PROYECTO DE SISTEMAS INFORMÁTICOS

UNIVERSIDAD COMPLUTENSE DE MADRID
Facultad de informática
Curso 2011-2012

SIMULACIONES DE EMERGENCIA EN ENTORNOS 3D

DIRECTORES
Gonzalo Méndez Pozo
Federico Peinado Gil

AUTOR
Guillermo Martínez del Camino

SIMULACIONES DE SITUACIONES DE EMERGENCIA EN ENTORNOS 3D

Proyecto de Sistemas Informáticos

Facultad de Informática

Universidad Complutense de Madrid

Autor:

Guillermo Martínez del Camino

Profesores directores:

Gonzalo Méndez Pozo

Federico Peinado Gil

Curso 2011/2012

Autorización:

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

Palabras clave para búsquedas bibliográficas

- Simulación
- Entornos 3D
- Agentes
- Evacuación
- Fuego
- Emergencia
- Sistema multi-agente
- Multitudes
- Log

Keywords

- Simulation
- 3D environments
- Agents
- Evacuation
- Fire
- Emergency
- Multi-Agent system
- Crowds
- Log

Quiero expresar mi agradecimiento:

- A mi madre y mi hermano, por traerme las dosis necesarias de café para la consecución de este proyecto, soportar mi humor cuando algo no funcionaba como debía e infundirme paciencia y tesón en los momentos adecuados.
- A Gonzalo Méndez Pozo, director del proyecto, por las innumerables reuniones mantenidas, correos intercambiados y orientaciones recibidas.
- A mis amigos, por liberarme por momentos de la tensión del curso.

1. Sobre el Proyecto

Este proyecto tiene como objetivo la representación de simulaciones de evacuación en situaciones de emergencia en un entorno tridimensional. Más concretamente, simula incendios en el edificio de la Facultad de Informática de la Universidad Complutense de Madrid. La aplicación, cuyo desarrollo está enmarcado en el proyecto de investigación MILES [MILES], toma como entrada los informes realizados por otra aplicación de MILES y los representa visualmente en un entorno realista.

Esta aplicación aporta un nuevo punto de vista a la simulación de incendios en la universidad, ya que permite moverse libremente por el edificio durante los momentos cruciales de la evacuación sin perder detalle de todo lo que acontece. Esta herramienta tiene como cometido extraer datos que puedan servir para mejorar tanto las instalaciones, como los protocolos de evacuación de forma que se minimicen los daños y, sobre todo, las víctimas en caso de incendio.

2. About this Project

The goal of this project is to perform emergency evacuation simulations in a three-dimensional environment. More specifically, it simulates fires that take place in the Faculty of Computer Science's building of the Complutense University of Madrid. The application, whose development is framed in the research project MILES [MILES], uses the reports generated by a different MILES application as input and portrays them in a visually realistic environment.

This application provides a new point of view to the fire simulation in the university, since it allows the user to move freely all along the building during the crucial moments of the evacuation without losing detail of anything that may occur. The aim of this tool is to gather information that may help to improve the facilities and the evacuation protocols so the damages and victims in case of fire can be minimized.

3. Sobre este documento

Este documento describe todas las fases del proceso de realización de este proyecto de Sistemas Informáticos. Los distintos capítulos revelan desde las motivaciones que hicieron que se llevara a cabo, a las conclusiones a su finalización pasando por todos los puntos intermedios como el diseño y su implementación. Las secciones son las siguientes:

1. Introducción
2. Trabajos relacionados: Da una idea de los proyectos e investigaciones realizados por otros investigadores hasta el momento en el campo de la simulación de evacuaciones y del fuego. De este modo se puede observar dónde queda este proyecto entre todos ellos.
3. Motivación y objetivos: Explica desde dónde se partía y lo que se quería conseguir con el proyecto.
4. Diseño: Describe la estructura de la aplicación y por qué se hizo así.
5. Implementación: explica el proceso seguido para plasmar el diseño en código ejecutable.
6. Pruebas y validación: ilustra los test realizados a la aplicación para comprobar su solidez.
7. Conclusiones
8. Referencias
9. Bibliografía
10. Apéndice

Todo el documento, exceptuando las secciones que la normativa exige en inglés, ha sido escrito en idioma español.

4. About this document

This document describes all of the stages of this project's accomplishment. The different chapters reveal from the motivations that carried it through, until the findings of its ending, passing throughout everything in between, like the design and its implementation. The sections are the following ones:

1. Introduction
2. Related Works: It gives the reader an idea of the projects and researches accomplished by other researchers until the moment in the evacuation and fire simulations field. This way, this Project can be placed and seen where it would fit among them.
3. Motivation and goals: It explains from where it started and what wanted to be accomplished with this project.
4. Design: It describes the structure of the application and why it was made that way.
5. Implementation: It shows the process followed in order to shape the design as executable code.
6. Tests and validation: It illustrates the tests made to check the application's soundness.
7. Conclusions
8. References
9. Bibliography
10. Appendix

The whole document, with the exception of the sections that the regulation require other way, has been written in Spanish.

ÍNDICE

1. INTRODUCCIÓN 5

2. TRABAJOS RELACIONADOS 11
SIMULACIÓN DEL FUEGO 11
CFAST 11
SIMULACIÓN DE MULTITUDES Y EVACUACIONES 12
TRABAJOS DE NURIA PELECHANO Y NORMAN I. BADLER 12
PEDESTRIAN DYNAMICS 13
SIMPCE 14
TRABAJOS DE DANIEL THALMANN 15

3. MOTIVACIÓN Y OBJETIVOS 17
PUNTO DE PARTIDA 17
MODELADO DE EVACUACIÓN DE MULTITUDES MEDIANTE AGENTES 17
PROYECTO AVANTI 18
ENTORNO VIRTUAL 3D MULTIUSUARIO PARA SIMULACIÓN DE ESCENARIOS DE EVACUACIÓN 18
OBJETIVOS 19

4. DISEÑO 21
DISEÑOS PRELIMINARES DESCARTADOS 21
CLASE AGENTE 21
GAMESTATES 21
DISEÑO FINAL 21
ARQUITECTURA DE OGRE 21
FLUJO DE UN PROGRAMA EN OGRE 23
FORMATO DEL FICHERO .LOG 25
PARSEO DEL FICHERO .LOG 26
PERSONAJES 27
FUEGO 29
MENSAJES E INDICADORES 31
CÁMARAS 32
VISIÓN GENERAL CENITAL 34
CAMBIO DE ESCALAS 35

5. IMPLEMENTACIÓN 37
PROCESO DE DESARROLLO 37
1º FASE – INICIALIZACIÓN Y CARGA DEL ENTORNO 37
2ª FASE – CARGA DE OBJETOS 38
3ª FASE – LISTA DE DESTINOS Y MOVIMIENTO 38
4ª FASE – LISTA DE FUEGO Y LISTA DE PERSONAJES 38
5ª FASE – PARSEO DEL FICHERO .LOG Y ADAPTACIÓN DE ESCALAS 39

6ª FASE – DETALLES FINALES 39
TECNOLOGÍAS BARAJADAS PERO NO UTILIZADAS 40
PYOGRE 40
MAGTOOLS 40
JAVA3D Y OGRE4J 40
TECNOLOGÍAS UTILIZADAS 41
C++ 41
OGRE (OBJECT-ORIENTED GRAPHICS RENDERING ENGINE) 41
OGREMAX 42
WIN32 API 42
HERRAMIENTAS UTILIZADAS 43
VISUAL STUDIO 2010 PROFESSIONAL 43
ECLIPSE 43
OGITOR 43
PARTICLEEDITOR 44
BLENDER 45

6. PRUEBAS Y VALIDACIÓN 47
1ª FASE – INICIALIZACIÓN 47
2ª FASE – CARGA DE OBJETOS 47
3ª FASE – LISTA DE DESTINOS Y MOVIMIENTO 47
4ª FASE – LISTA DE FUEGO Y LISTA DE PERSONAJES 47
5ª FASE – PARSEO DEL FICHERO .LOG Y ADAPTACIÓN DE ESCALAS. 47
6ª FASE – DETALLES FINALES Y DISEÑO DE PERSONAJES 48

7. CONCLUSIONES 49
DISCUSIÓN DE LOS RESULTADOS OBTENIDOS 49
REVISIÓN DE LOS OBJETIVOS INICIALES 49
RESULTADOS POSITIVOS 50
RESULTADOS NEGATIVOS 50
TRABAJO FUTURO 51
AMBIENTACIÓN 51

8. REFERENCIAS 53

9. BIBLIOGRAFÍA 57
LIBROS 57
FOROS DE INTERNET 57
TUTORIALES 57

APÉNDICE A – MEJORA EN LA GUI DEL SIMULADOR MULTIAGENTE 59

APÉNDICE B - MANUAL DE INSTALACIÓN 61

APÉNDICE C – IMPLEMENTACIÓN DE FUNCIONES **63**
FRAMERENDERINGQUEUED 63
UPDATECHARACTERLIST 64
UPDATE 65
CALCULATEY 66
ROTATE TO DIRECTION 68

APÉNDICE D - MANUAL DE USUARIO **69**
CONFIGURAR LA SIMULACIÓN **69**
CONFIGURAR EL ENTORNO GRÁFICO **69**
CONTROL DE LA SIMULACIÓN **70**

ÍNDICE DE FIGURAS

FIGURA 1: SIMULACIÓN DE FUEGO Y HUMO EN CFAST 12

FIGURA 2: EJEMPLO DE UNO DE LOS PROGRAMAS DE N. PELECHANO Y N. BADLER [PEBA06] 13

FIGURA 3: PEDESTRIAN DYNAMICS 14

FIGURA 4: SIMPCE 14

FIGURA 5: VIRTUAL CROWDS 15

FIGURA 6: DIAGRAMA DE CLASES DE OGRE 22

FIGURA 7: CICLO DE RENDERIZADO DE UN FRAME EN OGRE 24

FIGURA 8: DIAGRAMAS DE LAS CLASES PERSONAJE Y LISTA DE PERSONAJES 29

FIGURA 9: ASPECTO DEL FUEGO EN ESTE PROGRAMA 30

FIGURA 10: DIAGRAMAS DE LAS CLASES FUEGO Y LISTA DE FUEGO 31

FIGURA 11: ASPECTO DE UN PERSONAJE EN ESTE PROGRAMA 32

FIGURA 12: PROYECCIÓN CÓNICA 33

FIGURA 13: CÁMARA DE VISIÓN CENITAL 34

FIGURA 14: PROYECCIÓN ORTOGONAL 35

FIGURA 15: EJES DEL MAPA BIDIMENSIONAL DEL SIMULADOR MULTIAGENTE 35

FIGURA 16: EJES DEL ENTORNO TRIDIMENSIONAL 36

FIGURA 17: PANTALLA DE CONFIGURACIÓN DE OGRE 37

FIGURA 18: ESQUEMA DE LA FASE DE DISEÑO 39

FIGURA 19: LA HERRAMIENTA PARTICLE EDITOR 44

FIGURA 20: UNO DE LOS PERSONAJES EN BLENDER DURANTE LA FASE DE TEXTURIZADO. 45

FIGURA 21: MEJORA DE LA COMUNICACIÓN 51

FIGURA 22: GUI DELSIMULADOR MULTIAGENTE [NUGO11]. 59

FIGURA 23: VENTANA DE SIMULACIÓN 69

FIGURA 24: PANTALLA DE CONFIGURACIÓN DEL ENTORNO GRÁFICO. 70

FIGURA 25: PANTALLA DE CARGA DE LA APLICACIÓN 71

FIGURA 26: FUNCIÓN DE LAS TECLAS DURANTE LA SIMULACIÓN 71

1. INTRODUCCIÓN

La seguridad es un punto clave a la hora de construir cualquier edificio. En cualquier momento y lugar puede acaecer un incendio, terremoto, atentado o cualquier emergencia que derive en la rápida evacuación del mismo. La ley da pautas obligatorias sobre las instalaciones y características que todo edificio debe cumplir [RD2059/1981] [RD279/1991] [RD2177/1996]. Desde escaleras de emergencia a extintores pasando por señales, planos e indicaciones colocados en las paredes. Aun así, hay edificios que resultan más eficientes que otros a la hora de realizar estas evacuaciones. Para comprobar esta eficiencia, desde hace tiempo se hacen simulacros de incendio periódicos para medir el tiempo que se tarda en evacuar un edificio cualquiera, pero esto no tiene por qué ser suficiente debido a varios motivos:

- La gente sabe que es un simulacro y no se lo toma en serio.
- El simulacro sólo puede realizarse tras la consecución de la construcción y nunca a priori. Muchos errores de diseño son ya irremediables.
- Se paraliza el funcionamiento de todo el lugar durante la simulación, con su correspondiente coste económico.

Durante las evacuaciones, se producen diversas situaciones que pueden afectar a la correcta realización de las mismas y que no se pueden advertir durante los simulacros de forma precisa. Cuellos de botella en las salidas, ataques de pánico o gente que quiere recuperar algún objeto y se dirige a por él en vez de hacia la salida más próxima. La psicología humana supone un factor muy importante en una evacuación [UrMa05], pero la ley sobre simulacros de incendios sólo obliga a realizar simulacros de evacuación periódicamente [LE3195], sin decir qué hay que tener en cuenta en ellos, de modo que este factor puede pasarse por alto. De hecho, numerosos artículos y estudios como [FaPr97], [Keat82] y [KCDH65] confirman que los habituales simulacros de incendios no nos preparan de manera adecuada para afrontar estas situaciones ya que no se tratan cuestiones como el miedo o la desobediencia a los evacuadores por parte de ciertos individuos.

Desde hace tiempo se ha comprobado que una correcta simulación realizada por ordenador, con una inteligencia artificial dirigiendo a la masa de gente, puede dar resultados mucho más precisos sobre la seguridad de una edificación que un simulacro de incendios en un edificio real [[BuSé97]] o, al menos, dar información que resulte complementaria a la realización de este simulacro. Además, y éste es quizá el punto más interesante, una simulación virtual se puede realizar antes de la construcción, sobre el plano. De este modo pueden advertirse errores de diseño y aplicarse mejoras a priori, que a posteriori serían demasiado caras de introducir, o incluso inviables.

Para esta aplicación, enmarcada en un proyecto de investigación de la Universidad Complutense de Madrid [MILES], hemos elegido la Facultad de Informática de la

misma. La elección se ha realizado por ser el edificio más accesible y del que más información tenemos. Poseemos planos de todas las plantas y modelos 3D de éstas bastante detallados. Con esta aplicación, podemos comprobar lo que ocurriría si hubiera que evacuar la facultad de forma realista y sin necesidad de tener que paralizar el edificio al completo y molestar a sus ocupantes. Además es posible visualizar la simulación tantas veces queramos y con las condiciones que queramos sin necesidad de molestar a nadie.

Gracias a los avances tecnológicos hoy en día podemos simular, mediante cálculos basados en la proyección de elementos en tres dimensiones sobre pantallas bidimensionales como la de cualquier PC o televisión, entornos de apariencia realista. Es el llamado modelado 3D [Watt99]. Estos cálculos requieren de bastante potencia de cálculo, por lo que para realizarlos es necesario poseer una tarjeta gráfica para liberar de esta carga a la CPU y conseguir una simulación fluida. De estos avances se han beneficiado innumerables campos: desde programas de diseño industrial hasta sistemas de entretenimiento como videoconsolas o máquinas recreativas pasando por las simulaciones por ordenador como las que nos atañen.

En el grupo de investigación en el que se enmarca este proyecto [MILES] ya se habían desarrollado tres aplicaciones para simular los incendios y las evacuaciones en el edificio de la facultad [NuGo11] [MMH10] [CDS11]. En uno de ellos, el de Elena Núñez [NuGo11] del que hablaremos con más profundidad más adelante, realiza informes y tiene en cuenta a tres tipos de personas según su carácter. Habrá líderes, que en principio serán la parte del personal del edificio que forma su equipo de evacuación; habrá individuos dependientes, que seguirán a rajatabla las órdenes dadas por estos líderes; e individuos independientes, que seguirán sus propios impulsos para buscar la salida o para satisfacer cualquier necesidad que le avenga, como podría ser volver a rescatar algún objeto de sus taquillas.

La aplicación realizada en este proyecto se dedica a representar visualmente las simulaciones realizadas por la mencionada aplicación de Elena Núñez [NuGo11]. Para ello, toma sus informes de salida y los representa en un entorno tridimensional realista como los que introducíamos anteriormente. Así, en vez de tener que leer informes sobre lo acaecido, podremos ver in situ lo que ocurre, moviéndonos libremente por la facultad para poder ver los puntos que más nos interesen.

En la simulación, cada individuo es lo que en inteligencia artificial se denomina un agente inteligente, es decir, una entidad capaz de percibir su entorno por medio de sensores y responder en consecuencia de manera racional utilizando actuadores [Fone02]. La simulación consta de diversos agentes que interactúan entre ellos: Los individuos y el fuego, que se considerará también un agente y es implementado como tal. Estas características hacen que este sistema en el que se ejecuta la simulación se englobe dentro de los llamados sistemas multi-agente (SMA) [Ferb99].

La aplicación aquí realizada constará de dos partes. La primera es una simple ventana de Windows en la que se elegirá el informe que se quiere representar, el piso en el que la simulación tiene lugar y la calidad del modelo de la facultar a utilizar. Una vez introducidos estos datos, pasaremos a la segunda parte, que es la simulación propiamente dicha. En ella, tendremos disponibles diferentes tipos de cámara para no perdernos detalle de lo que sucede en la simulación. Adicionalmente, se ha añadido también una vista general de cada planta en perspectiva cenital de modo que podemos identificar en tiempo real los puntos de interés y poder dirigirnos a ellos. Asimismo, en pantalla podremos visualizar datos estadísticos como el número de individuos que se ha conseguido evacuar con éxito, la cantidad de personas que continúa intentando salir y la cifra de afectados que hayan resultado quemados. La aplicación representará con un indicador de distinto color a los distintos sujetos, de modo que se pueda identificar unívocamente el tipo de personalidad de los mismos ya sean independientes, dependientes o líderes.

2. TRABAJOS RELACIONADOS

Muchos equipos de trabajo, ya sea de grupos de investigación de universidades, organismos vinculados a empresas o a la administración de algún estado, han investigado y desarrollado aplicaciones destinadas a simular alguno de los comportamientos a los que se dirige este proyecto. Como veremos a continuación, hay algunos trabajos destinados a la simulación del fuego de verdadera calidad. Asimismo también podemos encontrar simuladores de evacuación de edificios realmente buenos. Este proyecto intenta aunar ambas simulaciones en una sola, para conseguir prever el comportamiento de la gente durante un incendio y sus consecuencias.

