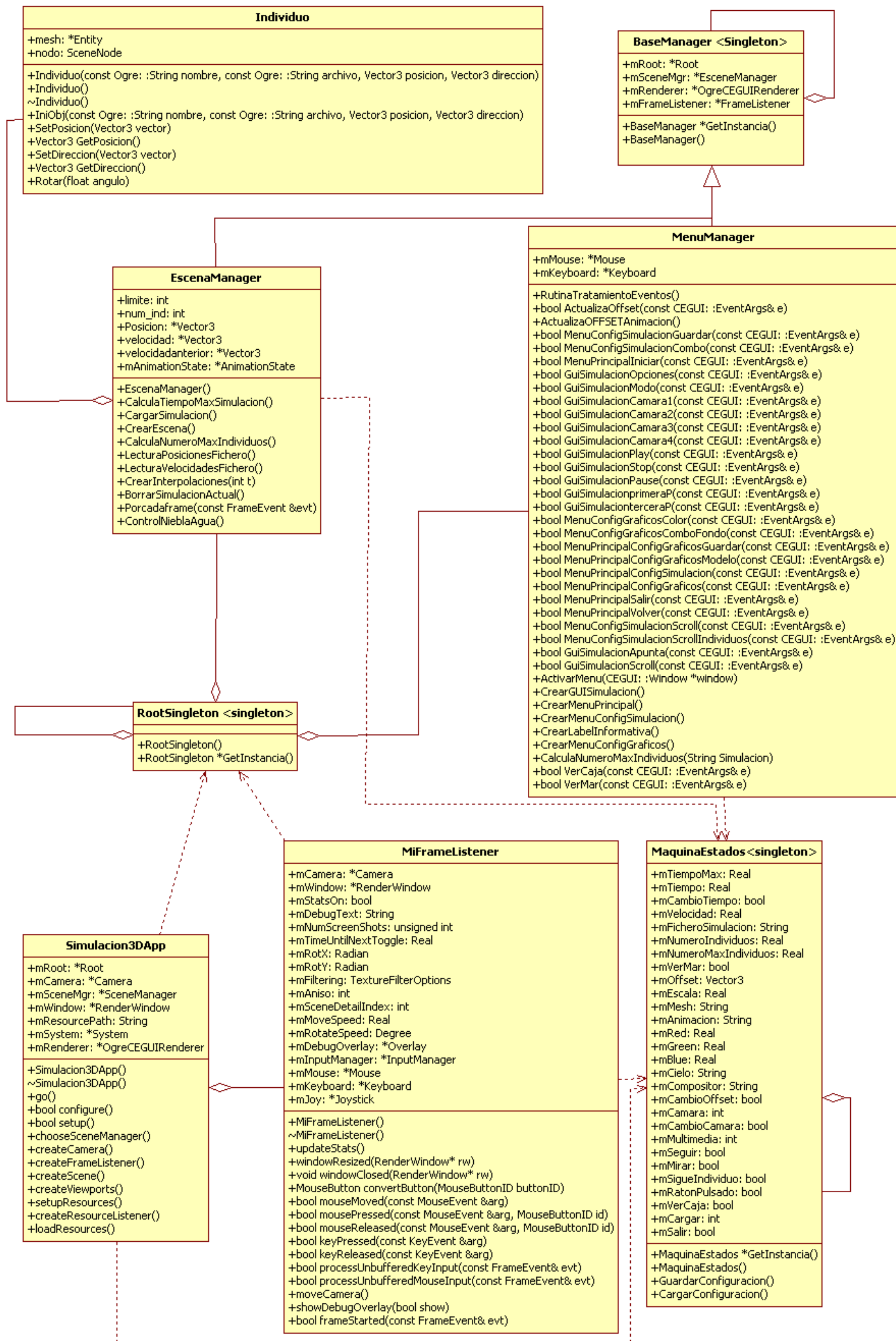


Figura 20

4.4 Diagrama de secuencias

4.4.1 Modificación de propiedades de configuración.

La modificación de las propiedades gráficas o de representación de la simulación siempre seguirá el mismo proceso (ver Figura 21).

Una vez el usuario ha cambiado la opción de configuración ésta quedará registrada en la máquina de estados y se actualizará la escena con las propiedades establecidas. Cuando el usuario pulse el botón guardar, se iniciará el proceso de escritura al fichero de configuración externo, quedando guardadas estas propiedades para la próxima vez que se inicie la aplicación.

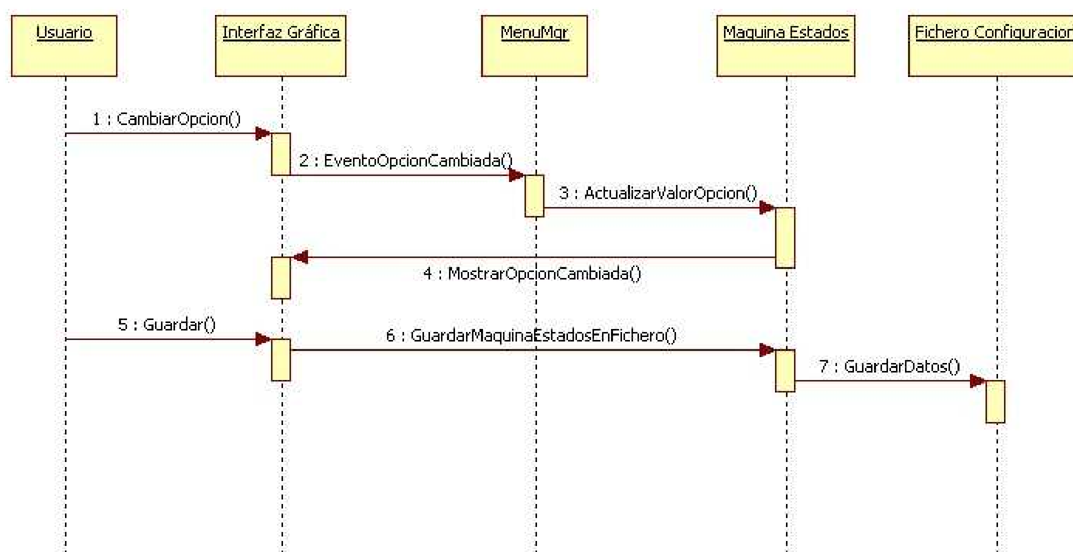


Figura 21

4.4.2 Carga de visualización de simulación

El proceso de carga de la escena (ver Figura 22) se hará en el momento en el que el usuario de la aplicación desee, es decir, no se arrancará directamente, de tal forma que éste pueda establecer las propiedades gráficas y de visualización de la simulación antes de la carga.

Una vez el usuario ha pulsado el botón de carga, internamente se activará un flag en la máquina de estados conforme la carga ha sido iniciada.

Este flag provocará la invocación de la carga de la visualización de la simulación.

El proceso será el borrado de la escena anterior (en el caso de que exista). Se calculará el número máximo de segundos de la representación, es decir el número de ficheros de salida que existen de simulación (el intervalo entre un fichero y otro se registrará como de 1 segundo).
Se realiza la lectura de las posiciones y direcciones iniciales para el segundo 0, asignándoselas a los agentes.
Posteriormente se crea la animación por keyframing. El keyframing es una técnica de animación en la que se interpolan dos KEYS (posición, orientación y escalado) en un intervalo de tiempo finito, realizando así su animación. Esto es lo que se hará para todas las posiciones y direcciones de los agentes. Una vez hecho esto se habilitará la animación general y las de la propia malla.
Finalmente se activa el flag de escena cargada en la máquina de estados.

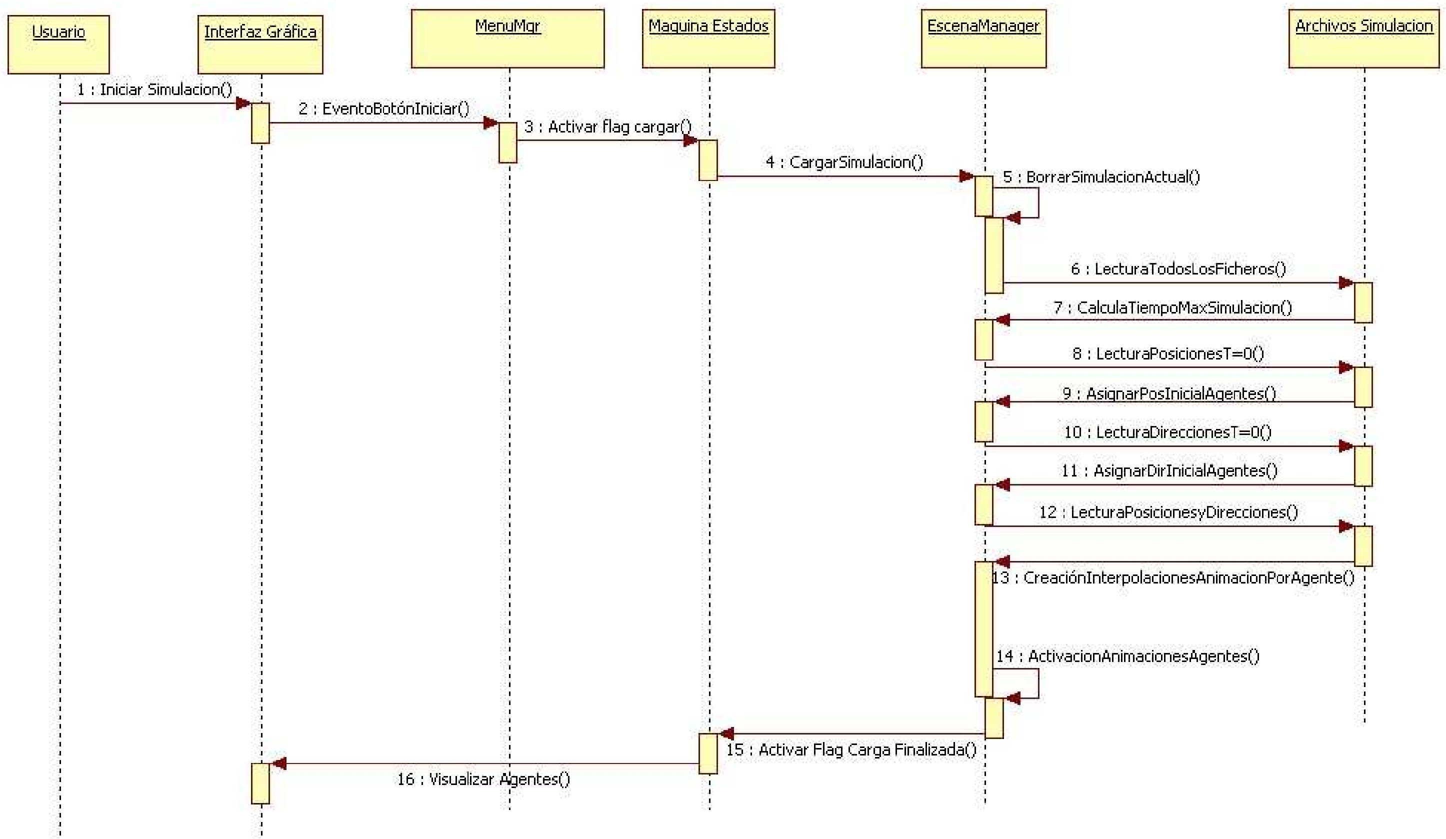


Figura 22

4.4.3 Interacción con la GUI

En este apartado se explica qué acciones se realizan internamente al pulsar algunas de las funciones de la GUI una vez se ha inicializado la visualización de la simulación (ver Figura 23).

Las funciones multimedia (Play, Pausa, Stop) activan un flag preestablecido en la máquina de estados que indican cual de las tres funciones se ha llamado. Posteriormente se calcula el tiempo que ha transcurrido entre frame y frame.
En el caso que el flag Play este activado, se añadirá este tiempo previamente calculado.
En el caso que el flag Pausa este activado, no se añadirá el tiempo a la animación y en consecuencia ésta quedará estática.
En el caso que este activado el flag Stop, se reseteará la animación general y tampoco se le añadirá el tiempo. En otras palabras se volverá al tiempo 0 de la animación sin que ésta avance.

La función de cambio de velocidad modificará en la máquina de estados dicha propiedad.

La función de seleccionar agente cambiará el modo a selección y se podrá seleccionar el individuo al cual queremos seguir. Esto servirá para las vistas en primera, tercera persona y selección múltiple de individuos.

La vista en primera persona hará que la cámara se ubique encima del individuo seleccionado previamente.

La vista en tercera persona hará que la cámara apunte siempre al individuo seleccionado previamente.

La selección múltiple de individuos marcará a todos los individuos que se hayan añadido a la lista con dicho fin para seguirlos durante la reproducción.

La otra función es la de cambio de cámara. Actualizaremos el número de cámara de la máquina de estados en función de que cámara queremos cambiar. Si se detecta éste cambio de cámara, buscaremos la que tenemos actualmente y la reemplazaremos por esta última.

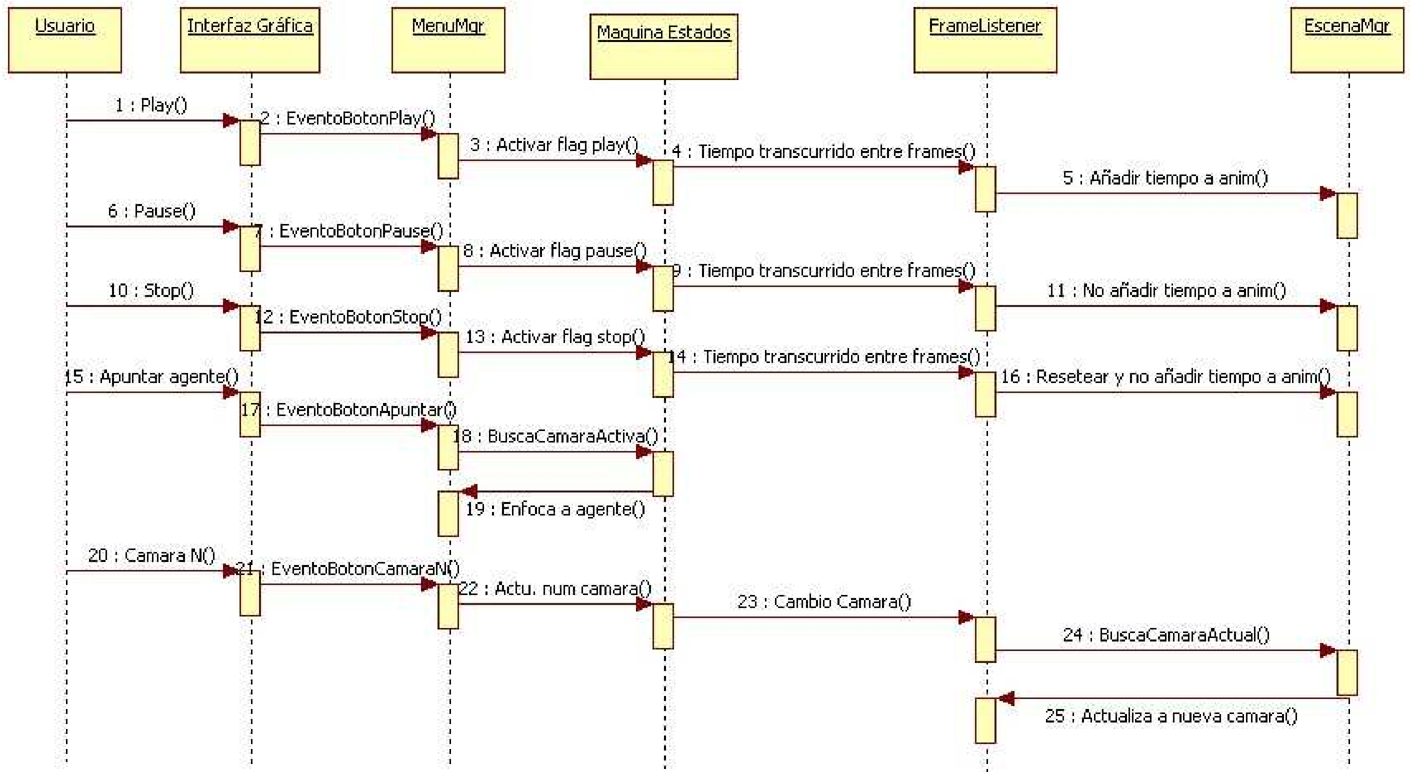


Figura 23

5. Implementación

En este capítulo se enseñarán los conceptos básicos sobre los motores para poder entenderlos, el algoritmo general de la aplicación y la explicación más a bajo nivel de cómo se ha realizado el programa.

5.1 Conceptos básicos sobre las API's

En este apartado se explica la estructura y el funcionamiento de las API's utilizadas para comprender, posteriormente, la implementación de la aplicación.

5.1.1 OGRE 3D

5.1.1.1 Root

Este objeto es el punto de entrada al motor gráfico OGRE, siendo el primero en ser creado y el último en ser destruido.
Una de sus funciones principales es la configuración del sistema del motor de renderizado como resolución, profundidad de color, pantalla completa, etc.
También permitirá obtener punteros de otros objetos importantes del sistema que se comentarán a posteriori.
Además de todo esto, permite renderizar la escena en un solo frame o, de forma más automática, incorporar un lazo de renderizado añadiendo un FrameListener a dicho objeto, provocando un render continuo de la escena.

5.1.1.2 RenderSystem

Este objeto hará de interfaz entre OGRE y la API de render utilizada (DirectX u OpenGL). Es el responsable de enviar operaciones a éstas y configurar todas las opciones de renderizado.

5.1.1.3 SceneManager

Este es uno de los objetos más importantes y de los más utilizados a la hora de interactuar con el motor. Dicho objeto contendrá todas las referencias y propiedades de la escena que se renderizarán en el motor.
Es el responsable de manipulación de cámaras, objetos movibles (entidades), objetos estáticos, luces, materiales, etc.

