DESARROLLO DE LINEAMIENTOS PARA LA CREACIÓN DE VIDEOJUEGOS APLICANDO EL LENGUAJE C++ UTILIZANDO UN MOTOR DE VIDEOJUEGOS LLAMADO IRRLICHT ENGINE.

Realizado por:
RODRIGUEZ CORRALES JAVIER RAFAEL

Asignatura:
INVESTIGACIÓN FORMATIVA IV

Docente:
FREDDY BRICEÑO

Tutor:
JAVIER HENRÍQUEZ

Curso:
10 ° B

CORPORACIÓN EDUCATIVA MAYOR DEL DESARROLLO SIMÓN BOLÍVAR
INGENIERÍA DE SISTEMAS
BARRANQUILLA
2006-2

Introducción

En el mundo siempre hay algo nuevo que conocer, las cosas llegan cuando menos se esperan, la imaginación es un don que nos han otorgado, siempre esta golpeando por la espalda y exige conocer mas, en ocasiones cosas imposibles de encontrar, pero que con la fuerza de voluntad se puede realizar, aunque sea de forma artificial.

El videojuego, es un medio virtual que nos permite disfrutar de nuestras más grandes fantasías.

Muchas personas tienen sueños, pero no encuentran la manera de materializarlo, por este motivo la intención de este proyecto, es brindarles una ayuda básica que les permita mostrar toda esa creatividad en un videojuego dándole las bases necesarias para su desarrollo.

Existen muchas herramientas gratuitas en Internet desconocidas para las personas, estas le pueden ser útiles en su deseo de crear videojuegos, ahora bien conoceremos los problemas más usuales en los videojuegos creados, para no cometer los mismos errores y así obtener lo que se desea de una forma comprensible y sin complicaciones.

1. PLANTEAMIENTO DEL PROBLEMA

1.1. Descripción del problema

Los avances de la tecnología han conseguido crear animaciones en 3D bastantes sofisticadas para prevenir desastres a nivel ambiental y otros semejantes a través de una simulación, como ver una película, y encontrar cómo enfrentar dicha adversidad.

Gracias a la animación 3D también ha evolucionado la industria del entretenimiento digital, los videojuegos, un programa de entretenimiento digital, que abarca desde jóvenes hasta adultos sin importar el sexo, ya que existen de todo tipo como lucha, deporte, estrategia, aventura, enigma, entre otros.

Anteriormente los videojuegos eran desarrollados en dos dimensiones, estos estaban compuestos por dibujos planos y manejando solo dos coordenadas 'X' y 'Y'.
Con el nacimiento de la animación 3D los videojuegos dieron un gran paso porque además de poseer tres coordenadas, facilitaba el desarrollo de animaciones más realistas e interesantes. Claro esta que la innovación requirió de la unión de un gran número de personas, puesto que la animación 2D no requería de muchas personas para su desarrollo, dando resultados impresionantes.

Los videojuegos son muy interesantes y le ofrecen al usuario un momento de diversión muy amplio, tanto que han ido superando otros tipos de entretenimiento como el cine, las películas de video y la música grabada.

En la actualidad existen muchas empresas desarrolladoras de este software animado, la mayor parte es de Oriente, en occidente también existen pero son escasas, muchas personas en Colombia y en otras partes de Occidente tienen un gran interés en esta materia, los videojuegos, algunos hasta han pensado el desarrollar uno propio, pero esto

requiere una gran investigación, conocimiento, recursos, tiempo y personal capacitado de la cual muchos no disponen.

Algunos han optado por conseguir información en Internet, descargando ayudas y programas útiles, pero la mayor parte de estos se encuentran muy complejos y requieren asesoría externa.

Existen programas para crear videojuegos sin programación, pero con opciones muy limitadas, la mayor parte con derechos de autor, una mala elección si se desea distribuir el producto.

También existen cursos en Internet pero la mayor parte requiere un costo por estancia, de la cual muchos no disponen, además de no garantizar un buen aprendizaje.

En Colombia existen 5 o 6 instituciones que facilitan estos cursos, pero son demasiado costosos, además de estar en otras ciudades y entre otros, lo que cual se le imposibilita a la mayoría de las personas.

¿Existe algún tipo de lineamiento base gratuita que posea la información necesaria, consejos, aclaraciones, ejemplos y demás utilidades que sirvan de soporte para entrar al mundo del desarrollo de videojuegos?

1.2. Formulación del problema

- ¿Qué se necesita a la hora de desarrollar un videojuego?

1.3. Sistematización del problema

- ¿Qué es un videojuego?
- ¿Cómo han evolucionado los videojuegos?
- ¿Qué es un Motor de videojuegos?
- ¿Qué motores existen en Internet para el desarrollo de videojuegos?
- ¿Por qué el motor de videojuegos "Irrlicht Engine"?
- ¿Qué se necesita para utilizar el motor "Irrlicht Engine"?
- ¿Cómo se instala "Irrlicht Engine"?
- ¿Cómo se trabaja con "Irrlicht Engine"?
- ¿Qué se debe tener en cuenta para Desarrollar un videojuego?
- ¿Qué reglas hay que respetar para distribuir videojuegos con "Irrlicht Engine"?

2. OBJETIVOS

2.1. Objetivo General

- Desarrollar unos lineamientos con la información necesaria para entrar al desarrollo de videojuegos bajo el lenguaje C++ y el motor "Irrlicht Engine".

2.2. Objetivos Específicos

- Obtener una definición clara de videojuego
- Realizar una investigación de los avances que han tenido los videojuegos
- Conocer la función de un motor de videojuegos
- Recopilar información de los motores disponibles en la Web
- Conocer el motor "Irrlicht Engine"
- Enseñar el modo apropiado de utilizar el motor "Irrlicht Engine"
- Explicar los pasos para instalar el motor "Irrlicht Engine"
- Realizar ejemplos prácticos que ayuden al lector a trabajar con el motor "Irrlicht Engine"
- Recopilar las bases necesarias para desarrollar videojuegos
- Conocer las reglas a seguir al distribuir un videojuego

3. JUSTIFICACIÓN

En la formación profesional de los estudiantes es necesario poner en práctica los conocimientos adquiridos, para ver que tanto se han asimilado con respecto a la carrera que seleccionaron, siendo importante las actividades que se realicen durante el proceso de aprendizaje.

Aplicarlos de forma innovadora ayuda a desarrollar la mente para poder enfrentar en un futuro problemas mas complejos que requieran soluciones rápidas, ya que en el mundo empresarial, los negocios requieren constantemente programas que permitan a sus empresas trabajar mas eficiente.

Obviamente un problema como este requiere un conocimiento y creatividad bastante amplio, que ayuden a proporcionarles lo que necesitan los mas rápido posible, esto se puede conseguir desarrollando métodos útiles en las practicas que realicemos, que podamos emplear en nuestra vida profesional.

Es fundamental que en la educación profesional se puedan enfrentar a cualquier problema sin importar la dificultad, esto hará madurar el conocimiento como profesionales y dará más confianza hacia trabajos futuros.

Es así que el Proyecto "**DESARROLLO DE LINEAMIENTOS PARA LA CREACIÓN DE VIDEOJUEGOS APLICANDO EL LENGUAJE C++ UTILIZANDO UN MOTOR DE VIDEOJUEGOS LLAMADO IRRLICHT ENGINE**", tiene como objetivo aportar una visión global de las experiencias que se enfrentaran en una futura vida como profesionales.

Es importante que los estudiantes apliquen sus conocimientos en proyectos formativos para saber los límites que no han superado y que tanto aprendizaje han adquirido para compensarlo y poder aprovecharlo.

Creando aplicaciones con herramientas que les aporta la educación profesional es una forma de hacerlo, pero hay que aclarar que no es prudente conformarse solo con lo que brinde una universidad o institución, ya que existen diversa cantidad de herramientas para el desarrollo de software, como es el diseño grafico, que es una herramienta para el desarrollo de animación, esta herramienta es llamativa ya que son escritas con herramientas que funcionan a base de línea de código (como por ejemplo **"visual estudio"**) en el sentido de que también trabaja con código para el desarrollo de sus aplicaciones.

Este proyecto tiene como fin aportar una ayuda a aquellos que deseen crear un videojuego y no tengan la información necesaria para su desarrollo, dándole las bases necesarias para crearlo, utilizando una herramienta GNU llamada **"IRRLICHT ENGINE"** una aplicación gratuita que puede conseguirse en Internet, este programa es muy eficaz ya que nos facilita las librerías necesarias para crear animación o en nuestro caso videojuegos.

Esta herramienta es la que se va a exponer en el lineamiento a desarrollar para la meta de construir un videojuego, mas adelante haremos una comparación de este motor con respecto a las otros motores disponibles en el mercado y en Internet, ya que posee gran cantidad de prestaciones, emplea el paradigma orientado a objetos se apoya en otras librerías LGPL para aumentar sus prestaciones, esta orientado a eventos, puede renderizar tanto por Hardware como por Software, una gran cantidad de proyectos la están utilizando en Internet, tiene buena documentación.

4. MARCO DE REFERENCIA

4.1 Marco teórico

El videojuego es un medio de entretenimiento que se basa en un soporte electrónico, como un computador, una consola, una maquina recreativa etc.

HISTORIA DE LOS VIDEO JUEGOS[1]

El primer videojuego fue inventado en el año 1958, su autor BILL NIGHINBOTTHAM, es por lo tanto el primer programador de videojuegos de la historia. Pero para él no fue tan importante crear su juego, en el cual no invirtió más de una semana, era una representación de un juego de tenis muy pobre que presentó a una feria científica de su ciudad, lo llamó "Tennis for two" y aunque en la feria el invento despertó mucho interés él no consideró que le llevase a ninguna parte y por lo tanto no se molestó en registrarlo. Aprovechando el error de NIGHINBOTTHAM en 1972, NOLAN BUSHNELL fundó una compañía llamada Atari y publicó el juego con el nombre de "Pong".

Pero aunque "Pong" fue el primer juego, no fue el primero en ver la luz. El primer juego comercial salió para arcade (Maquina traga moneda) un año antes y fue creado por el mismo NOLAN BUSHNELL antes de fundar Atari. Era un juego para 1 o 2 jugadores y se llamaba "Computer Space".

La evolución se empezaba a notar 4 años después, en 1975, con la salida del primer juego en color, Indi800, que permitía que jugasen simultáneamente 8 jugadores. Era un juego perteneciente también a Atari.

Un año después STEVE JOBS y STEVE WOZNIAK, programadores ambos de Atari, desarrollaron un juego que al igual que Pong ha pasado a la historia como uno de los grandes clásicos. Lo que no sabe casi nadie es que este juego era solamente una versión

[1] www.rinconsolero.com/rinconsolero.v2/historia_de_los_videojuegos.htm

para un jugador (del muy nombrado Pong), el famoso juego de los ladrillos. En 1978 los dos STEVES fundaron "Apple Computer" independientes de Atari y por lo tanto, gracias a Pong, ahora la amigable manzana de arco iris (logotipo de Apple) ha conseguido ser la marca más conocida en el universo de la computación.

En 1975, asociados con Mitsubisi, Nintendo empezó a crear lo más parecido a un videojuego. Por fin, 3 años después sacan sus primeras consolas: TV Game 15 y TV Game 16 con juegos típicos de tennis, coches...etc.

En 1980 se empiezan a distribuir las Game y Watch, recientemente piezas de coleccionista) las primeras consolas portátiles del mundo, precursoras de Game Boy.

Ese mismo año SHIGERU MIYAMOTO, un joven programador de Nintendo creó para arcade el hoy también súper clásico Donkey Kong. El protagonista era un monigote llamado en principio Jumpman que tenía que ir saltando barriles hasta llegar a su amada Princesa, secuestrada por un mono gigante. A raíz del tremendo éxito cosechado por el juego, Jumpman cambió su nombre por Mario, y... cosas de la vida, el fontanero Mario es uno de los personajes más famosos en el mundo de los videojuegos y la mascota de Nintendo desde entonces.

Las primeras consolas:

La primera consola que vio la luz del día fue la Magnazos Odyssey que salió a la venta en el año 1972. Todos sus componentes eran analógicos y sus juegos eran demasiado primitivos. La consola incluía cubiertas de plástico para la pantalla de televisión con la intención de hacer los juegos más interesantes. La consola era tan primitiva que los jugadores tenían que llevar la puntuación en papel, ya que juego no tenía memoria.

Poco después, en 1975 tras la salida en arcade de "Pong", Atari creó su primera consola: Atari "Pong", sólo permitía jugar al "Pong" pero ayudó en gran medida al florecimiento de los videojuegos ya que era como llevarte una máquina recreativa a casa. Esto causó un gran entusiasmo, pero no llegó a cuajar ya que al disponer de un único juego sin

posibilidad de alternarlo no salía muy rentable, por lo que se prefería seguir con las recreativas.

Por fin, dos años después, en 1977 Atari lanzó la consola que la llevo al éxito: **Atari 2600** Entre sus novedades se encontraba la innovación de poder cambiar de juego por medio del intercambio de cartuchos. Sólo tenía 8 bits de potencia, pero eran más que suficientes para aquella época y para combatir la competencia. Se mantuvo al pie del cañón liderando a sus rivales durante muchos años gracias a su amplio catálogo de juegos. Unos años después de su salida sacó una nueva versión beta que remató la jugada. Con su éxito, Atari se permitió comprar las licencias de películas tales como ET o Indiana Jones, lo que le aseguró el puesto. Se mantuvo en el mercado hasta el año 1990 por lo que se ha convertido en la consola hasta el momento, que más tiempo ha permanecido activa en el mercado. Destronada por la Nintendo.

Tan sólo un año después de la salida de Atari 2600 la segunda generación de consolas Odyssey se dispuso a combatir cara a cara con la ya citada Atari. Fue su rival más fuerte, pero la Magnavox-Odyssey2 nunca alcanzó la popularidad de la 2600. La gran novedad de esta consola, única en la generación de 8 bits, era incluir un teclado (emulando un ordenador). Entre las competidoras de Atari fue la más vendida, por debajo de la "Bally Astrocade" y la "Fairchild Channel F", la cual utilizaba la misma tecnología básica de la 2600.

En 1980 salió al mercado la mejor consola de la primera generación: "Mattel Intellivision", con un motor gráfico de 16 bits. Compitió con "Atari 2600" sin mucho éxito a pesar de tener unos gráficos muy superiores. También tenía un módulo sintetizador de Voz que incluía separadamente y fue la primera en sustituir los viejos joystick por mandos con botones, modelo a seguir en el resto de consolas de aquí en adelante.

La década de los clásicos (1978-1988):

La década 1978-1988 fue sin lugar a dudas la época de salida de juegos que más conversiones han tenido y de los que más se han extraído ideas y argumentos. Se puede decir que fue la edad de oro de los videojuegos:

En 1978 apareció "Space Invaders", un hito sin duda. Creado por Taito, este juego supuso una revolución en la industria de los videojuegos ya que por primera vez en la historia se podían guardar las puntuaciones más altas.

Un año más tarde, Atari desarrolló "Lunar Lander", un juego que en principio no pretendía serlo. Fue programado a modo de información sobre la distancia y velocidad que podría tener una nave espacial en la superficie lunar, según la gravedad. Pero de nuevo Atari aprovechando esta idea, le añadió gráficos y sonido y consiguió crear el primer simulador de vuelo de la historia.

También en 1979, Atari lanzó otro clásico que también obtuvo una gran aceptación: Asteroids, gracias al cual se admitieron los videojuegos como una forma de diversión capaz de quitar el estrés y la tensión acumulada del trabajo, atribuyéndolos así, por primera vez a los adultos, por la revista Newsweek. El videojuego era diferente a todo lo visto anteriormente.

Asteroids además, sí era capaz de almacenar junto a los puntajes máximos las iniciales del jugador.

Este año dio salida a muchos otros clásicos tales como "Galaxian" de Namco o "Monaco GP" de Sega.

En 1980 Atari seguía triunfando, esta vez con Battlezone, el primer simulador de tanques y también el primer juego de aspecto tridimensional. Causó tanta impresión que el ejército norteamericano encargó a Atari una nueva versión mejorada para que los soldados del ejército practicaran a modo de entrenamiento sin necesidad de correr riesgos y que impidiera tener que utilizar maquinaria pesada.

En este año también apareció Defender, de la mano de William. Fue el juego que más controles tuvo: "nada menos" que una palanca y 5 botones.

Otro de los "grandes" que apareció ese año fue Missile Command, de Atari, el típico juego de disparar a los misiles.

Pero sobre cualquier juego, el clásico del año no fue un juego de Atari, sino de Namco, un juego que ha pasado ha la historia como uno de los más divertidos y originales de todos los tiempos, el comecocos, Pacman: Este juego, conocido por todos, tuvo un éxito inigualable hasta entonces y produjo tal impacto que se le dedicó una portada en la revista Time Magazine, una serie de dibujos animados y una canción que causó furor.

Esa "bola" amarilla capaz de tragar cientos de "pelotas" evitando a su vez ser comido por unos fantasmas que se volvían débiles cuando el comecocos se tragaba un coco de casi su tamaño fue una fórmula que muchos videojuegos posteriores han cogido indirectamente como argumento principal.

Llegó el 1981, acompañado de un montón de súper clásicos tales como, Donkey Kong, del que ya se ha comentado antes, el primer juego en el que aparecía Mario. He de decir que Nintendo tuvo algunos problemas con "Universal Studios", ya que afirmaban que Donkey Kong tenía un parecido más que razonable con su King Kong, por lo que demandaron a Nintendo. Sin más problemas Nintendo ganó el juicio.

Tras aquel bombazo, Atari no se quedó atrás y lanzó Tempest, el primer juego con gráficos vectoriales a color y Centípede, el juego en el que había que dispararle a un ciempiés y el primero programado por una mujer, Dona Balley.

En este año aparecieron juegos que nunca se quedarán viejos como "Frogger" de Konami o "Galaga" de Namco, enormes clásicos, con cientos de "remakes" posteriores.

Además salió a la venta la segunda parte de Pacman, obra también de Namco: Miss Pacman, que aún siendo idéntica a la primera, a excepción del protagonista (que ahora era femenino), cosechó un éxito casi igual al de su predecesor. Atari empezaba a preocuparse por la competencia.

A diferencia del año anterior, 1982 no fue un año de "clásicos inolvidables", bueno, sí que lo fue, pero nunca lo podría haber sido de no ser por los años anteriores, por los que se puede decir que es el año de las segundas partes.

Al mercado este año aparecieron juegos en los que destacan: Súper Pacman, Millipede (la continuación de Centípede), Donkey Kong Jr, Burguer Time, Dig Dug, Moon Patrol, Popeye, Star Trek, Mr.Do Time Pilot, Q´Bert y el "clásico original" de la época,

propiamente dicho: Zaxxon, de Sega que presentaba la revolucionaria novedad de incluir un jefe al final de la fase.

1983 fue el año del cine en los videojuegos: por primera vez se incluía el argumento de una película en un videojuego con Tron (basado en la película de Disney del mismo nombre), y a raíz de él empezaron a salir más juegos con argumentación cinematográfica como Star Wars de Atari o la segunda parte de Tron.

El 1983 también trajo consigo a la primera parte de Dragon´s Lair, ese juego en el que se jugaba "dentro" de una película en la que se podía interactuar, creado por Cinematonics. Utilizaba por primera vez la tecnología "Laser Disc".

Algo después apareció M.A.C.H. 3, el cual también usaba esa tecnología y era el primero en usar para un juego escenas de video reales: el fondo era una secuencia de video real, mientras que en el centro de la pantalla aparecían aviones y helicópteros que eran los verdaderos protagonistas, creados por ordenador.

Además en este año, Nintendo sacó el Donkey Kong 3, un juego que cambiaba completamente de estilo con respecto a sus predecesores y que por ello, no llegó a triunfar lo que se esperaba. También lanzó el Mario Bros, en donde por fin, el protagonista del Donkey Kong, Mario y su hermano Luigi (este aparecía por primera vez) tenían la oportunidad de disfrutar de su propio juego en el que su tarea era limpiar las tuberías de todo tipo de bichos raros. A raíz del éxito que contrajo, Mario se convirtió oficialmente en la mascota de Nintendo, que precisamente, este mismo año sacó su primera consola independiente, ya que antes se había asociado con otras compañías, eso sí, sólo para Japón: la Famicom (Family Computer, conocida en Epaña como Nintendo) que no llegaría al resto del mundo hasta 1985.

Con el comienzo de 1984 apareció una nueva compañía: "Capsule Company" (más conocida con el nombre de Capcom) que traería mucho de que hablar en el futuro.

El 1984 no fue un año muy agraciado para la industria "videojueguil" a pesar de la salida de clásicos como: 1942, Bomb Jack, Star Force, Pac-Land (donde Pacman debutó en los juegos de plataformas), Lode Runner (gran clásico), Kung Fu Master y Paperboy.

El mercado de los videojuegos durante este año sufrió una grave crisis, tanto en el sector de los ordenadores, como en el de las consolas recreativas (arcade) o domésticas. Tanto fue así que hizo que muchas compañías se hundieran en un pozo sin fondo, entre las cuales destacan: Universal, los creadores de Mr. Do!, que en el momento de su cierre tenían en proyecto la quinta parte de su saga dispuesta a aparecer en formato Laser Disc y la sucursal norteamericana de Sega, la cual se retiró a Japón y no volvió a América hasta 2 años después fundando Sega of América, aún en activo.

El año 1985 hizo que los videojuegos comenzaran a recuperarse. Nintendo lanzó por fin su Famicom en América tras el tremendo éxito obtenido en Japón con su consola y con el recién estrenado Súper Mario Bros, un juego encargado por Yamauchi (el director de Nintendo) a Miyamoto, que hasta entonces programaba para Nintendo y Atari conjuntamente, que rompió moldes con todo lo visto anteriormente. Era un juego largo, lleno de colorido... ¡Vamos! ¿Quién no conoce el primer Super Mario?

La llegada de la Famicom a E.E.U.U se quería hacer a lo grande, por eso se le cambió el nombre por Nintendo Entertainment System y también el modelo por otro más serio a modo de "video doméstico". Nintendo con aquello lo que quería dejar claro es que su consola no era un simple "juguete" para niños, sino un electrodoméstico más que debía estar presente en todos los hogares. Tenía un motor gráfico de 8 bits muy superior a la Atari 2600, reina de las consolas desde su salida, y de hecho, la Nes fue la única consola capaz de destronar a la "reina" vendiendo más que cualquier sistema de videojuegos hasta entonces, y sólo comparable en la actualidad con Playstation. Marcó la segunda era de los videojuegos y el comienzo de la hegemonía de Nintendo en este mercado.

Los juegos más importantes de este año fueron: Pitfall 2 de Sega, Indiana Jones y the Temple of Doom y Epire Strikes Back de Atari, Commando y Ghosts'N Goblins de Capcom y Dig Dug 2 de Namco.

Por último merece que sea recordado uno de los primeros juegos de fútbol que apareció este año, Tehkan World Cup para arcade.

Y como quien no quiere la cosa llegó 1986 dispuesto a mostrar cantidad de videojuegos sobresalientes en calidad. Este año fueron lanzados una barbaridad de juegos que hoy

son leyenda, como Out Run, primer juego en utilizar la técnica del Scaling, mediante la cual los gráficos se hacen más grandes a medida que se acercan a la pantalla, proporcionando así más sensación de velocidad, a pesar de que debido a ello la calidad disminuya al hacerse los píxeles más grandes, destacó por su música y voces renderizadas y Wonderboy, por parte de Sega, Arkanoid y Bubble Bobble (los dos de Taito), y por último Rygar de Tecmo (antes Tehkan).

El 1987 fue el año de los arcade: Destacan por la importancia y la calidad de sus lanzamientos tres compañías: Sega, Capcom y Taito. Entre los juegos de Sega destacan: After Burner, Alien Syndrome, Shinobi, Heavyweight Champ, SDI -Strategic Defense Initiative y Wonder Boy In Monster Land. Entre los juegos de Capcom, solamente dos son los más importantes: 1943 Battle Of Midway y Street Fighter. 1943 Battle Of Midway es la continuación de 1942, un juego lanzado por Capcom en 1984. En cuanto a Street Fighter, las leyendas cuentan que está basado en un manga (comic japones) de nombre Kung Fu Tao.

1988 trajo con sigo algunos clásicos como Ghouls and Ghosts (segunda parte de Ghosts and Goblins), Ninja Gaiden, Super Contra de Konami... pero sobre todo, el clásico de este año es quizás el clásico más clásico y el videojuego más importante de la historia ¿sabes cuál?:

Tetris, obra de Atari, ese juego de puzzles que consiste en encajar piezas de diferentes formas. Ese y no otro, Tetris es hasta el momento, el juego más vendido y divulgado en toda la historia de los videojuegos. Por su simplicidad, y complejidad al mismo tiempo, este juego ha enganchado a todo el mundo a una consola. Apareció en casi todas las consolas e incluso hoy se siguen haciendo nuevas versiones en 3D, pero donde más éxito iba a tener era en Game Boy, que faltaba un año aún para su salida.

Si se piensa en el éxito de Tetris, se puede deducir que sus creadores, los Rusos Alexei Pazhitnov y su compañero Gerasimov se hicieron multimillonarios, pero la verdad es que el juego fue creado en una mala época: Para entonces, todavía existía la Unión Soviética y obviamente el gobierno comunista. El Estado se quedó con los derechos del juego, y los creadores de este clásico no recibieron ni un centavo. No fue hasta mediados de la década de los años 90, una vez caído el antiguo régimen, cuando el gobierno ruso reconoció los derechos de Pazhitnov y Gerasimov.

Esta década fue sin lugar a dudas, el punto de inspiración para muchos programadores futuros, y aunque aún quedaban muchas cosas por llegar, como el paso a las 3D, nada ha sido tan importante como esta época, la cual mucha gente ya ha olvidado.

CRISIS DE LOS VIDEO JUEGOS[2]

1ra Crisis

La primera crisis del videojuego ocurrió a principios de los años 1980, afectó principalmente al sector de empresas derivadas de los juegos tipo arcade, que se podría decir eran la mayoría de las empresas importantes en aquella época. Las causas fue la proliferación de sistemas domésticos (consolas de videojuegos) y microordenadores, y los clientes potenciales en vez de dejar el dinero en las huchas de las maquinas recreativas, optaban por adquirir los productos originales y piratas anteriormente mencionados.

Las primera acción paliativa de las empresas del sector fue aprovechar el mercado en expansión del ocio domestico, creando sus propias consolas o desarrollando para las ya existentes.

La segunda acción para salir del bache fue mejorar las maquinas recreativas añadiéndole un nuevo nivel a la interacción del usuario con el videojuego, el pionero fue el videojuego Áng. On de SEGA, que era un juego de carreras de motos con la novedad en aquel entonces de en vez de jugarlo con un joystick se montaba el jugador en una moto fija a la máquina recreativa pero que se podía tumbar para coger una *curva ficticia* y este movimiento era recogido por unos sensores que lo representaban en el videojuego como un giro, además de acelerar con un manillar idéntico al de las motos reales.

Así se salio de la primera crisis del videojuego y dio una nueva realidad a mundo de las maquinas recreativas extrapolándose la idea original, incluso coches con capacidad de girar y simular resistencia del volante frente a determinados terrenos.

2da Crisis

La segunda crisis es mas difusa situarla cronológicamente porque se dice que apareció con los videojuegos en los hogares, y no es otra que la piratería, realmente ha tenido dos

[2] www.delcorp.org/video_juegos/crisis.htm

índices altos en el pasado y en el futuro se supone que vendrán más porque ningún sistema informático actual y futuro es 100% seguro.

La primera situación de piratería ocurrió con el auge de los microordenadores de 8 bits que usaban como sistema de almacenamiento de datos las cintas casette, que se podían copiar sin mayor problemas con una doble pletina de media o alta gama, las soluciones que se dieron fue añadir modulaciones mas complejas de sonido, pedir una clave al inicio del videojuego que estaba en el manual del videojuego organizada en columnas, con lo que pedía la clave por un sistema de coordenadas o como el juego de los barquitos en una pagina de colores oscuros para que fuera infotocopiable y otros métodos mas originales.

La segunda situación de piratería masiva ocurrió al cabo de unos meses de la aparición de la consola PSX (Play Station X) de Sony, pero se sigue dudando si realmente fue una crisis para el sector del videojuego o para el sector del videojuego que no fuera Sony, porque las ventas de esta consola gracias a su *problema* aumentaron como la espuma, pese a ser tecnológicamente inferior a sus coetáneas (N64, Saturn). Se llegó a situaciones tales como que bazares y tiendas de electrodomésticos te ponían el chip para que se pudiera usar juegos piratas, situación ilegal en España (antes y actualmente 2005) pero ilegal en otros países, pero no quedaba solo en esa acción sino que algunos de estos establecimientos vendían copias piratas de juegos para esta consola, algo que es ilegal.

Y en el mundo de los PCs siempre ha habido piratería, simplemente se frenaba algo con la aparición de un nuevo formato de almacenamiento como la aparición del CD-ROM que duró hasta que el usuario pudo acceder a grabadoras CDs a precios asequibles, con el DVD el margen de tiempo para la calma ha durado menos aún.

3ra Crisis

La tercera crisis además de ser difusa en el tiempo porque algunos la sitúan con la aparición del PC, otros entendidos incluso antes, aunque la prensa del sector la sitúa con la aparición de la consola PSX (Play Station X), además de esto es un tanto subjetiva si se admira de manera global. La 3° crisis es la falta de originalidad y el mercantilismo, que esta haciendo que salvo raras ocasiones como empresas con solera, el mundo del

software libre y algún jefe de desarrollo que desarrolla videojuegos como si de cine de autor se tratara. Y refiriéndonos al cine es la misma crisis del cine que se *fabrican* películas, y no se desarrollan o se crean obras artísticas. Indicios claros se puede ver en ambos sectores en la proliferación de los remakes de clásicos, la extensión excesiva y carente de argumento de algunas de sus sagas famosas (ejemplo la saga Tomb Raider, la saga de juegos de fútbol FIFA) las cuales no aportan nada nuevo al anterior título, además de si un tipo de producto triunfa (ejemplo GTA 3) las demás compañías del sector copian la fórmula del triunfo sin añadir nada nuevo. Por tanto esta crisis al sólo afectar al lado de los usuarios/jugadores y no afectar al sector empresarial del videojuego ya que incluso obtienen más beneficios que arriesgándose a nuevos productos, es una crisis no declarada oficialmente.

La industria del Videojuego[3]

Según **la asociación Española de Distribuidores y Editores de Software de Entretenimiento (aDeSe)**, el mercado de videojuegos facturó 800 millones de euros en España en el año 2003; superando a otros tipos de entretenimiento como el cine, las películas de video o la música grabada. Por tipos, los más vendidos son los de estrategia, seguidos de aventura y acción (en PC) y de acción, aventura, carreras y deportes (en Consola).

aDeSe, es la patronal del sector de los videojuegos en España, desde 1997. Está integrada por trece compañías, que representan más de un 80% del mercado español: Acclaim, Atari, Electronic Arts, FX Interactive, Microsoft, Nokia, Planeta De Agostini Interactive, Proein, Sony, Take Two Interactive, Ubi Soft, Virgin Play y Vivendi Universal Publishing.

Los juegos más vendidos

La lista de los videojuegos más vendidos incluye normalmente nuevos lanzamientos, reediciones o recopilaciones. Hay actualmente en el mercado varios títulos a un precio

[3] http://www.3dyanimacion.com/foros/index.cfm

muy económico, bien porque ha quebrado la empresa que lo distribuía y se liquidan, bien porque son juegos muy antiguos y se lanza una reedición, o simplemente porque se sigue la estrategia de cosecha.

Estrategia de cosecha

La estratregia de cosecha se basa en lanzar un título a un precio suficientemente alto como para cubrir los enormes costes de publicidad, pero lo suficientemente bajo como para no ser descartado por la competencia o el mercado. Cuando ha alcanzado el disco de platino (80.000 copias vendidas), se reduce el coste a la mitad, teniendo en cuenta que es a partir de ese momento cuando se producen los verdaderos beneficios y, al haberse rentabilizado la inversión, se puede reducir el precio. Entre estos juegos se encuentran los de la compañía FX Interactive, una de las mejores empresas españolas de distribución de videojuegos, heredera de la experiencia de la extinta Dinamic.

La línea económica Xplosiv, de Planeta DeAgostini Interactive y Empire Interactive, ha superado las 250.000 unidades vendidas en España desde su lanzamiento en el año 2001, consolidándose como el primer distribuidor de series budget y el cuarto distribuidor de productos interactivos.

Respecto a la clasificación de videojuegos por edades y temáticas, el nuevo sistema Pan European Game Indicator (PEGI), propulsado por la Interactive Software Federation of Europe (ISFE) sustituirá en toda la Comunidad Europea (salvo Alemania) al etiquetado actual del Instituto Nacional de Consumo según los códigos de la Asociación Española de Software de Entretenimiento (aDeSe). En Estados Unidos se suele utilizar el etiquetado ESRB para categorizar los juegos.

El desarrollo de Videojuego

El desarrollo de videojuegos ha cambiado mucho desde que una única persona podía desarrollar un videjuego desde casa en un tiempo récord. Ahora es necesario un equipo multidisciplinar y años de desarrollo, además de una potente inversión financiera para sacar rentabilidad al proyecto. Sin embargo, existen varios kits de Desarrollo para quien

quiera programar videojuegos. Recientemente, la Universidad Pompeu i Fabra ha lanzado el Máster en Creación de Videojuegos.

Nebula Device es un motor gráfico 3D Open Source muy potente escrito en C++ que permite su pesonalización en Tcl/Tk, Lua, Python, Java y los lenguajes .NET; además de ser compatible con DirectX 9 y OpenGL. Funciona sobre Windows y se ha portado a Linux y Mac OS X.

Juegos Gratuitos

Existen muchos juegos gratuitos directamente descargables desde Internet, como Wolfestein: Enemy Territory, America's Army, Project Entropia, los RockStar Classics como Grand Theft Auto y Wild Metal, etc. Cada cierto tiempo, Microsoft y WUGNET (Windows Users Group Network) seleccionan una aplicación Shareware, revisada bajo estrictas condiciones que solamente sobrepasan el 2% de ellos; por ejemplo Ricochet. En Softonic se lanzan juegos con mucha frecuencia. Destaca el juego gratuito Cube, con un motor gráfico open-source excelente.

Abandonware

El término Abandonware se refiere a juegos antiguos que han perdido los derechos de autor o bien conversiones de fans de los programas originales. Por ello es posible descargar de forma gratuita juegos que en su momento fueron éxitos.

5. METODOLOGÍA

5.1. Tipo de estudio

El tipo de estudio es investigativo, se recopilara información para la elaboración del documento, se harán estadísticas para determinar prioridades y

Se hizo una encuesta con las siguientes preguntas dando la opción de una observación o un comentario que nos deseen agregar para enriquecer el proyecto:

La encuesta solo se realizo a un total de 52 personas.

a) ¿Te gustan los videojuegos?

- Respuestas positivas = 34
- Respuestas negativas = 18
- **Respuestas comentariadas = 15**

b) ¿Sabes como se hacen los videojuegos?

- Respuestas positivas = 14
- Respuestas negativas = 38
- **Respuestas comentariadas = 26**

c) ¿Conoces algún lugar de la ciudad donde den cursos de videojuegos?

- Respuestas positivas = 6
- Respuestas negativas = 46

- **Respuestas comentariadas = 2**

d) ¿Te gustaría crear tu propio videojuego?

- Respuestas positivas = 23
- Respuestas negativas = 29
- **Respuestas comentariadas = 19**

Respuestas comentariadas (a) 15:

- ❖ 7 escribieron que son muy divertidos
- ❖ 4 escribieron da plata cuando juegas fútbol en un local
- ❖ 4 escribieron es un buen pasatiempo.

Respuestas comentariadas (b) 26:

- ❖ 2 escribieron no me lo había preguntado
- ❖ 1 escribieron si lo vi en la TV
- ❖ 4 escribieron después que juegue lo demás no importa
- ❖ 1 escribieron conozco un programa llamado Entidad 3D para hacer juegos
- ❖ 5 escribió no se
- ❖ 9 escribieron me gustaría saber como se hacen
- ❖ 4 escribieron no he encontrado un libro que me explique como hacerlo la mayoría solo explican como hacer programas

Respuestas comentariadas (c) 2:

- 1 escribió aquí en la ciudad no se, pero se de Digipen en los Estados Unidos.

- 1 escribió en Internet hay un curso de videojuegos pero hay que pagar en una cuenta.

Respuestas comentariadas (d) 19:

- 1 escribió si, porque hay juegos muy malos y me gustaría hacerles uno para enseñarles como se hace.

- 1 escribió estuve investigando como se hacen, pero la mayor parte estaba en ingles y otros no los entendí.

- 5 escribieron si, por que es un negocio rentable.

- 3 escribieron si, pero no se como empezar, ni conozco las bases pero soy bueno programando.

- 4 escribieron si (Es un grupo) nosotros estamos trabajando con Blender para hacer personajes 3D para videojuegos, pero aun no hemos hecho un juego por que no somos muy buenos en programación, si vas a hacer un manual firme.

- 5 escribieron no me llama la atención, porque para esos hay otros que lo hacen.

5.2. Cronograma

CRONOGRAMA DE TRABAJO DE 2005

Mes	Febrero				Marzo				Abril				Mayo				Junio				Julio				Agosto				Septiembre				Octubre				Noviembre			
Actividades/semana	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
recopilacion de información																																								
Anteproyecto																																								
información de capitulos																																								
diseño de capitulos																																								
creacion de prototipo																																								

CRONOGRAMA DE TRABAJO DE 2006

Mes	Febrero				Marzo				Abril				Mayo				Junio				Julio				Agosto				Septiembre				Octubre				Noviembre			
Actividades/semana	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
investigacion de librerias																																								
codificacion en aplicación																																								
Generar videojuego																																								
Prueba en plataformas																																								
Jugar el juego																																								
Entrega de videojuego																																								

6. CAPÍTULOS

6.1 CAPÍTULO 1:

CONOCER ANTES DE CREAR

6.1.1. Planteamiento del capítulo

Este primer capítulo comienza exponiendo el concepto de consola, se definirán algunos conceptos para que en los próximos capítulos el lector no pierda el hilo, además de mencionar las características de las consolas de los videojuegos, sus innovaciones y su competencia.

6.1.2. Objetivos del capítulo

Es deseable que al finalizar el capítulo el estudiante:

.Conozca el concepto de consola.

.Sea capaz de comprender las mejoras en las consolas para mejorar sus prestaciones.

.Se familiarice mas con los "videojuegos"

6.1.3. Explicación del contenido

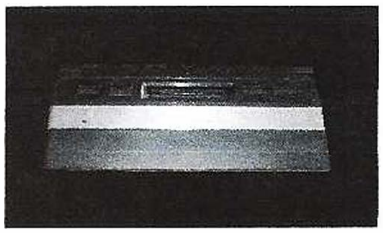
6.1.3.1 Introducción

En la era de la tecnología los videojuegos han abarcado un gran espacio en nuestras vidas, brindándonos mucha diversión, la mayoría de los videojuegos son reproducidos.

Una consola es en realidad un computador, diseñada única y exclusivamente a reproducir el videojuego de su capacidad o de la industria que la patrocina, porque la consola solo esta limitada a una tarea y es la de reproducir juegos, en cambio un computador puede hacer diversa cantidad de tareas y hacer mas con el hardware y software apropiado. La consola a de videojuegos ha ido evolucionando con los avances de la tecnología, existen ya algunos que son capaces reproducir música, como el "Play Station", dreamcast, y otros que se comentaran mas adelante).