SIMULACIÓN DEL FUEGO

La simulación del fuego es, computacionalmente, realmente complicada en dos sentidos completamente distintos. Simular su comportamiento, expansión y temperatura de forma realista requiere de modelos matemáticos muy complejos. Igualmente, el aspecto de la llama y el humo son verdaderamente difíciles de emular en entornos 3D, y cuando se consigue el coste computacional se eleva de forma considerable. Es relativamente sencillo mostrar por pantalla una llamarada o una vela encendida, pero cuando tratamos con un incendio al completo hay que tratar de buscar un equilibrio entre realismo y eficiencia.

CFAST

El Instituto Nacional de Normas y Tecnología (NIST) es una agencia dependiente del Departamento de Comercio de los Estados Unidos. El NIST promueve avances tecnológicos, métricos y normativos de modo que estos mejoren la calidad de vida de la sociedad americana. Entre muchas divisiones, encontramos la *Fire Research Division* (División de investigación del fuego), que en los últimos años ha estado desarrollando y mejorando el CFAST [CFAST], una aplicación que simula el crecimiento del fuego y el humo en los edificios mediante modelado 3D. CFAST se divide en dos partes: *Fire Dynamics Simulator* que efectúa la simulación y *Smokeview* que toma la salida para poder visualizarla en un entorno tridimensional.

Figura 1: Simulación de fuego y humo en CFAST

CFAST es de las pocas aplicaciones que, como la nuestra, combina la simulación física del fuego con su simulación visual. De hecho, su funcionamiento es, como podemos observar, prácticamente análogo al de nuestro proyecto con la excepción de que sólo toma en cuenta el fuego y no a la gente. En CFAST, el modelado matemático de la expansión del fuego es muy bueno, pero su aspecto gráfico es muy mejorable.

SIMULACIÓN DE MULTITUDES Y EVACUACIONES

La simulación de multitudes o masas de gente son también objetivo de muchos estudios de diversa índole. El estudio de multitudes no es sólo interesante desde el punto de vista de la seguridad como en nuestro caso. Otros estudios se centran en los flujos de gente desde un punto de vista económico [Ward05]. Por ejemplo, puede ser interesante saber por dónde pasa más la gente, y por qué, para saber dónde colocar un negocio. Hay también bastantes trabajos como los que veremos a continuación que estudian el comportamiento de masas de gente en situaciones de pánico o durante evacuaciones.

TRABAJOS DE NURIA PELECHANO Y NORMAN I. BADLER

Nuria Pelechano y Norman I. Badler, de la Universidad de Pennsylvania, han publicado numerosos trabajos relacionados con el comportamiento de multitudes [PSAB08] [DAPB08] [PSAB07] [PABa07]. Muchos de estos trabajos tratan el comportamiento de multitudes con carácter general, pero otros incluyen estudios de situaciones más concretas. El más interesante por su relación con este proyecto es, sin duda, *“Modelling Crowd and Trained Leader Behavior during Building Evacuation”* [PeBa06]. En él, se estudia el comportamiento de múltiples agentes en evacuaciones en dos situaciones distintas:

1. Todos los agentes son iguales y se comunican para conocer la ruta a seguir.
2. Diferentes agentes toman diferentes roles, como personal entrenado, líderes y seguidores.

Si bien los entornos gráficos en sus simulaciones no son estéticamente perfectos, sí que son funcionales, por lo que las simulaciones realizadas tienen gran interés.

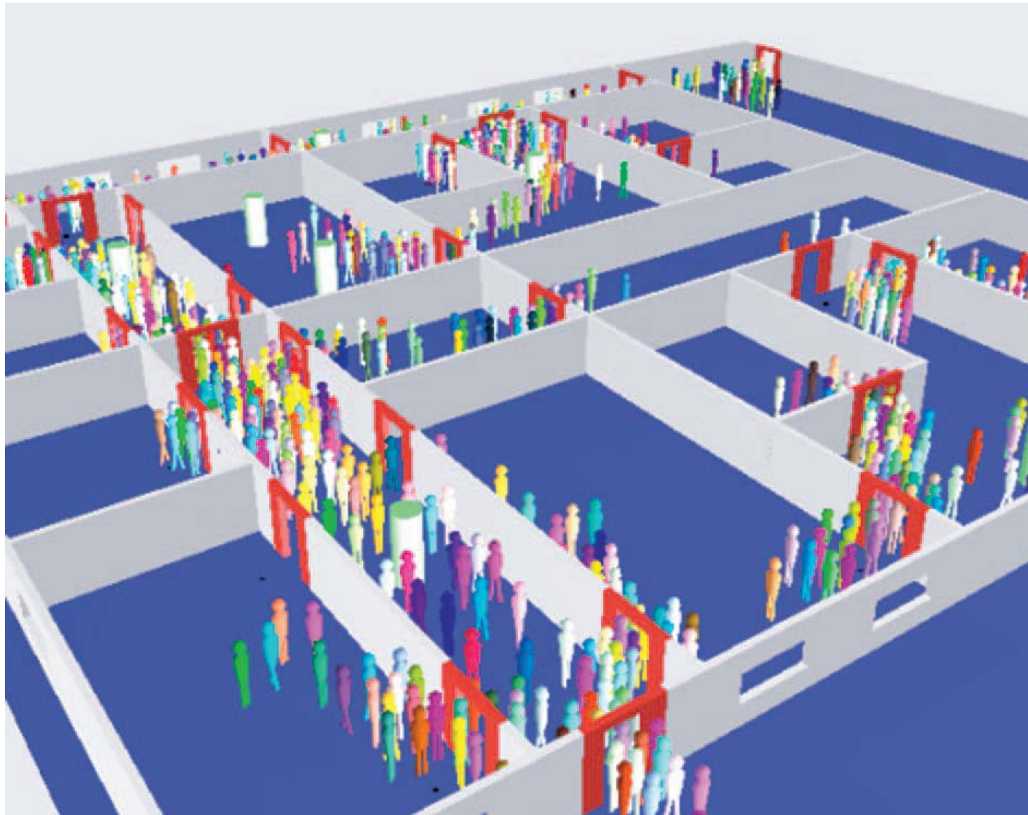

Figura 2: Ejemplo de uno de los programas de N. Pelechano y N. Badler [PeBa06]

PEDESTRIAN DYNAMICS

Pedestrian Dynamics es el trabajo realizado por John Ward para obtener su doctorado en el University College London en 2005 [Ward05]. En él se estudia el movimiento de los peatones en áreas urbanas. El modelo se desarrolló para analizar el flujo de peatones en situaciones normales, en situaciones de alta concentración de gente y, la más relevante para este proyecto, situaciones de emergencia.

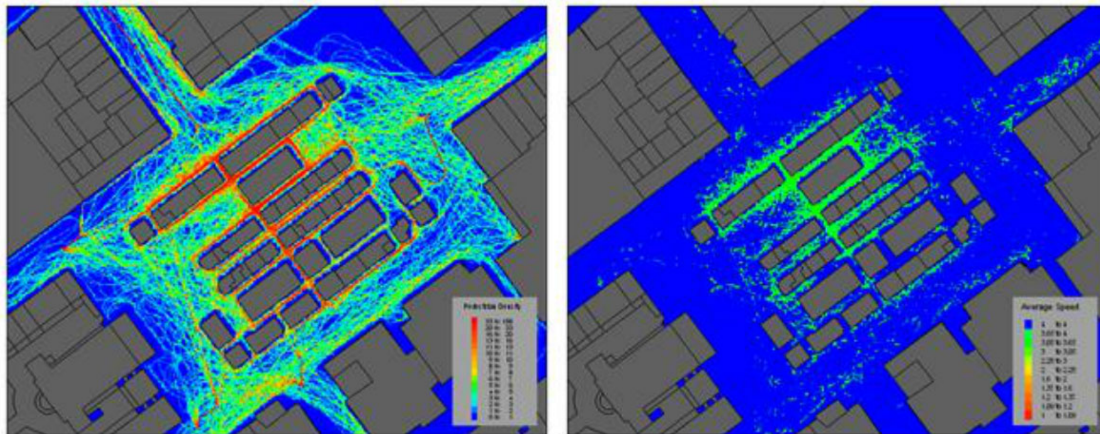

Figura 3: Pedestrian Dynamics simulando el flujo de gente en una plaza de Londres.

Tras las simulaciones, el modelo da estadísticas como la velocidad media de los viandantes por zonas o las zonas de mayor densidad. Además, se pueden introducir variables como el porcentaje de gente del lugar y el porcentaje de turistas que se detienen a mirar monumentos o tiendas, ralentizando el flujo de gente.

SIMPCE

SIMPCE, siglas de Simulador de Movilidad de Personas en espacios Cerrados es un proyecto de fin de carrera realizado por dos alumnos de la Facultad de Ingeniería de la Pontificia Universidad Javeriana de Bogotá [GRGG09]. En él se ejecutan distintas simulaciones en espacios cerrados utilizando distintos paradigmas para el comportamiento de los agentes.

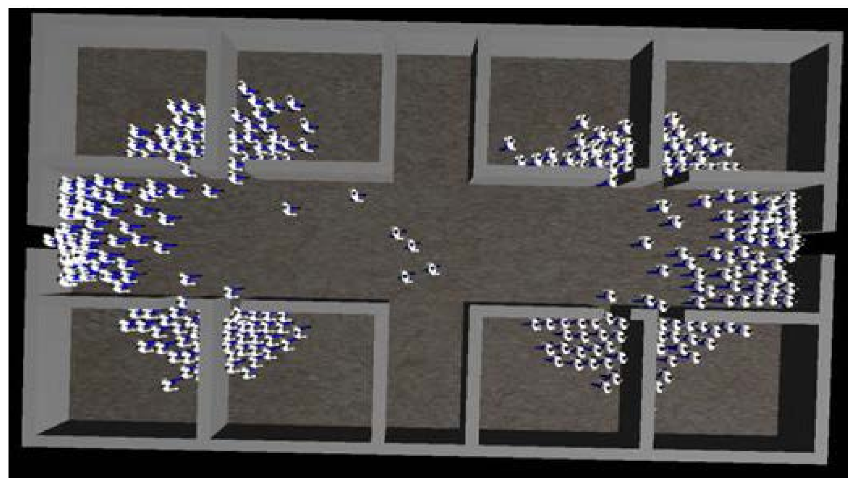

Figura 4: Ejemplo de simulación en SIMPCE

En un entorno gráfico austero con perspectiva cenital, se pueden ejecutar todo tipo de simulaciones, introduciendo una situación inicial con un edificio determinado, el número de agentes, su tipo y su comportamiento veremos cómo se desarrolla la evacuación. Se pueden tener en cuenta parámetros como el peso de una persona (que

repercute en el espacio que ocupa) o la introducción de niños y personas responsables de ellos.

TRABAJOS DE DANIEL THALMANN

Daniel Thalmann es un investigador de la Escuela Politécnica Federal de Lausana. Ha trabajado intensamente en la investigación, publicado numerosos artículos y participado en múltiples conferencias sobre la simulación de multitudes en entornos gráficos en 3D [TYPM09]. Thalmann es el fundador del Virtual Reality Lab (VRlab) perteneciente a la citada escuela politécnica, el cual se dedica principalmente al modelado y animación de mundos virtuales habitados en tres dimensiones.

Entre sus trabajos, destaca su motor de multitudes Virtual Crowds [TGMY092], un sistema multi agente con comportamientos individuales basados en máquinas de estados finitos. Con él pueden simularse masas de gente con comportamientos individuales en tiempo real.

Figura 5: El motor Virtual Crowds siendo utilizado para emular la antigua ciudad de Pompeya

Entre sus posibles usos, aparte de las simulaciones de evacuaciones, se encuentran las recreaciones históricas con usos didácticos, las simulaciones de estaciones de tren o aeropuertos para mejorar su diseño en base a los flujos de gente e incluso el tratamiento de la agorafobia, ya que puede simularse la interacción del paciente con la gente sin necesidad de exponerlo al trauma de salir de casa [TGMY09].

3. MOTIVACIÓN Y OBJETIVOS

Como comprobábamos en la sección anterior, existe un cierto vacío en cuanto a las simulaciones de evacuación existentes. Si bien hay buenas aplicaciones que simulan las evacuaciones en masa, todas lo hacen de manera genérica sin tomar en cuenta si la evacuación se produce por culpa de un ataque terrorista, un terremoto o un incendio como en nuestro caso. Por otro lado, existen proyectos que simulan el fuego y su expansión de forma realista, incluso dentro de edificios, pero se centran sólo en fuego y no simulan qué pasa con la gente que se encuentra en ellos. Este proyecto nace con la intención de aunar ambas facetas en una única simulación, que muestre de forma veraz las consecuencias de un incendio dentro de un edificio, de manera que puedan extraerse conclusiones que deriven en mejoras para la seguridad de sus ocupantes.

PUNTO DE PARTIDA

En esta facultad se habían realizado ya previamente ciertos proyectos relacionados tanto con las evacuaciones [NuGo11] [MMH10] como con los entornos realistas en tres dimensiones [BCPA03] [SIAL10]. Por esta razón, este proyecto nace como una forma de añadir cohesión a estos proyectos y ampliarlos para crear algo más vistoso y realista. Como comentábamos previamente, este proyecto representa visualmente los informes de las simulaciones realizadas en otro proyecto anterior, pero además utiliza algunos elementos adicionales.

MODELADO DE EVACUACIÓN DE MULTITUDES MEDIANTE AGENTES

Este proyecto de fin de máster realizado por Elena Núñez González y dirigido por Pablo Gervás y Gonzalo Méndez en septiembre de 2011 realiza simulaciones de evacuación durante incendios basadas en agentes [NuGo11]. Realizado en lenguaje JAVA, genera informes con muchísima información y además los resume y redacta de forma similar a como lo haría un humano, contando el incendio como si fuera una historia de modo que cualquiera pueda entenderlos. El proyecto maneja multitudes, pero cada una de las personas simuladas responde a su propio carácter, pudiendo éste ser dependiente, independiente o líder. Su punto más flojo es sin duda su interfaz gráfica, una GUI en 2D en la que las personas se representan como puntos y que, al no ser el objetivo del mismo, no se dedicó demasiado esfuerzo a su desarrollo.

En principio, la intención era comunicar ambos proyectos en tiempo real por medio de un socket. De esta forma, se podrían ejecutar ambos programas simultáneamente, y mientras uno simulara la evacuación, el otro recibiría los datos necesarios para representarla. Al final se desechó esta opción debido a problemas de complejidad y, sobre todo, de sincronía.

Para comprender estos problemas, hay que entender que el mapa del sistema multiagente funciona dividiendo el mapa en cuadrantes. Por esta razón, las personas, al moverse, desaparecen de un cuadrante y aparecen instantáneamente en uno

contiguo, que en la realidad está a casi un metro de distancia. Si trasladamos este método a gráficos 3D, veríamos personas desaparecer y aparecer un poco más adelante, lo cual produce un efecto visual poco estético. Si, en cambio, implementamos el movimiento de un punto a otro con una transición fluida por todos los puntos intermedios, puede que no hayamos llegado aún a la posición final del movimiento y nos llegue una nueva posición. ¿Qué hacemos? ¿Giramos desde el punto actual hacia el nuevo punto final y nos dirigimos a él? ¿Pasamos primero por el punto anterior? A esto se nos suma un problema aún mayor: cuando el proyecto en JAVA nos envía una posición, significa que el personaje está ya allí. Si nosotros representamos el movimiento completo, estamos llegando tarde a ese punto.

Para evitar estos problemas y, poder además reiniciar las simulaciones sin tener que enviar comunicaciones desde un programa al otro, este proyecto toma a posteriori los ficheros .log generados por el sistema multiagente [NuGo11] para recrear las simulaciones en él realizadas. Hubo que modificar el código del proyecto JAVA de modo que en los ficheros .log se escribiera toda la información necesaria para su representación posterior, y se hiciera además con una estructura normalizada, para que fuera después más fácil automatizar su lectura.

PROYECTO AVANTI

El Proyecto AVANTI, cuyos autores son Enrique López Mañas, Francisco Javier Moreno y Javier Plá Herrero y cuyos directores son Pablo Gervás y Gonzalo Méndez, fue realizado en el curso 2009/2010 [MMH10]. Es un sistema de asistencia a la evacuación de incendios. En él se utiliza el posicionamiento WiFi y realidad aumentada para asistir al usuario poseedor de un teléfono con sistema operativo Android a salir de la facultad de manera rápida.

ENTORNO VIRTUAL 3D MULTIUSUARIO PARA SIMULACIÓN DE ESCENARIOS DE EVACUACIÓN

Fernando Cruz Ruiz, Alberto Durbey Carrasco y Jorge Sanz Briones, dirigidos por Pablo Gervás y Carlos León desarrollaron durante el curso 2010/2011 una aplicación en la que, un usuario trata de salir de la facultad mientras otro le indica el camino por medio de un chat [CDS11]. En este proyecto se utiliza una versión del modelo 3D de la facultad más antiguo y está programado en lenguaje Python con motor gráfico OGRE, denominado PyOGRE. En un principio se pensó en utilizar la aplicación realizada en este proyecto como base para el desarrollo de la aplicación del proyecto sobre el que trata esta memoria, pero se descartó al poco tiempo debido al poco mantenimiento y la obsoleta información que hay en internet sobre PyOGRE, además de ciertos problemas de compatibilidad entre las diferentes versiones de Python. Sin embargo, sí se ha podido aprovechar el modelo del alumno realizado para esa aplicación para reutilizarlo aquí.

OBJETIVOS

Ya existía en el grupo de investigación [MILES] el proyecto de guiado para la evacuación durante incendios en la facultad [CDS11], así que se pensó que se podría añadir nueva funcionalidad y se pasó a trabajar en ello. El proyecto consistiría en representar las simulaciones realizadas por el sistema multiagente [NuGo11] en el entorno tridimensional de la facultad utilizado por [CDS11]. Tras desarrollar sockets para unir en tiempo real ambas aplicaciones se comprobó que mostrar las simulaciones en tiempo real uniendo ambos programas no era viable. Adicionalmente, pese a que se podría haber reutilizado el entorno de la aplicación como un Framework para este proyecto, se decidió que Python no era tampoco la plataforma adecuada. Por lo tanto los objetivos iniciales cambiaron.

Llegados a este punto, se pensó que lo mejor sería utilizar los informes del mencionado sistema multiagente [NuGo11] y representarlos en una aplicación completamente nueva, basada en OGRE y programada en C++. La elección de OGRE y C++, cuya activa comunidad de usuarios hace previsible que siga utilizándose y mejorándose en el futuro, preveía que el proyecto desarrollado podría ser ampliable fácilmente en el futuro y, además dejaríamos disponibles clases y recursos que podrían ser reutilizados para futuros proyectos con la Facultad como fondo.