5.1.1.4 ResourceGroups

Este objeto es el encargado de leer todos los recursos que el motor utilizará. Nos referimos a recursos con las mallas, animaciones, materiales, texturas, etc. Existe un fichero de configuración externo en el que se especifican dónde están ubicados los recursos llamado “resources.cfg” (ver Figura 24).

```
#Resources dónde se ubicarán los ficheros de datos de la simulación

[DataSimulacion]   <---- Nombre del grupo del recurso
FileSystem=../../data   <---- Directorio donde se encuentra el recurso

#Resources dónde se ubicarán los ficheros con los materiales de los fondos
[Fondos]
FileSystem=../../fondos

#Resources dónde se ubicarán los ficheros con las animaciones de los modelos
[Animaciones]
FileSystem=../../animaciones

#Resources dónde se ubicarán los ficheros con los meshes de los modelos
[Modelos]
FileSystem=../../modelos

#Resources dónde se ubicarán los ficheros con los materiales
[Materiales]
FileSystem=../../materiales
```

Figura 24

El nombre del grupo del recurso servirá para hacer filtros a posteriori. Si se observa la primera fila, ésta tiene un nombre de grupo de recurso llamado “DataSimulación”. Esto servirá para posteriormente buscar todos los ficheros que hayan dentro de esa/s ruta/s y en este caso en concreto, encontrar todos los ficheros resultantes de las simulaciones que son candidatas a ser visualizadas.

5.1.1.5 Meshes (Entity)

El objeto mesh representa un modelo discreto, es decir, un conjunto de geometría movible como personajes, objetos, etc.
Estos objetos son cargados a partir de archivos con extensión *.mesh .El cambio de un módelo 3D a dicho formato se hace a partir de exportadores ya creados y están disponibles para varios programas de modelado tridimensional como 3DStudioMax, Blender, Lightwave,... en los que se realiza el modelado y siguientemente la exportación al formato de OGRE.
Estos tipos de objetos llevan una referencia a un fichero de animaciones del modelo en el caso de que las tengan.
Los objetos Entity contienen estos formatos y tienen asociados la geometría, animaciones y materiales. Las entidades no necesitan una posición en el espacio, son simplemente la estructura del modelo en sí.
En el momento en que se instancia una entidad desde el EsceneManager, se le debe de asignar un nombre único que permitirá recuperarla y modificar sus propiedades en cuanto se requiera.

5.1.1.6 SceneNodes

Los objetos SceneNode sirven para enlazarse con las entidades y que éstas puedan ser utilizadas en la escena. Un Entity, como ya se ha dicho anteriormente, no tiene una posición en el espacio de renderizado, solo la relativa al modelo.
Al relacionar un SceneNode con un Entity ya es posible realizar movimientos, escalados y rotaciones del nodo sin alterar la entidad en sí, siendo las transformaciones relativas al nodo y no a la entidad.
Se pueden crear relaciones complejas en forma de árbol de SceneNodes, pudiendo realizar una transformación a un nodo padre y afectando ésta a todos los nodos hijos que tenga por debajo (ver Figura 25).

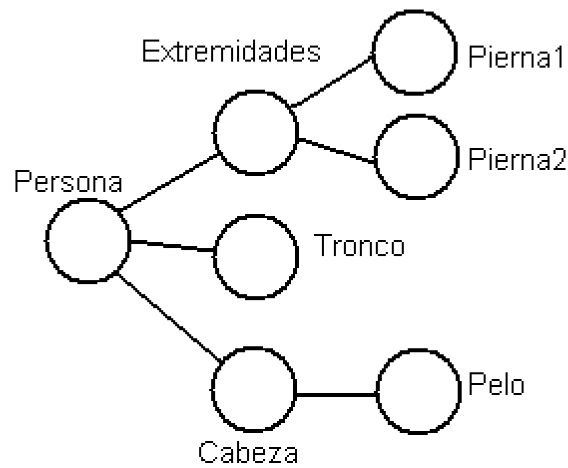


Figura 25

Por ejemplo, una traslación al nodo “Persona” afectará a todos los nodos hijo, aplicando la misma función de traslación a todos ellos.

5.1.1.7 Materials

Los materiales indicarán cómo se van a renderizar los objetos que los tengan asignados. De otra forma, establece las propiedades de la superficie, como la reflectáncia de los colores, el brillo, texturas, shaders que se van a aplicar, etc.
Los materiales se pueden editar directamente sobre código aunque para el proyecto se ha utilizado el potente sistema de scripting. Cada material se guardará en un fichero con las propiedades de éste. Posteriormente se lo asignaremos a una entidad (no a un nodo).
Los Mesh se pueden exportar (con los mismos exportadores de antes) también generando el fichero (o script) del material en cuestión.
Aquí se puede ver un ejemplo de material que usará un tipo de individuo, indicando únicamente la textura que se va a usar:

```
material pez
{
	technique
	{
		pass
		{
			texture_unit
			{
				texture pez.png
			}
		}
	}
}
```

Los materiales pueden tener llamadas a rutinas gráficas llamados shaders. Los shaders son programas (normalmente a más bajo nivel) que aplican funciones de transformación a los vértices o a los píxeles, de tal forma que podemos dar efectos más realistas a los renderizados. Estos programas son rápidos y se cargan directamente sobre el hardware de la tarjeta grafica (GPU), pudiendo así acelerar sus cálculos.

Tenemos tres tipos de shaders:

- Pixel shaders – Sirven para modificar el color de un cierto pixel en función de unos parámetros de entrada. Esta función se aplica a todos los píxeles del Mesh el cual tiene asignado el material.
- Vertex shaders - Sirven para modificar la posición de un vértice en el espacio en función de unos parámetros de entrada. Esta función se aplica a todos los vértices del Mesh el cual tiene asignado el material.
- Compositors – Sirven para modificar el color de la proyección de renderizado, es decir, el frame que se muestra por pantalla, en función de unos parámetros de entrada. Los píxeles de entrada serán en dos dimensiones (x,y). Con estos se podrán crear renderizados de difuminado, blanco y negro, etc.

5.1.1.8 Animations

Las animaciones pueden ser de dos tipos:

- Skeletal Animation – Estas animaciones indican como la malla se deforma en función del tiempo.

  El fichero mesh, en caso de que tenga una animación de este tipo, tendrá asignado también unos huesos y unos pesos que moverán los vértices de las caras más o menos en función de éstos.

  Los movimientos respecto del tiempo estarán en un fichero con extensión *.skeleton. Estos ficheros también serán generados por los exportadores en el caso de que haya una animación esqueletal.

- KeyFraming Animation – Este tipo de animaciones definen una transición entre un punto inicial y uno final aplicando una interpolación de movimientos entre un espacio de tiempo determinado. Se pueden realizar tres tipos de transformaciones: traslación, escalado y rotación.
  Aquí se han generado directamente desde código aunque los exportadores las generen en un XML definiendo el tiempo entre punto de inicio y final y las transformaciones correspondientes.

5.1.1.9 FrameListener

Un listener o escuchador controla un bucle infinito que se ejecuta y escucha ciertos parámetros de entrada para así poder controlar sus salidas.
El FrameListener es un listener y nos proporciona dos funciones que nos ayudan a controlar qué se requiere hacer antes y después del renderizado de un frame. Se ha decidido incluir en el listener la gestión de la entrada/salida, es decir, qué teclas se han pulsado y que eventos de ratón se han tenido.
Los FrameListener ya llevan consigo mismo un control de tiempo para evitar variaciones de velocidad dentro de la escena. Esto significa que se calcula cuanto tiempo ha pasado entre el render de un frame y otro, ajustando los movimientos y animaciones a dicho tiempo. De esta forma se evita que si se tiene una máquina con más prestaciones el programa se acelere y viceversa.

5.1.2 CEGUI

5.1.2.1 OgreCEGUIRenderer

CEGUI tiene compatibilidad con múltiples motores y API's. Esto indica que debe de poder comunicarse con ellos. Esta clase es particular para Ogre y será el objeto que enlazará el motor Ogre con CEGUI. Se debe de indicar cuál es la RenderWindow (ventana de renderizado) y nuestro SceneManager.

5.1.2.2 System

Una vez se ha inicializado el objeto anterior y está establecido como interfaz de comunicación entre API's, ya se puede inicializar el sistema CEGUI con éste objeto. Se definirá un Scheme (conjunto de controles ya creados) y a

continuación ya se indicará cuál es la fuente de escritura, el cursor para el ratón, etc.
Este objeto definirá que plantillas están activas para habilitar o deshabilitar los menús que interesen en todo momento.

5.1.2.3 Windows

Con estos objetos se crearán los controles que interesen definiendo sus propiedades como el tipo de control, el texto que se requiere que aparezca, sus dimensiones, sus posiciones, etc.
Un dato importante sobre este tipo de objeto es que se pueden crear árboles (ver Figura 26), en los que se podrán, por ejemplo, ocultar al padre y en consecuencia ocultar a todos sus controles hijos, ideal para la generación de los menús.

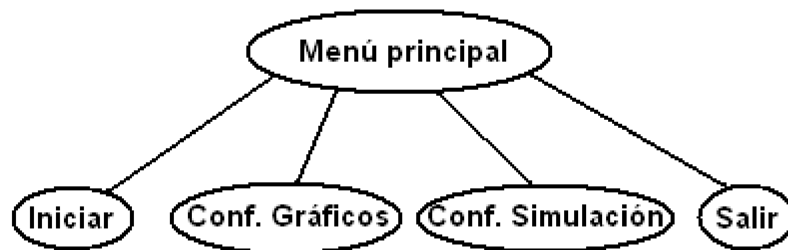


Figura 26

El punto más importante sobre esta clase es que se puede relacionar un evento generado de dicho control con la función que se requiera ejecutar. De esta manera se indicará a nuestra aplicación qué se quiere hacer en el caso de que se pulse, por ejemplo, un botón.

5.2 Implementación de la aplicación

5.2.1 Algoritmo general

La aplicación, una vez ejecutada, inicializa el motor de Ogre, carga el fichero de configuración y crea la escena a partir de los datos de este. Posteriormente se crea toda la interfaz gráfica 2D (GUI) y se inicia la E/S dentro del bucle de renderizado empezando ya a pintar la escena.

Por cada frame, se actualizara la cámara activa y se entrará en toda la lógica de control de la aplicación.

Si la carga no está iniciada se volverá otra vez al principio del bucle.
Si la carga se está iniciando se borrará la simulación anterior y se cargará la nueva, volviendo al inicio del bucle.
Si la carga ya está finalizada se controlarán los estados del menú multimedia. En caso de que la reproducción esté activa (Play), se calculará en qué posición está el scroll de animación y se le añadirá el tiempo que ha pasado entre frame y frame, asignándoselo también a las animaciones.
En caso de que la reproducción esté parada (Stop), se iniciará a tiempo = 0 la animación y se establecerá el scroll al principio.

Hecho esto se controlará, en el caso de que el mar esté activado, si la cámara está por debajo del mar o por encima y verificando la activación de la niebla azulada o no.

Finalmente se renderizará la escena y se volverá al principio del bucle realizando una y otra vez las operaciones.

Los eventos con la GUI, generalmente, solo se dedicarán a cambiar estados de la máquina de estados.

En la Figura 27 se puede observar, mediante diagrama de bloques, la estructura del algoritmo general.

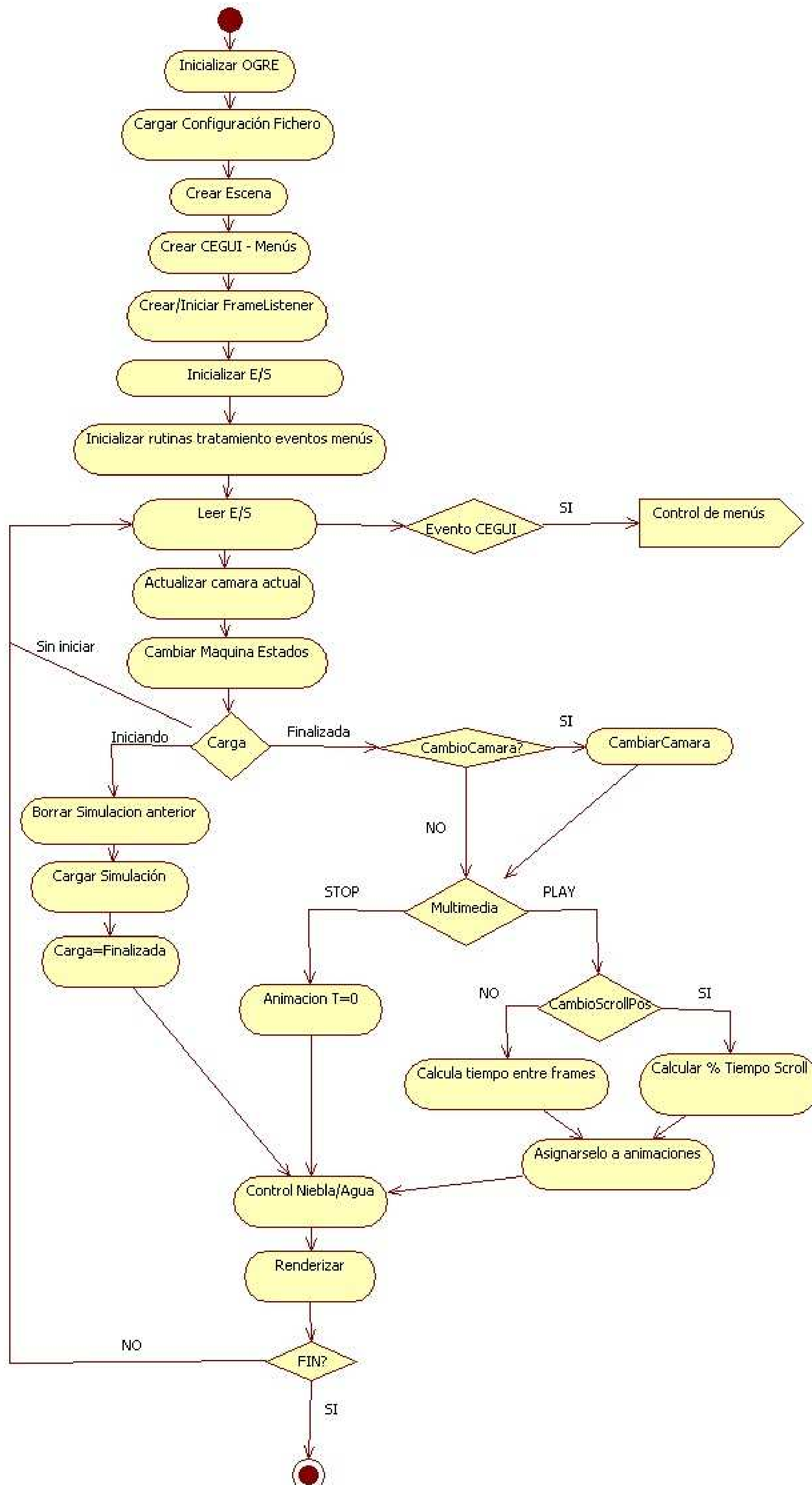


Figura 27

5.2.2 Aplicación de visualización de simulación 3d

5.2.2.1 Inicialización OGRE:

Una vez se ha creado el objeto de aplicación principal primeramente lo que se hará es indicar la ruta del fichero “plugins.cfg”. Este fichero contendrá todos los plugins que el motor deberá de inicializar para la correcta ejecución de la aplicación como el plugin de RenderSystemDirectX, el RenderSystemOpenGL, el de lectura de shaders “CG”, etc. También se buscará la ruta del fichero “Ogre.log” que servirá para ir escribiendo en éste los log’s que interesen durante la ejecución del programa. La ultima ruta será la de “Ogre.cfg” que será un fichero que guardará los parámetros de inicialización del motor, es decir, del cuadro de dialogo que aparece solo al arrancar la aplicación.

Una vez se han registrado estas tres ubicaciones de ficheros ya se puede crear el objeto raíz de Ogre llamado Root del que colgará todo el motor gráfico, pero, sin inicializar.

Posteriormente se buscaran los recursos o Resources. Esto nos añadirá al ResourcesManager todos los ficheros que tengamos indicados en las rutas del resources.cfg para, posteriormente, poder olvidarse de estas rutas y trabajar solo con el nombre del archivo.
En el caso de que los ficheros sean scripts, no guardará el nombre del archivo sino los nombres que contengan dentro los archivos. Como buen ejemplo se tienen los materiales que, dentro de un mismo fichero se pueden tener varios nombres de materiales asignados. Éstos son los que se usaran en el motor y no el nombre de los ficheros.

Una vez hecho esto, ya se puede mostrar al usuario el cuadro de diálogo de Ogre (ver Figura 28):

Figura 28

En el caso de que el usuario pulse OK, los parámetros quedarán registrados en el fichero anteriormente comentado llamado “Ogre.cfg” y seguidamente ya podremos inicializar el objeto Root con los parámetros indicados.

Ahora ya se puede crear el EsceneManager, de tal forma que se creará toda la estructura de la escena en su interior.
Primero se crearán las cuatro cámaras a utilizar en nuestra aplicación llamándolas “Camara1”, “Camara2”, “Camara3” y “Camara4”. En la visualización de la escena solo estará activada una de las cuatro, por defecto, siempre la primera.
Posteriormente se creará el ViewPort. El Viewport es la sección de la pantalla donde se quiere renderizar la visión de una cámara. Se puede imaginar un juego de carreras con dos jugadores. La pantalla queda partida en dos secciones en la que en cada una se muestra una cámara diferente. Tener en cuenta de que el renderizado es todo sobre la misma escena, con lo que existirían dos viewports asignados a ésta. En este caso, solo vamos a crear uno (de momento) y todas las cámaras asignadas a éste.

