Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

JUEGO DE MESA 3D INTERACTIVO

Alumno: **Irene Mª Martínez Cruz**

Tutores: Prof. D. José Manuel Pérez Lorenzo
Prof. D. Raquel Viciana Abad

Depto.: Ingeniería de Telecomunicación

Febrero, 2016

1. RESUMEN

Los videojuegos son en la actualidad el sector con mayor movimiento económico en la industria del entretenimiento, por encima incluso del cine. Cada vez tienden más al desarrollo de entornos más realistas no sólo desde el punto de vista gráfico sino también en la capacidad de interacción que le ofrecen a los usuarios. Desde el punto de vista de los juegos de mesa, surge además la posibilidad de convertir a estos productos en herramientas que mejoren la interacción entre personas a través de la red. Integrando en estos juegos la sensación, cada vez más realista, de estar con una persona con la que se está interaccionando de forma remota, lo que se denomina como copresencia.

En el presente trabajo se ha desarrollado una aplicación cliente/servidor, implementando el juego de mesa Conecta 4. El objetivo principal es el de analizar las necesidades tecnológicas del juego atendiendo al consumo de recursos de red y de computación gráfica en un entorno que permite la interacción de dos personas favoreciendo la copresencia. Este juego está compuesto por dos escenas que implementan un entorno virtual, donde la principal escena muestra el tablero en el que interactúan ambos jugadores mediante la colación de fichas 3D en tiempo real, con el uso de protocolos de comunicaciones. En la segunda escena un jugador será representado por un avatar, y el otro mediante una imagen de webcam. Se hará un estudio valorando los resultados obtenidos y su posterior conclusión. Las tecnologías usadas son: lenguaje C++, motor gráfico OGRE, librería de procesado de imagen OpenCV y librería de comunicación por sockets.

2. INTRODUCCIÓN

Las siguientes páginas sirven para introducir al lector del Trabajo Fin de Grado (TFG) en el tema abordado en dicho trabajo. Por este motivo se hablará de los comienzos y de la evolución del sector de los videojuegos a lo largo de las últimas décadas, atendiendo a los principales aspectos que son involucrados en este trabajo: el crecimiento experimentado debido al aumento de la capacidad de procesamiento en los equipos, y el gran desarrollo en los gráficos de los videojuegos, pudiéndose crear hoy en día imágenes cada vez más reales. Además, de su inversión en otros campos, como en la virtualidad aumentada o Augmented Virtuality, provocando que cada vez se analice más el grado de involucración de un jugador con el contenido, por lo que se hace más interesante el incluir la figura de un avatar o un elemento real que recree a los posibles participantes del juego. En este sentido también se hace más necesario el uso de protocolos que permitan transmitir información de vídeo en tiempo real o de una animación asociada con movimientos reales de una persona y de técnicas que permitan integrar esta información en simulaciones que ya de por si suelen tener un elevado coste computacional.

2.1 ANTECEDENTES

2.1.1 Primeros Videojuegos

Los videojuegos comenzaron su evolución hacia finales de la década de los 40 a raíz del fin de la Segunda Guerra Mundial, donde las principales potencias empezaron una carrera tecnológica en el desarrollo y construcción de las primeras supercomputadoras programables.

No se pudo esclarecer con exactitud cual fue el primer videojuego, pero finalmente se consideró el Nought and Crosses como el primero de la historia, también conocido como OXO, desarrollado por Alexander S.Douglas en la Universidad de Cambridge en 1952 como parte de su tesis doctoral, el cual consistía en una versión del tres en raya que enfrentaba al jugador con la máquina.

Figura 1. Juego OXO

Más tarde, en el año 1958 William Higginbothamun científico del Brookhaven National Laboratory en Nueva York desarrolló Tennis for Two (tenis para dos) el primer juego para dos jugadores humanos, lo creó a partir de un programa de cálculo de trayectorias y un osciloscopio.[21]

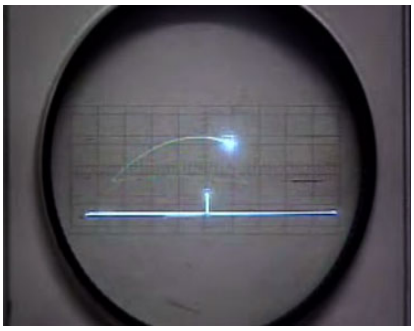

Figura 2. Juego Tennis for two.

Se desarrollaron otros juegos, pero pocos consiguieron salir del ámbito universitario en el que se habían creado.

Ya en 1972, con la llegada del Pong y su inmenso éxito se reestructuró por completo el negocio del entretenimiento. Fue considerada la versión comercial del juego Tennis for Two de Higginbotham. El juego fue diseñado por Al Alcom para Nolan Bushnell en la recién fundada Atari.

Figura 3. Juego Pong.

En 1973 quince compañías se habían lanzado al negocio de los videojuegos, un negocio que un año antes estaba exclusivamente en manos de Atari. Los videojuegos de estas compañías no dejaban de ser simples copias del Pong, mientras que la compañía que había creado el juego original seguía aportando nuevas innovaciones.

Los primeros ordenadores personales comenzaron a aparecer en esa época, pero al principio carecían de monitor, y el único dispositivo de visualización eran las lentas impresoras que los usuarios podían conectar a sus ordenadores .Por lo que estas limitaciones imponían obstáculos al desarrollo de nuevos videojuegos.

2.1.2 Videojuegos con Realidad Virtual

La idea de poder desarrollar un mundo virtual y más real se remonta a1965 cuando el profesor Ivan Sutherland creó un dispositivo que supuso ser clave para el desarrollo de la idea de realidad virtual, así como posteriormente de realidad aumentada, consistía en un sistema HMD [1](human mounted display*)* bastante primitivo, tanto en el realismo como en el interfaz de usuario.

Los gráficos se componían de un entorno virtual que simulaba una habitación. Tenía incorporado un sistema de seguimiento de cabeza, ya que la perspectiva que el software mostraba al usuario dependía de la posición en la que se mirase. El peso del HMD de Sutherland, y la necesidad de realizar un seguimiento de los movimientos de la cabeza hizo necesario que el HMD estuviera conectado a un brazo mecánico suspendido del techo del laboratorio. [25]

[1]Es un dispositivo de visualización similar a un casco, que permite reproducir imágenes creadas por

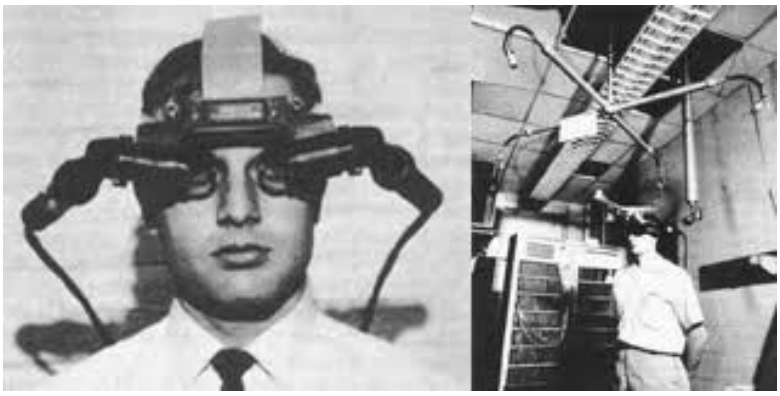

Figura 4. HMD de Sutherland.

Los primeros videojuegos que incorporaban gráficos rudimentarios en 3D fueron Tailgunner(1979) y Battlezone(1980), usaban gráficos lineales para delinear el contorno de los objetos dando así la ilusión de profundidad.

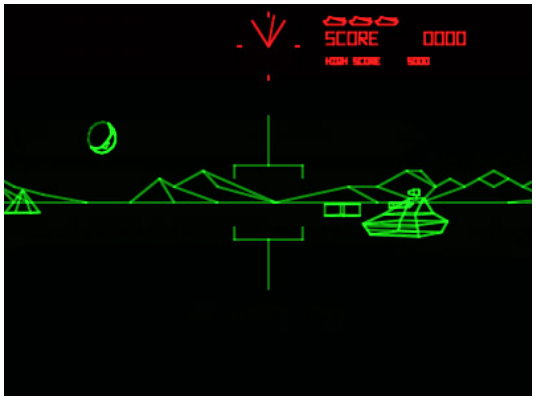

Figura 5. Gráficos Juego Battlezone.

La utilización de gráficos en 3D dejaba mucho que desear, por lo que la utilización de gráficos en 3D sobre ordenador tuvo que esperar algún tiempo más.

A principios de los 80's un grupo de investigadores crearon el primer mapa interactivo virtual de la ciudad de Aspen. Para el desarrollo de este mapa se emplearon cuatro cámaras, tomaban una foto cada tres metros y luego se reproducían a 30 fps. En esta misma época se reconocía el concepto de realidad virtual o Artificial Reality como una nueva tecnología.

En 1982, se recuperó la idea de Ivan Sutherland y su HMD. SEGA lanzó el Subroc-3D. Esta máquina tenía integrada un visor binocular con asas para agarrarlo, imitando al periscopio de un submarino.

Figura 6. Juego Subroc-3D.

Más tarde, en 1983 llegó el primer periférico virtual para una consola. Se llamaba 3D Imager, se creó para la consola de principios de los ochenta GCE Vectrex.

Figura 7. Juego 3D Imager.

Hubo varios lanzamientos más por parte de otras empresas de nuevas gafas de realidad virtual. La empresa VictorMaxx sacó al mercado el StuntMaster VR para SEGA. Consistía en unas gafas con realidad virtual interactiva con capacidad de rastreo de movimientos, estos movimientos eran registrados por una varilla vertical que se sujetaba al hombro. Cuando movías la cabeza se detectaba el movimiento de la varilla y este movimiento era interpretado en movimientos virtuales en la pantalla.

Figura 8. Gafas StuntMaster.

Hasta ese momento todas las tecnologías que incorporaban realidad virtual carecían de realismo, y suponían una gran inversión por parte de las compañías a la hora de poder perfeccionar y desarrollar nuevas técnicas en este campo, por lo que ésta tecnología quedó aplicada en campos de investigación.

Ya con la llegada de los noventa los avances tecnológicos de los sistemas de la época devolvieron la idea de retomar la concepción de crear un mundo virtual en tres dimensiones y proyectarlo en una pantalla de dos dimensiones.

En 1991, la empresa Virtuality comercializó su última producción en las salas de juegos, Dactyl Nightmare. Aún así este tipo de dispositivo no tuvo éxito, debido todavía a la baja calidad que tenía, por lo que las empresas se empezaron a centrar en el desarrollo y la mejora del entorno virtual en las escenas de los videojuegos, dejando de lado a este tipo de dispositivos.

Figura 9. Juego Virtuality.

Doom fue un videoojuego desarrollado en 1993 que sacudió la industria de los videojuegos ya que destacaba por sus gráficos y la solidez de su motor gráfico. Se convirtió asimismo en el primer videojuego de la historia que permitía oficialmente elmodding, es decir, la modificación de su diseño y sus niveles por parte de los jugadores. Además popularizó enormemente los videojuegos en red, bien a través de Internet, o mediante conexiones por cable. Convirtiéndose así en una pieza clave para el desarrollo de la industria de los videojuegos.

Figura 10. Escena del Videojuego Doom.

2.1.3 Videojuegos en Red

Para hablar de juegos con conexiones compartidas multijugador masivas debemos remontarnos a la década de 1970 donde el Empire (1973) y un año más tarde el Spasim (1974), hicieron uso por primera vez de esta característica. Ambos

corrían en la red PLATO System de la Universidad de Illinois que constaba de un superordenador y cientos de terminales repartidas por EEUU y algún país extranjero.

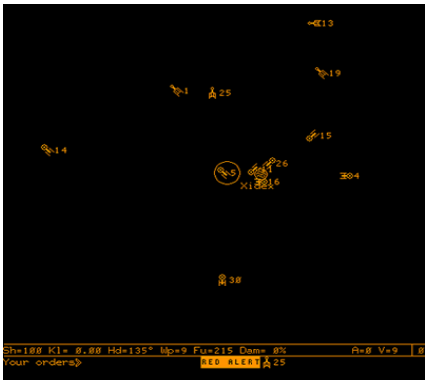

Figura 11. Juego Empire.

Aunque la idea de videojuegos en red no se concibió hasta 1979, cuando un grupo de estudiantes de la Universidad de Essex crearon una versión informática del célebre juego de rol Dungeons & Dragons. Esta versión electrónica era multiusuario y estaba basada en el uso de textos alfanuméricos.

Pero la verdadera revolución de los juegos en red surgió en 1993 con la creación de la World Wide Web. Los usuarios tenían la posibilidad de acceder gratuitamente a versiones reducidas de videojuegos para ordenador con fines básicamente promocionales, como es el caso del ya mencionado videojuego Doom. Además, la rápida difusión de Internet como medio de entretenimiento facilitó la mejora de las tecnologías para la conexión en red de usuarios y su acercamiento a la sociedad.

2.1.4 Transferencia de Vídeo en los Videojuegos

El invento revolucionario que abrió camino a la inclusión de complementar una realidad virtual con información real fue la webcam.

La primera webcam fue creada en 1991, curiosamente se desarrolló en el departamento de informática de la Universidad de Cambridge para comprobar el nivel de café en una cafetera sin tener que levantarse. Al protocolo de comunicación lo denominaron XCoffe y tras unos meses fue comercializado. En 1992 salió a la venta la primera webcam, XCam.[24]

Hasta el año 2000 no se presentó el primer dispositivo de cámara digital para emplearse en una videoconsola. La empresa SEGA desarrolló Dreameye, una cámara digital diseñada para ser usada como una webcam para la videoconsola Dreamcast. Solamente fue lanzada en Japón.

Figura 12. Cámara Dreameye de SEGA.

El primer prototipo de cámara que se creó para poder interactuar con tu propia imagen dentro del entorno de un juego fue el EyeToy, desarrollado por Sony en 1999. Tuvo gran éxito, por lo que la compañía de Sony siguió investigando en este campo, siendo pioneros en juegos para el fitness, en la creación de avatares con las imágenes de las caras de los jugadores y en el reconocimiento gestual y de objetos con los queel propio jugador podía interactuar, de esta forma también se le añadía al juego el concepto de virtualidad aumentada.

Figura 13. Juego lanzado en 2004 que empleaba el Eyetoy.

2.2 DESCRIPCIÓN DEL TRABAJO DESARROLLADO

Como ya se ha mencionado al principio de este trabajo se ha desarrollado un juego de mesa 3D interactivo, el cual consta de un entorno 3D donde los usuarios pueden interactuar entre sí. El juego está compuesto por dos partes: una aplicación servidor y una aplicación cliente, diferentes en relación con la comunicación pese a que a nivel de la interfaz no hay diferencia.

Cómo resultado final el juego constará de dos escenas, una donde se representa el tablero en el cual ambos jugadores podrán interactuar posicionando sobre él las fichas del juego, mientras que en la otra escena se mostrará la representación de un avatar o la imagen en tiempo real del contrincante.

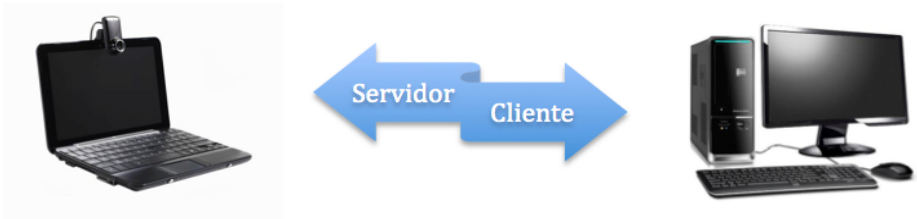

Figura 14. Esquema proyecto.

El trabajo consta de cuatro fases:

1. **Creación de un entorno virtual con Ogre.**
   En esta fase se implementan los elementos 3D que conforman el entorno virtual con el software adecuado.

2. **Implementación de un entorno virtual.**
   Utilizando los modelos diseñados para crear un entorno virtual 3D, mediante el uso de las funciones propias de una librería para el desarrollo de videojuegos y entornos tridimensionales de nivel medio, se le va proporcionar funcionalidad al juego.

3. **Desarrollo de un modelo cliente/servidor.**
   Basándose en la arquitectura Cliente Servidor y haciendo uso de la programación con Sockets para distintos protocolos.

4. **Implementación de transmisión de vídeo mediante una webcam.**
   Con los dispositivos adecuados y usando las librerías necesarias para la captación y transmisión de video. Además sumando la incorporación de una hebra específica para la transmisión de estos datos.

3. OBJETIVOS

El objetivo del proyecto es el desarrollo de una aplicación basada en un modelo cliente/servidor que implemente un juego de mesa interactivo. En particular los jugadores deberán estar representados mediante un avatar y la imagen obtenida por una webcam. El proyecto consistirá en desarrollar un juego 3D en C++ y en transmitir en tiempo real la información que permita la actualización del estado del juego y del resultado de la interacción del usuario.

Partiendo de la base del objetivo principal se puede desglosar en objetivos más específicos, los cuales se muestran a continuación:

- Familiarizarse con las funciones y características de OGRE, que permiten la creación de aplicaciones 3D.
- Creación de un entorno tridimensional programado en lenguaje C++ haciendo uso de las funciones que proporciona la API de OGRE.
- Los componentes del escenario serán proporcionados y animados mediante una herramienta de código abierto para el diseño de gráficos 3D, como es Blender.
- Conocer herramientas que incrementan el realismo de una simulación interactiva, como la inclusión de capturas de video en el entorno 3D.
- Analizar y determinar las limitaciones de una aplicación multiusuario interactiva en red.
- Familiarizarse con las funciones y características de OpenCV, que permiten la captura de video.
- El usuario será capaz de cambiar el punto de vista de la escena 3D, para ello se utilizará la API OIS (Output Input Standard) que provee de funciones para poder interactuar por medio de un dispositivo de entrada y salida como el teclado o el ratón.
- Implementar la comunicación necesaria ente el cliente y el servidor de la aplicación.

4. ESTADO DEL ARTE

En este apartado se realiza un estudio previo sobre el estado actual de las herramientas disponibles en la actualidad, para el desarrollo de este trabajo. Entre los grupos de elementos analizados encontramos:

- Motores gráficos 3D (*GameEngines*).
- Librerías para el desarrollo de videojuegos.
- Modeladores de gráficos 3D.
- Librerías de física.
- Librería de video.
- Comunicaciones en red.
- Casos de estudio.

4.1 MOTORES GRÁFICOS 3D

El motor gráfico es la parte de un juego que controla, gestiona y actualiza los gráficos 3D en tiempo real. Entre los motores más utilizados destacan el del Quake III y el de Unreal Tournament.

Debido a que los gráficos 3D se han convertido en algo muy utilizado, sobre todo en juegos de ordenador, se han creado API[2]s especializadas que facilitan el proceso de creación de los mismos, mediante entidades gráficas elementales, como pueden ser líneas, polígonos y texturas. Gracias a ellas los programadores acceden de manera abstracta a las funcionalidades del hardware gráfico del computador.

Un motor gráfico ofrece una gran cantidad de componentes reutilizables que pueden ser manipulados para la creación de un juego, como los encargados de la carga de modelos, la visualización, la animación o la detección de colisiones.

[2]Interfaz de programación de aplicaciones: es el conjunto de funciones y procedimientos o métodos, en la programación orientada a objetos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Durante mucho tiempo, las empresas creaban sus propios motores gráficos, pero los costes de este proceso, en la actualidad, han crecido significativamente, provocando que las compañías desarrolladoras de este tipo de software cambien su filosofía hacia la especialización. Así aparece el término middleware[3], en el que la empresa no provee a sus clientes del motor gráfico al completo, si no de alguno de los componentes según la necesidad, a un precio más razonable. En la siguiente figura se muestra esta división y la posible configuración de elementos necesarios para crear un videojuego.