Por lo tanto los objetivos finales a resolver en el proyecto quedaron así:

- Mostrar una simulación de una evacuación en caso de incendio en un entorno 3D realista basándonos en los ficheros .log del sistema multiagente [NuGo11].
- Hacer que la simulación sea fácil de seguir y que todos sus elementos sean fácilmente identificables durante la misma.
- Mostrar además el paso de mensajes de unas personas a otras de forma clara.
- Representar en una interfaz gráfica las estadísticas de la simulación.

4. DISEÑO

DISEÑOS PRELIMINARES DESCARTADOS

Durante la fase inicial de diseño se tuvieron en cuenta varias opciones que al final no fueron implementadas por diferentes motivos que se detallan a continuación.

CLASE AGENTE

Dado que la simulación sobre la que trabajamos es del tipo multiagente, se barajó la posibilidad de crear una clase abstracta agente de la cual heredarían los distintos tipos de agente: fuego, persona independiente, líder o persona dependiente. La opción se descartó debido a las grandes diferencias existentes entre la clase fuego y las demás, que hacían de la superclase y su herencia una opción poco recomendable.

GAMESTATES

La mayoría de los video juegos se realizan siguiendo el patrón *game state* [BjHo05]. Es una especialización del patrón *state* en la que cada estado es un nivel del juego. Cuando se necesita pasar de un nivel a otro, simplemente se cambia de estado. La idea en esta aplicación era crear el menú de inicialización con un estado y la simulación en otro. Al final, dada la simplicidad de la aplicación, se optó por inicializar la simulación con una ventana de Windows evitándonos así la complejidad añadida por el *game state*.

DISEÑO FINAL

Tras descartar las opciones anteriores, se llegó a un diseño sencillo, modular y fácilmente ampliable que se detalla más adelante, pero para su total compresión necesitamos saber primero cómo funciona OGRE y cómo está construido.

ARQUITECTURA DE OGRE

La arquitectura de OGRE es un tanto peculiar, dado que está hecho para funcionar sobre OpenGL o sobre Direct3D indistintamente. Por ello, la clase *Root* (Raíz) del motor está separada de la implementación específica del sistema de renderizado por la clase abstracta *RenderSystem* (Sistema de renderizado).

Otra clase importante en la arquitectura de este motor es *ResourceManager* (administrador de recursos). *ResourceManager* es una clase abstracta que se utiliza para gestionar los recursos. Se divide en subclases que manejan texturas, materiales y fuentes. Por lo general, esta clase se inicializa tomando como entrada un fichero en el que están escritos todos los datos necesarios de los recursos a utilizar.

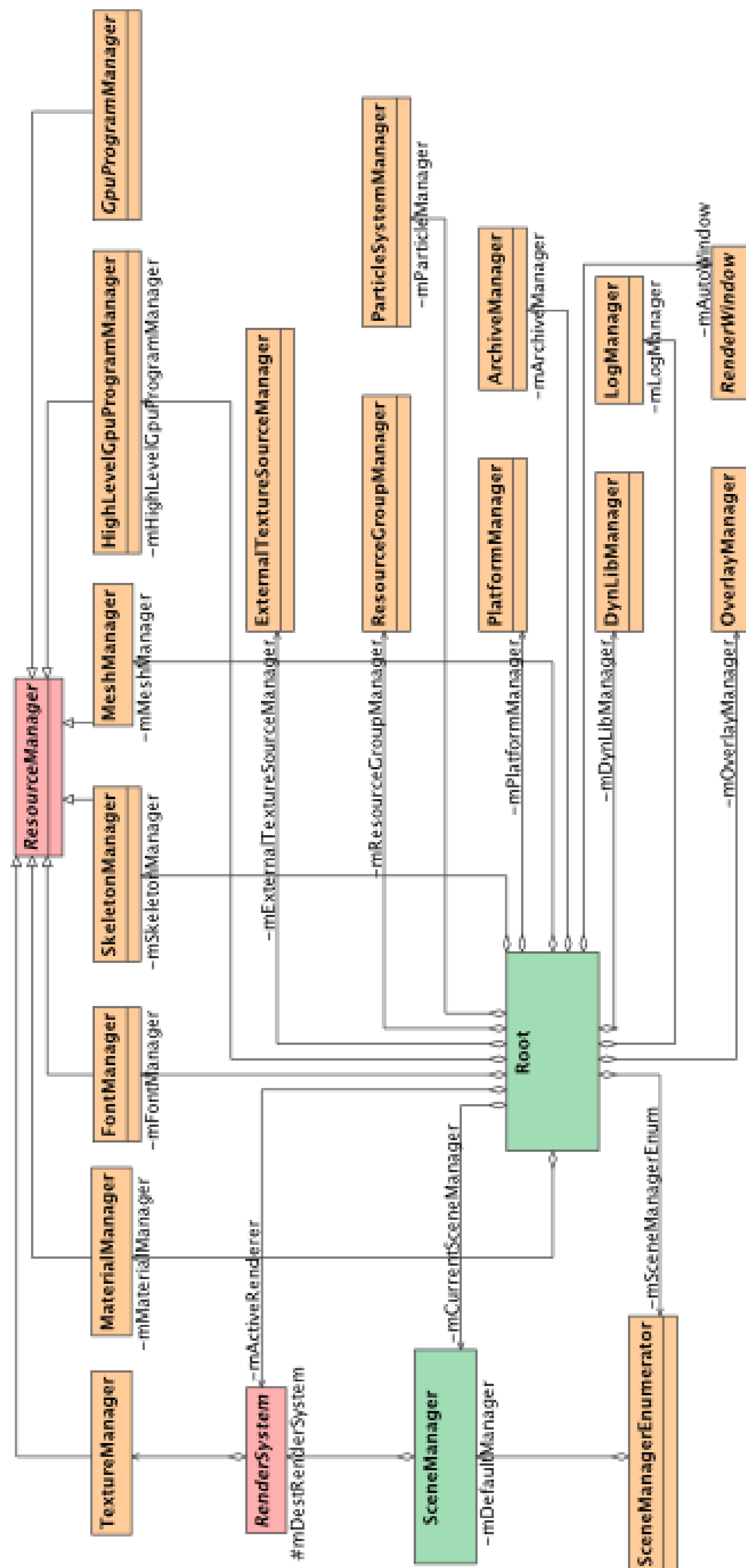

Figura 6: Este diagrama muestra la clase Root de OGRE y todas las clases Manager que dan acceso a los diferentes subsistemas.

Todas las clases del diagrama, con la excepción de RenderSystem, SceneManager y RenderWindow utilizan un patrón singleton para asegurar que sólo haya una instancia de cada una de ellas.

FLUJO DE UN PROGRAMA EN OGRE

A la hora de crear un programa en OGRE, hay dos fases bien diferenciadas. La primera es la fase de inicialización o configuración y la segunda es la fase de renderizado.

Durante la **fase de inicialización**, se inicializa el administrador de recursos, se inicializa el administrador de escena, se cargan los plugins a utilizar por el programa, se cargan en la memoria de la tarjeta gráfica los recursos a utilizar por la escena y se inicializan todos los valores y flags a utilizar por el programa. Una vez todo está preparado, se llama a la función *startup*.

Ciertos tipos de escena, con terrenos de grandes dimensiones, utilizan una clase SceneManager (Administrador de escena) específica que carga durante la fase de renderizado las partes del terreno que rodean a la cámara. Como nuestra escena no es de este tipo no profundizaremos en este tipo de administrador.

Una vez terminada la fase de inicialización, con la llamada a *startup* comienza la **fase de renderizado**. En ella, la aplicación entra en un bucle que por lo general, sólo dejará de iterarse cuando la aplicación termine. En este bucle, OGRE llama a tres funciones que nosotros utilizaremos para tener control de la aplicación.

1. frameStarted: Es llamada justo antes de comenzar a renderizar la escena para el siguiente frame.
2. frameRenderingQueued: Se llama tras renderizar la escena, pero antes de hacer el intercambio del doble buffer. Aquí es donde realizamos cualquier cambio que ocurra en la escena, por ejemplo el movimiento de un personaje. Lo normal, es dotar a todos nuestros objetos de una clase update, que será llamada desde aquí una vez por frame.
3. frameEnded: Se llama justo tras el intercambio del buffer, momento en el que nuestro nuevo frame aparece por pantalla.

Para que la aplicación funcione de forma fluida y no notemos saltos, es deseable que el número de veces que se itere este bucle sea de, al menos, veinticuatro veces por segundo. Así, la pantalla se refrescará más de veinticuatro veces por segundo y no notaremos ningún efecto indeseable.

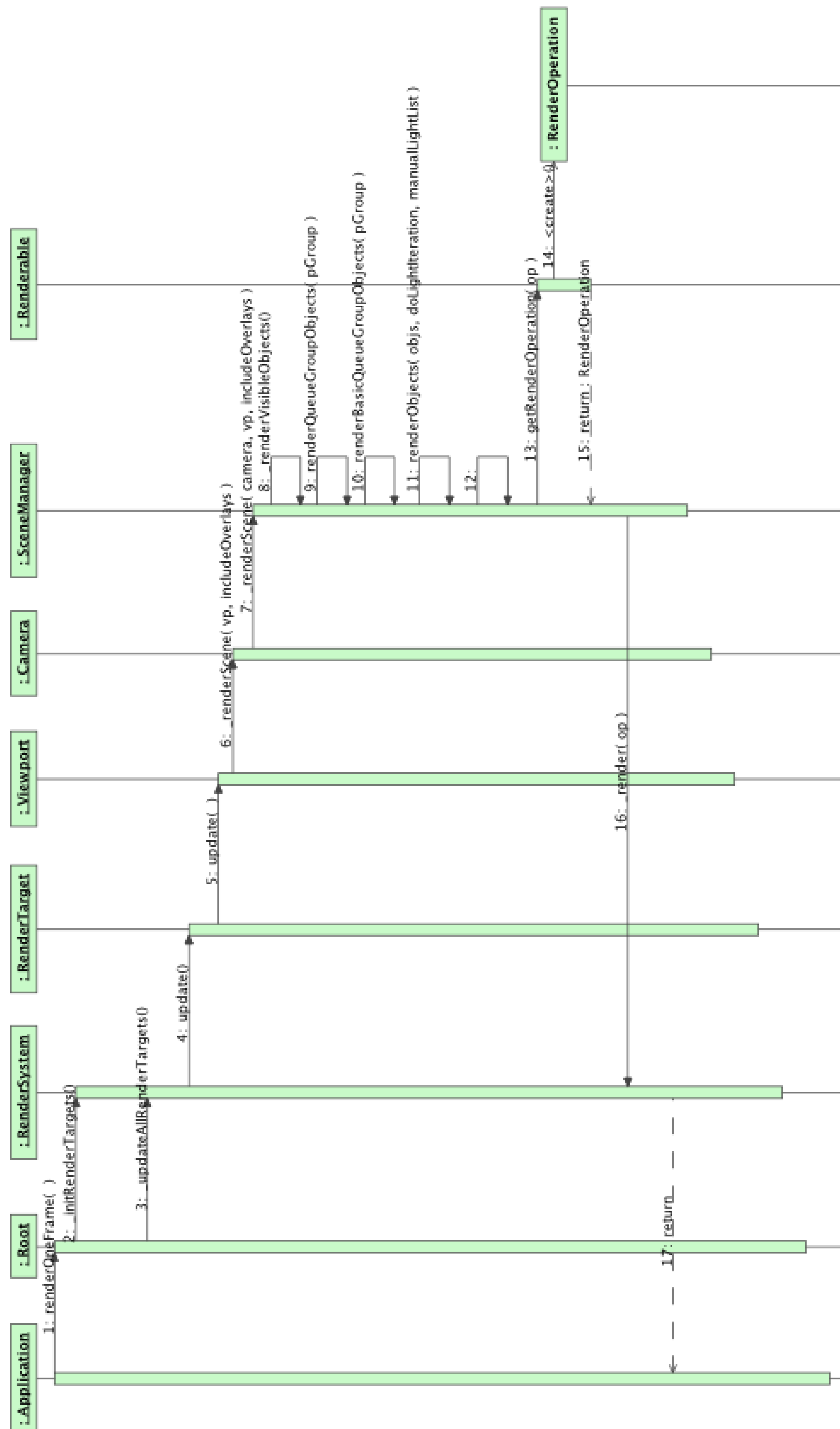

Figura 7: Ciclo de renderizado de un frame.

FORMATO DEL FICHERO .LOG

El fichero .log que creaba el proyecto [NuGo11] era un fichero de texto sin un formato claro. Cada línea representaba una acción y daba cierta información, pero no siempre en el mismo orden ni siguiendo el siempre el mismo patrón. Como nuestro programa debía leerlo de forma sistemática, la complejidad del parseo de un fichero de este tipo era demasiado alta, así que se procedió a rediseñar el formato del fichero de modo que siguiera un patrón que simplificara su lectura.

- **Patrón**

El nuevo fichero .log sigue el patrón descrito a continuación.

1. Cada línea representa una acción de un agente, ya sea personaje o fuego.
2. Cada línea tiene siempre los mismos campos y en el mismo orden:

```
Nombre_del_agente fecha hora num-msj (x,y) acción campo_extra
```

3. El campo extra depende de la acción, algunas veces no se utiliza, pero tiene relevancia en estos momentos:
    a. Si la acción es “envía-msj” el mensaje viene en el campo extra.
    b. Si la acción es “recibe-msj” el mensaje viene en el campo extra.
    c. Si la acción es “andar-hacia” la posición hacia la que se dirige viene en el campo extra.

- **Ejemplos**

Para su mejor comprensión, a continuación se muestran algunas líneas que pueden aparecer en el fichero .log con su contenido explicado:

```
Leader3 2012-05-13 13:25:00,749 77 (42,51) envia-msj ven_conmigo
```

El agente líder “Leader3” ejecuta en la posición 42, 51 la acción “envía-msj” siendo el mensaje enviado “ven_conmigo”

```
Dependent6 2012-05-13 13:25:15,748 524 (39,49) seguir-a Leader3
```

El agente dependiente “Dependent6” ejecuta en la posición 39, 49 la acción “seguir-a”. En el campo extra observamos que es “Leader3” el agente a seguir.

```
Dependent6 2012-05-13 13:25:15,748 525 (39,49) envia-msj te_sigo
```

Dependent6, que se encuentra en la misma posición de antes, envía el mensaje “te sigo”.

```
Leader3 2012-05-13 13:25:15,751 526 (42,51) recibe-msj te_sigo
```

Al momento, Leader3 recibe el mensaje “te_sigo”.

PARSEO DEL FICHERO .LOG

El fichero .log es el que, una vez inicializado el entorno 3D y se han cargado la escena, el cielo, el terreno y las luces, tiene la información suficiente para inicializar los agentes (personajes y fuego) y dar lugar a la simulación. Tanto la lista de personajes como la lista de focos de fuego tienen un método que inicializa las listas tomando como entrada el fichero a representar.

- **Lista de personajes**

Los personajes cargan en sus estructuras de datos diferente información dependiendo de cinco tipos de línea.

1. Con las líneas cuya acción es nacimiento se crea un personaje cuya posición es la posición indicada en la línea, se indica su tipo y se añade a la lista.

```
Leader1 2012-05-13 13:24:22,881 4 (58,62) nacimiento Liderazgo: 1
```

2. Con las líneas cuya acción es envia mensaje, se añade a la pila de mensajes el mensaje indicado y la posición desde la que se envía. De este modo, al llegar a este punto de la simulación, el personaje mostrará este mensaje y lo eliminará de su pila.

```
Leader2 2012-05-13 13:24:59,722 36 (32,60) envia-msj ven_conmigo-26-1
```

3. Con las líneas de accidente, se añade el lugar en el que se encuentran en el momento de la muerte, de este modo al llegar a él y tocar el fuego, se procederá a eliminarlos de la lista de personajes aumentando el contador de muertes.

```
Leader3 2012-08-30 20:10:55,356 410 (42,54) accidente muerte
```

4. Cuando la acción es permanecer en lugar seguro, quiere decir que el personaje se ha salvado. Por lo tanto, al llegar a dicho lugar, lo eliminaremos de la lista aumentando el contador de salvados.

```
Dependent7 2012-08-30 20:12:05,123 2596 (69,37) permanecer-lugar-seguro
```

5. Con todas estas líneas y con cualquier otra referida a un personaje, lo único que haremos será añadir su posición a la lista de posiciones a recorrer. Dado que todo personaje indica su posición cada vez que esta varía, podemos trazar sin problemas su viaje desde un punto inicial hasta uno final pasando por todos sus puntos intermedios. Además, si en alguno de ellos, envía un mensaje, lo mostraremos.

- **Lista de focos de fuego**

El fuego sólo tiene dos posibles mensajes. Su nacimiento y su expansión.

1. El inicio del foco crea el primer foco en la posición indicada.

```
Fire1 2012-08-30 20:10:42,222 14 (42,64) inicio-foco posiciones
```

2. Las propagaciones vienen indicadas en distintas líneas, pero siempre van juntas y se envían a la vez. Por ello, controlaremos con un valor booleano si la anterior línea parseada era una línea de propagación. Si es así, añadiremos al anterior bloque de propagación un nodo más, si no, crearemos un bloque de propagación nuevo.

```
Leader3 2012-08-30 20:10:46,238 113 (42,61) envia-msj 42,61_coordenadas
Fire1 2012-08-30 20:10:46,242 114 (40,63) envia-msj propagación
Fire1 2012-08-30 20:10:46,242 115 (41,62) envia-msj propagación
Fire1 2012-08-30 20:10:46,242 116 (42,61) envia-msj propagación
Fire1 2012-08-30 20:10:46,242 117 (43,62) envia-msj propagación
Fire1 2012-08-30 20:10:46,242 118 (44,63) envia-msj propagación
Leader2 2012-08-30 20:10:46,244 119 (32,60) envia-msj 32,60_coordenadas
```

Estas líneas añadirían al fuego una propagación de cinco focos.

PERSONAJES

Para implementar los personajes se han implementado dos clases: Character y CharacterList.

- **Character**

La clase Character es quizá la más importante del programa. Está formada por varios objetos que se detallan a continuación:

1. mName: Variable tipo String para identificar inequívocamente al personaje.
2. *mMainNode: Puntero al nodo principal del personaje. Nos dice su posición actual y de él dependen los demás nodos del personaje.
3. *mMsgNode: Puntero al letrero que sobrevuela su cabeza con su último mensaje enviado.
4. *mMsgList: Puntero a la pila de mensajes.
5. *mCharNode: Puntero al nodo al que se encuentra adjunta la malla del personaje.
6. *mBillBoardNode#: Hay cuatro y se encargan de crear el cuadrado indicador alrededor del personaje que identifica su tipo y que lo hace más visible, sobre todo en la cámara cenital.
7. *mSceneMgr: Puntero al *Scene Manager* de OGRE. Necesario a la hora de crear nodos, ya que hay que adjuntarlos al árbol principal.
8. *mEntity: Entidad del personaje.
9. *mWalkList: Lista de puntos por los que el personaje debe pasar. A medida que los recorre se van eliminando de la lista.
10. Col: Motor de colisiones.