5.2.2.2 Crear Escena:

Primero de todo se inicializará la máquina de estados, cogiendo los valores del fichero creado de configuración llamado “ConfigSim.cg”. El formato que tiene es el siguiente:

Velocidad
Fichero Simulacion
Mesh
Escala
Animacion
Red
Green
Blue
Cielo
Num. Individuos
Num. Max. Individuos
Offset x
Offset y
Offset z
Ver mar

Posteriormente, se inicializará la luz ambiental y la puntual de forma que quede una atmósfera adecuada.
El cielo o fondo establecido será el material que esté guardado en la máquina de estados, leído previamente.
Finalmente se leerá el Mesh que se quiere utilizar como fondo, cargándolo dentro de la escena y en la ubicación deseada.

5.2.2.3 Carga de la visualización de simulación:

La carga de visualización de la simulación en memoria es el proceso que más lecturas de E/S al disco físico provoca y puede tardar varios minutos en función de la duración de la simulación que quiere visualizar.
Una vez se han establecido los parámetros gráficos y de simulación ya se puede iniciar la carga en memoria.

Primeramente, al lanzar el proceso, se contarán cuantos segundos de simulación tenemos. Esto se hará buscando, en el directorio de la simulación, cuantos ficheros "fpos.*" existen. Un bucle desde 0, irá verificando la existencia de ese archivo e incrementando la variable hasta que no encuentre más. Una vez hecho esto, ya se puede asignar el tiempo máximo de representación de la simulación a la máquina de estados.

Siguientemente los agentes se situarán en tiempo inicial = 0 tanto con posición como con dirección.
Para hacer esto se recontará el número máximo de individuos recorriendo el primer fichero de posiciones (fpos.0) y contando cuantas filas tenemos hasta el final del fichero. Este valor se registrará en la máquina de estados.
Hecho esto, se crearán, con memoria dinámica, un vector de vectores tridimensionales y otro de Individuos (clase que contiene el Mesh y el SceneNode del individuo) ambos del tamaño del número máximo de agentes.
Ahora ya se podrá recorrer el fichero (ver formato en Figura 29) de tiempo 0 e ir asignando las posiciones, que se van a leer del fichero resultante de la simulación, al vector de posiciones creado anteriormente.

```
474 95 16 0
126 319 18 1
410 70 17 2
254 407 6 3
245 172 19 4
365 457 17 5
 x   y   z  nº ind
```

•
•
•

Figura 29

A continuación se inicializará el vector de Individuos indicándo un nombre de individuo, el Mesh a utilizar (el que haya en la máquina de estados) y la posición leída del vector de posiciones.

Llegados a este punto ya se puede empezar a generar la animación general. El motor OGRE permite crear por código y guardar en memoria una animación por KeyFraming, ya comentada anteriormente. De esta forma, se creará una animación por cada agente utilizando las interpolaciones de movimiento, pudiendo aplicar las transformaciones del tipo traslación, rotación y escalado a cada extremo de la interpolación.

Estando todavía en la función de creación en tiempo = 0, se creará la animación principal por cada pez indicándo un nombre de animación y un tiempo máximo de representación de la simulación (el de la máquina de estados). La interpolación se definirá del tipo SPLINE para que las transiciones sean suaves y definiremos que para tiempo = 0 de la simulación el agente estará en la posición que se acaba de leer enlazando el SceneNode del agente con este KEY o punto de referencia entre transiciones de la animación. En la Figura 30 vemos un ejemplo de animación por keyframing [32]:

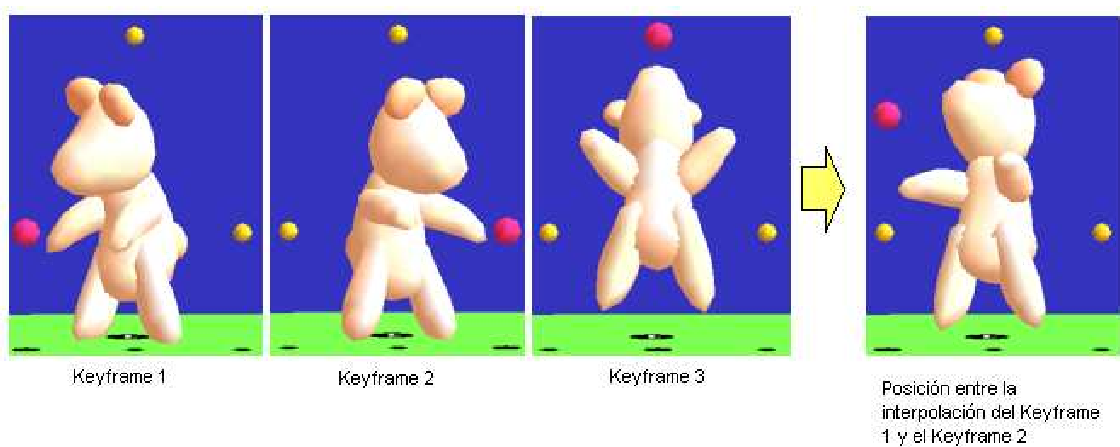


Figura 30

Hecho esto con las posiciones ya se podrán leer las direcciones iniciales de la misma manera, leyendo del fichero “fdir.0” para asignárselas a los agentes.

En este momento están las posiciones y direcciones preparadas para tiempo=0 pero se necesita crear el resto de la animación. Para segundo = 1 hasta el tiempo máximo de representación de la simulación se creará una interpolación de movimiento por cada uno de los agentes.

Para hacer esto, se leerá el fichero de posiciones del instante de tiempo que se va a controlar. A continuación se hará la lectura del fichero de direcciones del instante de tiempo que se está controlando y por cada agente se va a volver a crear otro KEY indicando la posición y la rotación. Para indicarle la rotación se guardará el vector de direcciones del instante de tiempo anterior para poder buscar cual ha sido la rotación entre el vector director anterior y el actual, pudiendo aplicar este resultado como rotación entre un segundo y el otro.
Esto creará la animación general de cada individuo por separado.

Hasta aquí ya está creada la animación general por KeyFraming, pero no existe un movimiento propio del individuo. Si por ejemplo, se utilizan peces, y se quiere ver el movimiento continuo de la aleta en caso del que el Mesh contenga esta animación esqueletal, ahora mismo no se apreciaría.
Lo que se hará es, que en caso de que tenga animación, se leerá de la máquina de estados (elegida en el menú de configuración de gráficos) y por cada individuo se activará.

Hecho esto la visualización de la simulación ya está preparada para ser lanzada.

5.2.2.4 Creación CEGUI y control de menús:

Lo primero que se hará para crear los menús es asignarle a la API CEGUI una interfaz por la cual pueda comunicarse con el sistema de renderizado de OGRE, ya que son API's diferentes.
Una vez el sistema ya sabe cómo renderizar le se asigna un esquema de CEGUI, en el que habrán una serie de controles (botones, barras de desplazamiento, cuadros de texto,…) ya preestablecidos y que se podrán usar libremente. Hecho esto le se asignará el cursor del mouse (también incluido en el esquema).

Lo siguiente es crear la estructura de todos los menús a utilizar en el programa. Cada menú tendrá como hijos a todos los controles del mismo. Para cada control se definirá de qué tipo es, su posición en la pantalla, su tamaño (estas dos últimas en coordenadas (x,y) siendo el mínimo 0 y el máximo 1) y la configuración de las propiedades intrínsecas del control en sí.
Siguientemente se explicarán los diferentes menús creados, con una breve explicación de lo que se ha hecho en cada control.

<u>MENU PRINCIPAL:</u>

Figura 31

En la Figura 31 se presenta el menú que se mostrará al iniciar la aplicación y el que orientará para llegar a todas las opciones de configuración y funcionalidad de la aplicación. Si se observa en la imagen se han utilizado cuatro controles del tipo “Button” para acceder a cada una de las secciones del programa.
Cada uno de estos botones tendrá su propia función de tratamiento de eventos. Para enlazar la función con el control se hará la relación del tipo de evento de éste con el principio de la dirección de memoria de la función que se requiera utilizar para esa acción.
En este caso para el botón “Salir” se vinculará el evento “Clic” del botón con la función “MenuPrincipalSalir()” en la que quedará activado el flag “mSalir” a verdadero, finalizando así la aplicación.

CONFIGURACIÓN SIMULACIÓN:

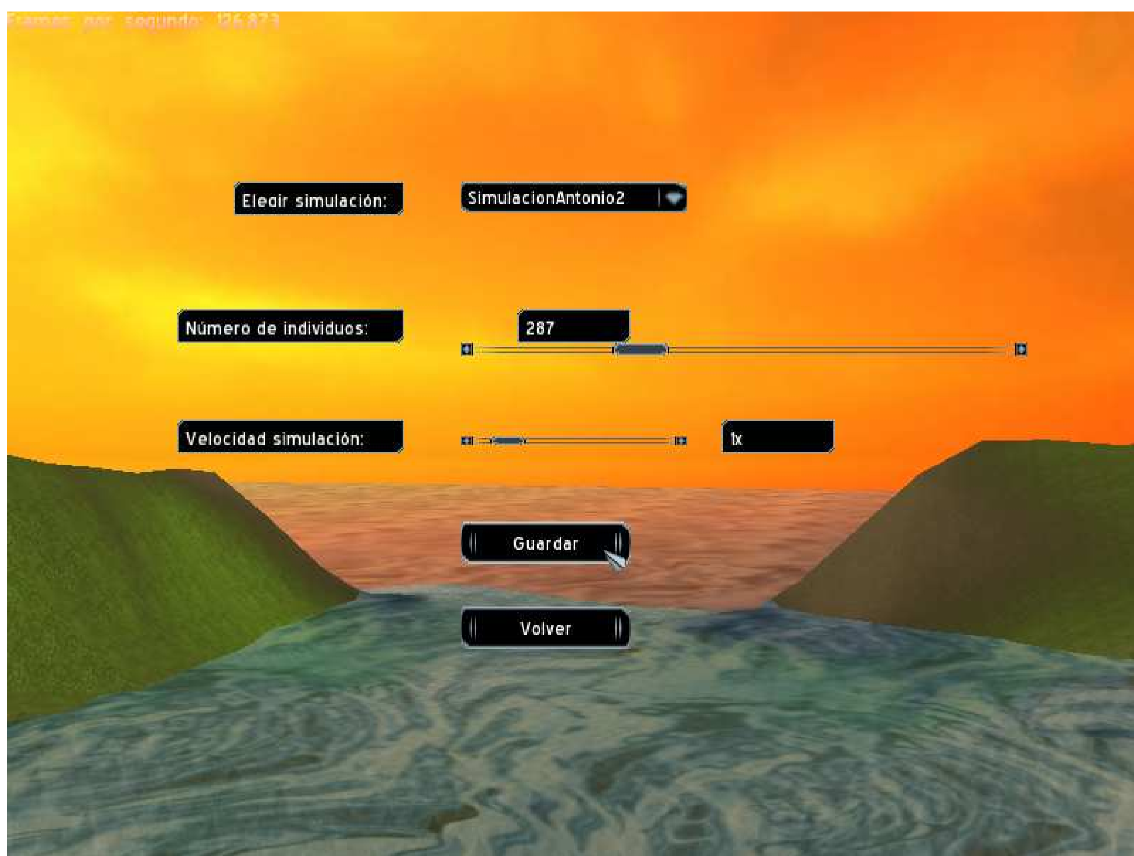
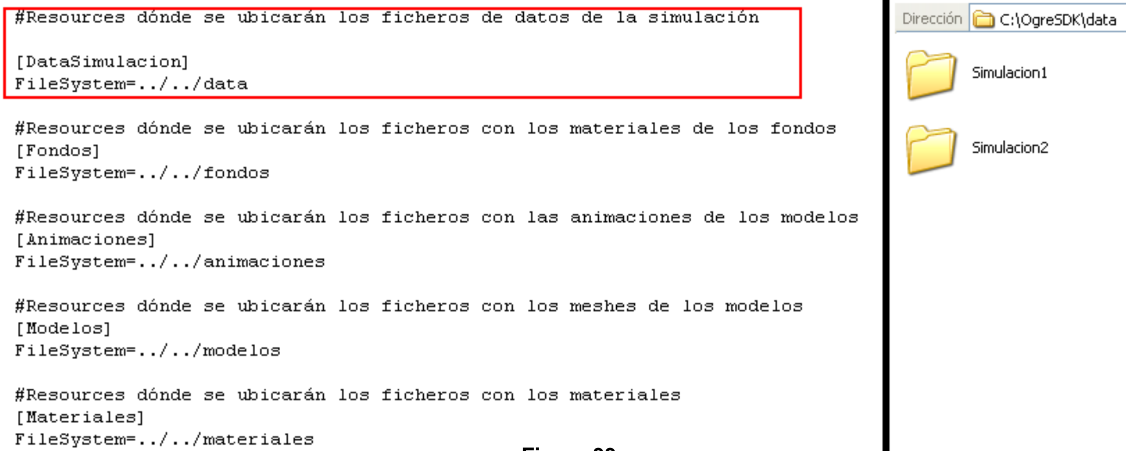


Figura 32

En la Figura 32 se presenta menú para configurar los parámetros de la representación de la simulación.
El primer control que se ubicará es un “Combobox”. Éste es una lista desplegable en el que se muestra el nombre de las carpetas que contienen los

ficheros resultantes de las simulaciones. Para hacer esto se hace referencia al fichero “resources.cfg”, introduciendo una línea indicando así la ruta donde estarán los directorios de dichos ficheros (ver Figura 33).



```
#Resources dónde se ubicarán los ficheros de datos de la simulación

[DataSimulacion]
FileSystem=../../data

#Resources dónde se ubicarán los ficheros con los materiales de los fondos
[Fondos]
FileSystem=../../fondos

#Resources dónde se ubicarán los ficheros con las animaciones de los modelos
[Animaciones]
FileSystem=../../animaciones

#Resources dónde se ubicarán los ficheros con los meshes de los modelos
[Modelos]
FileSystem=../../modelos

#Resources dónde se ubicarán los ficheros con los materiales
[Materiales]
FileSystem=../../materiales
```


Figura 33

Por lo tanto, lo que se hará es recorrer todos los directorios de “DataSimulacion” y añadirlos al combobox. De esta forma, cuando el usuario elija el directorio, ya se podrá establecer la ruta de los ficheros generados por la simulación.

En el momento que el usuario escoja alguna de las visualizaciones de simulaciones, se guardará el nombre en la máquina de estados y automáticamente se invocará a una función que recorrerá el primer fichero de posiciones. A continuación contará cuantos individuos como máximo hay en el resultado de la simulación seleccionada, guardando el resultado en la máquina de estados para actualizar el control que a continuación se detalla.

Para gestionar cuantos individuos se van a mostrar se han utilizado dos controles básicos. Uno es un “Textbox” y el otro un “Scrollbar”.

El primero es un cuadro de texto informativo y se actualizará con el número máximo de individuos para que el usuario tenga constancia.

El segundo es una barra de desplazamiento. Como valor interno del cursor se registrará el número uno, es decir el 100%, con lo que el cursor se nos irá al final de la barra.

En el momento que el usuario desplace el cursor (evento de posición cambiada) se hará una multiplicación entera del valor máximo de individuos multiplicado por el porcentaje del scroll. Por ejemplo si se coloca el cursor en la

mitad (valor 0.5) y hay un máximo de individuos de 100, el textbox nos mostrará el valor 50. Este valor se añadirá en la máquina de estados, siendo el número de individuos que se mostrarán en la escena.
La velocidad de la animación será otro punto importante y también se gestionará con un “Textbox” y un “Scrollbar”. Dicha velocidad nos irá de 0 a 10, es decir, se multiplicará el tiempo a añadir en la animación por el valor que se elija en esta opción.
La idea es la misma que en el punto anterior. Se realizará la multiplicación entera del porcentaje del scroll y por el valor 10. El valor del textbox también se registrará a la máquina de estados.

Una vez tenemos esto se añadirán dos botones más.
El botón “Guardar” escribirá en el fichero de configuración “ConfigSim.cg” los valores actuales de la máquina de estados, para que al entrar en la aplicación no se pierdan estas propiedades ya configuradas.

El botón “Volver” únicamente ocultará este menú y hará aparecer el menú principal.

CONFIGURACIÓN GRÁFICOS:

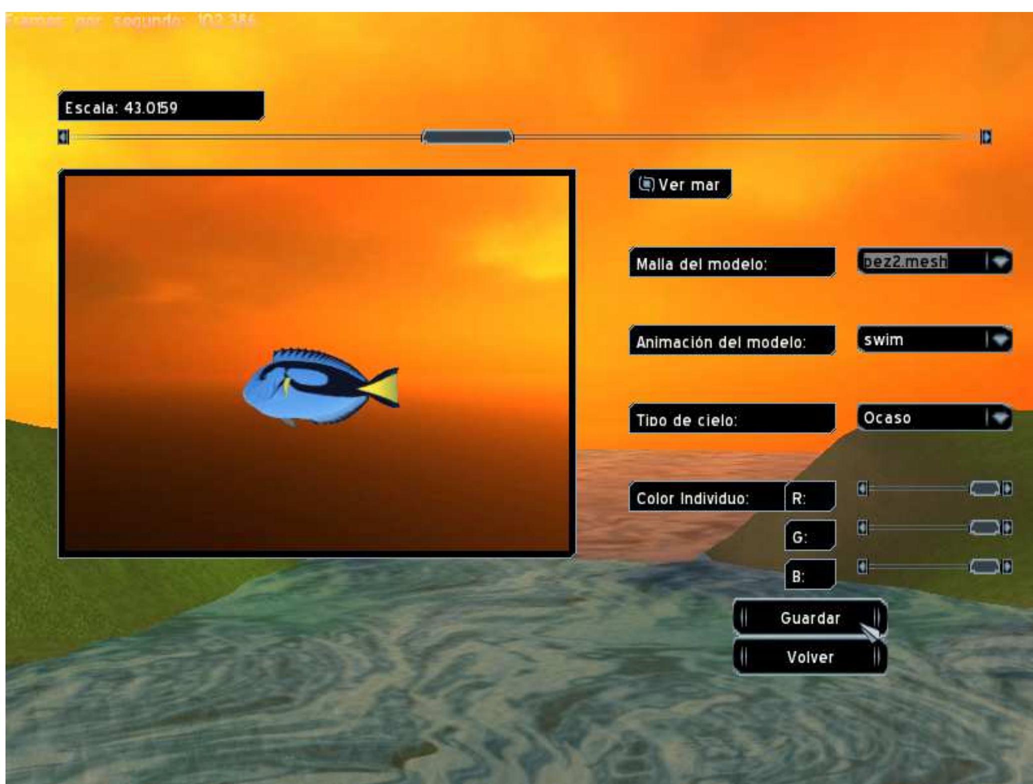


Figura 34

En la Figura 34 se muestra el menú que servirá para configurar todos los parámetros gráficos modificables de nuestra aplicación.
Muchos de estos controles lo que harán es consultar la información guardada en “resources.cfg” para así buscar los recursos que interesen.