6.1.3.2 Características de las consolas[4]

Atari 2600



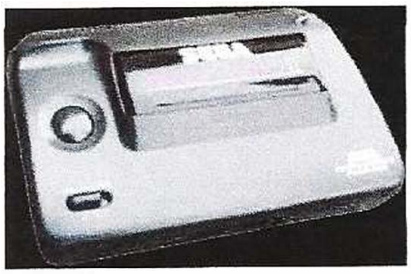
Procesador	Motorola 6507 a 1,19 MHz
Procesador Gráfico	1,79 Mhz
Memoria RAM	128 Bytes
Sonido	Controlado por el chip Stella, 2 canales.
Paleta de Colores	8 de una paleta de15.
Formato de juego	Cartucho
Puertos para mandos	2
Comentarios	Atari 2600 es una leyenda en el mundo de las consolas. Posee el título de consola con más años en el mercado y decenas de clásicos en su repertorio de juegos.

[4] www.rinconsolero.com/rinconsolero.v2/inf_tecnica.htm

Nintendo

Procesador	6508 Motorola (8 bits) a 1.79 Mhz
Procesador Gráfico	1,79 Mhz
Memoria RAM	2 Kb
Memoria de Vídeo	2 Kb
Canales de Sonido	2 squarewave, 1 triangle, 1 noise generator y 1 canal digital de audio
Paleta de Colores	24 de una paleta de 52.
Pantalla	256 x 240 píxeles.
Número de sprites	8 por línea, 64 en pantalla.
Formato de juego	Cartucho
Puertos para mandos	2
Comentarios	Nintendo (Nes) fue la única consola capaz de destronar a la veterana Atari 2600, hasta entonces reina de las consolas. Sus juegos Súper Mario Bros 3, y Donkey Kong contribuyeron en gran medida a conseguirlo. Permanece en segundo lugar como consola más vendida con 60 millones en todo el mundo

Master sistem



Procesador	Z80 de 8 bits 3,58 Mhz.
Memoria RAM	8 Kb
Memoria de Vídeo	16 Kb
Canales de Sonido	4
Paleta de Colores	52 de una paleta de 256
Pantalla	TFT 256 x 192 pixeles.
Número de sprites	16
Formato de juego	Cartucho
Puertos para mandos	2
Comentarios	La Master Sistem en sus intentos de hacer frente a la Nes todopoderosa de la época, no salió bien parada, además su hermana mayor "Megadrive" no tardaría en llegar.

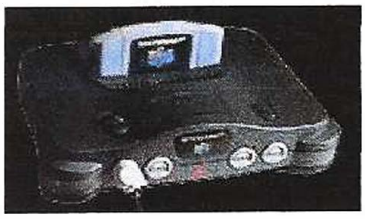
Megadrive

Procesador	Motorola 68000 a 7.67 Mhz
Resolución	Resolución: 320 x 224
Memoria RAM	64 Kb
Colores	64 en pantalla de 512
Memoria de Vídeo	64 Kb
Memoria de Sonido	8 Kb
Canales de Sonido	6 FM + 4 PSG
Puertos de Mandos	2
Formato de juego	Cartucho
Comentarios	Megadrive era un pedazo de máquina que nació para competir con la Nes, pero Nintendo no lo permitió y por ello fabricó la Super Nes, no más potente que la MD, pero sí a la altura. Y Nintendo salió victoriosa en la guerra.

Supernintendo (Znes)

Procesador	Custom 65C816 16 bit
Procesador Gráfico	Modo-7 para efectos 3D
Memoria RAM	128 Kb
Memoria de Vídeo	64 Kb
Memoria de Sonido	64 Kb
Canales de Sonido	8 PCM estéreo
Puertos de Mandos	2
Formato de juego	Cartucho
Extras	Compatibilidad con Game Boy mediante la Super Game Boy, que permitía jugar los juegos de GB en la T.V.
Comentarios	La Snes salió victoriosa en la batalla que entabló con Megadrive, a pesar de ser inferior técnicamente. Pero algunos juegos incluyeron packs de expansión que aumentaban la memoria y hacía que se igualase a su rival, o incluso la superara. Aunque nunca habría podido sacar un juego de la velocidad de Sonic.

Nintendo 64



Procesador	R4300I a 93 Mhz
Procesador Gráfico	Co-procesador a 62,5 Mhz
Memoria RAM	4,5 Mb (con posibilidad de expansión a 8 Mb)
Canales de Sonido	Sonido digital
Polígonos por segundo	160.000
Puertos de Mandos	4
Formato de juego	Cartucho
Extras	Posibilidad de aumento de memoria con el Expansion Pack y conectividad con GB por medio del Transfer Pack.
Comentarios	La pobre Nintendo 64 no fue muy bien acogida debido a que los juegos eran cartuchos y se criticaban de "infantiles". Tuvo joyas como Mario 64 o Zelda: Ocarina of Time.

Game boy

Procesador	Z80 de 8 bits.
Velocidad de Reloj	8 Mhz
Memoria RAM	32 K
Canales de Sonido	4
Paleta de Colores	4096 en modo mapa de bits, 56 en modo personaje (la GB normal tiene 4 tonos de gris).
Pantalla	TFT 160 x 144 píxeles color reflectante.
Formato de juego	Cartucho
Extras	Posibilidad de conectar 2 GB por cable link e intercomunicación con Nintendo 64.
Comentarios	Después de las clásicas Game & Watch, Game Boy revolucionó el mundo de las consolas vendiendo más que cualquier otra portátil ha vendido nunca, en 1998 salió GB Color la cual le aseguró el liderazgo a Nintendo en el mundo de las consolas de bolsillo.

Game gear

Procesador	Motorola Z80 3,58 Mhz 8 Bits
Velocidad de Reloj	3,58 Mhz
Memoria RAM	24 Kb
Memoria de Vídeo	16 Kb
Canales de Sonido	4
Paleta de Colores	4096
Pantalla	TFT 240 x 226 PIXELS color reflectante
Formato de juego	Cartucho
Extras	Posibilidad de conectar hasta 2 GG por cable link y adaptador de T.V.
Comentarios	Fue una de las muchas rivales de Game Boy, quizá la más dura. Contaba con pantalla a color muy superior a la de la GB en blanco y negro. Aún así fue derrotada por su gran gasto de pilas y su peso descomunal.

Game boy advanced

Procesador	RISC 32 Bits más 8 Bits en el CISC
Velocidad de Reloj	16 Mhz
Memoria RAM	128 K WRAM + 256 K fuera de la CPU
Memoria de Vídeo	96 K
Canales de Sonido	4
Paleta de Colores	32768 en modo mapa de bits, 512 en modo personaje.
Pantalla	TFT 240 x 160 pixeles color reflectante y panorámica.
Formato de juego	Cartucho
Extras	Posibilidad de conectar hasta 4 GBA por cable link e intercomunicación con GC.
Comentarios	La primera portatil de Nintendo de 32 bits, perfecta para los remakes de Snes y otras consolas, ha batido records de ventas y posee un catálogo de juegos tremendamente extenso, variado y divertido. Su versión SP cuenta con pantalla retroiluminada.

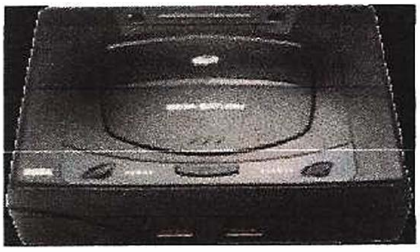
Neo geo

Procesador	Microprocesador 68000 14 Mhz y Z80 24 bits.
Memoria RAM	64Kb
Memoria de Vídeo	68Kb
Gráficos	4096 colores simultáneos de una paleta 65.536. Zoom
Canales de Sonido	7 digitales 8 FM
Puertos de Mandos	2
Formato de juego	Cartucho
Comentarios	La Neo Geo es una consola que nunca llegó a cuajar a pesar de ser la más avanzada técnicamente de su época. Tuvo varios juegos de lucha considerados clásicos, pero su alto precio y su falta de variedad de juegos la hizo caer en picado.

Play station

Procesador	R3000A RISC a 33 Mhz
Memoria RAM	2 Mb
Memoria de Vídeo	1 Mb
Memoria de Sonido	512 Kb
Canales de Sonido	24
Polígonos por segundo	360.000
Puertos de Mandos	2
Formato de juego	CD
Extras	Reproductor de CD de música..
Comentarios	La PSX es la consola más conocida y también la más vendida del mundo con más de 100 millones de consolas en todo el mundo. Triunfó sobre todo por su extensísimo catálogo de juegos debido a las múltiples compañías que trabajaron para ella.

Sega saturno



Procesador	Hitachi SH2 de 32 bits RISC 28,6 MHz
Procesador Gráfico	2 coprocesadores gráficos VDP1 y VDP2
Memoria RAM	2 Mb
Memoria de Vídeo	1,54 Mb
Memoria de Sonido	540 Kb
Canales de Sonido	32
Polígonos por segundo	200.000 texturados
Puertos de Mandos	2
Formato de juego	CD
Extras	Reproductor de Video CD con software especial.
Comentarios	Sega Saturn fue la perdedora en la generación de 32 bits. Se enfrentó a la Playstation y no consiguió batirla. Al contrario que en Japón, donde sí consiguió unas ventas fabulosas.

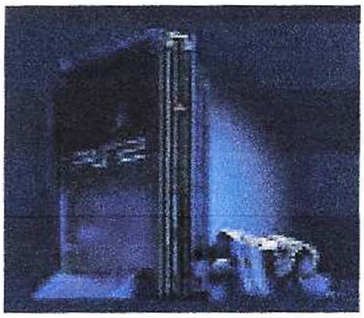
Dreamcast

Procesador	Hitachi SuperH SH-4 RISC CPU 200MHz
Procesador Gráfico	Nec Power VR2
Memoria RAM	16 Mb
Memoria de Vídeo	8 Mb
Memoria de Sonido	2 Mb
Canales de Sonido	48
Polígonos por segundo	3.500.000
Puertos de Mandos	4
Formato de juego	CD
Extras	Disponibilidad de juego online.
Comentarios	Dreamcast fue una consola que pasó sin pena ni gloria por la pasarela del videojuego. Tuvo dos Sonics revolucionarios, pero Sega paró su fabricación espontáneamente por lo que no pudo durar lo que muchos hubieran querido, siendo la última consola de la compañía.

Game Cube

Procesador	IBM "Gekko" 485 Mhz
Procesador Gráfico	NEC "Flipper" 162 Mhz
Memoria RAM	40 Mb
Memoria de Vídeo	24 Mb
Memoria de Sonido	16 Mb
Canales de Sonido	64
Polígonos por segundo	12.000.000 texturados
Puertos de Mandos	4
Formato de juego	Mini-DVD Panasonic
Extras	Conectividad con GBA y posibilidad de juego online.
Comentarios	Se puede decir que Game Cube esta entre la potencia de X-box y PS2, tiene el catalogo menos extenso de las tres pero el más carismático y original. Falla por su parte en las opciones de DVD e Internet, aunque cuenta con la mayor experiencia del mercado.

Play station 2



Procesador	Sony "Emotion Engine" 295 Mhz
Procesador Gráfico	Sony Graphics Synthesizer 150 Mhz
Memoria RAM	32 Mb
Memoria de Vídeo	4 Mb
Memoria de Sonido	4 Mb
Canales de Sonido	48
Polígonos por segundo	66.000.000
Puertos de Mandos	2
Formato de juego	DVD-Video Sony
Extras	Reproductor de DVD y CD de música, disponibilidad de juego online.
Comentarios	La PS2 es la consola más popular y vendida dentro de la generación de 128 bits, a pesar de ser la menos potente, cuenta con un extenso e inmejorable catálogo de juegos.

X-BOX

Procesador	Intel Pentium III / 733 Mhz
Procesador Gráfico	nVidia G-Force X-Box 250 Mhz
Memoria RAM	64 Mb (unificados)
Memoria de Vídeo	Opcional dentro de los 64 Mb
Memoria de Sonido	Opcional dentro de los 64 Mb
Canales de Sonido	64
Polígonos por segundo	125.000.000
Puertos de Mandos	4
Formato de juego	DVD-Video Thomsom
Extras	Reproductor de DVD y CD de música, disponibilidad de juego online, disco duro de serie 8 GB y decodificador de Dolby Digital 5.1.
Comentarios	Es la consola más potente de la generación de los 128 Bits, con un impresionante Hardware. Si no a triunfado ha sido por la faltas de juegos exclusivos. (Por muy explosiva que sea no tiene bombazos).

6.2. CAPÍTULO 2:

ANIMACIÓN 2D Y 3D

6.2.1. PLANTEAMIENTO DEL CAPÍTULO

En este capítulo estudiaremos la animación de los videojuegos, como son sus apariencias, la calidad de imagen, la animación 2D y 3D, el hardware que requiere, además de una exposición de los cambios que ha generado los avances de la animación.

6.2.2. OBJETIVOS DEL CAPÍTULO

Es deseable que al finalizar el capítulo el estudiante:

- Conozca el concepto de animación 2D y 3D.
- Comprenda y entienda las diferencias entre animación 2d y 3D.
- Conozca la evolución de la animación con respecto al video.
- Maneje la información de cómo son los productos deseados por el mercado.

6.2.3. Explicación del contenido

6.2.3.1. Introducción

La animación y la ilustración son piezas claves en para crear un buen videojuego Ambas nos permiten jugar con diversos elementos y así crear desde un logotipo animado hasta personajes y mascotas interactivas que apoyen el contenido del juego, generando de este modo un producto atractivo para el cliente.

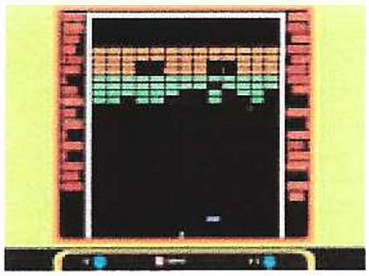
6.2.3.2. ANIMACIÓN 2D Concepto:

Las animaciones 2D (dos dimensiones) son aquéllas compuestas por dibujos planos, o sea, que para definir un punto se necesitan dos coordenadas (x-y).

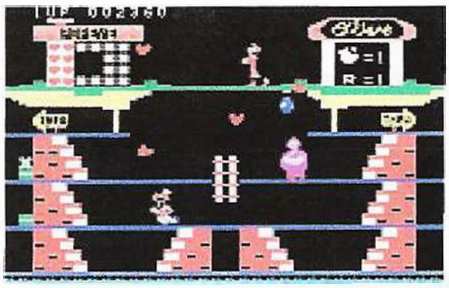
Ya no son muy aceptadas por el mercado.

6.2.3.2.1. Características:

- En un principio tenia poca realidad, solo eran figuras de colores con movimiento en solo 2 direcciones.



- La animación no pasa de mostrar una simple imagen moviendo en solo dos direcciones, izquierda o derecha



- En cuanto surgió la animación 3D su uso ha bajado bastante.
- Su poca complejidad hace de ella una herramienta fácil de usar

6.2.3.2.2. Exigencias en hardware

En realidad la animación podía funcionar con cualquier hardware, ya que por sus pocas prestaciones no exige tanto video o resoluciones altas para sus texturas.

6.2.3.2.3. Personal

Un juego con animación 2D podía ser desarrollado inclusive por una sola persona, Bill Nighinbottham por ejemplo, creador del primer videojuego,

Para su complejidad aumento hasta una cantidad de 10 personas.

6.2.3.3. ANIMACIÓN 3D Concepto:

Las animaciones con dibujos espaciales (con profundidad), son de tres dimensiones (3D). Sus puntos están definidos por tres coordenadas (x-y-z).

Más atractivos y llamativos al mercado.

6.2.3.3.1. Características

- Supera el efecto de la animación en 2 dimensiones basándose en tratar de imitar la realidad.

- Trabaja con texturas mas sofisticas para dar un efecto casi perfecto

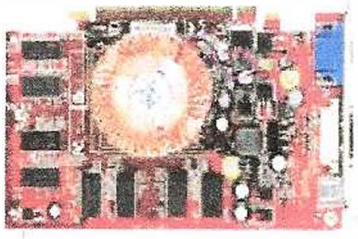
- Entre mas sofisticada sea la maquina que la reproduce mejor será su rendimiento.

- Para su entorno requiere orientarse a objetos.

- Con un buen desarrollo puede hacer hasta personajes humanos bastante reales.

6.2.3.3.2. Exigencias en hardware

La animación 3D a diferencia de la 2D para una mejor visualización requiere hardware especializado para sus texturas y animación, Una tarjeta ordinaria pocas veces podrá soportar sus exigencias y presentara conflicto o perdida de calidad, para ello corporaciones como ATI, Nvidia, y otros han desarrollado tarjetas de video mas poderosas para poder ofrecer un mejor rendimiento.



6.2.3.3.3. Personal

Para desarrolla un juego con animación 3D se requiere de un enorme personal capacitado, a diferencia de los juegos en 2D, este tipo de proyectos requiere equipos sofisticados y pueden durar años para terminar un juego de calidad.

6.3. CAPÍTULO 3:

MOTORES DE VIDEOJUEGOS[4]

6.3.1. Planteamiento del capítulo

En el capítulo anterior se comento acerca de la animación 2D y 3D, el gusto por las personas de la realidad que ofrecen cada vez más los videojuegos, además de sus grandes avances en las texturas de las animaciones, etc. En este capítulo se comentara de las herramientas que permiten su desarrollo, los motores de videojuegos con los que cuenta el mercado de software.

6.3.2. Objetivos del capítulo

Es deseable que al finalizar el capítulo el estudiante:

- Maneje el concepto de motor de videojuego.
- Conozca los motores de videojuegos usados en el mercado.
- Entienda las características de cada motor de juegos.
- Diferencie entre motores con licencia y motores sin licencia.
- Sea capaz de enumerar las ventajas y desventajas que conlleva elegir un motor de videojuegos.

6.3.3. Explicación del contenido

6.3.3.1. Introducción

En el capítulo anterior se estudiaron las características de la animación 2D y 3D, la mejora de la animación 3D con respecto a la 2D. Una de las características de la mejora en la animación 3D era el tipo de hardware de video del que disponga el cliente, y demás.

Para el desarrollo de esta animación (2D y 3D) es necesaria una herramienta sofisticada un motor grafico especializado para este fin.

En el mercado existen muchos paquetes de motores gráficos entre ellos tenemos los siguientes candidatos:

3D-GAMESTUDIO	**CRYSTAL SPACE**
FLY3D	**UNREAL**
GENESIS3D	**ENTIDAD 3D**
QUAKE 2	**SHARK3D**
OGRE	**NEBULA**
TORQUE (V12)	**IRRLICHT ENGINE**

6.3.3.2. Motores de videojuegos

6.3.3.2.1. 3D-GAMESTUDIO

Autores: Conitec Corporation
Página Web: http://www.conitec.net/a4info.htm

Descripción:
3D GameStudio es un kit de desarrollo para la realización de juegos de ordenador. Provee de un motor 3D, un motor 2D, un editor de niveles y modelos, compilador de scripts y librerías de modelos, texturas, etc. Manipula con igual rendimiento escenas de interior y de exterior. Tiene un motor de iluminación que soporta sombras verdaderas y fuentes de luz en movimiento. El principal objetivo que esta aplicación persigue es que el creador del juego no necesite ser un programador experimentado. Abogan por reducir al máximo el esfuerzo del creador a costa de perder flexibilidad en el diseño del juego. Sin embargo, admiten que los juegos realizados con poca o ninguna programación serán siempre juegos poco ambiciosos. Ofrecen tres posibilidades para crear un juego:

1. Juegos diseñados a base únicamente de ratón, para usuarios sin conocimientos de programación.

2. Juegos o efectos diseñados con algo de programación utilizando C-scripts, para el que quiere algo más.

3. Juegos o efectos programados en C++ o Delphi, para programadores con experiencia.

6.3.3.2.2. CRYSTAL SPACE

Autores: Jorrit Tyberghein

Pagina Web: http://crystal.sourceforge.net/

Descripción:

Cristal space es un kit de desarrollo de juegos 3D libre (LGPL) y portable escrito en C++. Soporta seis grados de libertad, luces de colores, mipmapping, portales, espejos, transparencias, superficies reflectivas, sprites 3D (basados en frames o animaciones de esqueleto), texturas procedurales, radiosidad, sistemas de partículas, halos, niebla volumétrica, lenguaje de script (Python y otros), soporte
para visualización a 8-bits, 16-bits y 32-bits, Direct3d, OpenGL, Glide, y render por software, soporte para fuentes, transformaciones jerárquicas, etc...

Actualmente Crystal Space puede ejecutarse sobre GNU/ Linux, Windows, Windows NT, OS/2, BeOS, NextStep, OpenStep, MacOS/X Server, DOS, y Macintosh entre otros. Crystal Space es un gran proyecto para el desarrollo de software abierto. Hay alrededor de 600 personas subscritas a las listas de correo del Crystal Space.

6.3.3.2.3. FLY3D

Autores: Alan Watt, Fabio Policarpo
Página Web: http://www.fly3d.com.br/

Descripción:

Fly3D es un motor de juegos que acompaña al libro 3D Games de Alan Watt y Fabio Policarpo. El motor se encuentra actualmente en la segunda revisión de su versión 2.0. La versión final estará disponible en el segundo volumen del libro que aparecerá en Septiembre del 2002.

Como la versión 1.0, la nueva versión esta desarrollada en C++ y todo el código específico del juego se encuentra desarrollado con DLLs plugins (se incluye una herramienta para crear plugins para Fly3D).

El nuevo motor incluye nuevas características como shaders y un nuevo editor de shaders, varios tipos de superficies curvas, ejecutables ActiveX para aplicaciones en Web, compatibilidad con los editores de niveles del Quake3 y nuevos formatos de mallas para objetos estáticos, animación de vértices (.F3D) y objetos con esqueletos animados (.K3G).

La nueva versión del motor requiere Windows 9x/Me/2k/NT4 para ejecutarse. Para desarrollar nuevos plugins es necesario utilizar Visual C++. Se recomienda una buena tarjeta gráfica (nvidia).

6.3.3.2.4. UNREAL

Autores: Empresa Epic MegaGames
Página Web: http://www.epicgames.com

Descripción:

Motor gráfico del juego Unreal desarrollado por la empresa Epic MegaGames basado en una extensión del renderizado en portales conocido como *Dynamic Scene Graph Technology (DSG)*, BSP y radiosidad. Su principal característica es la gran escalabilidad (muy modular).
Se encuentra disponible para las plataformas Linux, Windows, Macintosh, Playstaion 2 y Xbox.
Para desarrollar con este motor se tiene que adquirir una licencia
(USD 250.000 - USD 500.000) que da acceso a todo el código fuente, a las herramientas y juegos.

6.3.3.2.5. GENESIS3D

Autores: Eclipse Enterteinment

Página Web: http://www.genesis3d.com/

Descripción:

Genesis3D es un motor para la visualización de escenas tridimensionales en tiempo real y que permite construir aplicaciones gráficas 3D de altas prestaciones. Ha sido diseñado principalmente para la visualización de escenas de interior logrando un alto 'frame-rate' siempre y cuando estén compuestas por una cantidad moderada de polígonos. También puede ser utilizado para escenas de exterior si el diseño de las escenas se realiza tomando ciertas precauciones. Sus principales características son la detección rápida de colisiones, iluminación precalculada y chequeo de la visibilidad. Su principal inconveniente es la visualización de escenarios exteriores sin imponer algún tipo de restricción o límite al tamaño de la escena.

6.3.3.2.6. ENTIDAD 3D

Autores: Jordi Pérez

Página Web: http://www.entidad3d.com/

Descripción:

Entidad3D fue creado con base en el motos de videojuegos Genesis3D es un motor para la visualización de escenas tridimensionales en tiempo real y que permite construir aplicaciones gráficas 3D sin necesidad de programar, es una herramienta de libre uso de altas prestaciones. Ha sido diseñado principalmente para la visualización de escenas de interior logrando un alto 'frame-rate' siempre y cuando estén compuestas por una cantidad moderada de polígonos. También puede ser utilizado para escenas de exterior si el diseño de las escenas se realiza tomando ciertas precauciones. Sus principales características son la detección rápida de colisiones, iluminación precalculada y chequeo de la visibilidad. Su principal inconveniente es la visualización de escenarios exteriores sin imponer algún tipo de restricción o límite al tamaño de la escena.

6.3.3.2.7. QUAKE 2

Autores: John Carmak

Página Web: desconocido

Descripción:

Es el motor grafico del videojuego Quake 2, desarrollado por John Carmak de Id. Software.

Se basa en el renderizado por árboles BSP y radiosidad.

Soporta plataformas Windows, Linux i Macintosh.

Su Licencia oscila entorno los 250.000 USD que da acceso al código fuente.

6.3.3.2.8. SHARK3D

Autores: SPINOR

Página Web: http://www.shark3d.com

Descripción:

El motor de juegos Shark3D es un kit de desarrollo de alto nivel para aplicaciones3d en tiempo real. Está orientado a juegos, aplicaciones de Realidad Virtual visualización. Optimizado para aplicaciones multiusuario a través de una red de computadores o Internet.

6.3.3.2.9. OGRE

Autores: The OGRE Team
Página Web: http://ogre.sourceforge.net

Descripción:

OGRE (Object-Oriented Graphics Rendering Engine) es un motor escrito en C++ flexible, orientado a escenas, y diseñado para hacer más simple e intuitiva a los desarrolladores la producción de juegos utilizando hardware de aceleración 3D. La librería de clases permite abstraer los detalles asociados a las librerías de bajo nivel (OpenGL o Direct3D), proporcionando una interfaz basada en objetos.

6.3.3.2.10. NEBULA

Autores: Radon Labs

Página Web: http://www.radonlabs.de/

Descripción:

"The Nebula Device" es un nuevo motor de juegos de calidad profesional, que puede utilizarse de forma gratuita. Sus creadores, son el equipo que desarrollo "Urban Assault" (Publicado pro Microsoft en 1998). Actualmente utilizan nebula para desarrollar su nuevo juego "The Nomads" (autorizado para Xbox).

Nebula es un motor de arquitectura moderna. Desarrollado en C++ y orientado a objetos, sus clases dlls se cargan de forma independient en tiempo de ejecución. El motor puede ejecutarse en Linux, Win9X, WinNt. Permite intercambiar sin interrupción la visualización con OpenGL y Direct3D.

Nebula utiliza como lenguaje de script el estandar tcl/tk.

Los principales objetivos de nebula son los siguientes:

- Independencia de la plataforma
- Gestión de la base de datos del juego: jerarquías 3D, texturas, materiales, luces, sonidos, animaciones, estados y sus relaciones.
- Proporcionar herramientas básicas para el trabajo en equipo para el desarrollo del juego.

6.3.3.2.11. TORQUE (V12)

Autores: DINAMIX

Página Web: http://www.garagegames.com

Descripción:

The Torque Game Engine (TGE) es el motor desarrollado por DINAMIX para su juego *Tribes2*. Enfocado a la simulación de misiones militares, incluye utilidades para la creación de terrenos, superficies acuáticas, interiores estilo portal y sistemas de partículas. También incluye soporte multiplataforma (Windows, Mac OS y Linux), soporte para red, creación de interfaces de usuario y lenguaje de script estilo C++. Permite importar objetos desde 3D Studio MAX y dispone de librerías matemáticas, de detección de colisiones, de física de vehículos y una Base de Datos Espacial.

6.3.3.2.12. IRRLICHT ENGINE

Autores: Nikolaus Gebhardt

Página Web: irrlicht.sourceforge.net

Descripción:

El Irrlicht Engine esta diseñado para ser un engine 3D de fácil uso, posee gran cantidad de prestaciones, emplea el paradigma orientado a objetos se apoya en otras librerías LGPL para aumentar sus prestaciones, esta orientado a eventos, puede renderizar tanto por Hardware como por Software, una gran cantidad de proyectos la están utilizando en Internet, tiene buena documentación.
Maneja todo tipo de texturas, es compatible con Visual .NET, Visual C++ 60, Dev-CPP.
Es para las plataformas LINUX, MAC OS, WINDOWS.
Soporta varios tipos de maya, maneja librería de colisión,.....

6.4. CAPÍTULO 4:

CARACTERÍSTICAS DE LOS MOTORES DE VIDEOJUEGOS

6.4.1. Planteamiento del capítulo

En el capítulo anterior se comento acerca de los motores de videojuegos, los que más se utilizan, las herramientas con que trabaja, además de sus creadores y forma de obtenerlos. En este capítulo se mostraran las características de cada uno de ellos, sus ventajas y desventajas.

6.4.2. OBJETIVOS DEL CAPÍTULO

Es deseable que al finalizar el capítulo el estudiante:

- Conocer las características de cada uno de los motores de videojuego.
- Comprender por que se ha seleccionado el motor Irrlicht Engine para programar video juegos en este proyecto.
- Tener el conocimiento de las disponibilidades que brinda cada motor de videojuego.

6.4.3. Introducción

En el capítulo 3 se comento de los motores de videojuegos, se dio una definición de su funcionamiento y de sus creadores, en apariencia todos son excelentes, pero siempre hay uno que sobresale, es este capitulo se analizaran las características de cada 1 y se aclarara por que se trabajara con el motor Irrlicht Engine.

6.4.3.1. Características de cada motor de videojuego

A continuación se mostraran las utilidades que facilitan cada motor, su disponibilidad, compatibilidades y demás:

3D-GAMESTUDIO

- Plataformas : Widows
- Documentación: Con licencia
- Driver Soportados: Dired3d, OpenGL
- Mayas compatibles: 3DS
- Texturas compatibles : JPG, BMP
- Compatibilidad con otras librerias: No
- Efecto de partículas: No
- Detección de colisiones : Si
- Lenguajes : C++, Delphi
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : Copyright

CRYSTAL SPACE

- Plataformas : Windows, Linux, Mac OS
- Documentación: Sí
- Driver Soportados: OpenGL, Dired3D
- Mayas compatibles: 3DS, MD2 (Quake 2), OBJ, POV y ASE
- Texturas compatibles: Mipmapping, procedurales
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : C++
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : GNU Libre

FLY3D

- Plataformas : Windows
- Documentación: no
- Driver Soportados: OpenGl, Dired3D
- Mayas compatibles: 3DS, Quake3
- Texturas compatibles: multitextura
- Compatibilidad con otras librerías: No
- Efecto de partículas: No
- Detección de colisiones : Si
- Lenguajes : C++
- Código : C++
- Iluminación: Sí
- Renderización: Sí
- Licencia: Copyright

UNREAL

- Plataformas : Windows, Linux, Mac,PS2, XBOX
- Documentación: Sí
- Driver Soportados: OPENGL
- Mayas compatibles:Quake3,LOD
- Texturas compatibles: Multitextura
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : UnrealScript (Propio)
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : Copyright

GENESIS3D

- Plataformas : Windows
- Documentación: No
- Driver Soportados: Dired3d
- Mayas compatibles: BSP, LOD, 3DS
- Texturas compatibles: BMP
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : Scrip propio
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia: Copyright

ENTIDAD 3D

- Plataformas : Windows
- Documentación: Sí
- Driver Soportados: Dired3d
- Mayas compatibles: 3DS,BSP,LOD
- Texturas compatibles: BMP
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : Script propio
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : GNU

QUAKE 2

- Plataformas : Windows, Linux, Macintos
- Documentación: Sí
- Driver Soportados: Dired3d, OpenGL
- Mayas compatibles: BSP
- Texturas compatibles: Multitextura
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : script llamado QuakeC.
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : Copyright

SHARK3D

- Plataformas : Windows
- Documentación: No
- Driver Soportados: Dired3D
- Mayas compatibles: LOD
- Texturas compatibles: Per-pixel
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : (propio)
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia Copyright

OGRE

- Plataformas : Windows
- Documentación: Sí
- Driver Soportados: Dired3d
- Mayas compatibles: LOD,3DS, Quake3, Mesh
- Texturas compatibles: Mapping
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : Sí
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : Copyright

NEBULA

- Plataformas : Windows, Linux
- Documentación:No
- Driver Soportados: Dired3d
- Mayas compatibles:OBJ,3DS,Quake3
- Texturas compatibles Bmp
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : No mencionado
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : Copyright

TORQUE (V12)

- Plataformas : Windows, Mac, Linux
- Documentación: No
- Driver Soportados: OpenGl, Dired3D
- Mayas compatibles: LOD,3DS,MilkShape
- Texturas compatibles : pixel
- Compatibilidad con otras librerías: No
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : No mecionado
- Código : No mencionado
- Iluminación: Sí
- Renderización: Sí
- Licencia : Copyright

IRRLICHT ENGINE

- Plataformas : Windows, Linux, Mac
- Documentación: Sí
- Driver Soportados: OpenGl, Dired3d (8 y 9), Soft Render, Soft Render 2
- Mayas compatibles: OBJ, 3DS, OCT, B3D, CSM, DAE, XML, DMF, X MY3D, MS3D, MESH, LMTS, BSP, MD2 ...
- Texturas compatibles : Multitextura
- Compatibilidad con otras librerías: Sí
- Efecto de partículas: Sí
- Detección de colisiones : Sí
- Lenguajes : C++, NET, JAVA
- Código : Sí
- Iluminación: Sí
- Renderización: Sí
- Licencia : GNU Libre

Como se puede observar el motor IRRLICHT ENGINE en comparación con los demás motores es compatible con diversos tipos de mayas además de aceptar otras librerías fuera de su SDK, permite trabajar en tres tipos de lenguajes, es de libre licencia y posee la mayoría de las cualidades de los otros motores.

A partir de este capitulo se hará un estudio completo al motor de desarrollo Irrlicht Engine.

6.5. CAPÍTULO 5:

CONOCIENDO EL MOTOR IRRLICHT ENGINE

6.5.1. PLANTEAMIENTO DEL CAPÍTULO

En los capítulos 3 y 4 se comento de los motores de videojuegos disponibles en el mercado y sus características, después de haber estudiado cada motor grafico, llego el momento de hablar del motor con que se trabajara en los lineamientos, en este momento el lector va conocer las utilidades de el motor Irrlicht.

6.5.2. OBJETIVOS DEL CAPÍTULO

Es deseable que al finalizar el capítulo el estudiante:

- Conozca las disponibilidades que ofrece el motor **IRRLICHT ENGINE**.
- Identificar los objetos 3D que son compatibles con el **IRRLICHT ENGINE.**
- Conocer apropiadamente las clases del **IRRLICHT ENGINE**
- Manejar el código del motor **IRRLICHT ENGINE**

6.5.3. Explicación del contenido

6.5.3.1. Introducción

Los motores gráficos son la herramienta de desarrollo diseñados para la creación de videojuegos, existen de varios tipos, con licencia, GNU, GPL y LGPL.
Para el desarrollo de un videojuego es necesario usar la herramienta adecuada. Para ello es necesario analizar los motores disponibles en el mercado y tomar la mejor decisión.

6.5.3.2. MOTOR IRRLICHT ó IRRLICHT ENGINE

6.5.3.2.1. Descripción:

IRRLICHT ENGINE es un paquete de desarrollo de juegos 3D libre (LGPL) y portable escrito en C++. Soporta seis grados de libertad, luces de colores, mapas, portales, espejos, transparencias, sprites 3D (basados en frames o animaciones de esqueleto), texturas procedurales, radiosidad, sistemas de partículas, halos, niebla volumétrica, soporte para visualización a 8-bits, 16-bits y 32-bits, Direct3D, OpenGL y render por software, soporte para fuentes, transformaciones jerárquicas, etc.
Actualmente IRRLICHT ENGINE puede ejecutarse sobre GNU/ Linux, Windows, Windows NT, OS/2, MacOS/X Server, DOS, y Macintosh entre otros. IRRLICHT ENGINE es un gran proyecto para el desarrollo de software abierto.

6.5.3.2.2. Características que implementa:

- Modelado
    1. Motor de terreno
    2. Motor de física
    3. Mallas 3D con animación. Conversores desde los formatos Quake MDL y Quake II MD2 a IRRLICHT ENGINE. Importadores de objetos 3DS, MDL, MD2, OBJ, POV, OCT, My3D, Mesh, DAE, XML, DMF, X, B3D, CSM, MS3D, LMTS, y ASE. Las mallas son multirresolución "progressive meshes" permitiendo LOD dinámicos.
    4. Trabajo futuro para representar superficies curvas (nurbs)
    5. Soporte para superficies curvas (Beziers,...).

- Visibilidad
    1. Sistema de visibilidad basado en la combinación de portales, octrees.

- Iluminación
    1. Cielo iluminado dinámicamente, sol en movimiento
    2. Espejos
    3. Luces dinámicas, de colores con sobras suaves
    4. Radiosidad precalculada sobre los ligthmaps
    5. Niebla volumétrica

- Texturas
    1. Texturas de cualquier dimensión y formatos GIF, PNG, BMP, JPG y otros. Pueden aplicarse con transformaciones (escalado, rotación, espejo)
    2. Corrección perspectiva con interpolación cada 16 píxel
    3. Texturas con canal alpha
    4. Mipmapping
    5. Soporte para texturas dinámicas.
    6. Multitexturas con OpenGL

- Efectos
    1. 2D sprites y sistemas de partículas utilizando esos sprites

- Otros
    1. Plugins para fuentes
    2. Sistema de detección de colisiones jerárquico

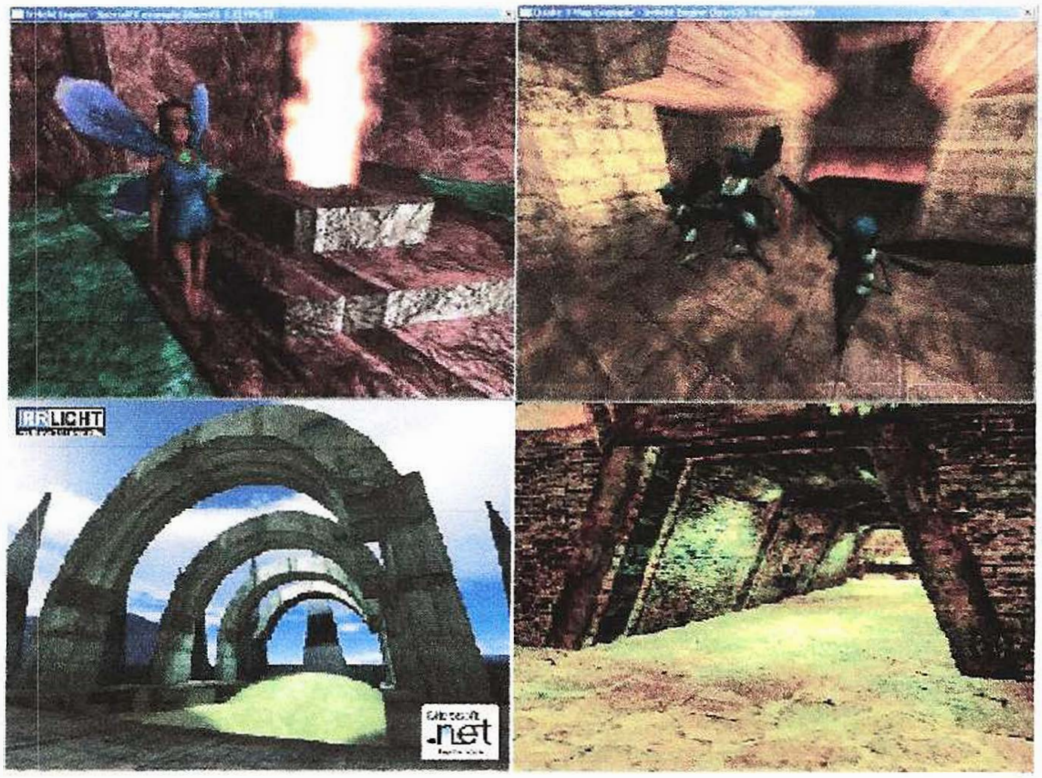
6.5.3.2.3. Utilidades:

- Se incluye un conversor de 3DS
- Se incluyen varios scripts de Blender para exportar modelos y mapas al Irrlicht Engine
-

Precio:

Gratuito LGPL

6.5.3.2.4. Imágenes



6.5.3.2.5. MODELADO

Patch/curved surf NO

Hierarchical obj. -

Scene graph -

Level of Detail SI

6.5.3.2.6. VISIBILIDAD

Zbuffer rendering -

Portal rendering SI

BSP rendering SI

6.5.3.2.7. ILUMINACIÓN

Radiosity SI

Mirrors SI

Reflective surfaces SI

Colored lighting SI

Point Lighting SI

Dynamic Lighting SI

Lightmap Rendering SI

Phong Shading -

Bump mapping -

Gouraud-shading SI

6.5.3.2.8. TEXTURAS

Texture mapping SI

Persp. Texture SI

MIP-mapping SI

Sub-pixel map -

Procedural textures SI

6.5.3.2.9. EFECTOS

Fogging SI

Halo / Corona SI

Motion blur -

Lens flare SI

Cartoon -

Particle system SI

6.5.3.2.10. APIS

OpenGL SI

Direct3D SI

6.5.3.2.11. OTROS

Anti- aliasing -

Degrees of freedom 6

Collision detect SI

3D sound NO

Language C ++

Platform Windows, Linux, Mac

Source SI

Cost

File formats

6.6. CAPÍTULO 6:

COMO UTILIZAR EL MOTOR IRRLICHT ENGINE

6.6.1. PLANTEAMIENTO DEL CAPÍTULO

Este es el capitulo mas extenso de todos, aquí se hará una explicación de el modo de utilizar el motor de desarrollo IRRLICHT ENGINE, los namespaces, clases y estructuras que se manejan además del tipo de declaración de variables para el buen usos de las librerías, se aclara que en este capitulo solo se explicara el paquete de desarrollo del Irrlicht, tenga en cuenta que esta es la base para manejar apropiadamente las funciones de Irrlicht si con una sola lectura no comprende este capitulo se recomienda volver a leer hasta comprender el uso de los componentes.