Además, el personaje posee información sobre su dirección, destino, animación actual, su mensaje a mostrar sobre la cabeza y flags que lo identifican como líder, vivo o

muerto o evacuado. También posee otro flag que sólo se utilizó durante la etapa de debug para controlar al personaje por teclado.

El personaje tiene, aparte de las funciones de set y get y su constructora y destructora, algunas funciones clave:

1. calculateY: Utilizando el motor de colisiones, mide la distancia del nodo principal al suelo y modifica su posición en consecuencia, de modo que el personaje pueda subir escalaras o bajar rampas.
2. RotateToDirection: Una vez llegado a un punto de la lista, si hay un punto siguiente, se basa en las leyes trigonométricas para rotar al personaje hasta que encare su nuevo destino.
3. CreateDecal: Crea el indicador cuadrado del color pasado como parámetro.
4. Update: Clase común a cualquier objeto de OGRE. Tiene como entrada el tiempo pasado desde el último frame y calcula en consecuencia el nuevo estado del objeto: su posición, animación o cualquier cosa que varíe con el tiempo.

- **CharacterList**

Como su propio nombre indica, CharacterList es una lista de objetos tipo Character que simplifica el acceso a cada uno de ellos y abstrae las funciones de creación, actualización y destrucción de cada uno de ellos. Internamente actúa con iteradores de modo que en la clase principal del programa sirva con una sola línea para crear, actualizar o destruir todos los personajes. CharacterList tiene además una función de vital importancia: se ocupa de parsear el log para inicializar el comportamiento de los personajes.

Durante el parseo, se crea un personaje nuevo por cada línea de nacimiento que se encuentra en el .log. Se inicializan las listas de destinos con los puntos por los que pasan los personajes, transponiendo las coordenadas de la simulación en JAVA de modo que correspondan con las coordenadas con las que trabajamos este proyecto, en las que una unidad en el código se corresponde con un centímetro en la realidad.

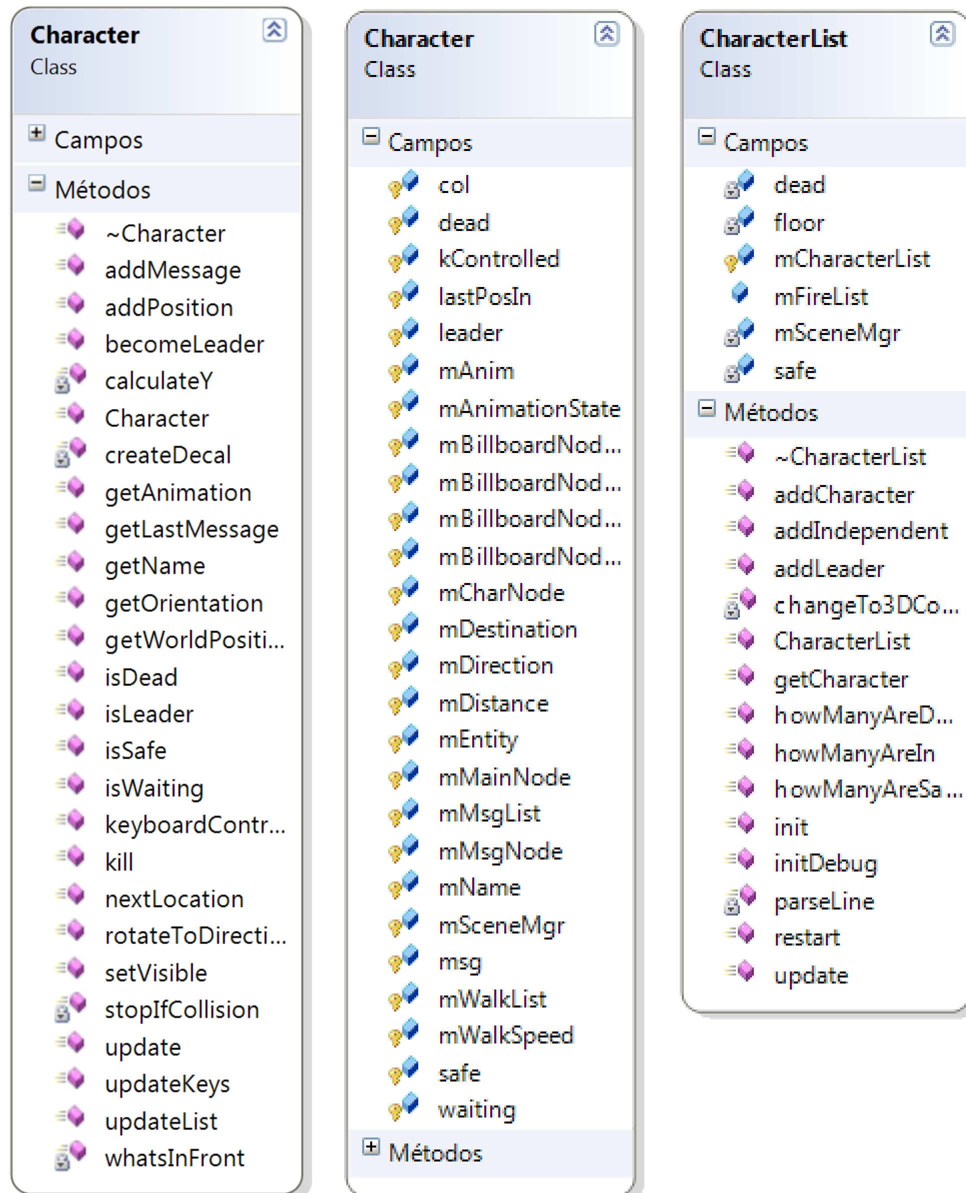

Figura 8: Diagramas de las clases personaje (Character) y lista de personajes (CharacterList). El diagrama de la clase Character ha sido divido en dos para su mejor visibilidad. En el bloque de la izquierda quedan los métodos y en el del centro los campos.

FUEGO

Para implementar el fuego se han implementado también dos clases de objetos: Fire y FireList. Una instancia de la clase Fire será un foco de fuego, y FireList contendrá una lista con todos los objetos de tipo Fire de la simulación.

- **Fire**

La clase fuego tiene alguna similitud con la clase personaje. Tiene su propio identificador y su nodo principal. En lugar de una entidad, se le adjunta un efecto de partículas que simula el fuego y el humo. Además, se añade una luz amarillenta en su posición. Por último, sus flags indican si ha sido activado o apagado.

Figura 9: Aspecto del fuego en este programa

- **FireList**

FireList tiene un funcionamiento análogo a CharacterList, pero con objetos de tipo Fire. Como ésta, parsea el fichero .log para inicializar el comportamiento y las distintas expansiones del fuego, pero tiene además un cometido extra: maneja y controla las etapas de expansión del fuego, activando focos nuevos y apagando los efectos de partículas en los antiguos, dejando el suelo de color negro para mostrar que esa zona está quemada. De esta forma, evitamos tener innumerables efectos de partículas activados simultáneamente que ralentizarían la aplicación de forma dramática.

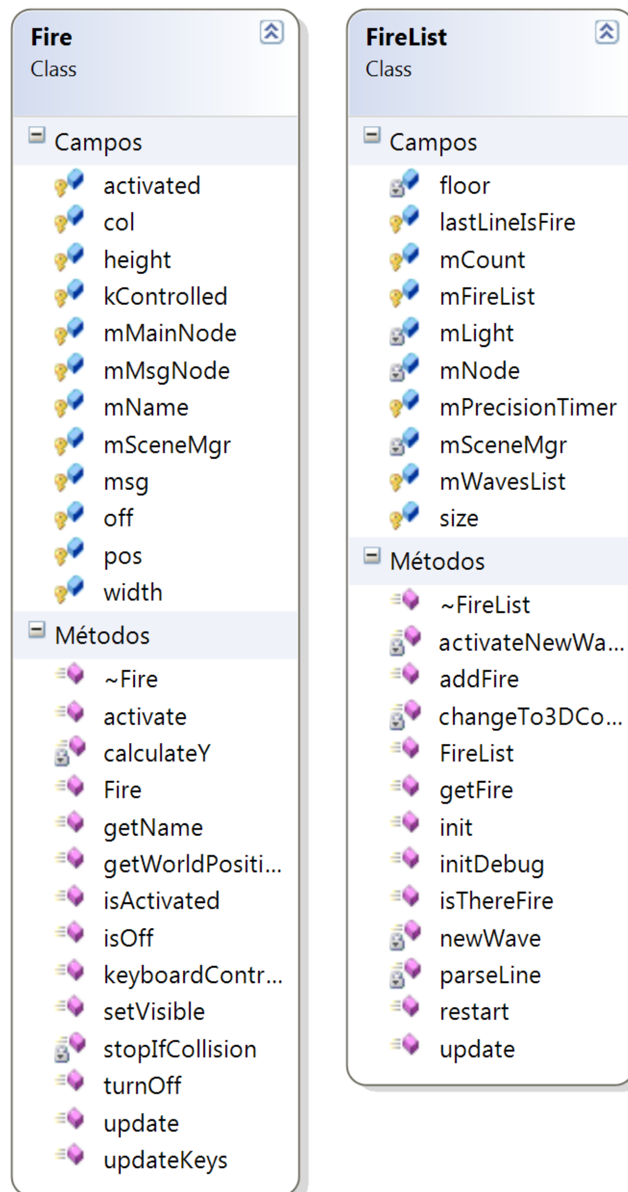

Figura 10: Diagramas de las clases fuego (Fire) y lista de fuego (FireList)

MENSAJES E INDICADORES

Para mejorar la visibilidad de los personajes e identificarlos a cada uno con su tipo, cada uno de ellos tiene un indicador cuadrado que será verde si el personaje es dependiente, rojo si es un líder y azul si es independiente. El indicador está formado por cuatro rectángulos que se superponen en las esquinas para aparentar unidad. No se hicieron sólidos, aunque esto reduciría la complejidad del código, por razones estéticas y porque interferían con el motor de colisiones a la hora de medir la distancia al suelo.

Además, cada personaje tiene un letrero sobre su cabeza en el que se representan sus mensajes enviados. Los agentes durante la simulación intercambian mensajes. Por ejemplo, un líder puede llamar a los agentes que tiene más cerca e indicarles que lo

sigan. El código para los letreros fue tomado de un proyecto libre encontrado en los foros de la web de OGRE y modificado con otra fuente, tamaño y espaciado de letra para que quedara bien en este programa. Hubo que modificar numerosas partes del código ya que estaba obsoleto y no funcionaba con la última versión de OGRE.

Figura 11: Con su indicador y su letrero, un personaje luce así

CÁMARAS

El programa, consta de cámaras colocadas en posiciones estratégicas a lo largo de la facultad que cubren prácticamente todos sus rincones. Además, cada una de las cámaras puede ser despegada de su posición y movida a lo largo de entorno para ir siguiendo la evacuación de una forma óptima.

Todas estas cámaras utilizan proyección cónica. Esta proyección tiene un funcionamiento muy simple. Se trazan líneas desde la posición del espectador hacia la dirección en la cual mira con un ángulo determinado. Se marcan además dos distancias límites que determinan dos planos. Todo lo que está contenido entre estos dos planos, forma una sección de pirámide cuadrada determinada *frustum*. Los objetos se proyectan en la dirección del espectador cortando al plano cercano. El plano cercano es lo que se muestra por pantalla.

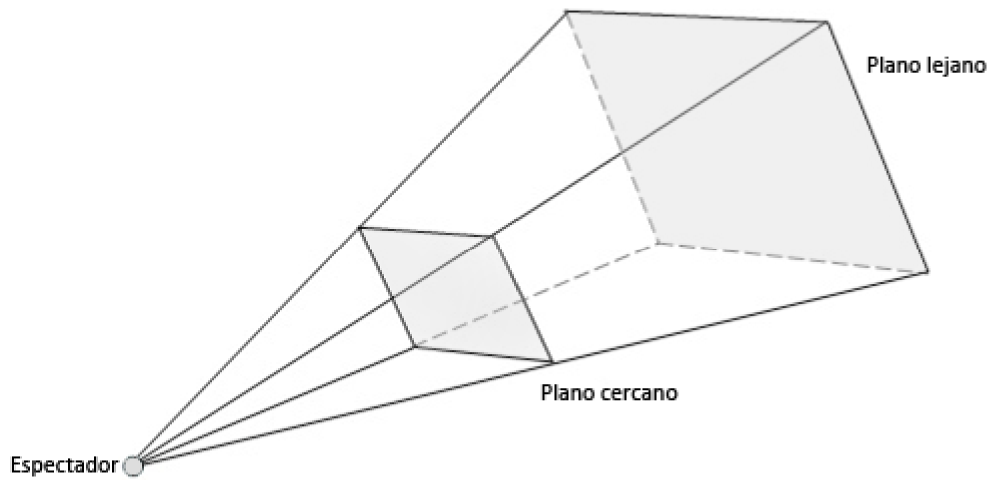

Figura 12: Proyección cónica.

Los planos y ángulos de cada cámara están introducidos para tener una amplia visión de cada una de las zonas tratando de utilizar los recursos mínimos del ordenador. Todo aquello que queda tras una pared está cortado por el plano lejano para no tener que ser computado innecesariamente. Motores gráficos más avanzados como Source implementan la técnica de *Occlusion culling*. Así no se computan todos aquellos objetos que, aunque estén dentro del volumen del frustum, queden fuera del campo de vista del espectador al ser tapados por otros. Dado que OGRE no implementa dicha técnica de forma nativa, hemos optado por un uso inteligente de las cámaras para liberar de carga a la GPU del ordenador.

La posición de las cámaras, dirección en la que miran y los límites de sus planos, están guardadas en un fichero .xml que el programa parsea para inicializarlas. Esto facilita enormemente la inclusión de nuevas cámaras o la eliminación de otras, ya que puede hacerse modificando este fichero sin tocar el funcionamiento interno del programa.

Dado que para cargar la escena necesitábamos las bibliotecas de OgreMAX y, para que éstas funcionaran, se necesitaban las bibliotecas de TinyXML. Aprovechamos que ya teníamos TinyXML instalado para parsear el fichero XML de las cámaras.

A continuación vemos un fragmento del interior del fichero. Esta parte representa la creación de una cámara en el hall de la facultad.

```
</camera>
    <camera name="Hall 1">
    <position x="2120" y="263" z="1273" />
    <orientation ow="-0.98" ox="0.094" oy="-0.16" oz="-0.015"/>
    <distance horizon="2075"/>
</camera>
```

Como en todo momento podemos ver dentro del programa en qué lugar está colocada la cámara en caso de estar utilizando la cámara libre, podemos volar con ella hasta la posición que deseemos para conocer sus datos, y crear así una cámara nueva.

VISIÓN GENERAL CENITAL

Debido a la naturaleza de las simulaciones, en las que grupos distintos de gente pueden estar en lugares opuestos del edificio, puede ser complicado seguir la evacuación completa utilizando sólo las cámaras descritas previamente. Por ejemplo, estábamos siguiendo a un grupo de cuatro personas que han conseguido alcanzar las escaleras, pero el programa nos indica que hay seis personas más en el edificio y no sabemos dónde.

Por eso, se ha incluido una visión general del edificio en tiempo real, que hace más fácil identificar dónde está teniendo lugar la acción. Así siempre podremos saber dónde está ocurriendo algo y dirigirnos hacia allá.

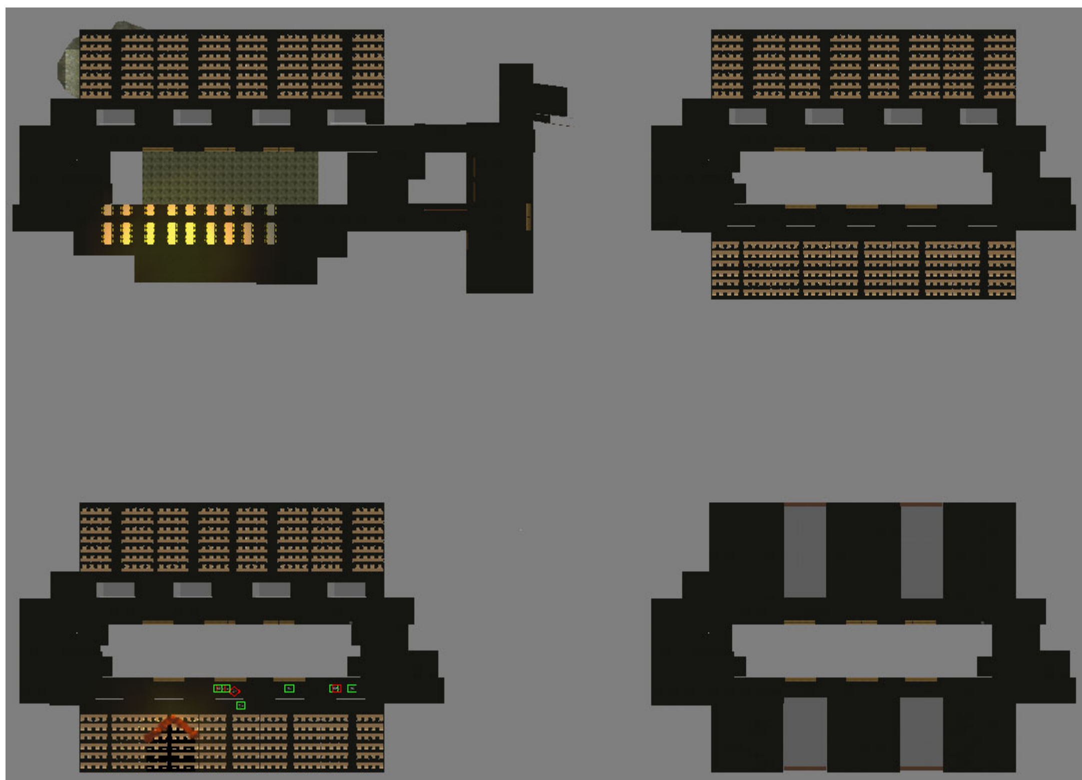

Figura 13: La visión cenital nos muestra como hay personajes en la zona sur del segundo piso

Esta visión cenital está implementada con cuatro cámaras que se le muestran al espectador de forma simultánea. Una por cada piso de la facultad. El quinto no ha sido representado ya que el modelo del que disponíamos no lo tiene modelado. Estas cuatro cámaras utilizan proyección ortogonal, como si de un plano arquitectónico en planta se tratara.