El primer control que se ha añadido es el de cambio de malla o de modelo. Éste será un “Combobox” el cual será inicializado con todos los Meshes que esten dentro de las rutas de la sección “modelos” del fichero “resources.cfg”
Una vez el usuario haya elegido la malla a mostrar, automáticamente se rellenará el siguiente control.
El segundo control es el de cambio de animación del Mesh. Este control se rellenará en función del Mesh seleccionado en el control anterior. Como ya se ha comentado, cada modelo tiene su *.skeleton asociado o lo que es lo mismo, el fichero con las animaciones intrínsecas a éste.
Una vez seleccionado el modelo, se añadirán al combobox todos los nombres de animación que la malla tiene. De esta forma el usuario podrá elegir qué animación de modelo mostrar cuando se esté ejecutando la representación de la simulación.

El siguiente control es también un “Combobox”. A éste se le añadirá todos los materiales que se correspondan como cielo o fondo. En el archivo “resources.cfg” existe una línea llamada “Fondos” en la que se establecerán las rutas de los materiales que se podrán utilizar como fondos.
Estos ficheros contendrán solo un material llamándose exactamente igual que el nombre del fichero, de tal forma que facilitará su lectura y gestión. De este modo, los nombres que se añadirán al combo serán los nombres de los ficheros *.material de esas rutas.

También se ha añadido un “Checkbox “ para mostrar u ocultar el mar. Este valor booleano del control se guardará en la máquina de estados para posteriormente saber si se ha de renderizar o no.

Se ha añadido un apartado de filtro de colores de la textura del individuo. Esto servirá para filtrar en tanto por ciento, los colores (Red, Green, Blue) de esa

textura. Cuando alguno de los tres scrolls cambie de posición, se registrarán en la máquina de estados los tres porcentajes de color y consecuentemente se reescribirá el material del individuo en memoria asignando el nuevo color.

De la misma manera que en el menú de configuración de la visualización de la simulación existirán los botones “Guardar” y “Volver” que realizarán exactamente la misma función.

GUI VISUALIZACIÓN SIMULACIÓN:

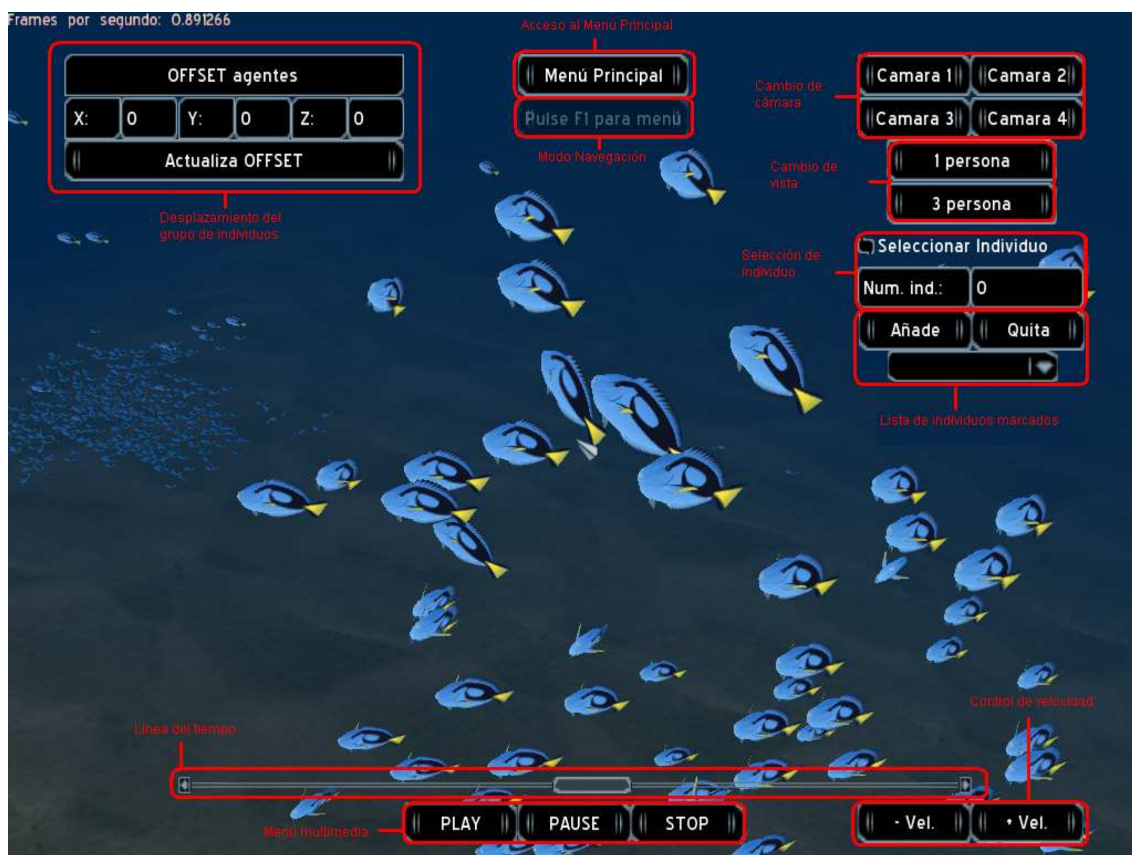


Figura 35

En la Figura 35 se muestra la GUI de interacción con el mundo tridimensional de la visualización de la simulación. El menú se dividirá en 8 secciones de funcionalidad que a continuación se irán mostrando.

Para el menú multimedia se añadirán tres botones que controlarán el estado de la animación. La única función que tendrán será cambiar el estado del flag “mMultimedia” a 0, 1, 2 en función de si se ha pulsado el botón Stop, Play o

Pausa, para gestionar desde el Framelistener si se va a añadir tiempo a la animación o no.

Si observamos en la parte inferior de la pantalla se ha añadido un “Scrollbar”. Este servirá de indicador o modificador del tiempo transcurrido en la visualización.
A su izquierda tendremos dos botones de incremento o decremento de velocidad de simulación. Éstos sumarán o restarán en una unidad dicha velocidad guardada en la máquina de estados.

En el momento que se inicia la animación existirá un tiempo máximo de la representación guardado ya en la máquina de estados. El tiempo transcurrido de ésta también se irá registrando, actualizándose a cada frame renderizado. La posición en tanto por ciento del cursor del scroll se irá actualizando con esta relación (tiempoactual/tiempototal). De esta forma, se observará como el cursor va avanzando conforme la simulación evoluciona por el tiempo hasta llegar a su totalidad.
En el momento en el que el control invoca el evento de posición del scroll cambiada, es decir el usuario ha movido el cursor del scroll, se activa un flag booleano en la máquina de estados indicando este cambio manual, para posteriormente lanzar la rutina en la que se calcula hasta que tiempo se ha movido el cursor. Este tiempo se calcula capturando la posición en tanto por ciento del scroll y multiplicando por el tiempo total representación. De este modo se habrá obtenido un valor de tiempo, el cual será asignado a la animación global.

Otra opción en la GUI es el cambio de modo. Este botón permitirá pasar al modo “Navegación” en el que el usuario podrá moverse por el mundo tridimensional con el teclado y el ratón.
Al pulsar el botón se desactiva el buffering de la E/S y directamente se captura la información de los dispositivos. Esto hará que se modifique la orientación de la cámara con el ratón de manera que el usuario se pueda mover por el mundo.

Una vez se ha pulsado el botón, éste cambiará su texto a “Pulse F1 para menú”. De esta forma se podrá volver al modo GUI y poder, otra vez tener, el enfoque sobre el cursor.

A la derecha de la pantalla tenemos cuatro botones que servirán para realizar el cambio a alguna de las cuatro cámaras. Al pulsar uno de ellos cambiará la máquina de estados con el número de la cámara activada, habilitando el flag de cambio de cámara para que éste se realice. Una vez cambiada, este flag se volverá a deshabilitar.

En la sección de “Selección de individuo” se muestra un Checkbox. Si se activa se podrá seleccionar el individuo con el cursor y realizando un clic cuando el individuo deseado esté enmarcado por una caja transparente (habilitando la Caja mínima del mesh) mostrada en la Figura 36. También se podrá cambiar la selección del individuo escribiendo directamente sobre el texto el número de individuo que se desee tener seleccionado. El número de individuo quedará registrado en la máquina de estados para tener constancia de él.

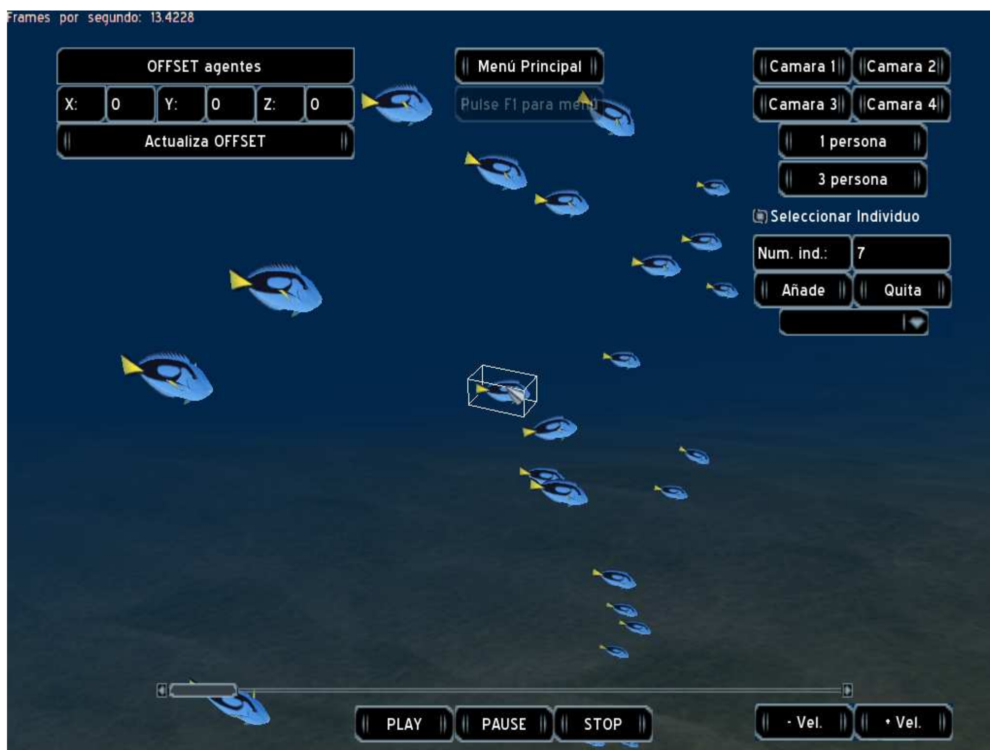


Figura 36

Debajo de las cámaras tenemos dos botones.
Al hacer clic sobre el primero (primera persona), colocará al usuario encima del individuo seleccionado anteriormente. La cámara seleccionada se ubicará en la posición del agente y heredará su orientación en función del tiempo, simulando así la visión del individuo durante toda su trayectoria.
Si se realiza un clic sobre el segundo (tercera persona), la cámara quedará libre en cuanto a posición pudiendo así cambiarla, en cambio, quedará fija en cuanto a orientación, ya que siempre estará orientada hacia el agente que esté seleccionado. El vector director de esta orientación será: PosCamara – PosAgente.
En ambos casos guardaremos en la máquina de estados si hay alguno de los dos botones activado y cuál.

Un poco más hacia abajo se muestra un combobox y dos botones. Estos servirán para marcar (o hacer distinción) a los agentes que estén dentro de esa lista. Estas marcas se harán, también, con cajas transparentes (Cajas mínimas). Cada vez que se pulse el botón “Añadir” se incorporará en el combobox el número de individuo que esté seleccionado. Para borrar un individuo de la lista, se seleccionará previamente del combobox y se pulsará el botón “Borrar”.
Esta misma función se puede realizar de la misma manera con el checkbox “Seleccionar individuo” seleccionado. Pulsando con el botón derecho sobre el individuo, se añadirá el identificador de individuo a la lista. Para borrarlo, volveremos a pulsar sobre el individuo que se desea seleccionar y borrándolo automáticamente del combobox.

Estas tres funcionalidades serán de gran utilidad para realizar un seguimiento de un determinado o múltiples individuos.

En la parte superior izquierda existe un panel de control del Offset de los agentes. Esto es el desplazamiento absoluto en posición inicial que el grupo tendrá a la hora de reproducir la animación. Si por ejemplo se introduce X=0, Y=10, Z=0 y se pulsa el botón “Actualiza Offset”, se desplazará todo el grupo 10 unidades hacia la superficie del mar. Internamente se guardará este Offset y

se sumará al que había anteriormente, guardando la posición absoluta en la máquina de estados. A la hora de reproducir la animación, por cada individuo se sumará la posición absoluta en <x, y, z> y la posición relativa del fichero de simulación, siendo esta suma la posición final de cada individuo.

Finalmente hay un botón de retorno al Menú Principal, pudiendo acceder a todos los puntos de configuración tanto gráficos como de visualización de la simulación.

6. Pruebas y conclusiones

En este capítulo se mostrarán los resultados finales de dicho aplicativo, realizando un análisis sobre los resultados, mostrando los problemas encontrados, buscando posibles mejoras y sacando conclusiones finales sobre el proyecto.

6.1 Pruebas

El medidor de rendimiento a utilizar será el de *Frames por Segundo* o FPS. Este medidor indicará el número de veces que un fotograma se ha renderizado en un segundo, con lo que, a mayor FPS, mayor fluidez se contemplará en la visualización de la simulación, esperando que éste sea el mayor valor posible. Para que una animación se considere fluida los Frames por segundo no deben de bajar de 25 [33], con lo que este valor será la frontera a la hora de considerar los requisitos.
Otro medidor de rendimiento es el de la memoria RAM consumida por la aplicación. La cantidad de datos a leer es muy grande y es un punto a tener en cuenta. Éste dependerá del número de individuos y de la duración de la reproducción de la simulación.

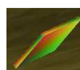
6.1.1 Medidor – Frames por segundo

Se han hecho dos tipos de pruebas con modelos de compilación diferentes en dos ordenadores distintos. Estas pruebas consisten en generar el binario en versión Debug y versión Release.
La versión Debug es generada añadiendo código adicional para que el ejecutable pueda ser debugado paso a paso, sumando este código y el original, con lo que la versión ocupa más espacio en disco (en nuestro caso 372 KB) y es mucho más lenta.
La versión Release solo incorpora el código máquina del programa con lo que el ejecutable ocupa menos espacio en disco (en nuestro caso 139 KB) y es mucho más rápido.

De esto deducimos que nuestra versión Debug ocupa en espacio físico 2,7 veces más que la Release (el incremento es solo código de control adicional y no de la aplicación propia).
Estas dos pruebas se han realizado para calcular los requisitos mínimos del PC, diferenciando entre el desarrollador (Versión Debug) y el usuario final de la aplicación (Versión Release).

Como ya se ha dicho anteriormente, las pruebas se han realizado sobre dos ordenadores distintos. Ambos están equipados de un procesador Intel Core 2 Duo a 2.0 GHz y 2GB de RAM y lo único que los diferencia es la tarjeta gráfica que utilizan:

1. Mobile Intel Graphics Media Accelerator X3100 (Baja calidad)
2. *GeForce 8600 GT (Alta calidad)*

NOTA: Todas las pruebas se han efectuado con el modelo (ver Figura 37) con menos polígonos, un total de 12 triángulos:  **Figura 37**
La carga de modelos y exportaciones se han realizado todas con blender.

COMPILACIÓN VERSIÓN *DEBUG*:

Mobile Intel Graphics Media Accelerator X3100:

nº Individuos	Pausado	Reproducción
100	14,3	11,7
200	7,7	5,3
300	5,3	3,8
400	3,4	3,1
500	2,7	2,7
600	2,5	2,5
700	2,4	2,4
800	2,3	2,3
900	2,2	2,2
1000	2,11	2,11

Figura 38

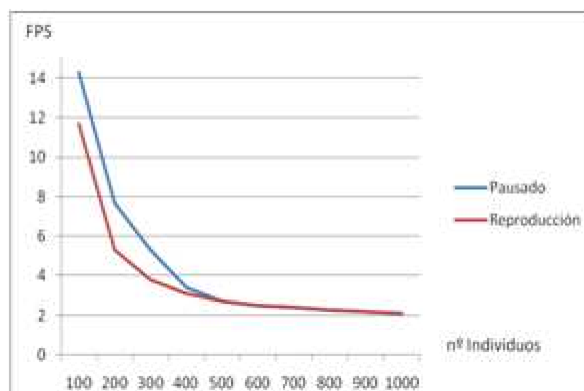


Figura 39

Como se puede apreciar en la tabla de la Figura 38, el punto máximo de velocidad lo alcanzamos en 14,3 FPS y el punto mínimo en 2,11 FPS. Así se observa que se está un poco lejos de los 25 FPS para que la animación sea fluida.