6.6.2. OBJETIVOS DEL CAPÍTULO

Es deseable que al finalizar el capítulo el estudiante:

- Identificar los namespaces que maneja el **IRRLICHT ENGINE.**
- Identificar la jerarquía de los namespaces del motor
- Conocer las clases y estructuras de cada namespace del **IRRLICHT.**
- Identificar clases y estructuras del namespace **irr**
- Saber como declarar una variable tipo **irr**
- Manejar una clase del namespace **irr**

- Identificar clases y estructuras del namespace **core (irr::core)**
- Saber como declarar una variable tipo **core**
- Manejar una clase del namespace **core**
- Identificar clases y estructuras del namespace **gui (irr::gui)**
- Saber como declarar una variable tipo **gui**
- Manejar una clase del namespace **gui**
- Identificar clases y estructuras del namespace **io (irr::io)**
- Saber como declarar una variable tipo **io**
- Manejar una clase del namespace **io**
- Identificar clases y estructuras del namespace **scene (irr::scene)**
- Saber como declarar una variable tipo **scene**
- Manejar una clase del namespace **scene**
- Identificar clases y estructuras del namespace **video (irr::video)**
- Saber como declarar una variable tipo **video**
- Manejar una clase del namespace **video**

6.6.3. Explicación del contenido

6.6.3.1. Introducción

El manejo apropiado del motor IRRLICHT ENGINE significa saber manejar apropiadamente cada clase, estructura y variable disponible en su paquete no poseer esta información significaría adivinar a manejar IRRLICHT.

La mayoría de las clases no son obligatorias solo están ahí disponibles para cuando la requiramos, además de permitirlo agregar otra librería si el motor no lo posee, no explicado en este capitulo para no confundir las librerías.

En la pagina oficial (irrlicht.sourceforge.net) se puede conseguir mas información acerca de manejos apropiados con el motor IRRLICHT además de un foro que te permitirá ampliar tu conocimiento, además de compartir opiniones e ideas para ayudarse entre si.

6.6.3.2. Lo primero que se debe conocer

Para comenzar a usar el IRRLICHT ENGINE es necesario saber cual es el paso a seguir para un buen desarrollo aquí empezamos:

Los namespaces

Como se puede observar, el Engine usa namespaces. Todo en el Engine se encuentra en el namespace **irr**, pero hay también 5 sub-namespaces. Puedes encontrar una lista de todos los namespaces con las descripciones en la página del namespaces. Éste es el mejor lugar para empezar la lectura la documentación. Si usted no quiere escribir los namespaces todo el tiempo, simplemente haga una declaración global de la siguiente forma: Los namespaces son:

```
using namespace irr;
using namespace core;
using namespace scene;
using namespace video;
using namespace io;
using namespace gui;
```

Aquí esta una lista de todos los namespaces del **IRRLICHT ENGINE** con sus descripciones:

irr	Todos los recursos del Irrlicht Engine se encuentran en este namespace.
irr::core	En este namespace pueden encontrarse las clases básicas para vectores, planos, arreglos, listas, y todo lo relacionado a manejo de coordenadas.
irr::gui	Este namespace contiene clases útiles para la fácil creación de interfaces graficas de usuario
irr::io	Este namespace provee interfaces de input/output (Entrada/Salida): leyendo y escribiendo archivos, acceso a archivos .ZIP, archivos .XML,...
irr::scene	Todo lo necesario para administrar la escena puede encontrarse en este namespace: Cargar mayas, Nodos de escena especiales, Octree y Billboards,...
irr::video	El namespace video contiene las clases para acceder al controlador de video. Todos las vistas 2D y 3D se renderizan aqui.

Namespace irr;

Las clases básicas de este namespace son las siguientes:

- **Clase IEventReceiver**
  Interfaz de un objeto que puede recibir eventos.

- **Clase ILogger**
  Interfaz para mostrar mensaje, advertencias y errores.

- **Clase IOSOperator**
  El operador del sistema operativo proporciona los métodos específicos del sistema e informaciones.

- **Clase IrrlichtDevice**
  El Irrlicht Device. Permite crear **createDevice ( )** o **createDeviceEx ( ).**

- **Clase ITimer**
  Interfaz para poder manipular el tiempo virtual.

- **Clase Iunknown**
  Clase base para agregar más objetos al Irrlicht Engine

- **EEstructuraura SEvent**
  EEstructuraura para sostener los datos de eventos. Un evento puede ser de Interfaz Grafica de Usuario, ratón o eventos del teclado.

- **EEstructuraura SirrlichtCreationParameters**
  EEstructuraura para sostener las creaciones de parámetros avanzados para el Irrlicht Device.

- **EEstructuraura SKeyMap**
  EEstructuraura que guarda una llave perteneciente a una acción
  Ej. llave → Key_Key_K, acción → un evento cualquiera,

- **Sub-namespaces**

  core : En este namespace pueden encontrarse las clases básicas para vectores, planos, arreglos, listas, y todo lo relacionado a manejo de coordenadas.

  gui : Este namespace contiene clases útiles para la fácil creación de interfaces graficas de usuario.

  io : Este namespace provee interfaces de input/output (Entrada/Salida): leyendo y escribiendo archivos, acceso a archivos .ZIP, archivos .XML,...

  scene : Todo lo necesario para administrar la escena puede encontrarse en este namespace: Cargar mayas, Nodos de escena especiales, Octree y Billboards,...

  video : contiene las clases para acceder al controlador de video. Todos las vistas 2D y 3D se renderizan aquí.

- **Typedefs (tipos definidos)**

typedef char c8 : Variable tipo carácter de 8 bit.
typedef float f32: Variable tipo punto flotante de 32 bit
typedef double f64 : Variable tipo punto flotante de 64 bit
typedef signed short s16: Variable tipo signed de 16 bit
typedef signed int s32: Variable tipo signed de 32 bit
typedef signed char s8: Variable tipo signed de 8 bit
typedef unsigned short u16: Variable tipo unsigned de 16 bit
typedef unsigned int u32: Variable tipo unsigned de 32 bit
typedef unsigned char u8: Variable tipo unsigned de 8 bit

- **Funciones**

createDevice ( )
Crea un Irrlicht Device. El Irrlicht Device es el objeto raíz para usar el Engine.

createDeviceEx ( )
Crea un Irrlicht Engine con la opción de especificar parámetros avanzados.

Enumeración para todos los tipos de eventos disponibles.

Valores de las enumeraciones:

EET_GUI_EVENT: Un evento de la interfaz gráfica del usuario.

EET_MOUSE_INPUT_EVENT: Un evento de entrada de ratón.

EET_KEY_INPUT_EVENT: Un evento de entrada del teclado.

EET_LOG_TEXT_EVENT: Un evento LOG.

EET_USER_EVENT: Un evento por datos del usuario. Esto no es usado por Irrlicht y puede usarse para enviarle los datos específicos al sistema.

Enumeración para las acciones importantes. Usado por ejemplo en la Cámara de FPS.

Los valores de la enumeración:
EKA_MOVE_FORWARD
EKA_MOVE_BACKWARD
EKA_STRAFE_LEFT
EKA_STRAFE_RIGHT
EKA_COUNT
EKA_FORCE_32BIT

Estos valores no se usan. Es solo para forzar esta numeración a compilar en 32 bit.
La definición a línea 14 del archivo **SKeyMap.h.**

Enumeración de valores de las teclas:

KEY_LBUTTON	KEY_SNAPSHOT	KEY_KEY_Z	KEY_F19
KEY_RBUTTON	KEY_INSERT	KEY_LWIN	KEY_F20
KEY_CANCEL	KEY_DELETE	KEY_RWIN	KEY_F21
KEY_MBUTTON	KEY_HELP	KEY_APPS	KEY_F22
KEY_XBUTTON1	KEY_KEY_0	KEY_SLEEP	KEY_F23
KEY_XBUTTON2	KEY_KEY_1	KEY_NUMPAD0	KEY_F24
KEY_BACK	KEY_KEY_2	KEY_NUMPAD1	KEY_NUMLOCK
KEY_TAB	KEY_KEY_3	KEY_NUMPAD2	KEY_SCROLL
KEY_CLEAR	KEY_KEY_4	KEY_NUMPAD3	KEY_LSHIFT
KEY_RETURN	KEY_KEY_5	KEY_NUMPAD4	KEY_RSHIFT
KEY_SHIFT	KEY_KEY_6	KEY_NUMPAD5	KEY_LCONTROL
KEY_CONTROL	KEY_KEY_7	KEY_NUMPAD6	KEY_RCONTROL
KEY_MENU	KEY_KEY_8	KEY_NUMPAD7	KEY_LMENU
KEY_PAUSE	KEY_KEY_9	KEY_NUMPAD8	KEY_RMENU
KEY_CAPITAL	KEY_KEY_A	KEY_NUMPAD9	KEY_PLUS
KEY_KANA	KEY_KEY_B	KEY_MULTIPLY	KEY_COMMA
KEY_HANGUEL	KEY_KEY_C	KEY_ADD	KEY_MINUS
KEY_HANGUL	KEY_KEY_D	KEY_SEPARATO	KEY_PERIOD
KEY_JUNJA	KEY_KEY_E	KEY_SUBTRACT	KEY_ATTN
KEY_FINAL	KEY_KEY_F	KEY_DECIMAL	KEY_CRSEL
KEY_HANJA	KEY_KEY_G	KEY_DIVIDE	KEY_EXSEL
KEY_KANJI	KEY_KEY_H	KEY_F1	KEY_EREOF
KEY_ESCAPE	KEY_KEY_I	KEY_F2	KEY_PLAY
KEY_CONVERT	KEY_KEY_J	KEY_F3	KEY_ZOOM
KEY_NONCONVERT	KEY_KEY_K	KEY_F4	KEY_PA1
KEY_ACCEPT	KEY_KEY_L	KEY_F5	KEY_OEM_CLEAR
KEY_MODECHANGE	KEY_KEY_M	KEY_F6	KEY_KEY_CODES_COUNT
KEY_SPACE	KEY_KEY_N	KEY_F7	
KEY_PRIOR	KEY_KEY_O	KEY_F8	
KEY_NEXT	KEY_KEY_P	KEY_F9	
KEY_END	KEY_KEY_Q	KEY_F10	
KEY_HOME	KEY_KEY_R	KEY_F11	
KEY_LEFT	KEY_KEY_S	KEY_F12	
KEY_UP	KEY_KEY_T	KEY_F13	
KEY_RIGHT	KEY_KEY_U	KEY_F14	
KEY_DOWN	KEY_KEY_V	KEY_F15	
KEY_SELECT	KEY_KEY_W	KEY_F16	
KEY_PRINT	KEY_KEY_X	KEY_F17	
KEY_EXECUT	KEY_KEY_Y	KEY_F18	

Definición el la línea 11 del archivo **Keycodes.h.**

Enumeración de valores:

ELL_INFORMATION: el nivel del leño Alto, advertencias, errores y textos de información importantes están fuera impresos.

ELL_WARNING: Defecto el nivel del leño, advertencias y errores están fuera impresos.

ELL_ERROR: el nivel del leño Bajo, sólo errores están impresos en el leño.

ELL_NONE: Nada está fuera impreso al leño.

La definición a línea 13 de archivo ILogger.h.

La enumeración de los eventos de entrada del raton.

Enumeración de valores:

EMIE_LMOUSE_PRESSED_DOWN: Si el click izquierdo del ratón es presionado.

EMIE_RMOUSE_PRESSED_DOWN: Si el click derecho del ratón es presionado.

EMIE_MMOUSE_PRESSED_DOWN: Si el click central del ratón es presionado.

EMIE_LMOUSE_LEFT_UP: Si el click izquierdo del ratón es liberado.

EMIE_RMOUSE_LEFT_UP: Si el click derecho del ratón es liberado.

EMIE_MMOUSE_LEFT_UP: Si el click central del ratón es liberado.

EMIE_MOUSE_MOVED: Si el cursor del ratón fue movido.

EMIE_MOUSE_WHEEL: Si la rueda del ratón fue movida. Use el valor de la Rueda en los datos de evento averiguar en qué dirección se desplazo.

EMIE_COUNT: Ningún evento real. Simplemente para la conveniencia de conseguir más de eventos.

La definición en la línea 35 de archivo IEventReceiver.h.

Namespace irr::core;

En este namespace pueden encontrarse las clases básicas para vectores, planos, arreglos, listas, y todo lo relacionado a manejo de coordenadas.

Clases

- **Clases aabbox3d**: alinea una caja en un espacio 3d dimensional
- **Clases array**: re-localiza temporalmente un arreglo (vector del stl) con rasgos adicionales.
- **Clase dimension2d**: Especifica el tamaño de 2 dimensiones
- **Clase irrAllocator**: El Allocator, solo puede usar recipientes por los límites del dll.
- **Clase irrAllocatorFast**: el fastAllocator, sólo es usado en los contenedores dentro de algunas memorias.
- **Clase line2d**: línea 2D entre dos puntos con métodos de intercepción.
- **Clase line3d**: línea 3D entre dos puntos con métodos de intercepción.
- **Clase list**: link double de una lista temporal.
- **Clase matrix4**: matriz 4x4. Principalmente usado como transformaciones de la matriz para cálculos 3d.
- **Clase plane3d**: Clase de plano temporal con algunas intercepciones probando métodos.
- **Clase positio2d**: simple clase para el seguimiento de coordenadas 2d.
- **Clase quaternion**: clase Quaternion
- **Clase rect**: rectángulo temporal.
- **Clase string**: clase muy simple string con algunos rasgos útiles.
- **Clase triangle3d**: clase Triangulo 3d temporal para realizar detecciones de colisión y otras cosas.
- **Clase vectro2d**: clase vector 2d temporal con muchos operadores y métodos.
- **Clase vectro2d**: clase vector 3d temporal con muchos operadores y métodos.

Typedefs (Definiciones de tipos)

typedef aabbox3d< f32 > aabbox3df : Tipo definido para un compilado 3d f32.

typedef aabbox3d< s32 > aabbox3di : Tipo definido para un entero 3d de una caja compilada.

typedef dimension2d< f32 > dimension2df: Tipo definido para un dimensión f32.

typedef dimension2d< s32 > dimension2di: Tipo definido para una dimencion entera.

typedef line2d< f32 > line2df: Tipo defiido para una linea f32.

typedef line2d< s32 > line2di: Tipo definido para una linea entera.

typedef line3d< f32 > line3df: Tipo definido para una linea f32.

typedef line3d< s32 > line3di: Tipo definido para una linea entera.

typedef plane3d< f32 > plane3df: Tipo definido para un plano 3d f32.

typedef plane3d< s32 > plane3di: Tipo definido para un plano 3d entero.

typedef position2d< f32 > position2df: Tipo definido para una posision f32.

typedef position2d< s32 > position2di: Tipo definido para una pocision entera.

typedef string< irr::c8 > stringc: Tipo definido para un character Strings.

typedef string< wchar_t > stringw: Tipo definido para carcateres largos Strings.

typedef triangle3d< f32 > triangle3df: Tipo definido para un triangulo 3d f32.

typedef triangle3d< s32 > triangle3di: Tipo definido para un triangulo 3d entero.

typedef vector2d< f32 > vector2df: Tipo definido de un vector 2d f32.

typedef vector2d< s32 > vector2di: Tipo definido para un vector 2d entero.

typedef vector3d< f32 > vector3df: Tipo definido para un vector 3d f32.

typedef vector3d< s32 >vector3di: Tipo definido para un vector 3d entero.

Enumeraciones para la intersecciones relacionadas de objetos 3d

Enumeración de valores:

ISREL3D_FRONT
ISREL3D_BACK
ISREL3D_PLANAR
ISREL3D_SPANNING
ISREL3D_CLIPPED

Namespace irr::gui;

Este namespace contiene clases útiles para la fácil creación de interfaces graficas de usuario

Clasees

Clase ICursorControl: Interfaz para manipular el cursor del ratón.

Clase IGUIButton: GUI Boton interfaz.

Clase IGUICheckBox: GUI Check box interfaz.

Clase IGUIComboBox: Combobox widget.

Clase IGUIContextMenu: GUI interfaz de menú Contexto.

Clase IGUIEditBox: Línea editar de caja para texto simple.

Clase IGUIElement: Clase base de todos los elementos GUI.

Clase IGUIEnvironment: GUI Environment. Usado como creador y administrador de los elementos GUI.

Clase IGUIFileOpenDialog: Archivo estandar ventana de dialogo.

Clase IGUIFont: interfaz Font.

Clase IGUIImage: elemento GUI mostrar una imagen.

Clase IGUIInOutFader: Element para entradas o salidas de Fader.

Clase IGUIListBox: Caja list box estandar GUI.

Clase IGUIMeshViewer: vista de una maya 3d en un elemento GUI.
Clase IGUIScrollBar: Scroll por defecto barra elemento GUI.

Clase IGUISkin: Un Skin de las vista de los elementos del GUI.

Clase IGUIStaticText: Múltiple o simple label de línea de texto.

Clase IGUITab: Un TAB, En el cual puede agregarse otros elementos GUI.

Clase IGUITabControl: Un control TAB estándar

Clase IGUIToolBar: Se queda en la cima de su padre, es la barra de menú y contiene los botones de la herramienta.

Clase IGUIWindow: El valor predeterminado el elemento de GUI de ventana móvil con la ventana, subtítulo e iconos íntimos.

Enumeración de Skin de colores.

Enumeration de valores:

EGDC_3D_DARK_SHADOW **:** Sombra oscura para una dimensión 3d.

EGDC_3D_SHADOW : color de sombra para una dimensión 3d vista (para aristas enfrentando fuera de la fuente de iluminación).

EGDC_3D_FACE: El color de la cara para los elementos del despliegue tridimensionales y para los fondos del cuadro de diálogo.

EGDC_3D_HIGH_LIGHT **:** Resalte el color para los elementos del despliegue tridimensionales (para bordes que enfrentan la fuente de iluminación.).

EGDC_3D_LIGHT: color para los elementos del despliegue tridimensionales (para bordes que enfrentan la fuente de iluminación.).

EGDC_ACTIVE_BORDER: Activar borde de la ventana

EGDC_ACTIVE_CAPTION**:** Activar título de la barra de texto.

EGDC_APP_WORKSPACE: El color del fondo de interfaz del documento múltiple (MDI) las aplicaciones.

EGDC_BUTTON_TEXT : Texto en un botón.

EGDC_GRAY_TEXT**:** en opaco (deshabilitar) texto.

EGDC_HIGH_LIGHT: Objeto(s) seleccionados en un control.

EGDC_HIGH_LIGHT_TEXT**:** Texto de objeto(s) seleccionados en un control.

EGDC_INACTIVE_BORDER: deshabilitar borde de ventana.

EGDC_INACTIVE_CAPTION: Deshabilitar nombre de la ventana.

EGDC_TOOLTIP**:** Tool tip color.

EGDC_SCROLLBAR**:** Scrollbar opaca (deshabilitada) área.

EGDC_WINDOW**:** fondo de ventana.

EGDC_COUNT:

Definición en línea 39 del archivo **IGUISkin.h**.

Enumeration for default sizes.

Enumeration values

EGDS_SCROLLBAR_SIZE **:** predefina con / la altura de barra de desplazamiento

EGDS_MENU_HEIGHT : la altura de menú

EGDS_WINDOW_BUTTON_WIDTH: la anchura de un botón de la ventana

EGDS_CHECK_BOX_WIDTH : la anchura de un cheque de la casilla de verificación

EGDS_MESSAGE_BOX_WIDTH: la anchura de un messagebox

EGDS_MESSAGE_BOX_HEIGHT: la altura de un messagebox

EGDS_BUTTON_WIDTH : la anchura de un botón predefinido

EGDS_BUTTON_HEIGHT : la altura de un botón predefinido

EGDS_COUNT **:** este valor no se usa, sólo especifica la cantidad de tamaños predefinidos disponible.

Definición en la línea 80 del archivo **IGUISkin.h**.

Enumeración de valores

EGDT_MSG_BOX_OK **:** El texto para el botón de OK en una caja del mensaje.

EGDT_MSG_BOX_CANCEL: : El texto para el No el botón en una caja del mensaje.

EGDT_MSG_BOX_YES: El texto para el Sí el botón en una caja del mensaje.

EGDT_MSG_BOX_NO: El texto para el botón de la Cancelación en una caja del mensaje.

EGDT_COUNT: : este valor no se usa, sólo especifica la cantidad de textos predefinidos disponible

Definición en la línea 112 del archivo **IGUISkin.h**.

EGUIET_ELEMENT: **El tipo desconocido. Use esto al crear sus propios elementos.**

EGUIET_BUTTON: **Un botón (IGUIButton).**

EGUIET_CHECK_BOX: **: Una casilla de verificación (IGUICheckBox).**

EGUIET_COMBO_BOX **: Una caja de la combinación (IGUIComboBox).**

EGUIET_CONTEXT_MENU: **Un menú del contexto (IGUIContextMenu).**

EGUIET_EDIT_BOX **: Una caja editable (IGUIEditBox).**

EGUIET_FILE_OPEN_DIALOG: **Abre un archivo de dialogo (IGUIFileOpenDialog).**

EGUIET_IN_OUT_FADER: UN fader in/out (**IGUIInOutFader**).

EGUIET_IMAGE: Una imagen (**IGUIImage**).

EGUIET_LIST_BOX: Un list Box (**IGUIListBox**).

EGUIET_MESH_VIEWER: Ver una maya (**IGUIMeshViewer**).

EGUIET_MODAL_SCREEN: Una modal de pantalla.

EGUIET_SCROLL_BAR: Un scroll bar (**IGUIScrollBar**).

EGUIET_STATIC_TEXT: Un texto estático (**IGUIStaticText**).

EGUIET_TAB: Un TAB (**IGUITab**).

EGUIET_TAB_CONTROL: Un control TAB.

EGUIET_TOOL_BAR: Un tool bar (**IGUIToolBar**).

EGUIET_WINDOW: Una ventana.

EGUIET_COUNT: No un elemento, la cantidad de elementos en allí.

Definición en línea 15 del archivo **EGUIElementTypes.h**.

Enumeración para todos los eventos que son los sendeables por el sistema de gui.

Enumeración de valores:

***EGET_ELEMENT_FOCUS_LOST*: Un elemento del GUI ha perdido su enfoque.**

***EGET_ELEMENT_HOVERED:* Un elemento del GUI fue cubierto.**

***EGET_ELEMENT_LEFT*: Un elemento GUI fue cubierto por la izquierda.**

***EGET_BUTTON_CLICKED:* Un boton fue presionado.**

***EGET_SCROLL_BAR_CHANGED:* El scrollbar Ha cambiado de posicion.**

***EGET_CHECKBOX_CHANGED*: Un check box cambio su estado de chekeo**

***EGET_LISTBOX_CHANGED*: Un Nuevo item para el ListBox seleccionado.**

***EGET_LISTBOX_SELECTED_AGAIN:* Un item del ListBox seleccionado, El cual ya ha sido seleccionado.**

***EGET_FILE_SELECTED*: Un archive ha sido seleccionado en el "file open dialog".**

EGET_FILE_CHOOSE_DIALOG_CANCELLED: Un **"file open dialog"** ha sido cerrado sin escoger un archivo.

***EGET_MESSAGEBOX_YES*: Sí'** fue presionado en un **messagebox**

***EGET_MESSAGEBOX_NO*: 'No'** fue presionado en un **messagebox**

***EGET_MESSAGEBOX_OK*: 'OK'** fue presionado en un **messagebox**

***EGET_MESSAGEBOX_CANCEL:* 'Cancel'** fue presionado en un **messagebox**

***EGET_EDITBOX_ENTER*:** En un **editbox** fue presionado **'ENTER'.**

***EGET_TAB_CHANGED*:** El **TAB** fue cambiado a **TAB control.**

EGET_MENU_ITEM_SELECTED: Un menú objeto fue seleccionado en un menú (context).

EGET_COMBO_BOX_CHANGED: La selección en un **Combo Box** fue cambiada

Definición en la línea 72 del archivo IEventReceiver.h

EGST_WINDOWS_CLASEIC: Las ventanas predefinidas aparecen en estado clásico.

EGST_WINDOWS_METALLIC: Preferencia **EGST_WINDOWS_CLASEIC**, pero con las ventanas sombreadas en metálicas y los botones.

Definición en línea 29 del archivo **IGUISkin.h**.

Enumeración para los mensaje caja diseño

Enumeración de valores:

***EMBF_OK*:** Flag para el botón **OK**

EMBF_CANCEL: Flag para el botón **CANCELAR**.

***EMBF_YES*:** Flag para el botón **YES.**

***EMBF_NO*:** Flag para el botón **NO.**

EMBF_FORCE_32BIT: Este valor no es usado. Es solo para forzar la enumeración a compilar 32 bit.

Definición en línea 15 del archivo IGUIWindow.h.

Namespace irr::io;

Este namespace provee interfaces de input/output (Entrada/Salida): leyendo y escribiendo archivos, acceso a archivos .ZIP, archivos .XML,...

Clases
Un objeto que puede fabricar en serie y de-serie sus atributos en un objeto de los atributos.

Clase IAttributeExchangingObject: Un objeto que puede fabricar en serie y de-serie este los atributos en los atributos de un objetos

Clase IAttributes: Mantiene una interfaz genérica los atributos y sus valores y los posibilita para fabricarlos en serie.

Clase IFileList: El Filelist lista todos los archivos en un directorio.

clase IFileReadCallBack: Callback clasifican para el archivo lea la abstracción.

Clase IFileSystem: El FileSystem administra archivos, los archiva y provee acceso a ellos.

Clase IIrrXMLReader: Interfaz que proporciona el acceso de fácil lectura a un archivo de XML.

Clase IReadFile: Interfaz permite leer accediendo a un archivo.

Clase IWriteFile: Interfaz permite escribir accediendo a un archivo.

Clase IXMLBase: Clase Empy para usar la clase padre del IrrXMLReader.

Clase IXMLWriter: Interfaz que facilita métodos haciendo más fácil la escritura en archivos de XML.

EEstructurauras

EEstructuraura SAttributeReadWriteOptions: Estructura que sostiene datos que describen las opciones.

Typedefs

typedef unsigned short char16: Define el tipo utf-16.

typedef unsigned long char32: Define el tipo utf-32.

typedef IIrrXMLReader< char,IXMLBase > IrrXMLReader: Un UTF-8 o ASCII carácter xml parser.

typedef IIrrXMLReader< char16,IXMLBase >IrrXMLReaderUTF16: Un UTF-16 xml parser.

typedef IIrrXMLReader< char32,IXMLBase >IrrXMLReaderUTF32: Un UTF-32 xml parser.

typedef IIrrXMLReader< wchar_t,IUnknown >IXMLReader

typedef IIrrXMLReader< c8,IUnknown >IXMLReaderUTF8

Funciones

IrrXMLReader * createIrrXMLReader (IFileReadCallBack *callback): Crear una instancia de un parser UFT-8 o carácter ASCII parse XML.

IrrXMLReader * createIrrXMLReader (FILE *file): Crear una instancia de un parser UFT-8 o carácter ASCII parse XML.

IrrXMLReader * createIrrXMLReader (const char *filename): Crear una instancia de un parser UFT-8 o carácter ASCII parse XML.

IrrXMLReaderUTF16 * createIrrXMLReaderUTF16 (IFileReadCallBack *callback): Crear una instancia de un parser UFT-16 XML.

IrrXMLReaderUTF16 * createIrrXMLReaderUTF16 (FILE *file): Crear una instancia de un parser UFT-16 XML.

IrrXMLReaderUTF16 * createIrrXMLReaderUTF16 (const char *filename): Crear una instancia de un parser UFT-16 XML.

IrrXMLReaderUTF32 * createIrrXMLReaderUTF32 (IFileReadCallBack *callback): Crear una instancia de un parser UFT-32 XML.

IrrXMLReaderUTF32 * createIrrXMLReaderUTF32 (FILE *file): Crear una instancia de un parser UFT-32 XML.

IrrXMLReaderUTF32 * createIrrXMLReaderUTF32 (const char *filename): Crear una instancia de un parser UFT-32 XML.

IReadFile * createLimitReadFile (const c8 *fileName, IReadFile *alreadyOpenedFile, s32 areaSize): function interna, no usar por favor.

IReadFile * createMemoryReadFile (void *memory, s32 size, const c8 *fileName, bool deleteMemoryWhenDropped): function interna, no usar por favor.

IReadFile * createReadFile (const c8 *fileName): function interna, no usar por favor.

IWriteFile * createWriteFile (const c8 *fileName, bool append): function interna, no usar por favor.

Enumeración de valores:

EARWF_FOR_FILE: Serialization/Deserializion se hace para un archivo del XML.

EARWF_FOR_EDITOR: Serialization/Deserializion se hace para una caja de propiedad de editor.

EARWF_USE_RELATIVE_PATHS: Al escribir los nombres de archivo, deben usarse los caminos relativos.

Definición en línea 20 del archive **IAttributeExchangingObject.h.**

EAT_INT
EAT_FLOAT
EAT_STRING
EAT_BOOL
EAT_ENUM
EAT_COLOR
EAT_COLORF
EAT_VECTOR3D
EAT_BINARY
EAT_TEXTURE
EAT_UNKNOWN

Definición en línea 26 del archive **IAttributes.h.**

ETF_ASCII: ASCII, el archivo sin el byte la marca del orden, o no un archivo del texto.

***ETF_UTF8*:** UTF-8 formato.

ETF_UTF16_BE: UTF-16 formato, grande endian.

***ETF_UTF16_LE*:** UTF-16 formato, pequeña endian.

***ETF_UTF32_BE*:** UTF-32 formato, grande endian.

***ETF_UTF32_LE*:** UTF-32 formato, pequeño endian.

Definición en la línea 157 del archivo **irrXML.h.**

***EXN_NONE*:** Ningún nodo del XML. Éste normalmente es el nodo si el usuario aun no a leído nada.

EXN_ELEMENT: Un elemento XML como <foo>.

EXN_ELEMENT_END: Fin de un elemento XML como</foo>.

***EXN_TEXT*:** El texto dentro de un elemento XML: <foo> Este es el texto. <foo>.

EXN_COMMENT: Un comentario XML <!—Estoy hacienda un comentario --> o una definición DTD.

***EXN_CDATA*:** Un sección cdata xml <![CDATA[Este es algún CDATA]]>.

EXN_UNKNOWN: Elemento desconocido.

Definición en la linea 180 del archivo **irrXML.h**.

Namespace irr::scene;

Todo lo necesario para administrar la escena puede encontrarse en este namespace: Cargar mayas, Nodos de escena especiales, Octree y Billboards,...

Clases

Clase IAnimatedMesh: Interfaz para una maya animada..

Clase IAnimatedMeshB3d: Interfaz para usar algunas funciones especiales de la maya B3d.

Clase IAnimatedMeshMD2: Interfaz para usar algunas funciones especiales de la maya MD2.

Clase IAnimatedMeshMS3D: Interfaz para usar algunas funciones especiales de la maya MS3D.

Clase IAnimatedMeshSceneNode: El nodo de la escena capaz de desplegar una malla animada y su sombra.

Clase IAnimatedMeshX: Una por usar algunas funciones especiales de mallas de X.

Clase IAnimationEndCallBack: Callback unen para los eventos contagiosos de acabó las animaciones.

Clase IBillboardSceneNode: Un billboard nodo de escena.

Clase ICameraSceneNode: Nodo de la escena que es un (el controlable) la cámara.

Clase IDummyTransformationSceneNode: El nodo de la escena mudo por agregar las transformaciones adicionales al gráfico de la escena.

Clase ILightSceneNode: Nodo de la escena que es una luz dinámica.

Clase IMesh: Clasifiqua para acceder una malla con los pulidores de la malla múltiples.

Clase IMeshBuffer: EEstructuraura para estar de acuerdo una malla con un solo material.

Clase IMeshCache: La MeshCache carga las mallas y proporcionan una interfaz a ellos.

Clase IMeshLoader: Clase qué puede cargar una malla animada de un archivo.

Clase IMeshManipulator: Una interfaz para fácilmente manipule las mallas.

Clase IMeshSceneNode: Un nodo de la escena que despliega una malla estática.

Clase IMetaTriangleSelector: Una por hacer a los seleccionadores del triángulo múltiples trabajar como un seleccionador grande.

Clase IParticleAffector: Un affector de la particula modifica las partículas.

Clase IParticleEmitter: Un emisor de la partícula por usar con los sistemas de la partícula.

Clase IParticleSystemSceneNode: Un partícula sistema escena nodo por crear la nieve, el fuego, el exlosions, niebla...

Clase IQ3LevelMesh: Interfaz para una maya el cual puede cargar directamente de un archivo Quake3 .BSP.

Clase ISceneCollisionManager: El administrador de Colisión de Escena mantiene los métodos realizando la colisión prueba y metiéndose con los nodos de la escena.

Clase ISceneManager: El administrador de Escena maneja nodos de la escena, recources de la malla, cámaras y todo el otro material.

Clase ISceneNode: Interfaz de nodo de escena.

Clase ISceneNodeAnimator: Anima un nodo de la escena. Puede animar posición, la rotación, el material, y así sucesivamente.

Clase ISceneNodeAnimatorCollisionResponse: El animador de nodo de escena especial por hacer descubrimiento de la colisión automático y contestación.

Clase ISceneNodeAnimatorFactory: Interfaz que lo hace posible al crea a animador dinámico de nodos de escena.
Clase ISceneNodeFactory: Interfaz que lo hace posible al dynamicly crea los nodos de la escena.

Clase ISceneUserDataSerializer: Lee y escribe los datos del usuario a y los archivos irr.

Clase IShadowVolumeSceneNode: El nodo de la escena por dar un volumen de la sombra en un pulidor del estarcido.

Clase ITerrainSceneNode: Un nodo de la escena por desplegar terreno que usa el mip del geo traza el algoritmo.

Clase ITextSceneNode: Un nodo de la escena por desplegar 2d texto a una posición en tres espacio dimensional.

Clase ITriangleSelector: Una para devolver los triángulos con las propiedades específicas.

EEstructuraura SAnimatedMesh: La aplicación simple de la interfaz de IAnimatedMesh.

EEstructuraura SMesh: Implementacion simple de una interfaz IMesh.

EEstructuraura SMeshBuffer: implementacion simple del interfaz IMeshBuffer con vertices S3DVertex.

EEstructuraura SMeshBufferLightMap: Simple aplicación de la interfaz de IMeshBuffer con vertices de S3DVertex2TCoords.

EEstructuraura SMeshBufferTangents: Simple implementation of the IMeshBuffer interface with S3DVertexTangents vertices.

EEstructuraura SParticle: EEstructuraura por sostener los datos de la partícula.

EEstructuraura SViewFrustrum: Define el SViewFrustrum. Que el espacio visto por la cámara.

Variables

const char *const COLLADA_CREATE_SCENE_INSTANCES = "COLLADA_CreateSceneInstances": Nombre especifico de los parametros de la COLLADA en el modo cargado de maya.

const char *const CSM_TEXTURE_PATH = "CSM_TexturePath": El nombre del parámetro por cambiar el camino de la textura del cargador del csm incorporado.
const char *const DMF_ALPHA_CHANNEL_REF = "DMF_AlphaRef": El nombre del parámetro por poner valor de la referencia de alfa en los materiales transparentes.
const char *const DMF_FLIP_ALPHA_TEXTURES = "DMF_FlipAlpha": El nombre del **parámetro para escoge arrojar o no los archivos del tga. const char *const DMF_TEXTURE_PATH = "DMF_TexturePath":** El nombre del parámetro por cambiar el camino de la textura del cargador de DMF incorporado.

const char *const DMF_USE_MATERIALS_DIRS = "DMF_MaterialsDir": Los nombres de los arametros para preserver el directorio de texturas DMF con el cargado built-in DMF.

const char *const IRR_SCENE_MANAGER_IS_EDITOR = "IRR_Editor": Flag se coloca como parámetros donde el administrador de escena usa el editor.

const char *const LMTS_TEXTURE_PATH = "LMTS_TexturePath": Los nombres de los parámetros para cambiar el path de texturas de el cargado built-in lmts.

const char *const MY3D_TEXTURE_PATH = "MY3D_TexturePath": Los nombres de los parametros para cambiar el path de textura al cargar el built-in my3d.

const char *const ParticleAffectorTypeNames []:Los nombres del compilado en el particulas **affectors**.

const char *const ParticleEmitterTypeNames []:Los nombres para construyó los emisores de partícula.

Enumeration values:

EAMT_UNKNOWN: Tipo de maya animada desconocida.

EAMT_MD2: Archivo modelado Quake 2 MD2.

EAMT_MS3D: Archivo animado con esqueleto Milkshape 3d.

EAMT_OBJ: Maya .obj modelado no animado .

EAMT_BSP: Quake 3 .bsp Mapa, no animado.

EAMT_3DS: Archivo 3D Studio .3ds.

EAMT_X: Microsoft Direct3D .x-file. Puede contener mayas estaticas y esqueleticas, este es el mas estandar y el formato major soportado por el Irrlicht Engine.

EAMT_MY3D: Maya .My3D, Este formato fue creado por Zhuck Dimitry.

EAMT_LMTS: Archivo pulsar LMTools (.lmts). Cargador para el Irrlicht esto fue escrito por Jonas Petersen.

EAMT_CSM: Categoría Almacén Archivo .CSM. El cargador para Irrlicht fue creado por Saurav Mohapatra.