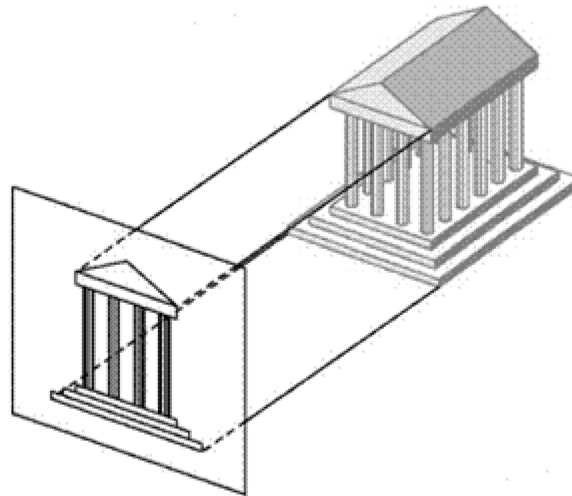

Figura 14: Proyección ortogonal

La proyección ortogonal es matemáticamente más simple que la cónica, ya que simplemente se proyectan los objetos de forma perpendicular a un plano dado al que cortan, que es el que se muestra por pantalla. Pese a su simplicidad, OGRE no da la opción de crear cámaras de proyección ortogonal, por lo cual hubo que transformar matemáticamente la proyección mediante una función.

CAMBIO DE ESCALAS

Los ficheros .log de los que obtenemos la información para representar la simulación, vienen escritos con las unidades del proyecto [NuGo11]. Este programa realiza sus simulaciones sobre un plano del edificio, al que divide en celdas cuadradas a lo largo de los ejes x e y. Los cuadrantes se identifican por su posición respecto a los ejes, que no pueden tomar valores negativos, situándose el origen en el extremo del plano situado más arriba y a la izquierda.

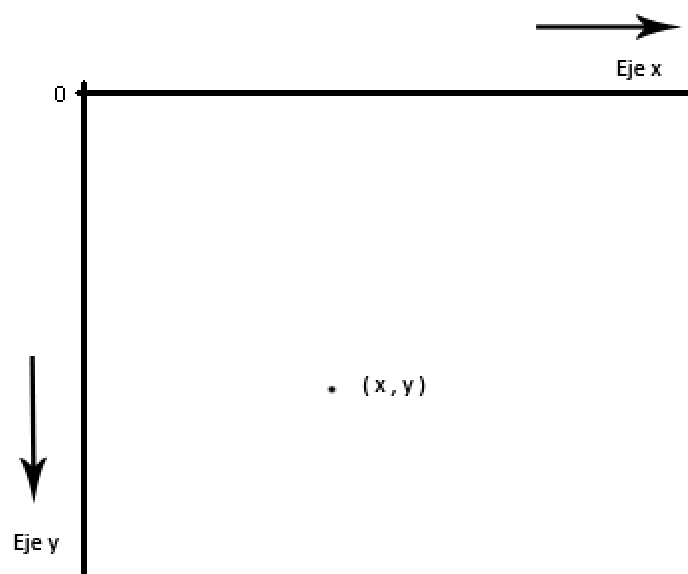

Figura 15: Ejes del mapa bidimensional del proyecto que genera los informes

Nuestra aplicación, al ambientarse en un entorno tridimensional, necesita de una dimensión más para tomar en cuenta la altura. Además, mientras los cuadrantes del programa que realiza los informes son de un tamaño arbitrario, siendo cada uno de ellos de aproximadamente 55 cm de ancho y 54 de alto, nuestro proyecto está realizado a escala 1:1 y tiene el origen en el centro del patio de la facultad a la altura del primer piso, pudiéndonos mover por cualquiera de los tres ejes tanto hacia el lado positivo como hacia el negativo.

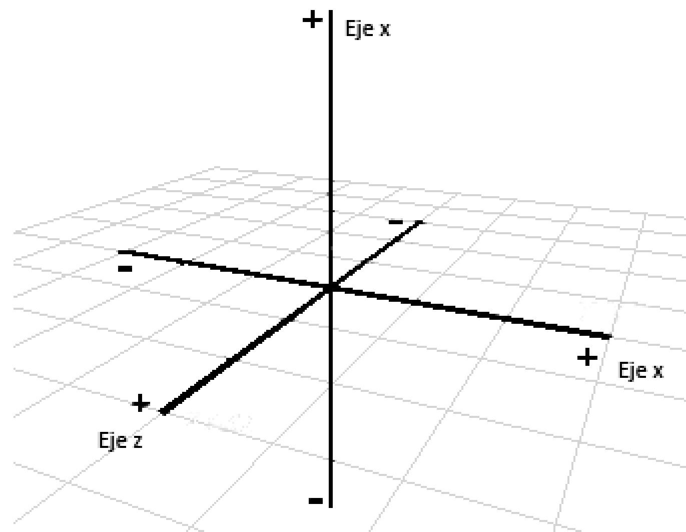

Figura 16: Ejes del entorno tridimensional

Por esta razón, se requería una función que tras parsear el .log traspusiera todos los valores de posición leídos en él a la escala del nuevo proyecto antes de inicializar con ellos tanto los caminos de los personajes como las propagaciones del fuego.

5. IMPLEMENTACIÓN

PROCESO DE DESARROLLO

Una vez los objetivos del proyecto quedaron establecidos, se eligieron las tecnologías a utilizar y se especificó el diseño, se comenzó con la implementación de la aplicación. Se comenzó desde cero y poco a poco se fue añadiendo funcionalidad mediante distintas fases. Tras cada fase se obtenía un prototipo que se testeaba hasta que quedaba libre de fallos y se procedía a iniciar la fase siguiente.

1º FASE – INICIALIZACIÓN Y CARGA DEL ENTORNO

Se comenzó con el desarrollo partir de una aplicación básica de OGRE, que puede descargarse en su repositorio oficial. Esta aplicación simplemente carga las librerías de OGRE y su ventana de inicialización, en la que puede elegirse si queremos que OGRE utilice las librerías de OpenGL o de DirectX, la resolución de la pantalla, o si queremos ejecutar en pantalla completa o en ventana la aplicación.

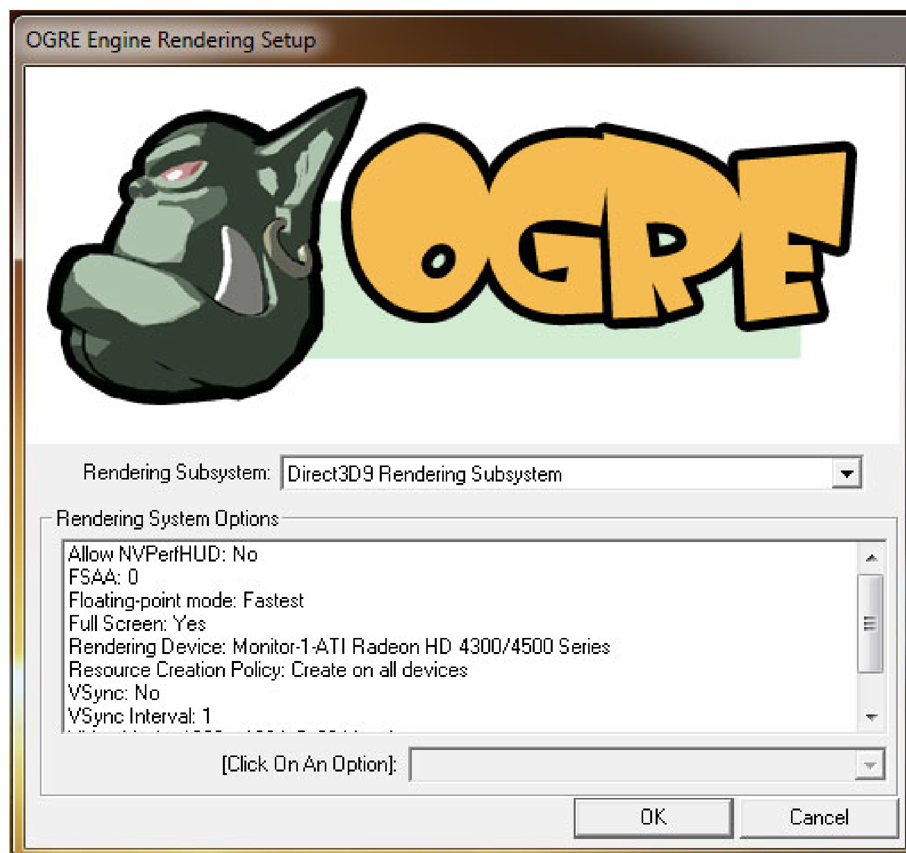

Figura 17: Pantalla de configuración de OGRE

Una vez tuvimos OGRE funcionando correctamente, se añadieron las bibliotecas de OgreMAX, cuyo cometido principal es cargar la escena de la facultad. Se implementó asimismo una cámara que permitía volar a través de la facultad manejada con el ratón y el teclado.

2ª FASE – CARGA DE OBJETOS

Tras la primera fase, el prototipo de la aplicación nos permitía configurar las opciones gráficas de la aplicación y, una vez inicializada, cargaba la escena de la facultad y podríamos movernos por ella. Ya teníamos el escenario propicio para proceder a la carga de entidades y efectos de partícula y comenzar así a perfilar las clases Character y Fire.

OGRE viene con varios objetos por defecto que pueden utilizarse para cargarlos y hacer pruebas con ellos. Se utilizó la cabeza del propio logo de OGRE para hacer las primeras pruebas y familiarizarse con el entorno. Tras las primeras pruebas comenzamos con la implementación de las clases.

La clase Character inicial tenía como malla un robot que viene incluido con OGRE y se desarrolló para ella un control por teclado idéntico al utilizado en multitud de videojuegos. Se integró un control de colisiones para que el personaje pudiera subir y bajar escaleras y rampas, que no eran tenidas en cuenta en el fichero .log al funcionar la simulación previa en dos dimensiones sin tener en cuenta la altura. En cambio, no se tuvieron en cuenta las colisiones con las paredes, ya que, al estar los caminos a recorrer por los personajes definidos de antemano, estos no atravesarían ni golpearían ninguna durante las simulaciones de las evacuaciones. Se añadió también el letrero sobre las cabezas de los robots que más tarde mostraría los mensajes.

La clase Fuego inicial era simplemente un nodo al que se le adjuntaba un efecto de partículas. Visualmente se trabajó en el efecto para que quedara lo más realista posible, y se decidió diseñar un efecto que tuviera el humo incluido, para no tener que utilizar dos simultáneos que multiplicarían el trabajo de la GPU.

3ª FASE – LISTA DE DESTINOS Y MOVIMIENTO

Tras la segunda fase, el prototipo cargaba la facultad y un personaje que podía moverse a lo largo y ancho de la facultad a voluntad del usuario utilizando el teclado, pero lo que se necesitaba era que el personaje siguiera un camino preestablecido que leería del informe con el que se inicializaría. Por ello, se procedió a implementar una nueva función de movimiento. Esta función, dada una lista de puntos del mapa, los iría recorriendo todos uno a uno y eliminándolos de la lista al llegar a ellos hasta quedar la lista vacía, momento en el que el personaje se paraba.

4ª FASE – LISTA DE FUEGO Y LISTA DE PERSONAJES

Se procedió a englobar a todos los personajes y a todos los focos del fuego en dos clases: CharacterList y FireList. Se implementaron funciones de construcción, inicialización, actualización y destrucción para ambas clases, que por medio de un iterador simplificaban y unificaban el control de todos los objetos de la simulación. Las funciones de inicialización se realizaron para testear el resto de funciones, pero más tarde sería sobrescrita con el parseo del .log.

Si bien para la lista de personajes envolverlos en una lista era un trabajo trivial, para la clase del fuego hubo que implementar las propagaciones del fuego. La inicialización encendía los fuegos iniciales, pero además archivaba una lista de propagaciones. Estas propagaciones tenían lugar en la función de actualización, en periodos de tiempo regulares.

5ª FASE – PARSEO DEL FICHERO .LOG Y ADAPTACIÓN DE ESCALAS

Una de las fases clave del proyecto fue la del parseo del fichero .log. Para ello hubo que estandarizar la salida del programa de Elena Núñez [NuGo11], ya que cada línea del log tenía un formato distinto.

Una vez realizado el parseo, hubo que adaptar las escalas. Esta fue una de las partes más problemáticas del proyecto. El motivo es que la escena 3D de la facultad no está hecha a una escala perfecta. Por ejemplo, mientras en [NuGo11] las paredes tienen grosor, en la escena 3D de la facultad éstas no son más que un plano de grosor infinitesimal. Por ello, pese a haber medido e implementado una función que, de ser ambos planos proporcionales funcionaría perfectamente, los personajes y focos del fuego atravesaban paredes.

Tras innumerables ciclos de mediciones, modificaciones de la escena en un editor y de la función y test de las mismas, se obtuvo una función de trasposición y un mapa de la facultad que hacían que los caminos trazados en la simulación del programa [NuGo11] se correspondieran con los trazados en nuestra aplicación.

6ª FASE – DETALLES FINALES

En esta fase se diseñaron y animaron los personajes en Blender y sustituyeron a la malla anterior que utilizábamos, que provenía de un proyecto anterior [CDS11]. Para una mejor visibilidad de los mismos, se les añadió el cuadrado identificador.

Además, se desarrolló una pequeña interfaz con Win32 para configurar la simulación y elegir el .log a representar. También se configuraron las opciones de luces y sombras para una correcta y atractiva iluminación de la escena. Por último se añadieron el terreno sobre el que reposa la facultad y el cielo para añadir realismo.

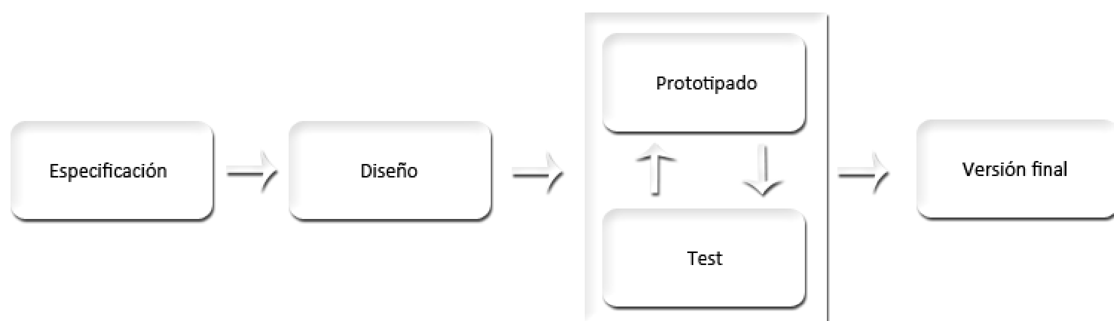

Figura 18: Esquema de la fase de diseño.

Tras la realización de la 6ª fase de prototipado y test, se obtuvo la versión final de la aplicación.

TECNOLOGÍAS BARAJADAS PERO NO UTILIZADAS

Durante la planificación del desarrollo de la aplicación, se barajaron numerosas plataformas y herramientas a utilizar que, finalmente, por diversas razones no se llegaron a utilizar.

PYOGRE

En un principio, la aplicación aquí realizada estaría basada en las clases desarrolladas por un proyecto anterior [CDS11]. La opción se descartó al poco tiempo debido a que la información oficial sobre PyOgre no se actualiza desde 2009. Además, existen problemas de compatibilidad entre algunas versiones de PyOgre y algunas versiones de Python. Esto hacía que, si bien el desarrollo era posible, la estabilidad, compatibilidad y la posibilidad de ampliarlo en el futuro no eran las más halagüeñas a priori. Además, la obsoleta información que existía en la red sobre PyOgre no facilitaba la realización de la aplicación, por lo que esta opción se descartó.

MAGTOOLS

MAG Tools (Masive Avatar Generation) es una aplicación desarrollada como proyecto de Sistemas Informáticos en la Universidad Complutense en 2010 [MML10]. Sus autores, José María Méndez González, Lidia Martínez Prado y Gabriel López de Armas, crearon un conjunto de herramientas para la visualización y personalización de personajes tridimensionales. De este modo, a partir de un personaje básico, pueden crearse numerosos personajes individuales distintos con relativa facilidad. MAG Tools pone a disposición del programador dos librerías que permite empaquetar los recursos gráficos y manipularlos. En un principio parecía una opción perfecta para crear el número necesario de alumnos con características aleatorias, de forma que todos fueran distintos y sin necesidad de modelarlos uno a uno. El problema surgió cuando vimos que los ficheros generados son del tipo Collada, un estándar que OGRE no maneja. Existen un par de intentos fallidos de cargadores de ficheros Collada para OGRE, pero son muy antiguos y están llenos de bugs. Asimismo, MAG Tools ni siquiera sigue al completo las especificaciones de Collada, si no que basándose en estas, describe unas propias que son utilizadas en un motor realizado por uno de los autores del proyecto.

JAVA3D Y OGRE4J

Tras haber descartado ya la realización del proyecto en Python, se pensó que, ya que el proyecto que realiza los informes está realizado en JAVA, podría simplemente añadírsele la funcionalidad de una vista 3D de la simulación realizada con algunos de los motores 3D disponibles en este lenguaje. Java3D es un motor bastante sencillo y con bastante información disponible pero le faltan algunas funcionalidades básicas como la animación de personajes, y además nuestro modelo de la facultad estaba

realizado para OGRE. Tras haber descartado Java3D surgió la idea de Ogre4J, es decir, OGRE para Java. El problema con Ogre4J fue el mismo que el de PyOgre, la última versión data de 2009, la actividad en sus foros era casi nula y, de haberla, sólo servía para mencionar Bugs, de ahí que se descartara también.

TECNOLOGÍAS UTILIZADAS

Tras haber descartado ya todas las tecnologías mencionadas anteriormente, nos decidimos por comenzar el desarrollo utilizando C++ como lenguaje y OGRE3D como motor 3D. Las razones de esta elección se detallan a continuación.

C++

C++ es un lenguaje de programación diseñado a mediados de los años ochenta por Bjarne Stroustrup. El lenguaje extiende el lenguaje de programación C con mecanismos orientados a la programación a objetos entre otras mejores. Actualmente es el cuarto lenguaje de programación más utilizado, teniendo especial importancia en el desarrollo de aplicaciones que requieren alto rendimiento o gráficos en tres dimensiones [TISO12]. Su elección viene dada por tres razones básicas:

1. Rendimiento sobradamente demostrado en diversos motores de videojuegos comerciales como Steam o Id Tech. [LOGE]
2. Experiencia previa realizando proyectos en C++.
3. OGRE funciona originalmente y de forma óptima con C++, aunque luego haya sido portado por medio de *wrappers* a otros lenguajes.