Se observa que con un número de individuos inferior a 500 el modo pausado aumenta más los FPS que en reproducción y con un número superior es irrelevante (las dos gráficas de la Figura 39 se solapan). En conclusiones se explicará el porqué.

GeForce 8600 GT

nº Individuos	Pausado	Reproducción
100	24,6	20
200	13,4	11,5
300	9,23	8,1
400	7,01	6,16
500	5,66	5,05
600	4,73	4,21
700	4,08	3,65
800	3,57	3,18
900	3,17	2,86
1000	2,88	2,58

Figura 40

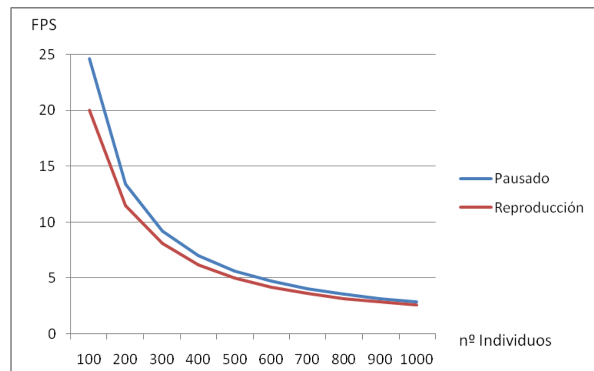


Figura 41

Aquí el punto máximo de velocidad se alcanza en 24,6 FPS y el punto mínimo en 2,88 FPS (ver Figuras 40 y 41).

Con esta tarjeta gráfica alcanzamos casi los 25 FPS en caso de 100 individuos pero prácticamente igual que con la tarjeta anterior para los 1000 individuos.

De esto se puede deducir que para el desarrollo de la aplicación se necesita una tarjeta gráfica y procesador muy potentes si se requiere trabajar con todos los individuos. Se podría limitar el desarrollo con pocos y generalizar posteriormente, ya que el algoritmo a seguir será exactamente el mismo, pudiendo debugar la aplicación a 25 FPS.

COMPILACIÓN VERSIÓN *RELEASE*:

Mobile Intel Graphics Media Accelerator X3100:

nº Individuos	Pausado	Reproducción
100	56,71	56,71
200	55,22	55,22
300	53,88	53,83
400	52,47	52,47
500	51,13	48,2
600	45,3	41,4
700	39,6	34,5
800	34,6	31,2
900	30,5	27,6
1000	27,5	24,4

Figura 42

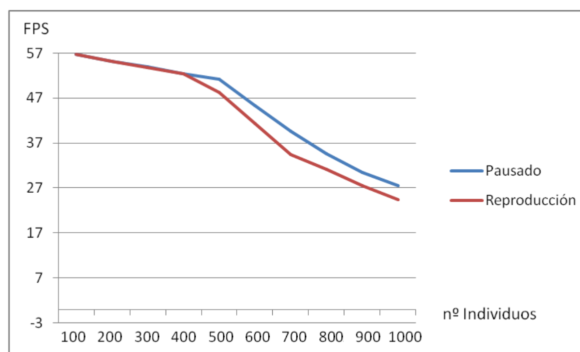


Figura 43

En las Figuras 42 y 43 se puede observar la espectacular subida de FPS de un modo de compilación a otro. El punto máximo aquí es de 56,71 FPS y el mínimo en 24,5 FPS con lo que se deduce que los resultados son más que aceptables en una tarjeta gráfica con poca memoria y velocidad de proceso baja [17].

GeForce 8600 GT

nº Individuos	Pausado	Reproducción
103	259,74	259,74
202	253,49	253,49
302	202,59	212,36
403	156,21	161,67
501	125,87	128,87
602	104,68	108,13
702	89,37	92,53
801	77,61	80,51
901	68,96	71,99
1000	62,68	65,73

Figura 44

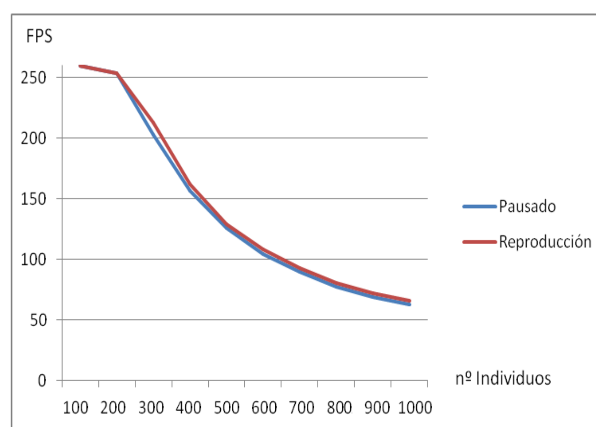


Figura 45

Con esta tarjeta gráfica más potente [34] ya se observa que la fluidez es superior a la que nuestro ojo necesita, teniendo un máximo de 259,74 FPS y un mínimo de 62,68 (ver Figura 44 y 45).

Se puede analizar que existen gráficas en las que los FPS no varían entre en Reproducción y Pausa.
Esto es debido al cuello de botella en el renderizado. La CPU es más rápida que la GPU, con lo que, el cálculo de los datos es más rápido que el renderizado.
Cuando el modo Pausa acelera los FPS es debido a que la adición de tiempo a la animación es más lenta que el renderizado y cuando el modo Reproducción acelera los FPS es porque el bucle de control del modo Pausa es más lento que el de adición de tiempo a la animación.

Recordar que este análisis se ha realizado con un modelo con muy pocos polígonos (12 triángulos concretamente). Se necesitarían tarjetas aceleradoras con más memoria y más velocidad de proceso en función del número de polígonos por individuo a renderizar. En requisitos mínimos se realizará una concreción más exacta.

6.1.2 Medidor – Memoria RAM consumida.

Para analizar el consumo de memoria RAM se utilizará una simulación variando el número de individuos. Siguientemente se irán obteniendo datos en función de la duración de la visualización de la simulación a representar (ver Figura 46 y 47).

Tiempo (seg)	Num. Ind	RAM consumida (MB)
1000		
	333	115
	666	166
	1000	219
2000		
	333	169
	666	277
	1000	385
3000		
	333	221
	666	381
	1000	542
4000		
	333	278
	666	486
	1000	700
5000		
	333	321
	666	578
	1000	836

Figura 46

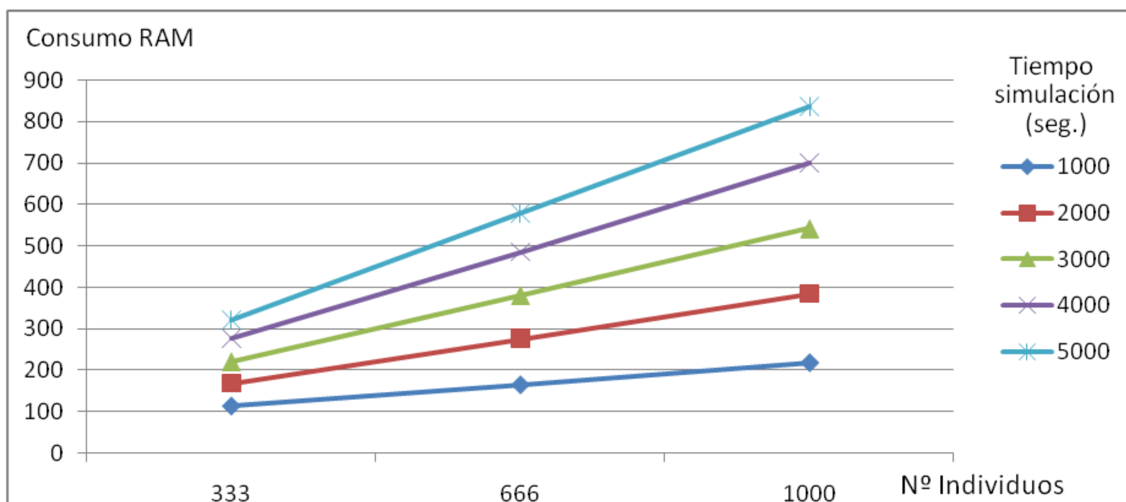


Figura 47

Con los datos de la tabla (Figura 46) se puede determinar que el consumo de memoria es grande y se necesitará un ordenador dotado con bastante memoria. Si se observan la gráficas (Figura 47), se puede apreciar que son casi lineales, con lo que se podría estipular, aproximadamente, cuanta memoria se necesitaría para una visualización de simulación de "Y" individuos y tiempo en segundos "X".

Esto se hará realizando el cálculo de una función que nos determine, en función de estos dos parámetros, cuanta memoria aproximada se necesitará para ejecutar la visualización de la simulación.
Esta función será una recta de la forma: ***y= mx+b*** [35]
Primeramente, calcularemos la pendiente de las cinco gráficas, que se diferencian por tener duraciones de simulación diferentes.

Calculo de las pendientes de las 5 rectas:

m=(219-115)/(1000-333)= **0,156** 1000 seg.
m=(219-115)/(1000-333**)= 0,324** 2000 seg.
m=(219-115)/(1000-333)= **0,481** 3000 seg.
m=(219-115)/(1000-333)= **0,633** 4000 seg.
m=(219-115)/(1000-333)= **0,772** 5000 seg.

Posteriormente, se calculará la diferencia entre las pendientes, para ver si se puede establecer algún patrón lineal.

Diferencia de las pendientes entre las rectas:
0,324-0,156 = 0,168
0,481-0,324= 0,157
0,633-0,481= 0,152
0,772-0,633= 0,139

Se observa que la diferencia entre cada una de ellas es muy aproximada, con lo que se calculará la diferencia media de pendientes por cada 1000 segundos de simulación. Dicho de otra manera, el incremento de pendiente por cada 1000 segundos de visualización de simulación.

Incremento de pendiente media por cada 1000 ind. → (0,168+0,157+0,152+0,139)/4= **0,154**

De esta manera, se puede calcular la pendiente genérica, dados el **número de individuos "X"** y el **número de segundos de la simulación "Y"** (ver Figura 48).

$$Pendiente(x,y) = \frac{((x-333)*y*0{,}154)}{1000}$$

Figura 48

Realizamos un desplazamiento en X de 333. Esto es porque los cálculos se han realizado partiendo de ese número de individuos, con lo que ahora se debe desplazar ese valor al origen.

La división por 1000 es porque los cálculos se han realizado en base a cada 1000 segundos de simulación, con lo que hay que cambiar su base a la unitaria.

Ahora se va a calcular el término independiente genérico de la recta.

Se calculará la diferencia de valores de memoria para el origen de todas las rectas (por dónde pasa la recta en el origen). Hay que recordar que nuestro origen está en 333 individuos.

169-115 = 54
221-169 = 52
278-221 = 57
321-278 = 43

Se puede apreciar que las diferencias son aproximadamente iguales, con lo que se linealizarán calculando el incremento medio de memoria por cada recta en el origen.

Incremento de memoria media de las rectas en el origen→ (54+52+57+43)/4 = **51,5**

Con este valor ya se puede deducir el termino independiente genérico de la recta, dados el **número de individuos "X"** y el **número de segundos de la simulación "Y"** (ver Figura 49).

$$TerminoIndependiente(y) = \frac{(y-1000)*51{,}5}{1000} + 115$$

Figura 49

Se puede observar que realizamos un desplazamiento en Y de 1000. Esto es porque los cálculos se han realizado partiendo de ese número de segundos de simulación, con lo que ahora hay que desplazar ese valor al origen.
La división por 1000 es porque los cálculos se han realizado en base a cada 1000 segundos de simulación, con lo que se tiene que cambiar su base a la unitaria.
La suma de 115 es debido a que la primera recta tiene ese valor de memoria en el origen, con lo que todos los cálculos se han realizado en base a ese valor.
Juntando las dos partes de la ecuación de la recta, pendiente y término independiente, queda como ecuación final (Figura 50) de consumo de memoria RAM:

$$ConsumoMemoria(x,y) = \frac{((x-333)*y*0,154)+((y-1000)*51,5)}{1000} + 115$$

$$x = Num.individuos$$
$$y = Num.segundossim.$$
$$\forall x >= 333, y >= 1000$$

Figura 50

Tener en cuenta que para X más pequeñas que 333 y para Y más pequeñas que 1000 la función no es fiable, ya que en algunos puntos la función deja de ser lineal.
Los valores de memoria resultantes de la función estarán en MegaBytes.

6.1.3 Requisitos mínimos

Los requisitos mínimos para que un programador pueda mantener la aplicación debugándola (modo de compilación Debug) se requeriría de un mínimo de un procesador con doble núcleo a 2 GHz, 2GB de memoria RAM (limitando el número de individuos para las pruebas) y una tarjeta gráfica aceleradora de 256 MB de memoria a 1,2 GHz.
Para que un usuario de la aplicación la pueda ejecutar fluidamente (modo de compilación Release) requeriría de un mínimo de un procesador con doble núcleo a 2 GHz, memoria RAM calculada a partir de la función de cálculo memoria anterior y una tarjeta gráfica aceleradora con 8 MB de memoria a 533 MHz.

En ambos casos se debe de tener un disco duro con la suficiente capacidad (250 MB en un caso de una simulación de 1000 individuos y 1 hora y 20 minutos) para almacenar los datos resultantes de las simulaciones, siendo SATA si queremos acelerar la carga de los datos resultantes de la simulación.

6.2 Problemas encontrados

El primer problema encontrado fue el descuadre de la planificación inicial ya que por causas ajenas se tuvo que pausar el proyecto durante un periodo de tiempo. Finalmente se pudo retomar sin problemas desplazando en el tiempo todas las tareas restantes del proyecto.

La integración de CEGUI con Ogre tampoco fue sencilla ya que surgieron problemas con las librerías de Entrada/Salida al mezclar las dos API's. Finalmente se consiguió manejando los datos en los buffer de E/S de los dispositivos externos mas a bajo nivel y no automáticamente.

Otro de los problemas fue el orientar correctamente a los individuos ya que Ogre utiliza orientación de nodos por Quaterniones y no por vectores. Se tuvo que buscar cómo hacer el traslado de vector tridimensional a quaternion.

El problema más importante encontrado fue el de conseguir unos FPS adecuados y un tiempo de carga razonable para la visualización de la simulación. Para una representación de 5000 segundos la carga tardaba unos 20 minutos y con 1000 individuos la aplicación se renderizaba a unos 2 FPS lo que era totalmente inviable.
Finalmente se encontró el problema. La versión Debug añadía mucho código y testing interno adicional con lo que ralentizaba enormemente la ejecución del programa. Al generar la versión Release la representación de 5000 segundos pasó a cargarse en menos de un minuto y con 1000 individuos renderizó a 25 FPS obteniendo una mejora de un 1200% en la tarjeta Intel y a 63 FPS con una mejora de un 2200% en el caso de la tarjeta GeForce solucionando así el problema de rendimiento.

6.3 Posibles mejoras

En un futuro se podrían contemplar los siguientes aspectos con el objetivo de mejorar el software final.

- Integración del simulador dentro del visualizador, realizando los cálculos de las posiciones y direcciones en tiempo real (requeriría máquinas muy potentes en función del número de reglas que queremos introducir al simulador).

- Incorporación de cambios de escenario desde la aplicación.

- Mejora de los shaders del mar intentándole dar más realismo.

- Posibilidad de cargar solo las posiciones evitando las direcciones. De esta forma, se calcularían los vectores directores de los individuos mediante la resta de dos posiciones, evitando así la lectura de todos los ficheros de direcciones (los números están en punto flotante) acelerando el tiempo de carga a menos de la mitad.

- Posibilidad de aplicar shaders dinámicamente a los modelos.

- Creación de un compositor que, cuando la cámara esté sumergida en el agua, genere un desenfoque y enfoque aleatorio en todo el renderizado simulando la inmersión.

- Posibilidad de crear una animación por keyframing de la cámara en función del tiempo desde la aplicación. Esto provocará que la reproducción de la simulación sea más espectacular y que la cámara se mueva sola. Como ya se ha comentado, para hacer el keyframing se insertará, en el tiempo deseado, la posición y orientación de la cámara (KEY) para finalmente interpolarlas.

- Posibilidad de cambio de posición y tonalidad de las luces.

- Hilo ambiental de sonido.

6.4 Conclusiones

Finalmente, se puede decir que el resultado del proyecto ha cumplido satisfactoriamente los objetivos principales preestablecidos. La planificación inicial no se pudo cumplir estrictamente debido al paro del proyecto durante un periodo de tiempo, pero sí se ha cumplido la previsión de la duración de las tareas.

El diseño eficiente ha sido un punto clave a la hora de la implementación del software, permitiendo eficiencia y claridad en el software.

Se ha conseguido realizar un visualizador de simulaciones con un notable grado de realismo en su representación.

La interacción con el mundo tridimensional es absoluta, permitiendo total libertad de movimiento por éste y ofreciendo funcionalidades muy útiles para estudiar el comportamiento de los individuos.

Se han incluido dos secciones de personalización, una para la simulación y otra para los gráficos. En función de los parámetros seleccionados, aquí se puede acelerar la ejecución del programa en máquinas menos potentes y personalizarla según el tipo de agentes que se vayan a simular.

La interfaz de usuario integrada es intuitiva y de fácil manejo para cualquier usuario con nociones básicas de informática y videojuegos (debido a la manera de navegar por el mundo tridimensional).