EAMTS_OCT: Archivo .OCT por Paul Nette's FSRad o de Murphy McCauley's Blender Exportador .OCT . El formato de archive :OCT contiene geometria 3D y luces y puede ser cargado directamente por el Irrlicht
EAMTS_B3D: Blitz Basico Archivo .B3D, Formato de archivo por Mark Sibly.

Tipo de animaciones estandar MD2

Numeración de valores:

EMAT_STAND
EMAT_RUN
EMAT_ATTACK
EMAT_PAIN_A
EMAT_PAIN_B
EMAT_PAIN_C
EMAT_JUMP
EMAT_FLIP
EMAT_SALUTE

EMAT_FALLBACK
EMAT_WAVE
EMAT_POINT
EMAT_CROUCH_STAND
EMAT_CROUCH_WALK
EMAT_CROUCH_ATTACK
EMAT_CROUCH_PAIN
EMAT_CROUCH_DEATH
EMAT_DEATH_FALLBACK
EMAT_DEATH_FALLFORWARD
EMAT_DEATH_FALLBACKSLOW
EMAT_BOOM
EMAT_COUNT

No una animación, pero cantidad de tipos de la animación incorporados

Definición en línea 16 del archivo IAnimatedMeshMD2.h.

Una enumeración para todos los tipos animadores de nodos de escena internos.

Enumeración de valores:

ESNAT_FLY_CIRCLE: Circulo de vuelo en un nodo de escena animado.

ESNAT_FLY_STRAIGHT: Animador volar recto de nodo de escena.

ESNAT_FOLLOW_SPLINE: Nodo de escena de Follow Spline.

ESNAT_ROTATION: Nodo de escena animador de rotación.

ESNAT_TEXTURE: Nodo de escena animador de texturas.

ESNAT_DELETION: Deletion scene node animator.

ESNAT_COLLISION_RESPONSE: Nodo animador de respuesta de colisión.

ESNAT_COUNT: Cantidad de figura en animador de nodo de escena.

ESNAT_UNKNOWN: Nodo de escena animado desconocido.

ESNAT_FORCE_32_BIT: Este valor no es usado. Es solo para forzar la enumeración a compilar 32 bit.

Definición en línea 13 del archivo ESceneNodeAnimatorTypes.h.

Una enumeración para todos los tipos de nodos de la escena incorporados.

Enumeración de valores:

ESNT_CUBE: Nodo de escena un simple cubo

ESNT_SPHERE: Nodo de escena esfera.

ESNT_TEXT: Nodo de escena texto.

ESNT_WATER_SURFACE: Nodo de escena Superficie de agua.

ESNT_TERRAIN: Nodo de escena Terreno.

ESNT_SKY_BOX: Nodo de escena Caja cielo.

ESNT_SHADOW_VOLUME: Nodo de escena volumen de sombra.

ESNT_OCT_TREE: Nodo de escena OctTree.

ESNT_MESH: Nodo de escena maya.

ESNT_LIGHT: Nodo de escena Luz.

ESNT_EMPTY: Nodo de escena vació.

ESNT_DUMMY_TRANSFORMATION: Nodo de escena transformacion Dummy.

ESNT_CAMERA: Nodo de escena Cámara.

ESNT_CAMERA_MAYA: Nodo de escena Maya Cámara..

ESNT_CAMERA_FPS: Estilo cámara FPS.

ESNT_BILLBOARD: Nodo de escena Billboard.

ESNT_ANIMATED_MESH: Nodo de escena Maya animada.

ESNT_PARTICLE_SYSTEM: Nodo de escena de partículas del sistema.

ESNT_COUNT: La cantidad de figura en los Nodos de la Escena.

ESNT_UNKNOWN: Nodo de escena desconocido.

ESNT_FORCE_32_BIT: Este valor no es usado. Es solo para forzar la enumeración a compilar 32 bit.

Definición en línea 13 del archivo **ESceneNodeTypes.h**.

Namespace irr::video;

El namespace video contiene las clases para acceder al controlador de video. Todos las vistas 2D y 3D se renderizan aquí.

Clases

Clase IGPUProgrammingServices: Interfaz que lo hace posible crear y usar programas que corren en el GPU.

Clase IImage: Interfaz para software de datos de imagen

Clase IImageLoader: Clase qué puede crear una imagen de un archivo.

Clase IImageWriter: interfaz para escribir los datos de imagen de software.

Clase IMaterialRenderer: Interfaz para renderizar materials. Puede ser usado para extender al Engine conm nuevosa materials.

Clase IMaterialRendererServices: Interfaz que mantiene algunos métodos cambiando los estados avanzados, interiores de un IVideoDriver.

Clase IShaderConstantSetCallBack: Interfaz que le hace posibles constantes puestas para los programas del gpu cada marco.

Clase ITexture: Interfaz para un Video Driver dependiendo de la textura.

Clase IVideoDriver: Interfaz del con el cual pueden realizar funciones gfx 2d y 3d.

Clase IVideoModeList: Una lista de modos videos todo disponibles.

Estructura S3DVertex: vértice normal usado por el Irrlicht engine.

Estructura S3DVertex2TCoords: El vértice con dos coordenadas de la textura.

Estructura S3DVertexTangents: El vértice con una tangente y vector del binormal.

Clase SColor: Clase representando un 32 pedazo color de ARGB.

Clase SColorf: Clase representando un color con cuatro flotadores.

Estructura SExposedVideoData: Estructura por sostener datos que describen chófer y sistema operativo los datos especificos.

Estructura SLight: Estructura para sostener datos que describen una luz del punto dinámica.

Estructura SMaterial: Estructura por sostener los parámetros para un renderizar un material.

Funciones

u32 A1R5G5B5toA8R8G8B8 (u16 color): Devuelve color A8R8G8B8 de color A1R5G5B5 .

s16 A1R5G5B5toR5G6B5 (s16 color): Devuelve color R5G6B5 de color A1R5G5B5.

u16 A8R8G8B8toA1R5G5B5 (u32 color): Convierte un color (A8R8G8B8) 32 bit a un color A1R5G5B5 16 bit.

s32 getAlpha (s16 color): Regresa el componente alpha del color A1R5G5B5.

s32 getBlue (s16 color)

s32 getGreen (s16 color)

s32 getLuminance (s16 color): Devuelve iluminacion de un color A1R5G5B5 de16 bit.

s32 getRed (s16 color)

E_TEXTURE_CREATION_FLAG getTextureFormatFromFlags (u32 flags)

s16 R5G6B5toA1R5G5B5 (s16 color): Devulve Color A1R5G5B5 de color R5G6B5.

s32 R5G6B5toA8R8G8B8 (s16 color): Devulve Color A8R8G8B8 de color R5G6B5.

s16 RGB16 (s32 r, s32 g, s32 b): Crea un color A1R5G5B5 de 16 bit.

s16 RGBA16 (s32 r, s32 g, s32 b, s32 a): Crea un color A1R5G5B5 de 16 bit.

s16 X8R8G8B8toA1R5G5B5 (s32 color): Convierte un color (X8R8G8B8) 32 bit a un color A1R5G5B5 16 bit.

Variables

const char *const LightTypeNames []:Los nombres para los tipos de luz.

const s32 MATERIAL_MAX_TEXTURES = 4: El número máximo de textura que un SMaterial puede tener.

const char *const PIXEL_SHADER_TYPE_NAMES []:Los nombres para todo los pixel shader tecleados, cada entrada corresponde a una entrada E_PIXEL_SHADER_TYPE.

const char *const VERTEX_SHADER_TYPE_NAMES []:Los nombres para todo los vértices shader teclean, cada entrada corresponde a una entrada E_VERTEX_SHADER_TYPE.

Una enumeracion de todos los tipos de drivers soportados por el Irrlicht Engine.

Enumeración de valores:

EDT_NULL: El dispositivo NULL, útil para las aplicaciones para ejecutar el engine sin visualisation. El dispositivo NULL puede cargar las texturas, pero no las renderiza, ni las muestra gráficamente.

EDT_SOFTWARE: El Software renderer de Irrlicht Engine, corre en todas las plataformas, con cada hardware. Sólo debe usarse para 2d gráficos, pero también puede realizar algunas 3d funciones primitivas. Estas 3d funciones del dibujo son bastante rápidas, pero muy inexactas, y no apoya incluso sujetando en 3D modo.

EDT_SOFTWARE2: El Software Renderer Apfelbaum, un software renderer alternativo para Irrlicht. Básicamente puede describirse como el Irrlicht Software renderer con mejoras. Él renderiza la geometría 3D perfectamente: Puede realizar recorte 3d correctos, perspectiva el textura trazando correcto, perspectiva la cartografía colorida correcta, y da al pixel sub texel correcto, sub-primitivas correctas. Además, hace texel del bilineales que se filtra y apoya más materiales que el driver de EDT_SOFTWARE. Este renderizador ha sido completamente escrito por Thomas Alten, muchas gracias para esta gran contribución.

EDT_DIRECT3D8: El dispositivo Direct3D 8, sólo disponible en las plataformas de Win32 incluso Win95, Win98, WinNT, Win2K, WinXP. Realiza que el hardware aceleró dando de 3D y 2D primitivas.

EDT_DIRECT3D9: El dispositivo Direct3D 9, sólo disponible en las plataformas de Win32 incluso Win95, Win98, WinNT, Win2K, WinXP. Realiza que el hardware aceleró dando de 3D y 2D primitivas.

EDT_OPENGL: El dispositivo OpenGL, disponible en todas las plataformas de Win32 y en Linux. Realiza que el hardware aceleró dando de 3D y 2D primitives.

Definición en línea 13 del archivo **EDriverTypes.h.**

Enumeración de los diferentes tipos de luces

Enumeración de valores:

ELT_POINT: Puntod de luz, tiene una posición en el espacio y radia la luz en todas las direcciones.

ELT_DIRECTIONAL: Direccion de la luz, viniendo de una dirección de una distancia infinita.

Definición en linea 16 del archivo SLight.h.

Materiales Flag

Enumeración de valores:

EMF_WIREFRAME: ¿Dibuje como wireframe los triángulos llenos? El valor predeterminado: falso.

EMF_POINTCLOUD: ¿Dibuje como nube del punto o los triángulos llenos? El valor predeterminado: falso.

EMF_GOURAUD_SHADING: ¿Piso o Gouraud obscureciendo? El valor predeterminado: verdadero.

EMF_LIGHTING: ¿Este material se encenderá? El valor predeterminado: verdadero.

EMF_ZBUFFER: ¿El ZBuffer se habilita? El valor predeterminado: verdadero.

EMF_ZWRITE_ENABLE: Puede escribirse al zbuffer o puede serse él el readonly. El valor predeterminado: arregle que Esta bandera se ignora, si el tipo material es un tipo transparente.

EMF_BACK_FACE_CULLING: ¿Se habilitan los backfaceculling? El valor predeterminado: verdadero.

EMF_BILINEAR_FILTER: ¿El bilinear está filtrándose habilitado? El valor predeterminado: verdadero.

EMF_TRILINEAR_FILTER: ¿El trilinear está filtrándose habilitado? El valor predeterminado: falso Si el los trilinear se filtran la bandera se habilita, el bilinear que se filtra la bandera se ignora.

EMF_ANISOTROPIC_FILTER: ¿El anisotropic está filtrándose? El valor predeterminado: falso En Irrlicht usted puede usar textura del anisotropic que se filtra junto con bilinear o textura del trilinear que se filtran para mejorar dando los resultados. Primitives parecerá menos borroso con esta bandera encendida.

EMF_FOG_ENABLE: ¿Se habilitan las nieblas? El valor predeterminado: falso.

EMF_NORMALIZE_NORMALS: Normaliza que normals.You puede habilitar esto si usted necesita descascarar a un modelo encendido dinámico. Normalmente, sus normal se descascararán demasiado entonces y anochecerá. Si usted habilita que los EMF_NORMALIZE_NORMALS marcan, los normal se normalizarán de nuevo, y el modelo parecerá tan luminoso como debe.

EMF_MATERIAL_FLAG_COUNT: Este no es un flag, pero evalúa un valor indicando cuando hay muchos Flag.

Definición en la línea 188 del archivo SMaterial.h

Resumido y fácil para usar la tubería del function/programmable fija los modos materiales.

Enumeration values:

EMT_SOLID:
El material del sólido normal. Sólo primero la textura se usa que se supone que es el material difuso.

EMT_SOLID_2_LAYER:
El material sólido con 2 capas de la textura. El segundo está mezclado hacia el usar el valor del alfa de los colores del vértice primero. Este material no se lleva a cabo actualmente en OpenGL, pero funciona con DirectX.

EMT_LIGHTMAP:
El tipo material con la técnica del lightmap normal: Debe haber 2 texturas: La primera capa de la textura es un mapa difuso, el segundo es un mapa ligero. La luz del vértice se ignora.

EMT_LIGHTMAP_ADD:
El tipo material con la técnica del lightmap como EMT_LIGHTMAP, pero no se modulan lightmap y la textura difusa, pero agregó en cambio.

EMT_LIGHTMAP_M2:
El tipo material con la técnica del lightmap normal: Debe haber 2 texturas: La primera capa de la textura es un mapa difuso, el segundo es un mapa ligero. La luz del vértice se ignora. Los colores de la textura son eficazmente los multiplyied por 2 por aclarar. como conocido en DirectX como D3DTOP_MODULATE2X.

EMT_LIGHTMAP_M4 :
El tipo material con la técnica del lightmap normal: Debe haber 2 texturas: La primera capa de la textura es un mapa difuso, el segundo es un mapa ligero. La luz del vértice se ignora. Los colores de la textura son eficazmente los multiplyied por 4 por aclarar. como conocido en DirectX como D3DTOP_MODULATE4X.

EMT_LIGHTMAP_LIGHTING :
Guste EMT_LIGHTMAP, pero también los apoyos la iluminación dinámica.

EMT_LIGHTMAP_LIGHTING_M2 :
Guste EMT_LIGHTMAP_M2, pero también los apoyos la iluminación dinámica.

EMT_LIGHTMAP_LIGHTING_M4:
Guste EMT_LIGHTMAP_4, pero también los apoyos la iluminación dinámica.

EMT_DETAIL_MAP :
El detalle trazó el material. La primera textura es el mapa de color difuso, el segundo se agrega a esto y normalmente desplegó con un valor de la balanza más grande para que agregue más detalle. El mapa de detalle se agrega al mapa difuso que usa ADD_SIGNED, para que sea posible agregar y los substract coloran del mapa difuso. Por ejemplo un valor de (127,127,127) no cambiará la apariencia del mapa difuso en absoluto. A menudo usado por el terreno dar.

EMT_SPHERE_MAP:
Hace el material parézcaselo era la reflexión el ambiente alrededor de él. Para hacer este posible, una textura llamó 'el mapa de la esfera' se usa que debe ponerse como Texture1.

EMT_REFLECTION_2_LAYER:
Un material reflejando con un non adicionales optativos que reflejan la capa de la textura. El mapa de la reflexión debe ponerse como Textura 1.

EMT_TRANSPARENT_ADD_COLOR :
Un material transparente. Sólo la primera textura se usa. El nuevo color es calculado agregando el color de la fuente y el color del dest simplemente. Esto significa si por ejemplo una cartelera que usa una textura con el fondo negro y un círculo rojo en él es arrastrado con este material, el resultado es eso que sólo el círculo rojo se dibujará un poco transparente, y todo que era el negro es 100% transparente y no visible. Este tipo material es útil para por ejemplo los efectos de la partícula.

EMT_TRANSPARENT_ALPHA_CHANNEL:
Las hechuras el material transparente basado en el cauce de alfa de textura. El color final está juntos mezclado del color del destino y la textura colora, mientras usando el valor de cauce de alfa como el factor de la mezcla. Sólo primero la textura se usa. Si usted está usando este material con las texturas pequeñas, es una idea buena para cargar la textura en la modalidad de 32 bit (el video::IVideoDriver::setTextureCreationFlag ()). También, una ref del alfa se usa que puede manipularse usando SMaterial::MaterialTypeParam. Si el juego a 0, la ref del alfa consigue su valor predefinido que es 0.5f y qué medios que los pixeles con un alfa valor >127 se escribirá, otros no. En otras, simples palabras: este valor controla cómo afilado los bordes se vueltos al ir de un transparente a una mancha sólida en la textura.

EMT_TRANSPARENT_ALPHA_CHANNEL_REF:
Las hechuras el material transparente basado en el cauce de alfa de textura. Si el valor de cauce de alfa es mayor que 127, un pixel se escribe al blanco, por otra parte no. Este material no usa alfa que mezcla y es mucho más rápido que EMT_TRANSPARENT_ALPHA_CHANNEL. Es ideal para dibujar el material como el leafes de plantas, porque las fronteras no están borrosas pero afiladas. Sólo primero la textura se usa. Si usted está usando este material con las texturas pequeñas y 3d objeto, es una idea buena para cargar la textura en la modalidad de 32 bit (el video::IVideoDriver::setTextureCreationFlag ()).

EMT_TRANSPARENT_VERTEX_ALPHA:
Las hechuras el material transparente basado en el valor de alfa de vértice.

EMT_TRANSPARENT_REFLECTION_2_LAYER:
Un material reflejando transparente con un non adicionales optativos que reflejan la capa de la textura. El mapa de la reflexión debe ponerse como Textura 1. La transparencia depende del valor del alfa en los colores del vértice. Una textura que no reflejará puede ponerse Textura 2 al als. Por favor la nota que este tipo material no es actualmente 100% llevó a cabo en OpenGL. Funciona en Direct3D.

EMT_NORMAL_MAP_SOLID:
Un renderer del mapa normales sólidos. Primero la textura es el mapa colorido, el segundo debe ser el mapa normal. La nota que usted sólo debe usar este material al dibujar geometría que consiste en vertices de tipo S3DVertexTangents (EVT_TANGENTS). Usted puede convertir cualquier malla en este formato que usa IMeshManipulator::createMeshWithTangents () (Vea la Guía didáctica de SpecialFX2). Este shader corre atrás en el vértice shader 1.1 y shader del pixel 1.1 hardware capaz y caídas en una función fija encendió el material si este hardware no está disponible. Sólo dos luces son soportadas por este shader, si hay más, el más cercano dos son escogidos. Actualmente, este shader sólo se lleva a cabo para el D3D8 y renderers de D3D9.

EMT_NORMAL_MAP_TRANSPARENT_ADD_COLOR:
Un renderer del mapa normales transparentes. Primero la textura es el mapa colorido, el segundo debe ser el mapa normal. La nota que usted sólo debe usar este material al dibujar geometría que consiste en vertices de tipo S3DVertexTangents (EVT_TANGENTS). Usted puede convertir cualquier malla en este formato que usa IMeshManipulator::createMeshWithTangents () (Vea la Guía didáctica de SpecialFX2). Este shader corre atrás en el vértice shader 1.1 y shader del pixel 1.1 hardware capaz y caídas en una función fija encendió el material si este hardware no está disponible. Sólo dos luces son soportadas por este shader, si hay más, el más cercano dos son escogidos. Actualmente, este shader sólo se lleva a cabo para el D3D8 y renderers de D3D9.

EMT_NORMAL_MAP_TRANSPARENT_VERTEX_ALPHA:
Un transparente (basado en el valor de alfa de vértice) el renderer del mapa normal. Primero la textura es el mapa colorido, el segundo debe ser el mapa normal. La nota que usted sólo debe usar este material al dibujar geometría que consiste en vertices de tipo S3DVertexTangents (EVT_TANGENTS). Usted puede convertir cualquier malla en este formato que usa IMeshManipulator::createMeshWithTangents () (Vea la Guía didáctica de SpecialFX2). Este shader corre atrás en el vértice shader 1.1 y shader del pixel 1.1 hardware capaz y caídas en una función fija encendió el material si este hardware no está disponible. Sólo dos luces son soportadas por este shader, si hay más, el más cercano dos son escogidos. Actualmente, este shader sólo se lleva a cabo para el D3D8 y renderers de D3D9.

EMT_PARALLAX_MAP_SOLID:
Simplemente como EMT_NORMAL_MAP_SOLID, pero paralaje de los usos también trazando que parece mucho más realista. Esto sólo trabaja cuando el hardware apoya el vértice por lo menos shader 1.1 y pixel shader 1.4. Primero la textura es el mapa colorido, el segundo debe ser el mapa normal. La textura del mapa normal debe contener el valor de altura en el componente del alfa. El IVideoDriver::makeNormalMapTexture () el método escribe este automaticly de valor al crear los mapas normales de un heightmap al usar una 32 textura del pedazo. La balanza de altura del material (afectando el bumpiness) está controlándose por el

miembro de SMaterial::MaterialTypeParam. Si el juego ponga a cero, el valor predefinido (0.02f) se aplicará. Por otra parte el valor puesto en SMaterial::MaterialTypeParam se toma. Este valor depende en con que balanza que la textura se traza en el material. Los valores demasiado altos o bajos de MaterialTypeParam pueden producir los artefactos extraños.

EMT_PARALLAX_MAP_TRANSPARENT_ADD_COLOR:
Un material sólo como EMT_PARALLAX_MAP_SOLID, pero es transparente, mientras usando EMT_TRANSPARENT_ADD_COLOR como el material de la base.

EMT_PARALLAX_MAP_TRANSPARENT_VERTEX_ALPHA :
Un material sólo como EMT_PARALLAX_MAP_SOLID, pero es transparente, mientras usando EMT_TRANSPARENT_VERTEX_ALPHA como el material de la base.

EMT_FORCE_32BIT:
Este valor no se usa. Sólo obliga a esta enumeración compilar en 32 momento.

La definición en la línea 16 de archivo SMaterial.h.

Compile la enumeración designado para el addHighLevelShaderMaterial () el método.

Los valores de la enumeración:

EPST_PS_1_1

EPST_PS_1_2

EPST_PS_1_3

EPST_PS_1_4

EPST_PS_2_0

EPST_PS_2_a

EPST_PS_2_b

EPST_PS_3_0

EPST_COUNT

Éste no es un tipo, pero un valor que indica cuánto teclea que hay.

La definición a línea 46 de archivo IGPUProgrammingServices.h

La enumeración marca diciéndole al driver de video en que deben crearse las texturas del formato.

Los valores de la enumeración:

ETCF_ALWAYS_16_BIT:

Las fuerzas el chófer para siempre crear 16 texturas del pedazo, indepenent de que el formato el archivo en el disco tiene. Cuando escogiendo esto usted pueden soltar algún detalle de color, pero gana mucha velocidad y memoria. pueden transferirse 16 texturas del pedazo dos veces tan rápido como 32 texturas del pedazo y sólo pueden usarse la mitad del espacio en la memoria. Al usar esta bandera, no hace el sence para usar las banderas al mismo tiempo ETCF_ALWAYS_32_BIT, ETCF_OPTIMIZED_FOR_QUALITY, o ETCF_OPTIMIZED_FOR_SPEED.

ETCF_ALWAYS_32_BIT:

Las fuerzas el chófer para siempre crear 32 texturas del pedazo, indepenent de que el formato el archivo en el disco tiene. Por favor note que algunos chóferes (como el dispositivo del software) ignorará esto, porque ellos sólo pueden crear y usar 16 texturas del pedazo. Al usar esta bandera, no hace el sence para usar las banderas al mismo tiempo ETCF_ALWAYS_16_BIT, ETCF_OPTIMIZED_FOR_QUALITY, o ETCF_OPTIMIZED_FOR_SPEED.

ETCF_OPTIMIZED_FOR_QUALITY:

Permite al chófer decidir en que el formato los texutures se crean e intentan hacer las texturas parecer tan bueno como posible. Normalmente escoge el formato en que la textura se guardó en el disco simplemente. Al usar esta bandera, no hace el sence para usar las banderas al mismo tiempo ETCF_ALWAYS_16_BIT, ETCF_ALWAYS_32_BIT, o ETCF_OPTIMIZED_FOR_SPEED.

ETCF_OPTIMIZED_FOR_SPEED:

Permite al chófer decidir en que el formato los texutures se crean e intentan crearlos aumentando al máximo dé la velocidad. Al usar esta bandera, no hace al sence usar las banderas ETCF_ALWAYS_16_BIT, ETCF_ALWAYS_32_BIT, o ETCF_OPTIMIZED_FOR_QUALITY, al mismo tiempo.

ETCF_CREATE_MIP_MAPS: Automaticly crea los mip trazan nivela para las texturas.

ETCF_FORCE_32_BIT_DO_NOT_USE: Esta bandera nunca se usa, sólo fuerza al recopilador a compile éstos los valores de la enumeración a 32 pedazo.

La definición en la línea 22 de archivo ITexture.h.

la enumeración para los estados de transformación de geometría

Los valores de la enumeración:

ETS_VIEW: Vea la transformación.

ETS_WORLD: La transformación mundial.

ETS_PROJECTION : La transformación de la proyección.

ETS_COUNT: No usado.

La definición de la línea 101 de archivo IVideoDriver.h.

Compile la enumeración designado para el addHighLevelShaderMaterial () el método.

Los valores de la enumeración:

EVST_VS_1_1

EVST_VS_2_0

EVST_VS_2_a

EVST_VS_3_0

EVST_COUNT

Éste no es un tipo, pero un valor que indica cuánto teclea que hay.

La definición a línea 26 de archivo IGPUProgrammingServices.h.

La enumeración para todo el vértice teclea hay.

Los valores de la enumeración:

EVT_STANDARD : Tipo del vértice normal usado por el artefacto de Irrlicht, el video::S3DVertex.

EVT_2TCOORDS: El vértice con dos la textura coordina, video::S3DVertex2TCoords. Normalmente usó para la geometría con lightmaps u otros materiales especiales.

EVT_TANGENTS: El vértice con una tangente y vector del binormal, video::S3DVertexTangents. Normalmente usó para el espacio tangente la cartografía normal.

La definición en la línea 18 de archivo S3DVertex.h

la enumeración por preguntar rasgos del chófer video.

Los valores de la enumeración:

EVDF_RENDER_TO_TARGET: ¿El Driver puede dar a una superficie?

EVDF_BILINEAR_FILTER: ¿ Driver puede aplicar un filtro bilineal?

EVDF_HARDWARE_TL: ¿Es los hardeware transforman y encendiendo soportado?

EVDF_MIP_MAP: ¿El Driver puede ocuparse de mapas del mip?

EVDF_STENCIL_BUFFER: ¿Los stencilbuffers se encienden y el apoyo del dispositivo estarce los pulidores?

EVDF_VERTEX_SHADER_1_1: ¿El Vértice es un Shader 1.1 soportado?

EVDF_VERTEX_SHADER_2_0: ¿El Vértice es un Shader 2.0 soportado?

EVDF_VERTEX_SHADER_3_0: ¿El Vértice es Shader 3.0 soportado?

EVDF_PIXEL_SHADER_1_1: ¿El Pixel es Shader 1.1 soportado?

EVDF_PIXEL_SHADER_1_2: ¿El Pixel es Shader 1.2 soportado?

EVDF_PIXEL_SHADER_1_3: ¿El Pixel es Shader 1.3 soportado?

EVDF_PIXEL_SHADER_1_4 : ¿El Pixel es Shader 1.4 soportado?

EVDF_PIXEL_SHADER_2_0: ¿El Pixel es Shader 2.0 soportado?

EVDF_PIXEL_SHADER_3_0: ¿El Pixel es Shader 3.0 soportado?

EVDF_ARB_VERTEX_PROGRAM_1: ¿Es el vértice de ARB programa v1.0 **apoyó?**

EVDF_ARB_FRAGMENT_PROGRAM_1: ¿Es ARB fragmentan programa v1.0 **apoyó?**

EVDF_ARB_GLSL: ¿GLSL se apoya?

EVDF_HLSL: ¿HLSL se apoya?

La definición en la línea 43 de archivo IVideoDriver.h.

Un enunciado para el formato colorido de texturas usado por el Artefacto de Irrlicht.

Un formato colorido especifica cómo la información colorida se guarda. El Artefacto de Irrlicht usa el formato principalmente ECF_A1R5G5B5.

Los valores de la enumeración:

ECF_A1R5G5B5 : 16 pedazo formato colorido usado por el chófer del software, y así prefirió por todo el otro artefacto del irrlicht los chóferes videos. Hay 5 bits para cada componente colorido, y un solo pedazo se deja para la información del alfa.

ECF_R5G6B5 : El 16 pedazo formato colorido normal.

ECF_R8G8B8: 24 color del pedazo, ningún cauce del alfa, pero 8 pedazo para la red, verde y azul.

ECF_A8R8G8B8 : Predefina 32 pedazo formato colorido. se usan 8 bits para cada componente: rojo, verde, azul y alfa.

La definición en la linea 22 de archivo IImage.h.

Irrlicht Engine lista de Clases

Aquí son las clases, estructuras, uniones e interfaces con las descripciones breves:

irr::core::aabbox3d< T >	Alinea una caja en un espacio 3d dimencional
irr::core::array< T, TAlloc >	Ego que reasigna la serie de la plantilla (como el vector del stl) con los rasgos adicionales
irr::core::dimension2d< T >	Especifica un tamaño en 2 dimenciones
irr::scene::IAnimatedMesh	Interfaz para animar una maya
irr::scene::IAnimatedMeshB3d	Interfaz para usar algunas funciones especiales de mayas B3d.
irr::scene::IAnimatedMeshMD2	Interfaz para usar algunas funciones especiales de mayas MD2
irr::scene::IAnimatedMeshMS3D	Interfaz para usar algunas funciones especiales de mayas MS3D
irr::scene::IAnimatedMeshSceneNode	Nodo de escena para visualizar las mayas animadas y sus sombras
irr::scene::IAnimatedMeshX	Interfaz para usar algunas funciones especiales de mayas X
irr::scene::IAuimationEndCallBack	Callback interfaz para eventos de final de la animacion
irr::io::IAttributeExchangingObject	Un objeto que puede fabricar en serie y de-serialsus atributos en un objeto de los atributos
irr::io::IAttributes	Mantiene una interfaz genérica los atributos y sus valores y los possiblity para fabricarlos en serie
irr::scene::IBillboardSceneNode	Nodo de escena Billboard
irr::scene::ICameraSceneNode	Nodo de la escena que es un (controlable) la cámara
irr::gui::ICursorControl	Interfaz para manipular el cursor del raton
irr::scene::IDummyTransformationSceneNode	El nodo de la escena mudo por agregar las transformaciones adicionales al gráfico de la escena
irr::IEventReceiver	Interfaz de un objeto el cual

	puede recibir eventos
irr::io::IFileList	El Filelist lista todos los archivos de un directorio
irr::io::IFileReadCallBack	Callback clasifican para el archivo lea la abstracción
irr::io::IFileSystem	El FileSystem maneja archivos y archivos y proporciona el acceso a ellos
irr::video::IGPUProgrammingServices	Interfaz que lo hace posible crear y usar programas que corren en el GPU
irr::gui::IGUIButton	Interfaz boton GUI
irr::gui::IGUICheckBox	Interfaz check box GUI
irr::gui::IGUIComboBox	Combobox widget
irr::gui::IGUIContextMenu	Interfaz menú
irr::gui::IGUIEditBox	Simple linea caja editable para mostra un simple texto
irr::gui::IGUIElement	Clase base para todos los elementos GUI
irr::gui::IGUIEnvironment	GUI Environment. Usado para fabricar y administrar los elementos GUI
irr::gui::IGUIFileOpenDialog	Archivo estándar OpenDialog
irr::gui::IGUIFont	Interface de font
irr::gui::IGUIImage	Interfaz para visualizar imagines GUI
irr::gui::IGUIInOutFader	Elementos para fader salidas o entradas
irr::gui::IGUIListBox	Elemento list box por defecto
irr::gui::IGUIMeshViewer	Visualizar maya en un elemento GUI
irr::gui::IGUIScrollBar	Elemento Scroll bar por defecto
irr::gui::IGUISkin	Vistas de formas de los elementos GUI
irr::gui::IGUIStaticText	Multiple o simple linea de texto
irr::gui::IGUITab	Un tab, hacia que podrían agregarse otros elementos del gui
irr::gui::IGUITabControl	Un control TAB estandar
irr::gui::IGUIToolBar	Se queda a la cima de su padre gusta la barra de menú y contiene los botones de la herramienta
irr::gui::IGUIWindow	El valor predeterminado el elemento de GUI de ventana móvil con la frontera, subtitulo e

	iconos íntimos
irr::video::IImage	La interfaz para los datos de imagen de software
irr::video::IImageLoader	Clase que puede crear una imagen de un archivo
irr::video::IImageWriter	Una por escribir los datos de imagen de software
irr::io::IIrrXMLReader< char_type, super_class >	Interfaz que provee fácil acceso a un archive .XML
irr::scene::ILightSceneNode	Nodo de escena con luz dinamica
irr::ILogger	Interfaz que muestra mensage, precausiones y errorres.
irr::video::IMaterialRenderer	Interfaz para renderizar materials puede usarse para extender el engine con otros materiales
irr::video::IMaterialRendererServices	Interfaz qiue prove algunos metodos para modificaciones avanzadas estados internos de IVideoDriver
irr::scene::IMesh	Clasifique por acceder una malla con los pulidores de la malla múltiples
irr::scene::IMeshBuffer	Struct por estar de acuerdo una malla con un solo material
irr::scene::IMeshCache	Las tiendas de escondite de malla ya cargaron las mallas y proporcionan una interfaz a ellos
irr::scene::IMeshLoader	Clase que puede cargar una malla animada de un archivo
irr::scene::IMeshManipulator	Una interfaz para fácilmente manipule las mallas
irr::scene::IMeshSceneNode	Un nodo de escena para visualizar una maya estatica
irr::scene::IMetaTriangleSelector	Una por hacer a los seleccionadores del triángulo múltiples trabajar como un seleccionador grande
irr::IOSOperator	El operador del sistema operativo proporciona el sistema del funcionamiento los métodos específicos e informaciones
irr::scene::IParticleAffector	Un particle affector modifica particulas
irr::scene::IParticleEmitter	Un emisor de la partícula por usar con los sistemas de la

	partícula
irr::scene::IParticleSystemSceneNode	Un partícula sistema escena nodo por crear la nieve, el fuego, el exlosions, niebla..
irr::scene::IQ3LevelMesh	Una para un wich de la Malla puede cargarse directamente de un Quake3. el bsp-archivo
irr::io::IReadFile	Interfaz qoe prove acceso de lectura a un archivo
irr::core::irrAllocator< T >	
irr::core::irrAllocatorFast< T >	El allocator rápido, sólo ser usado en los recipientes dentro del mismo montón de memoria,
irr::IrrlichtDevice	The Irrlicht device. Permite crear createDevice() o createDeviceEx()
irr::scene::ISceneCollisionManager	El administrador de escenas de collision facilita manipoulacion de nodos de escena para hacer colision.
irr::scene::ISceneManager	El administrado de escena permite el uso de nodos, mayas, camaras y todo lo que se muestre en la pantalla.
irr::scene::ISceneNode	Interfaz de nodo de escena
irr::scene::ISceneNodeAnimator	Animador de nodsos de escena rotacion, pocision, textura,...
irr::scene::ISceneNodeAnimatorCollisionResponse	Nodo de escena especial que hace possible la respuesta de colision
irr::scene::ISceneNodeAnimatorFactory	Interfaz que hace possible la creacion de nodos de escena animados dinamicos.
irr::scene::ISceneNodeFactory	Interfaz que hace possible la creacion de nodos de escena dinamicos.
irr::scene::ISceneUserDataSerializer	Interfaz para datos de lectura y escritura de archivos de usario con archivos .irr
irr::video::IShaderConstantSetCallBack	Interfaz que hace posible mantener constante los programa del gpu en frames.
irr::scene::IShadowVolumeSceneNode	Nodo de escena para renderizar sombras dentro del stencil buffer
irr::scene::ITerrainSceneNode	Un nodo de escena que visualiza unb terreno apartir de un mapa

	algioritmico.
irr::scene::ITextSceneNode	Un nodo de escena en vista de texto 2d en un espacio 3d dimencional.
irr::video::ITexture	Interfaz para el Video Driver dependiendo de la Textura
irr::ITimer	Interfaz para manipular el tiempo virtual.
irr::scene::ITriangleSelector	Interfaz que regresa un triangulo con propiedades especificas
irr::IUnknown	Clase base Para agregar mas objetos al Irrlicht Engine
irr::video::IVideoDriver	Interfaz que permite el uso de las clases de irr::video
irr::video::IVideoModeList	Una lista de todos los modos de videos
irr::io::IWriteFile	Interfaz que provee acceso a escribir en un archivo
irr::io::IXMLBase	Clase Empy usadas para variables a la clase IrrXMLReader
irr::io::IXMLWriter	Interfaz que prove metodos para escribir fácilmente en archivo .XML
irr::core::line2d< T >	Linea 2d entre 2 puntos con intersecciones y métodos
irr::core::line3d< T >	Linea 3d entre dos puntos con intersecciones y métodos
irr::core::list< T >	Lista double temporal
irr::core::list< T >::Iterator	Lista iterator
irr::core::matrix4	Matrix 4x4 mostrando trandformaciones en matrices y calculos 3d
irr::core::plane3d< T >	Clase plana temporal con algunas intersecciones y metodos
irr::core::position2d< T >	Clase simple manejando 2 coordenadas
irr::core::quaternion	Clase Quaternion
irr::core::rect< T >	Rectangulo temporal
irr::video::S3DVertex	Vértice normal usado por el Irrlicht Engine
irr::video::S3DVertex2TCoords	El vértice con dos coordenadas de la textura
irr::video::S3DVertexTangents	El vértice con una tangente y vector del binormal

irr::scene::SAnimatedMesh	La aplicación simple de la interfaz de IAnimatedMesh
irr::io::SAttributeReadWriteOptions	Struct que sostiene datos que describen las opciones
irr::video::SColor	Clase que representa un 32 pedazo color de ARGB
irr::video::SColorf	Clase que representa con cuatro colores punto flotante
irr::SEvent	Struct por sostener los datos de evento. Un evento puede ser un gui, ratón o evento del teclado
irr::video::SExposedVideoData	Estructure por sostener datos que describen drivers y sistema operativo los datos específicos
irr::SIrrlichtCreationParameters	Estructure por sostener los Irrlicht Dispositivo creación parámetros avanzados
irr::SKeyMap	Struct que guarda qué llave pertenece a que la acción
irr::video::SLight	Estructure por sostener datos que describen una luz del punto dinámica
irr::video::SMaterial	Struct por sostener los parámetros para un renderer material
irr::scene::SMesh	La aplicación simple de la interfaz de IMesh
irr::scene::SMeshBuffer	La aplicación simple de la interfaz de IMeshBuffer con el vertices de S3DVertex
irr::scene::SMeshBufferLightMap	La aplicación simple de la interfaz de IMeshBuffer con el vertices de S3DVertex2TCoords
irr::scene::SMeshBufferTangents	La aplicación simple de la interfaz de IMeshBuffer con el vertices de S3DVertexTangents
irr::scene::SParticle	Estructura por sostener los datos de la particula
irr::core::string< T, TAlloc >	La clase del cordón muy simple con algunos rasgos útiles
irr::scene::SViewFrustrum	Define el frustrum de vista. Thats el espacio visto por la cámara
irr::core::triangle3d< T >	3d clase de plantilla de triángulo por hacer descubrimiento de la colisión y otras cosas

irr::core::vector2d< T >	2d clase de plantilla de vector con los muchos operadores y métodos
irr::core::vector3d< T >	3d clase de plantilla de vector con los muchos operadores y métodos

Irrlicht Engine Class Hierarchy

Esta lista de herencia se ordena aproximadamente, pero no completamente, alfabéticamente:

- irr::core::aabbox3d< T >
- irr::core::array< T, TAlloc >
- irr::core::dimension2d< T >
- irr::IEventReceiver
  - irr::gui::IGUIElement
    - irr::gui::IGUIButton
    - irr::gui::IGUICheckBox
    - irr::gui::IGUIComboBox
    - irr::gui::IGUIContextMenu
    - irr::gui::IGUIEditBox
    - irr::gui::IGUIFileOpenDialog
    - irr::gui::IGUIImage
    - irr::gui::IGUIInOutFader
    - irr::gui::IGUIListBox
    - irr::gui::IGUIMeshViewer
    - irr::gui::IGUIScrollBar
    - irr::gui::IGUIStaticText
    - irr::gui::IGUITab
    - irr::gui::IGUITabControl
    - irr::gui::IGUIToolBar
    - irr::gui::IGUIWindow
  - irr::scene::ICameraSceneNode
- irr::io::IFileReadCallBack
- irr::video::IGPUProgrammingServices
- irr::io::IIrrXMLReader< char_type, super_class >
- irr::video::IMaterialRendererServices
- irr::core::irrAllocator< T >
- irr::core::irrAllocatorFast< T >
- irr::scene::ISceneUserDataSerializer
- irr::IUnknown
  - irr::gui::ICursorControl
  - irr::gui::IGUIElement
  - irr::gui::IGUIEnvironment
  - irr::gui::IGUIFont
  - irr::gui::IGUISkin

- irr::ILogger
- irr::io::IAttributeExchangingObject
  - irr::scene::ISceneNode
    - irr::scene::IAnimatedMeshSceneNode
    - irr::scene::IBillboardSceneNode
    - irr::scene::ICameraSceneNode
    - irr::scene::IDummyTransformationSceneNode
    - irr::scene::ILightSceneNode
    - irr::scene::IMeshSceneNode
    - irr::scene::IParticleSystemSceneNode
    - irr::scene::IShadowVolumeSceneNode
    - irr::scene::ITerrainSceneNode
    - irr::scene::ITextSceneNode
  - irr::scene::ISceneNodeAnimator
    - irr::scene::ISceneNodeAnimatorCollisionResponse
- irr::io::IAttributes
- irr::io::IFileList
- irr::io::IFileSystem
- irr::io::IReadFile
- irr::io::IWriteFile
- irr::io::IXMLWriter
- irr::IOSOperator
- irr::IrrlichtDevice
- irr::ITimer
- irr::scene::IAnimatedMesh
  - irr::scene::IAnimatedMeshB3d
  - irr::scene::IAnimatedMeshMD2
  - irr::scene::IAnimatedMeshMS3D
  - irr::scene::IAnimatedMeshX
  - irr::scene::IQ3LevelMesh
  - irr::scene::SAnimatedMesh
- irr::scene::IAnimationEndCallBack
- irr::scene::IMesh
  - irr::scene::SMesh
- irr::scene::IMeshBuffer
  - irr::scene::SMeshBuffer
  - irr::scene::SMeshBufferLightMap
  - irr::scene::SMeshBufferTangents
- irr::scene::IMeshCache
- irr::scene::IMeshLoader
- irr::scene::IMeshManipulator
- irr::scene::IParticleAffector
- irr::scene::IParticleEmitter
- irr::scene::ISceneCollisionManager
- irr::scene::ISceneManager
- irr::scene::ISceneNodeAnimatorFactory
- irr::scene::ISceneNodeFactory
- irr::scene::ITriangleSelector
  - irr::scene::IMetaTriangleSelector
- irr::video::IImage

    - irr::video::IImageLoader
    - irr::video::IImageWriter
    - irr::video::IMaterialRenderer
    - irr::video::IShaderConstantSetCallBack
    - irr::video::ITexture
    - irr::video::IVideoDriver
    - irr::video::IVideoModeList
- irr::io::IXMLBase
- irr::core::line2d< T >
- irr::core::line3d< T >
- irr::core::list< T >
- irr::core::list< T >::Iterator
- irr::core::matrix4
- irr::core::plane3d< T >
- irr::core::position2d< T >
- irr::core::quaternion
- irr::core::rect< T >
- irr::video::S3DVertex
- irr::video::S3DVertex2TCoords
- irr::video::S3DVertexTangents
- irr::io::SAttributeReadWriteOptions
- irr::video::SColor
- irr::video::SColorf
- irr::SEvent
- irr::video::SExposedVideoData
- irr::SIrrlichtCreationParameters
- irr::SKeyMap
- irr::video::SLight
- irr::video::SMaterial
- irr::scene::SParticle
- irr::core::string< T, TAlloc >
- irr::scene::SViewFrustrum
- irr::core::triangle3d< T >
- irr::core::vector2d< T >
- irr::core::vector3d< T >

6.7. CAPÍTULO 7:

COMO INSTALAR EL MOTOR IRRLICHT ENGINE

6.7.1. Planteamiento del capítulo

Este es el capitulo se va a entrar ya a lo que es el modo de instalación del IRRLICHT ENGINE para poder utilizarlo, también se explicara como instalar el compilador DEV-CPP.