OGRE (OBJECT-ORIENTED GRAPHICS RENDERING ENGINE)

OGRE es un flexible motor tridimensional orientado a escenas y escrito en C++. Está diseñado para hacer más fácil e intuitivo para los desarrolladores la programación de aplicaciones que utilizan aceleración gráfica 3D por hardware. La biblioteca de clases de OGRE abstrae al programador de las clases y funciones de las bibliotecas subyacentes sobre las que funciona: Direct3D y OpenGL. La interfaz basada en objetos de OGRE simplifica al desarrollador la creación de cámaras, la carga de mallas de objetos y muchas otras de las acciones más comunes que se realizan en una aplicación de este tipo.

La principal ventaja de OGRE, y de cualquier otra interfaz gráfica, es la minimización del esfuerzo a la hora de renderizar escenas 3D, al estar la implementación 3D abstraída por el motor. De este modo, el programador no tiene que ocuparse de operaciones como el culling jerárquico, las transparencias o el manejo del estado del renderizado. Además, su diseño orientado a objetos facilita enormemente el manejo de escenas complejas.

Otra de sus grandes ventajas es su sistema de materiales y sombreados. Estos son cargados de ficheros de texto independientes del código, lo cual simplifica el código

enormemente desacoplándolo y haciendo de la modificación de materiales y shaders algo mucho más cómodo y simple.

OGRE es completamente gratuito y está disponible bajo la licencia MIT. El código fuente está a disposición de cualquiera y todo el mundo puede modificarlo, dejando a su elección si publicar dichas mejoras en su comunidad o no. La gratuidad de OGRE, hace que exista una comunidad de desarrolladores bastante amplia que se dedica a crear bibliotecas, escribir tutoriales y ayudar a sus usuarios por medio de Wikis y foros. Adicionalmente, se encuentra disponible una amplia documentación para todas sus clases. Todas estas razones propician que numerosos videojuegos y aplicaciones, tanto amateur como profesionales, hayan sido realizados utilizando OGRE.

Esta aplicación utiliza OGRE como base para la representación de las simulaciones, utilizando sus bibliotecas para la carga de personajes y sus animaciones, los efectos de partículas del fuego, las luces y las sombras, las cámaras, etc.

OGREMAX

OgreMax es una colección de exportadores y visores para usar con el motor gráfico OGRE. Ogremax permite exportar tanto mallas individuales como escenas completas a OGRE desde 3DS Max y Maya. Asimismo, soporta los mismos tipos de animaciones por lo que sus visores pueden abrir cualquier tipo de recurso para OGRE para previsualizarlo antes de utilizarlo en tu programa. Casi toda la comunidad de OGRE utiliza sus importadores para cargar escenas completas y eso es exactamente lo que se ha hecho en este proyecto para la escena de la Facultad.

WIN32 API

Microsoft pone a disposición de cualquier desarrollador la interfaz de programación de aplicaciones de Windows, es decir, la API de Windows. La API es un conjunto de funciones residentes en bibliotecas que permiten al programador aprovechar los recursos del sistema operativo. Las funciones API sirven para cometidos completamente diversos, desde el manejo de errores, hasta la comunicación entre procesos, pasando por el manejo de la memoria o la obtención de información del sistema.

En esta aplicación se ha utilizado la API de Windows para generar la ventana de configuración que precede a la simulación 3D realizada con OGRE. La API hace que generar ventanas, botones y los diversos objetos de los que están formadas las interfaces gráficas en Windows sea verdaderamente sencillo. Además, Microsoft proporciona un kit de desarrollo de software en el que se incluye la documentación de la API, facilitando así el trabajo a los usuarios de ésta.

HERRAMIENTAS UTILIZADAS

Para el desarrollo de este proyecto se han utilizado varias herramientas desarrolladas por terceros. Estas herramientas son de diversos ámbitos y han tenido cada una de ellas un cometido específico. Todo ello es descrito a continuación.

VISUAL STUDIO 2010 PROFESSIONAL

Visual Studio es un IDE (entorno de desarrollo integrado por sus siglas en inglés) desarrollado por Microsoft. Se utiliza para desarrollar aplicaciones, tanto de consola como de interfaz gráfica, sitios web, aplicaciones web y servicios web. Visual Studio incluye un editor de código que soporta IntelliSense (la función de autocompletado de Microsoft) y refactorización de código. Además, viene con un debugger integrado que funciona tanto a nivel de código como a nivel máquina. Otras herramientas incluidas son un diseñador de GUI (interfaces gráficas), un diseñador web, un diseñador de clases y un esquema de base de datos.

Visual Studio soporta diferentes lenguajes de programación. Tras la instalación, están disponibles C, C++, VB.NET, XML, HTML, JavaScript y C#, pero se pueden descargar paquetes que amplían la funcionalidad de Visual Studio para soportar M, Python, Ruby y muchos otros.

Exceptuando las modificaciones realizadas en el proyecto del que se toman los ficheros de informe, todo el código de este proyecto ha sido escrito utilizando Visual Studio en su versión 2010 Professional. La razón es que Visual Studio es uno de los IDE más potentes del mercado y está completamente integrado en Windows, por lo que es la solución óptima para desarrollar herramientas para esta plataforma. Las desventajas que podrían hacer al programador decantarse por otro IDE son su precio y sus requisitos, pero ya que la descarga gratuita de Visual Studio se encontraba disponible en el repositorio de Windows para la UCM (MSDN) y que el PC disponible para el desarrollo era suficientemente potente, se descartaron opciones como CodeBlocks y se utilizó Visual Studio.

ECLIPSE

Eclipse es un IDE gratuito de código abierto, multiplataforma y expandible mediante *plug-ins*. Está escrito en su mayor parte en Java y puede utilizarse para desarrollar tanto en Java, como en otros lenguajes de programación, incluidos Ada, C, C++, COBOL, Haskell, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy, Android y Scheme.

El proyecto que desarrolla los informes representados por éste está desarrollado en Java, por lo que para las pocas modificaciones que se realizaron en él se ha utilizado Eclipse.

OGITOR

OGRE utiliza su propio formato de fichero para cargar mallas de objetos. Estos ficheros, tienen la extensión .mesh. Además, OGRE tiene su propio motor para crear terrenos a

partir de mapas de bits y sus propias herramientas para manejar el cielo o el agua. Los ficheros .mesh y estas características se pueden agrupar para formar escenas completas en ficheros .scene.

Ogitor, como su propio nombre da a entender, es un editor de escenas para OGRE. Esta herramienta tiene algunos bugs dado que está aún en desarrollo y no es demasiado ágil ni potente. De hecho, la mayoría de artistas y desarrolladores utiliza herramientas comerciales más potentes como Maya para modelar escenas para sus videojuegos o aplicaciones. Aun así, Ogitor tiene dos ventajas frente a ellas cuando trabajas con OGRE. La primera es que puedes trabajar con el formato nativo de OGRE y sin las trabas que tiene el tener que convertir constantemente de un tipo de archivo a otro. La segunda es que es una herramienta gratuita. Como la escena que utiliza este proyecto estaba ya casi acabada, no necesité realizar demasiados cambios en ella. Por eso no se necesitó una aplicación más potente y con Ogitor fue suficiente.

PARTICLE EDITOR

Para representar fuego y humo en un entorno 3d de forma realista la mejor forma es utilizar efectos de partículas. OGRE tiene su propio sistema de efectos que carga ficheros .particle. En estos ficheros .particle está definido el efecto que queremos mostrar. Aunque los ficheros .particle no son más que ficheros de texto que pueden ser editados a mano, la comunidad de OGRE ha creado y puesto a disposición de los usuarios múltiples herramientas para configurar estos efectos. ParticleEditor es una de las más completas de entre estas herramientas y, pese a no haberse actualizado desde 2006, sigue haciendo su labor perfectamente.

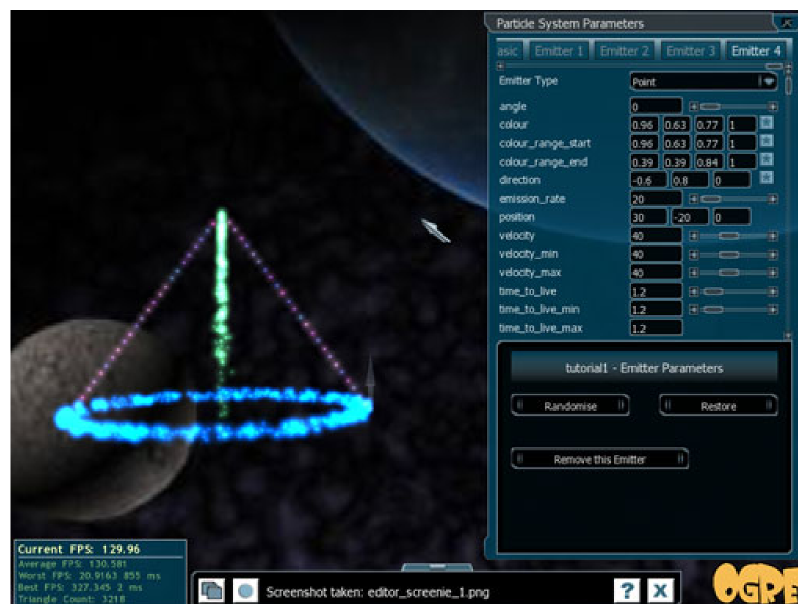

Figura 19: La herramienta Particle Editor

BLENDER

Blender es una potente herramienta de modelado 3D que funciona en diversas plataformas. Está desarrollado bajo una licencia GNU y código abierto y es completamente gratuito. Blender es una herramienta muy completa, capaz de casi todo lo que puede hacerse con otras mucho más caras como Maya. OGRE tiene a disposición de sus usuarios scripts de exportación que pueden añadirse a Blender con facilidad. De esta forma, podemos exportar al formato de OGRE (.mesh) nuestras creaciones.

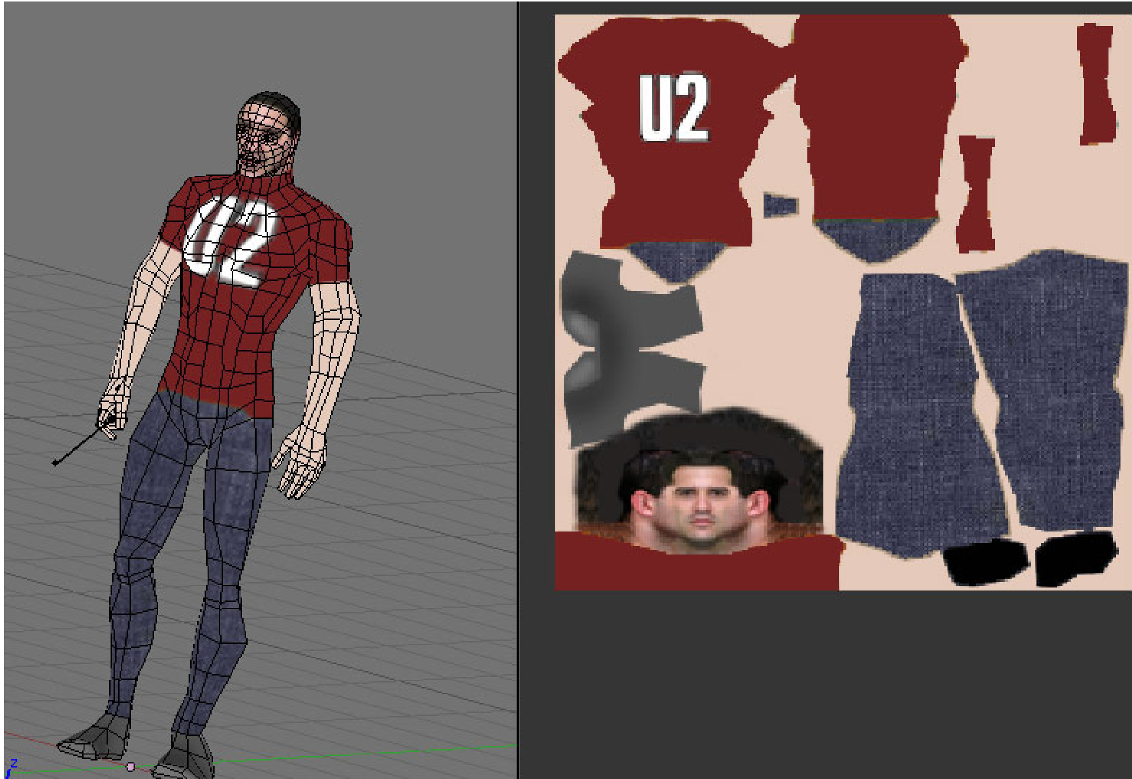

Figura 20: Uno de los personajes en Blender durante la fase de texturizado.

Los personajes han sido diseñados y animados en Blender y exportados para ser utilizados en el programa. Pese a que hay versiones más modernas y completas, la versión utilizada ha sido la 2.49b. Con versiones más nuevas existen problemas de compatibilidad con el script de exportación de Ogre, ya que los scripts funcionan sobre versiones de Python más modernas. La comunidad de Ogre está trabajando en un nuevo exportador a día de hoy, pero aún no está disponible.

6. PRUEBAS Y VALIDACIÓN

Durante la elaboración del proyecto, como veíamos en la sección de implementación, se realizaron seis fases de prototipado con sus respectivas fases de pruebas para conseguir una aplicación robusta y libre de fallos. En cada una de las fases de pruebas se probaban las nuevas funcionalidades introducidas en la fase de prototipado correspondiente. A continuación se resumen las fases de pruebas.

1ª FASE – INICIALIZACIÓN

Tras conseguir cargar y ejecutar OGRE correctamente, se probaron todas las posibles resoluciones tanto en OpenGL como Direct3D para comprobar que todas ellas funcionaban correctamente.

2ª FASE – CARGA DE OBJETOS

Esta fase de prueba fue de las más largas y laboriosas. Integrar el control de colisiones costó mucho trabajo debido a ciertas irregularidades en el entorno de la facultad. En un principio, los personajes comprobaban la distancia al suelo desde su nodo principal, pero pequeños y casi inapreciables desajustes entre dos mallas de suelo hacían que el personaje cayera por ellos al pasar de una malla a la otra. Hubo que rescribir la función numerosas veces hasta que se dio con la clave y se pudo recorrer la facultad sin problemas con el personaje, que recordemos que a estas alturas era aún un robot y se controlaba por medio del teclado.

3ª FASE – LISTA DE DESTINOS Y MOVIMIENTO

La única dificultad presentada en esta iteración fue solventar un error de división por cero que ocurría al llegar a ciertos puntos. Tras probar y probar se descubrió que el error aparecía al hacer giros de exactamente 180 grados y se encontró una rápida solución.

4ª FASE – LISTA DE FUEGO Y LISTA DE PERSONAJES

Tras implementar la cuarta fase de prototipado cuya principal introducción eran las propagaciones del fuego, se vio cómo con masas de fuego muy grandes, el programa se ralentizaba hasta niveles inaceptables de frames por segundo. Por esta razón, se decidió modificar la clase fuego, de modo que sólo los focos del frente tuvieran adjunto el efecto de partículas mientras que el resto sólo teñían el suelo de negro creando el efecto de zona quemada. De esta forma el programa funciona mucho más fluído y la sensación de estar inmerso en un incendio sigue siendo bastante realista.

5ª FASE – PARSEO DEL FICHERO .LOG Y ADAPTACIÓN DE ESCALAS.

Esta fue, sin lugar a dudas, la más larga y trabajosa de todas. La adaptación de las escalas de un proyecto al otro parecía trivial, pero resultó ser todo lo contrario. Los dos mapas con los que se trabajaba, uno por cada proyecto, no eran exactamente proporcionales y tenían ciertas variaciones que, pese a ser casi inapreciables al observador, para el programa resultaban críticas.

Tras una primera aproximación, las pruebas mostraban a personajes y focos de fuego atravesando paredes e incluso saliendo desde el segundo piso al patio del edificio. Pronto se descubrieron esas pequeñas anomalías que creaban el cambio: el grosor de las paredes y un grupo de columnas. En el proyecto de Elena Núñez [NuGo11], al trabajar sobre el plano real del edificio, las paredes tienen grosor, mientras que en el mapa tridimensional sobre el que trabaja este proyecto, las paredes son mallas de grosor cero.

Una vez identificado el problema, se generaron varios informes en los que un solo personaje debía seguir un camino que pasaba por las zonas críticas que habíamos identificado. Además, se creó otro informe con un gran fuego que, al ir tiznando el suelo al pasar, nos permitía superponer las zonas del mapa original sobre nuestra escena a base de quemarlas. Se fue adaptando la función de traslación hasta que el fuego se propagaba exactamente por las aulas y los pasillos, saliendo por las puertas y dejando las paredes sin atravesar ni tiznar. Una vez conseguido, se comprobó que en las simulaciones con agentes estos recorrían exactamente los caminos trazados en el programa que las generó.

6ª FASE – DETALLES FINALES Y DISEÑO DE PERSONAJES

En esta fase sólo surgió un error a subsanar. El identificador cuadrado que luce cada personaje a sus pies, se diseñó como un cuadrado opaco en principio, lo que hacía que el motor de colisiones lo detectara en lugar de detectar al suelo. Al quitarle la parte interna al cuadrado dejando sólo las aristas, el error quedó subsanado.

Se realizaron adicionalmente diversas pruebas con luces y sombras hasta que se encontraron las idóneas para representar tanto la facultad como el incendio de forma realista.

7. CONCLUSIONES

DISCUSIÓN DE LOS RESULTADOS OBTENIDOS

REVISIÓN DE LOS OBJETIVOS INICIALES

Durante la sección de motivación y objetivo se enumeraron los objetivos iniciales de este proyecto. Dichos objetivos se evaluarán a continuación a tenor de los resultados obtenidos.

- **Mostrar una simulación de una evacuación en caso de incendio en un entorno 3D realista basándonos en los ficheros .log del proyecto [NuGo11].**

Los informes se representan de forma adecuada y el entorno de la facultad es bastante detallado. La ambientación gráfica es buena y visualmente es fácil identificar todos los componentes de la simulación. Adicionalmente, si se cuenta con un PC con una tarjeta gráfica no demasiado potente, el programa se ejecuta de manera fluida. Si bien el autor había realizado ya aplicaciones relacionadas con la informática gráfica en 3D, ésta era la más grande a la que se había enfrentado hasta la fecha y el resultado es realmente satisfactorio.

- **Hacer que la simulación sea fácil de seguir y todos sus elementos sean fácilmente identificable durante la misma.**

Para usuarios con un PC potente, lo más cómodo es utilizar la cámara voladora e ir moviéndola a los puntos de interés para un seguimiento óptimo. Además, gracias a la perspectiva cenital podemos seguir la simulación sin perder detalle.

La funcionalidad que permite reiniciar la simulación siempre que se quiera es también útil en caso de que haya múltiples puntos de interés simultáneos en puntos opuestos de la facultad.