A nivel de aprendizaje se ha conseguido:

- Análisis de datos ajenos al proyecto y utilización de los mismos en éste.
- Traspaso de requerimientos a un nivel de abstracción superior, implementando así un diseño eficiente, tanto de la aplicación como de las clases utilizadas en ella.

- Adquisición de amplios conocimientos en el campo de los gráficos por computador.
    - Nuevas técnicas gráficas.
    - Habilidades matemáticas 3D.
    - Uso de un motor gráfico 3D.
    - Programación de shaders CG.
    - Optimización de código en secciones iterativas acelerando así el renderizado. Por ejemplo: minimizar accesos a memoria en el interior de los bucles de renderización.
- Refuerzo del nivel del lenguaje de programación C++.
- Integración de distintas API's en una sola aplicación.
- Análisis, interpretación y valoración de los resultados finales.
- Elaboración de una función matemática creada a partir del análisis de los resultados finales, que permitirá estipular los requisitos de memoria que se necesitarán para visualizar los datos resultantes de cualquier simulación.

Finalmente, se puede decir que, el rendimiento de la aplicación es alto.
Con una tarjeta gráfica con poca memoria y poca velocidad de cómputo (Mobile Intel Graphics) alcanzamos los 25 FPS mínimos para un resultado fluido en una reproducción de simulación con 1000 individuos, dando finalmente por válidos los objetivos que nos habíamos marcado inicialmente.

7. Bibliografía

1. Gregory Junker, Pro OGRE 3D programming, Ed. Apress, 2006.
2. Jason Gregory, Game Engine Architecture, 2009
3. Ogre3D Open Source 3D Graphics Engine [14 Oct 2009] http://www.ogre3d.org/
4. Ogre Forums [15 Oct 2009] http://www.ogre3d.org/forums/
5. Joshbeam.com - Cg Pixel Shaders in OpenGL [1 Dic 2009] http://joshbeam.com/articles/cg_pixel_shaders_in_opengl/
6. Crazy Eddie's GUI System [15 Nov 2009] http://www.cegui.org.uk/wiki/index.php/Main_Page
7. CG References & Tutorials [2 Dic 2009] http://www.fundza.com/
8. Blender Home Page [12 Oct 2009] http://www.blender.org/
9. To Game or not to Game - Blender: exportar modelos a Ogre [13 Oct 2009] http://www.aserrano.com/2007/12/19/blender-exportar-modelos-a-ogre/
10. GameDev.net – all your game development needs [24 Nov 2009] http://www.gamedev.net/
11. Cplusplus.com - The C++ Resources Network [30 Oct 2009] http://www.cplusplus.com/
12. Foros 3D Poder [29 Oct 2009] http://www.foro3d.com/
13. Stratos – Punto de encuentro de desarrolladores [10 Nov 2009] http://www.stratos-ad.com/
14. Creación de videojuegos [14 Oct 2009] http://www2.ing.puc.cl/~iic3686/directx.html
15. GIMP- Alternativa libre a Photoshop [8 Oct 2009] http://www.gimp.org.es/

16. Martin b.r. – Quaterniones [12 Dic 2009]
http://www.martinbr.com/tutoriales/matematicas/quaterniones/
17. Intel - Mobile Intel® GM965 Express Chipset [15 Ene 2010]
http://www.intel.com/products/notebook/chipsets/GM965/GM965-overview.htm
18. Tocan – Virtual museum [4 Ene 2010]
http://toucan.web.infoseek.co.jp/3DCG/3ds/FishModelsE.html
19. TeleWatcher –Role of Television in Character Formation [15 Ene 2010]
http://telewatcher.com/telewatching/role-of-television-in-character-formation/
20. Introduction to 3D Graphics [17 Ene 2010]
http://www.cs.iusb.edu/~danav/teach/c481/intro_3d.html
21. Ingeniería de los sistemas basados en el conocimiento. [12 Ene 2010]
http://iaaa.cps.unizar.es/docencia/ISBC.html
22. SIMED – Desarrollo de software médico [11 Ene 2010]
http://www.simed.es/
23. Sinciforma – Proyecto Siveace [11 Ene 2010]
http://www.sinciforma.com/sinciformacomes/html/siveace/index.php
24. El Pais – Física, Estudio de la atmósfera [11 Ene 2010]
http://ic.daad.de/barcelona/download/simulador-nubes.pdf
25. Red Científica- Metodologías de la IA [7 Ene 2010]
http://www.redcientifica.com/doc/doc200401210112.html
26. UPV - Computación de altas prestaciones sobre entornos grid en aplicaciones biomédicas [11 Ene 2010]
http://dspace.upv.es/xmlui/handle/10251/1831
27. OpenGL – OpenGL Overview [19 Ene 2010]
http://www.opengl.org/about/overview/
28. ActiveNetwork – Microsoft DirectX [19 Ene 2010]
http://www.activewin.com/faq/faq_7.shtml
29. SONY – Relase of PHYREENGINE [120 Ene 2010]
http://www.scee.presscentre.com/Content/Detail.asp?ReleaseID=4762&NewsAreaID=2

30. CrystalSpace [20 Ene 2010]
http://www.crystalspace3d.org/main/Main_Page
31. Análisis de requerimientos – Ing. Luis Zuloaga Rotta [18 Ene 2010]
http://www.galeon.com/zuloaga/Doc/AnalisisRequer.pdf
32. Spatial Keyframing for Performance – driven Animation [18 Ene 2010]
http://www-ui.is.s.u-tokyo.ac.jp/~takeo/research/squirrel/index.html
33. Soporte Microsoft – Descripción de marcos por segundo (FPS) [14 Ene 2010]
http://support.microsoft.com/kb/269068/es
34. nVIDIA – GeForce 8600 GT [15 Ene 2010]
http://www.nvidia.es/object/geforce_8600_es.html
35. Vitutor – Ecuaciones de la recta [22
36. Ene 2010]
http://www.vitutor.net/1/2.html

8. Anexos

8.1 Anexo 1 - Manual de usuario

8.1.1 Instalación del visualizador de simulaciones

La instalación de la aplicación es muy sencilla. Se ha creado un instalador que automáticamente ubica todos los ficheros necesarios en su sitio. Se hará doble clic sobre "setup.exe" para iniciar la instalación (ver Figura 51):

Figura 51

Una vez ejecutado, se deben de seguir los pasos de la Figura 52:

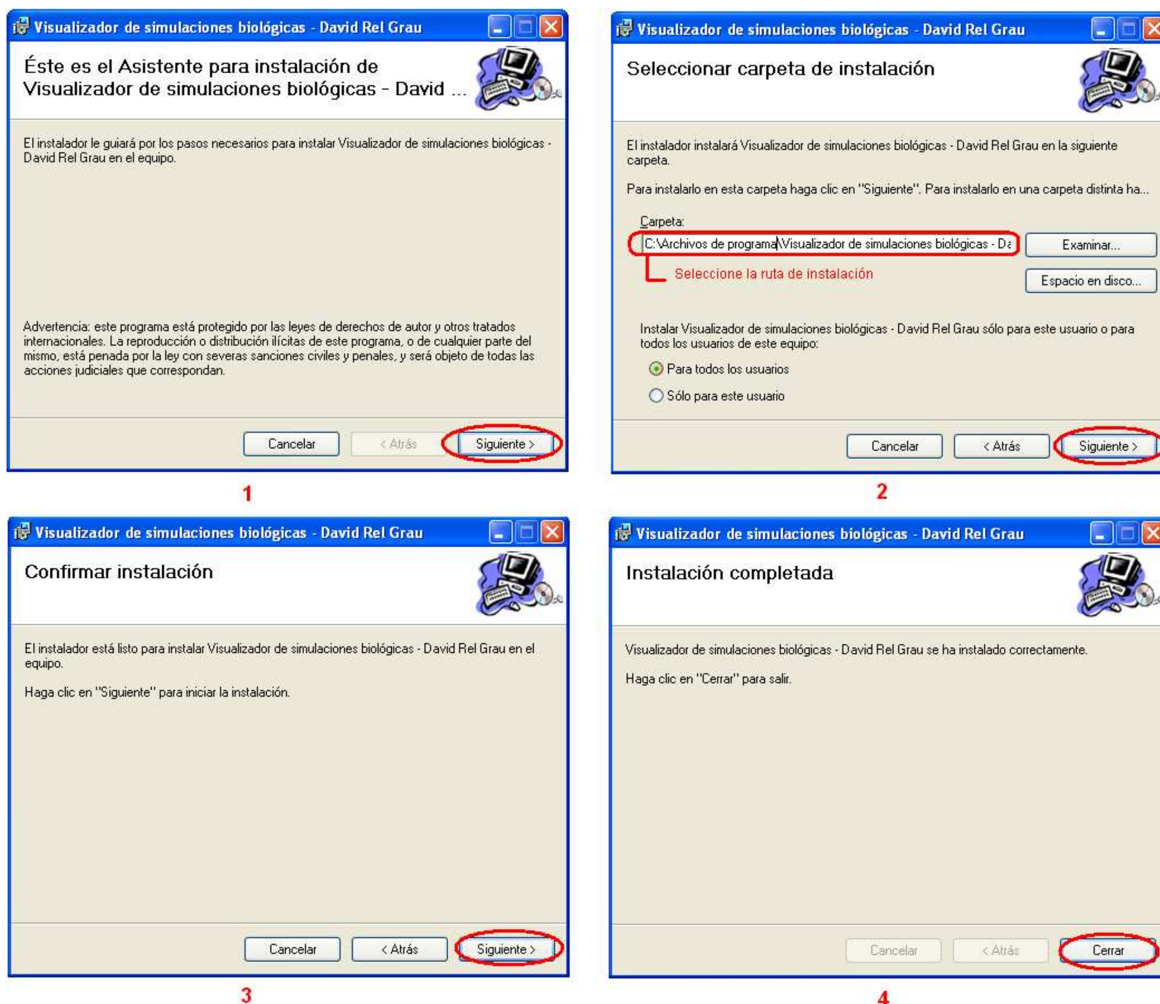


Figura 52

Hecho esto, ya se puede ejecutar la aplicación desde el escritorio, menú inicio o desde el directorio donde se haya instalado.

IMPORTANTE:

Se requiere tener instalado el Service Pack 2 en el caso de que se tenga instalado el Windows XP.

También se requiere tener instalado DirectX. En el caso de que no lo esté se puede descargar desde el siguiente enlace:

http://download.microsoft.com/download/1/7/1/1718CCC4-6315-4D8E-9543-8E28A4E18C4C/dxwebsetup.exe

8.1.2 Configuración del visualizador de simulaciones

8.1.2.1 Incorporar los ficheros de salida de las simulaciones

Para poder visualizar las simulaciones, primero se deben de incorporar los ficheros que han generado. La manera de incorporarlos para que la aplicación los detecte, será añadirlos en la carpeta llamada "data" en el directorio de la aplicación (ver Figura 53).

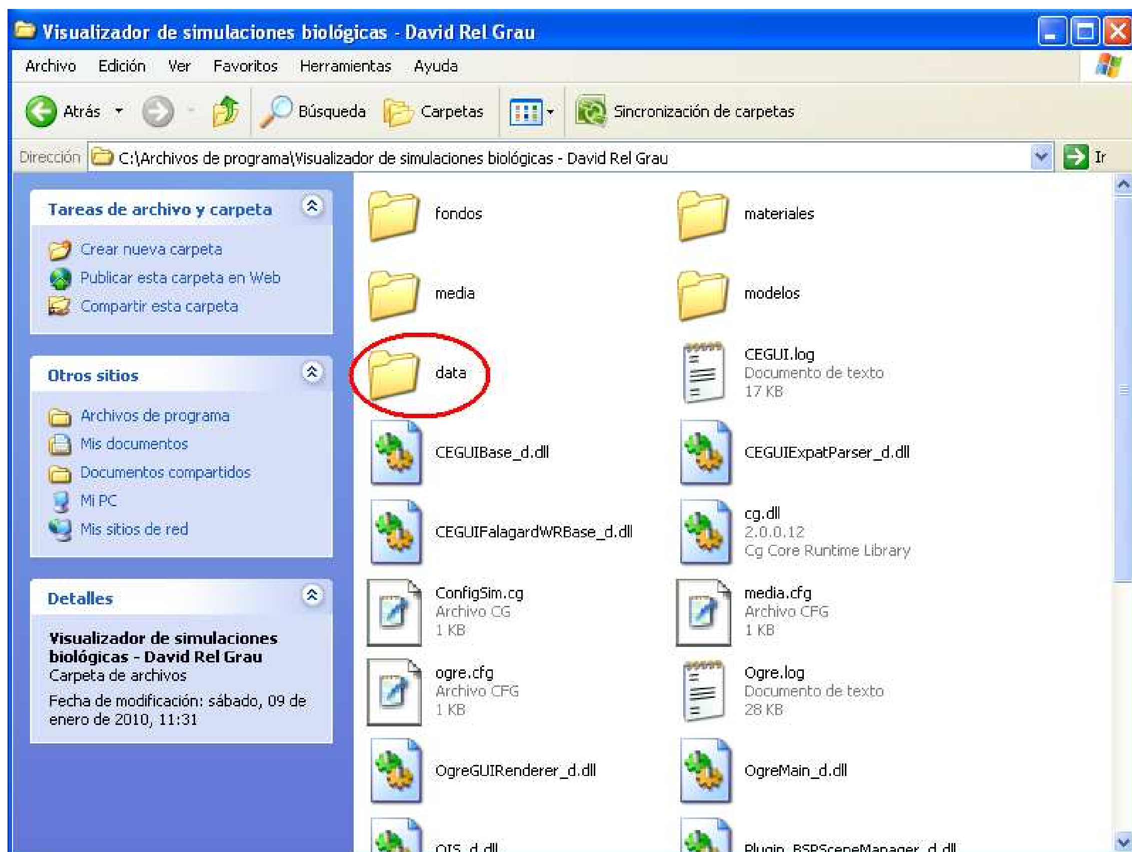


Figura 53

Dentro de esta carpeta están todos los datos de las simulaciones almacenadas en directorios distintos, uno por cada resultado de la simulación (ver Figura 54).

Figura 54

Así, la aplicación leerá automáticamente estos directorios pudiendo seleccionar a posteriori la simulación que interese visualizar (Figura 55).

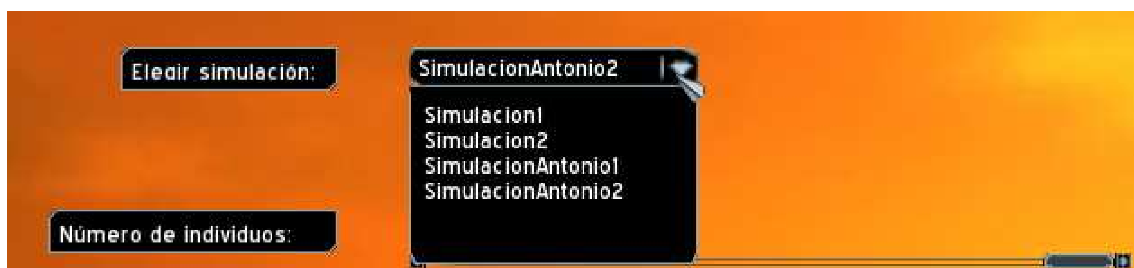


Figura 55

NOTA: Los nombres de los directorios que contienen los datos de las simulaciones no deben de contener espacios en blanco.

8.1.2.2 Primera ejecución

La primera vez que se ejecuta la aplicación aparecerá el panel de la Figura 56.

Figura 56

Esto es porque todavía no se ha autodetectado la tarjeta gráfica. Se hará clic sobre el desplegable y se seleccionará como motor de renderizado "Direct3D" (Figura 57).

Figura 57

Al seleccionar, aparecerá nuestra tarjeta gráfica y un conjunto de parámetros para el motor de renderizado que se podrán cambiar cada vez que se ejecute la aplicación. Una vez se tengan los parámetros deseados se pulsará el botón "OK", ejecutándose así la aplicación (Figura 58).

Figura 58

8.1.3 Manejo de la aplicación

8.1.3.1 Menú principal:

Desde el menú principal se pueden tomar 4 acciones básicas (ver Figura 59).

Figura 59

8.1.3.2 Configuración visualización:

Existen 3 posibles opciones de configuración (ver Figura 60).

Figura 60

8.1.3.3 Configuración gráficos:

Aparecen 6 posibles opciones de configuración (ver Figura 61).

Figura 61

8.1.3.4 GUI entorno de visualización:

Aquí habrá varias funcionalidades que se explicarán a continuación (Figura 62).