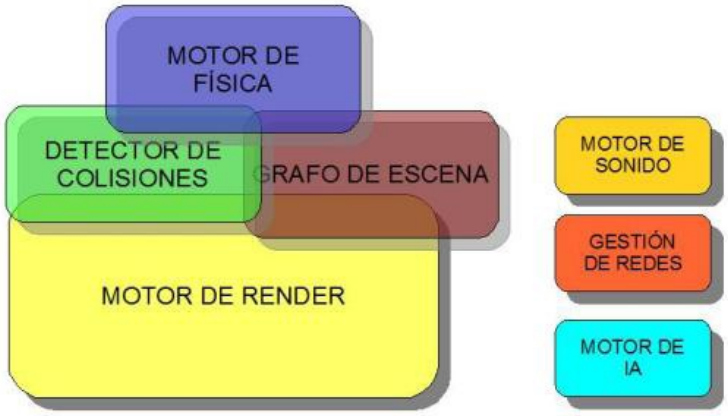

Figura 15. Elementos para crear un videojuego.

Existen muchos tipos de motores para videojuegos, atendiendo a múltiples criterios de clasificación. En este apartado solo se comentarán tres clases según el nivel de conocimientos de programación necesario para su utilización.

- **Roll-your-own game engines** (bajo nivel): a pesar del coste muchas compañías continúan desarrollando sus propios motores, utilizando los interfaces de aplicación disponibles, como DirectX u OpenGL. Además permiten la inclusión de otras bibliotecas que faciliten la programación. En general, dan mayor flexibilidad a los programadores, permitiéndoles elegir los componentes que desean integrar, pero también conlleva un mayor tiempo invertido en el desarrollo.
- **Mostly-ready game engines** (nivel medio): tienen herramientas desarrolladas que simplifican aspectos básicos de la creación de videojuego, como puede ser la representación. Necesitan menos conocimientos de programación que los anteriores, aunque pueden limitar

[3]Software que asiste a una aplicación para interactuar o comunicarse con otras.

las acciones del creador. Son los más utilizados en la actualidad, proporcionando un gran rendimiento con menos esfuerzo.

- **Point-and-click engines** (alto nivel): son cada vez más comunes. No necesita conocimientos de programación, ya que incluye una completa cadena de herramientas que le permite crear un juego únicamente haciendo clic. Están diseñados para tener un interfaz amigable con poca codificación. El problema es que limitan mucho las posibilidades de diseño y generalmente solo permiten hacer uno o dos tipos de géneros de juegos o modos gráficos. Por otra parte, consigue resultados muy rápidamente y con pocas horas de trabajo.

Las APIs para los motores de juegos más populares son:

- **OpenGL**(Open Graphics Library): es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes. Fue desarrollada por Silicon Graphics Inc. en 1992 y se usa en aplicaciones en CAD[4], realidad virtual, representación científica, visualización de información y simulación de vuelo. También se usa en desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Windows.
- **Direct3D**: es parte de DirectX[5], propiedad de Microsoft. Consiste en una API para la programación de gráficos 3D. Está disponible tanto en los sistemas Windows de 32 y 64 bits, como para sus consolas Xbox y Xbox 360. Se usa principalmente en aplicaciones donde el rendimiento es fundamental, como en los videojuegos, aprovechando el hardware de aceleración gráfica disponible en la tarjeta.
- **RenderMan**: es propiedad de Pixar. Ha sido especialmente diseñado para satisfacer los retos demandados por la animación 3D y los efectos visuales. Se caracteriza por su rapidez, eficiencia, y permite manejar una sorprendente cantidad de figuras geométricas complejas para conseguir efectos más realistas.

[4]Diseño asistido por ordenador, hace referencia a la utilización de herramientas computacionales para el diseño.

[5] Conjunto de bibliotecas para multimedia.

4.1.1 Técnicas Utilizadas en los Motores Gráficos

Existen infinidad de técnicas empleadas por los motores gráficos y se siguen desarrollando nuevas, en esta sección se detallan algunas de las más conocidas.

- **Árboles BSP**: Es una estructura de datos usados para organizar objetos dentro de un espacio. Tiene aplicaciones en la remoción de áreas ocultas y en el trazado de rayos.

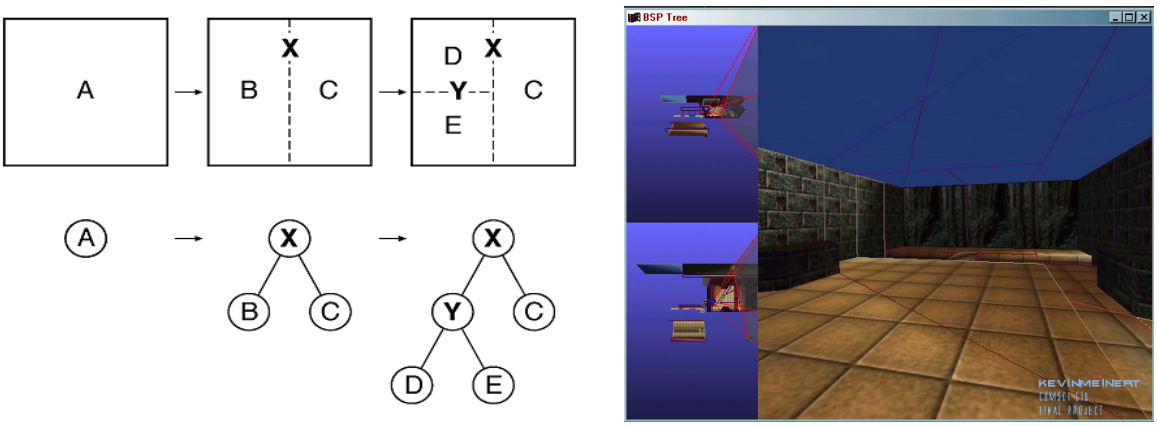

Figura 16. Técnica árboles BSP.

- **Radiosidad:** Técnica para el cálculo de la iluminación global de un ambiente cerrado. La idea en la cual se basa esta técnica es en buscar el equilibrio de la energía que es emitida por los objetos emisores de luz y la energía que es absorbida por los objetos en el ambiente.

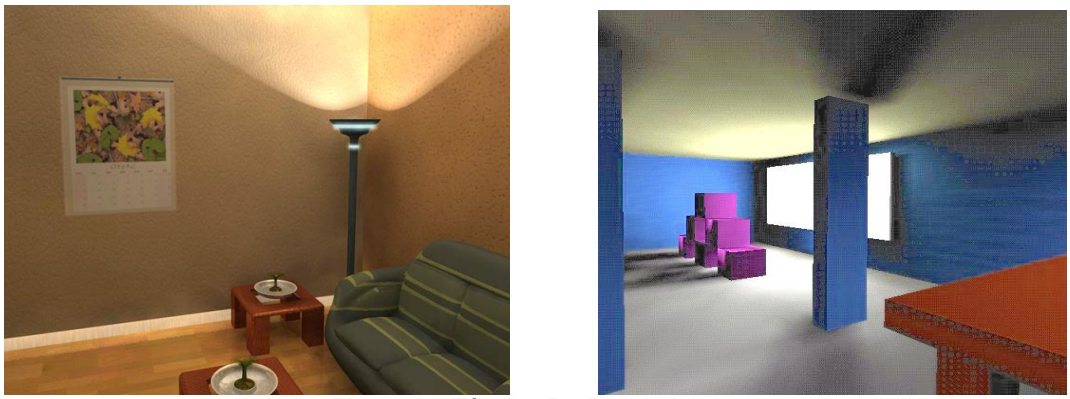

Figura 17. Técnica Radiosidad.

- **MipMapping:** Técnica de manejo de texturas que cambia la textura de un polígono en un objeto 3D dentro de un juego según el ángulo de vista del jugador o las condiciones del juego. Conforme nos acercamos a un objeto, éste gana calidad. De esta forma los objetos alejados tendrán un nivel de resolución bajo y objetos cercanos alto, consiguiendo así un mayor rendimiento.

Figura 18. Técnica MipMapping.

- **Phong y Gourand:** Ambas son técnicas de iluminación donde Gourand aproxima la superficie de los polígonos a una superficie curva, calculando las intensidades de los vértices, yPhong interpola las normales en lugar de las intensidades.

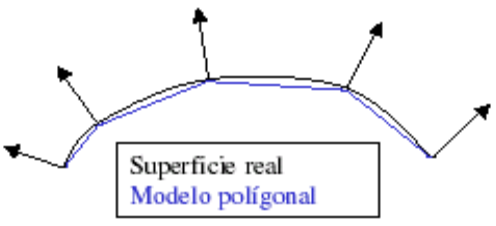

Figura 19. Técnica Gourand.

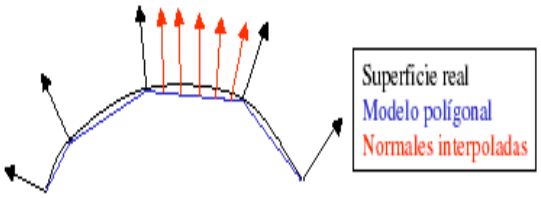

Figura 20. Técnica Phong.

- **Bump-Mapping:** Se utiliza para agregar el detalle a una imagen sin aumentar el número de polígonos. Se basa en algoritmos que captan la textura inicial de la imagen y la convierten en otra. Crea pequeños Bump-mapping en la superficie del objeto para darle texturas sin cambiar la superficie del objeto.

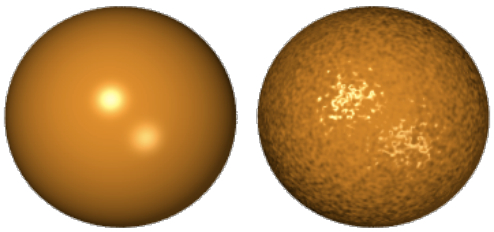

Figura 21. Técnica Bump-Mapping.

- **Lightmaps**: Esta técnica se empezó a usar en el 1996, y la crearon la gente de Id Software en el Quake. Los lightmaps (mapas de luz) simplemente consisten en añadir una segunda textura a todas y cada una de las caras existentes en una escena 3D. Es un buen método para ahorrarnos el uso de la Radiosidad.

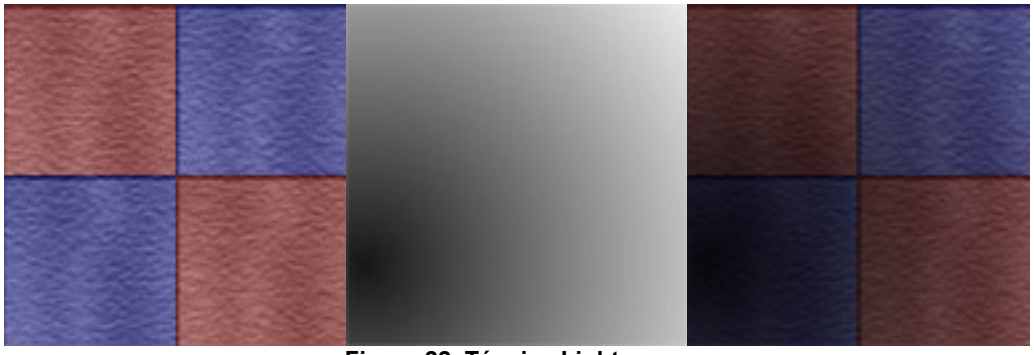

Figura 22. Técnica Lightmaps.

4.2 LIBRERÍAS PARA EL DESARROLLO DE VIDEOJUEGOS

La principal función de las librerías para el desarrollo de videojuegos, que utilizan lenguaje de programación de nivel medio, es la de facilitar la integración en un entorno de modelos complejos, abstrayendo al programador de las definiciones basadas en geometrías más básicas, típicas de lenguajes de bajo nivel como DirectX u OpenGL.

En este apartado se hace una lista con algunas de las librerías de nivel medio, que se pueden utilizar en la actualidad para la creación de videojuegos, clasificadas según el lenguaje de programación que utilizan como base:

- **C/C++/Java**:

  - Allegro: es portable a muchas plataformas. Incluye rutinas básicas para videojuegos, gráficos 2D y 3D, sonido y controles.
  - SDL (Simple Direct Media Layer): es algo más antigua, aunque sigue siendo muy utilizada. Ofrece gráficos en 2D, sonido y controles.
  - OGRE (Object-Oriented Graphics Rendering): es open source[6]. Solo realiza gráficos en 3D, no incluye sonido, ni otras funcionalidades adicionales.
  - OpenSceneGraph: de código abierto usado para el desarrollo de aplicaciones como simulación visual, videojuegos, realidad virtual, visualización científica y modelado 3D. La plataforma está escrita en C++ usando OpenGL.
  - Virtools: empresa que ofrece un entorno para desarrollo de aplicaciones 3D en tiempo real y servicios relacionados, dirigidos a estudios de creación de videojuegos e integradores de sistemas.
  - CrystalSpace: realiza gráficos 3D en tiempo real, además de ser open source.
  - ClanLib: para la creación de juegos de video, es de código abierto y gratuita, para uso comercial está bajo licencia ClanLib.
  - Java3D: proporciona una colección de constructores de alto-nivel para crear y manipular geometrías 3D y estructuras para dibujar esta geometría.

[6] Código fuente de un programa que está disponible bajo licencias que permiten su estudio, modificación y mejora, además de su posterior distribución.

Se utiliza para creación de imágenes, visualizaciones, animaciones y aplicaciones gráficas 3D interactivas.

- **Python**[7]:

  - PyGame: utiliza por debajo la librería SDL.
  - Panda3D: es open source, está implementada en Python y C++ por el EntertainmentTechnology Center, en la universidad Carnegie-Mellon.
  - WorldViz: es un kit completo de herramientas para el desarrollo de contenido interactivo y aplicaciones 3D.

- **Con lenguaje propio**:

  - Multimedia Fusion: tiene un software que permite conseguir resultados de manera sencilla sin conocimientos de programación.
  - Torque GameEngine: desarrollado originalmente por Dynamix, su código puede ser compilado en múltiples plataformas.
  - Unity: es para hacer juegos en el navegador y el resultado es compatible con cualquier plataforma, bajando un plugin[8].
  - GameMaker: forma una plataforma completa para la creación de videojuegos.
  - UDK: consta de varias herramientas para crear juegos, visualizaciones avanzadas y simulaciones 3D. Puede ser descargado gratuitamente siempre que no sea para fines comerciales.
  - DarkBasic: basado en DirectX, es rápido y fácil de usar, aunque es menos potente que los lenguajes similares.

Además de las librerías que se describieron brevemente en los párrafos anteriores, existen lenguajes pareados como VRML y X3D, que permiten la creación y animación de un entono virtual. Su visualización se puede realizar mediante la incorporación de un plugin en un navegador web. El principal inconveniente de estos lenguajes frente al uso de librerías específicas, se reduce a la limitación en el uso de dispositivos de interacción en el entorno virtual.

[7] Lenguaje de programación de alto nivel, que intenta que la sintaxis sea sencilla para que el código sea fácilmente legible. Posee licencia de código abierto.

[8] Aplicación que se relaciona con otra para proporcionar una nueva funcionalidad.

4.3 MODELADORES GRÁFICOS 3D

Las primeras versiones de entornos 3D se desarrollaron a partir de funciones gráficas que permitían construirlos a partir de geometrías básicas como son triángulos y circunferencias. Pero la demanda cada vez mayor de entornos más realistas ha hecho que la complejidad de crearlos con estos métodos no sea factible provocando la aparición de modeladores gráficos basados en el uso de GUIs (Interfaz gráfica de usuario). Este software permite la implementación de entornos virtuales muchos más grandes y complejos pero de una manera mucho más fácil.

Los gráficos 3D se originan mediante cálculos matemáticos sobre entidades geométricas tridimensionales producidas en un ordenador. La creación de los mismos puede comprender varias fases:

- **Modelado:** consiste en ir dando forma a objetos individuales que luego formarán la escena. Existen diversos tipos de geometría para modelar con polígonos, subdividiendo superficies o modelado basado en imágenes.
- **Iluminación**: se crean puntos de luz con dirección, color, etc.
- **Animación:** esta fase no se produce siempre, solo cuando se quiere que el modelo tenga movimiento. Un objeto se puede animar en cuanto a:
    - Transformaciones en los ejes: rotación, escala o traslación.
    - Forma:
        - Mediante esqueletos: se asigna un esqueleto al objeto con la capacidad de afectar a la forma y movimiento del mismo.
        - Mediante deformaciones.
        - Dinámicas: para simular movimiento de ropa, pelo, etc.
- **Renderizado:** es el proceso final, en el cual se genera el resultado, imagen o animación, para ello existen muchas técnicas como el rénder de alambre o el rénder basado en polígonos. Los típicos motores de renderizado pueden simular efectos cinematográficos o imperfecciones mecánicas de la fotografía física, pero a las que el ojo humano está acostumbrado, intentando aportar realismo a la escena, como por ejemplo sistemas de partículas que simulan lluvia o fuego. Necesita gran capacidad de cálculo, pues requiere simular procesos físicos complejos.

Las aplicaciones informáticas para la creación de gráficos 3D más conocidas son:

- **Maya** (Autodesk): es el software de modelado más popular en la industria. Tiene un completo conjunto de herramientas para animación, modelado, simulación, efectos especiales y rendering. Se utiliza en cine, televisión, publicidad o diseño gráfico.
- **SOFTIMAGE|XSI** (Autodesk): es un gran intérprete de animaciones 3D y aplicaciones de efectos especiales. Incluye un entorno creativo interactivo y las herramientas de animación Autodesk Face Robot, que complementan a Maya y 3ds Max.
- **3DStudio MAX** (Autodesk): su arquitectura está basada en *plugins*. Es uno de los software de animación 3D más utilizado, sobre todo para videojuegos, televisión y cine. Permite, gracias a su potencia, la creación de diseños de forma rápida y sencilla.
- **LightWave3D** (Newtek): su motor de renderizado soporta características avanzadas. Está formado por dos subprogramas, Modeler, que es donde se realiza el modelado del objeto con filosofía orientada a capas, y el Layout, donde se realiza el renderizado y la configuración de luces y cámaras.
- **Blender:** es un software libre con un interfaz gráfico de usuario poco intuitivo, ya que no se basa en ventanas, aunque si permite la personalización de los menús y vistas de cámara.
- **Cinema 4D:** se caracteriza por su facilidad de uso, velocidad y resultados profesionales. Incluye tutoriales y puede transformar imágenes 2D en 3D con sus herramientas de importación.
- **Houdini:** proporciona facilidad de uso, con una amplia gama de funciones que permiten realizar producciones rápidamente y de forma flexible.
- **Rhinoceros:** está basado en NURBS[9]. Se ha popularizado por su diversidad al adaptarse a las necesidades de diferentes industrias y su relativo bajo coste en comparación con otras aplicaciones similares. Permite operar con mucho formatos, mediante sus herramientas de conversión, evitando así problemas de compatibilidad.

[9] Algoritmo para síntesis de imágenes tridimensionales, basado en el Ray Casting. Traza rayos desde el observador para determinar que objetos de la escena serán visibles.

- **Pov-ray:** basado en raytracing[10], es gratuito pero no libre, su código está disponible bajo las condiciones de la licencia POV-Ray. No dispone de interfaz gráfica para modelado, pero puede interpretar ficheros ASCII, que contengan información sobre la escena.
- **Cheetah 3D:** proporciona una gran cantidad de herramientas organizadas en un interfaz elegante y poderoso, de manejo intuitivo y sencillo.
- **SketchUp (**Google): muy utilizado en entornos arquitectónicos, ingeniería civil, diseño industrial, videojuegos y películas. Se caracteriza por ser intuitivo y flexible, incluyendo un tutorial paso a paso en video.
- **Zbrush:** ofrece una de las mayores cantidades de herramientas para artistas gráficos del mercado. Los menús están diseñados con el principio de circularidad. Dispone de un potente exportador que permite usar los modelos sin necesidad de tener una aplicación digital.