La creación de diversas cámaras para el seguimiento es una buena idea para ordenadores de características limitadas, ya que éstas están colocadas en lugares estratégicos para necesitar procesar el mínimo número de triángulos mostrando el máximo campo de visión.

Con un ordenador no demasiado antiguo y con una tarjeta integrada con aceleración 3D sin memoria compartida se obtendrá un rendimiento óptimo que permitirá un seguimiento adecuado de la simulación. Con un PC menos potente, para no perjudicar el rendimiento, lo mejor será utilizar las cámaras predefinidas. Su uso perjudicará al óptimo seguimiento de la simulación al tener que ir navegando por ellas para seguir la acción. Aun así, si el usuario se pierde algo, siempre podrá reiniciar la simulación tantas veces como quiera.

- **Mostrar el paso de mensajes de unas personas a otras de forma clara.**

El paso de mensajes es legible fácilmente siempre y cuando estemos a una distancia no muy lejana al remitente. El problema es que algunos mensajes tienen un destinatario concreto y no siempre es fácil deducir a quién va dirigido un mensaje dado que el programa que realiza el informe no lo especifica. En ese sentido este objetivo es mejorable.

- **Representar en una interfaz gráfica las estadísticas de la simulación para su permanente visualización.**

Durante la simulación, tendremos siempre a la vista el número de personas que hay en el edificio, cuántas han muerto a causa del fuego y cuántas han sido ya evacuadas. Podremos también ver permanentemente el número de metros cuadrados del edificio que han ardido.

RESULTADOS POSITIVOS

Pese a ser un enorme desafío en un principio –mucho más al ser realizado por una sola persona y no un equipo de tres como habitualmente- y barajarse diversas opciones que al final no se llevaron a cabo, al final el programa se orientó de una forma concreta y sus resultados fueron bastante satisfactorios.

El desarrollo de un proyecto de estas características obliga al autor a estudiar y aprender muchísimo para poder llevarlo a cabo. Durante el desarrollo del proyecto, el autor ha aprendido entre otras cosas a utilizar desde cero un motor gráfico, ha creado sockets que comunicaban proyectos en Java, Python y C++, ha leído y escrito sistemáticamente de ficheros XML, ha realizado su propio protocolo de texto para comunicar dos programas realizados en distintos lenguajes y ha utilizado diversos programas de modelado 3D.

Como conclusión positiva de todo esto queda el haber aprendido mucho y el haber adquirido experiencia en numerosos ámbitos de la informática que en las asignaturas de nuestra facultad no son tratados tan intensamente. Si bien el tiempo invertido ha sido mucho, el resultado y la experiencia han merecido la pena.

RESULTADOS NEGATIVOS

Durante el planteamiento del proyecto antes de tener claro qué se quería hacer se barajaron varias opciones que habrían enriquecido el proyecto y que no se llevaron finalmente a cabo. Introducir MAG Tools habría mejorado mucho la estética del programa. A su vez, poder realizar la ejecución de la simulación a la vez que su representación 3D, sin el paso previo de la generación del informe, facilitaría enormemente la utilización del programa a sus usuarios. La complejidad y problemas de sincronía a la hora de comunicar ambos programas desaconsejaba su comunicación, pero aún así la idea no deja de resultar atractiva.

TRABAJO FUTURO

La realización de un proyecto de Sistemas Informáticos tiene diversas limitaciones, siendo la más importante de ellas el límite de tiempo que impone un curso académico, por eso es muy posible que no se lleven a cabo todas las ideas que se querían realizar en un principio. Asimismo, también es posible que, incluso habiendo realizado todo lo que se pretendía, durante la realización del proyecto se vean cosas a perfeccionar o surjan ideas que mejorarían el resultado. A continuación se explican algunas de ellas aplicables al proyecto actual.

AMBIENTACIÓN

Si bien la ambientación del proyecto es bastante buena, especialmente sabiendo lo difícil que es crear fuegos realistas, el aspecto visual y la inmersión del usuario mejorarían con una mayor variedad de personajes e introduciendo sonido a la simulación.

En un futuro, podrían introducirse sonido, ya que la funcionalidad de OGRE puede ser ampliada con relativa facilidad con bibliotecas orientadas al uso de sonido envolvente. Además, pese al intento fallido de utilizar MAGTools, debido al constante desarrollo de OGRE y a su amplia comunidad, quizá en un futuro alguien desarrolle un buen importador de Collada y MAGTools pueda integrarse en el proyecto.

- **Paso de mensajes**

Para mejorar el paso de mensajes y conocer el destinatario de cada uno de ellos, podría realizarse un modelo visual que, mediante alguna señal, indique quién recibirá el mensaje. El videojuego de Square Enix, Final Fantasy XII, incorpora un sistema de señales visuales con forma parabólica para indicar a quién ataca o cura cada personaje. Se podría, en un futuro, implementar una funcionalidad análoga para el paso de mensajes, creando una señal que vaya del remitente al destinatario.

Figura 21: Unas líneas como éstas ayudarían a ver quién se comunica con quién de forma atractiva

- **Inclusión de la simulación en el propio proyecto**

Como la arquitectura del programa está ya diseñada e implementada, sólo habría que modificar la función update de cada tipo de agente, implementando en ella una inteligencia artificial. Con la inclusión de la toma de decisiones de cada agente dentro de este proyecto, no sería necesario leer los informes del otro para simular. De este modo tendríamos una simulación de incendios en un solo programa, mucho más robusto y más fácilmente ampliable.

La realización de esta inteligencia artificial no es trivial y su correcto desarrollo daría, quizá, para otro proyecto de Sistemas Informáticos completo. De todos modos, debido al diseño de la aplicación, las bases están puestas para poder modificarla, ampliarla y mejorarla.

8. REFERENCIAS

[BCPA03] BELTRÁN FERRUZ, P.J. DEL CERRO AGUILAR, A. CUENCA PASCUAL, D y GARCÍA ACEITUNO, G. "Motor gráfico para el desarrollo de videojuegos". Universidad Complutense de Madrid. 2003. http://eprints.ucm.es/8897/1/TC_2003-12.pdf

[BjHo05] BJÖRK, S. HOLOPAINEN, J. "Patterns in Game Design" Charles River Media, Inc. Hingham, Massachusetts 2055.

[BuSé97] BUKOWSKi, R. y SÉQUIN, C. "Interactive simulation of fire in virtual building environments" Univesity of California. 1997

[CFAST] Documentación oficial de CFAST en la web oficial del NIST http://fire.nist.gov/fds/documentation.html

[CDS11] CRUZ RUIZ, F. DURBEY CARRASCO, A. SANZ BRIONES, J. "Entorno virtual 3D multiusuario para simulación de escenarios de evacuación". Universidad Complutense de Madrid. 2011. http://eprints.ucm.es/13048/1/memoria.pdf

[DAPB08] DURIPINAR, F. ALLBECK, J. PELECHANO, N. BADLER, N. "Creating Crowd Variation with the OCEAN Personality Model" (Short paper) Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems. (AAMAS'08) Estoril (Portugal) May 12-16, 2008

[FaPr97] FAHY, R.F. y PROUXL, G. "Human behavior in the world trade center evacuation" En el International Association for Fire Safety Science, Fifth International Symposium. Páginas: 713-724. 1997.

[Ferb99] FERBER, J. "Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence" ISBN: 0201360489. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA 1999.

[Fone02] FONER, Leonard N, *What's An Agent, Anyway? A Sociological Case Study,* MIT Media Lab, 2*002* http://www.upv.es/sma/teoria/agentes/what%20is%20an%20agent-foner.pdf

[GRGG09] GALVIS RODRÍGUEZ, A.C. y GONZÁLEZ GÓMEZ, J.F. SIMPCE (Simulador de Movilidad de Personas en espacios Cerrados) Pontificia Universidad Javeriana, Bogotá, 2009. http://pegasus.javeriana.edu.co/~CIS0910TK01/documentos/pdf/Memoria_de_Trabajo_de_Grado_SIMPCE.pdf

[HaSuSi03] HAKONEN, H, SUSI, T, SIIKONEN, ML, Evacuation simulation of tall buildings, Copyright © 2003 KONE Corporation

[KCDH65] Kelley, H.H., Condry, J.C.J., Dahlke, A.E., y Hill, A.H. "Collective behavior in a simulated panic situation" Journal of Experimental Social Psychology, Número 1, Páginas: 20-54. 1965.

[Keat82] KEATING, J.P. "The myth of panic. Fire Journal" 1982.

[LE3195] "Ley 31/1995 de prevención de Riesgos Laborales" Boletín Oficial del Estado num 269 http://www.boe.es/boe/dias/1995/11/10/pdfs/A32590-32611.pdf

[LOGE] "List of game engines" Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/List_of_game_engines

[MILES] MILES: Models of Interaction centered on Language, spacE and computational Semantics. Universidad Complutense de Madrid. http://nil.fdi.ucm.es/index.php?q=node/414

[MMH10] LÓPEZ MAÑAS, E. JAVIER MORENO, F. PLÁ HERRERO, J. "Proyecto AVANTI: Sistema de asistencia a la evacuación de incendios" Universidad Complutense de Madrid. 2010. http://eprints.ucm.es/11048/1/MemoriaAVANTI.pdf

[MML10] MÉNDEZ GONZÁLEZ, J.M. MARTÍNEZ PRADO, L y LÓPEZ DE ARMAS, G. "M.A.G. Massive Avatar Generation". Universidad Complutense de Madrid. 2010. http://eprints.ucm.es/11047/1/Memoria.pdf

[NuGo11] NÚÑEZ GONZÁLEZ, E. "Modelado de evacuación de multitudes mediante agentes y transcripción de comportamientos." Trabajo Fin Máster en Ingeniería del software e inteligencia artificial. Universidad Complutense de Madrid. 2011. http://eprints.ucm.es/13495/1/memoria.pdf

[PABa07] PELECHANO, C. ALLBECK, J. BADLER, N. "Controlling Individual Agents in High-Density Crowd Simulation" ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA'07) August 3-4, San Diego (USA). 2007

[PeBa06] PELECHANO, N. BADLER, N. "Modeling Crowd and Trained Leader Behavior during Building Evacuation" IEEE Computer Graphics and Applications. vol. 26, no. 6, pp. 80-86, Nov/Dec, 2006. http://repository.upenn.edu/cgi/viewcontent.cgi?article=1288&context=cis_papers

[PSAB07] PELECHANO, N. STOCKER, C. ALLBECK, J. BADLER, N. "Feeling Crowded? Exploring Presence in Virtual Crowds" Proceedings of PRESENCE 2007. The 10th annual International Workshop on Presence. October 25-27, Barcelona (Spain). pp 373-376.

[PSAB08] PELECHANO, N. STOCKER, C. ALLBECK, J. BADLER, N. "Being a Part of the Crowd: Towards Validating VR Crowds Using Presence" Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems. (AAMAS'08) Estoril (Portugal) May 12-16, 2008

[RD2059/1981] Real Decreto 2059/1981. Aprobación de la Norma Básica de la Edificación.- Condiciones de Protección Contra Incendios en los Edificios. Modificada por Real Decreto 1587/1982, 25 de Junio. http://www.boe.es/boe/dias/1981/09/18/pdfs/A21707-21727.pdf

[RD279/1991] Real Decreto 279/1991. Aprobación de NBE-CPI/91, en Marzo 1991 y su anexo para uso comercial en Octubre de 1994. http://www.boe.es/boe/dias/1991/03/08/pdfs/A07911-07952.pdf

[RD2177/1996] Real Decreto 2177/1996. Aprobación de la norma básica de la edificación NBE-CPI/96 "Condiciones de Protección Contra Incendios en los Edificios" http://www.boe.es/boe/dias/1996/11/13/pdfs/BOE-S-1996-274.pdf

[SIAL10] SIMÓN, A. "Editor de terreno y máquina de estados para el desarrollo de videojuegos" Universidad Complutense de Madrid. 2010. http://eprints.ucm.es/11282/1/TC_2010-17.pdf

[TGMY09] THALMANN, D. GRILLON, H. MAIM, J. YERSIN, B. "Challenges in Crowd Simulation". EPFL VRlab 2009, Lausana. http://infoscience.epfl.ch/record/159013/files/CW2009.pdf?version=1

[TGMY092] THALMANN, D. GRILLON, H. MAIM, J. YERSIN, B. "Gaze Behaviors For Virtual Crowd Characters". EPFL 2009, Lausana. http://infoscience.epfl.ch/record/159013/files/CW2009.pdf?version=1

[TYPM09] THALMANN, PETTRÉ, J. MAÏM, J. YERSIN, B. "Crowd Patches: Populating Large-Scale Virtual Environments for Real-Time Applications" VRLab, 2009, Lausana http://infoscience.epfl.ch/record/134637/files/i3d2009.pdf?version=1

[TISO12] "Programming Community Index for July 2012" Tiobe Software, July 2012. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[UrMa05] MARTÍN URIBE, Humberto, Comportamiento Humano antes, durante y después de emergencias. II Congreso Nacional de Salud Mental y Asistencia Primaria en Catástrofes, Madrid, 2005. http://www.sld.cu/galerias/pdf/sitios/desastres/introduccion_psicologia_emergencia_hmarin.pdf

[Ward05] WARD, J. "Pedestrian Movement: The mathematical modeling of pedestrian dynamics." UCL Centre for Advanced Spatial Analysis. University College London. 2005.

[Watt99] WATT, A.H. "3D Computer Graphics" ISBN: 0201398559. Addison-Wesley Educational Publishers Inc. Boston, MA, USA 1999.

9. BIBLIOGRAFÍA

LIBROS

KERGER, F. "Ogre 3D 1.7 Beginner's Guide" ISBN: 978-1-879512-48-0. Copyright ©2010 Packt publishing. Birmingham, B27 6PA, UK.

MARZAL, A. GRACIA, I. "Introducción a la programación con Python" ISBN: 978-84-692-5869-9 Licencia Creative Commons Reconocimiento-NoComercial-CompartirIgual. Col-kecció Sapientia, 23. Universitat Jaume I, Castellón de la Plana, España.

EBERLY, D." 3D Game Engine Design: A Practical Approach To Real Time Computer Graphics. The Morgan Kaufmann Series in Interactive 3D Technology" Segunda edición. ISBN-13: 978-0122290633. © 2007 Elsevier Inc. San Fancisco, USA.

PORTER, A. "Programación en C++ para Windows" Primera edición. ISBN: 0-07-881881-8 © 1994 McGraw-Hill/Interamericana de España. Aravaca, España.

TORRES DEL CASTILLO, GF. "La representación de rotaciones mediante cuaterniones" Departamento de Física Matemática, Univesidad Autónoma de Puebla, México.

FOROS DE INTERNET

Foro oficial de OGRE: http://www.ogre3d.org/forums/

Foro oficial de Ogitor: http://forum.ogitor.org/

Foro de Blender artists: http://www.blenderartists.org

TUTORIALES

Tutoriales oficiales de OGRE: http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Tutorials

Tutoriales oficiales de Blender: http://www.blender.org/education-help/tutorials/

Tutorial de TinyXML: http://www.grinninglizard.com/tinyxmldocs/tutorial0.html

Tutorial de OGRE Particle Editor: http://ogre.cvs.sourceforge.net/viewvc/ogre/ogreaddons/particleeditor/documentation/PETutorial.htm

APÉNDICE A – MEJORA EN LA GUI DEL SIMULADOR MULTIAGENTE

Previamente se comentó que el proyecto que genera los informes [NuGo11] ha sido modificado para conseguir una salida estandarizada de los mismos de modo que su posterior lectura pudiera ser automatizada más fácilmente. Si bien éste ha sido el cambio principal en el mismo, se ha de mencionar también una mejora realizada a su GUI.

Aunque su principal cometido era la generación de los informes, el proyecto de Elena Núñez González [NuGo11] tenía una GUI que representaba la simulación en 2D. La GUI al inicio de las simulaciones funcionaba de forma decente, pero cuando el fuego se expandía, al no estar implementado el refresco de pantalla con doble buffer, la GUI se volvía lenta y se podían ver constantemente objetos aparecer y desaparecer.

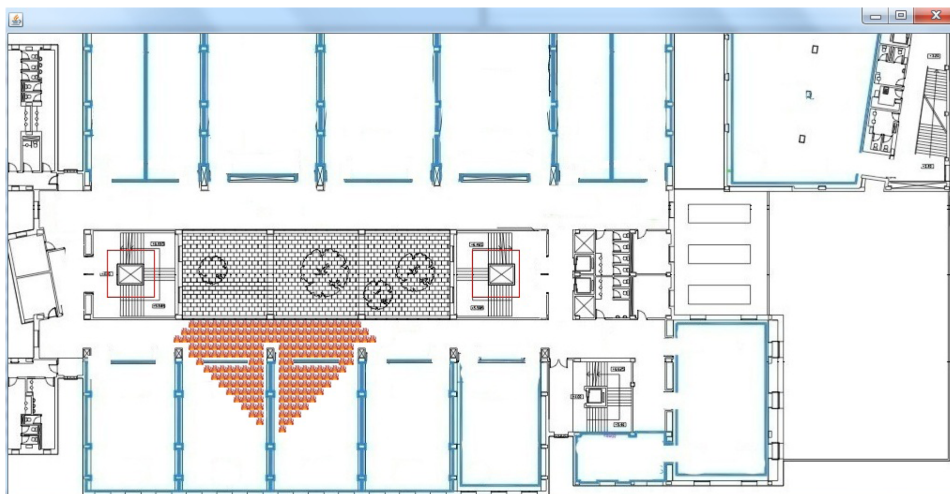

Figura 22: GUI del programa realizado por Elena Núñez González [NuGo11].

El problema fue subsanado utilizando la técnica del doble buffer, es decir, dibujando el frame en una imagen que sustituye al frame anterior una vez está completa, evitando dibujar el frame mientras éste está en pantalla.

APÉNDICE B - MANUAL DE INSTALACIÓN

- La aplicación es portable, así que basta con copiar su carpeta raíz y pegarla en la dirección elegida de su PC o ejecutarla desde el soporte en el que se encuentre.
- Los ficheros .log deben copiarse a la carpeta “log” de la aplicación. Si la aplicación se encuentra en un CD o en un soporte donde no se pueda escribir, cópiela a un disco duro o pendrive.
- Para ejecutar la aplicación, hacer doble clic sobre el fichero “Facultad3D.exe”.

APÉNDICE C – IMPLEMENTACIÓN DE FUNCIONES

A continuación se explica cómo se han implementado las funciones que intervienen en la actualización de un frame y que por tanto determinan el comportamiento de la aplicación a lo largo del tiempo.