Además de cómo instalar IRRLICHT ENGINE al compilador DEV-CPP para empezar a programar.

6.7.2. Objetivos del capítulo

Es deseable que al finalizar el capítulo el estudiante:

- Saber como instalar el motor de desarrollo IRRLICHT ENGINE.
- Conocer los pasos a seguir para instalar el Compilador DEV-CPP

6.7.3. Explicación del contenido

6.7.3.1. Introducción

Para poder hacer uso del motor IRRLICHT ENGINE es necesario de otra herramienta, conocida como compilador, esta herramienta se encarga de convertir el archivo fuente a archivo objeto y así poder generar un archivo ejecutable.
Para este capitulo se trabajara con el Compilador DEV-CPP una herramienta gratuita para poder escribir nuestro código.

6.7.3.2. Comenzar a instalar

A continuación se explicara paso a paso la instalación de el motor IRRLICHT ENGINE y del compilador DEV-CPP, para poder comenzar a programar, lea detenidamente paso a paso las instrucciones, para evitar errores en la instalación

Las bases para poder desarrollar

Antes de empezar a estudiar los módulos de ejemplos, se van a dar unas pequeñas aclaraciones, que son indispensables a la hora de desarrollar una aplicación con las herramientas de desarrollo aplicadas (**IRRLICHT ENGINE y DEV-CPP**), estos recursos los encontraras en el CD.

IRRLICHT ENGINE 1.1

irrlicht-1.1

Se recomienda poner esta carpeta en la partición principal de Windows para de este modo poderlo ubicar fácilmente.

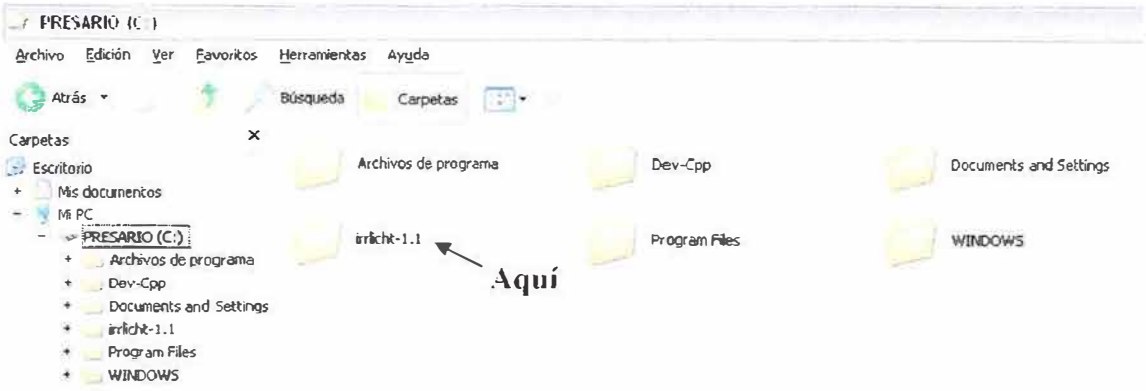


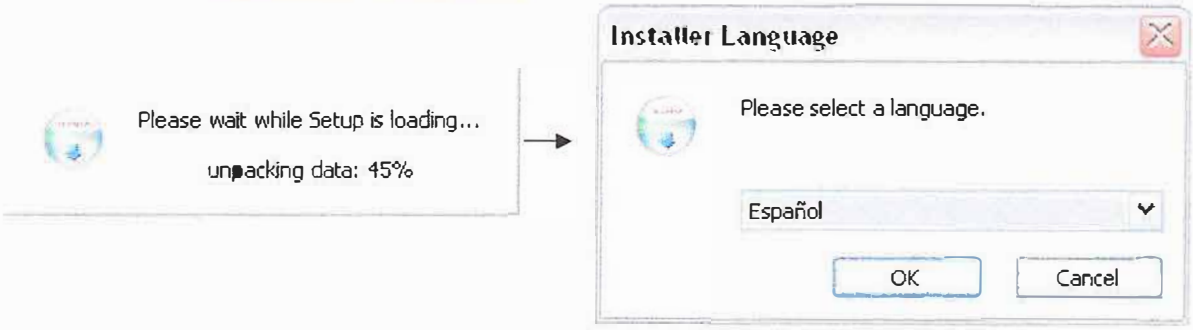
Figura 6.0.1

Para poder manipular este paquete SDK necesitamos una herramienta de programación en este modulo trataremos con la aplicación **DEV-CPP**

devcpp-4.9.9.2_setup

Este es un ejecutable a diferencia del anterior sigue una serie de pasos a la hora de su instalación.

El instalador nos da la bienvenida. Nos solicita que no instalemos esta versión sobre una instalación existente, es decir si ya se encuentra instalado, ya que no posee un verificador de instalación. Si no tenemos ninguna versión de esta aplicación instalada en el equipo damos aceptar, de lo contrario luego de que cargue la siguiente ventana nos da las opciones [OK] ó [Cancel]

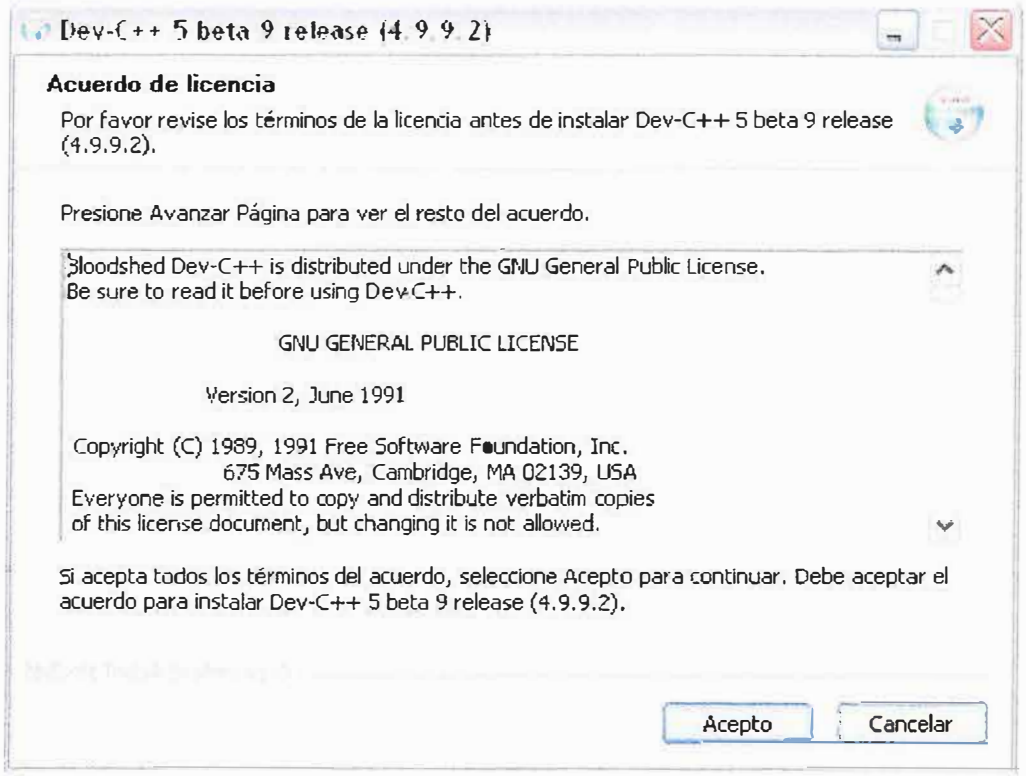


Como se comenta anteriormente si ya posees una versión de esta aplicación instalada, lo recomendable es desinstalarla primero y luego proseguir con la instalación:

[Cancel] ⟶ Cancela la instalación de la aplicación DEV-CPP.

[OK] ⟶ Prosigue la instalación.

Si deseamos otro idioma en este Combo Box (Caja de texto) vasta con dar un clic en el, y seleccionar los disponibles, luego de esto pasa a la siguiente ventana.

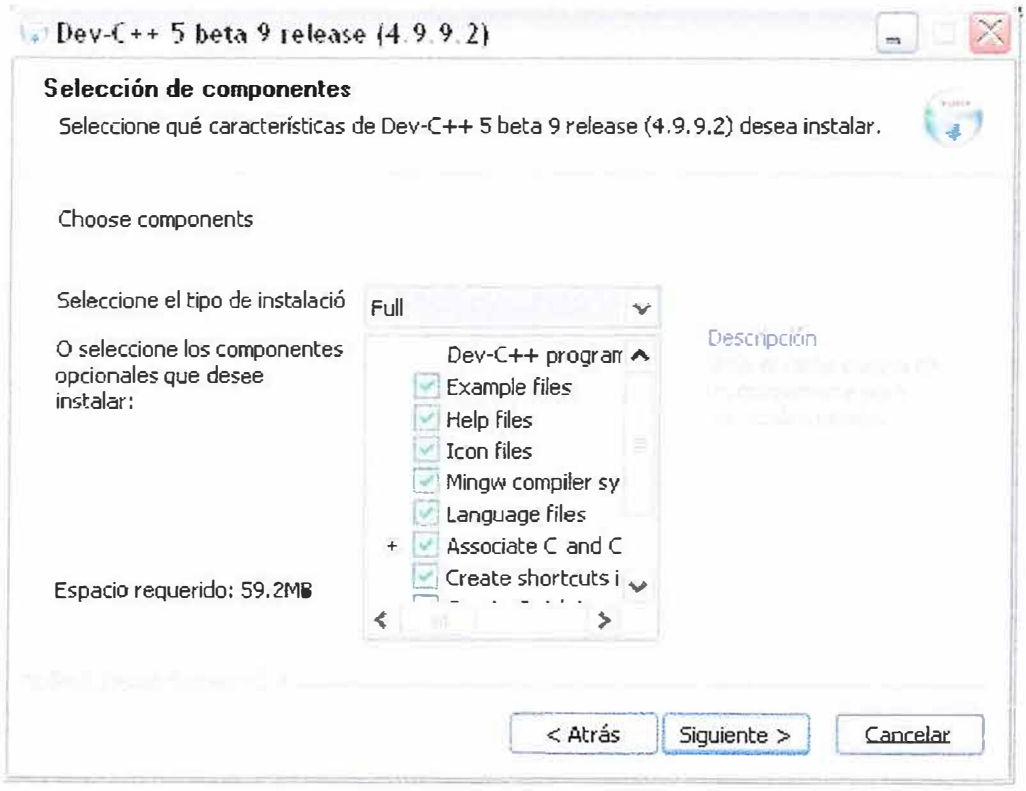


Esta ventana contiene un Acuerdo de licencia, un acuerdo del creador con el cliente de los usos que puede brindarle y los derechos que debe acreditarle, nos da dos opciones:

Acepto ⟶ Estamos de acuerdo con sus condiciones y continúa la instalación.

Cancelar ⟶ No estamos de acuerdo salimos de la instalación.

Si dio clic en Acepto mostrara la siguiente ventana:

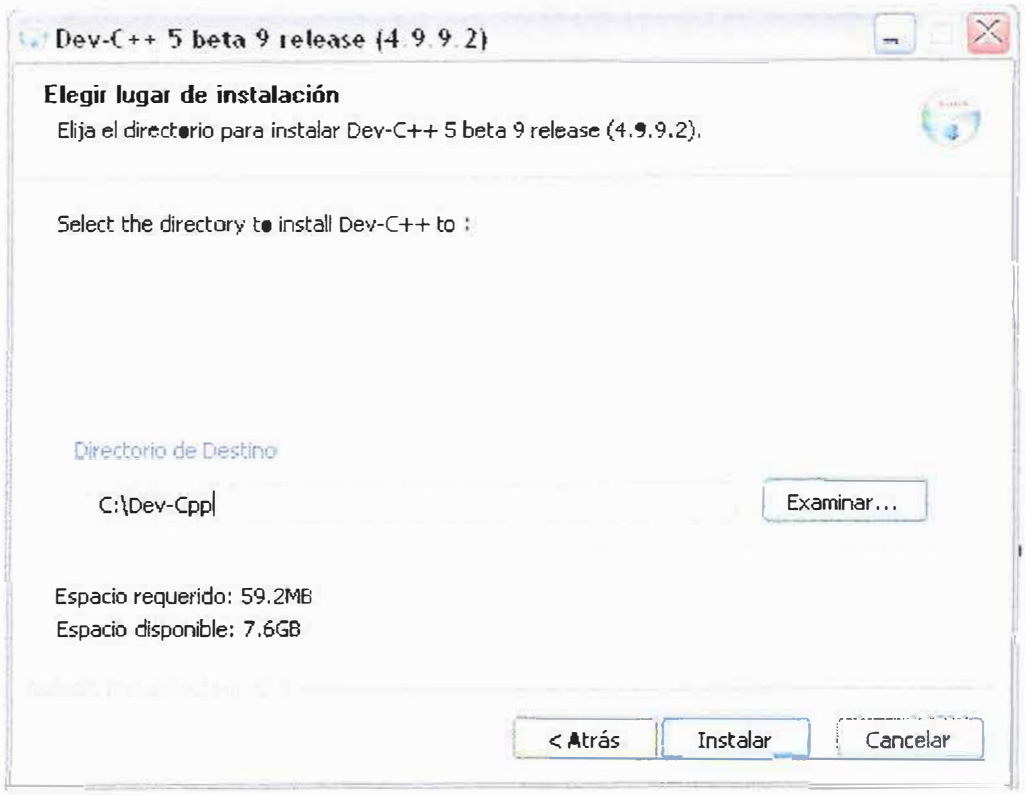


En esta ventana nos muestra los componentes de los cuales dispone por el momento es irrelevante analizar esta parte, también nos ofrece 3 opciones:

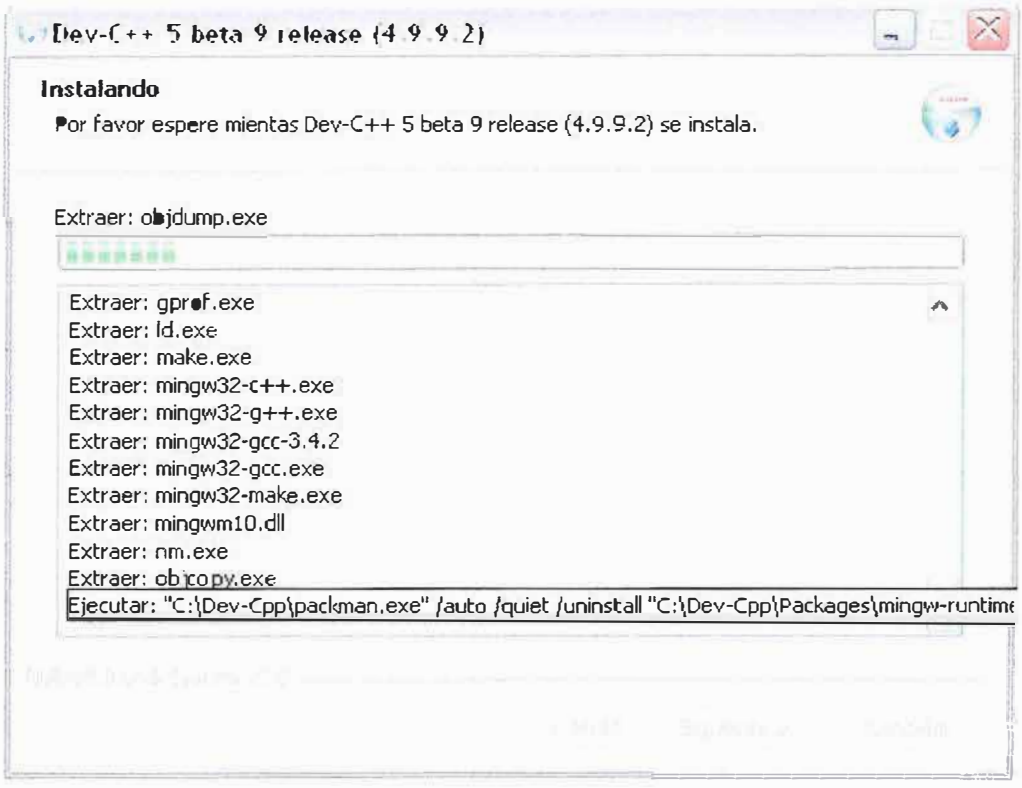
< Atrás Regresar a la ventana anterior.
Siguiente > Proseguir con la instalación.
Cancelar Salir de la instalación.

Si se presiono el botón **"Siguiente"** nos mostrara la siguiente ventana esta nos muestra la ruta por defecto donde lo va a instalar, si deseamos instalarlo en un parte especificada, solo tenemos que dar un clic en Examinar... y especificamos la ruta que deseamos una vez echo esto, la aplicación realiza una última confirmación:

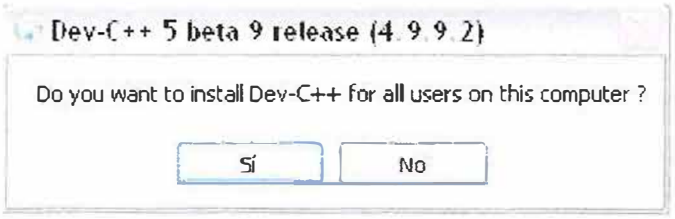
< Atrás Regresar a la ventana anterior.
Instalar Instalar la aplicación.
Cancelar Cancelar la instalación.



Si nuestra respuesta fue Instalar Solo restara espera que se instale la aplicación y listo.



Una vez terminado nos envia el siguiente pantanazo:



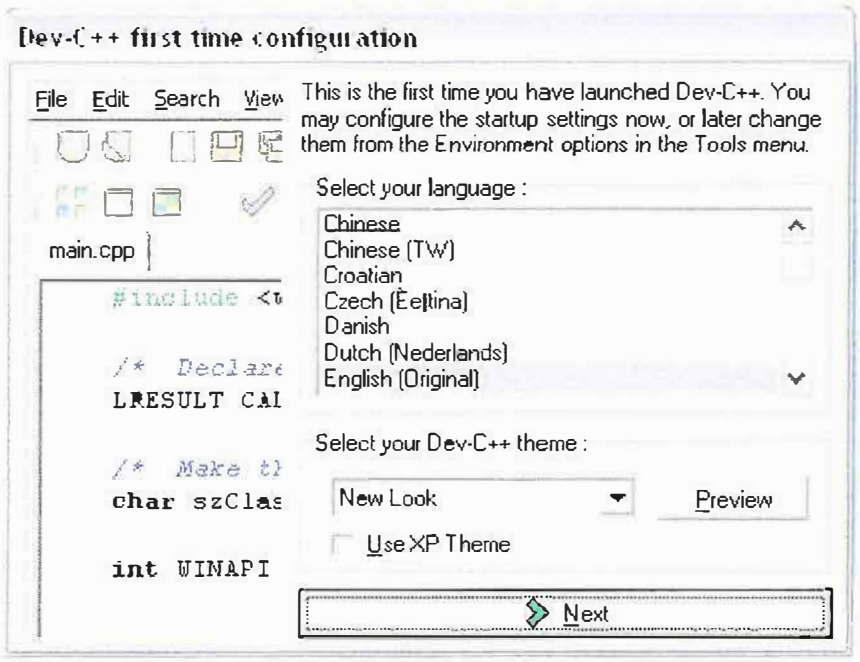
Esta ventana dice lo siguiente, "¿Desea que esta aplicación este disponible para todos los usuarios de esta computadora?"

Sí	Todos los usuarios registrados en el equipo tendrán esta aplicación disponible.
No	Solo lo tendrá la cuenta (usuario) que lo instalo.

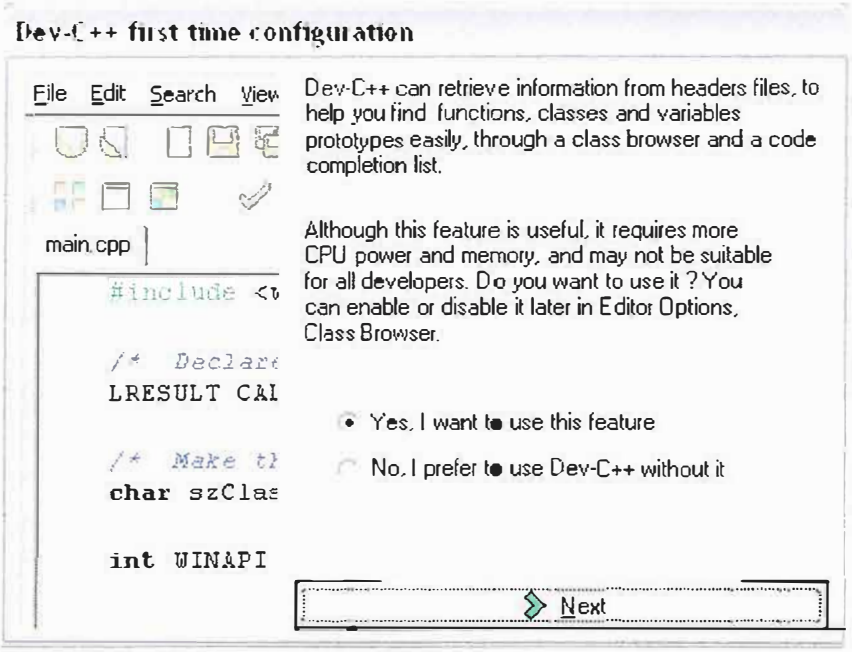
Finalmente damos terminar y dejamos seleccionado la opción de
"Ejecutar Dev-C++ 5 beta 9 reléase (4.9.9.2)"

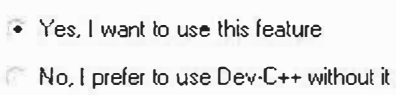
Una vez hecho esto aparece otra ventana dándonos unas recomendaciones para con el programa viendo que no hay mas opciones damos **ACEPTAR.**

Seleccionamos el idioma, ingles por defecto y damos clic en Next:



Nos sale otra ventana, para activar las clases del código:

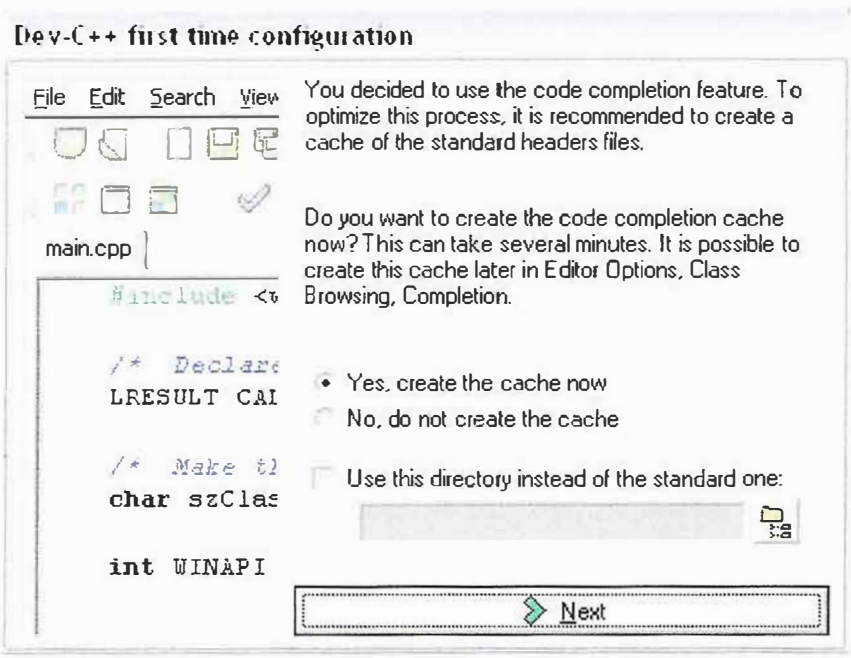


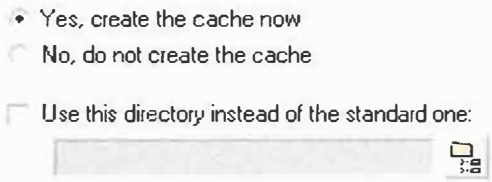


Cargar las librerías en la aplicación.
Usarlo sin librerías.

Aunque no son tan necesarias para lo que se va desarrollar es recomendable instalarlo, ya que si domina algo de programación podrá emplearlas para el desarrollo de videojuegos.

Nos hace una última pregunta, de si deseamos crear el cache:



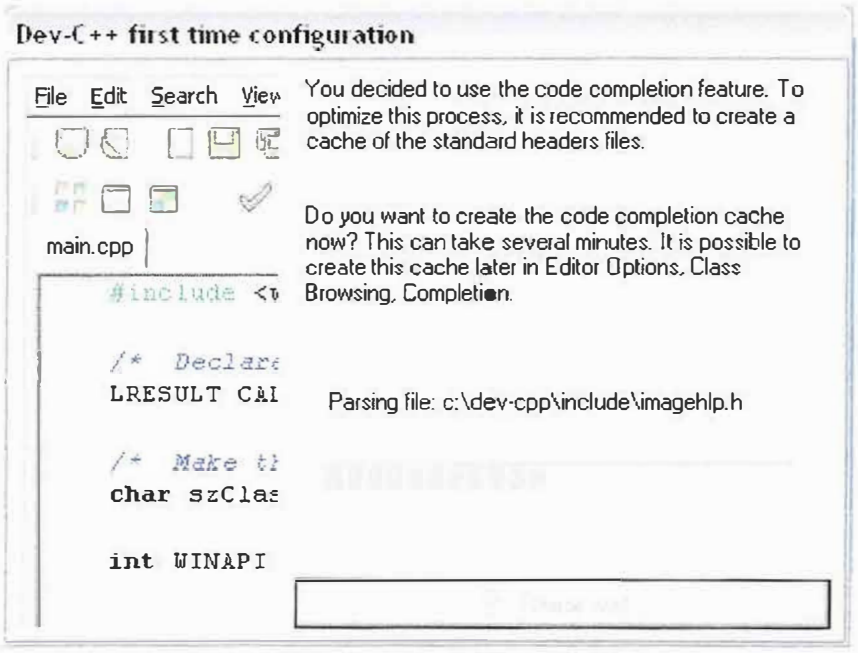


→ Si crea la cache.

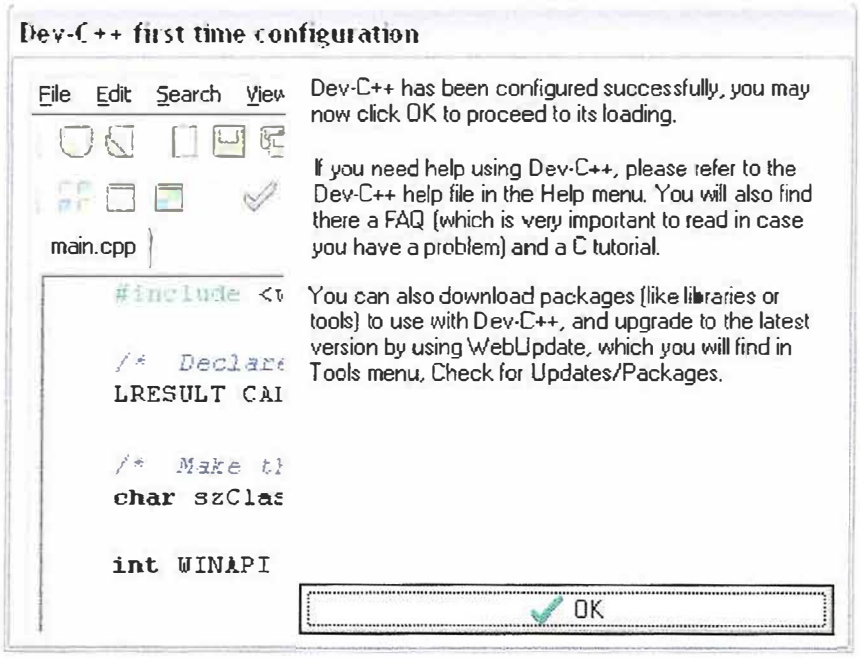
→ No la crea la cache.

→ Si activa esta pestaña puede especificar el cache que desee.

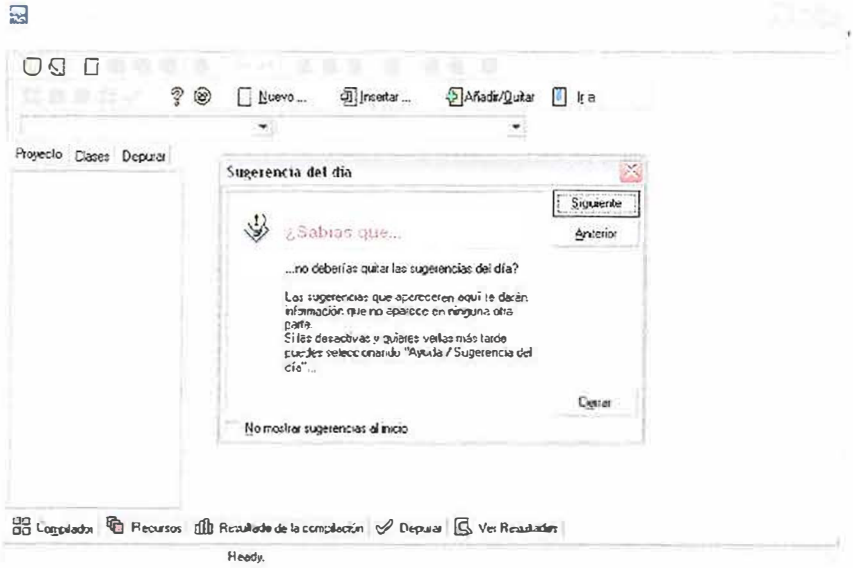
Para este proyecto es recomendado crear la cache, la primera opción ella por defecto ya esta seleccionada, sin embargo siempre debe seccionarse, una vez se cargue la cache estaremos listos para trabajar el **IRRLICHT ENGINE** en el **DEV-CPP**.



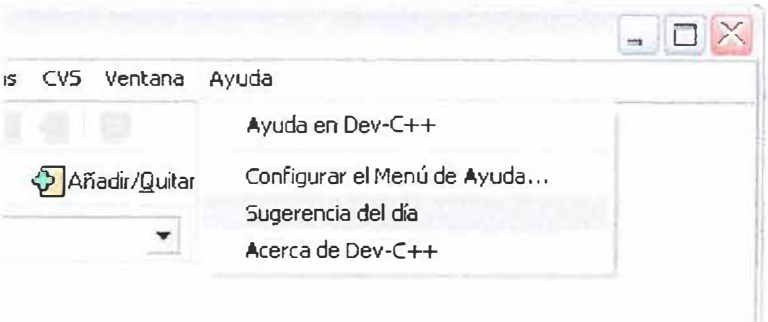
Finalmente damos click en OK y esta la aplicación lista para trabajar.



Ya estamos listos para trabajar en el **DEV-CPP**



Por cuando se abre la aplicación aparece una ventana de sugerencias para programar, si no las desea chulee la opción "No mostrar sugerencia al inicio y listo" y si desea recuperarla en ayuda podremos recuperar esta ventana y cambiar de opinión.



6.8. CAPÍTULO 8:

CREAR NUESTRO PRIMER PROYECTO DE IRRLICHT ENGINE

6.8.1. Planteamiento del capítulo

Para comenzar a trabajar ya con el motor es necesario seguir unas recomendaciones, las cuales se aclararan en este capitulo, se recomienda analizar cada información facilitada por el documento para comprender a la perfección lo que se desea realizar.

6.8.2. Objetivos del capítulo

Es deseable que al finalizar el capítulo el estudiante:

- Conocer los pasos a seguir para crear un proyecto de IRRLICHT.
- Saber que debe hacer para que el proyecto funcione correctamente.
- Crear el primer código fuente y asignarle un nombre

6.8.3. Explicación del contenido

6.8.3.1. Introducción

La programación con IRRLICHT no es muy compleja solo necesita de la creatividad del programador, claro esta que para ello debe conocer los métodos con los que cuenta el motor, se va a crear un proyecto con el cual podremos crear programa IRRLICHT.

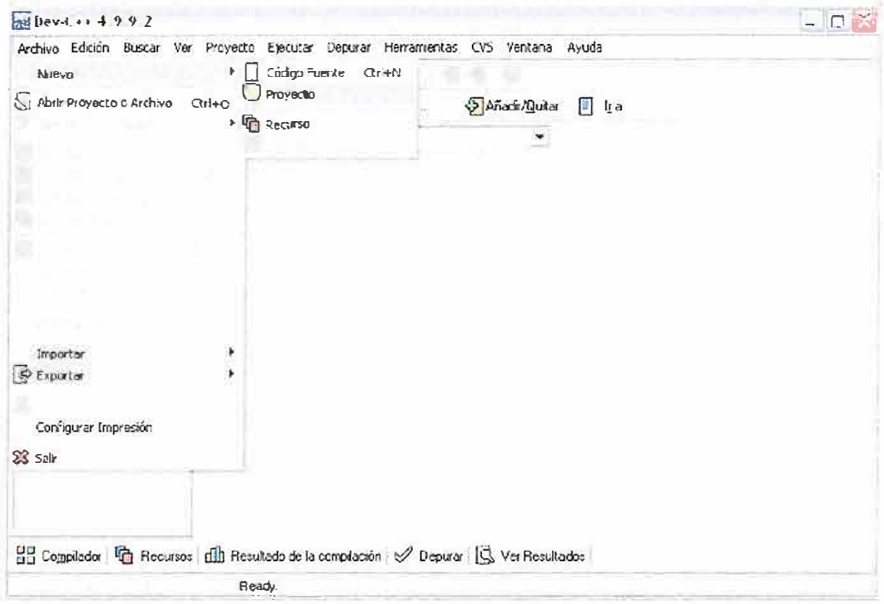
6.8.3.2. Comenzar a instalar

Lee detenidamente cada línea de información para comprender los pasos a seguir para desarrollar proyectos en el compilador aptos para el motor de videojuegos IRRLICHT ENGINE.