4.4 LIBRERÍAS FÍSICAS

Las librerías físicas complementan al motor gráfico de diseño 3D o a las librerías de programación de videojuegos de alto nivel, añadiendo las características propias para que se puedan simular fenómenos físicos y de esta manera conseguir que el entorno creado sea más real al estar sujeto a las leyes físicas. Utilizan variables como la velocidad, la masa, la gravedad, la fuerza, etc., proporcionando un mayor impacto visual de la escena.

Entre los más populares se encuentran:

- **ODE** (Open Dynamics Engine): es una librería de alto rendimiento para la simulación de la dinámica de los cuerpos rígidos, multiplataforma y de código abierto. Se utiliza en juegos y simulación.
- **Newton Game Dynamics:** es de código abierto con licencia zlib, para la simulación en tiempo real de la física del entorno. Ocupa poco tamaño, es rápido, estable y fácil de usar.

[10] Acrónimo inglés de la expresión Non UniformRational B-splines. Se trata de un modelo matemático para generar y representar curvas y superficies.

- **PhysX**: es un kit de desarrollo diseñado para llevar a cabo cálculos físicos complejos.
- **Havok**: se utiliza en juegos y recrea las interacciones entre objetos y personajes del mismo, por lo que detecta colisiones, gravedad, masa y velocidad en tiempo real llegando a conseguir ambientes muy realistas.
- **Bullet:** es un motor de física de código abierto, que incluye detección de colisiones 3D, dinámica de cuerpo blando y dinámica de cuerpo rígido. Se utiliza en juegos y efectos visuales para cine. Se encuentra bajo licencia zlib.

4.5 LIBRERÍAS DE VIDEO

OpenCV es la más conocida, destaca por su uso en aplicaciones de realidad aumentada, y consiste en una librería de código abierto multiplataforma (Linux, Windows y Mac OS X) escrita en los lenguajes de programación C/C++ y distribuida bajo licencia BSD[11]. Sus orígenes parten de una iniciativa de Intel Reseach con carácter libre, para ser empleada con fines comerciales, educativos y de investigación.

Fue diseñada para ser eficiente en aplicaciones de tiempo real y proveer de un framework de Visión por Computador que fuera sencillo de usar, permitiendo así la construcción rápida de aplicaciones de Visión por Computador que fueran potentes y robustas. Cuenta además con interfaces para otros lenguajes de programación como Python, Ruby, Java, Matlab, entre otros.

Las librerías OpenCV están compuestas por cinco módulos:

- **CV:** Contiene las funciones principales de OpenCV, tales como procesamiento de imágenes, análisis de la estructura de la imagen, detección del movimiento y rastreo de objetos, reconocimiento de patrones, etc.
- **Machine Learning**: Implementa funciones para agrupación (clustering), clasificación y análisis de datos.

[11] Licencia de software otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution*)

- **CXCORE**: Define las estructuras de datos y funciones de soporte para algebra lineal, persistencia de objetos, transformación de datos, manejo de errores, etc.
- **HighGUI**: Empleada para construcción de interfaces de usuario sencillas y muy ligeras.
- **CVAUX:** Formada por un conjunto de funciones auxiliares/experimentales de OpenCV

4.6 COMUNICACIONES EN RED

La necesidad de comunicar programas de manera sencilla y eficaz dio origen a la aparición de los sockets[12], que permitían la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados.

Utilizan una serie de primitivas para establecer el punto de comunicación, para conectarse a una máquina remota en un determinado puerto que esté disponible, para escuchar en él, para leer o escribir y publicar información, y finalmente para desconectarse.

Para que los programas puedan comunicarse entre sí deben de cumplir:

- Que ambos programas sean capaces de localizarse.
- Que puedan intercambiarse datos.

Para definir un socket es necesario, como mínimo, tres recursos:

- Un protocolo de comunicación, para el intercambio de los datos.
- Un par de direcciones de red que identifiquen el origen y el destino de la comunicación.
- Un par de números de puertos que identifican los procesos los procesos que se ejecutan a nivel de aplicación.

De esta manera, permiten implementar una arquitectura cliente/servidor.

4.6.1 Funcionamiento

[12]Un socket es una abstracción software a través de la cual una aplicación puede enviar y recibir información. Un socket queda definido por las direcciones IP local y remota, un protocolo de transporte y un puerto local y remoto asociado.

Normalmente, un servidor se ejecuta sobre una máquina y tiene un *socket* que responde en un puerto específico. El servidor únicamente espera, escuchando a través del *socket* a que un cliente haga una petición.

En el lado del cliente, este conoce el nombre del host en el cual el servidor se encuentra ejecutado y el número de puerto en el cual está conectado. Para realizar una petición de conexión, el cliente intenta conectar con el servidor, que se encuentra en una máquina de dirección conocida a través del puerto especificado.

Si todo va bien, el servidor acepta la conexión. Además, el servidor crea un nuevo *socket* sobre un puerto diferente que asigna a esta conexión, ya que, en el que se hizo el establecimiento debe permanecer a la escucha de otras posibles peticiones por parte de los clientes.

Por parte del cliente, si la conexión es aceptada, un *socket* se crea satisfactoriamente y puede ser usado para comunicarse con el servidor. Es importante darse cuenta que el *socket* del cliente no se está utilizando el número de puerto por el cual se realizó la petición al servidor, en su lugar se asigna un número de puerto local a la máquina en la cual está siendo ejecutado.

Ahora el cliente y el servidor pueden comunicarse escribiendo o leyendo en o desde sus respectivos *sockets*.

Algunas de las librerías de sockets más utilizadas son las siguientes:

- **Winsock** (WINdowsSOCKet): es una biblioteca dinámica de funciones DLL[13] para Windows que se hizo con la finalidad de implementar aplicaciones basadas en el protocolo TCP/IP. Incluye soporte para envío y recepción de paquetes de datos a través de sockets.
- **Java Sockets**: es una clase *Socket* proporcionada por el paquete java.net de la plataforma de desarrollo Java, la cual implementa una de las partes de la comunicación bidireccional entre un programa y otro, en la red. Utilizando esta clase en lugar del código nativo de la plataforma donde se está

[13]Biblioteca de enlace dinámico, se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo.

ejecutando, los programas Java pueden comunicarse a través de la red de una forma totalmente independiente a la plataforma que usen.

- **VRPN** (Virtual-RealityPeripheral Network): es un sistema de servicio independiente y de red transparente para acceso de periféricos a aplicaciones de realidad virtual. Fue originalmente implementado por Russell M. Taylor II del departamento de ciencia de la computación de la universidad de Carolina del Norte. Este sistema programa interfaces entre la aplicación cliente y el servidor encargado de comunicarse con el hardware.

4.7 CASOS DE ESTUDIO

Para mencionar algunos casos de estudio que actualmente han sido pioneros en el desarrollo de las comunicaciones, en los gráficos y en la implementación de nuevos dispositivos en los videojuegos que existen actualmente, es necesario conocer y diferenciar bien los conceptos de realidad virtual y realidad aumentada.

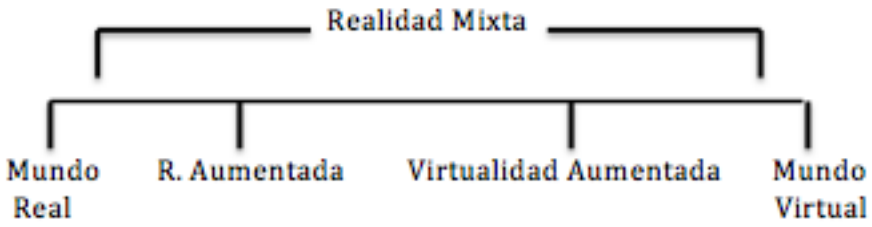

Figura 23. Esquema tipos de realidades.

Cuando se habla de realidad aumentada hablamos de incluir y perfeccionar la realidad, agregándole y superponiendo objetos virtuales a la misma. Mientras que por otra parte con la realidad virtual lo que se intenta es aplicar la sustitución de la realidad a través de un dispositivo, sumergiéndonos así en una realidad construida que no existe. Ambas tecnologías comparten muchos puntos en común, pero la finalidad de cada una es diferente.

Por otro lado existe la realidad mixta, que combina mundos virtuales con el mundo real (físico). Permitiendo interactuar con personas u objetos tanto reales como virtuales.

En medio también encontramos la virtualidad aumentada que a diferencia de la realidad aumentada, esta está más cerca de un entorno virtual que de uno real. Algunos de los juegos que han empleado éste tipo de realidad han utilizado la captación de movimiento por medio de una cámara. Por ejemplo, con el accesorio popular y propulsor en este campo, el EyeToy de la consola PlayStation se conseguía insertar la silueta del jugador en la escena del juego y permitía interactuar con elementos virtuales.

4.7.1 Dispositivo Kinect

Siguiendo en la línea de dispositivos que permiten al usuario realizar una interacción en un entorno virtual empleando el concepto de virtualidad aumentada, se encuentra el dispositivo Kinect. Este dispositivo fue lanzado en 2009, desarrollado por Microsoft para su videoconsola Xbox 360.

Figura 24. Imagen del dispositivo Kinect.

Microsoft buscaba una tecnología que empleara cámaras webcam “con profundidad”. El sensor de profundidad consiste en una combinación de un laser infrarrojo y un sensor monocromo CMOS de infrarrojos. La información de profundidad se captura emitiendo pulsos de luz infrarroja a todos los objetos de la escena y detectando la luz reflejada en la superficie de todos los objetos. Posteriormente todos los objetos de la

escena son organizados en capas. La información de color se extrae por un sensor de imagen en color convencional.[8]

Como puede verse en la figura 25, Kinect tiene una composición horizontal en la que se encuentran:

- Una cámara RGB que proporciona una resolución VGA de 8 bits (640x480).
- Un sensor de profundidad monocromo de resolución VGA de 11 bits que proporciona 2^{11} = 2048 niveles de sensibilidad.
- Un array de cuatro micrófonos.

De lo juegos más destacados que usan este dispositivo empleando el concepto de virtualidad aumentada, se encuentra el Kinect Sports, sacado al mercado en 2010.

Figura 25. Juego Kinect Spots.

Otros de los juegos más populares de kinect es el Dance Masters y el Kinect Adventures, ambos lanzados también en 2010.

Figura 26. Juego Kinect Adventures.

5. DESARROLLO JUEGO DE MESA 3D INTERACTIVO

Para el desarrollo de esta sección hay que tener en cuenta que los motores de los juegos se basan en una arquitectura estructurada en capas, y como van estructuradas estas capas. De este modo, las capas de nivel superior dependen de las capas de nivel inferior, pero no de manera inversa. Este planteamiento permite ir añadiendo capas de manera progresiva y, lo que es más importante, permite modificar determinados aspectos de una capa en concreto sin que el resto de capas inferiores se vean afectadas por dicho cambio.

A continuación, se describen los principales módulos que forman parte de la arquitectura.

1. Hardware: La capa relativa al hardware está vincula a la plataforma en la que se ejecutará el motor del juego, ya sea por ejemplo una videoconsola.
2. Drivers: Soporta aquellos componentes del software de bajo nivel que permiten la correcta gestión de determinados dispositivos.
3. Sistema Operativo: Representa la capa de comunicación entre los procesos que se ejecutan en el mismo y los recursos hardware asociados a la plataforma en cuestión.
4. SDKs y middlewares: El desarrollo de un motor de un juego se suela apoyar en bibliotecas existentes y Software Development Kits (SDKs) para proporcionar una determinada funcionalidad. Aunque este software está bastante optimizado, algunos desarrolladores prefieren personalizarlo para sus necesidades particulares. Otro ejemplo representativo de SDKs vinculados al desarrollo de videojuegos son aquellos que dan soporte a la detección y tratamiento de colisiones y a la gestión de la física de un juego.
5. Capa Independiente de la plataforma: Gran parte de los juegos se desarrollan para su posterior lanzamiento en diversas plataformas. Por lo que es bastante común encontrar una capa software que aísle al resto de capas superiores de cualquier aspecto que sea dependiente de la plataforma. Dicha capa se suele denominar capa independiente de la plataforma.
6. Subsistemas principales: Está capa está vinculada a todas aquellas utilidades o bibliotecas de utilidades que dan soporte al motor de juegos. Algunas de las bibliotecas más relevantes son:
    - Biblioteca matemática, responsable de proporcionar al desarrollador diversas utilidades que faciliten el tratamiento de operaciones relativas a vectores, matrices u operaciones. Son esenciales en el desarrollo de un juego, ya que éstos tienen una naturaleza inherentemente matemática.
    - Estructura de datos y algoritmos, responsable de proporcionar una implementación más personalizada y optimizada de diversas estructuras de datos, como listas enlazadas o árboles binarios. Esta resulta crucial cuando la memoria está limitada.
    - Gestión de memoria, responsable de garantizar la asignación y liberación de memoria.
    - Depuración y logging, responsable de proporcionar herramientas para facilitar la depuración y el volcado de logs para su posterior análisis.

7. Gestor de recursos: Proporciona una interfaz unificada para acceder a las distintas entidades software que conforman el motor de juegos, como por ejemplo la escena o los propios objetos 3D.
8. Subsistemas específicos de juego: Por encima de la capa de subsistema de juego y otros componentes de más bajo nivel se sitúa la capa de subsistemas específicos de juego, en la que se integran aquellos módulos responsables de ofrecer las características propias del juego.[5]

5.1 COMPORTAMIENTO DEL JUEGO

El juego se basa en el clásico juego de mesa conecta 4, en este trabajo se ilustra en la escena principal un tablero de 7x6, con 42 posibles posiciones con las que interactuarán ambos jugadores simulando al juego clásico del conecta 4, en el que van cayendo las fichas hasta hacer 4 en raya.

El primero en empezar la partida es el usuario conectado con la parte de la aplicación que gestiona al servidor de comunicaciones, tendrá que empezar a colocar las fichas por la parte inferior del tablero y así sucesivamente de forma escalonada como si se tratará del juego clásico.

Un jugador será representado por un avatar en la pantalla de su contrincante, en este caso el cliente, y el otro jugador será representado por las imágenes recogidas por su webcam, las cuales serán enviadas por la red a la aplicación cliente y estas serán mostradas en pantalla. Por lo que se intenta crear un entorno virtual donde cada jugador sea representado de distinta forma.

Finalmente cuando alguno de los jugadores consiga crear la combinación de cuatro en raya en la pantalla aparecerá un panel, el cual mostrará el resultado de cada jugador. El juego también consta con la posibilidad de que si ninguno de los jugadores ha conseguido hacer la combinación ganadora y se han hecho uso ya de todas las posiciones en este caso el ganador es el tablero, produciéndose un empate.

5.2 CREACIÓN DEL ENTORNO VIRTUAL

El juego consta de dos partes iguales en interfaz pero diferentes en comunicación:

- **Aplicación servidor:** comprendida de dos escenas creadas y renderizadas con Ogre, donde en la principal escena se encuentra el tablero del juego compuesto por los elementos en 3D con los que interaccionarán los jugadores, mientras que en la segunda escena se muestra un avatar que representa al jugador de la aplicación cliente.
- **Aplicación cliente:** comprendida también por dos escenas creadas con Ogre, la primera escena es igual que en el servidor mientras que en la segunda se muestran las imágenes procedentes de la webcam situada en el equipo del servidor utilizando la librería OpenCV.

En ambas partes tanto para la transmisión como recepción de información se usa la librería Winsock .

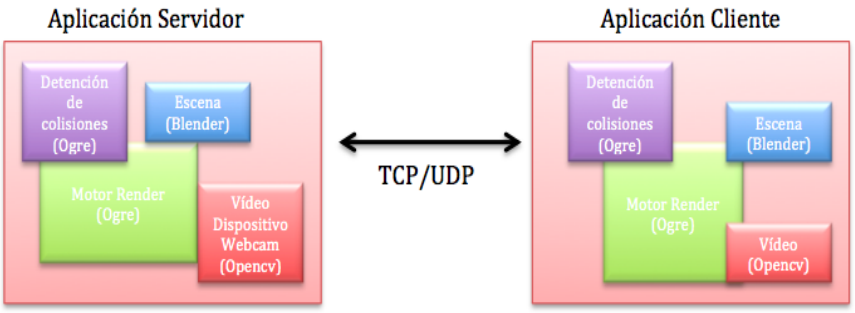

Figura 27. Esquema aplicación Servidor/Cliente.

5.2.1 OGRE

Para el desarrollo de esta parte del trabajo se ha usado el motor gráfico OGRE desarrollado en C++(el lenguaje estándar en el desarrollo de videojuegos). OGRE se centra en este lenguaje sobre los sistemas operativos CNU/LINUX, Mac OS X y Microsoft Windows. No obstante, existen *wrappers*[14] de la API a otros lenguajes (como Java,

[14]Capas de software adicionales

Python o C#) mantenidos por la comunidad de usuarios. Así mismo en su página web[15] se pueden encontrar tutoriales y el intercambio de ideas entre distintos usuarios.

Además como ya se ha mencionado en otra sección anterior la gran ventaja de OGRE es que es un proyecto open source bajo licencia LPGL[16] (o GPL Reducida).

OGRE también hace uso de varios patrones de diseño, los cuales hacen referencia a una solución común que mejora la flexibilidad y el uso de la biblioteca. Por ejemplo, el patrón *Singleton* se emplea en numerosas métodos para forzar que solo exista una instancia de una clase. El patrón *Observador* se utiliza para informar a la aplicación sobre eventos y cambios de estado. El patrón *Visitor* se emplea para permitir operaciones sobre un objeto sin necesidad de modificarlo. Y el encargado de recorrer el contenido de una estructura de datos es el patrón *Iterador*.

Aunque el principal uso de OGRE está destinado al desarrollo de videojuegos, no está únicamente diseñado para ello, y por lo tanto no cuenta con un soporte nativo para otras características tales como sonido, física, red, etc. Ya que OGRE sólo se dedica a representar escenas en 3D, no maneja las entradas del usuario, no controla el estado del juego, ni aporta audio, entre otras cosas. En el caso de querer añadir estas funcionalidades, se deben integrar con el uso de librerías externas.

El kit de desarrollo de OGRE (SDK) proporciona un interfaz de programación de aplicación sencillo con los servicios del sistema, sin tener que recurrir a programación de bajo nivel, soportando la utilización de las librerías de bajo nivel (Direct3D y OpenGL). Este SDK provee al programador de los archivos de cabecera y librerías de importación necesarias para enlazar un programa con las librerías del sistema necesarias.

Se ha convertido en un estándar como motor de interpretación de gráficos en aplicaciones orientadas a objetos[17], permitiendo que el desarrollador se centre en los detalles de la aplicación y no tanto en la forma en la que se interpretará la escena 3D, que es la única función de OGRE dentro de un programa. Con relativamente pocas entradas, es capaz de invocar una asombrosa cantidad de procesos en nombre de la

[15] http://www.ogre3d.org

[16] Siglas: Lesser General PublicLicense – Licencia Pública General Menor

[17] La programación orientada a objetos está basada en una colección de unidades individuales, u objetos, que interactúan entre ellos y con otros

escena 3D creada, para que esta aparezca en la pantalla del ordenador o en otros lugares, en comparación con las que serían necesarias si se usara de forma directa Direct3D u OpenGL.

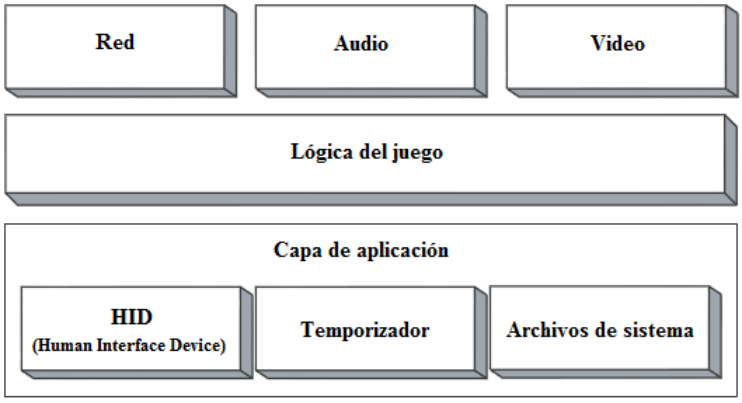

Figura 28. Arquitectura de un juego en tiempo real.