FRAMERENDERINGQUEUED

Como se explicaba previamente en la sección sobre la arquitectura de un programa hecho con OGRE, la función frameRenderingQueued sirve para actualizar todo aquello que ocurre en la escena antes de mostrarla por pantalla en un nuevo frame. En este proyecto la función queda así:

```
bool BaseApplication::frameRenderingQueued(const Ogre::FrameEvent& evt){
```

Como observamos en la cabecera de la función, devuelve un valor booleano. En caso de devolver *false* la aplicación parará. Como parámetro tenemos los eventos propios del frame, que contienen el tiempo desde el frame anterior y desde el evento anterior.

```
  if(mWindow->isClosed())
    return false;
```

Si la ventana del programa no ha sido cerrada, continuamos con la ejecución.

```
  if(mShutDown)
    return false;
```

Si no ha sido iniciada la secuencia de cierre del programa, continuamos.

```
  mKeyboard->capture();
  mMouse->capture();
```

Capturamos las entradas del teclado y del ratón.

```
  mTrayMgr->frameRenderingQueued(evt);
```

Mandamos al gestor de ventanas que se renderice también. Esto hace que se refresquen las distintas ventanas y botones de OGRE que puedan estar abiertos.

```
  if (!mTrayMgr->isDialogVisible()){
    mCameraMan->frameRenderingQueued(evt);
```

Si no hay una advertencia tipo Dialog que paralice la ejecución, actualizamos la cámara.

```
    if (mCameraPanel->isVisible()){
    mCameraPanel->setParamValue(0, mCamera->getName());
      […]
    }
```

Si el panel con la información de la cámara está activado, lo actualizamos.

```
    if (mDetailsPanel->isVisible()){
      […]
    }
```

Hacemos lo mismo para el panel de estadísticas

```
   }
   mCharacterList->update(evt.timeSinceLastFrame,mKeyboard);
   mFireList->update(evt.timeSinceLastFrame,mKeyboard);
   return true;
}
```

Y por último mandamos a la lista de personajes y la lista de focos de fuego que se actualicen, enviándoles el tiempo que ha pasado desde el frame anterior y los eventos de teclado como parámetros, ya que, aunque no son necesarios para la aplicación final, algunos métodos de debug los utilizan.

UPDATECHARACTERLIST

La lista de focos de fuego tiene su propia función de actualización. Se detallará sólo la de los personajes ya que ambas funcionan de una manera similar. La diferencia radica en que, mientras los personajes van moviéndose por el terreno, cada foco de fuego está siempre en el mismo sitio, pero la lista va añadiendo nuevos nodos con focos activos y apagando los antiguos convirtiéndolos en ceniza.

```
void CharacterList::update(Ogre::Real elapsedTime, OIS::Keyboard *input){
```

Como veíamos en FrameRenderingQueued, recibimos como parámetros el tiempo pasado desde el frame anterior y la entrada del teclado.

```
   std::deque<Character*>::const_iterator c1_Iter;
   c1_Iter = mCharacterList->cbegin();
```

Creamos un iterador que apunta al primer objeto de la lista, es decir, al primer personaje.

```
   for (; c1_Iter < mCharacterList->cend(); c1_Iter++){
```

Recorremos la lista de personajes con el iterador uno a uno.

```
      Character* ch = *c1_Iter;
      if (ch->isDead()){
         delete ch;
         mCharacterList->erase(c1_Iter);
         dead++;
      }
```

Si ha muerto, lo eliminamos de la lista y aumentamos el contador de muertos.

```
      else if (ch->isSafe()){
         delete ch;
         mCharacterList->erase(c1_Iter);
         safe++;
      }
```

Si se ha salvado, también lo eliminamos de la lista, pero esta vez aumentamos el contador de salvados.

```
      else{
         Character ch->update(elapsedTime,input);
```

```
      }
   }
}
```

Si ni se ha muerto, ni se ha salvado, lo actualizaremos con la función que detallamos a continuación.

UPDATE

Update actualiza el comportamiento de cada personaje modificando su posición y animación a lo largo del tiempo. El fuego tiene también su propia función update, pero es mucho más simple ya que sólo necesita actualizar su efecto de partículas, por su trivialidad, sólo se detallará la función del personaje.

```
void Character::update (Ogre::Real elapsedTime, OIS::Keyboard *input){
```

Update toma como parámetros el tiempo desde el frame anterior y los eventos de teclado.

```
   if (mDirection == Ogre::Vector3::ZERO){
```

Si la dirección es un vector 0, quiere decir que el personaje ha llegado a su destino y necesita uno nuevo o que ha muerto o se ha salvado.

```
      if (nextLocation() && !dead && !safe){
         // Set walking animation
         rotateToDirection();
         mAnimationState = mEntity->getAnimationState("Caminar");
         mAnimationState->setLoop(true);
         mAnimationState->setEnabled(true);
         mAnimationState->addTime(elapsedTime);
      }
```

Si no se ha muerto ni se ha salvado, la función nextLocation() hará que mDirection tome un nuevo valor, que es el destino al que se dirigirá. RotateToDirection() rotará al personaje para que encare el nuevo destino. Por último, se inicializará la animación de caminar.

```
      else{
         mAnimationState = mEntity->getAnimationState("Caminar");
         mAnimationState->setLoop(false);
         mAnimationState->setEnabled(false);
         mAnimationState->setTimePosition(0);
      }
```

Si el personaje, ha muerto, se ha salvado o ha llegado destino final y no quedan destinos en su lista, se parará la animación y se dejará al personaje donde estaba.

```
   }
   else{
```

Si la dirección no es un vector 0, quiere decir que estamos caminando hacia un punto destino.

```
      Ogre::Real move = mWalkSpeed * elapsedTime;
      mDistance -= move;
```

Calculamos la distancia recorrida por el personaje de forma proporcional al tiempo pasado desde el último frame.

```
      if (mDistance <= 0.0f){
         mMainNode->setPosition(mDestination);
         mDirection = Ogre::Vector3::ZERO;
```

Si la distancia tras el movimiento es igual o menor que cero, hemos llegado a nuestro destino.

```
      }
      else{
         mMainNode->translate(mDirection * move);
      }
```

Si no, simplemente trasladamos al personaje a la nueva posición calculada

```
      mAnimationState->addTime(elapsedTime);
   }
```

Actualizamos la animación en consonancia al tiempo pasado.

```
   calculateY();
}
```

Por último, calculamos el valor de Y, que es la altura a la que se encuentra el personaje, ya que en caso de que estemos bajando o subiendo escaleras o una rampa, habrá que actualizarlo.

CALCULATEY

Como su nombre indica, calculateY calcula el valor de Y en el vector de posición del personaje, para colocarlo a la altura del suelo sobre el que está. Es de las funciones más complejas del proyecto y ha sido realizada gracias a la biblioteca MOC (Minimal Object Collision) ya que nos permite usar funciones de *raycasting* que trazan rayos y desde un punto en una dirección determinada, y devuelven el primer objeto atravesado y su distancia a él. De este modo podemos lanzar un rayo perpendicular al suelo, ver a qué altura de él estamos y obrar en consecuencia.

```
void Character::calculateY(){
   Ogre::Vector3 results = Ogre::Vector3::ZERO;
   Ogre::Entity *tmpE = NULL;
   Ogre::Real distToColl = 0;
   Ogre::Real x = mMainNode->getPosition().x;
   Ogre::Real z = mMainNode->getPosition().z;
   Ogre::Real colY = 1000;
```

Inicializamos las variables auxiliares de la función. *tmpE es un puntero que apuntará al primer objeto atravesado por el rayo. DistToColl será la distancia a la que esté ése objeto. X y Z son las posiciones iniciales, y colY toma un valor grande, ya que luego se irá quedando con la colisión que esté a menor distancia de las que se produzcan.

```
   if(col->raycast(
      Ogre::Ray(Ogre::Vector3(x,mMainNode->getPosition().y+40, z),
Ogre::Vector3::NEGATIVE_UNIT_Y),
```

```
      results,
      tmpE,
      distToColl,
      Ogre::SceneManager::ENTITY_TYPE_MASK
   )){
      Ogre::String tmp = "";
      if (tmpE != NULL) tmp = tmpE->getName();
```

Trazamos un rayo desde 40cm de altura hacia abajo en la dirección negativa del eje Y.

```
      if (tmp != "") {
         if (tmp.compare(mName + "msg")!=0)
               colY = distToColl;
         else
               colY = 1000;
      }
      else colY = 1000;
   }
```

Si hemos tocado algo que no sea nuestro propio mensaje sobre la cabeza del personaje, calculamos su distancia y la asignamos a colY.

```
   if (colY >= 1000){
      Ogre::Real col2Y = 1000;
```

Si hemos tocado algún objeto muy lejano, reinicializamos colY a 1000 unidades (10m).

```
      if(col->raycast(
         Ogre::Ray(
               Ogre::Vector3(x,mMainNode->getPosition().y+40,z) + mMainNode-
>getOrientation() * Ogre::Vector3::UNIT_X,
               Ogre::Vector3::NEGATIVE_UNIT_Y
         ),
         results,
         tmpE,
         distToColl,
         Ogre::SceneManager::ENTITY_TYPE_MASK
      )){
```

Debido a que a veces, por la construcción de la escena, puede haber dos mallas de suelo con un pequeño hueco entre medias, trazamos otro rayo desde un centímetro más adelante. Nos quedaremos con el valor de colY más bajo.

```
         Ogre::String tmp = "";
         if (tmpE != NULL) tmp = tmpE->getName();
         if (tmp != "") {
               if (tmp.compare(mName + "msg")!=0)
                     col2Y = distToColl;
               else
                     col2Y = 1000;
         }
         else col2Y = 1000;
      }
      if (colY > col2Y) colY = col2Y;
   }
```

Una vez tenemos los datos necesarios, comenzamos con los cálculos para conocer nuestra posición respecto al suelo.

```
   if (colY < 200){
```

```
    mMainNode->setPosition(Ogre::Vector3(x,mMainNode->getPosition().y -
colY + 40,z));
  }
}
```

Si la distancia a colY es menor de dos metros, significará que hemos subido menos de 40cm o que hemos bajado menos de 160cm. Debido a la escena con la que trabajamos, en la que no hay escalones mayores de 40cm, no consideraremos otras distancias.

ROTATE TO DIRECTION

Una vez se llega a uno de los puntos de destino, hay que rotar al personaje de modo que encare al nuevo para comenzar a andar hacia él. Esta función, pese a su aparente simplicidad, no es trivial ya que tiene que ver con matemáticas de cuaterniones.

```
void Character::rotateToDirection(void) {
  Ogre::Vector3 src = mMainNode->getOrientation() * Ogre::Vector3::UNIT_Z;
```

Obtenemos el vector de la dirección final al multiplicar el cuaternión de orientación por un vector unitario (0, 0, 1). Esta es la dirección en la que debe mirar nuestro personaje al terminar de ejecutarse la función.

```
  src.y = 0;
  src.normalise();
```

Quitamos al vector obtenido su valor de y, ya que queremos un vector paralelo al plano suelo, y lo normalizamos.

```
  Ogre::Vector3 dir = mDirection;
  dir.y = 0;
  dir.normalise();
```

Tomamos el vector de dirección actual del personaje y retiramos su valor de y. Normalizamos.

```
  if ((1.0f + src.dotProduct(dir)) < 0.0001f){
    mMainNode->yaw(Ogre::Degree(180));
  }
```

Si la distancia a la que giramos es de exactamente 180 grados, puede provocarse al girar una división por cero, por ello para este caso utilizamos la función yaw de OGRE que gira un objeto en torno al eje Y.

```
  else{
    Ogre::Quaternion quat = src.getRotationTo(dir);
    mMainNode->rotate(quat);
  }
}
```

En caso de no encontrarnos con ese problema, utilizamos la función getRotationTo para obtener el número de grados que hay que rotar el nodo para hacerlo encarar la dirección adecuada. Una vez obtenido el valor necesario, procedemos a rotar el nodo.

APÉNDICE D - MANUAL DE USUARIO

CONFIGURAR LA SIMULACIÓN

Tras ejecutar el fichero Facultad3D.exe aparecerá en pantalla la ventana de configuración. En ella seleccionaremos el fichero.log a simular. La propia aplicación mostrará los ficheros disponibles, que serán los que se encuentren en la carpeta raíz de la aplicación.

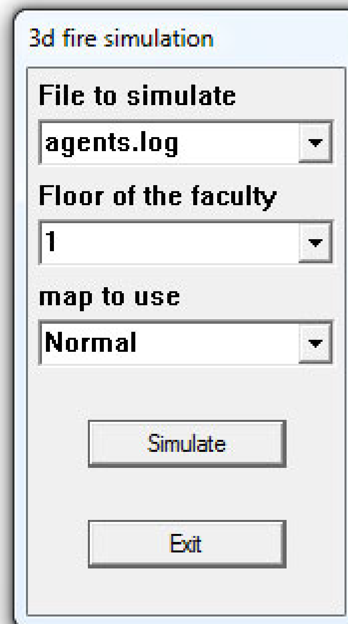

Figura 23: Ventana de simulación

Tras seleccionar el fichero, indicaremos en qué piso de la facultad se desarrolla la simulación y el mapa a utilizar, a elegir entre uno más detallado y otro con menos detalle para ordenadores más lentos.

Tras elegir las tres opciones en las tres pestañas correspondientes, hacemos clic en “Simulate” para pasar a la siguiente fase.

CONFIGURAR EL ENTORNO GRÁFICO

Aparecerá ahora una nueva ventana en la que configuraremos las opciones gráficas a utilizar. Podremos elegir entre OpenGL y Direct3D, la resolución y codificación del color, si ejecutar a pantalla completa o en ventana…

Recomendamos ajustar la resolución a la potencia del ordenador a utilizar. Con un PC que tenga una GPU potente, podremos utilizar una resolución mayor que con un PC con una Tarjeta gráfica más pobre.

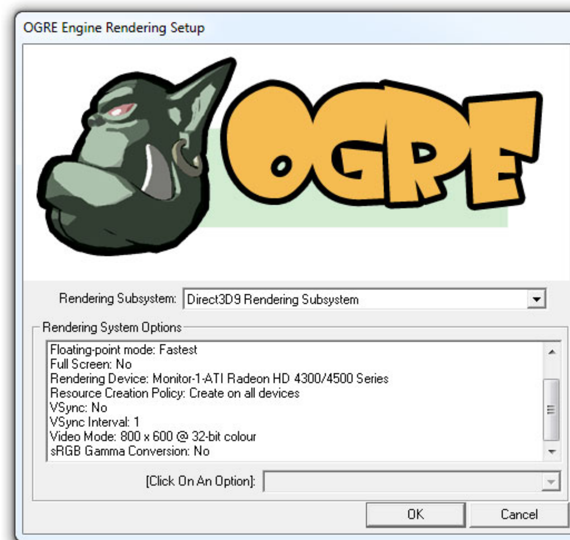

Figura 24: Pantalla de configuración del entorno gráfico.

La simulación debería funcionar indistintamente tanto con OpenGL como con Direct3D, pero si hay problemas con alguno de ellos siempre puede probarse a ver si funciona con el otro sistema. Si ninguno funciona, lo más probable es que haya que actualizar algún driver de la tarjeta gráfica.

Una vez hayamos seleccionado las opciones gráficas, que quedarán guardadas para la ejecución siguiente, bastará con hacer clic en OK para iniciar la ejecución de la simulación.

CONTROL DE LA SIMULACIÓN

Lo primero que veremos es la pantalla de carga y, una vez termine de cargar el programa, comenzará la simulación. La pantalla de carga muestra la carga de recursos de OGRE, pero no tiene en cuenta la carga de la escena. Por ello, la pantalla quedará con la barra de carga llena durante unos instantes sin movimiento aparente, pero el programa no está parado si no que está cargando la escena.

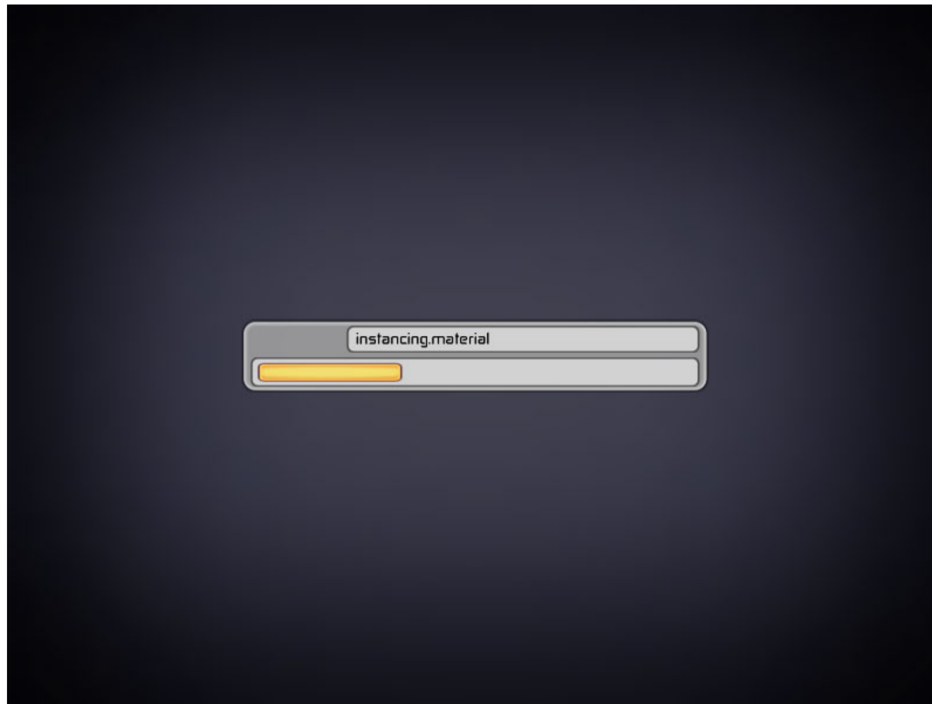

Figura 25: Pantalla de carga de la aplicación

Durante la simulación, el control de la cámara que esté seleccionada, se realizará con los cursores y el ratón. El resto de opciones se enumeran a continuación:

Figura 26: Función de las teclas durante la simulación

- **Control de la simulación**
  - Terminar la simulación: Esc.
  - Reiniciar la simulación: R.
- **Control de las cámaras**
  - Vista cenital: T.
  - Mover una cámara predefinida: Y.
  - Cámara predefinida siguiente: AvPág.
  - Cámara predefinida anterior: RePág.
  - Ampliar horizonte: +.
  - Reducir horizonte: -.
- **Paneles de información**
  - Mostrar/Ocultar estadísticas de Frames por segundo: Z.
  - Mostrar/Ocultar estadísticas de la simulación: X.
  - Mostrar/Ocultar datos de la cámara actual: C.
  - Mostrar/Ocultar las opciones gráficas avanzadas: V.