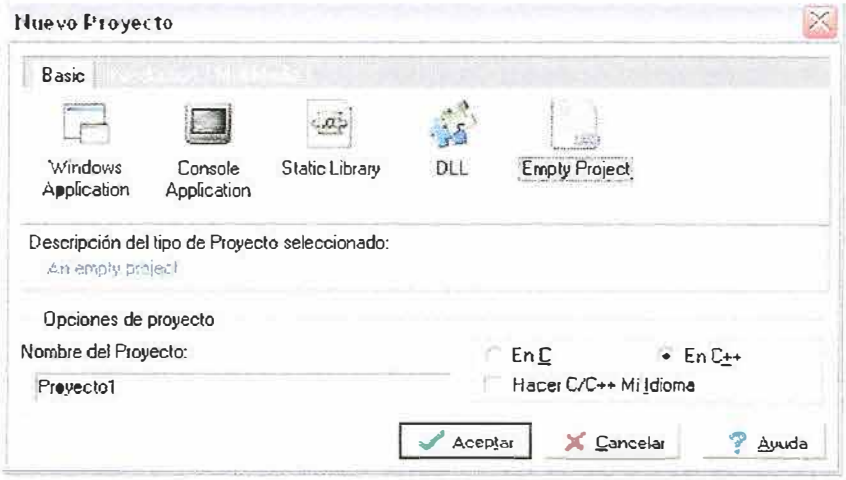
Como desarrollar un proyecto para el motor IRRLICHT ENGINE

Estas son unas recomendaciones para crear un proyecto de **IRRLICHT ENGINE:**

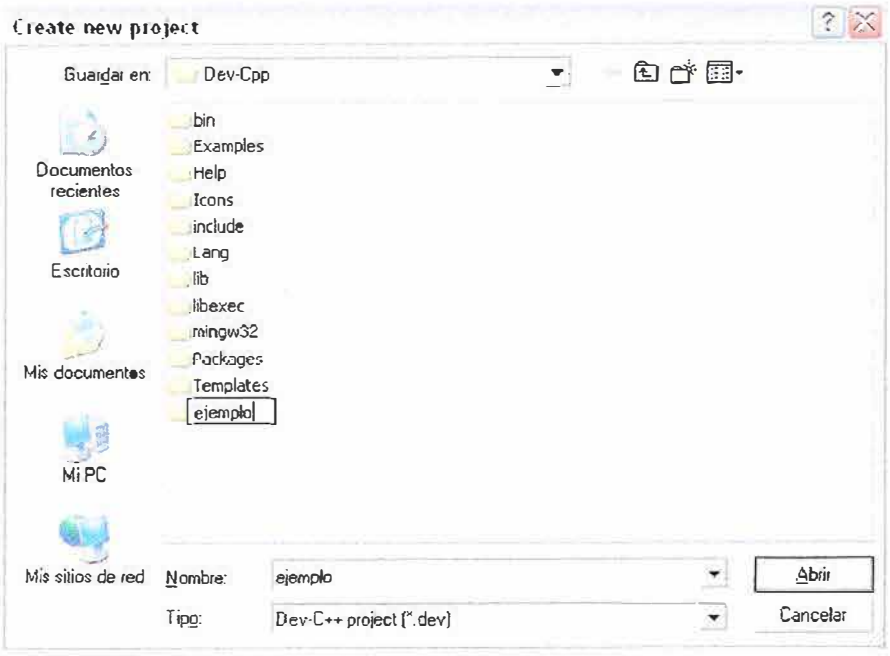
- Dirigirse a archivo y escoger nuevo proyecto.



- Escogemos un proyecto vació (Empty Project) y damos le damos un nombre en nombre de proyecto (te recomendamos guardarlo en una carpeta con el mismo nombre para no confundirte, en el resto del modulo por el momento se le dara el nombre "**ejemplo**" y le damos aceptar.



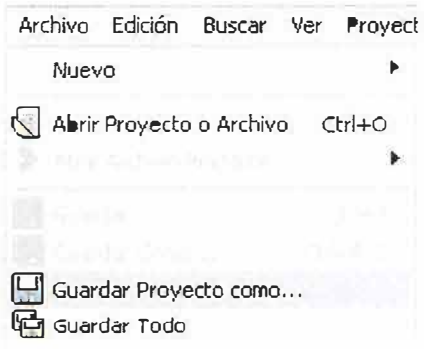
Por defecto nos ofrece guardar en la misma carpeta donde se instalo el DEV-CPP pero en realidad podemos guardarlo donde nos plazca, para evitar confusiones por el momento guardémosla en la misma carpeta, recuerda antes crear una carpeta con el mismo nombre del proyecto para evitar confusiones.



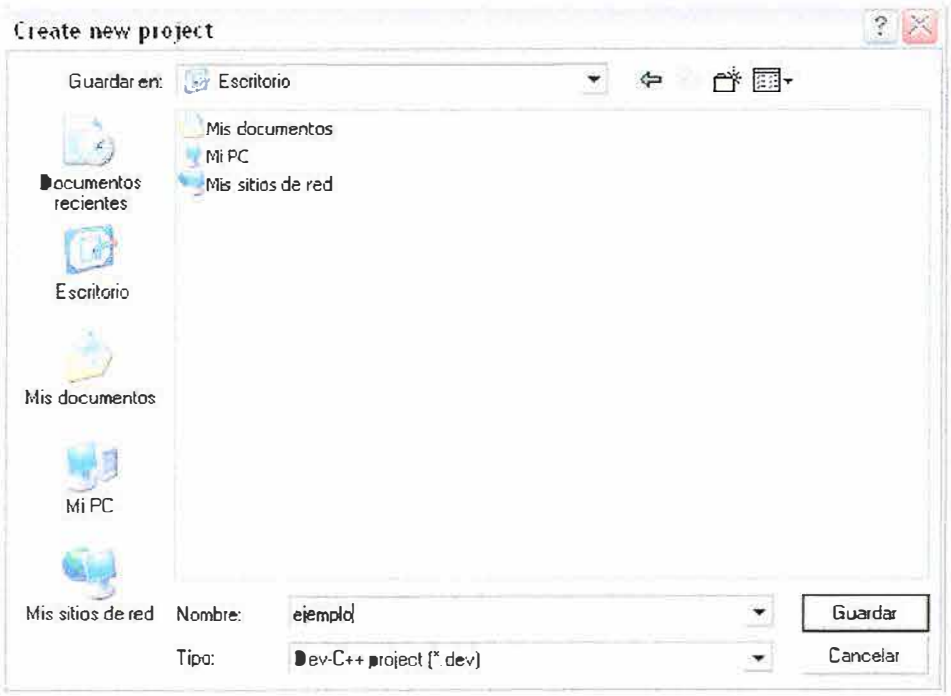
Y lo guardamos ahí.

Por alguna razón, el archivo "**ejemplo.dev**" no aparece guardado en la carpeta se recomienda hacer lo siguiente:

Nos vamos a archivo y seleccionamos **"Guardar Proyecto como..."**:

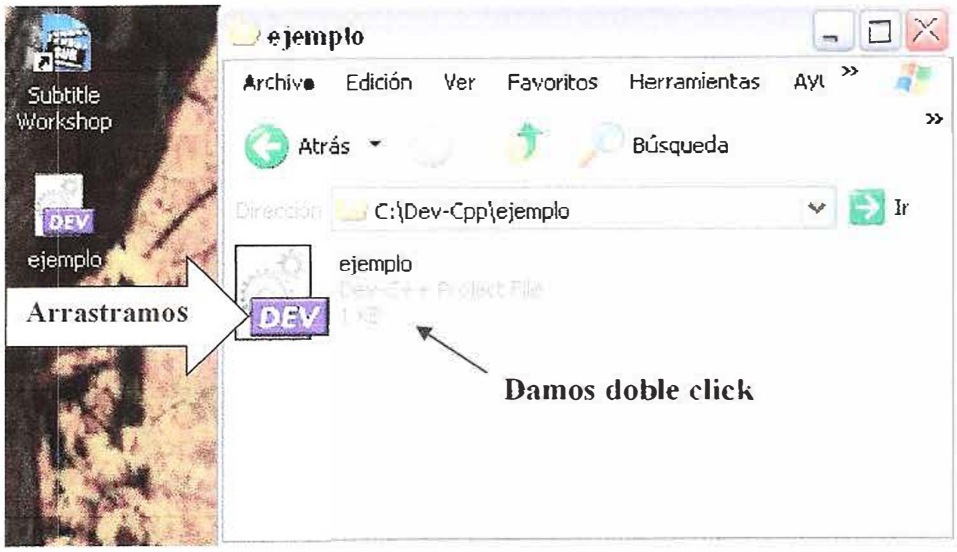


Guardémoslo en cualquier parte menos en donde se guardo primero, por ejemplo yo lo guardo en escritorio.

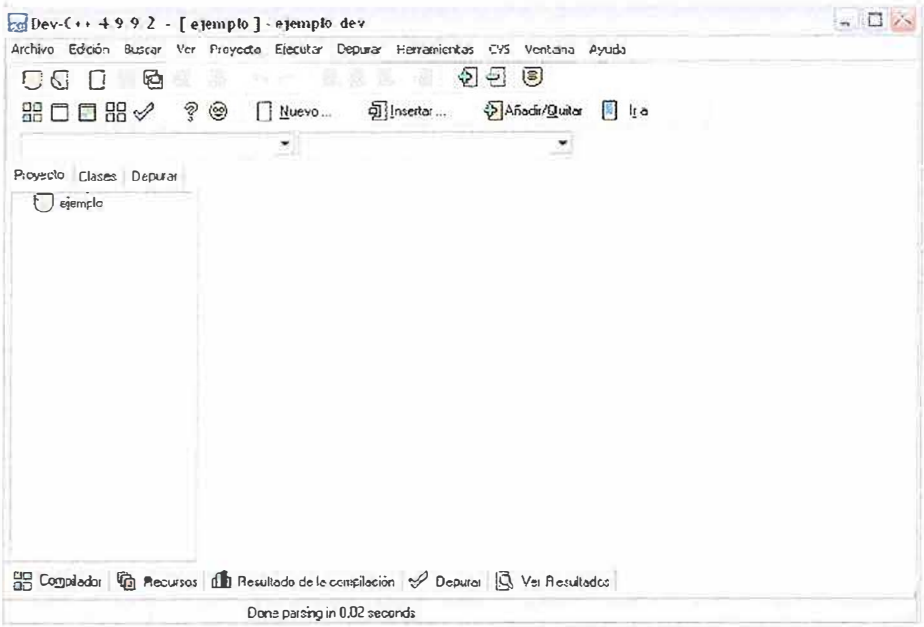


Luego Cierro la aplicación.

Una vez echo esto copiamos el archivo **"ejemplo.dev"** que se encuentra en escritorio y lo pego en la carpeta **"ejemplo"** que creamos anteriormente.

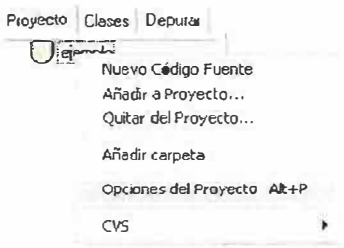


Bueno damos doble click sobre el archivo **"ejemplo.dev"** que se encuentra en la carpeta ejemplo.
Devuelta en la aplicación **DEV-CPP**



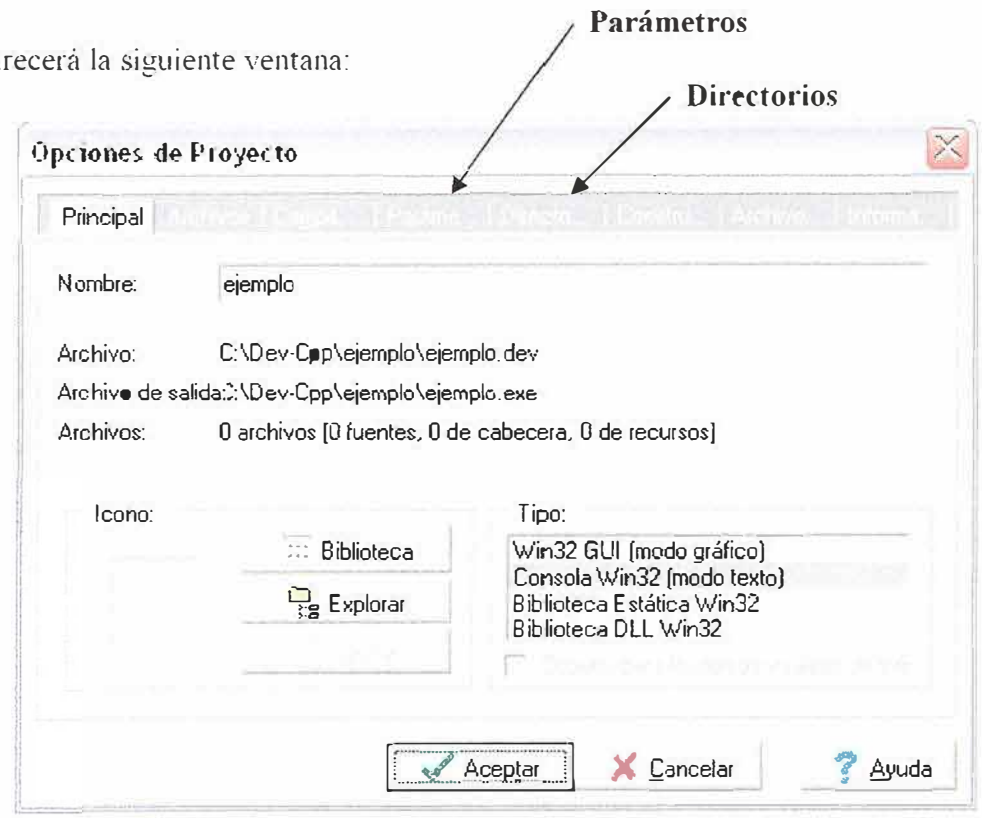
Necesitamos ahora incluir las librerías y los includes del **IRRLICHT ENGINE** al proyecto para poder trabajar.

Paso 1.



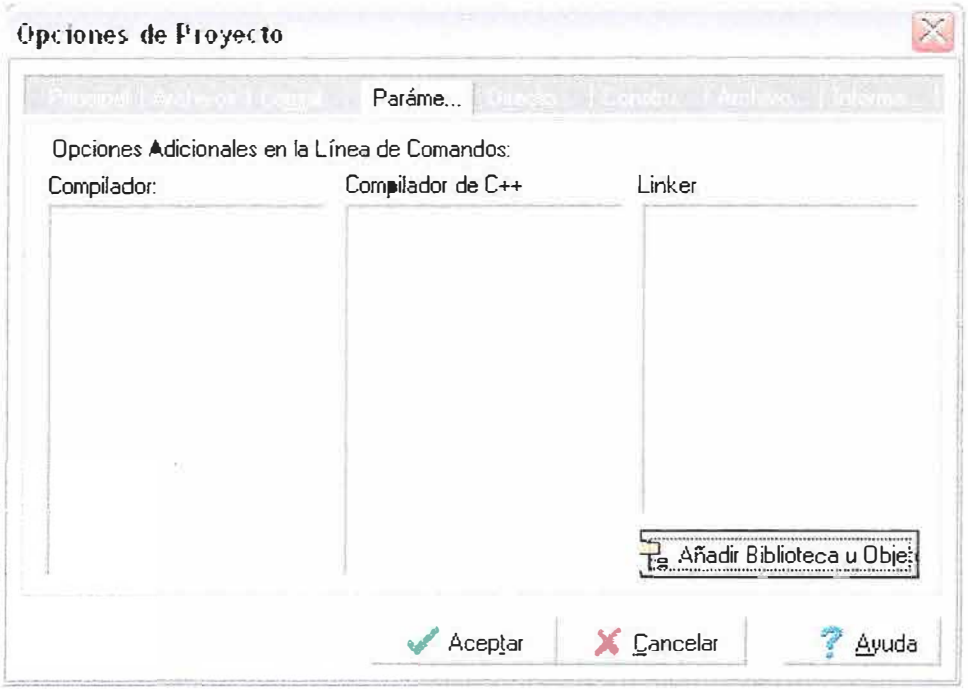
En la pestaña **"proyecto"** a mano izquierda, damos un click derecho al proyecto **"ejemplo"** y nos dirigimos a opciones del proyecto.

Nos aparecerá la siguiente ventana:



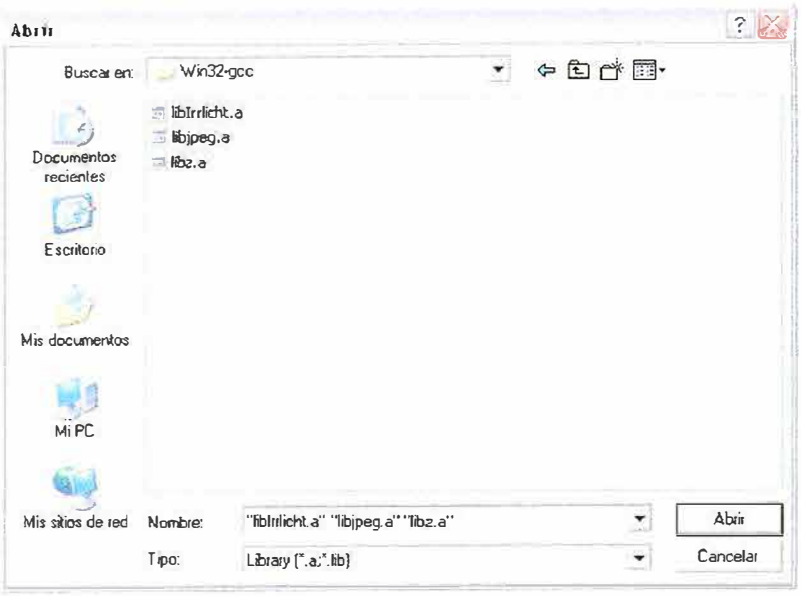
Vamos a trabajar en las pestañas **Parámetros** y **Directorios**

En parámetros vamos hacer lo siguiente:

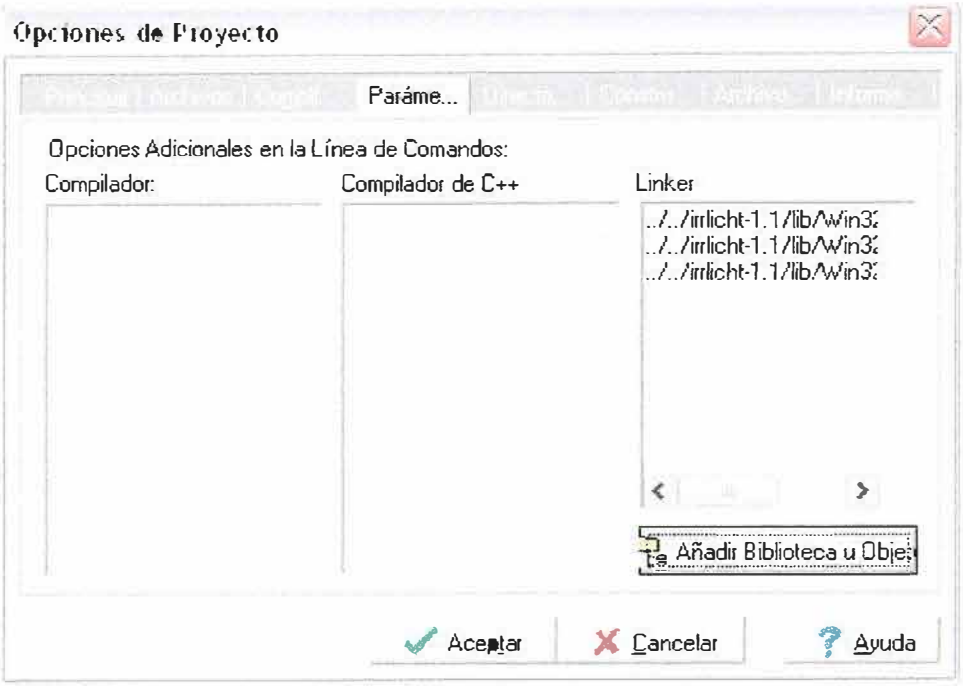


A → Esta es la ruta de la librería = "C:\irrlicht-1.1\lib\Win32-gcc" damos un clic en el botón **"Añadir Biblioteca u Objeto"**

B → Seleccionamos las librerías que necesitamos, en nuestro caso escojan todas



Y dan click abrir.

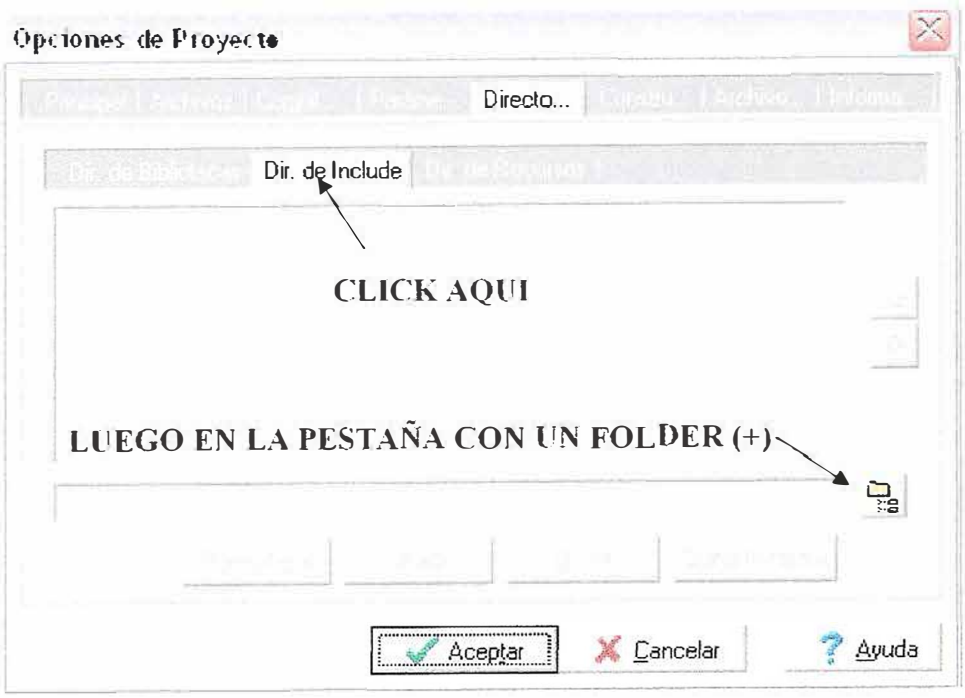


Así debe quedar una vez realizado con éxito la carga, y damos aceptar.

Realizamos los mismos pasos anteriores, pero esta vez en lugar de parámetros se elegirá directorios.

En Directorios vamos hacer lo siguiente:

A → Damos click en la pestaña DIR. De incluye.

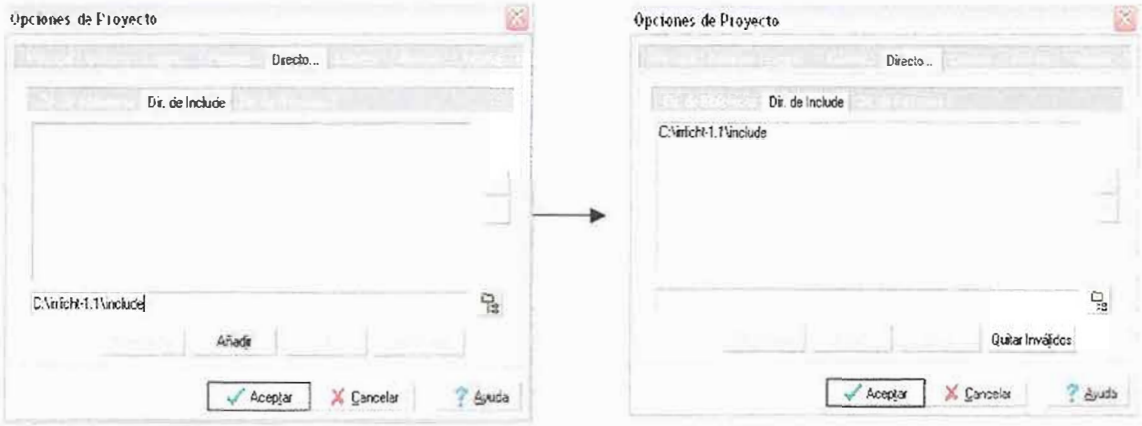


B → En la pestaña con el dibujo de un fólder (+) damos un click, aparecerá la siguiente ventana:

Esta vez lo que elegiremos será una carpeta, por lo que el modo de abrir es diferente, intenten que quede así:

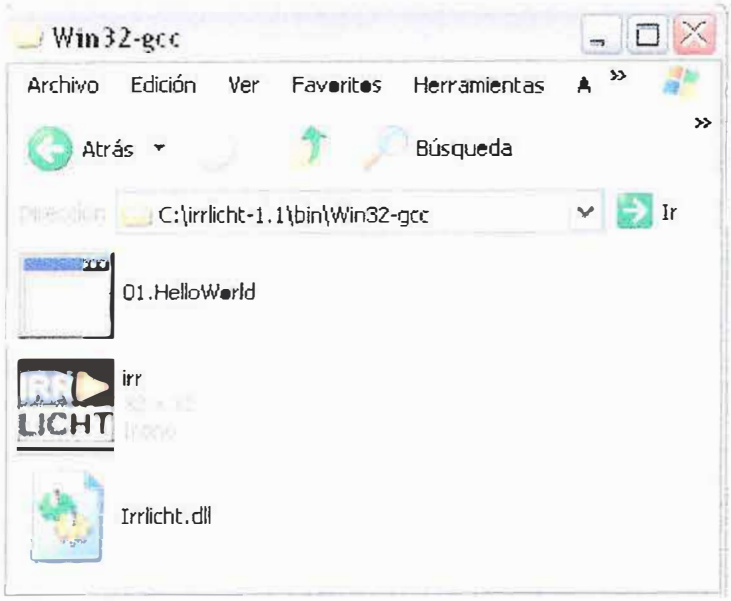
Ten en cuenta que lo que se desea es que el proyecto posea las librerías del motor **IRRLICHT ENGINE.**

Una vez hecho esto, damos click en añadir y luego damos aceptar.



Listo ya nuestro proyecto posee las librerías de **IRRLICHT ENGINE** para desarrollar las simulaciones.

No olvides un dato importante, en la carpeta bin de IRRLICHT ENGINE, con la ruta **C:\irrlicht-1.1\bin\Win32.gcc**

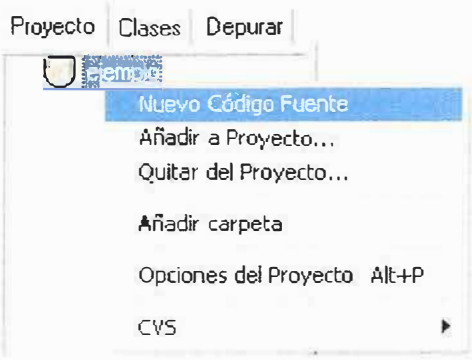


Se encuentra un DLL importante para que corran las aplicaciones del IRRLICHT ENGINE, sin el no funciona, hay varias formas de linkearlo, pero en este modulo, vamos hacer un link perfecto, copiemos el archivo **"irrlicht.dll"** a la carpeta **"ejemplo"**.

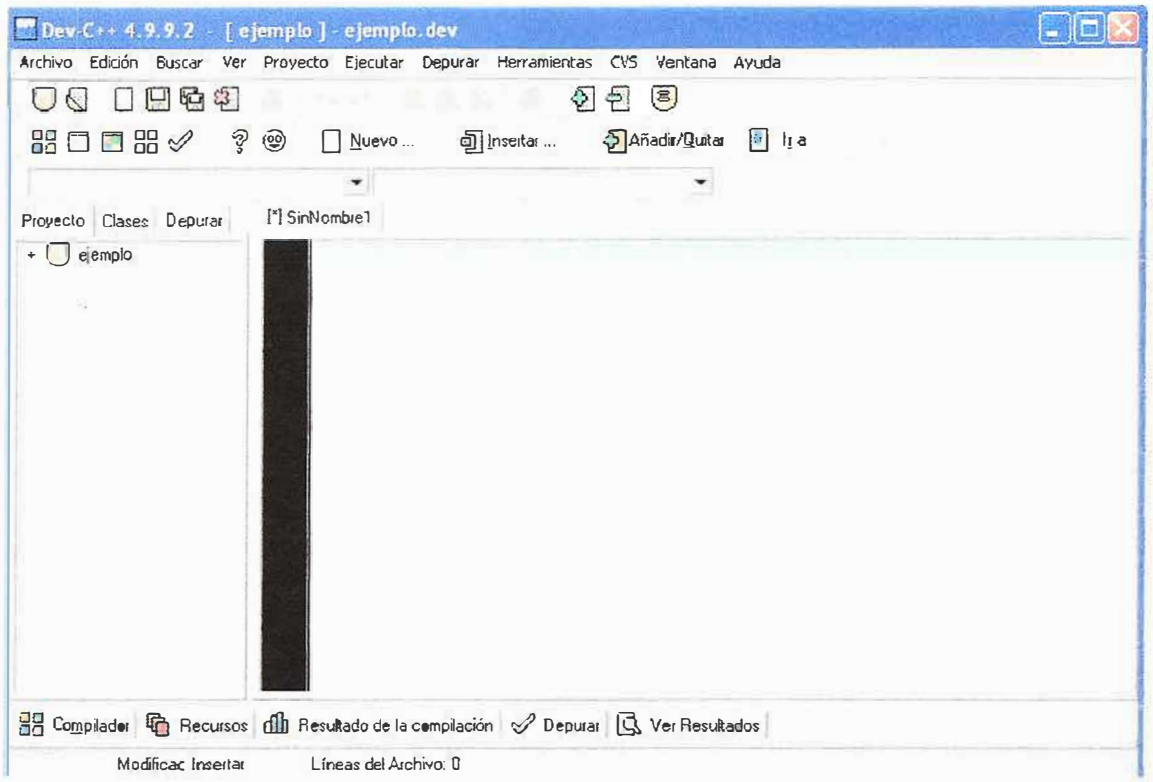
Esto nos permitirá abrir el ejecutable sin la necesidad de tener que linkearlo, bueno estará en la misma carpeta, es un método bastante efectivo, y se ahorra el problema de tener que buscar al final los componentes necesarios, para que funcione correctamente, si se utilizan otros DLL es recomendable que los guarde en la misma carpeta del proyecto para una vez terminado solo sea empaquetar.

Finalmente agregamos nuestro primer archivo CPP, en el cual escribiremos el código. Los pasos a seguir son los siguientes:

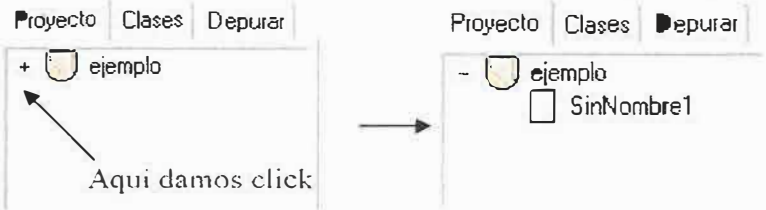
Paso 1:



A mano izquierda la pestaña proyecto, como se comento anteriormente, debe dar un click derecho en el proyecto **"ejemplo"** en este caso se seleccionara la opción **"Nuevo Código Fuente"** y nos aparecerá un archivo con el nombre **"SinNombre1"** este es un archivo sin extensión al que le asignaremos la extensión correspondiente.

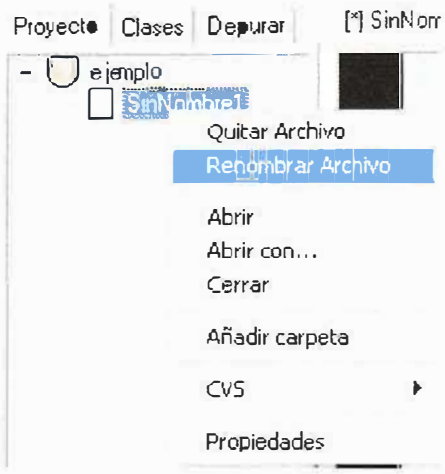


De momento nuestro archivo no posee ningún tipo de extensión solo es un archivo en el cual podemos escribir, vamos a hacer lo siguiente si se fija a mano izquierda en la pestaña proyectos encontrara que ahora el proyecto ejemplo a su lado izquierdo posee un simbolo [+] demos un click en el.



Sigue en la siguiente hoja

Ahora notaremos que el símbolo [+] se cambio por un [-] y que el nuevo código fuente que creamos esta como sub. Del proyecto **"ejemplo",** vamos ahora a darle un nombre:



Si damos un click derecho sobre el archivo **"SinNombre1"** se van a aclarar las opciones que trabajaremos:

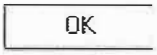
→ Permite borrar el archivo seleccionado
→ Permite cambiar el nombre del archivo selec.
→ Hace visible el archivo en el proyecto.
→ (No se explicara su utilidad en este proyecto).
→ Cierra el archivo es similar a **"Quitar Archivo".**

→Permite crear una carpeta al proyecto.
→(No se explicara su utilidad en este proyecto).
→ Muestra las características del archivo.

Damos un click en **"Renombrar Archivo"** y nos aparecerá la siguiente ventana:

Establecemos un nombre escribamos para mantener el ejemplo comprensible CodEjemplo.CPP, este nombre se le asigna para no perder el hilo del modulo, puede ser cualquier nombre, la extensión tampoco es necesaria, pero en este ejemplo se va a agregar para que saber que es un archivo CPP.

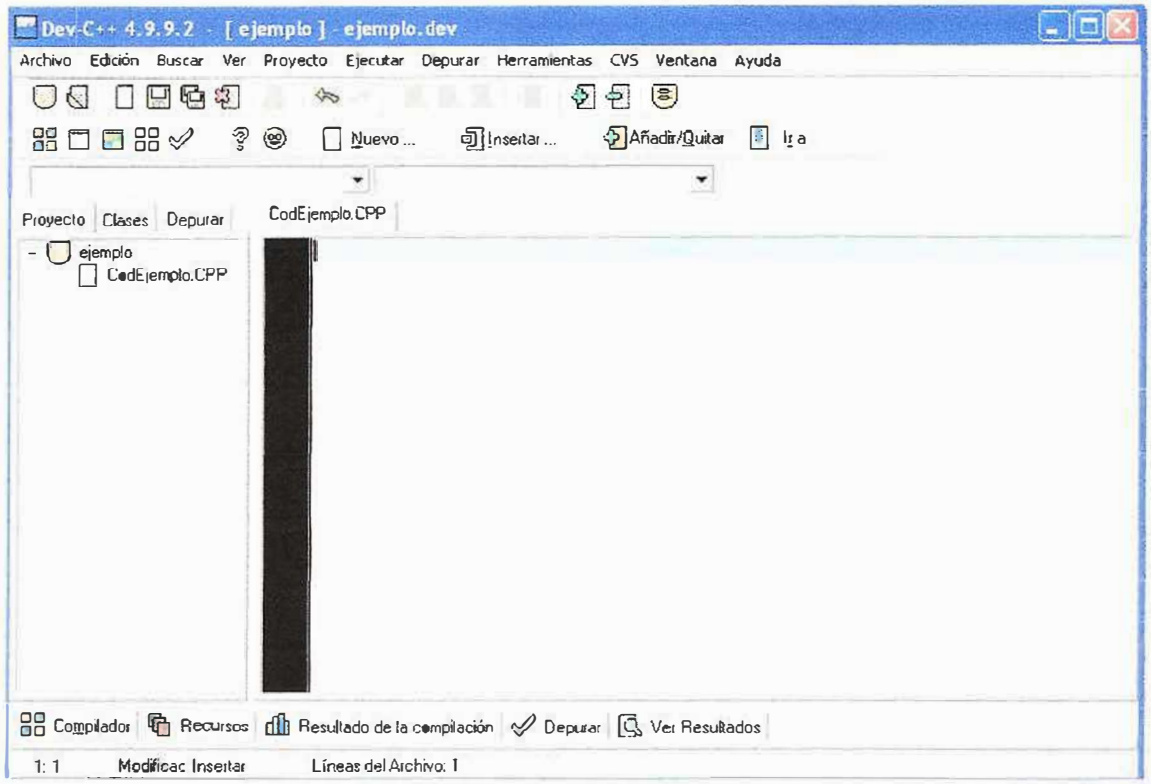
Bueno resta dar las siguientes opciones:



Para confirmar el nombre.

No guarda los cambios.

Para este modulo damos OK...



En este momento, ya la aplicación esta lista para trabajar, ahora podemos desarrollar programas. Esta es la base para crear aplicaciones con IRRLICHT ENGINE y el DEV-CPP en los siguientes módulos iniciaremos la programación, se iniciara con el código básico para trabajar con **IRRLICHT ENGINE** y consejos para programa.

Consejo 1:

Cuando quieras desarrollar programas con irrlicht, debe tener bien claro que al proyecto nuevo que cree siempre debe realizar los pasos del **"Modulo 0-1"** de lo contrario el programa no funcionara.

7. Bibliografía:

Libros:

Aprenda Microsoft visual C++ 6.0 ya
Autor: Check Spar
Editorial: Mc Graw Hill
ISBN: 84-481-2421-9

The OpenGL Utility Toolkit (GLUT) Programming Interface
Autor: Mark J. Kilgard
Editorial: Silicon Graphics, Inc.
API Versión 3

Historia de los videos juegos. Juan R. Pérez Q.

www.rinconsolero.com/nuke-7/html/iframe.php?

file=http://www.rinconsolero.com/rinconsolero.v2/historia_de_los_videojuegos.htm

Crisis de los videos juegos

www.delcorp.org/video_juegos/crisis.htm

Información Técnica de las Consolas. Juan R. Pérez Q.

www.rinconsolero.com/nuke-7/html/iframe.php?file=http://

www.rinconsolero.com/rinconsolero.v2/inf_tecnica.htm

Información sobre Código Abierto

http://opensource.org/

Información sobre Irrlicht Engine

http://irrlicht.sourceforge.net/

www.wikipedia.org/irrlicht

Demostraciones con el motor Irrlicht Engine

http://irrwizard.sourceforge.net/wiki/index.php/

Componentes Externos

http://www.skyesurfer.net/keless/IrrLicht/ICE/

http://audiere.sourceforge.net/documentation.php

www.gldomain.com

http://www.alobbs.com/revistas/opengl1

www.UltimateGameProgramming.com

Motores de juegos

http://ima.udg.es//iiia/GGG/TIC2001-2416-C03-01/docs/engines.pdf

DESARROLLO DE LINEAMIENTOS PARA LA CREACIÓN DE VIDEOJUEGOS APLICANDO EL LENGUAJE C++ UTILIZANDO UN MOTOR DE VIDEOJUEGOS LLAMADO IRRLICHT ENGINE.

Realizado por:

RODRIGUEZ CORRALES JAVIER RAFAEL

Asignatura:

INVESTIGACIÓN FORMATIVA IV

Docente:

FREDDY BRICEÑO

Tutor:

JAVIER HENRÍQUEZ

Curso:

10 °B

CORPORACIÓN EDUCATIVA MAYOR DEL DESARROLLO SIMÓN BOLÍVAR

INGENIERÍA DE SISTEMAS

BARRANQUILLA

2006-2

Introducción

En el mundo siempre hay algo nuevo que conocer, las cosas llegan cuando menos se esperan, la imaginación es un don que nos han otorgado, siempre esta golpeando por la espalda y exige conocer mas, en ocasiones cosas imposibles de encontrar, pero que con la fuerza de voluntad se puede realizar, aunque sea de forma artificial.

El videojuego, es un medio virtual que nos permite disfrutar de nuestras más grandes fantasías.

Muchas personas tienen sueños, pero no encuentran la manera de materializarlo, por este motivo la intención de este proyecto, es brindarles una ayuda básica que les permita mostrar toda esa creatividad en un videojuego dándole las bases necesarias para su desarrollo.

Existen muchas herramientas gratuitas en Internet desconocidas para las personas, estas le pueden ser útiles en su deseo de crear videojuegos, ahora bien conoceremos los problemas más usuales en los videojuegos creados, para no cometer los mismos errores y así obtener lo que se desea de una forma comprensible y sin complicaciones.