En la figura 30 se puede ver una posible arquitectura, a grandes rasgos, de un juego en tiempo real. El sistema actual de computación (hardware) es transparente al usuario gracias a la capa de aplicación, y la lógica del juego es la encargada de negociar con otros sistemas como los temporizadores, HID (Interfaz de dispositivo humano) de entrada y los archivos de sistema. La capa lógica contiene lo necesario para interpretar las entradas obtenidas de la capa de aplicación, así como los eventos transmitidos sobre la red. Estos estados son periódicamente presentados al usuario en forma de audio y video.

Resumiendo las capacidades más destacadas que ofrece con respecto a otros sistemas semejantes son:

- Soporte completo tanto para OpenGL como para Direct3D.
- Es multiplataforma, válido para Windows, Linux y Mac OS.
- Estructura simple y extensible, fácilmente integrable dentro de una aplicación ya existente.
- Poderoso y sofisticado gestor de materiales y texturas, con soporte para formatos como JPG, GIF o PNG, además de variadas técnicas de sombras.
- Comunidad oficial para soporte y desarrollo, con exportadores de modelos 3D desde las aplicaciones de modelado comerciales.
- Soporte completo para animaciones mediante esqueletos y poses.
- Permite la carga en el sistema y la gestión de archivos, de tipo ZIP y PK3.

- Incluye un *plugin* de efectos basado en partículas llamado ParticleFX.
- Cola única para la gestión de la representación permitiendo el control en el orden del proceso.
- Soporta geometría estática.
- Es un software de código abierto, bajo licencia GPL.

La filosofía de diseño de OGRE es la de proporcionar métodos orientados a objetos para el acceso a las herramientas de procesado, de esta manera las complicaciones inherentes del trabajo con la geometría básica son eliminadas, centrándose únicamente el programador en la creación de su escena, objetos estáticos y dinámicos, luces, cámaras, etc.

Para manipular los objetos de la escena, rotarlos y trasladarlos, no es necesario trabajar con transformaciones de matrices, ya que OGRE proporciona instrucciones para ello que permiten hacerlo únicamente indicando la cantidad de movimiento en unidades, como los grados o los radianes.

Una de las características más potentes de OGRE es el desacople, la separación del escenario gráfico con los contenidos de la escena, definiéndose así una arquitectura de plugins.

A diferencia de otros motores gráficos que se basan en la Herencia a la hora del diseño del escenario gráfico, OGRE se basa en la Composición. De esta manera el escenario gráfico funciona a nivel de interfaz con sus métodos propios sin tener en cuenta el algoritmo gráfico que se utiliza. La escena se compone de nodos de escena, adjuntándolos a los objetos móviles. Los nodos a su vez no pueden acceder a los contenidos y funciones del escenario. Las características geométricas y propiedades de los nodos son accesibles para la escena a través de los objetos movibles. El nodo y sus características dan lugar a una entidad, que a su vez posee unas propiedades para su renderizado y puede estar formada por una o más subentidades, que son las que realmente se representan.

Por lo tanto OGRE permite realizar cambios significativos en el escenario sin influir a otras entidades. La descripción anterior se puede apreciar de manera visual mediante la figura adjunta.

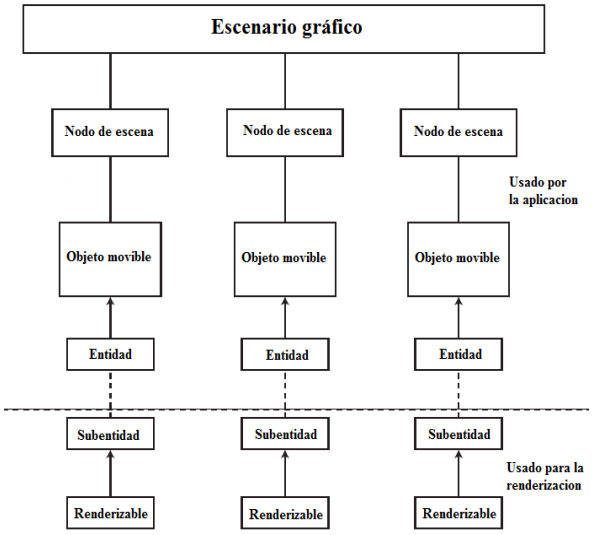

Figura 29. Esquema general de la gestión del escenario gráfico.

Estos conceptos forman parte de la clase *SceneManager*, que es la que da origen a los nodos de la escena (SceneNode), que son los que se manipulan, siguiendo una estructura jerárquica.

Todas las opciones de diseño son ampliables mediante *plugins*, siempre y cuando no entre en conflicto con la arquitectura de OGRE, como ya se comentó, sin necesidad de reconstruir toda la librería para incorporarlas.

OGRE requiere de un coprocesador de gráficos, como los fabricados por NVIDIA y ATI, para optimizar el sistema.

Utiliza un sistema de colas de renderizado flexible, el cual consiste en tener varias colas con diferentes prioridades, y dentro de las mismas los objetos también tendrán un orden, así, siguiendo algún criterio se accederá a una cola y a su contenido. La flexibilidad la da el hecho de que las prioridades se pueden cambiar.

Por motivos de eficiencia y rapidez los objetos gráficos deben tener un formato determinado, para conseguirlo la comunidad ha creado convertidores, que utilizan como formato intermedio XML, que posibilita la revisión y modificación de los datos.

Dentro de las animaciones existen de tres tipos, esqueleto, y basadas en vértices, transformación y pose. En las animaciones se utiliza el tiempo como valor para modificar otras propiedades, como la posición. La animación y la geometría de las animaciones se

guardan en formato binario optimizado. El proceso empleado se basa en la exportación desde la aplicación de modelado y animación 3D a un formato XML, donde posteriormente mediante con la herramienta OgreXMLConverter se convertirá al formato binario optimizado.

Los recursos, normalmente modelos gráficos, son gestionados para controlar la cantidad de memoria que ocuparán. Estos elementos son los materiales, texturas, fuentes, etc. Pueden estar en alguno de los siguientes estados: indefinido, no se hace referencia a él en el programa; declarado, si ha sido definido; descargado, cuando el recurso ha sido inicializado y creado; y cargado, cuando ya ocupa espacio en la memoria.

5.2.1.1 Componentes de OGRE

Los sistemas más básicos y comunes con los que se interactúan en una aplicación típica de OGRE son:

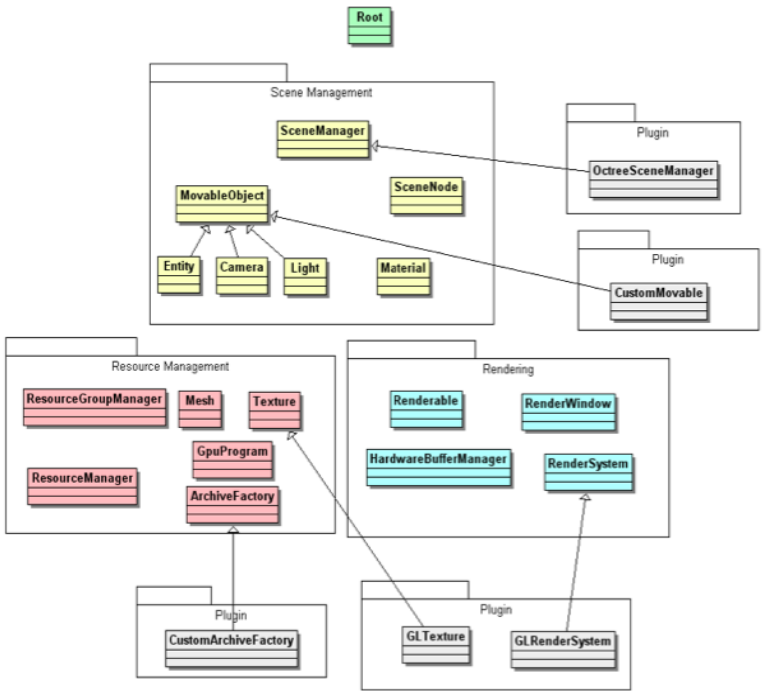

Figura 30. Esquema de los componentes de OGRE [2].

- **Objeto raíz**: Es el principal punto de acceso a los subsistemas de OGRE en una aplicación. Este objeto debe ser el primero en crearse, y el último en

destruirse. Proporciona un mecanismo para la creación de los objetos de alto nivel que nos permitirán gestionar las escenas, ventanas, carga de plugins, etc.

- **Gestor de recursos** (*ResourceGroupManager*): Se encarga de gestionar todo lo necesario para la representación de la escena o cualquier otro recurso, los localiza y los inicializa. Todos los recursos son gestionados por un único objeto llamado ResourceGroupManager. Los tipos de recursos que conoce OGRE son:

  - Mallas (*mesh*). Es un formato de malla binario (.mesh) que está optimizado para una carga rápida y se genera normalmente con el OgreXMLConverter.
  - Esqueleto (*skeleton*). Por lo general hace referencia a un archivo de malla, define la jerarquía de huesos y los fotogramas utilizados con la animación, también se crean con el OgreXMLConverter.
  - Material (*material*).Se especifica mediante scripts (ficheros de extensión .material) con los datos de los archivos para su correcta representación. Pueden estar referenciados en el archivo *.mesh* o ser enlazados a la malla por código.
  - Programas de alto nivel GPU[18] y de bajo nivel ASM[19]: OGRE los analizará pero no los compilará.
  - Texturas.OGRE soporta todos los formatos en 2D admitidos por la biblioteca OpenIL.
  - Compositor. Su funcionalidad es muy similar a la de los materiales, con la diferencia de cargar sus scripts en la con extensión *.compositor*.
  - Fuentes (*fontdef*). Define las fuentes que se utilizan en los recubrimientos en un archivo *.fontdef*.[2]

[18]Es un lenguaje de programación de alto nivel, desarrollado por NVIDIA, basado en C, que fue diseñado para las tarjetas y APIs gráficas.

[19] Es un lenguaje de programación de bajo nivel para computadores, microprocesadores y cualquier circuito programable, mediante símbolos que representan los códigos binarios de las máquinas.

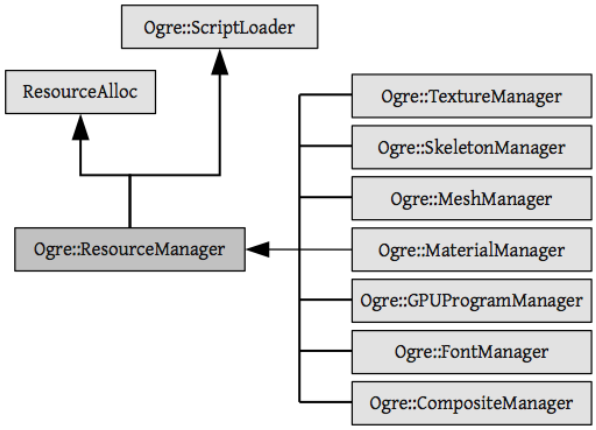

Figura 31. Diagrama de clases asociado al gestor de recursos de Ogre 3D.

- **Gestor de escenas** (SceneManager): Es la encargada del contenido de la escena que será renderizada por el motor gráfico. Crea los nodos de escena y los manipula. Es una clase importante para implementar el desarrollo del contenido de la escena. Como ya se mencionó anteriormente, los nodos de la escena SceneNode son elementos relacionados jerárquicamente, los cuales pueden adjuntarse o desligarse a una escena en su momento de ejecución. El contenido de los nodos se acopla en la forma de Entidades (Entity), que son implementaciones de los objetos móviles. El SceneManager es necesario para añadir una cámara en la escena, o también cuando se quiere obtener o eliminar una luz de la escena. Además mantiene un conjunto de nombres para todos los objetos de la escena. Por lo que cuando se quiere obtener información de un objeto de la escena se pueden usar los métodos getCamera, getLight, …
- **Sistema de renderizado**: Para no tener que interactuar directamente con él, se utiliza la clase *RenderSystem*, que es una interfaz entre OGRE y el hardware adyacente (OpenGL o Direct3D). La forma más habitual de crear la ventana de renderizado (*RenderWindow*)es a través del objeto *Root* o mediante *RenderSystem*. Una creación manual permite el uso de una mayor cantidad de parámetros. En una aplicación estándar no se necesita manipular esta clase directamente, ya que desde las clases SceneManager y Material tenemos todo lo necesario para renderizar objetos y para configurar las propiedades.
- **Clase Entity**: Pertenece a la clase MovableObject. Esta entidad puede representar cualquier objeto que se ponga en la escena. De esta forma para crear una entidad se llama al SceneManager, y proporcionándole el nombre

del objeto Mesh el SceneManager será el encargado de comprobar que la malla ha sido cargada y para eso llama al MeshManager.
Los objetos Mesh pueden estar compuestos por objetos SubMesh. Si una malla utiliza sólo un Material, utilizará únicamente una SubMesh.

OGRE utiliza *Gestores*, que son clases sencillas que administran el acceso a las características de los diferentes subsistemas definidos en OGRE. Por ejemplo, existe el *ArchiveManager*, que proporciona funcionalidad sobre los archivos.

5.2.1.2 CEGUI

CEGUI (*CrazyEddie'sGuiSystem*) es una librería libre que proporciona ventanas y widgets para las API gráficas o motores gráficos que no tienen disponible esa funcionalidad de forma nativa. La librería está orientada a objetos y está programada en C++. CEGUI se utiliza como interfaz gráfica en Ogre3D.

5.2.2 Inicialización de la Aplicación

En el fichero BaseApplication se llevan a cabo las primeras operaciones necesarias antes de poder ejecutar otras operaciones con OGRE.

El primer objeto que se debe crear es el objeto Root, inicializando así la aplicación, mostrando su configuración y cargando la configuración en el archivo 'ogre.cfg', que contiene la información de configuración que se iniciará cada vez que iniciamos la aplicación en OGRE.

Ya una vez creado el objeto Root y ya finalizada su inicialización, pasamos a crear la ventana, la cual será denominada como server o client, dependiendo de la función que desempeñe la aplicación.

La llamada a este método para crear la ventana de forma automática devuelve un puntero a un objeto RenderWindow, que es guardado en la variable mwindow.

```
mWindow = mRoot->initialise(true, "CLIENT");
```

Esta ventana hay que asociarla con una superficie, algo así como un plano de la imagen sobre la que se dibujarán los objetos en 3D. A esta superficie se le denomina viewport. Para crear el viewport utilizamos la instancia addViewport, método de la RenderWindow.

```
Ogre:: Viewport* vp = mWindow ->addViewport (mCamera);
```

Establecemos también el color de fondo de la ventana gráfica como negro.

```
Vp ->setBackgroundColour (Ogre::ColourValue (0,0,0));
```

Lo último, y más importante es establecer la relación de aspecto de la cámara. Esta relación de aspecto se calcula como el número de píxeles en horizontal con respecto del número de píxeles en vertical.

Se recupera la anchura y la altura de la ventana gráfica para ajustar su relación aspecto, el valor predeterminado dependerá del tipo de resolución. Por ejemplo, resoluciones de 4/3 (como 1024x768) tendrán asociado un valor de ratio de 1.333, mientras que otras resoluciones panorámicas de 16/9 tendrán un valor de 1.77, y así según con el tipo de resolución que se tenga.

```
mCamera->setAspectRatio(Ogre::Real(vp->getActualWidth())/
            Ogre::Real(vp->getActualHeight()));
```

En la inicialización también se deben llevar a cabo la configuración de los recursos, donde dentro de la función SetupResources con la instancia, cf.load("resources.cfg") cargamos el archivo 'resources.cfg', el cual permite a OGRE conocer donde debe buscar los recursos multimedia. La única sección del archivo se denomina General.

```
[General]
FileSystem=C:/OgreSDK/media
```

Figura 34. Contenido del archivo resources.cfg

Dentro de esta función se abre el archivo anterior, y va recorriendo cada directorio donde se encuentran los recursos multimedia. De estos archivos se saca el tipo, el nombre, y son registrados dentro del gestor de recursos Ogre::ResourcerGroupManager. Una vez registrados todos los recursos, se inicializan.

5.2.3 Creación de la Escena Principal con Ogre

El juego está compuesto por dos escenas, que se pueden mostrar de forma individual o simultáneamente.

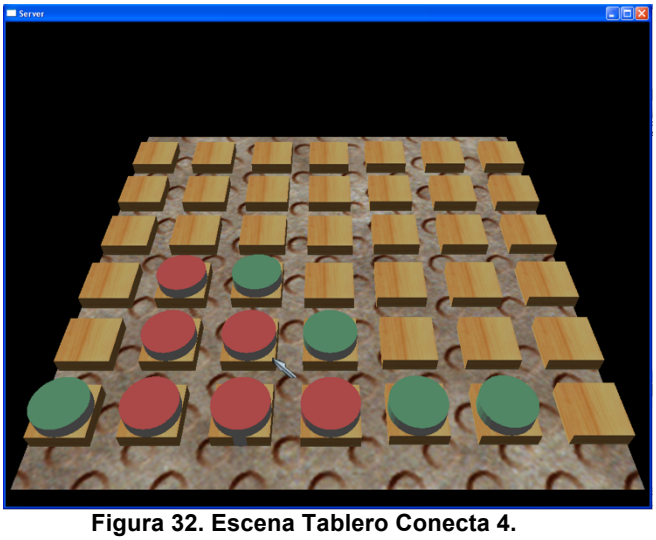

Figura 32. Escena Tablero Conecta 4.

La escena principal es donde se encuentra el tablero de 7x6 y en el que se posicionarán los objetos. Para la creación de esta escena se usa la parte de Ogre de la gestión de escena, en donde encontramos información sobre el contenido de la escena, como está estructurada, como se ve desde las cámaras, etc.

La clase SceneManager, como ya se ha mencionado anteriormente está a cargo de los contenidos de la escena que se renderizan por el motor gráfico. También es responsable de organizar el contenido usando cualquier técnica que considere más

favorable, para crear y gestionar todas las cámaras, los objetos móviles (entidades), las luces y los materiales (propiedades de la superficie de los objetos).

Asimismo, la clase SceneManager se ocupa de la disposición de objetos en el escenario y su secuencia de representación.

Con la clase SceneNode se organizan los objetos dentro de la escena. También con cada nodo se puede almacenar otros objetos como luces, cámaras, etc. La forma de almacenar a las entidades se hace con estructura de árbol, por lo que se podrá tener un padre y los demás hijos podrán heredar propiedades del padre. La animación del SceneNode se crea desde el SceneManager.

En el fichero GraphicsManager en la rutina createScene se irán creando la entidades implicadas en la escena.

- En primer lugar se llama a la instancia de configuración CEGUI.

```
mGuiRenderer = &CEGUI::OgreRenderer::bootstrapSystem();
```

- También se crea el cursor del ratón.

```
CEGUI::SchemeManager::getSingleton().create((CEGUI::utf8*)"Tahar
ezLook.scheme");
CEGUI::MouseCursor::getSingleton().setImage("TaharezLook",
"MouseArrow");
```

- Se crea una entidad por cada objeto de la escena mediante la función miembro createEntity, en la cual se le dará un nombre y se le asociará un archivo .mesh. El SceneManager se asegurará de que la malla se cargue mediante una llamada al administrador de recursos MeshManager.

```
Ogre::MeshManager::getSingleton().createPlane("PlaneMesh",
Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);
```

Las entidades no se consideran parte de la escena hasta que no se adjuntan a un SceneNode. Una vez adjuntadas las entidades a SceneNodes se pueden crear complejas relaciones jerárquicas entre las posiciones y orientaciones de las entidades.