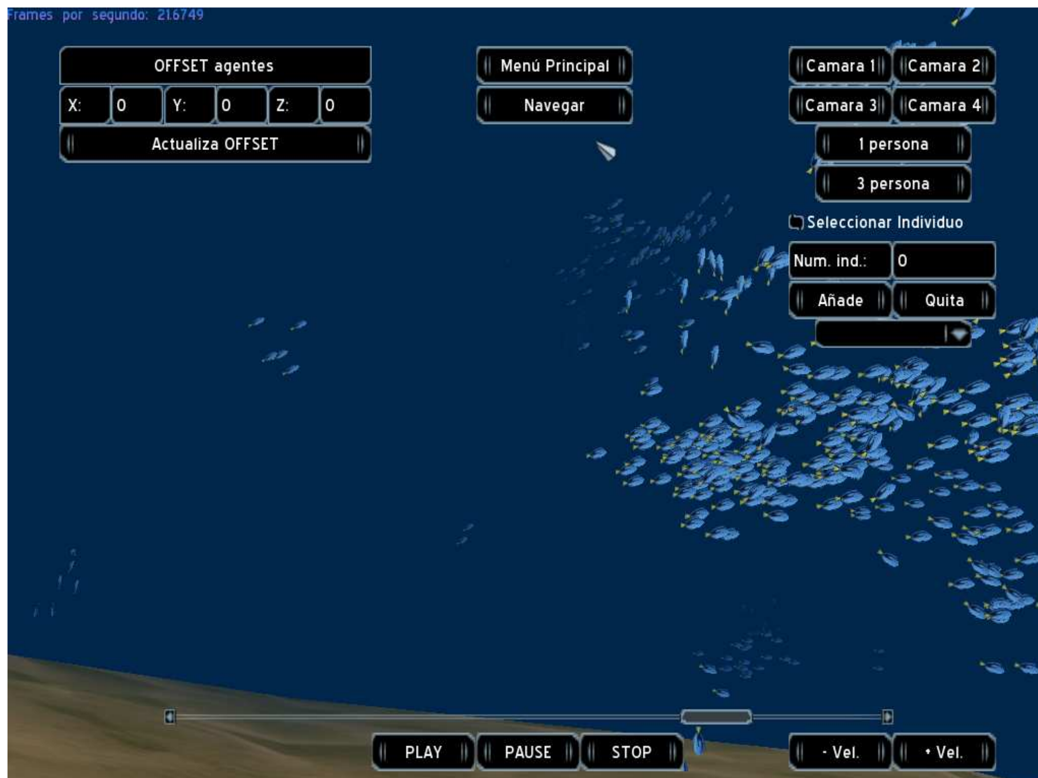


Figura 62

- Menú Principal
  - Al pulsar este botón se vuelve al menú principal.
- Navegar
  - Este botón servirá para cambiar al modo navegación. Esto significa que se obtendrá el control de la cámara y se podrá cambiar su posición y orientación mediante el teclado y ratón, pudiendo mover la cámara libremente por el mundo (en el caso de que no estén los botones 1ª persona y 3ª persona activados).
  - Para volver al modo GUI (al mover el ratón se mueve el cursor y se puede interactuar con los botones) habrá que pulsar el botón F1 o simplemente un clic con el botón izquierdo del ratón.

- OFFSET agentes
  - Este botón servirá para desplazar el grupo de individuos mediante un vector de desplazamiento (ver Figura 63).

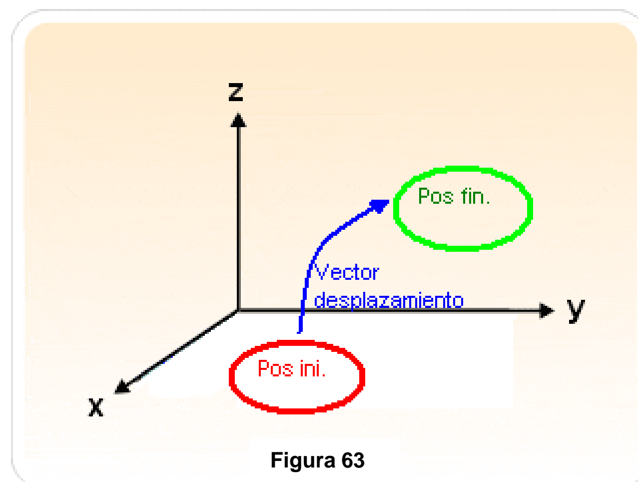


Figura 63

El vector desplazamiento se introducirá en los 3 cuadros de texto <X, Y, Z> mostrados en la Figura 64.

OFFSET agentes
X: 0 Y: 0 Z: 0

Figura 64

Si se quiere trasladar -10 unidades en X y 200 en Y se introducirá tal y como se muestra en la Figura 65.

Figura 65

Una vez está el vector desplazamiento introducido se pulsará el botón “Actualiza OFFSET” (Figura 66) para aplicar el vector al grupo de individuos observando que éstos han cambiado de posición.

Figura 66

- Cámara N
  - Existen 4 cámaras que se podrán utilizar para observar la escena desde 4 puntos de vista diferentes si así se prefiere. Pulsando estos botones se cambiará de cámara, modificando siempre la posición de la que esté seleccionada.
- 1 Persona
  - Pulsando este botón la vista se ubicará encima del individuo como si se estuviese viendo el mundo desde sus ojos. (Si se está en modo navegación no se podrá mover la cámara, ya que su posición y orientación será la del individuo).
- 3 Persona
  - Pulsando este botón la cámara se orientará hacia el individuo y lo seguirá, pero su posición se podrá cambiar desde el modo Navegación. Esto sirve para realizar el seguimiento de un individuo en concreto.
- Seleccionar Individuo
  - Cuando pulsemos este seleccionable se cambiará a modo de selección de individuo. Esto servirá para, en modo navegación, seleccionar el individuo libremente por el mundo, envolviendo a éste en una caja transparente. Una vez enmarcado, se pulsará el botón izquierdo del ratón para seleccionarlo definitivamente. Si se observa en el texto “Núm. Ind.” aparecerá en todo momento el

número del individuo que está siendo seleccionado seleccionando.

- Num. Ind.
    - Otra manera de seleccionar el individuo al que se quiere seguir es escribiendo directamente el número de éste en el cuadro de texto.
- Desplegable de individuos
    - Este combobox contendrá todos los identificadores de individuos que queramos marcar con una caja blanca transparente (ver Figura 67). Esta funcionalidad servirá para realizar un seguimiento de un conjunto de individuos. Bajo previa selección de un individuo, pulsaremos el botón “Añadir” para incorporarlo a la lista. Si lo seleccionamos de la lista y pulsamos “Quita”, éste se eliminará.

      Otra manera de añadirlos o quitarlos es activando la selección de individuo y pulsando con el ratón derecho sobre el individuo a seleccionar. Si no estaba seleccionado se añadirá y si lo estaba se quitará de la lista.

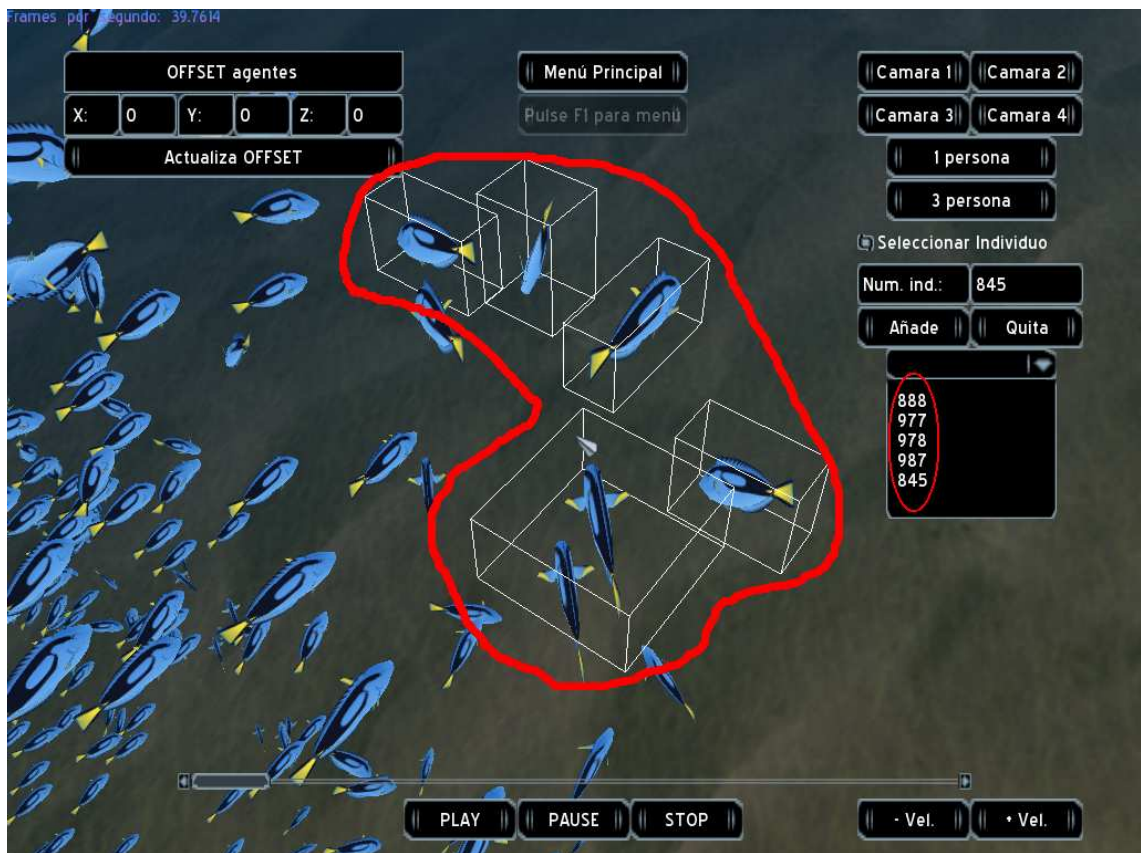


Figura 67

- Menú multimedia
  - Este menú tiene la misma funcionalidad que la de un reproductor de video o música estándar. Se podrá reproducir, pausar y parar la reproducción. De igual manera, se podrá mover la barra de desplazamiento para moverse por el tiempo de la reproducción a conveniencia del usuario.

8.1.4 "Shortcuts" o atajos con el teclado

Hay una serie de teclas que ayudan a visualizar la simulación con más fluidez y comodidad. Estas teclas SOLO están disponibles en modo Navegación, es decir, cuando el usuario se está moviendo por el mundo con el ratón y el teclado. En la Figura 68 se muestran cuales son y para qué sirven:

Tecla	Funcionalidad
W	Avanzar cámara
S	Retroceder cámara
A	Mover cámara a la izquierda
D	Mover cámara a la derecha
←	Rotar cámara a la izquierda
→	Rotar cámara a la derecha
RePág	Subir cámara
AvPág	Bajar cámara
F1	Cambio al modo GUI (en el caso de que se esté en modo Navegación)
Esc o Q	Salir de la aplicación
Impr pant	Crea, en el directorio donde está el ejecutable, los screenshots realizados en formato .png
R	Cambia el modo de visualización de la escena a Texturizado, Alambre o Puntos.
Control	Para la representación de la escena y vuelve al tiempo 0.
Espacio	En caso de que la representación esté parada o pausa la pone en marcha. Si está en marcha la pausa.
+	Incrementa la velocidad en 1
-	Decrementa la velocidad en 1

Figura 68

8.2 Anexo 2 - Explicación detallada de las clases

8.2.1 Invididuo:

- mesh: *Entity
    - La entidad, es decir el modelo o malla con sus materiales y animaciones.
- nodo: SceneNode
    - El nodo al cual se le asigna la entidad marcando una posición, escala y dirección en el mundo.
- Individuo(const Ogre: :String nombre, const Ogre: :String archivo, Vector3 posicion, Vector3 direccion)
    - Constructor sobrecargado que instancia los dos objetos anteriores con un nombre, el nombre del fichero donde está la malla (*.mesh) y se le asigna una posición y una dirección iniciales.
- Individuo()
    - Constructor por defecto de la clase
- ~Individuo()
    - Destructor por defecto de la clase. Destruye las instancias de los dos objetos anteriores.
- IniObj(const Ogre: :String nombre, const Ogre: :String archivo, Vector3 posicion, Vector3 direccion)
    - Inicializa e instancia los dos objetos anteriores con un nombre, el nombre del fichero donde está la malla (*.mesh) y se le asigna una posición y una dirección iniciales.

- SetPosicion(Vector3 vector)
    - Asigna una posición al nodo
- Vector3 GetPosicion()
    - Devuelve la posición del nodo
- SetDireccion(Vector3 vector)
    - Asigna una dirección al nodo
- Vector3 GetDireccion()
    - Retorna la dirección del nodo

- Rotar(float angulo)
    - Rota el nodo un determinado ángulo con el eje Y fijo

8.2.2 BaseManager:

- mRoot: *Root
    - Objeto root o raíz de Ogre
- mSceneMgr: *SceneManager
    - Objeto manipulador de la escena o SceneManager de Ogre
- mRenderer: *OgreCEGUIRenderer
    - Objeto renderizador de la GUI para Ogre de CEGUI
- mFrameListener: *FrameListener
    - Objeto que gestiona y controla el lazo del renderizado o FrameListener de Ogre.
- Static BaseManager *mBaseManager
    - Objeto Singleton de la propia clase
- BaseManager *GetInstancia()
    - Si el objeto ya se ha instanciado una vez retorna el que había y si no lo crea.
- BaseManager()
    - Inicializa los objetos de la clase

8.2.3 EscenaManager:

- limite: int
    - Número deseado de individuos a visualizar.
- num_ind: int
    - Número máximo de individuos en los archivos de simulación.
- Posicion: *Vector3
    - Array de Vectores(x,y,z) de posiciones que se generará dinámicamente en función del número de individuos a visualizar.
- velocidad: *Vector3
    - Array de Vectores(x,y,z) de direcciones que se generará dinámicamente en función del número de individuos a visualizar.

- mAnimationState: *AnimationState
    - Estado de la animación del Mesh de cada individuo en caso de que tengan.
- Individuo *peces
    - Array de individuos que se generará dinámicamente en función del número de individuos a visualizar.
- EscenaManager()
    - Constructor por defecto de la clase
- CalculaTiempoMaxSimulacion()
    - Hace la lectura de todos los ficheros de posiciones de la simulación incrementando un contador hasta que ya no existan. Este contador será el número de segundos de la simulación que se guardará en la máquina de estados en mTiempoMax.
- CargarSimulacion()
    - Lee de la máquina de estados el número máximo de individuos y el número que deseamos visualizar. También calcula el tiempo máximo de simulación. Hecho esto establecemos las posiciones y direcciones iniciales de los nodos de los individuos. A continuación hacemos la lectura del resto de todos los ficheros de posiciones y direcciones creando la animación por keyframing mediante interpolaciones. Para acabar activamos las animaciones de los Mesh (en el caso de que tengan) y asignamos a los materiales de los individuos los colores guardados en la máquina de estados.
- CrearEscena()
    - Inicialmente creamos las luces de la escena y asignamos el cielo que tenemos guardado en la máquina de estados. Siguientemente creamos el mar, el escenario y el individuo que se visualizará en la pequeña ventana del menú Configuración de Gráficos.

- CalculaNumeroMaxIndividuos()
  - Leemos el primer fichero de posiciones y leemos línea a línea realizando un recuento de éstas. Una vez se ha llegado al final del fichero guardamos este recuento en el atributo mNumeroMaxIndividuos de la máquina de estados.

- LecturaPosicionesFichero()
  - Creamos el Array de posiciones en función del número máximo de individuos y hacemos la correspondiente lectura del primer fichero de posiciones (posiciones iniciales). Aquí creamos el array de individuos y lo inicializamos por cada línea del fichero con la posición debida. Finalmente creamos la animación general por keyframing y le asignamos el primer KeyFrame para tiempo=0 con la posición correspondiente de cada individuo.
- LecturaVelocidadesFichero()
  - Creamos el Array de direcciones en función del número máximo de individuos y hacemos la correspondiente lectura del primer fichero de direcciones (direcciones iniciales). Recuperaremos el KeyFrame para tiempo=0 y le asignaremos la dirección correspondiente de cada individuo.
- CrearInterpolaciones(int t)
  - Realiza la lectura de las posiciones y direcciones para un tiempo T determinado (para cada fichero). Hecho esto va creando un KeyFrame en el tiempo T con la posición y dirección de cada individuo. Finalmente activa la animación general por cada uno de los individuos realizando la interpolación de cada uno de los keyframes de éstos.
- BorrarSimulacionActual()
  - Destruye todas las entidades (meshes) y nodos de los individuos, borrando también sus animaciones.

- Porcadaframe(const FrameEvent &evt)
    - Esta función se llamará cada vez antes de renderizar un frame. Calcularemos el tiempo a añadir a las animaciones en función de la velocidad que tengamos en la máquina de estados.
      Si en ésta tenemos el flag de Play activado añadiremos este tiempo previamente antes calculado a la animación general de cada individuo y a la de la propio mesh, posteriormente actualizaremos la posición del scrollbar del menú multimedia.
      Si el flag de la máquina de estados está en Stop nos iremos al tiempo 0 de la animación y pondremos la posición del scrollbar a 0. Finalmente llamaremos a la función de control para saber si estamos dentro del mar o no.
- ControlNieblaAgua()
    - Si estamos por encima del mar en coordenada Y desactivaremos la niebla. En el caso de que estemos por debajo activaremos la niebla azul y en función de si la cámara está enfocando hacia arriba o hacia abajo la intensidad de la niebla será distinta.

8.2.4 MenuManager:

- mMouse: *Mouse
    - Objeto que gestiona y controla la E/S del ratón
- mKeyboard: *Keyboard
    - Objeto que gestiona y controla la E/S del teclado
- RutinaTratamientoEventos()
    - En esta función relacionaremos los eventos que nos interesen de todos los objetos CEGUI creados con una rutina de atención al evento.

- bool ActualizaOffset(const CEGUI: :EventArgs& e)
    - Activamos el flag de actualización de offset de la máquina de estados y llamamos a ActualizaOFFSETAnimacion()

- ActualizaOFFSETAnimacion()
    - Esta función atiende al evento clic del botón Actualizar Offset. Aquí leemos todos los keyframes de las animaciones de los individuos y le aplicamos el desplazamiento que tengamos en los cuadros de texto <X, Y, Z>.
- bool MenuConfigSimulacionGuardar(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Guardar del menú de configuración de simulación. Actualizamos todos los flags de la máquina de estados relacionados este menú. Posteriormente realizamos la escritura al fichero de configuración.
- bool MenuConfigSimulacionCombo(const CEGUI: :EventArgs& e)
    - Función que atiende al evento de cambio de valor del combobox donde seleccionamos la simulación que queremos. Calculamos el número de individuos de la simulación seleccionada y actualizamos el valor del scrollbar de número de individuos con éste valor.
- bool MenuPrincipalIniciar(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Iniciar visualización del menú principal. Lo único que hace es activar el flag de carga de la visualización de la máquina de estados.
- bool GuiSimulacionOpciones(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Menú principal del menú de la GUI. Esto hace invisible el menú de la GUI y activa el Menú principal.
- bool GuiSimulacionModo(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Navegar del menú de la GUI. Actualizamos dicho botón con el texto "Pulse F1 para menú" , desactivamos el buffering de los objetos del ratón y el teclado, ubicamos el cursor en el centro de la pantalla e indicamos que se haga una lectura del ratón y el teclado.