Contenido

MÓDULOS

Modulo 1: Arrancando el motor

Modulo 2: El hola Mundo

Modulo 3: Cargando una maya animada

Modulo 4: Cargando una maya estática

Modulo 5: Colisión

Modulo 6: Maya estática y Maya animada

Modulo 7: Agregando armas

PEQUEÑO CURSO DE CÓMO SE HACEN LOS VIDEOJUEGOS

1. DIME QUE JUEGAS

FPS - First Person Shooter

RTS - Real Time Strategy:

RPG - Role playing games:

SIMULADORES:

TBG - TURN BASE GAMES:

AVENTURAS GRAFICAS:

2. LO NUEVO

OPEN ENDED GAMES:

MMORPG - Massive Multiplayer Online Role Playing Game:

3. INTRODUCCIÓN A LA FILOSOFÍA DE LOS JUEGOS

4. ANALIZANDO LA IDEA

5. LENGUAJES DE PROGRAMACIÓN

6. LOS ELEMENTOS DE UN JUEGO

7. HERRAMIENTAS GRAFICAS

8. HERRAMIENTAS 3D

9. EL SONIDO

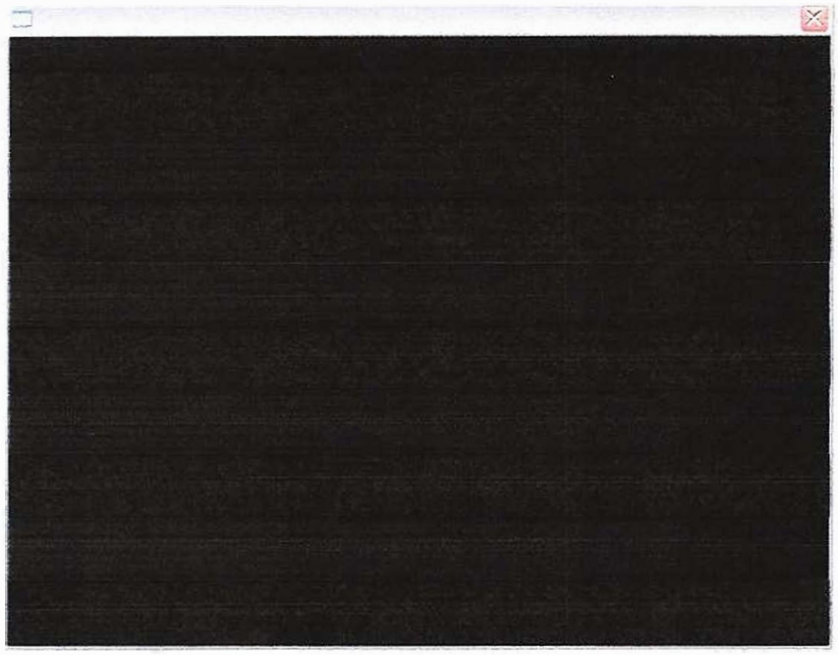
10. DIRECTX Y OPENGL

11. CABALLEROS ARRANQUEN SUS MOTORES

12. ESOS BENDITOS MODS

13. FINAL DEL CURSO

Modulo 1: Arrancando el motor



En este modulo se conocera las sentencias basicas necesarias para arrancar el motor, asi como una pequeña descripcion del trabaja de cada una de ellas.
Tenga en cuenta que para todos los proyectos CPP de Irrlicht es necesario agregar su librería principal <irrlicht.h>, de lo contrario no funcionara.

→ Apartir de aquí comienza el codigo las letras en verde son las pequeñas descripciones del funcionamiento de la sentencia y estructura. Tenga en cuenta que si pasa este codigo a un archivo CPP devera encerrar estos comentarios entre los simbolos, si los desea conservar, no necesitan estar obligatoriamente en el codigo.

(/* Comentario */) para evitar error de compilacion por informacion desconocida.

```
Esta es la libreria principal del Motor IRRLICHT ENGINE es necesaria
declararla antes de utilizar sus clases de lo contrario, nos mostrara
un mensaje de clases y namespace desconocidos.

#include <irrlicht.h>

Para poder utilizar el archivo Irrlicht.DLL,necesitamos incluir en
codigo una llamada al archivo Irrlicht.lib. Se puede agregar esta
declaracion en las opciones del proyecto pero, pero para hacerlo
fácil, es mejor agregarlo con un #pragma comment().

#pragma comment(lib,"Irrlicht.lib")
```

Se puede hacer una declaracion global de las variables mas importantes del codigo.

La variable caja en este codigo es el mas importante ya que en el se egecutara las clases y metodos del motor IRRLICHT ENGINE.

```
irr::IrrlichtDevice* caja = 0;
```

Se inicia la funcion main una funcion interna del compilador.

```
int main()
{
```

A caja le agregamos la clase "createDevice()" que es la encargada de generar los atributos de la ventana grafica del IRRLICHT ENGINE.
Su orden es: (driver de video,dimenciones X y Y de pantalla,bit de pantalla, fulscreen, sombras,sincronizacion vertical, recibidor de eventos [Por defecto lo declaramos en cero(0)])

```
      caja = irr::createDevice
             (
                irr::video::EDT_OPENGL,
                irr::core::dimension2d<irr::s32>(640,480),
                16,
                false,
                false,
                false,
                0
             );
```

Se inicia un "while", que es un bucle repetitivo mientras una condicion se cumpla el no se detendra, por lo que caja, atributo principal, es el responsable de que este bucle funcione.
Significa "Mientras que caja este corriendo realice las sentencias internas"

```
      while(caja->run())
      [
            Se manipula el driver de video, esta dentro del bucle para
            recargar el beginScene

            caja->getVideoDriver()->beginScene(true,true,0);

            El endScene permite liberar el video, para que el video se
            mantenga refrescado

            caja->getVideoDriver()->endScene();
      } Fin While

      closeDevice, se encarga de borrar de la memoria a caja para
      liberar la memoria del  equipo

      caja->closeDevice();

      La funcion main requiere devolver un valor
      return 0;
} Fin funcion main()
```

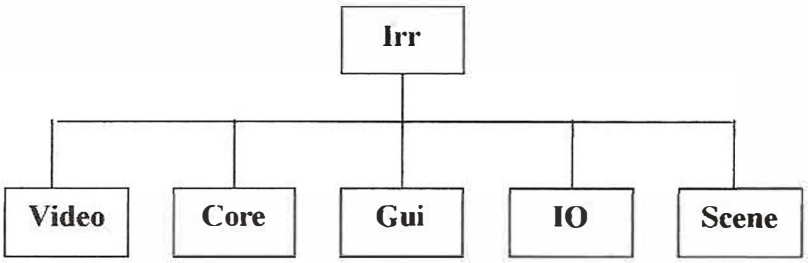
Bueno estos son los requerimientos minimos para arrancar el MOTOR, en el siguiente capitulo utilizaremos otro namespaces para agregar nuevas funciones y estructuras al motor.

Tipo de declaraciones importantes del modulo

```
Irr::irrlichtDevice * Caja;
```

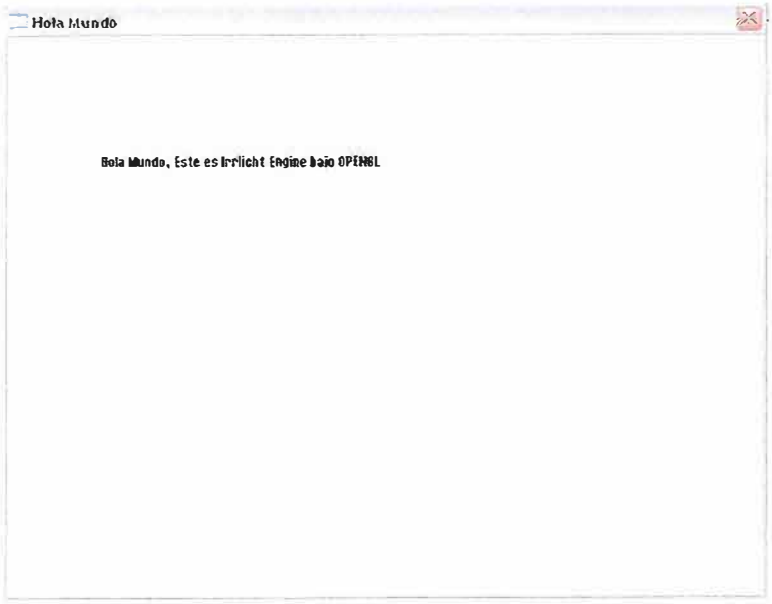
En esta variable se almacenan todas las estructuras y clases del motor IRRLICHT ENGINE, los namespaces de video, de escena, de io (entrada/salida), interfaz grafica de usuario (gui), core (vectores y matrices) tambien pueden ser accedidos desde esta variable ya que estos namespaces son sub-namespaces de irr;

Jerarquia



Se puede observar dentro el bucle while como caja (**la variable tipo irr::IrrlichtDevice**) toma control de una clase del namespace video para su debido uso. De esta forma caja también podrá utilizar las clases y estructuras de los otros namespaces.

Modulo 2: El hola mundo !!



En este modulo manipularemos el video para imprimir un mensaje en pantalla, tenga en cuenta que solo se va a explicar el nuevo codigo, si tiene alguna duda con una de las sentencias vuelva a leer el modulo anterior.

```
#include <irrlicht.h>

#pragma comment(lib,"Irrlicht.lib")

Variables globales
irr::IrrlichtDevice* caja = 0;

Se inicia la funcion main una funcion interna del compilador.
int main()
{
      caja = irr::createDevice
            (
                irr::video::EDT_OPENGL,
                irr::core::dimension2d<irr::s32>(640,480),
                16,
                false,
                false,
                false,
                0
             );


Esta sentencia caja->getVideoDriver() puede ser guardada dentro de una
variable tipo irr::video::IVideoDriver para evitar complejidad en el
codigo, se llamara para los siguientes ejemplos "dvideo" que sera la
sigla de "driver de video" se iguala a cero mientras no sea usada.

      irr::video::IVideoDriver* dvideo = 0;
```

Caja le facilita a dvideo las "clases y estructuras" de las librerias del controlador de video, teniendo en cuenta de que dvideo debe ser de tipo irr::video::IVideoDriver (Mas informacion en el manual de usuario CAPITULO 6)

```
dvideo = caja->getVideoDriver();
```

Esta sentencia caja->getGUIEnvironment()puede ser guardada dentro de una variable tipo irr::gui::IGUIEnvironment para evitar complejidad en el codigo, se llamara para los siguientes ejemplos "interfazgu" que sera la sigla de "interaz grafica de usuario" se iguala a cero mientras no sea usada..

```
irr::gui::IGUIEnvironment* interfazgu = 0;
```

Caja le facilita a interfazgu las "clases y estructuras" de las librerias de interfaz grafica de usuario, teniendo en cuenta de que interfazgu debe ser de tipo irr::gui::IGUIEnvironment (Mas informacion en el manual de usuario CAPITULO 6)

```
interfazgu = caja->getGUIEnvironment();
```

Esta sentencia caja->getSceneManager()puede ser guardada dentro de una variable tipo irr::scene::ISceneManager para evitar complejidad en el codigo, se llamara para los siguientes ejemplos "admies" que sera la sigla de "administrador de escena" se iguala a cero mientras no sea usada..

```
irr::scene::ISceneManager* admies = 0;
```

Caja le facilita a admies las "clases y estructuras" de las librerias de administrador de escenas, teniendo en cuenta de que admies debe ser de tipo irr::scene::ISceneManager (Mas informacion en el manual de usuario CAPITULO 6)

```
admies = caja->getSceneManager();
```

Hacemos una nueva delaracion se llamara skin que sera una variable tipo (Interfaz Grafica de Usuario) → irr::gui::IGUISkin.

```
irr::gui::IGUISkin* skin = 0;
```

interfazgu le facilita a skin las "clases y estructuras" de las librerias de interfaz grafica de usuario, en este caso el getSkin (Almacenador de Skin), teniendo en cuenta de que skin debe ser de tipo irr::gui::IGUISkin (Mas informacion en el manual de usuario CAPITULO 6)

```
skin = interfazgu->getSkin();
```

Hacemos una nueva delaracion se llamara font que sera una variable tipo (Interfaz Grafica de Usuario) → irr::gui::IGUIFont.

```
irr::gui::IGUIFont* font = 0;
```

interfazgu le facilita a skin las "clases y estructuras" de las librerias de interfaz grafica de usuario, en este caso el getFont (Almacenador de Font), teniendo en cuenta de que skin debe ser de tipo irr::gui::IGUIFont (Mas informacion en el manual de usuario CAPITULO 6)

```
font = interfazgu->getFont("../objetos_externos/tipoletra.bmp");
```

Verificamos que si exista font, es decir que el archivo de la ruta si existe.

```
if(font)
{
```

Al comprobar que el archivo asignado al getFont() existe, se procede a utilizar sus caracteristicas.

```
font->getDimension(L"ds");
skin->setFont(font);
}
```

Para poder imprimir datos dentro de la ventana necesitamos de las clases y estructuras de "interfaz grafica de usuario GUI" (Mas informacion en el manual de usuario CAPITULO 6)

Utilizamos la clase addStaticText con la ayuda de la variable interfazgu (variable tipo irr::gui::IGUIEnvironment), esta clase imprime en pantalla cualquier texto que sea escrito dentro de sus parentises, tenga en cuenta que las comillas son indispensables para saber que son caracteres de lo contrario no imprimira nada.

```
interfazgu->addStaticText(
             L"Hola Mundo, Este es Irrlicht Engine bajo OPENGL",
             irr::core::rect<int>(80,100,460,122),false
            );
```

Agregamos una camara y le asignamos unas cordenadas especificas para que se vea lo que se escribio con la ayuda de irr::core.

```
admies->addCameraSceneNode(
          0,
          irr::core::vector3df(0,30,-40),
          irr::core::vector3df(0,5,0)
        );

while(caja->run())
{
```

En este modulo utilizamos las caracteristicas del beginScene, para mas información consulte el capitulo 6 del manual de usuario.

```
dvideo->beginScene(
             true,
             true,
             irr::video::SColor(255,255,255,255)
           );
```

Liberamos de memoria las variables interfazgu, admies, dvideo, que almacenan las clases de gui, admies, dvideo respectivamente.

```
interfazgu->drawAll();
admies->drawAll();
dvideo->endScene();
```

```
No es obligatorio pero con esta sentencia podemos darle un nombre a la
ventana en la parte del caption (Mas informacion en el manual de
usuario Capitulo 6)

            caja->setWindowCaption(L"Hola Mundo");
      }Fin bucle while
      caja->closeDevice();
      return 0;
}Fin función main
```

Modulo 3: Maya animada

En este modulo se van a mostrar las sentencias básicas para cargar una maya animada, y los pasos a seguir para ubicarlo en la pantalla, es recomendable estudiar la librería **IanimatedMesh.h** para conocer los diferentes tipos de maya que puede cargar este motor, y sus diversas funcionalidades y clases disponibles para trabajarlos.

```
#include <irrlicht.h>

#pragma comment(lib,"Irrlicht.lib")

Variables globales

irr::IrrlichtDevice* caja = 0;
irr::video::IVideoDriver* dvideo = 0;
irr::gui::IGUIEnvironment* interfazgu = 0;
irr::scene::ISceneManager* admies = 0;


Se inicia la funcion main una funcion interna del compilador.
int main()
{
      caja = irr::createDevice(
                  irr::video::EDT_OPENGL,
                  irr::core::dimension2d<irr::s32>(640,480),
                  16,
                  false,
                  false,
                  false,
                  0
               );

      dvideo = caja->getVideoDriver();
```

```
	interfazgu = caja->getGUIEnvironment();
	admies = caja->getSceneManager();


Declaramos una variable tipo irr::scene::IAnimatedMesh que tendra el
nombre de amaya se le asigna cero mientras no se use

	irr::scene::IAnimatedMesh* amaya = 0;

A la variable amaya le asignamos la ruta de una maya animada tipo MD2
con la ayuda de el admies->getMesh("<--ruta-->"); (Mas informacion en
el documento de usuario CAPITULO 6)

	amaya = admies->getMesh("../objeto_externo/Chastity.md2");

Declaramos una variable tipo irr::scene::IAnimatedMeshSceneNode que
tendra el nombre de anodo se le asigna cero mientras no se use

	irr::scene::IAnimatedMeshSceneNode* anodo = 0;

A la variable anodo la relacionamos con amaya con la ayuda del
administrador de escena :
admies->addAnimatedMeshSceneNode(mayaseleccionada);
(Mas informacion en el documento de usuario CAPITULO 6)

	anodo = admies->addAnimatedMeshSceneNode(amaya);

	if(anodo)
	{
		Le asignamos una luz personal al nodo de la maya
		anodo->setMaterialFlag(irr::video::EMF_LIGHTING, false);

		Le asignamos a la maya una de las animaciones internas en:
		irr::scene
		anodo->setMD2Animation(irr::scene::EMAT_STAND);

		Le asignamos una textura a la maya animada
		anodo->setMaterialTexture(
		   0,
		   dvideo->getTexture("../objeto_externo/Chastity1.pcx")
		);
	}

	Le asignamos a la camara las coordenadas exactas de la maya
	animada para poder verla.(Mas informacion en el documento CAP 6)

	admies->addCameraSceneNode(0,irr::core::vector3df(0,30,-40),
		irr::core::vector3df(0,5,0));

	while(caja->run())
	{
		dvideo->beginScene(true,true,0);
		interfazgu->drawAll();
		admies->drawAll();
		dvideo->endScene();

		caja->setWindowCaption(L"Maya animada");
	}
	caja->closeDevice();
	return 0;
}
```

Modulo 4: Maya estatica

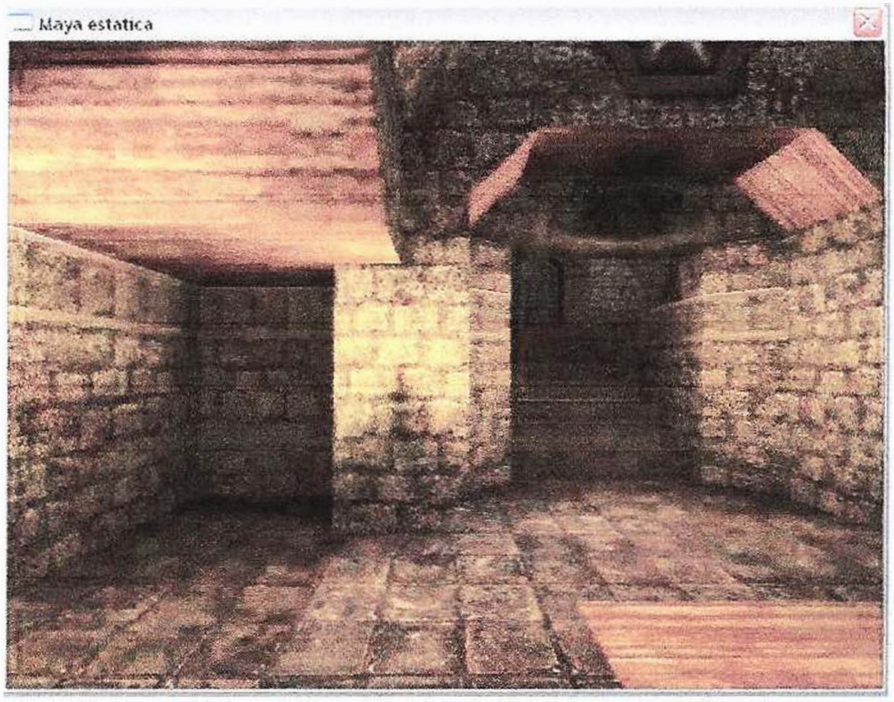
En este modulo se van a mostrar las sentencias básicas para cargar una maya estática, y los pasos a seguir para ubicarlo en la pantalla, es recomendable estudiar la librería **IanimatedMesh.h** para conocer los diferentes tipos de maya que puede cargar este motor, y sus diversas funcionalidades y clases disponibles para trabajarlos.

```
#include <irrlicht.h>

#pragma comment(lib,"Irrlicht.lib")

Variables globales

irr::IrrlichtDevice* caja = 0;
irr::video::IVideoDriver* dvideo = 0;
irr::gui::IGUIEnvironment* interfazgu = 0;
irr::scene::ISceneManager* admies = 0;


Se inicia la funcion main una funcion interna del compilador.
int main()
{
      caja = irr::createDevice(
                  irr::video::EDT_OPENGL,
                  irr::core::dimension2d<irr::s32>(640,480),
                  16,
                  false,
                  false,
                  false,
                  0
               );
```

```
dvideo = caja->getVideoDriver();
interfazgu = caja->getGUIEnvironment();
admies = caja->getSceneManager();


Declaramos una variable tipo irr::scene::IAnimatedMesh que
tendra el nombre de smaya se le asigna cero mientras no se use

irr::scene::IAnimatedMesh* smaya = 0;


El caja->getFileSystem()->addZipFileArchive(..ZIP..) permite
abrir archivos comprimidos con extencion .ZIP

caja->getFileSystem()->addZipFileArchive(
                          "../objeto_externo/map-20kdm2.pk3"
                        );


A la variable smaya le asignamos la ruta de una maya estatica
con la ayuda de el admies->getMesh("<--ruta-->");
Mas informacion en el documento CAPITULO 01

smaya = admies->getMesh("20kdm2.bsp");



Declaramos una variable tipo irr::scene::ISceneNode que tendra
el nombre de snodo se le asigna cero mientras no se use

irr::scene::ISceneNode* snodo = 0;

Se verifica que smaya sea diferente de cero
if (smaya)
      relacionamos snodo con smaya
      snodo = admies->addOctTreeSceneNode(smaya->getMesh(0));

Se verifica que snodo sea diferente de cero
if (snodo)
      se asigna la posicion con relacion a la camara
      snodo->setPosition(irr::core::vector3df(-1300,-144,-1249));

agregamos una camara al administrador de escena
admies->addCameraSceneNodeFPS();

hacemos invisible el cursor del mouse
caja->getCursorControl()->setVisible(false);

while(caja->run())
{
      dvideo->beginScene(
                true,
                true,
                irr::video::SColor(100,100,100,100)
              );

      interfazgu->drawAll();
      admies->drawAll();
      dvideo->endScene();

      caja->setWindowCaption(L"Maya estatica");
```

```
        }

        caja->closeDevice();
        return 0;
}
```

Modulo 5: Colision de Maya



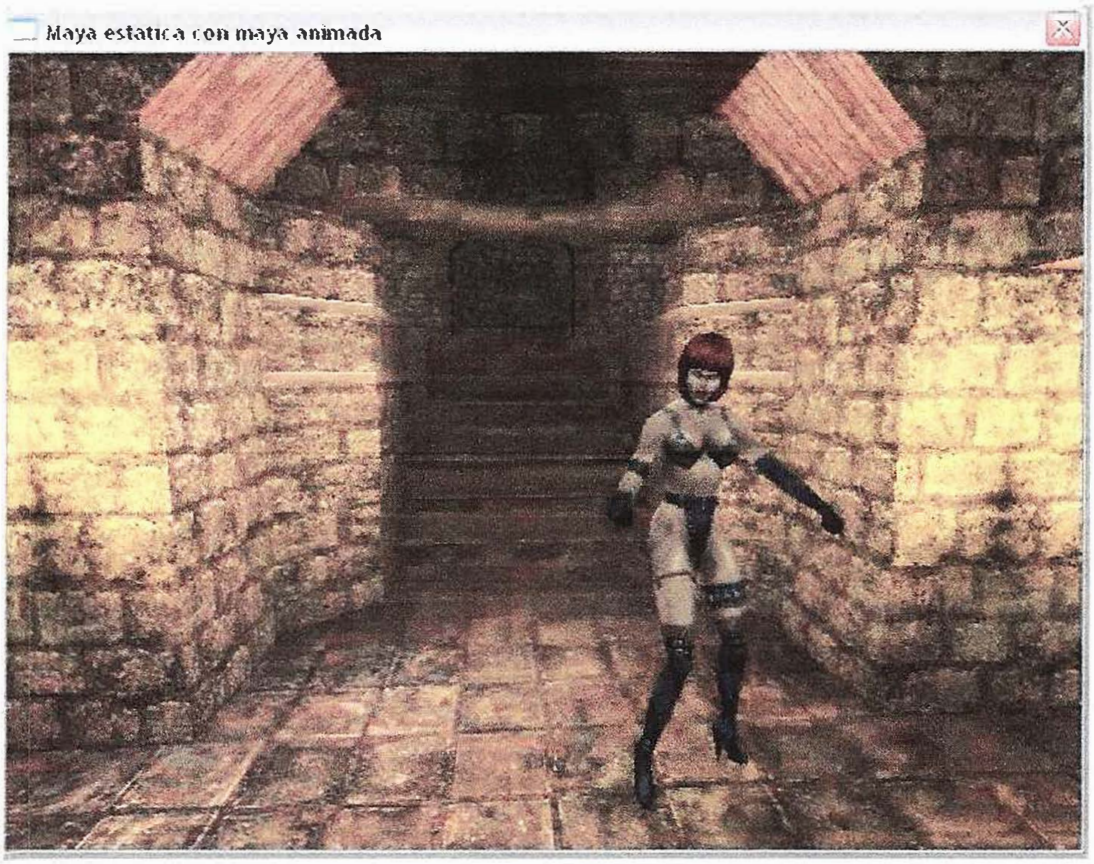
En este modulo se darán las bases para crear efectos de colisión de la cámara con respecto a la maya estática, y las clases y estructuras que deben intervenir para su funcionamiento.

```
#include <irrlicht.h>

#pragma comment(lib,"Irrlicht.lib")

Variables globales
irr::IrrlichtDevice* caja = 0;
irr::video::IVideoDriver* dvideo = 0;
irr::gui::IGUIEnvironment* interfazgu = 0;
irr::scene::ISceneManager* admies = 0;


Se inicia la funcion main una funcion interna del compilador.
int main()
{
      caja = irr::createDevice(
                  irr::video::EDT_OPENGL,
                  irr::core::dimension2d<irr::s32>(640,480),
                  16,
                  false,
                  false,
                  false,
                  0
                );


      dvideo = caja->getVideoDriver();
      interfazgu = caja->getGUIEnvironment();
```

```
admies = caja->getSceneManager();



Declaramos una variable tipo irr::scene::IAnimatedMesh que
tendria el nombre de smaya se le asigna cero mientras no se use


irr::scene::IAnimatedMesh* smaya = 0;

caja->getFileSystem()->addZipFileArchive(
                          "../objeto_externo/map-20kdm2.pk3"
                        );


A la variable smaya le asignamos la ruta de una maya estatica
con la ayuda de el admies->getMesh("<-- ruta -->");
(Mas informacion en el documento de usuario CAPITULO 6)

smaya = admies->getMesh("20kdm2.bsp");


Declaramos una variable tipo irr::scene::ISceneNode que tendria
el nombre de snodo se le asigna cero mientras no se use


irr::scene::ISceneNode* snodo = 0;


if (smaya)
      snodo = admies->addOctTreeSceneNode(smaya->getMesh(0));


Declaramos una variable tipo irr::scene::ITriangleSelector con
el nombre de selector se le asigna cero mientras no se use
(Mas informacion en el documento de usuario CAPITULO 6)


irr::scene::ITriangleSelector* selector = 0;


if (snodo)
{
      snodo->setPosition(irr::core::vector3df(-1370,-130,-1400));

      A selector se le asigna las caracteristicas de la maya
      selector = admies->createOctTreeTriangleSelector(
                              smaya->getMesh(0),
                              snodo,
                              128
                            );

      se relaciona snodo con selector
```

```
		snodo->setTriangleSelector(selector);
		selector->drop();
	}

	Declaramos una variable tipo irr::scene::ICameraSceneNode con el
	nombre de camara se le asigna cero mientras no se use.
	(Mas informacion en el documento de usuario CAPITULO 6)

	irr::scene::ICameraSceneNode* camara = 0;

	camara = admies->addCameraSceneNodeFPS(
				0,
				100.0f,
				150.0f,
				-1,
				0,
				0,
				true);
	camara->setPosition(irr::core::vector3df(-100,50,-150));

	Declaramos una variable tipo irr::scene::ISceneNodeAnimator con
	el nombre de animador se le asigna cero mientras no se use
	(Mas informacion en el documento de usuario CAPITULO 6)

	irr::scene::ISceneNodeAnimator* animador = 0;

	Animador contendra la clase createCollisionResponseAnimator, el
	cual hara que la camara colisiones con la maya, teniendo las ca-
	Racteristicas de la maya con un selector
	(Mas informacion en el documento de usuario CAPITULO 6)

	animador = admies->createCollisionResponseAnimator(
				selector,
				camara,
				irr::core::vector3df(30,50,30),
				irr::core::vector3df(0,-3,0),
				irr::core::vector3df(0,50,0)
			);

	camara->addAnimator(animador);
	animador->drop();

	caja->getCursorControl()->setVisible(false);

	while(caja->run())
	{
		dvideo->beginScene(
				true,
				true,
				irr::video::SColor(100,100,100,100)
			);

		interfazgu->drawAll();
		admies->drawAll();
		dvideo->endScene();

		caja->setWindowCaption(L"Maya estatica");
	}
	caja->closeDevice();
	return 0;
}
```

Modulo 6: Colision Maya Estatica y Una maya animada



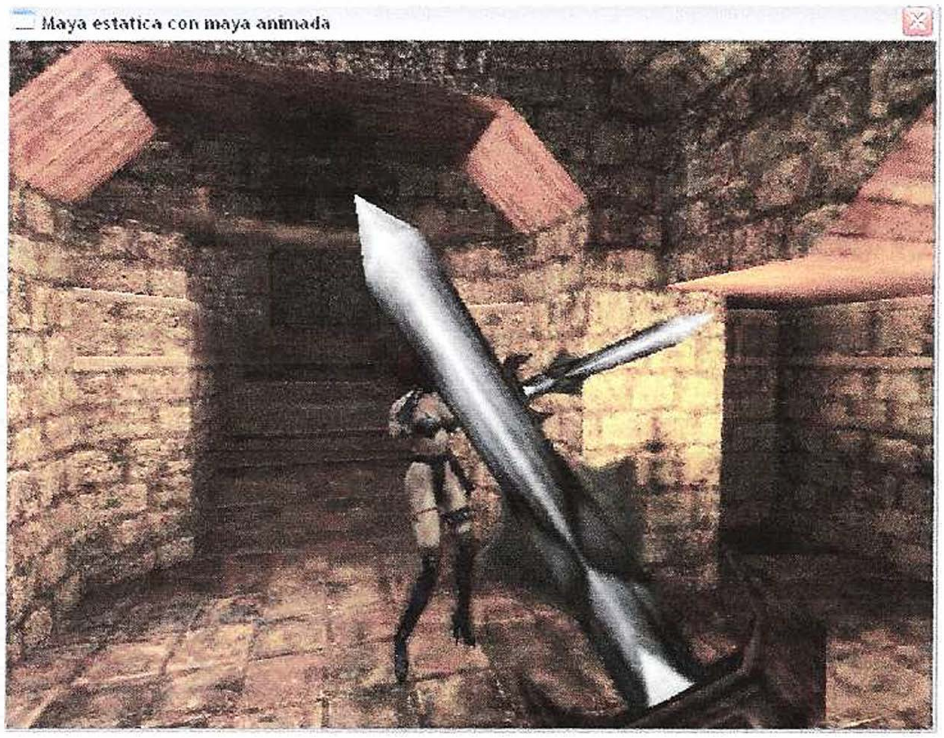
En este modulo se agregara una maya animada a una maya estática, además de dejar validado los efectos de colisión, de la cámara con la maya estática

```
#include <irrlicht.h>

Si no deseamos agregar una carpeta con mayas y texturas a cada
proyecto que se crea, se puede agregar al codigo un la sentencia
#define variable ruta , para evitar tantos datos repetidos y tenerlos
ubicados en un solo lugar para su uso.

#define Dir_ObjExternos "C:/Objetos_Externos/"

#pragma comment(lib,"Irrlicht.lib")

Variables globales

irr::IrrlichtDevice* caja = 0;

irr::video::IVideoDriver* dvideo = 0;

irr::gui::IGUIEnvironment* interfazgu = 0;
```

```
irr::scene::ISceneManager* admies = 0;

irr::scene::ITriangleSelector* selector = 0;

irr::scene::ICameraSceneNode* camara = 0;

irr::scene::ISceneNodeAnimator* animador = 0;

irr::scene::IAnimatedMesh* smaya = 0;

irr::scene::IAnimatedMesh* amaya = 0;

irr::scene::IAnimatedMeshSceneNode* anodo = 0;

irr::scene::ISceneNode* snodo = 0;


Se usara la funcion main una funcion propia del compilador.
int main()
{
      caja = irr::createDevice(
                irr::video::EDT_OPENGL,
                irr::core::dimension2d<irr::s32>(640,480),
                16,
                false,
                false,
                false,
                0
               );


      dvideo = caja->getVideoDriver();
      interfazgu = caja->getGUIEnvironment();
      admies = caja->getSceneManager();

      caja->getFileSystem()->addZipFileArchive(
                              Dir_ObjExternos"map-20kdm2.pk3"
                             );

      smaya = admies->getMesh("20kdm2.bsp");

      if (smaya)
            snodo = admies->addOctTreeSceneNode(smaya->getMesh(0));

      if (snodo)
      {
            snodo->setPosition(irr::core::vector3df(-1370,-130,-1400));

            selector = admies->createOctTreeTriangleSelector(
                              smaya->getMesh(0),
                              snodo,
                              128
                        );

            snodo->setTriangleSelector(selector);
            selector->drop();
      }
```

```
camara = admies->addCameraSceneNodeFPS(
                        0,
                        100.0f,
                        150.0f,
                        -1,
                        0,
                        0,
                        True
                    );

camara->setPosition(irr::core::vector3df(-100,50,-150));


      animador = admies->createCollisionResponseAnimator(
                        selector,
                        camara,
                        irr::core::vector3df(30,50,30),
                        irr::core::vector3df(0,-3,0),
                        irr::core::vector3df(0,50,0)
                    );

camara->addAnimator(animador);
animador->drop();


amaya = admies->getMesh(Dir_ObjExternos "Chastity.md2");

anodo = admies->addAnimatedMeshSceneNode(amaya);

if(anodo)
{
      anodo->setMaterialFlag(irr::video::EMF_LIGHTING, false);
      anodo->setScale(irr::core::vector3df(2.2,2.2,2.2));
      anodo->setMD2Animation(irr::scene::EMAT_STAND);
      anodo->setPosition(irr::core::vector3df(70,-15,50));
      anodo->setMaterialTexture(0,
                                dvideo->getTexture(
                                Dir_ObjExternos "Chastity1.pcx"
                                )
                               );
}

caja->getCursorControl()->setVisible(false);

while(caja->run())
{
      dvideo->beginScene(
                  true,
                  true,
                  irr::video::SColor(100,100,100,100)
                );

      interfazgu->drawAll();
      admies->drawAll();
      dvideo->endScene();

      caja->setWindowCaption(L"Maya estática con maya animada");
}
caja->closeDevice();
return 0;
}
```

Modulo 7: Agregando arma



En este modulo se a agregar un arma a la cámara el efecto PRIMERA PERSONA, ya aquí nos adentramos mas a las bases para el desarrollo de videojuegos, a partir de aquí se van hacer generalizaciones para aprender a programar videojuegos.

Ahora se va a tratar de categorizar el codigo, por ejemplo en este ejemplo haremos una libreria sencilla que solo contendra las variables del programa, lo llamaremos en este ejemplo variables.h

```
/**********variables.h **************************************/


//Variables globales
irr::IrrlichtDevice* caja = 0;

irr::video::IVideoDriver* dvideo = 0;

irr::gui::IGUIEnvironment* interfazgu = 0;

irr::scene::ISceneManager* admies = 0;

irr::scene::ITriangleSelector* selector = 0;

irr::scene::ICameraSceneNode* camara = 0;

irr::scene::ISceneNodeAnimator* animador = 0;

irr::scene::IAnimatedMesh* smaya = 0;
```

```
irr::scene::IAnimatedMesh* amaya = 0;

irr::scene::IAnimatedMeshSceneNode* anodo = 0;

irr::scene::IAnimatedMesh* amaya2 = 0;

irr::scene::IAnimatedMeshSceneNode* anodo2 = 0;
irr::scene::IAnimatedMeshSceneNode* anodo3 = 0;

irr::scene::ISceneNode* snodo = 0;

/*****************FIN DE variables.h ********************************/



/*****************Principal.h*****************************************/



#include <irrlicht.h>
#include "variables.h"

#define Dir_ObjExternos "C:/Objetos_Externos/"

#pragma comment(lib,"Irrlicht.lib")
```

Ahora vamos a agregar un manejador de eventos, por el momento solo tendrá la función de finalizar la aplicación al presionar la tecla ESCAPE, para interiorizarse con este manejador de eventos, revisar la librería IEventReceiver.h para conocer su funcionamiento, también necesita saber algo de programación orientada a objetos, si no dispone de esta información el CD trae consigo en la carpeta Utilidades, manuales PDF para aprender C++ Básico (para iniciadores), y el C++ Avanzado (programación orientada a objetos)

```
class MisEventos : public irr::IEventReceiver
{

   bool isDown[irr::KEY_KEY_CODES_COUNT];

   virtual bool OnEvent(irr::SEvent event)
   {
          if (event.EventType == irr::EET_KEY_INPUT_EVENT)
          isDown[event.KeyInput.Key] = event.KeyInput.PressedDown;
              switch (event.KeyInput.Key)
              {
                    case irr::KEY_ESCAPE:
                          caja->closeDevice();
                          return true;
              }

       return false;
   }
};
```

```
Se inicia la función main, una función interna del compilador.
int main()
{


      Aquí llamamos a la clase que se creo, la asignamos a una
      variable llamada receptor, puede ser cualquier nombre, y la
      asignamos al recibidor de eventos de caja

      MisEventos receptor;

      caja = irr::createDevice(
                irr::video::EDT_OPENGL,
                irr::core::dimension2d<irr::s32>(640,480),
                16,
                false,
                false,
                false,
                &receptor //Aqui agregamos el manejador de eventos
              );

      dvideo = caja->getVideoDriver();
      interfazgu = caja->getGUIEnvironment();
      admies = caja->getSceneManager();

      caja->getFileSystem()->addZipFileArchive(
                              Dir_ObjExternos"map-20kdm2.pk3"
                             );

      smaya = admies->getMesh("20kdm2.bsp");

      if (smaya)
            snodo = admies->addOctTreeSceneNode(smaya->getMesh(0));

      if (snodo)
      {
            snodo->setPosition(irr::core::vector3df(-1370,-130,-1400));

            selector = admies->createOctTreeTriangleSelector(
                                    smaya->getMesh(0),
                                    snodo,
                                    128
                                   );

            snodo->setTriangleSelector(selector);
            selector->drop();
      }

      camara = admies->addCameraSceneNodeFPS(
                              0,
                              100.0f,
                              150.0f,
                              -1,
                              0,
                              0,
                              True
                            );
      camara->setPosition(irr::core::vector3df(-100,50,-150));
      camara->getProjectionMatrix();
```

```
        animador = admies->createCollisionResponseAnimator(
                            selector,
                            camara,
                            irr::core::vector3df(30,50,30),
                            irr::core::vector3df(0,-3,0),
                            irr::core::vector3df(0,50,0)
                        );
camara->addAnimator(animador);
animador->drop();

amaya = admies->getMesh(Dir_ObjExternos"Chastity.md2");

anodo = admies->addAnimatedMeshSceneNode(amaya);

if(anodo)
{
        anodo->setMaterialFlag(irr::video::EMF_LIGHTING, false);
        anodo->setScale(irr::core::vector3df(2,2,2));
        anodo->setMD2Animation(irr::scene::EMAT_STAND);
        anodo->setPosition(irr::core::vector3df(70,-15,50));
        anodo->setRotation(irr::core::vector3df(0,0,0));
        anodo->setMaterialTexture(0,
                                  dvideo->getTexture(
                                  Dir_ObjExternos"Chastityi.pcx"
                                  )
                                );
}


amaya2 = admies->getMesh(Dir_ObjExternos"sword.md2");

anodo2 = admies->addAnimatedMeshSceneNode(amaya2);
anodo3 = admies->addAnimatedMeshSceneNode(amaya2);
anodo3->setMaterialTexture(0,
                           dvideo->getTexture(
                             Dir_ObjExternos"sword.bmp"
                           )
                          );
anodo3->setMD2Animation(irr::scene::EMAT_STAND);
anodo3->setScale(irr::core::vector3df(1,1,1));
anodo3->setMaterialFlag(irr::video::EMF_LIGHTING, false);
anodo3->setPosition(irr::core::vector3df(0,20,0));

Aqui le decimos a la maya animada espada que le va a pertenecer
a la maya animada muneca chastity
(Ver manual de usuario CAPITULO 6)
anodo3->setParent(anodo);

if(anodo2)
{
        anodo2->setMaterialFlag(irr::video::EMF_LIGHTING, false);
        anodo2->setMD2Animation(irr::scene::EMAT_STAND);
        anodo2->setPosition(irr::core::vector3df(20,-30,0));
        anodo2->setRotation(irr::core::vector3df(190,-15,150));
        anodo2->setScale(irr::core::vector3df(2,2,2));
        anodo2->setMaterialTexture(0,
                                   dvideo->getTexture(
                                       Dir_ObjExternos"sword.bmp"
                                    )
                                   );
}
```

```
	//Aqui le decimos a camara que espada va a ser su hijo y va a
	//estar ligado a el (Ver Manual de usuario CAPITULO 6)
	camara->addChild(anodo2);


	caja->getCursorControl()->setVisible(false);

	while(caja->run())
	{
		dvideo->beginScene(
				true,
				true,
				irr::video::SColor(100,100,100,100)
			);
		interfazgu->drawAll();
		admies->drawAll();
		dvideo->endScene();

		caja->setWindowCaption(L"Maya estatica con maya animada");
	}
	caja->closeDevice();
	return 0;
}

/*********** FIN principal.h **********************************/
```

1. Dime que juegas

En este primer capitulo indicaremos como están divididos los diferentes **géneros** que existen actualmente en el mercado de los juegos. No he indicado todos, pero si la gran mayoría de los que existen actualmente. Día a día inclusive se generan nuevos géneros pero la tendencia del mercado nos demuestra que cuando sale un genero que tiene éxito enseguida todas las empresas se abocan a tal y durante unos años salen juegos similares unos a otros, hasta que BUM! Un nuevo genero nace y hace puedo continuar la historia por siempre

FPS - First Person Shooter: acción en primera persona, Quake, Half life, Unreal les suena. Pues sí. En estos juegos lo fundamental es matar todo bicho que camina y justamente bichos, mutantes y otros suelen ser nuestros enemigos. Si bien existieron juegos que innovaron en el género, la idea básica es la misma: ir desde A a B resolviendo puzzles sencillos, matando todo lo que se nos cruza en el camino con todas las armas que disponemos para llegar a la victoria. Hubo juegos que le dieron una vuelta más de tuerca al género, que les agregaron estrategia.