Cuando se carga una malla viene automáticamente con una serie de materiales definidos. Cualquier entidad creada a partir de la malla usará de forma automática los materiales por defecto. No obstante, se pueden cambiar en función de cada entidad, de esta forma se pueden crear una serie de entidades basadas en la misma malla, pero con diferentes texturas. Por ejemplo, utilizando el método setMaterialName se puede cambiar de material.

```
Ogre::Entity*mPlaneEnt=mPrimarySceneMgr-
>createEntity("PlaneEntity" , "PlaneMesh");

   mPlaneEnt->setMaterialName("PlaneMat");
```

- Ya creada la entidad, se crea el SceneNode para poder acoplarlo. La función SceneNode::createChildSceneNode posee tres parámetros, el nombre de la escena, la orientación y la posición.

```
Ogre::SceneNode *mPlaneNode = mPrimarySceneMgr-
>getRootSceneNode()->createChildSceneNode();
```

- Finalmente se adjunta la entidad creada al nodo de la escena para dar la ubicación de renderizado.

```
mPlaneNode->attachObject(mPlaneEnt);
```

OGRE al igual que otros motores gráficos utiliza los ejes X y Z para el plano horizontal y el eje Y para el vertical.

Además Ogre posee la clase vector para poder representar una posición y dirección. Existen vectores para diferentes dimensiones (Vector2, Vector3, Vector4) para dos, tres y cuatro dimensiones respectivamente, siendo el Vector3 el más utilizado.

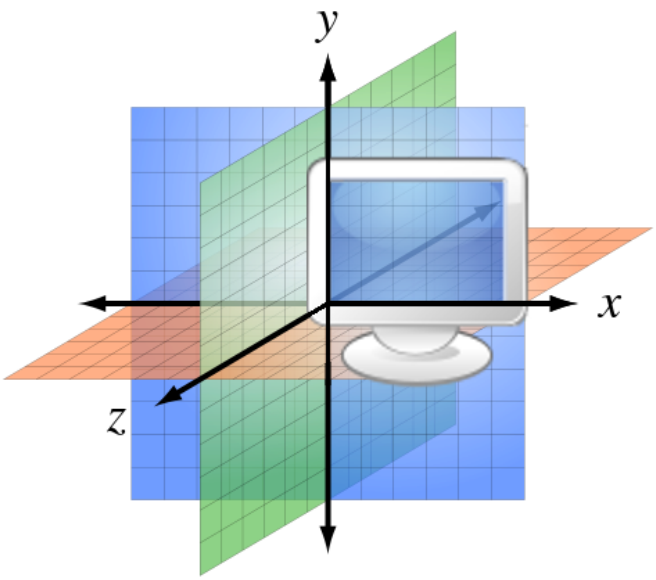

Figura 33. Coordenadas en OGRE.

Además hay otros métodos que son empleados a la hora de crear la escena:

- Para crear una luz ambiental y ver lo que tenemos en la escena, se hace llamando a la función setAmbientLight y especificando el color que deseamos. El constructor del valor del color espera los valores para rojo, verde y azul en el rango de 0 a 1.
- Y con setPosition le indicamos la posición donde colocaremos el objeto con relación a su nodo padre (raíz), situado en (0,0,0).

5.2.4 Creación de una cámara

Una cámara es el objeto que utilizamos para ver nuestra escena, funciona de forma similar a un SceneNode, ya que para darle movimiento y una rotación se puede tratar como tal. Utiliza también el método setPosition para indicarle una posición específica.

Para crear una cámara hay que seguir los siguientes pasos:

A. El primer paso será pedirle al SceneManager que cree una nueva cámara.

```
mCamera = mSceneMgr ->createCamera ("PlayerCam");
```

B. A continuación colocamos la cámara y utilizamos el método lookAt para establecer su posición, centrando su línea de visión en el vector que queramos.

```
mCamera->setPosition(Ogre::Vector3(0,0,80));
mCamera->lookAt(Ogre::Vector3(0,0,-300));
```

Con los valores dados la cámara tiene un alcance de visión a lo largo del eje –z y queda posicionada en dirección del eje z.

C. Por último se va a establecer la distancia de clip, es la distancia a la que la cámara no hará renderizar cualquier malla.

```
mCamera->setNearClipDistance (5);
```

D. Finalmente se crea el SdkCameraMan, que es el controlador de la cámara proporcionada en OgreBites.

```
mCameraMan = new OgreBites::SdkCameraMan(mCamera);
```

Al contar la aplicación con dos escenas se deben crear dos cámaras independientes para cada tipo de escena.

5.2.5 Creación de Luces

Una vez creadas las entidades en el createScene, añadimos luz a nuestra escena, para ello hay que llamar a la función miembro createLight del SceneManager.

```
Ogre::Light* l = mPrimarySceneMgr ->createLight ("MainLight");
```

Existen tres tipos de iluminación:

- POINT: Esta luz se expande por igual en todas las direcciones a partir de un punto.
- SPOTLIGHT: Esta luz funciona como una linterna. Crea un cilindro sólido de luz más brillante en el centro y se va desvaneciendo.

- DIRECTIONAL: Esta luz simula una gran fuente de luz que procede de lejos y golpea a toda la escena por igual en todas partes.

Para definir el tipo de luz, posición y direccionamiento usamos los métodos setType, setPosition y setDirection.

```
l->setPosition(0,10,0);
l->setType(Ogre::Light::LT_DIRECTIONAL);
l->setDirection(0.33, -0.33, 0.33);
```

5.2.6 Gestión del Movimiento

Para realizar la gestión de los eventos del teclado, ratón y actualización de los frames de la escena es necesario utilizar la clase llamada FrameListener, que está formada por tres funciones, frameStarted, frameEnded y frameRenderingQueued, esta última la empleada en este trabajo.

En la función createFrameListener, se obtiene el manejador de la ventana y se convierte a tipo cadena.

```
mWindow->getCustomAttribute("WINDOW", &windowHnd);
windowHndStr<<windowHnd;
pl.insert(std::make_pair(std::string("WINDOW"),
windowHndStr.str()));
```

Ahora con el parámetro obtenido se crea una variable del tipo InputManager, que nos permitirá crear diferentes interfaces para poder emplear el uso del teclado y ratón. Además, también se crean dos variables del tipo OIS, la cual es una biblioteca de código de entrada orientada a objetos compatible con todo tipo de dispositivos; teclados, ratones y joysticks. En este caso se crean del tipo OIS::Keyboard y OIS::Mouse, que se encargarán de la gestión del teclado y el ratón.

Posteriormente dentro de la misma función se registran los eventos del teclado y del ratón con una llamada al método setEventCallBack.

```
mMouse->setEventCallback(this);
```

```
mKeyboard->setEventCallback(this);
```

Donde las variables mMouse y mKeyboard guardan el estado del ratón y del teclado, utilizando la instancia OIS de OGRE.

Finalmente hay que registrar el FrameListener como Ogre::Root, utilizando el método addFrameListener.

```
mRoot->addFrameListener(this);
```

Para gestionar el empleo de los eventos del ratón y teclado se usan otras funciones; como son keyPressed y KeyReleased, cuando se pulsa o se deja de pulsar pasando como parámetros KeyEvent.

De igual forma pasa con el caso del ratón, ya que tiene funciones para ver si fue pulsado o no, mousePressed y mouseReleased, y otra para el movimiento, mouseMoved. También se reciben a un objeto del tipo MouseEvent con el estado actual y así poder realizar el cambio en la escena.

5.2.7 Creación de Paneles

Para mostrar la información del juego es interesante la creación de paneles, para este trabajo se han empleado para mostrar el resultado final de la partida.

En primer lugar es necesario definir una variable en OGRE del tipo ParamsPanel, donde se guardará la información que aparecerá en el panel.

```
OgreBites::ParamsPanel* mResultPanel;
```

Posteriormente hay que definir una cadena de caracteres que será la que contendrá la información del título del panel. Además, con la siguiente instrucción se debe elegir la posición de la pantalla donde aparecerá el panel, siendo en este caso en el centro de la pantalla. Y procediendo finalmente a guardar la información creada en el puntero definido al principio, por lo que cuando se invoque a la función showResult, se mostrará el resultado final.

```
mTrayMgr->moveWidgetToTray(mResultPanel, OgreBites::TL_CENTER,
0);
mResultPanel->setParamValue(0, result);
mResultPanel->show();
```

5.2.8 Selección de Entidades

Para detectar que posición ha sido seleccionada por el ratón en Ogre se emplea el método de Raycasting.

El método Raycasting, consiste en lanzar rayos desde el plano de la imagen, uno por cada pixel, calculando el punto de intersección (si hay alguno) con el objeto más cercano de la escena.

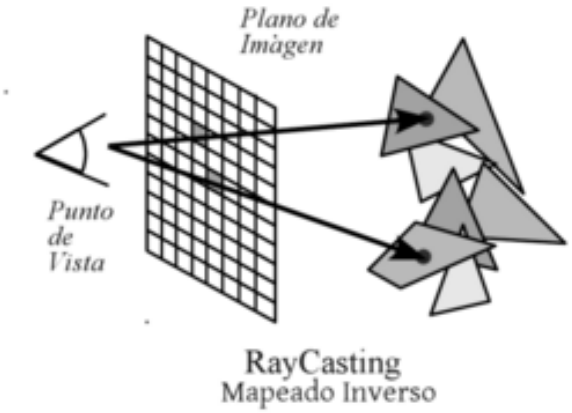

Figura 34. Método Raycasting.

Se utiliza la clase RaySceneQuery, que devuelve todos los objetos que se cruzan con un rayo arbitrario en el espacio. Para ello hay que ajustar el ancho y la altura del estado del ratón. Dependiendo de sí se muestra una escena o ambas a la vez habrá que hacer un ajuste distinto en la altura y el ancho. Ya una vez obtenidos los valores de las coordenadas x e y, se calcula el rayo utilizando las coordenadas.

```
Ogre::RaymouseRay = mCamera->getCameraToViewportRay(x,          y);
mRayScnQuery->setRay(mouseRay);
```

El resultado del rayo se analiza comparándolo con la distancia al objeto más cercana. A partir de ella se ejecuta una respuesta y se ordena por distancia.

```
Ogre::RaySceneQueryResult&result = mRayScnQuery->execute();
mRayScnQuery->setSortByDistance(true);
```

5.2.9 Implementación de un Avatar en la Escena

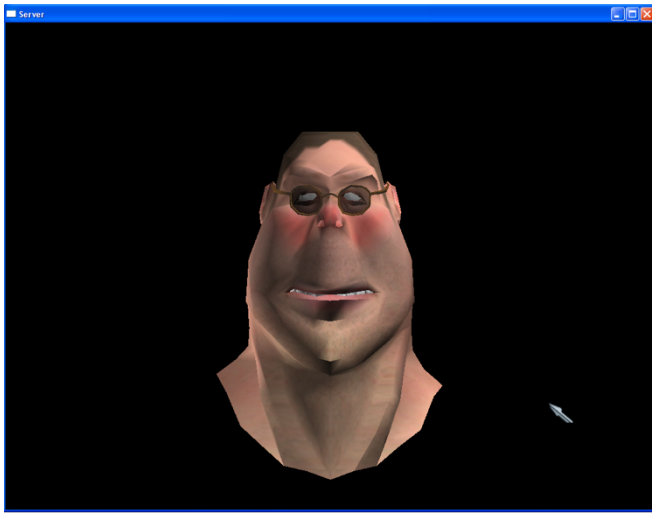

Figura 35. Imagen Avatar.

La cara del avatar nos la proporciona la librería de Ogre, llamada “facial.mesh”. Se carga, y al igual que con la escena anterior se crea una iluminación y una cámara. Para implementar una nueva cámara en esta escena se declara de igual forma que en la primera escena, pero con la diferencia que en este caso se va implementar un estilo de cámara distinto, ORBIT, en este estilo de control la cámara orbta alrededor de un objeto de interés.

Este avatar también irá acompañado con movimientos que se irán actualizando, simulando que habla.

- Para incluir el avatar en la escena en primer lugar se debe crear una entidad, “Head”, y se acopla a un nodo.

```
Ogre::Entity*head=mSecondarySceneMgr->createEntity("Head",
"facial.mesh");
```

- A continuación, se deberá actualizar el estado de la animación basado en cuánto tiempo ha pasado desde el último fotograma.

```
mSpeakAnimState = head->getAnimationState("Speak");
```

- Por último, en el método frameRenderingQueued tenemos que actualizar el estado de la animación basado en el marco de tiempo transcurrido, por lo que justo después de que el teclado y el ratón sean capturados hay que añadir lo siguiente.

```
mSpeakAnimState->addTime(evt.timeSinceLastFrame);
```

5.2.10 Inclusión de la Captura de Vídeo

Figura 36. Segunda escena imagen del vídeo.

En la parte de createscene se va a declarar un material base "vt_material" con una textura por defecto. Este material será actualizado por otro material hijo que sobrescribirá la textura del vídeo en este material.

Es en la función Update del fichero de B3D_VideoTexture en donde se realiza el proceso de representar la imagen recibida. Creando un material hijo con una textura

manual donde se van a cargar sobre ella las imágenes recibidas en 2D, además se ajustan a un ancho y altura. Posteriormente la actualización de los búferes del píxel se llevan a cabo usando el método HardwarePixelBuffer::blitFromMemory. Este método obtiene un objeto PixelBox, el cuadro donde se van a ir representando las imágenes recogidas por la textura y el cual se va a encargar de realizar las conversiones de formato de píxel y las escalas necesarias. Finalmente esta función será invocada al final del método frameRenderingQueued.

5.3 DESARROLLO DE UN MODELO CLIENTE/SERVIDOR

Para la implementación de un modelo cliente/servidor se ha usado las librerías de programación con sockets, los cuales ya mencionados anteriormente, consisten en una abstracción software a través de la cual una aplicación puede enviar y recibir información.

Un socket queda definido por:

- Los puertos, que son los identificadores usados para asociar los datos entrantes a un proceso específico de la máquina. De 1024 a 65535 son de uso libre.
- La dirección IP (32 bits) que consiste en el direccionamiento a nivel de red, la cual se debe conocer del equipo donde se va a ejecutar el proceso.

Existen dos modalidades, según el protocolo de transporte que se vaya a usar, los sockets TCP, orientados a conexión (stream sockets) y los UDP, no orientados a conexión (datagram sockets), ambos sobre IP.

Ambas conexiones tanto TCP como UDP usan una serie de primitivas, las cuales se utilizan para establecer un punto de comunicación, para conectarse a una máquina remota en un determinado puerto disponible, para escuchar en él, leer o escribir información en él, y finalmente para desconectarse. Con todas las primitivas de las que disponen los sockets se puede crear un sistema de diálogo bastante completo.

Para la transmisión y recepción de las funciones del juego se ha usado un tipo de conexión TCP, ya que este tipo de servicio garantiza que no se pierda la información involucrada en el proceso de recibir o emitir información. En este tipo de servicio los datos se transfieren sin encuadrarlos en registros o en bloques, asegurándose de esta manera que los datos lleguen al destino en el orden de transmisión.

Si se rompe la conexión entre los procesos, estos serán informados del suceso y de esta forma se tomarán las medidas oportunas, de ahí que sean denominados como sockets libres de errores.

Para establecer una conexión TCP en primer lugar hay que establecer una conexión entre un par de sockets. Uno de los sockets atenderá peticiones de conexión (servidor), y el otro solicitará una conexión (cliente). Ya una vez realizada la conexión se puede empezar a transmitir los datos en ambas direcciones.

Las primitivas empleadas para las creación, vinculación y destrucción de sockets en ambos tipos de servicios son:

- Socket: Crea el descriptor de socket. Toma como parámetro la familia de protocolos, y el tipo de servicio (stream o datagram).
- Close: Cierra la conexión y libera el socket .
- Bind: Especifica dirección (IP + puerto) y asocia esta dirección local a un socket.

Para realizar la conexión en un servicio orientado a conexión (TCP) se deben crear las primitivas siguiendo la siguiente estructura:

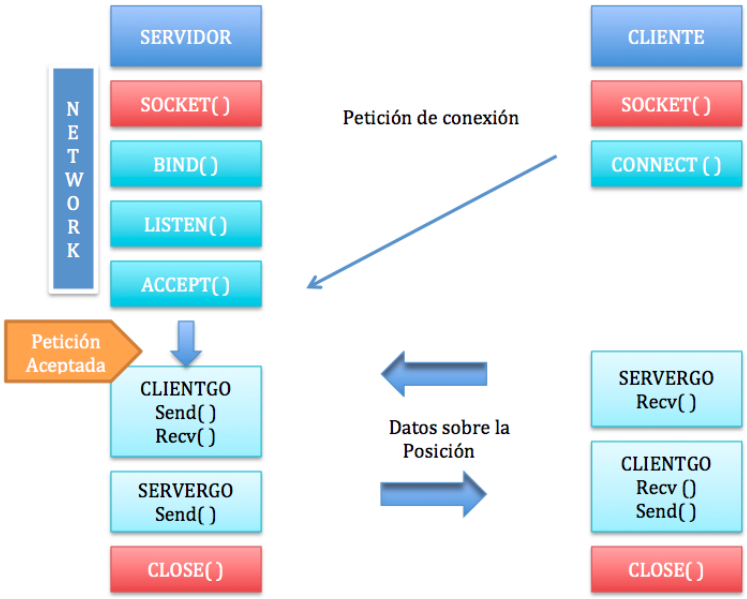

Figura 37. Diagrama del servicio TCP.

Donde las principales primitivas en este tipo de servicio son las siguientes:

-**Listen:** Pone al socket en modo pasivo y crea una cola de espera para almacenar un número máximo de solicitudes de conexión.
-**Accept:** Espera una solicitud de conexión.
-**Connect**: Inicia la conexión con el conector remoto, recibiendo como parámetros la dirección y el puerto de destino.
-**Shutdown:** Deshabilita la recepción y el envío de datos por el socket.
-**Send:** Envía el mensaje.
-**Recv:** Recibe el mensaje.

En la función Network del fichero TTTGame del juego es donde se procede a crear el socket y este estará bloqueado hasta recibir una petición de conexión por parte del cliente. Ya una vez gestionada la petición hecha por el cliente se comenzará a enviar y a recibir información.

En este caso las primitivas de enviar y recibir se realizan dentro de las funciones ClientGo y ServerGo, y dependiendo de quién sea el turno se llamará a una u a otra. En el servidor la función ServerGo sólo tendrá como misión enviar información sobre las posiciones que se van tomando en el juego, mientras que en el ClientGo el servidor envía

los datos de respuesta y recibe la información sobre las posiciones tomadas en el tablero del cliente.

De igual forma ocurre en el caso del cliente, pero en este caso en el ServerGo sólo se implementa la primitiva de recibir información, y en el ClientGo de recibir y enviar.

Para la transmisión del video se ha usado un servicio de transporte UDP. En este tipo de servicio no se establece conexión. Es más eficiente que el servicio TCP, pero no garantiza fiabilidad ya que sus datos se envían y reciben en paquetes independientes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un diferente orden al que se envió.

En UDP hay un límite de tamaño de datagramas, establecido en 65,5 kilobytes.

Aún así, cuando se requiere como en este caso para una transmisión de datos en tiempo real, es más eficaz el uso de datagramas que aunque no garanticen un servicio óptimo resultan un mecanismo muy útil para no perder un tiempo adicional en la verificación de datos. Además de ser un servicio menos complejo y con menor sobrecarga en la conexión.