- bool GuiSimulacionCamara1(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Camara1 del menú de la GUI. Actualizamos el valor de cámara a 1 de la máquina de estados y a continuación activamos el flag de cambio de cámara.
- bool GuiSimulacionCamara2(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Camara2 del menú de la GUI. Actualizamos el valor de cámara a 2 de la máquina de estados y a continuación activamos el flag de cambio de cámara.
- bool GuiSimulacionCamara3(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Camara3 del menú de la GUI. Actualizamos el valor de cámara a 3 de la máquina de estados y a continuación activamos el flag de cambio de cámara.
- bool GuiSimulacionCamara4(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Camara4 del menú de la GUI. Actualizamos el valor de cámara a 4 de la máquina de estados y a continuación activamos el flag de cambio de cámara.
- bool GuiSimulacionPlay(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Play del menú de la GUI. Activamos el flag Play de la máquina de estados.
- bool GuiSimulacionStop(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Stop del menú de la GUI. Activamos el flag Stop de la máquina de estados.
- bool GuiSimulacionPause(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Pause del menú de la GUI. Activamos el flag Pause de la máquina de estados.
- bool GuiSimulacionprimeraP(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón 1 Persona. Activamos el flag de 1 persona (mSeguir) y desactivamos el de 3 Persona (mMirar) si el primero no estaba activado. En el caso de que estuviese activado, lo desactivaremos.

- bool GuiSimulacionterceraP(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón 3 Persona. Activamos el flag de 3 persona (mMirar) y desactivamos el de 1 Persona (mSeguir) si el primero no estaba activado. En el caso de que estuviese activado, lo desactivaremos.
- bool MenuConfigGraficosColor(const CEGUI: :EventArgs& e)
    - Función que atiende al evento de cambio de posición de alguno de los 3 scrolls de color del menú de configuración de gráficos. Haremos la lectura del % de los tres colores (RGB) filtrándoselos al material del mesh que este actualmente asignado.
- bool MenuConfigGraficosComboFondo(const CEGUI: :EventArgs& e)
    - Función que atiende al evento de cambio de valor del combobox para cambiar el fondo del Menu de configuración de gráficos. Aquí se leerá el valor del combo y se asignará el cielo elegido a la escena.
- bool MenuPrincipalConfigGraficosGuardar(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Guardar del menú de configuración de gráficos. Actualizamos todos los flags de la máquina de estados relacionados este menú. Posteriormente realizamos la escritura al fichero de configuración.
- bool MenuPrincipalConfigGraficosModelo(const CEGUI: :EventArgs& e)
    - Función que atiende al evento de cambio de valor del combobox para cambiar el modelo del menú de configuración de gráficos. Cargamos el modelo escogido, miramos si tiene animaciones esqueletales y se las cargamos en el combobox de animaciones. Finalmente asignamos el % de color RGB que tengamos al material del nuevo modelo.
- bool MenuPrincipalConfigSimulacion(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Configuración visualización del Menú principal. Siguientemente mostraremos dicho menú.

- bool MenuPrincipalConfigGraficos(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Configuración gráficos del Menú principal. Siguientemente mostraremos dicho menú.

- bool MenuPrincipalSalir(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Salir del Menú principal. Esto activará el flag Salir de la máquina de estados.
- bool MenuPrincipalVolver(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del botón Volver de los Menús de configuración. Esto activa otra vez el Menú principal.
- bool MenuConfigSimulacionScroll(const CEGUI: :EventArgs& e)
    - Función que atiende al evento de cambio de posición del scroll Velocidad del Menú de configuración de la simulación. Esto actualizará el texto del cuadro de texto con la velocidad seleccionada con dicho scroll.
- bool MenuConfigSimulacionScrollIndividuos(const CEGUI: :EventArgs& e)
    - Función que atiende al evento de cambio de posición del scroll Número de individuos del Menú de configuración de la simulación. Esto actualizará el texto del cuadro de texto con el número de individuos seleccionados con dicho scroll.
- bool GuiSimulacionScroll(const CEGUI: :EventArgs& e)
    - Función que atiende al evento de cambio de posición del scroll línea del tiempo del Menu GUI. Captura el % del tiempo total (valor del scroll del tiempo) y se lo asigna al miembro mTiempo de la máquina de estados activando siguientemente el flag de cambio de tiempo.
- ActivarMenu(CEGUI: :Window *window)
    - Muestra u oculta un determinado menú que pasamos a la función.
- CrearGUISimulacion()
    - Función que crea todos los controles o elementos del Menú GUI asignándoles un nombre y las propiedades que deseamos.

- CrearMenuPrincipal()
    - Función que crea todos los controles o elementos del Menú Principal asignándoles un nombre y las propiedades que deseamos.

- CrearMenuConfigSimulacion()
    - Función que crea todos los controles o elementos del Menú de configuración de visualización asignándoles un nombre y las propiedades que deseamos.
- CrearLabelInformativa()
    - Función que crea el panel de “Cargando….”
- CrearMenuConfigGraficos()
    - Función que crea todos los controles o elementos del Menú de configuración de gráficos asignándoles un nombre y las propiedades que deseamos.
- CalculaNumeroMaxIndividuos(String Simulacion)
    - Leemos el primer fichero de posiciones y leemos línea a línea realizando un recuento de éstas. Una vez se ha llegado al final del fichero guardamos este recuento en el atributo mNumeroMaxIndividuos de la máquina de estados.
- bool VerCaja(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del checkbox de selección de individuo del menú GUI. En caso de que el checkbox esté activado se activa el modo navegación y el flag de verCaja de la máquina de estados para que al pasar el cursor por encima de un individuo se vea su caja envolvente.
- bool VerMar(const CEGUI: :EventArgs& e)
    - Función que atiende al evento clic del checkbox de verMar del Menú configuración de gráficos. En el caso de que esté activado mostrará el mar y en caso contrario no lo mostrará.

8.2.5 RootSingleton:

- static RootSingleton *mRootSingleton
    - Objeto Singleton de la propia clase.
- EscenaManager *EscenaMgr
    - Objeto manager de la escena, donde se controlará toda la escena.
- MenuManager *MenuMgr
    - Objeto manager de los menús, donde se controlarán todos los menús.
- RootSingleton()
    - Instancia los objetos de la clase.
- RootSingleton *GetInstancia()
    - Si el objeto ya se ha instanciado una vez retorna el que había y si no lo crea.

8.2.6 MáquinaEstados:

- static MáquinaEstados *ME
    - Objeto Singleton de la propia clase.
- mTiempoMax: Real
    - Tiempo máximo de la reproducción de la simulación.
- mTiempo: Real
    - Tiempo transcurrido de la reproducción de la simulación.
- mCambioTiempo: bool
    - Flag que indica si ha habido un cambio de tiempo o no.
- mVelocidad: Real
    - Velocidad de la reproducción de la simulación.
- mFicheroSimulacion: String
    - Nombre de la carpeta en la que se ubican los ficheros de salida de la simulación a reproducir.
- mNumeroIndividuos: Real
    - Número de individuos que se mostrarán en la reproducción de la simulación.

- mNumeroMaxIndividuos: Real
    - Número máximo de individuos que hay en los ficheros de salida de la simulación a reproducir.
- mVerMar: bool
    - Flag que indicará si se visualizará el mar o no.
- mOffset: Vector3
    - Desplazamiento absoluto que tendrán todos los individuos en la reproducción de la simulación.
- mEscala: Real
    - Escalado del modelo.
- mMesh: String
    - Nombre del fichero del modelo.
- mAnimacion: String
    - Animación del modelo actual.
- mRed: Real
    - % de rojo que se mostrará en el material del modelo actual.
- mGreen: Real
    - % de verde que se mostrará en el material del modelo actual.
- mBlue: Real
    - % de azul que se mostrará en el material del modelo actual.
- mCielo: String
    - Nombre del material del cielo a mostrar en la escena.
- mCambioOffset: bool
    - Flag que indica si ha habido un cambio de offset.
- mCamara: int
    - Número de cámara activa en la escena.
- mCambioCamara: bool
    - Flag que indica si ha habido un cambio de cámara.
- mMultimedia: int
    - Objeto que representa si la visualización está en reproducción (1), pausada (2) o parada (0).
- mSeguir: bool
    - Flag que indica si la 1 persona esta activada.

- mMirar: bool
    - Flag que indica si la 3 persona esta activada.
- mRatonPulsado: bool
    - Flag que indica si se ha pulsado el botón izquierdo del mouse (para la selección del individuo).
- mVerCaja: bool
    - Flag que indica si hemos activado el modo de selección de individuo.
- mCargar: int
    - Flag que indica el estado de la carga de la visualización de simulación: -1=sin cargar; 0= muestra "Cargando…" ; 1= Carga simulación ; 2= simulación cargada.

- mSalir: bool
    - Flag que indica si salimos de la aplicación o no.

- mIndSelect: bool
    - Objeto que contiene el número de individuo seleccionado.

- MáquinaEstados *GetInstancia()
    - Si el objeto ya se ha instanciado una vez retorna el que había y si no lo crea.
- MáquinaEstados()
    - Inicializa los objetos de la clase.
- GuardarConfiguracion()
    - Escribe en el fichero "ConfigSim.cg" los miembros de la clase que interesarán para no perder la configuración personalizada de la aplicación.
- CargarConfiguracion()
    - Leemos del fichero "ConfigSim.cg" los miembros de la clase que interesaran para no perder la configuración personalizada de la aplicación, guardándolos en dicha máquina de estados.

8.2.7 MiFrameListener:

- mCamera: *Camera
  - Objeto que representa la cámara activa en la escena.
- mWindow: *RenderWindow
  - Objeto que representa la ventana de renderizado.
- mTimeUntilNextToggle: Real
  - Tiempo que esperará hasta que se pueda realizar la lectura de otra tecla.
- mTimeUntilNextToggleMouse: Real
  - Tiempo que esperará hasta que se pueda realizar la lectura de clic de botón derecho de ratón.
- mRotX: Radian
  - Rotación de la cámara en X hecha con el mouse.
- mRotY: Radian
  - Rotación de la cámara en Y hecha con el mouse.
- mMoveSpeed: Real
  - Velocidad de desplazamiento de la cámara.
- mRotateSpeed: Degree
  - Velocidad de rotación de la cámara.
- mDebugOverlay: *Overlay
  - Overlay o capa donde se muestran los frames por segundo.
- mInputManager: *InputManager
  - Objeto que representa el Manager de E/S
- mMouse: *Mouse
  - Objeto que representa el ratón
- mKeyboard: *Keyboard
  - Objeto que representa el teclado

- MiFrameListener()
    - Es el constructor de la clase. Coloca en su sitio el título y los frames por segundo. Crea e inicializa el InputManager así como el teclado y el mouse. Crea todas las rutinas de tratamiento de eventos de CEGUI y crea el rayo que se lanza en el eje de las Z para ver si el cursor se ha cruzado con algún objeto (selección de individuos).
- ~MiFrameListener()
    - Llama a la función windowClosed explicada más abajo.
- updateStats()
    - Actualiza el overlay de Frames por segundo.
- windowResized(RenderWindow* rw)
    - Actualiza la posición del ratón en caso de que la ventana se haya redimensionado.
- void windowClosed(RenderWindow* rw)
    - Destruye el InputManager así como los objetos que utiliza.
- MouseButton convertButton(MouseButtonID buttonID)
    - Transforma el botón capturado del ratón a una enumeración más inteligible.
- bool mouseMoved(const MouseEvent &arg)
    - Función que se llama cuando el ratón se ha movido pasándole el movimiento a la API CEGUI.
- bool mousePressed(const MouseEvent &arg, MouseButtonID id)
    - Función que se llama cuando se ha pulsado un botón del ratón pasándole el ID del botón a la API CEGUI.
- bool mouseReleased(const MouseEvent &arg, MouseButtonID id)
    - Función que se llama cuando se ha soltado un botón del ratón pasándole el ID del botón a la API CEGUI.
- bool keyPressed(const KeyEvent &arg)
    - Función que se llama cuando se ha pulsado un botón del teclado pasándole el ID del botón a la API CEGUI.

- bool keyReleased(const KeyEvent &arg)
    - Función que se llama cuando se ha soltado un botón del teclado pasándole el ID del botón a la API CEGUI.
- bool processUnbufferedKeyInput(const FrameEvent& evt)
    - Función que gestiona qué tecla se ha pulsado del ratón y en función de cuál sea qué hacemos en cada caso.
- bool processUnbufferedMouseInput(const FrameEvent& evt)
    - Función que gestiona qué evento de ratón ha habido y qué hacemos en cada caso.
- moveCamera()
    - Realiza el movimiento de la cámara en función de los movimientos guardados de mouse y teclado.
- showDebugOverlay(bool show)
    - Función que muestra los FPS o no en función de la variable booleana que le pasamos.
- bool frameStarted(const FrameEvent& evt)
    - Función que se ejecuta antes de renderizar cada frame. Hace la lectura del teclado y el ratón y llama a las funciones que gestionan la E/S comentadas anteriormente. A continuación comprobamos si se ha cargado o no la visualización y actuamos en función de esto. Si se va a cargar la simulación borramos la actual y hacemos la carga. Si ya está cargada gestionamos los cambios de cámara, los cambios de persona, la visualización de la caja, el cambio de offset y si se ha pulsado salir o no. Finalmente llamamos a Por CadaFrame del EscenaManager que gestionará toda la actualización de las animaciones.

8.2.8 Simulacion3DApp:

- MiFrameListener* mFrameListener
    - FrameListener o clase que gestiona y controla el lazo de renderizado (Hereda del framelistener de Ogre)
- mRoot: *Root
    - Objeto raíz o Root de Ogre.

- mCamera: *Camera
    - Cámara actual.
- mSceneMgr: *SceneManager
    - Objeto manager de la escena, donde se controlará toda la escena.
- mWindow: *RenderWindow
    - Objeto que representa la ventana de renderizado de Ogre.
- mResourcePath: String
    - String que contiene el Path del fichero “Resources.cg”
- mSystem: *System
    - Sistema central de la API CEGUI.
- mRenderer: *OgreCEGUIRenderer
    - Objeto renderizador de la GUI para Ogre de CEGUI
- Simulacion3DApp()
    - Constructor de la clase que inicializa los objetos Root y Framelistener de Ogre a null.
- ~ Simulacion3DApp()
    - Destructor de la clase que destruye los objetos Root y FrameListener.
- go()
    - Llama a Setup() y empieza el renderizado.
- bool configure()
    - Muestra la ventana de configuración inicial de la aplicación y si se pulsa OK crea la ventana de renderizado.
- chooseSceneManager()
    - Crea el SceneManager de Ogre y le asigna un nombre.
- createCamera()
    - Crea las 4 cámaras que tenemos.
- createFrameListener()
    - Instancia el FrameListener y se lo asignamos al objeto raíz o Root.

- createScene()
  - Creamos el cursor de CEGUI, hacemos la lectura del fichero "ConfigSim.cg" para guardarlo en la máquina de estados, creamos la escena del EscenaManager, creamos todos los menús de CEGUI con MenuManager y activamos como menú de CEGUI el Menú principal.
- createViewports()
  - Creamos un viewport y se lo asignamos a las 4 cámaras que tenemos (las 4 se verán por el mismo).

- setupResources()
  - Leemos las secciones que hay en el "Resources.cg" y guardamos las rutas donde encontraremos los recursos (modelos, materiales, texturas, etc.)
- loadResources()
  - Inicializa todos los recursos en Ogre.
- bool setup()
  - Hace la lectura de los plugins y de la configuración inicial de la aplicación creando el objeto principal de Ogre llamado Root, configura los recursos, crea la máquina de estados, crea el SceneManager de Ogre, crea las cámaras, crea los viewports, carga los recursos, crea la escena y crea el FrameListener.

Signat: David Rel Grau

Bellaterra, 8 de Febrer de 2010

Aquesta memòria conté tota la evolució del desenvolupament del projecte a tractar, es a dir, un entorn gràfic per a simulacions de sistemes biològics. Es partirà des de uns objectius a complir, buscant quina via es la que millor que s'adapta a aquests. Seguidament s'expliquen els requeriments establerts, y fent una abstracció d'ells, el disseny general del software per a posteriorment passar al desenvolupament del mateix. Finalment, es mostren les proves realitzades sobre aquest, deduint conclusions que ajudaran a la seva posterior utilització y manteniment.

Esta memoria contiene toda la evolución del desarrollo del proyecto que se va a tratar, es decir, un entorno gráfico para simulaciones de sistemas biológicos.
Se partirá desde unos objetivos a cumplir, buscando que vía es la que mejor se adapta a ellos. Seguidamente se explican los requerimientos establecidos, y abstrayéndolos, el diseño general del software para posteriormente pasar a el desarrollo del mismo. Finalmente, se muestran las pruebas realizadas sobre él, sacando conclusiones que ayudarán a su posterior uso y mantenimiento.

This report has inside the whole evolution of the development of the project that we're going to talk about. This is a graphical environment for simulations of biological systems.
First of all, we'll start looking for which is the better way to reach the aims. Then we'll explain the requirements and how to abstract them to do a great design of the software, using it in the development. Finally, we'll analyze the tests done to the application, extracting conclusions that will help us to use and maintenance it.