RTS - Real Time Strategy: son los juegos de estrategia en tiempo real donde se suele ver todo el campo de batalla, nuestras unidades en forma pequeña y donde predomina la estrategia por sobre las balas en si. Juegos como Warcraft y Age of Empire fueron los que le dieron el concepto general de los juegos y a partir de allí surgieron cientos y cientos. Estos juegos poseen interesantes sistema de Inteligencia Artificial y un arte grafico y sonidos excelentes

RPG - Role playing games: juegos de Rol, donde uno, comúnmente un personaje único, adquiere diferentes tipos de habilidades, magias, experiencia, lo que sea para fortalecer el personaje. Entablar conversaciones, comprar elementos para mejorar el personaje, mundos gigantes, una cierta libertad de decisión hacia donde ir son las características de estos juegos. Actualmente juegos como Knight of the Old Republic demuestran a que nivel estos juegos pueden llegar.

SIMULADORES: en este rubro se encuentran todos los juegos que simulan algo de la vida real, ya sea un equipo de fútbol como así también un avión de combate o un tren.

Suelen ser reconocidos por dos aspectos fundamentales, la calidad del arte de estos juegos (sonido, gráficos, etc.) y el nivel de realismo que logren alcanzar estos.

TBG - TURN BASE GAMES: son juegos similares a los RTS pero con la diferencia que no son en tiempo real sino por turnos, siguiendo el concepto de los juegos de mesa. Suelen ser juegos altamente estratégicos y cada uno debe esperar su turno para jugar y luego debe observar que movimientos hizo el adversario. Actualmente la gama de juegos de este tipo están siendo dejados de lado. Sin embargo juegos como Medieval Total War lo han combinado con los RTS (estrategia por turnos, batallas en tiempo real)

AVENTURAS GRAFICAS: solían ser pantallas estáticas donde se veía el personaje en tercera persona y carecían de acción. Las pantallas solían tener complejísimos puzzles para resolver. Lucarts y Sierra eran las empresas que generaban los mas exitosos juegos del genero (Larry, Monkey Island). Hubo un juego (Alone in the Dark) que llevo los complicados puzzles y el concepto de aventura grafica al 3D.

2. Lo nuevo

Los nuevos géneros que han aparecido en los últimos tiempos y que han hecho furor.

OPEN ENDED GAMES: el gran inventor del género fue sin lugar a dudas GTA 3 (grand theft auto). El concepto de estos juegos es que son de estilo muy libre donde el jugador decide donde ir. Si bien tienen una historia central, el jugador puede en cualquier momento dejarla de lado para seguir su juego en otra parte. Es uno de los últimos géneros que se ha creado. Los juegos de este tipo tienen sus bases en los juegos de Rol.

MMORPG - Massive Multiplayer Online Role Playing Game: el concepto que se sigue en estos juegos es similar al de los juegos de rol pero los mundos que intervienen en el juego son GIGANTES y puede haber miles de personas jugándolo al mismo tiempo a través de Internet.

Existen varios géneros más e inclusive existen juegos que por si solos pueden ser considerados un género, pero los fundamentales y los que actualmente se encuentran en auge son los que he mencionado anteriormente

3. Introducción a la filosofía de los juegos

Ahora si, adentrándonos en el área que nos compete, dejando ABSOLUTAMENTE de lado todos los aspectos técnicos, incluyendo, por que no, la computadora. Primero debemos analizar lo fundamental del juego: **La idea**

Ningún juego parte sino de una idea y lo mejor a la hora de programar un juego es un planeamiento concienzudo. Saber que es lo que queremos hacer es el primer paso y después viene el como vamos a hacerlo. En este punto quizás es mejor papel y lápiz que una computadora en si.

Aquí vale todo, apoyándose en cualquier experiencia que tengan anteriormente debemos tener bien en claro en la cabeza (mejor en papel) que queremos hacer. Debemos preguntarnos cosas que parecen triviales pero que luego serán fundamentales. Cosas como: ¿Cómo será el juego? ¿Será un juego de tiros, de estrategia, de puzzles, de cartas, de que? ¿En torno a que se desarrollara el juego? ¿Será un personaje, una nave, varios personajes? ¿La misión principal será destruir naves, descubrir un complejo complot político, pilotar un avión?

Utilizar textos, otros juegos como referencias sobre lo bueno y lo malo, lo que nos gusta de cada uno, realizar dibujos, definir como va a interactuar el jugador con el juego en si son los aspectos fundamentales a tener en cuenta en esta etapa.

Esta es, aunque parezca mentira, la tarea más difícil a la hora de desarrollar un juego. Un caso típico del nacimiento de una idea es "Me gustaría un juego de....." y a partir de allí establecer los conceptos y dejar volar nuestra imaginación.

Uno de los juegos mas vendidos de la historia y que ha sufrido cientos de versiones es por ejemplo el Tetris. Un juego que una vez definido el concepto, los aspectos técnicos son relativamente sencillos: "quiero hacer un juego que mediante formas de unas especie de ladrillos vayan cayendo uno sobre otro, si toda una línea completa se forma, la misma desaparece, ahora si la pared llega a una cierta altura, el jugador pierde". ¿Quien iba a creer que esta idea seria un éxito comercial de más de dos décadas?

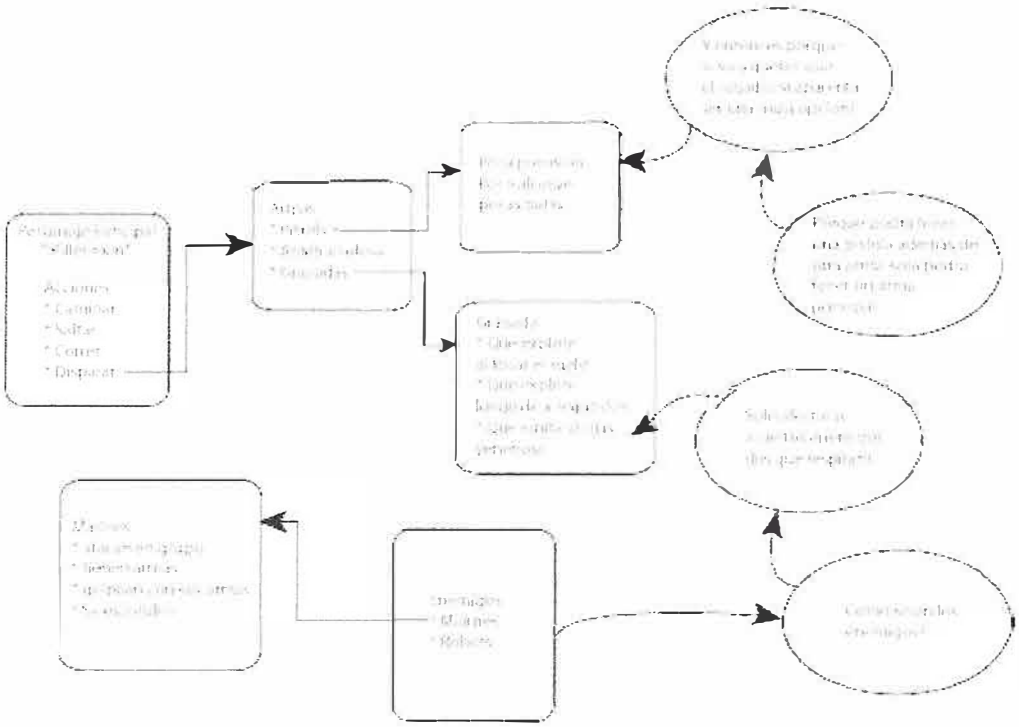
Adquirir una idea lo hace la experiencia misma. Y adquirir la idea para un juego que mejor que jugando otros juegos para analizarlos, para ver en que son buenos, en que

fallan, que les faltaría para ser perfectos si no lo son. Por experiencia les digo que una vez que conocen como están hechos los juegos, jugarlos se torna mucho más emocionante que el juego mismo y el juego en si se convierte en algo de segundo plano.

Siempre me pasa que durante un juego me quedo pensando sobre aspectos de cómo lo han hecho algunas cosas y mas de una vez el juego en si pierde su prioridad y me digo a mi mismo "seguro que esto lo hicieron así.". Inclusive me ha pasado que me he quedado en el área de un juego simplemente para observar sus texturas, su física, su IA. Nuevamente les digo es algo realmente emocionante.

4. Analizando la idea

Una vez definida la historia, nos podemos apoyar en las herramientas que existen para analizar y diseñar juegos. Algunas son muy interesantes para poner en práctica. Estas técnicas si bien tienen un fundamento teórico, cada uno las adapta a su conveniencia. Supongamos que queramos realizar un juego tipo FPS. Entonces comenzaremos a analizar cada uno de sus elementos como un Objeto.



Crearemos así el **Objeto Jugador**, en el indicaremos que puede hacer (saltar, caminar, correr, usar armas, abrir puertas, tirar granadas) esto a su vez nos dará la pauta de otros

objetos del juego. **Armas**: pistola, ametralladora, gancho. **Granadas**: detonación, dejar ciego al enemigo. Luego nos seguimos adentrando a cada elemento. **Pistola**: pocas balas, poco alcance, poca precisión....

Así podemos seguir y como pueden ver en el grafico una cosa lleva a la otra. De esta manera a la hora de programar ya tenemos algunas referencias básicas (tendremos que programar que el jugador pueda corre, pueda disparar, que las armas tengan cierto alcance, que tengan cierta precisión, que los marines puedan esconderse, etc.)

Estos elementos son los que formaran parte del desarrollo. Mas adelante veremos como se programan los juegos y entenderán un poco más para que sirva esta forma de pensar las cosas.

5. Lenguajes de programación

Los lenguajes de programación son muchos y en cada uno de los mismos se puede hacer un juego. Ya sea Java, C, Visual Basic o el que se desee. Sin embargo los lenguajes poseen dos tipos de clasificaciones. El tipo de nivel y su generación. Cabe aclarar que la idea de los diferentes tipos de lenguajes se debe a la necesidad de simplificar la tarea de programar y de utilizar para el mismo un lenguaje que se acerque lo más posible al humano.

Con respecto a los niveles encontramos de 4 tipos.

1.- Lenguaje Maquina: es el lenguaje de los unos y ceros. Es decir como procesan la información los distintos tipos de dispositivos de la computadora. Un lenguaje que entiende la maquina perfectamente pero que es muy complicado de programar.

2.- Lenguaje de Bajo nivel: Lenguajes que dependen de la arquitectura que se este manejando. En ellos se poseen instrucciones básicas que son traducidas a lenguaje maquina. Su programación no es simple, pero se obtiene un control absoluto de los dispositivos. El ensamblador es el ejemplo típico

3.- Lenguaje de Alto nivel: Son independientes de la máquina y se pueden utilizar en cualquier computadora. Los lenguajes de más alto nivel no ofrecen necesariamente mayores capacidades de programación, pero si ofrecen una interacción

programador/computadora más avanzada. Cuanto más alto es el nivel del lenguaje, más sencillo es comprenderlo y utilizarlo. El C es el ejemplo típico de estos lenguajes

4.- Lenguaje de Muy Alto nivel: En el se trata de obtener un dialogo mas "humano" con la maquina. Son los lenguajes orientados a objetos como C++, Java o Visual Basic

En cada nuevo nivel se requieren menos instrucciones para indicar a la computadora que efectúe una tarea en particular. Pero los lenguajes de alto nivel son sólo una ayuda para el programador. Un mayor nivel significa que son necesarios menos comandos, debido a que cada comando o mandato de alto nivel reemplaza muchas instrucciones de nivel inferior

En cuanto a su generación, esta clasificación se basa en la línea de tiempo histórica de los lenguajes de programación.

- *Primera Generación:* lenguaje de máquina 1940-1950. Consistía en sucesiones de dígitos binarios. Aún en la actualidad, es el único lenguaje interno que entiende la computadora; los programas se escriben en lenguajes de mayor nivel y se traducen a lenguaje de máquina.

- *Segunda Generación:* lenguajes ensambladores fines 1950. En lugar de usar códigos binarios, las instrucciones se representan con símbolos fáciles de reconocer, conocidos como mnemotécnicos. Aún se utilizan estos lenguajes cuando interesa un nivel máximo de eficiencia en la ejecución o cuando se requieren manipulaciones intrincadas.

- *Tercera Generación:* Años '60. Los lenguajes de esta generación se dividen en tres categorías, según se orienten a: Procedimientos: la forma en la que se programan. Problema: El problema que intentan resolver. Objeto: Como se encarara el objeto a desarrollar. Lenguajes como COBOL, FORTRAN y Basic son los ejemplos de esta generación

- *Cuarta Generación:* Las características generales de los lenguajes de cuarta generación son: Uso de frases y oraciones parecidas al inglés para emitir instrucciones.

No operan por procedimientos, por lo que permiten a los usuarios centrarse en lo que hay que hacer no en cómo hacerlo. Al hacerse cargo de muchos de los detalles de cómo

hacer las cosas, incrementan la productividad. Ejemplo de esta generación son los lenguajes visuales.

El desarrollo de cada una de las generaciones fue de la mano del desarrollo del hardware.

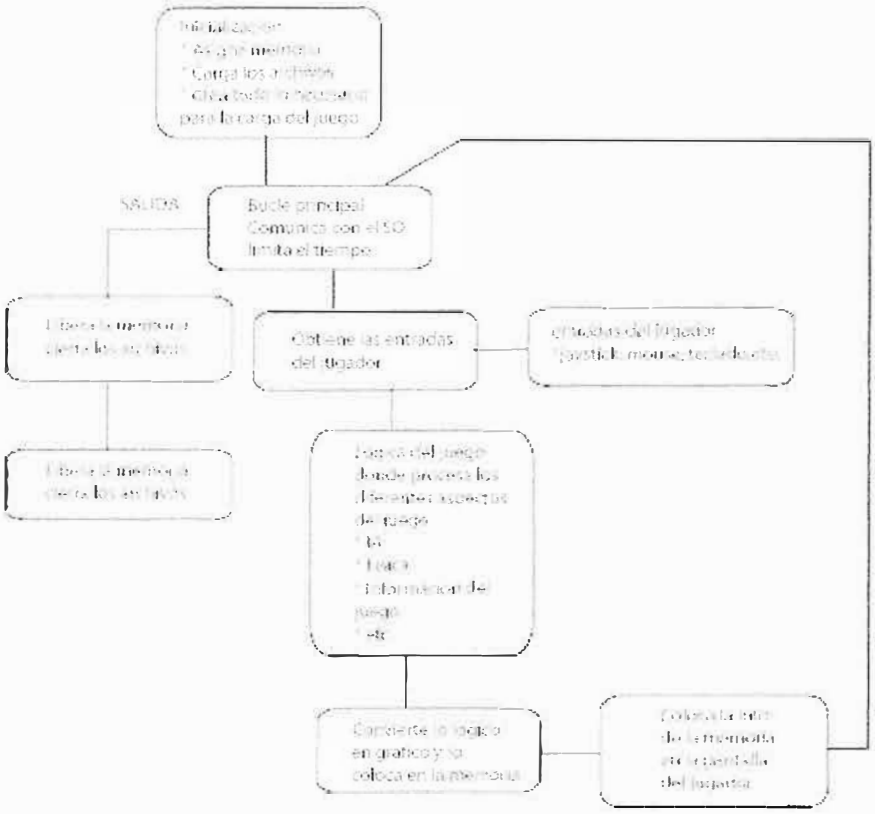
6. Los elementos de un juego

Independientemente de que juego estemos creando, algunos parámetros basicos son tenidos en cuenta en todos los juegos. Ya sea que estemos creando un Pong o un Tetris como que estemos desarrollando un juego de ultima generacion. Todos poseen una estructura similar y todos poseen algunos elementos similares. En diferentes magnitudes claro esta.

El elemento mas importante es la paciencia. Si han comenzado a leer este curso creyendo que al final iban a programar un Jedi Knight olvidenlo. Prestando mucha atención van a entender como encarar un Tetris, un Pong y si son bastante astutos un Pacman o un Galaxian. El resto de los juegos, lamento informarles, esta fuera del alzance de la escritura de este curso (y de muchos otros cursos mas). Piensen que 12 personas se demoraron 12 meses trabajando 8 horas para obtener por ejemplo un juego como Age of Empires. Que en comparación a los juegos actuales, parece un juego extremadamente básico. Si les puedo asegurar que al final del curso sabrán como estan hechos estos juegos y que se necesita aprender para hacerlos.

Estructura

Los juegos más sencillos poseen una estructura de programación similar. Si ven algún código de un programa completo sencillo observaran que tienen una estructura similar entre si.



En resumidas cuenta, al ejecutar el juego, reserva el espacio de memoria que va a necesitar, abre todas las filas que necesita (de grafico, de sonido, etc) y entra en un ciclo que solo sale cuando se termina el juego. En dicho ciclo obtiene los ingresos del usuario (vía teclado, mouse,etc), realiza los cálculos necesarios (IA, física, movimientos) y los vuelca en la memoria. Esta memoria se la conoce como buffer. Una vez en el buffer es presentado a la pantalla y se vuelve al inicio del bucle.

7. Herramientas graficas

Antes que nada, una vez llegada a la selección de herramientas se debe tener en consideración la idea del juego y el punto más representativo. Si el mismo será 2D o 3D. En el primer caso hablamos de herramienta para el desarrollo de los sorites del juego. En esta tenemos que considerar 2 programas diferentes. Los programas de diseño y retoque fotográfico como Adobe Photoshop y los de animación basados en sprite como el viejo animador pro. Un sprite es un elemento en una determinada posición (ya veremos mas adelante que significa)

Sin necesidad de entrar en el detalle de que es un sprite. En lo que respecta a 2D la técnica de animación de los personajes o elementos es la misma que se observa en los Dibujos animados tradicionales. Para realizar un movimiento, supongamos elevar un brazo, debemos hacer uno a uno los dibujos que muestran la trayectoria del brazo. Si realizamos 2 dibujos por ejemplo tendremos solamente 2 estados (brazo en posición inicial, brazo en posición final).

Si a estos 2 le agregamos mas estados obtenemos una mayor fluidez. Ahora bien, el ojo humano capta 24 cuadros por segundo, por lo que dibujar 30 estados generaría una excelente fluidez en los movimientos. Más de esta cantidad de cuadros, variaría la fluidez casi imperceptiblemente para el ojo humano.

Supongamos que estamos desarrollando un juego de plataformas, en el cual tenemos un personaje y deseamos que este tenga una función de correr. Debemos crear la serie de "cuadros" que fueran necesarios para su movimiento. Estos cuadros comúnmente se guardan en una misma imagen (todos con el mismo tamaño, uno al lado del otro) y luego se desarrollan las herramientas necesarias para cargar estos en memoria, por ejemplo en una matriz. En cada campo de la matriz se encuentra un sector de la imagen

Que corresponde a un estado del personaje:

Un conjunto de sprites del clásico Megaman

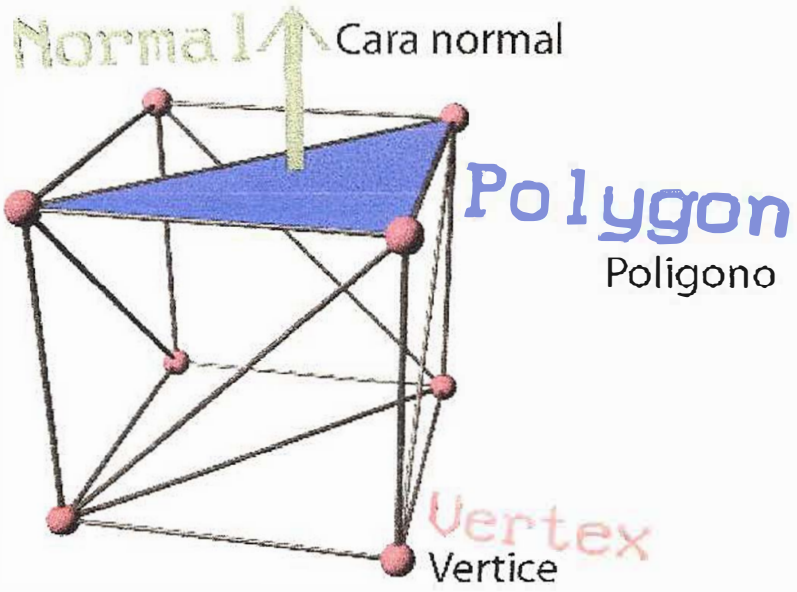
Si luego ejecutamos una función tan sencilla como la de recorrer una matriz y mostrarlo en pantalla, lograremos que el personaje posea movimiento. Así de "sencillo" es el proceso. De mas esta decir que fue "robado" de los mismos dibujos animados. Un detalle a aclarar es que solamente queremos que muestre el personaje y no como si fuera un cuadrado por lo que hay áreas de la imagen que tienen un color que luego se le indica que este color es transparente.

Como mencionamos antes, existen programas que facilitan esta engorrosa tarea de dibujar varios estados del personaje para simular movimiento.

8. Herramientas 3D

Cuando hablamos de 3D, no se dejan de lado las herramientas bidimensionales ya que algunos aspectos del juego todavía requieren de ellas. Entre los diferentes aspectos se destaca la necesidad de aplicarle una textura a los objetos. Ya se descarta por completo que un juego en 3D muestre sus figuras geométricas "desnudas" sin proveerles detalles que agraden al ojo humano. E inclusive esto mejora el rendimiento. Ya volveremos al tema.

No voy a entrar en detalle con respecto a los objetos que se manejan en 3D. Esto no es el motivo de este curso, les agregue una imagen para que puedan verlos y sepan más o menos de que estamos hablando cuando mencionamos palabras como Polígonos, caras, vértices, aristas, etc.



Elementos de un objeto 3D

Las herramientas 3D que existen en el mercado son innumerables. Las hay desde gratuitas hasta paquetes que superan los miles de dólares en valor. Si estamos hablando de un trabajo profesional, paquetes como 3D MAX, MAYA y Lightwave entran en juego (paquetes fuera del alcance del bolsillo de un novato). Pero han visto la luz programas como Wings3d (gratuito) o como Milkshape (de aproximadamente 30 dólares) que cubren la necesidad de los novatos en materia del desarrollo de figuras

geométricas. Inclusive este último mencionado posee un altísimo auge dada la compatibilidad de formatos que posee.

Como hemos mencionado antes, las texturas mejoran el rendimiento de los juegos y esto se debe a que mediante ellas, se puede "engañar" el ojo del jugador agregando detalles que evitan un mayor diseño. Al tener mas detalles el objeto en si (sin la textura) se requiere una mayor cantidad de procesamiento. Similar algunos de estos detalles a través de las texturas es un método simple y muy eficaz.

Para entender esto, hay que explicar que un modelo 3D es un conjunto de vértices y caras en el espacio, agregar detalle implica que estamos agregando mas vértices y mas caras, por lo que estamos agregando mas tiempo de proceso.

Una vez finalizada la elección de dichas herramientas. Se deben tener en cuenta otros programas que harán de soporte a estas, dependiendo de su necesidad. Dado que por ejemplo programas como Milkshape carecen del manejo de un esqueleto para animar, requiere de otro producto para realizar esta labor. En el caso de productos de soporte, los que se llevan los laureles son los paquetes profesionales mencionados actualmente ya que estos disponen en su configuración inicial de prácticamente todas las necesidades de diseño lo que los hace muy independientes.

Cabe mencionar que existen otros programas como Bryce el cual es muy utilizado en los juegos 3D a la hora de crear terrenos, dada su extrema facilidad de manejo. Programas como 3D MAX pueden cumplir con esta labor sin problemas, pero requieren de un mayor tiempo de proceso. Dado que la programación de juegos es bastante compleja, reducir los tiempos de desarrollo en cualquier área son más que importantes.

En resumidas cuentas, el uso de un programa u otro depende pura y exclusivamente del programador del juego (y de su bolsillo). El o los programas que elija son los cuales se debe sentir más cómodo a la hora de trabajar. Eso si, una ADVERTENCIA, aunque programas como Lightwave son relativamente sencillos de aprender, en su gran mayoría el aprendizaje del uso de 3D es bastante arduo y complejo, de echo, son elementos que por si solos pueden generar un producto en si, como una imagen, e inclusive una película (ej. Toy Story, Shrek, etc.)

9. El sonido

En los últimos tiempos, el sonido se ha convertido en una herramienta fundamental para hacer una gran atmósfera en un juego. Como Uds. ya saben, un juego sin sonido es como una película con grandes efectos visual pero que carezcan de el. Mucha gente dice que en un 30% la película adquiere una cierta atmósfera gracias al sonido. Estimo que en los juegos sucede lo mismo.

Para manejar sonidos en el juego, la tarea es relativamente sencilla, solo debemos cargar el sonido en la memoria del equipo y solicitarlo cuando sea necesario (ejemplo, un Bang! A la hora de apretar un botón que indica que se dispara un arma). Es por esto que los sonidos suelen guardarse en archivos individuales y suelen ser en formato WAV.

Otro formato muy usado son los Midis para la música de fondo de los juegos, pero estos poco a poco van perdiendo terreno contra autenticas bandas sonoras.

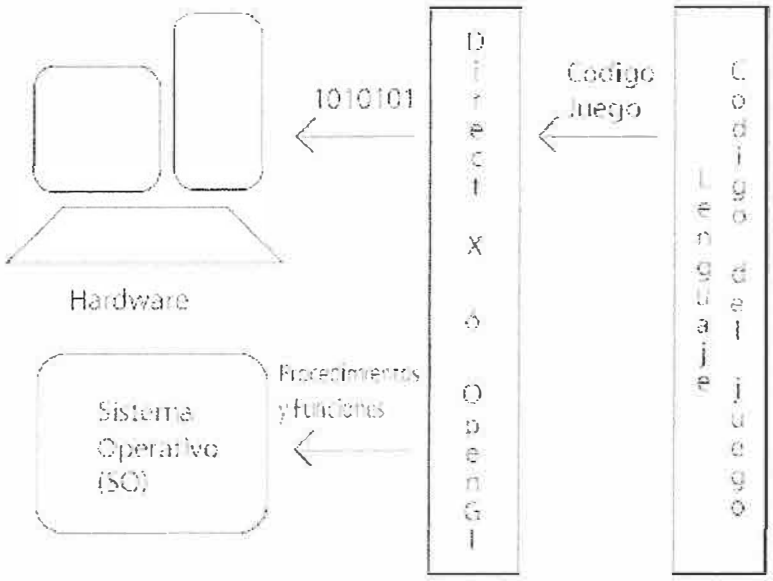
Existen cientos de programas para la edición de sonidos, pero no los mencionaremos aquí. Si hay que mencionar que se requiere una extensa librería de sonidos para poder encontrar el indicado a un cañonazo, unos pájaros de fondo o lo que fuere que el programador desee para generar su atmósfera. Dado que probablemente no dispongamos de un estudio de sonido como lo tienen las grandes compañías de juegos. Es muy probable que en nuestros inicios usemos sonidos que ya fueron creados.

Sin entrar en aspectos legales, es muy común a la hora de programar un juego utilizar sonidos de juegos ya creados. Pero estos elementos poseen copyright y no pueden ser utilizados si vamos a distribuir el juego. No quiero hacer ningún tipo de apología, pero fíjensen en los juego que posean si no es que tienen un directorio con filas con los sonidos de los juegos. De más esta decir que también existen empresas que se dedican a comercializar librerías enteras a las cuales puede acceder cualquier persona que pague por ellas.

10. DirectX y OpenGl

A menudo escuchamos sobre estas dos tecnologías y si bien podemos desconocerlas completamente. Sabemos que son necesarias para ejecutar los juegos. Entonces ¿qué son DirectX y OpenGL? Básicamente, evitando una explicación compleja son unas

librerías que nos permiten comunicamos con el hardware de manera directa sin tener que escribir complicados códigos para que estos los interpreten. De esta manera evitamos una gran carga de programación a la hora de realizar los juegos. En definitiva son unas librerías que servirán de interprete entre nuestro juego y los diferentes dispositivos de hardware (placa de video, procesador, Dispositivos de entra y salida, etc.).



Existen varias diferencias entre ellos pero básicamente realizan la misma tarea. Es cuestión del programador utilizar la que se sienta más cómoda

DirectX VS OpenGL

Existen muchos aspectos tecnológicos en referencia a las diferencias de dichas librerías. La más importante es que **DirectX** es una herramienta de **Windows** y **OpenGL** puede ser utilizado en una gran **variedad de plataformas**. Existe una mini guerra entre estas dos librerías y están las personas que defienden una y otra, como así también las personas que se sienten a gusto con las dos. Lo fundamental de las dos es su cometido. Ambas facilitan la programación y reducen sus tiempos. Si el programador quisiera puede comunicarse directamente con el Hardware pero esto requeriría de un mayor tiempo de programación.

La mejor manera de considerar a ambos es como un interprete que traduce nuestras solicitudes en un lenguaje natural (similar al lenguaje humano) a CÓDIGO MAQUINA

(unos y ceros) para que el hardware pueda interpretarlo. Además evitan que tengamos que tener una especificación mas ampliada de cómo procesa la información en el hardware.

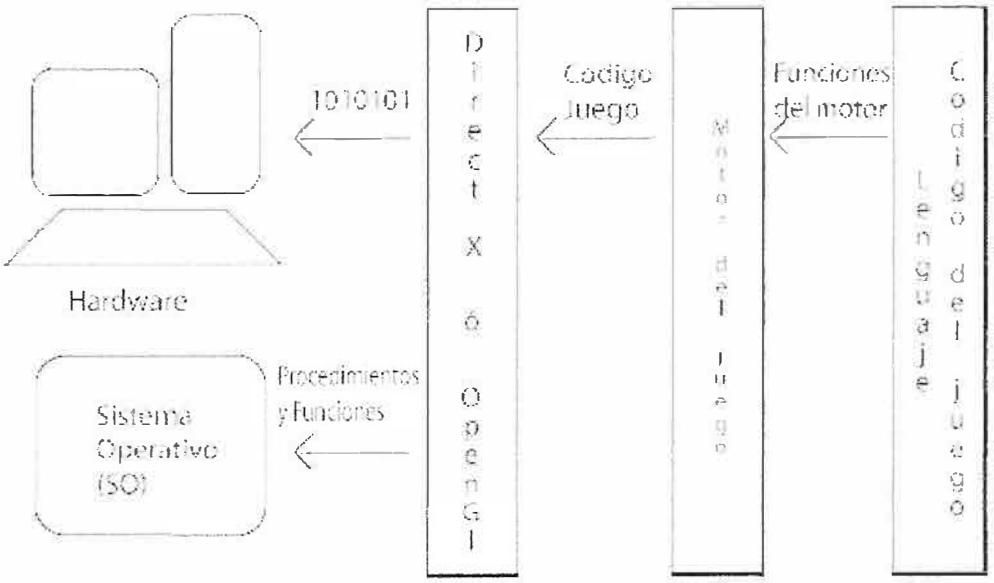
Ambas librerías reducen la complejidad y buscan la estandarización. Si no deseáramos usarla deberíamos programar código para cada placa específica que existe en el mercado. Y como bien sabemos existen muchas. De esta manera las empresas productoras de hardware realizan sus productos compatibles con estas tecnologías.

Cabe mencionar que DirectX es el nombre de un conjunto de herramientas como Direct3D, DirectSound, DirectInput. Como observan en sus nombres, esta bien indicado para que sirve cada una de estas librerías (manejo del 3d, del sonido, etc.).

Si tienen la oportunidad de observar las características de una placa de video por ejemplo, observaran que las mismas indican la compatibilidad ya sea de una librería o de la otra y además indican la versión actual de las mismas ya que ambas sufren modificaciones y mejoras constantemente.

11. Caballeros arranquen sus motores

A la hora de desarrollar un juego, el programador tiene 2 opciones básicas, programar directamente con las librerías **DirectX** u **OpenGl** (u otras similares). O utilizar un set de herramientas que ya a sido creado para el desarrollo de un juego. Esto quiere decir que agregamos un nivel más de programas entre el Hardware y el código del juego. El objetivo de esto es sencillo. Que el programador dedique mas tiempo a la programación del juego en si y menos a los procedimientos que este requerirá. Esto es un motor 3D



Existen muchos motores 3D en la actualidad. Algunos son gratuitos y otros son motores desarrollados para juegos que luego puede licenciarse. Algunos casos de éxito de esto son los motores del Quake como así también los del Half life, etc.

Estos motores luego fueron utilizados en un sinfín de productos más. Comúnmente cuando una empresa consigue la licencia de un motor, lo mejora y lo adapta a su necesidad.

Es necesario entender que los motores no son más que códigos que fueron creados por otra persona que maneje ciertos aspectos del juego que no son necesarios reprogramarlos para realizar otro tipo de juego. Al tener muchos aspectos del juego ya creados, un equipo de desarrollo puede invertir sus esfuerzos en otros elementos claves de un juego.

Existen motores 3D los cuales se encargaran de manejar los gráficos del juego. También existen otros motores específicos como los de física los cuales le brindan la posibilidad de agregarle física ya sea real o no a nuestro juego (que un avión mantenga su sustentación mientras tenga una x velocidad del aire es un caso típico de física). Otros motores como los de Inteligencia Artificial es necesario mencionarlos.

Para que vaya a programar como debe interpreta mi placa de video el movimiento de un personaje mediante un esqueleto si ya tengo una rutina (un motor) que se encarga de

hacerlo. De esta manera tengo mas tiempo para pensar como se va a mover el personaje que en como voy a hacer que la maquina lo entienda.

Eso es el concepto de motor, sea gratuito o no. Su función básica como fue la de **DirectX** y **OpenGl** es simplificarle el trabajo al programador. Están pensando lo complicado que es programar un juego dado que se invirtió mucho tiempo para que el procedimiento sea simple no? Es complicado, si, pero a la vez muy adictivo.

12. Esos benditos mods

La idea de dejar parte del código de un juego "abierto" para que cualquier persona pueda programar con el fue una brillante idea que apareció en la primera versión de Quake. De este modo permitían a cualquier persona desarrollar un juego basándose en el original.

De este concepto nacieron grandes MODs que pueden ser juegos en si mismo como es el fantástico Counter Strike, basándose en el motor del Half Life. También gracias a que le permitieron a cualquier persona programar parte del juego, nacieron nuevas modalidades hoy consideradas Standard en ciertos juegos como el clásico Capture the Flag (obtener la bandera).

Todo esto sin considerar que el juego prevalece más en el tiempo ya que se explota al máximo sus recursos. Lo cual aumenta las ventas del juego en si, por lo que muchos consideran que fue una fantástica idea.

Se puede distinguir entre 2 tipos de MODs, los Single Placer y los Multiplayer. Los primeros no son más que extensiones del juego original. En algunos casos podemos ver que ciertos MODs son historias más interesantes que la historia "oficial" del juego.

En mi experiencia particular considero uno de los MODs mas interesantes el desarrollado para el juego Pirates of the Caribbean (juego de rol). Si bien su historia original es atractiva, la historia del MOD es emocionante y se observa un mayor aprovechamiento del motor del juego.

En el caso de los multiplayer, acá es donde los MODs poseen su mayor éxito. Existen modalidades excelentes así como también muy locas y todas llevan a lo mismo, el juego

en grupos cooperativos. Tengo que destacar un MOD, además del Counter Strike claro esta.

Natural Selection es otro MOD de Half Life el cual le agrego a un FPS una excelente estrategia cooperativa.

Resumiendo, con los MODs las empresas ganan mas (mantiene mas tiempo "vivo" el juego) los jugadores juegan mas (aprovechan mas su compra ya que los MODs suelen ser gratuitos) y los programadores disfrutan mas. Programar un MOD es una agradable experiencia que se encuentra en el medio de jugar un juego y programarlo desde 0.

13. Final del curso

Bien, hemos cubierto la mayoría de los tópicos de un juego: sus géneros, sus herramientas de programación, su estructura esencial, DirectX y OpenGl, los motores, sus gráficos, sus sonidos. Hasta ahora he escrito mucho pero probablemente todavía no entiendan exactamente como se realiza un juego.

Pues bien, todo lo aprendido hasta el momento es lo necesario que se debe saber de todos los aspectos que poseen los juegos. Cada tema requiere uno y muchos cursos propios y cada tema posee su complejidad lo cual no puede abarcarse todo en un solo curso, ya que lo haría tedioso y finalmente no se llegaría bien a ningún puerto.

Tratando de no desanimar a nadie, simplemente les he intentado brindar el puntapié inicial. Es una decisión de Uds. Decidir si esta información les fue importante, si aprendieron algo con ella.

Mi objetivo principal a lo largo del proyecto, fue hablar de los componentes de los juegos sin entrar en cuestiones técnicas y que a partir de aquí Uds. sepan que buscar ya que en la programación de juegos existen cientos de miles de paginas Web escritas como así también libros, tutoriales y demás.

Muchos de los elementos mencionados aquí requieren de una enseñanza que en cierta manera se aleja de la programación de juegos e ingresa a áreas como el Arte, lo visual, el gusto auditivo, etc.