La estructura que se realiza de éste servicio es la siguiente:

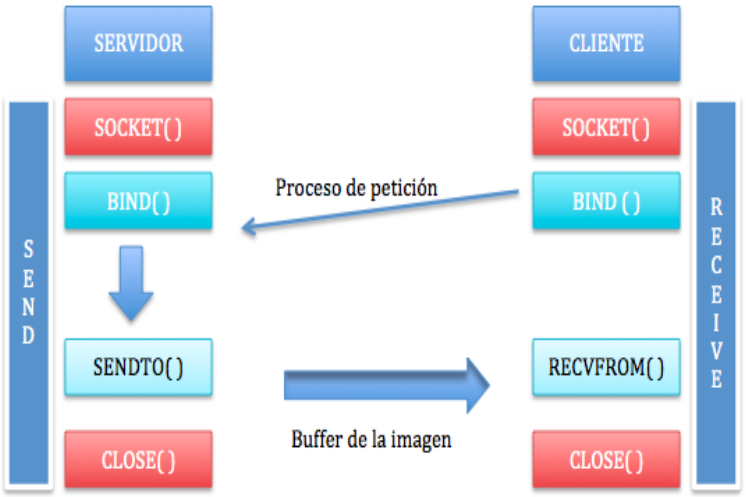

Figura 38. Diagrama del servicio UDP.

UDP emplea un número menor de primitivas:

- Sendto: Envía el mensaje.
- Recvfrom: Recibe el mensaje.

En el servidor en la función send se crea el socket indicando la dirección del cliente. Una vez implementado el socket se procede a realizar la captura de video y enviar las imágenes a través de la primitiva sendto, asegurándose de que el buffer de la imagen no sobrepase el tamaño de 65500 bytes.

El cliente en la función receive implementa la descripción del socket y con la primitiva recvfrom recibe la información, rellenando así el buffer de la imagen que será mostrada.

Para poder ejecutar ambos servicios de forma simultánea hay que crear una aplicación multihebra, de esta forma cada tarea se ejecutará individualmente dentro del sistema.

En esta ocasión la aplicación se ha diseñado con dos procesos paralelos que funcionan de forma independiente, las funciones del juego por un lado, y el envío y recepción del video procedente de la webcam por otro. Lo cual resulta muy beneficioso en términos de desacoplamiento de código (tareas independientes se implementan por separado) y en calidad del diseño (frente a futuras ampliaciones o modificaciones).

Para crear una hebra la API de Windows proporciona la función CreateThread. Cada hebra cuenta con una pila propia, a la cual se le puede especificar el tamaño que deseemos en el segundo argumento de la función CreateThread. En el tercer argumento, y más importante, va a consistir la llamada de la función en cuestión que queremos que la hebra ejecute, por lo que para la ejecución del video se llamarán a las funciones que corresponden con la emisión y recepción del video, mientras que para las funciones del juego se llamarán a las funciones ClientGo y ServerGo, que son las que gestionan la recepción y envío de información para el posicionamiento de las fichas en el tablero.

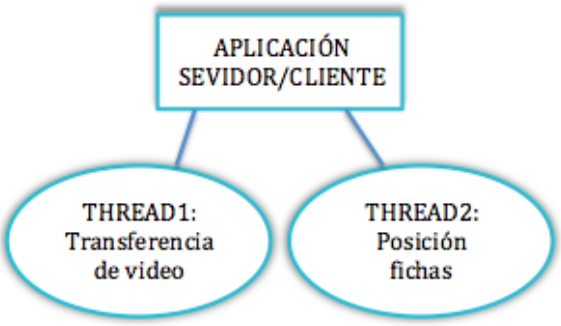

Figura 39. Esquema hebras implicadas en el Juego.

5.4 TRANSFERENCIA DE VÍDEO

Para la transmisión de video se ha utilizado como dispositivo una webcam y las librerías de video OpenCV.

La captura de video y su posterior visualización se ha desarrollado de forma independiente, procediendo a la captura en la aplicación servidor y enviando el video en imágenes comprimidas en formato JPEG a la aplicación cliente, donde se descomprimen las imágenes y son actualizadas, mostrándose en pantalla con la ayuda de Ogre.

OpenCV tiene una estructura modular, ya mencionado anteriormente, lo que significa que se deben incluir varias bibliotecas compartidas o estáticas en nuestra aplicación, dependiendo de las funciones que deseemos desempeñar.

En este caso para el desarrollo e implementación de la captura de video se hacen uso de las bibliotecas CV, CXCORE y HighGUI, las cuales engloban las estructuras y funciones que son necesarias para la captura del video, la compresión de imágenes en JPG y luego su posterior descompresión.

En el fichero B3D_VideoTexture del servidor se define la función send, en la cual se declaran y se crean los sockets UDP para enviar la información del video. Con el método de cvCaptureFromCam (int device) o cvCreateCamaraCapture (int device) podemos empezar a capturar desde nuestra webcam. El parámetro device consiste en el identificador de nuestro dispositivo de captura de video abierto (es decir, un índice de la cámara). Si sólo hay una cámara conectada, por defecto se tiene que usar el valor de 0, pero si puede haber varias cámaras conectadas se le da el valor de -1, también se puede especificar la cámara que queremos usar indicando su nombre específico. La información proveniente de la webcam se guarda en la clase CvCapture, esta clase se emplea en la lectura de archivos de video y en secuencias de imágenes.

Para almacenar la información recogida de la webcam y después poder procesar las imágenes recogidas por ella se hace uso de dos tipos de estructuras claves en el uso de las bibliotecas de OpenCV.

- Cv::Mat. La clase Mat representa una matriz n-dimensional. Donde se pueden almacenar vectores y matrices, ya sean de imágenes a color o en

escala de grises. Consta de dos partes de datos: la cabecera de matriz y un puntero a la matriz que contiene los valores de píxel (dimensión que puede tomar dependiendo del método elegido).

- Estructura IplImage. Esta estructura se definió originalmente como parte de la imagen de Intel Biblioteca Procesamiento (IPL). Es una subclase de CvMat.

Es importante hacer uso de este tipo de estructuras ya que las funciones de OpenCV requieren que los tamaños de las imágenes o de las ROI[20] de las imágenes fuente y destino coincidan exactamente.

La función cvQueryFrame devuelve al puntero del búfer interno de Opencv la imagen en una estructura del tipo IplImage del último fotograma capturado, llenando así el búfer del socket.

Para proceder al envío del video se comprimen las imágenes capturadas, guardándolas en la clase Mat. Una vez pasada la imagen al formato Mat con la función imencode comprimimos la imagen y la almacenamos en el búfer de memoria que cambiará de tamaño para ajustarse al resultado. Esta función consta de los siguientes parámetros:

- **Ext:** Extensión del archivo que define el formato de salida. Se ha usado la extensión ".jpg".
- **Img:** Imagen donde será escrita. Será de la clase Mat.
- **Buf:** Búfer de salida que cambia de tamaño para adaptarse a la imagen comprimida.
- **Params:** Parámetro del formato específico de la imagen. Dependiendo del valor que se le asigne a este parámetro variaremos la calidad de la imagen y el tamaño comprimida. Este parámetro será manipulado en la sección de pruebas y resultados.

Teniendo en cuenta que en la utilización de un servicio de transporte UDP existe un límite en el tamaño de los datagramas, debemos controlar que el búfer de las imágenes no sobrepasen este límite. Ya una vez comprobado el tamaño de los datagramas, se enviarán las imágenes comprimidas con la primitiva sendto.

[20] Regiones de interés.

Finalmente con la sentencia cvReleaseCapture, se libera y se cierra la captura de video.

En la aplicación cliente, también en el mismo fichero B3D_VideoTexture, es donde con la función receive se procede a recibir las imágenes enviadas del servidor y se descomprimen para posteriormente mostrarlas. Con la primitiva de UDP recvfrom recibimos la información de las imágenes que se irá almacenando con la función imdecode en una estructura del tipo Mat, y ya con la función cvCloneImage pasamos la imagen al tipo de estructura IplImage, obteniendo ya la imagen final descomprimida.

Posteriormente en la función Update se actualizan las imágenes recibidas y se crea el cuadro donde se mostrarán, adaptando a cada imagen al tamaño del cuadro. Esta función será llamada en el fichero GraphicsManager una vez creadas las escenas y se empiece a recibir información del servidor.

5.5 DIFICULTADES ENCONTRADAS EN LA IMPLEMENTACIÓN

En la implementación del juego han existido ciertas dificultades a la hora de desarrollar algunas de las partes que componen el juego. Aquí se explicaran de forma resumida cuales han sido los principales inconvenientes encontrados que han dificultado y retrasado el desarrollo de este trabajo.

En el desarrollo y el correcto funcionamiento de las hebras que se implementan en el juego se han tenido complicaciones, ya que al tratarse de una aplicación multihebra pueden existir problemas de concurrencia que dificulten la implementación de dos procesos secuenciales. Este ha sido uno de los problemas encontrados en la creación y vinculación de las hebras que se encargan de gestionar el envío y recepción de la información del posicionamiento de las fichas en el juego, este problema se debe a la coordinación que debe garantizarse entre las distintas hebras. Para garantizar esta correcta coordinación se hace uso del mecanismo de sincronización basado en un semáforo, este mecanismo funciona como funcionaría un semáforo. La variable de la hebra está declarada como un parámetro booleano, por lo tanto si no es el turno del

jugador la hebra estará bloqueada siendo su valor *false*, en caso de que sea el turno del jugador la hebra toma el valor de *true* y deja que el proceso continúe, devolviendo al final el valor de *false* otra vez a la hebra. No obstante otro factor importante que ha complicado el correcto funcionamiento del juego ha sido la implementación de las variables empleadas en la creación de la hebra, ya que estas deben ser declaradas una sola vez para que no ocasionen problemas de concurrencia entre ellas.

También la parte de integración de Ogre con OpenCV ha resultado algo compleja puesto que para mostrar la imagen obtenida por OpenCV la imagen que se obtiene debe de implementarse del tipo de formato IplImage y ser del tipo estático para que posteriormente con Ogre crear una textura manual cuadrada de dos dimensiones, especificando altura y anchura, obteniendo así el PixelBuffer de la textura y actualizarlo con Ogre.

6. PRUEBAS Y RESULTADOS

En las próximas páginas se recogen las pruebas realizadas al juego en diferentes condiciones y valorando así los resultados obtenidos.

6.1 RECURSOS Y LIMITACIONES

Para la realización de las siguientes pruebas y su posterior valoración hay que tener en cuenta los equipos empleados, ya que de ellos dependerán los resultados obtenidos.

El primer factor que se ha de tener en cuenta para un rendimiento y funcionamiento óptimo del juego son los equipos empleados puesto que para realizar estas pruebas se han utilizado dos equipos de diferentes características.

Tabla 1. Equipos software y hardware empleado.

TIPO	EQUIPO 1	EQUIPO 2
HARDWARE	Procesador 3200 + Velocidad procesador 2GHz Tarjeta Gráfica NVIDIA GeForce 6200 Disco duro 90GB	Intel Core 2 Duo Velocidad procesador 2,4 GHz Número de núcleos: 2 Tarjeta gráfica NVIDIA GeForce 320M Disco duro 500GB
SOFTWARE	Microsoft Windows XP OpenCV2.1 OgreSDK vc9 v1.7.2	Sistema Operativo Windows 7 OgreSDK vc9 v1.7.2 OpenCV2.1

Otro factor que se ha de tener en cuenta es el tipo de conexión en red. En este caso se ha utilizado un cable cruzado para realizar la conexión entre ambos equipos por el puerto Ethernet, proporcionando una velocidad máxima de 1Gbps. De esta forma se ha intentado utilizar un tipo de conexión en red lo más óptima posible.

6.2 PRUEBAS

Se ha usado la herramienta que nos proporciona Ogre para obtener los valores de la frecuencia de renderizado en tiempo real donde en el siguiente panel (ver Fig. 40) nos va mostrando los resultados obtenidos para los distintos valores de frames por segundo. Nos muestra en primer lugar la media, en segundo y en tercer lugar nos muestra el mejor y el peor valor de FPS registrados hasta el momento.

Figura 40. Panel de Ogre.

Aparte de los parámetros que nos aporta el motor gráfico Ogre, también se han incorporado en estas pruebas posteriores otros parámetros configurables que son cambiados manualmente en la parte de la aplicación del servidor. Estos parámetros pertenecen a las propiedades del vídeo y la frecuencia con la que se envía la información recogida por la webcam.

Uno de ellos son los valores de las imágenes por segundo que corresponden con la frecuencia con la que se envía información a la otra parte de la aplicación. Se han tomado valores que van desde el envío de una única imagen por segundo hasta el envío de 30 imágenes por segundo. Este parámetro se ha implementado como la diferencia de tiempos desde que se envía la primera imagen hasta el tiempo que debe transcurrir para que se produzca el envío de la siguiente imagen dependiendo del período de tiempo que se desee.

El valor de los FPS proporcionados por el motor gráfico de Ogre influenciaran en la velocidad de renderizado, proporcionando al usuario más rapidez y agilidad a mayor número de frames por segundo en sus movimientos con los elementos del juego, ratón, teclado y fichas, Por otro lado el número de imágenes por segundo con el que se

transmite el vídeo afectará al usuario en la percepción del vídeo, a más imágenes por segundo existirá una mejor percepción de los movimientos del usuario en tiempo real.

En la siguiente gráfica se muestran los peores valores del período de renderizado para los distintos valores de imágenes por segundo enviadas.

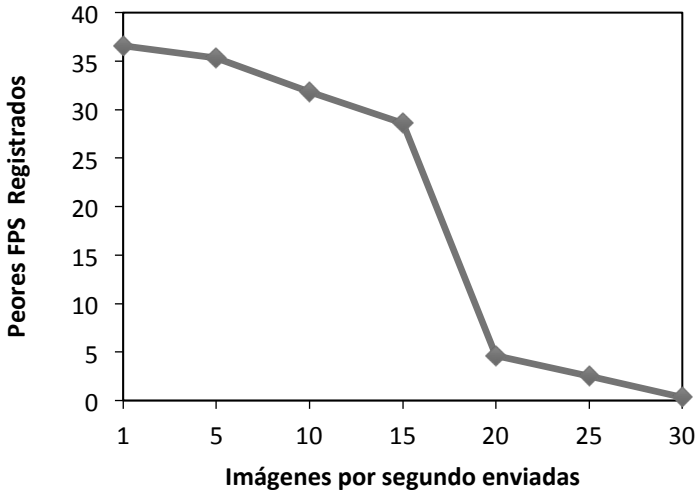

Figura 41. Gráfica de Imágenes enviadas por segundo para los peores FPS.

Observando la gráfica anterior (Fig.41) vemos que en el envío de una única imagen por segundo se obtiene como valor más bajo 36,6 fps, mientras que en el envío de 30 imágenes por segundo se obtiene un valor mínimo de 0,3 fps en Ogre, provocando que el rendimiento en el juego sea peor, dificultando la partida y provocando una imposible interacción con los elementos del juego en tiempo real. No obstante, vemos en la gráfica anterior que la zona entre 10 y 15 imágenes por segundo es la más estable, obteniendo una frecuencia mínima de renderizado óptima para el funcionamiento del juego. Además, estos valores proporcionan una calidad de información transferida bastante razonable, ya que teniendo en cuenta que la misión del vídeo es capturar la imagen de un jugador en tiempo real y no de proporcionar una gran nitidez y exactitud en la percepción de sus movimientos, no es necesario un envío de más cantidad de información del que se ha probado. Es por ello que se ha elegido el valor de 15 imágenes por segundo, ya que este nivel de transmisión proporciona una calidad al vídeo que se ajusta a lo requerido.

Capturando el tráfico por la aplicación Wireshark podemos observar que el protocolo que más ancho de banda consume es el UDP, mientras que TCP apenas

consume ancho de banda. Esto se debe a que la información que transporta UDP, al tratarse de imágenes, tiene más peso que la información que transporta TCP, las posiciones de las fichas en el tablero. Podemos observar que el ancho de banda que consume está en torno 29Kbit/s.

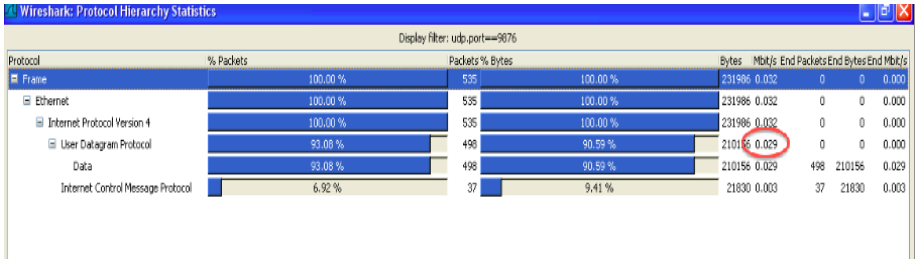

Figura 42. Estadísticas Wireshark UDP.

El segundo parámetro con el que vamos a realizar las siguientes pruebas es el parámetro de la calidad de compresión que nos proporciona la librería OpenCV a la hora de transmitir los frames capturados en formato JPEG. Este parámetro se específica en el método imencode de OpenCV. Esta función guarda la imagen en el buffer especificado pasándole un valor que asigna para JPEG un nivel de calidad que puede ir de 0 a 100 (mientras más alto menor compresión y, por tanto, mejor calidad).

Ahora variando la calidad del vídeo observamos el tráfico en Wireshark para comprobar si se obtienen variaciones en el ancho de banda. Fijando la transmisión en 15 imágenes por segundo observamos en la siguiente gráfica (Fig.43) los distintos valores que se obtienen de ancho de banda para cada tipo de calidad de compresión, obteniendo unos resultados en torno a los 35Kbit/s con los primeros valores y conforme se aumentan la calidad de compresión del vídeo el ancho de banda se ve influenciado, aumentando hasta llegar a 41Kbit/s.

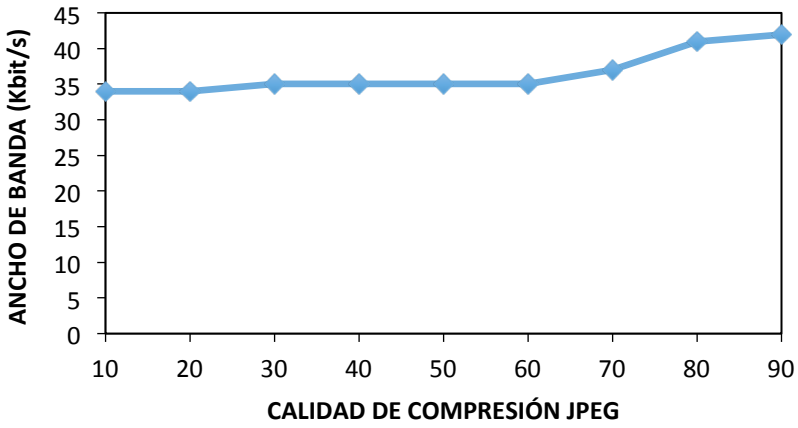

Figura 43. Gráfica Ancho de banda respecto con la calidad.

A continuación en las siguientes pruebas (Fig. 44 y 45) se han registrado los datos obtenidos del renderizado obtenidos en Ogre para ambas partes de la aplicación, cliente y servidor, teniendo en cuenta que la segunda escena es distinta para cada aplicación, por lo tanto el motor gráfico de Ogre no se comporta igual con la misma eficacia y rapidez en ambas aplicaciones. Perteneciendo los valores más altos (primera línea) a la aplicación servidor, y los valores más bajos (segunda línea) a la parte de la aplicación cliente.

Los resultados obtenidos son estimados y pueden oscilar dependiendo de las circunstancias de cada partida, ya que dependen de los siguientes factores externos: el equipo, la duración de la partida, y la transmisión en red.

En la primera gráfica (Fig. 44) se registra el promedio de frames por segundos que se han obtenido en cada parte de la aplicación dependiendo de la calidad de compresión del vídeo. Finalmente en la última gráfica (Fig.45) se registran los valores de los peores fps obtenidos a lo largo de una partida para distintas calidades de vídeo. Siendo en el servidor el peor resultado en 77 fps, y en el cliente en 34 fps.

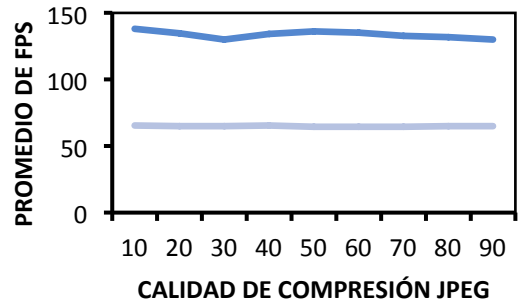

Figura 44. Gráfica media de FPS registrados en Ogre.

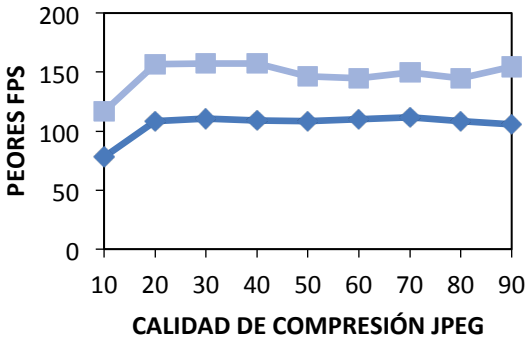

Figura 45. Gráfica peores FPS registrados en Ogre.

Como se puede observar para ambas partes la media de fps se mantiene muy constante, pero vemos que en el servidor se obtiene mejores resultados de renderizado frente con el cliente que obtiene un promedio en torno de 65 fps.

Por lo tanto, en ambas gráficas vemos que la calidad con la que se transmite y se muestra el video no va a influir de forma significativa en el proceso del renderizado en Ogre. En todas las gráficas conforme se aumenta la calidad de compresión del video no se obtienen grandes variaciones a la hora de renderizar, pero si podemos ver que existe una gran diferencia en la frecuencia de renderizado entre la aplicación cliente y la aplicación servidor, viendo así que el rendimiento en el cliente se ve afectado al mostrar en tiempo real la imágenes recibidas por el servidor, ralentizando el renderizado de la aplicación, mientras que mostrar la cara del avatar en movimiento no supone una gran carga, por lo que en el equipo del servidor el rendimiento es mejor que en el cliente.

7. CONCLUSIONES Y LÍNEAS DE FUTURO

En este trabajo se han alcanzado los objetivos propuestos, los cuales engloban el desarrollo de un entorno virtual en el que dos jugadores puedan interactuar a partir de una arquitectura cliente/servidor creada mediante la programación con sockets, así como la inclusión de la transferencia de video por una webcam y su posterior visualización con la ayuda de las herramientas proporcionadas por la librería OpenCV.

A continuación, se detalla el grado en el que se han cumplido los objetivos de partida del trabajo:

- Posibilidad de crear un entorno 3D con OGRE, en particular un entorno basado en un juego que permite una interacción entre dos usuarios en el mismo.
- Las tecnologías usadas (OGRE, Blender y OpenCV) presentan unas características idóneas para este tipo de aplicaciones frente a las presentadas en el estudio detallado del estado del arte sobre las herramientas que se han utilizado y las soluciones que para tal propósito existen en la actualidad en el mercado.
- Viabilidad de integrar avatares animados creados con Blender u otro modelador gráfico en el entorno virtual desarrollado.
- La integración de OpenCV y OGRE para la transferencia de video en tiempo real en el entorno virtual del juego, pudiendo elegir la calidad de video que se desee.
- Mediante la utilización de la API OIS de OGRE se ha comprobado que se pueden combinar varios periféricos, como el ratón y el teclado, para complementar las funcionalidades de la aplicación.
- Se ha conseguido mediante la arquitectura cliente/servidor proporcionada por la librería Winsock, la interacción satisfactoria entre ambos jugadores en tiempo real.
- La posibilidad de que todo el software necesario para la implementación propiamente dicha de la aplicación sea open source, comprobando que además se adaptan a las necesidades requeridas.

Ante los resultados obtenidos tras el desarrollo del juego, hemos comprobado la infinidad de herramientas que existen actualmente y la gran variedad de recursos que pueden ofrecer.

Además, una de las principales ventajas que se ha comprobado es la viabilidad de crear un entorno virtual con la integración de dispositivos de RV con los que los distintos jugadores puedan interaccionar e incluir su propia imagen o movimientos en tiempo real al juego.

Finalmente también se ha podido comprobar que hoy en día existen diferentes comunidades en internet donde se desarrollan librerías de código abierto para la implementación de nuevas funciones en futuras aplicaciones.

7.1 LÍNEAS DE FUTURO

Dada la limitación temporal para el desarrollo de esta aplicación, podríamos decir que han quedado abiertas algunas líneas de trabajo que permitirían mejorarla. Se enumeran a continuación esas posibles mejoras:

- Aumento del realismo del entorno virtual añadiendo otras propiedades físicas.
- Aumento de la complejidad en la interacción con el entorno, mediante la posibilidad de manipular otros objetos que se encuentren en la escena.
- Creación de un menú en el que se recojan los rankings de distintos jugadores.
- Integración de sonido en el juego para conseguir un mayor realismo.
- La integración del envío de audio a través de la webcam entre ambos jugadores.
- El uso de una codificación de video para la transmisión de las imágenes, como mpeg ó H.264.
- Integración de la posibilidad de elegir entre varios avatares más personalizados.
- Comparativa de esta aplicación con otra exactamente igual pero realizada con otros mecanismos.
- Distribución en red, como en las plataformas de juego online.
- Modificación de las imágenes enviadas, ya que la librería OpenCV tiene herramientas que permiten el tratamiento de imágenes.

Además de todo lo mencionado anteriormente hay que destacar otra línea futura que se podría ajustar bastante a este trabajo. Consistiría en la posibilidad de añadirle al avatar mostrado en pantalla en tiempo real la función de tracking, esta función permitiría al jugador mover el avatar según los movimientos que realice ante una cámara, viéndose aumentada la involucración de los jugadores en el juego, y a su vez produciría una mayor interacción por parte de ambos jugadores entre el entorno virtual y el real. La posibilidad de captación y análisis de movimientos en tiempo real que ofrece la librería de OpenCV.

8. PLIEGO DE CONDICIONES Y ESTUDIO ECONÓMICO

Se denomina pliego de condiciones a un documento contractual, de carácter comprensivo y obligatorio donde se establecen las condiciones o cláusulas que se aceptan en un contrato de obras o servicios, una concesión administrativa, una subasta, etc.

En un pliego de condiciones se indica cómo y con qué hay que hacer realidad los proyectos de obras y servicios que se contratan. En el pliego que se concuerda y firma, contiene las relaciones que existirán y que tienen que cumplirse, entre el propietario y el ejecutor de cualquier proyecto, servicio o concesión administrativa. Este documento debe contener toda la información necesaria para que el proyecto llegue a buen fin de acuerdo con las características constructivas del mismo, indica las condiciones generales del trabajo, la descripción y características de los materiales a utilizar, los planos constructivos de existir estos, y la localización de la obra o servicio. También señala los derechos, obligaciones y responsabilidades de las partes que los suscriben. Señala, así mismo, como se desarrollará el trabajo y como se resolverán los conflictos que puedan surgir. Normalmente los pliegos de condiciones se dividen en varias partes, siendo las más usuales las siguientes:

- Pliego de condiciones generales: esta parte del documento debe incluir la descripción general del contenido del proyecto, los criterios o aspectos normativos, legales y administrativos a considerar por las empresas que intervengan, listados de planos que componen el proyecto, etc. En España la normativa que se aplica es los pliegos de condiciones generales es la norma UNE 24042.
- Pliego de especificaciones técnicas: dispone de dos apartados perfectamente diferenciados:
    - Especificaciones de materiales y equipos: donde deben estar bien definidos todos los materiales, equipos, máquinas, instalaciones, etc. que se utilizarán en el proyecto. La definición se hará en función de códigos y reglamentos reconocidos. Las especificaciones hacen referencia a Normas y Reglamentos nacionales tipo (UNE, Normas MOPU, NBE, etc.) o internacionales (DIN, ISO, etc.).
    - Especificaciones de ejecución: es este apartado del pliego se hace constar como será realizado el proyecto, es decir, su proceso de fabricación o construcción a partir de los materiales que serán utilizados.

- Pliego de cláusulas administrativas: en este apartado del pliego se determina la forma de medir las partes ejecutadas del proyecto, valorarlas y pagarlas.

En este caso, se presentará un pliego de condiciones técnicas donde se realizará un breve resumen de las especificaciones materiales y equipos necesarios para la ejecución de este trabajo.

8.1 PLIEGO DE CONDICIONES TÉCNICAS

Estas condiciones fueron detalladas en el apartado de pruebas y resultados, ya que era necesario conocer previamente de qué características técnicas se disponían para la realización de las distintas pruebas y su posterior discusión. Haciendo hincapié en lo dicho en el apartado anterior se aconseja usar los componentes ya descritos para asegurar un correcto funcionamiento de la aplicación.

Por otra parte, en cuanto a las condiciones técnicas de la aplicación hay que destacar, que debe ser intuitiva y visual, facilitando su comprensión por parte de los usuarios que la manejen. No obstante, su capacidad de actualización ha de ser ágil para poder obtener resultados en tiempo real con la mayor rapidez posible. Y por último, destacar que su proceso de instalación, así como su mantenimiento ha de ser sencillo, evitando problemas.

8.2 ESTUDIO ECONÓMICO

El estudio económico trata de determinar la cantidad de recursos económicos que son necesarios para que el proyecto se realice, es decir, cuánto dinero se necesita para que la plataforma opere. El presupuesto se desglosará en partes identificadas por el tema que engloban y un resumen total.

A continuación se detalla el importe al que asciende el presupuesto del proyecto.

8.2.1 Costes de Materiales

Se denominan costes materiales a aquellos derivados de las herramientas físicas necesarias para implementar la labor de este trabajo. Seguidamente, se expone un presupuesto estimado de los equipos utilizados por el cliente y servidor.

Tabla 2. Costes hardware.

UD.	CONCEPTO	P.UNITARIO	SUBTOTAL
1	Ordenador de mesa Procesador 3200 + Velocidad procesador 2GHz Tarjeta gráfica NVIDIA GeForce 6200 Disco duro 90 GB	Torre: 200 € Pantalla: 120 € Teclado y ratón: 30 €	350 €
1	Portatil Inter Core 2 Duo Velocidad procesador 2,4 GHz Número de núcleos: 2 Tarjeta gráfica NVIDIA GeForce 320M Disco duro 500 GB	700 €	700 €
1	Cámara Webcam (en caso de no tener ninguno de los equipos)	25 €	25 €
		Importe total:	1.075 €
		IVA 21%:	226 €
		Total con IVA:	**1.300,8 €**

8.2.2 Costes Técnicos

En este apartado se presenta el coste técnico del software utilizado para la puesta en marcha y funcionamiento de la aplicación. Seguidamente, se expone el presupuesto estimado.

Tabla 3. Costes técnicos.

UD.	CONCEPTO	P.UNITARIO	SUBTOTAL
1	Sistema Operativo Windows XP/ 7	30 €	30 €
1	Microsoft Visual C++ 2008 Espress Edition	0 €	0 €
1	Blender 2.49b	0 €	0 €
1	OgreSDK vc9 v1.7.2	0 €	0 €
		Importe total:	30 €
		IVA 21%:	6,3 €
		Total con IVA:	**36,3 €**

8.2.3 Honorarios

Estimando que la duración en el desarrollo y creación de la aplicación pueden suponer una duración de 3 meses aproximadamente, empleando únicamente los días laborales se queda un promedio de 20 días laborables por mes, con un promedio de 8 horas laborales.

Calculando salario, seguridad social y otros gastos indirectos estimamos que el costo por hora de trabajo sería 25€.

Por lo tanto supondría un coste de:

60 días laborables x 8 horas diarias x 25€ = 4000€

8.2.4 Presupuesto Final

Tabla 4. Presupuesto Final.

UD.	CONCEPTO	P.UNITARIO	SUBTOTAL
1	Costes Materiales Costes técnicos	36,6 €	37 €
1	Costes Hardware y Software	1.300,8 €	1.301 €
1	Honorarios	4.000 €	4.000 €
		TOTAL:	**5.337,4 €**

9. BIBLIOGRAFÍA

9.1 OGRE

Libros consultados:

[1] – Gregory Junker. “Pro OGRE 3D Programming, leverage the power of modern real-time hardware-accelerated 3D graphics with the best-in-class 3D graphics library”. Apress.

[2] – “Manual de Ogre3D”. Universidad Carlos III de Madrid.

[3] – FelixKerger. “OGRE 3D 1.7 Beginner’s Guide 2010”. PACKT Publishing.

[4] – IlyaGrinblat y Alex Peterson. “OGRE 3D 1.7 Application Development Cookbook”. PACKT Publishing.

[5] – David Vallejo Fernández y Cleto Martín Angelina. “Arquitectura del motor de videojuegos”. Universidad de Castilla-La Mancha.

[6] – Carlos González, Javier A. Albusac, César Mora y Sergio Fernández. “Programación Gráfica”. Universidad de Castilla-La Mancha.

[7] – Francisco Moya, Carlos González, David Villa, Sergio Pérez, Miguel A. Redondo, César Mora, Félix J. Villanueva y Miguel García. “Técnicas Avanzadas”. Universidad de Castilla-La Mancha.

[8] – F. Jurado, J.J Castro, J. Albusac, D. Villa, C. González, G. Simmross y L. Jiménez. “Desarrollo de Componentes”. Universidad de Castilla-La Mancha.

Páginas Web visitadas:

[9] - Página con documentación e información de apoyo para OGRE:
http://www.ogre3d.org/tikiwiki/tiki-index.php

[10] – Página con Tutoriales Básicos de OGRE:
https://cubansephiroth.wordpress.com/category/ogre-3d/

[11] –Página principal de OGRE:
http://www.ogre3d.org/

[12] – Introducción a OGRE:
http://isaaclacoba.github.io/tinman/posts/introduccion-ogre3d/introduccion-a-ogre3d.html

[13] – Clases OGRE:
https://sites.google.com/site/ogreesp/api-1/modulos/api-general

[14] – Foro de OGRE:
http://www.ogre3d.org/forums/

9.2 ARQUITECTURA CLIENTE/SERVIDOR

Páginas Web visitadas:

[15] – Programación Básica con Sockets:
http://es.tldp.org/Tutoriales/PROG-SOCKETS/prog-sockets.html
[16] – Protocolos UDP y TCP:
https://es.wikibooks.org/wiki/Redes_inform%C3%A1ticas/Protocolos_TCP_y_UDP_en_el_nivel_de_transporte

9.3 OPENCV

Libros consultados:

[17] -F. Jurado, J.J Castro, J. Albusac, D. Villa, C. González, G. Simmross y L. Jiménez. "Desarrollo de Componentes". Universidad de Castilla-La Mancha.
[18] – Carlos González, David Vallejo, Javier A. Albusac y José J. Castro. "Realidad Aumentada: Un Enfoque Práctico con ARToolKit y Blender". Bubok Publishing S.L.

Páginas Web visitadas:

[19] – Página de OpenCV:
http://docs.opencv.org/2.4/index.html
[20] – Manejo de bibliotecas OpenCV:
http://www2.electron.frba.utn.edu.ar/~afurfaro/Info1/Opencv/opencv.pdf
[21] – Foro OpenCV:
http://answers.opencv.org/question/74121/fps-for-usb-web-cam/
[22] – Tutorial OpenCV para C++:
http://opencv-srf.blogspot.com.es/2011/09/capturing-images-videos.html
[23] – Algoritmos OpenCV.
http://opencvexamples.blogspot.com/2013/10/capture-video-from-camera.html

9.4 OTRAS

Páginas Web visitadas:

[24] – Historia Videojuegos:

http://www.fib.upc.edu/retro-informatica/historia/videojocs.html

[25] - Historia Videojuegos:

https://es.wikipedia.org/wiki/Historia_de_los_videojuegos

[26] – Realidad Virtual en los Videojuegos:

http://www.teknoplof.com/2014/01/20/historia-de-la-realidad-virtual-en-el-mundo-de-los-videojuegos/

[27] – Visión Artificial e interacción en los juegos:

http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/VisionArtificial/iteracion.html

[28] – Ivan Sutherland:

https://es.wikipedia.org/wiki/Ivan_Sutherland

[29] – Historia de los videojuegos:

https://es.wikipedia.org/wiki/Historia_de_los_videojuegos

[30] – Programación con Hebras:

http://elvex.ugr.es/decsai/builder/threads/

10. ANEXOS

A. Manual de Instalación.

A.1 Microsoft Visual C++ 2008 Express Edition.

Desde la página oficial de Microsoft (www.microsoft.com/downloads/es-es/default.aspx), puede obtener una versión gratuita de este compilador.
Su instalación es bastante sencilla, solo tiene que hacer "doble-clic" en el archivo obtenido, aunque requiere en principio de una conexión a Internet para la descarga de los paquetes que forman la aplicación. Una vez acabada puede continuar la instalación sin conexión.

Como en cualquier otra, se le pedirá que acepte la licencia de uso y se le creará una carpeta en la unidad C, donde se instalará el programa, añadiendo al final del proceso un acceso directo en el menú de Inicio de Windows.

A.2 OGRE y OpenCV.

Ambas librerías han sido descargadas de sus páginas oficiales: http://www.ogre3d.org http://opencv.org/downloads.html

Una vez descargadas se extraen ambas en el disco duro C.

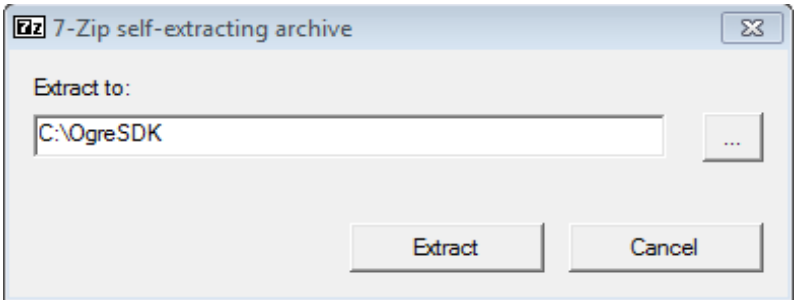

Figura 46. Ubicación librería OgreSDK.

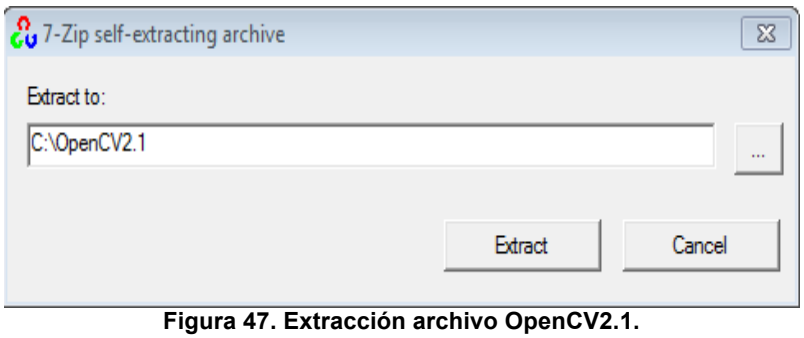

Figura 47. Extracción archivo OpenCV2.1.

Una vez extraídas hay que asegurarse de que se han agregado las variables de entorno de ambas. Por defecto la de OpenCV se crea sola, pero no pasa lo mismo con Ogre.

Para ello nos vamos al Panel de Control del sistema y ya en Sistema y Seguridad pinchar en la opción de configuración avanzada del sistema.

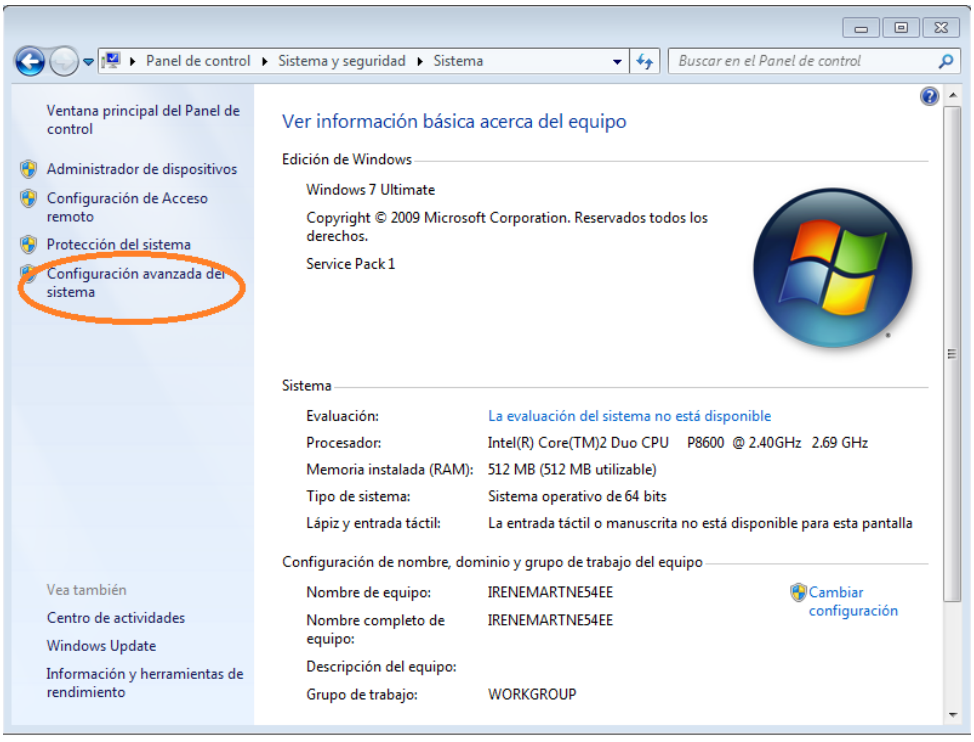

Figura 48. Panel de control de sistema.

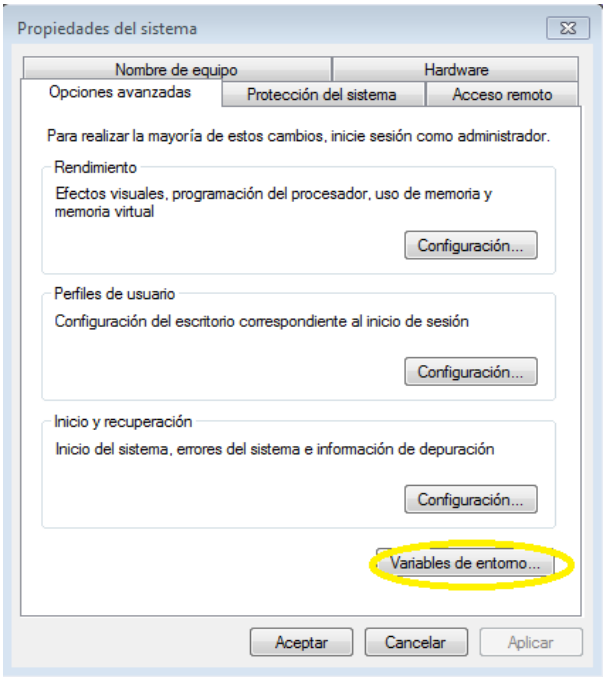

Figura 49. Variables de entorno.

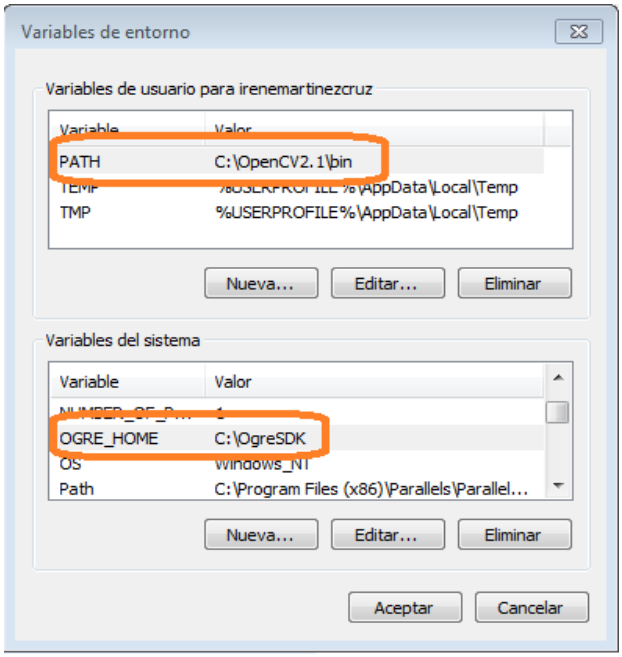

Figura 50. Variables de entorno declaradas.

Finalmente, una vez comprobado que están agregadas ambas variables de entorno es necesario reiniciar el equipo.

En el Visual Studio 2008 también es necesario irse a las propiedades de nuestro proyecto y allí vincular las librerías ya extraídas anteriormente para que nuestra aplicación pueda hacer uso de ellas.

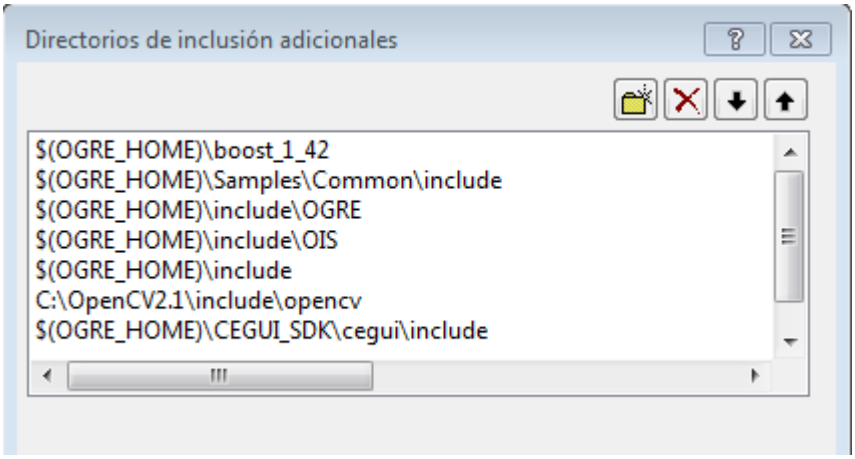

Figura 51. Librerías incluidas en el modo Release.

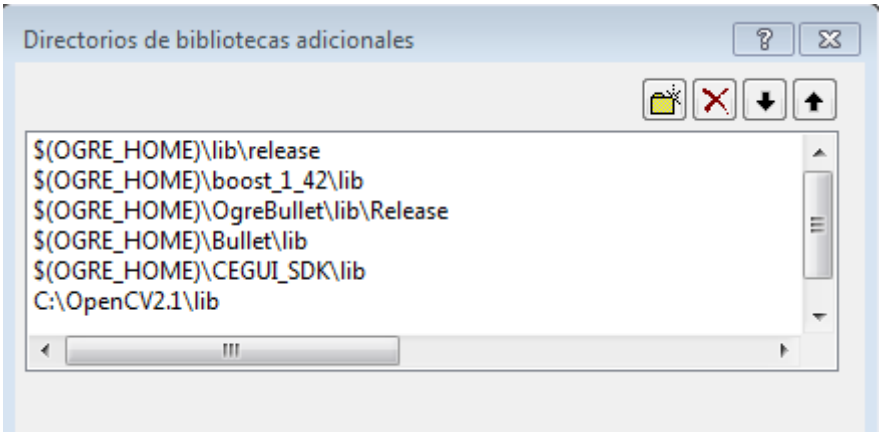

Figura 52. Librerías incluidas en el modo Debug.

A.3 Microsoft DirectX.

Es necesario instalar un conjunto de librerías multimedia de Microsoft. Estas librerías se encargan de poner en comunicación al sistema operativo con los complementos del juego. Este Kit de desarrollo se puede descargar en la página de Microsoft (https://www.microsoft.com/en-us/download/details.aspx?id=23549).

B. MANUAL DE USUARIO

1. Conecta 4.

También conocido como 4 en raya, fue creado en 1974 por Ned Strongin y Howard Wexler para Milton Bradley Company.

El objetivo del juego es alinear cuatro fichas sobre un tablero formado por 7x6 columnas. Cada jugador dispone de 21 fichas de un color. Por turnos los jugadores deberán de ir introduciendo las fichas en la columna que prefieran y ésta caerá en la posición más baja. Finalmente gana el primero en conseguir la combinación de cuatro fichas consecutivas del mismo color en horizontal, vertical o diagonal. En caso de llenarse todas las columnas y se ha hecho ninguna combinación válida, se considera que gana el tablero o lo que es lo mismo a un empate.

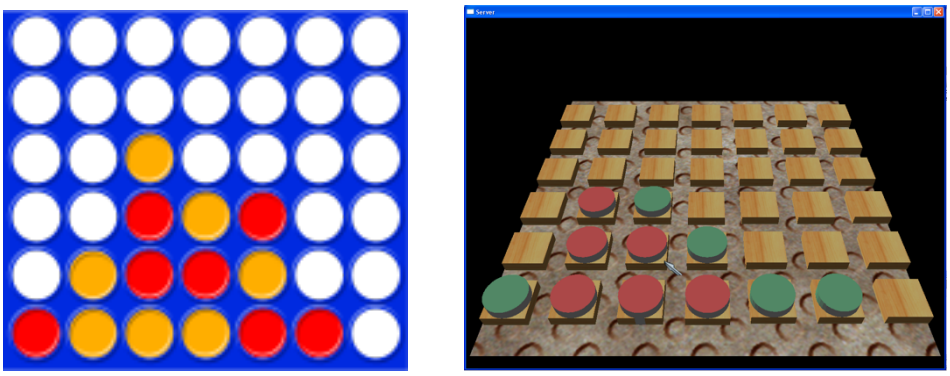

Figura 53. Tableros Conecta 4.

2. Jugadores.

En este juego intervienen dos jugadores, un jugador será el servidor y otro el cliente. El servidor manejará las fichas de color rojo y el cliente las de color verde.

3. Guía del Juego.

En esta parte se intenta describir la funcionalidad de las distintas teclas o movimientos del ratón y teclado.

A. Funciones del Teclado:

V – Ésta acción pone las dos escenas del juego en pantalla y también las intercala.

C – Intercala ambas pantallas, las cambia de posición, ya sea para mostrarlas en la parte superior de la pantalla o en la inferior.

X – Sale del Juego.

B. Funciones del Ratón:

Botón derecho – Posiciona la ficha en la ubicación señalada del tablero.

C. MANUAL DE REFERENCIA

En el siguiente diagrama se muestran las dependencias que existen entre las clases involucradas, mostrando el sentido de dependencia.

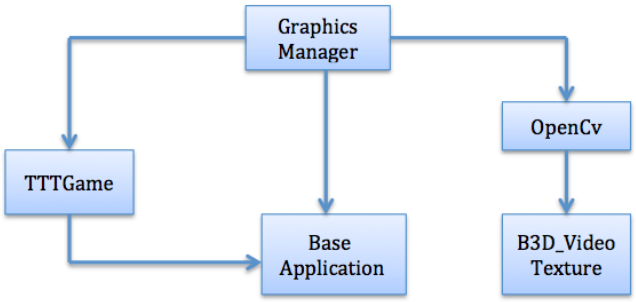

Figura 54. Diagrama clases.

A continuación se hace un listado de las funciones que se utilizan en cada una.

1. **BaseApplication.**

Se encarga de las funciones básicas para crear una aplicación con OGRE.

- **BaseApplication(void);**

  Inicializa las varibles que se utilizan en el programa.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **~BaseApplication(void);**

  Elimina varibles y el WindowListener, cierra la ventana de ejecución y borra el nodo raiz.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **void go(void);**

  Identifica los archivos de configuración que se utilizaran e inicializa el renderizado.

  Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **bool setup();**

  Crea un nodo raiz de OGRE y realiza llamadas a otras funciones para cargar recursos, la escena, etc.

  Parámetros de entrada: ninguno.

  Parámetros de salida: booleano con el resultado de la configuración.

- **bool configure(void);**

  Muestra el cuadro de diálogo de la configuración e inicialización del sistema.

  Parámetros de entrada: ninguno.

  Parámetros de salida: booleano con el resultado de la configuración.

- **void chooseSceneManager(void);**

  Define el SceneManager que se utilizará.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **void setupResources(void);**

  Carga los archivos de configuración y fuentes.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **void loadResources(void);**

  Inicializa todos los grupos de fuentes que se utilizan en el programa.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **bool keyPressed(const OIS::KeyEvent &arg);**

  Procesa el evento producido por la pulsación de una tecla, mostrando paneles con información o moviendo la cámara.

  Parámetros de entrada: información de la tecla pulsada.

  Parámetros de salida: booleano indicando que se realizó el proceso.

- **bool keyReleased(const OIS::KeyEvent &arg);**

  Inicializa la tecla despues de ser pulsada.

  Parámetros de entrada: la tecla que se pulsó.

  Parámetros de salida: true, indicando que el proceso se realizó.

- **bool mouseMoved(const OIS::MouseEvent &arg);**

  Procesa la información recibida del ratón.

  Parámetros de entrada: las características de la interrupción del ratón.

  Parámetros de salida: indicación de que se procesó mediante true.

- **bool mousePressed(const OIS::MouseEvent &arg, OIS::MouseButtonID id);**

  Analiza y procesa eventos producidos por los botones del ratón.

  Parámetros de entrada: estado del ratón.
  identificación del botón pulsado.

  Parámetros de salida: verdadero para indicar el procesado.

- **bool mouseReleased(const OIS::MouseEvent &arg, OIS::MouseButtonID id);**

  Inicializa el estado del ratón despues de su utilización.

  Parámetros de entrada: estado del ratón, identificación del botón pulsado.

  Parámetros de salida: booleano, true, confirmando el procesado.

- **void windowResized(Ogre::RenderWindow* rw);**

  Define las características de la ventana de renderizado, como el tamaño.

  Parámetros de entrada: la ventana de renderización.

  Parámetros de salida: ninguno.

- **void windowClosed(Ogre::RenderWindow* rw);**

  Cierra la ventana principal del programa.

  Parámetros de entrada: la ventana de renderizado.

  Parámetros de salida: ninguno.

2. GraphicsManager.

- **GraphicsManager(void);**

  Inicializa algunas varibles que se utilizan en la clase.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **~GraphicsManager (void);**

  Elimina las variables de de las posiciones y la cámara.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **Initialize();**

  Inicializa la clase llamando a la función go() de la clase BaseApplication.

- **void createScene(void);**

Crea todos los elementos que aparecen en ambas escenas, personajes, objetos, luces, etc.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **void createFrameListener(void);**

  Añade funcionalidad a la función del mismo nombre que hay en la BaseApplication.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **bool frameRenderingQueued(const Ogre::FrameEvent& evt);**

  Actualiza y controla las posiciones realizadas por el otro jugador. A su vez también se encarga de actualizar los movimientos del avatar y las imágenes del vídeo.

  Parámetros de entrada: información sobre el evento que se produce.

  Parámetros de salida: booleano con el resultado del proceso.

- **bool keyPressed(constOIS::keyEvent &arg);**

  Registra los eventos recogidos por el teclado cuando este se está utilizando.

  Parámetros de entrada: información sobre el evento que se produce.

  Parámetros de salida: booleano con el resultado del proceso.

- **void showResult(Ogre::String result);**

  Genera un panel de información con el resultado final del juego.

  Parámetros de entrada: cadena de caracteres con el resultado.

  Parámetros de salida: ninguno.

- **bool keyReleased(constOIS::keyEvent &arg);**

  Recoge la actividad del teclado cuando se ha dejado de utilizar.

  Parámetros de entrada: información sobre el evento que se produce.

  Parámetros de salida: booleano con el resultado del proceso.

- **bool mouseMoved(constOIS::MouseEvent &arg);**

  Actuliza los movimientos generados por el ratón.

  Parámetros de entrada: información sobre el evento que se produce.

  Parámetros de salida: booleano con el resultado del proceso.

- **bool mousePressed(constOIS::MouseEvent &arg);**

  Recoge la actividad producida cuando el ratón es presionado.

  Parámetros de entrada: información sobre el evento que se produce.

  Parámetros de salida: booleano con el resultado del proceso.

- **bool mouseReleased(constOIS::MouseEvent &arg);**

  Libera la información recogida por el ratón.

  Parámetros de entrada: información sobre el evento que se produce.

  Parámetros de salida: booleano con el resultado del proceso.

- **void setupViewport(Ogre::SceneManager *curr);**

  Crea un punto de vista y establece un color de fondo, ajustando la pantalla a un ancho y altura.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **voiddualViewport(Ogre::SceneManager*primarySceneMgr, Ogre::SceneManager *secondarySceneMgr);**

  Crea lo mismo que la función anterior, pero a diferencia esta crea dos fondos uno por cada escena, y se emplea cuando se está representando de forma simultánea ambas escenas.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **Static voidswap(Ogre::SceneManager *&first, Ogre::SceneManager *&second);**

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **void chooseSceneManager(void);**

  Crea la escena que se va a mostrar.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **void createCamera(void);**

  Crea una cámara para cada escena en una posición.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **Void GraphicsManager::createViewports();**

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

3. TTTGame.

- **TTTGame();**

  Llama a la función Initialize.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **~TTTGame();**

  Cierra los sockets creados.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **Instance();**

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- **Void Initialize();**

Inicializa el tablero y las posiciones.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **Bool Network();**

Declara el socket destino y se realiza la conexión.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **Void ClientGo(char*position/void *dato);**

Realiza la función de recibir y enviar información por parte del cliente. Dependiendo si se realiza en la parte del cliente o el servidor se le pasa un parámetro distinto.

Parámetros de entrada: En caso de tratarse del cliente se pasa el parámetro position, y en caso de tratarse del servidor se usa el parámetro dato.

Parámetros de salida: ninguno.

- **Void ServerGo(char*position/void *dato);**

Realiza la función de recibir y enviar información por parte del servidor. Dependiendo si se realiza en la parte del cliente o el servidor se le pasa un parámetro distinto.

Parámetros de entrada: En caso de tratarse del cliente se pasa el parámetro position, y en caso de tratarse del servidor se usa el parámetro dato.

Parámetros de salida: ninguno.

- **USER GetWinState(intcounters);**

Se declaran las combinaciones ganadoras y sí se cumple alguna de ellas se adjudica un ganador.

Parámetros de entrada: El contador de veces que se ha posicionado una ficha.

Parámetros de salida: Devuelve a un ganador.

4.B3D_VideoTexture.

- **B3D_VideoTexture();**

Inicializa las variables que sean necesarias.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **~B3D_VideoTexture();**

Elimina las variables.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **Void send(); (Sólo está en el Servidor)**

Se crea el socket y se envía con la ayuda de las funciones de OpenCV las imágenes recogidas por la cámara.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- **Void receive(void);(Sólo está en el Cliente)**

Se crea el socket para recibir las imágenes procedentes del servidor.

Parámetros de entrada: ninguno.

Parámetros de salida: frame recibido.

- **Void Update();(Sólo está en el Cliente)**

Actualiza los frames recibidos.

Parámetros de entrada: frame recibido.

Parámetros de salida: ninguno.

5. DotSceneLoader.

En este fichero se implementan los métodos que serán los encargados de cargar los ficheros de la escena. Convierten las mallas de los elementos al formato XML que mediante la herramienta OgreXMLConverter se pasan a binario optimizado.

6. Wrappers.

Se crean los descriptores para los sockets.